UNIVERSITY OF ALBERTA

COMPUTER VISION SYSTEM FOR ON-LINE LASER BEAM DIAGNOSTICS AND CONTROL

BY

CHRISTOPHER VIREINDRA SELLATHAMBY ©

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR

THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF ELECTRICAL ENGINEERING

EDMONTON, ALBERTA

FALL, 1992

ISBN 0-315-77371-5

UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: Christopher V. Sellathamby

TITLE OF THESIS: Computer Vision System for On-Line Laser Beam Diagnostics and Control

DEGREE: Doctor of Philosophy

YEAR THIS DEGREE GRANTED: 1992

Permission is hereby granted to the University of Alberta Library to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves all other publication and other rights in association with the copyright in the thesis, and except as hereinbefore provided neither the thesis nor any substantial portion thereof may be printed or otherwise reproduced in any material form whatever without the author's prior written permission.

Christopher V. Sellathamby
29 Elkin Close
Red Deer, Alberta
CANADA T4R 1Y8

Date: Oct. 8/92

UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled COMPUTER VISION SYSTEM FOR ON-LINE LASER BEAM DIAGNOSTICS AND CONTROL submitted by CHRISTOPHER VIREINDRA SELLATHAMBY in partial fulfillment of the requirements for the degree of DOCTOR OF PHILOSOPHY.

Supervisor: Dr. H. J. J. Seguin

Supervisory Comm. Member: Dr. N. G. Durdle

Roger W Toogood

Supervisory Comm. Member: Dr. R. W. Toogood

C. G. Englefield.

Committee Member: Dr. C. G. Englefield

Z J Koles

Committee Member: Dr. Z. J. Koles

Ronald P W Lawson for

External Examiner: Dr. B. Ahlborn

Date: Oct. 8/92

Dedicated to my mother,

for her encouragement and guidance in all my endeavors, and

for her immeasurable love and devotion.

ABSTRACT

The use of lasers in industrial and research applications has increased dramatically throughout the world in the past decade. In order to derive maximum benefit from the new generation of lasers, with improved power and processing capabilities, it is necessary to maintain output beam parameters at optimal levels. As the complexity of these lasers increase, more effort must be placed on the design of automated systems, which relieve the operator from performing many laborious and intricate control adjustments. Such systems must be capable of monitoring the laser's output beam without interfering with the application, and generating the necessary control actions to stabilize the desired beam parameters.

An on-line, real-time vision-based system has been developed for automatic beam control and power stabilization of a high power CO_2 laser. This feedback control system continually analyzes the cross-sectional profile of a sample of the laser's output beam and utilizes the intensity information present in this image to servo-regulate the resonator cavity optics and the excitation level of the discharge. Beam uniformity and the direction of beam propagation are maintained by adjusting the primary and secondary resonator optics respectively. The far-infrared CO_2 laser output is converted to the visible range via a thermoluminescent viewing screen, thereby enabling beam supervision with the use of a standard video camera, frame grabber, and microcomputer. Also incorporated into the system is an automatically controlled, variable coupling percentage reflective wand (for beam sampling), which ensures infrared sensor operation within its optimal dynamic range.

The technique stabilizes the average laser output power to within 4%, while maintaining the best output power distribution achievable by the particular laser. In addition, beam steering effects are significantly reduced by restricting the direction of beam propagation to within 60 μrad of the desired setpoint. The variable sampling

percentage system permits laser operation with on-line power variations from 1 to 10 kW. These results indicate that the system considerably improves laser performance by maximizing beam uniformity, minimizing beam steering, and stabilizing total output power. This ensures the degree of consistency and repeatability in the output beam parameters that are essential to precision laser applications. Control system design also provides valuable diagnostic information pertaining to output beam quality, such as intensity and energy profiles, which may be monitored simultaneously or stored for off-line analysis.

ACKNOWLEDGMENTS

It is with great pleasure that the author expresses his appreciation to the many people whose assistance enabled the completion of this research. First and foremost, a sincere note of gratitude is extended to his supervisor, Dr. H. J. J. Seguin, whose patience, guidance, and knowledgeable advice over the last five years have made this an invaluable and enriching experience.

Additional appreciation must be conveyed to the all members of the high power laser research group for their technical and intellectual support. In particular, the author is grateful to Dr. D. R. Akitt for his technical help, insightful suggestions, and stimulating discussions, and to Mr. H. Reshef for his expertise in operation and maintenance of the laser systems, and fabrication of many precision components. Also acknowledged are Mr. D. Presakarchuk for his aid with the laser electronics, Mr. K. Regozinski for his help with operating and modifying the PIE-4 laser, Mr. H. Dexel for his exceptional machinist skills, and Mr. V. Pohnert for his assistance with the CAD system. Dr. S. K. Nikumb, Dr. E. F. Yelden, Mr. J. Strohschein, and Mr. S. W. Scott are thanked, as well, for their interest in this research and their contributions through many enjoyable discussions.

Further gratitude is extended to Dr. N. G. Durdle for the use of his programmable logic design facilities, and to Mr. K. denHartigh for his technical aid and instructional guidance in operating the system.

The author also gratefully appreciates the financial support of the University of Alberta, the Province of Alberta, and the Natural Sciences and Engineering Research Council of Canada.

TABLE OF CONTENTS

Chapter Page

LIST OF ILLUSTRATIONS

Figure Page

Figure **Page**

Figure Page

Figure **Page**

LIST OF TABLES

LIST OF ABBREVIATIONS

Symbol	Definition
2D	two dimensional
3D	three dimensional
α	angular deviation of beam propagation direction from optical axis
a_i	denominator parameter of discrete-time model
A	ampere
AC	alternating current
A/D	analog to digital
ANSI	American National Standards Institute
ASCII	American Standard Code for Information Interchange
ASM	assembly code or program
b_i	denominator parameter of discrete-time model
C	capacitance or capacitor
C	centroid
CCD	charge coupled device
CO_2	carbon dioxide
CW	continuous wave
d	process delay in number of sample periods
D_i	diameter of resonator optic
D/A	digital to analog
DARMA	Deterministic Auto-Regressive Moving-Average
DC	direct current
ε	error signal
EPLD	erasable programmable logic device
g_i	resonator characteristic parameter

Symbol	Definition
gl	gray level
gv	sum of gray values, used as a measurement of image intensity
G_i	transfer function
HeNe	helium-neon
IFF	Interchange File Format
I/O	input-output
IMC	internal-model-control
ITAE	integral of time absolute error
K	gain
L	length of resonator cavity
λ	wavelength of laser radiation
M	unweighted first moment of inertia
MIMO	multi-input, multi output
MW	weighted first moment of inertia
n	sample number
N	total number of samples
ϕ	angular width of reflective surface of rotating wand beam sampler
Φ	input-output matrix for least squares parameter estimation
Δp	absolute displacement of beam centre from optical axis
P	laser output power (in s-domain)
PID	proportional-integral-derivative
PIE	photo-initiated, impulse-enhanced, electrically-excited
Q_i	total intensity of specified quadrant
R	resistance or resistor
R_i	radius of curvature of resonator optic
RAM	random access memory

Symbol	Definition
ROM	read-only memory
RPM	revolutions per minute
s	continuous Laplace transform variable
S	total intensity of infrared detection and imaging screen (in s-domain)
SCR	silicon controlled rectifier
SISO	single-input, single-output
τ	time constant
θ	angle of slot in variable coupling wand cover
Θ	parameter vector for least squares estimation
t	time
T_d	process time delay
T_i	integral or reset time
T_s	sampling period
U_i	controller output to associated process (in s-domain)
W	watt
W	wand speed controller output in RPM (in s-domain)
X_p	horizontal position error (in s-domain)
Y_p	vertical position error (in s-domain)
X_u	horizontal uniformity error (in s-domain)
Y_u	vertical uniformity error (in s-domain)
z	discrete-time transform variable
ZnSe	zinc-selenide

CHAPTER 1

INTRODUCTION

Since the invention of the ruby laser in 1960 by Maiman,[1] lasers have assumed a dominant role in both industrial and research applications. With the discovery of new types of lasers, the scope of laser applications has expanded to include a wide range of fields, from communications and medical surgery to metal processing and nuclear fusion. This remarkable market development has arisen from the fact that lasers can be implemented into a variety of industrial environments in a cost-effective and automated manner.

The introduction of the carbon dioxide laser by Patel in 1964,[2,3] has dramatically increased the feasibility of utilizing lasers in many areas of industrial material processing.[4-7] The CO_2 laser is currently the most popular for material processing applications, such as welding, cutting, cladding, heat treating, and surface hardening,[8-11] since it is capable of high average powers at multikilowatt levels, with relatively high efficiencies (15% to 25%).[12,13] Industrial quality CO_2 lasers capable of output powers exceeding 25 kW have been developed,[14] while construction of those with powers of up to 100 kW are under investigation.[15] Future developments may result in other types of gas and solid state lasers achieving these high power levels, thereby widening the scope for laser applications.[16]

In order to properly implement these high power lasers in industry and perform at maximum efficiency, the quality of the output beam must be sufficiently maintained.[17,18] In material processing applications, the process is extremely dependent on the output beam parameters of the laser. Thus, to avoid substandard processing of materials, it is essential to regulate the output beam at a consistent and optimal power level, with a uniform power distribution across its profile.

Preserving this beam quality at high power levels for extended periods of operation has proven to be a non-trivial task. Unfortunately, lasers having sufficient power to permit significant material processing are characterized by large physical size, complexity, and extreme variability in their operational features. Excessive heating of the laser optics and mountings causes distortion and misalignment of the resonator, resulting in a non-uniform, poor quality output beam. Although this hurdle can be eliminated to a certain extent by the implementation of various "passive" systems and methods, such as the use of thermally stable optical benches and adequate cooling of the optics, it cannot be prevented completely. Consequently, it is usually impractical to engineer these systems with inherent mode and power stability. Continuous monitoring and external control of the laser's resonator and optics by a human operator is required to generate a uniform output beam and facilitate long-term operation at optimum levels. A more effective approach would entail replacing the operator with a completely automated supervisory system, having a sufficient degree of sensory capability and decision making ability to provide the necessary control functions for obtaining results with high degrees of quality and consistency.

In addition to drifts in alignment, the output power level may fluctuate due to changes in properties of the gain medium (for example, variations in temperature and gas chemistry in a CO_2 laser). This could produce detrimental effects on a material processing application sensitive to the output power level, such as a defective weld or incomplete cut. As a result, an automated system which incorporates both output power stabilization and mode control would significantly increase the performance of any laser system.

The focus of this research is the development of a vision-based system suitable for analysing the laser's output beam and instigating the required control actions to maintain the desired mode quality, beam position, and output power. By continuously preserving the optimal beam parameters, such an expert feedback control system would ensure laser operation at maximum efficiency, fostering superior and reproducible material processing at accelerated rates. Although this study utilizes a high power CO_2 laser to evaluate the performance of this vision-based system, the principles and methods involved can be extended for use with all types of lasers.

1.1 Principles of Laser Operation

The laser (an acronym for Light Amplification by Stimulated Emission of Radiation) utilizes the fact that an atom in an upper energy state makes a transition to a lower state if it interacts with a photon having an energy equal to the energy difference between the two states. The result of this transition is the release of a second photon which is coherent (equal to in phase and frequency) to the incident photon. For this process of stimulated emission to dominate, the population of atoms in the upper state must exceed that of the lower. Under normal equilibrium conditions, such a population inversion does not exist. Thus, any type of gain medium utilized within a laser, requires a suitable pumping mechanism to achieve this necessary population inversion.[19]

A general schematic diagram of a laser is depicted in Fig. 1.1. Upon establishment of a population inversion, spontaneous emission occurs randomly in all directions. Most of the emissions are lost since they propagate in directions away from the laser cavity. However, the relatively small amount of radiation which propagates perpendicular to the mirrors of the cavity, provides an origin for stimulated emission, and is coherently amplified as it repeatedly passes through the gain medium. Providing the gain of the medium is adequate to compensate for the losses in the cavity, the resulting oscillation perpetuates, and a suitable percentage of the energy (usually less than 10%) may be extracted through a partially transmissive mirror. Since the lasing process depletes the population of the upper energy state, oscillations may be possible on continuous (CW) or pulsed basis, depending on the properties of the gain medium and the pumping scheme.[20]

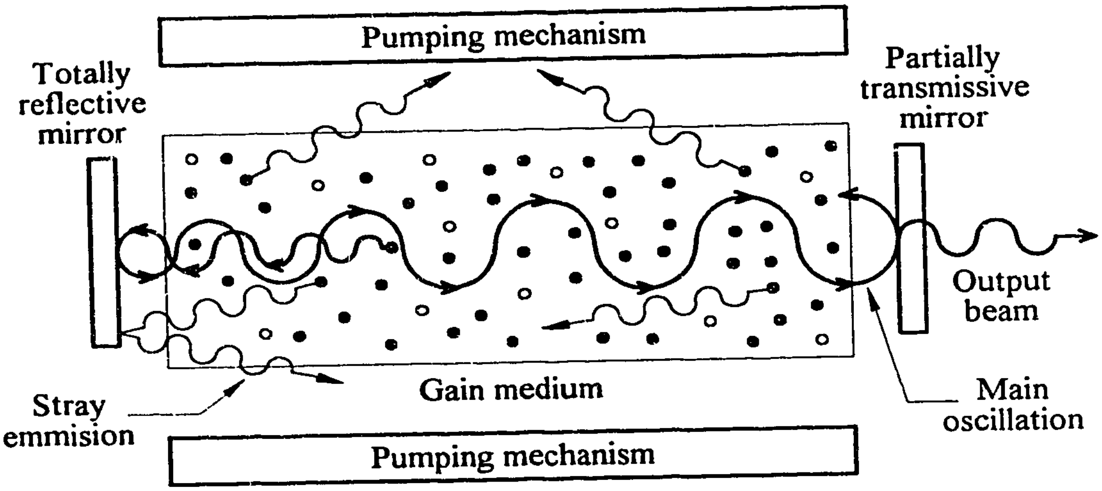


Fig. 1.1. Schematic diagram of a laser system.

1.2 Characteristics of Laser Resonators

The losses within the laser cavity and the uniformity of the output beam are extremely sensitive to the alignment of the resonator mirrors. A slight deviation of the two mirrors from their mutually paralleled state may result in a part of the reflected beam straying outside the bounds of the resonator cavity, causing excessive losses and an inability to attain oscillation. Once the lasing process is established, thermal gradients can cause physical distortions to the optical surfaces and mounts, as well as aberrations within the gas media itself. These distortions are sufficient to produce a misalignment of the resonator, contributing to output mode and beam uniformity deterioration, and beam steering.

Most modern laser systems do not employ the simplistic, flat-mirror, Fabry-Perot resonators, since they are generally troublesome to align and experience high diffraction losses. Instead, the radii of curvature of the mirrors are modified to achieve various desired operating characteristics. These curvatures are used to categorize resonators into two main groups - stable and unstable. Stable resonators typically have low losses and are suitable for lasers with low gain coefficients. However, their associated lowest order modes do not interact with a large volume of the gain medium, yielding poor mode filling factors. On the contrary, unstable resonators posses large mode filling factors and are relatively more insensitive to misalignment of the cavity.[21] The basic parameters used to characterize a given resonator as stable or unstable are its length and the radii of curvature of its mirrors, as described in Fig. 1.2. Based on these dimensions, the cavity parameters, g_1 and g_2, may be computed as follows:

$$g_1 = 1 - \frac{L}{R_1} \tag{1.1}$$

$$g_2 = 1 - \frac{L}{R_2} \tag{1.2}$$

The required condition for resonator stability is given by:

$$0 \le g_1 g_2 \le 1 \tag{1.3}$$

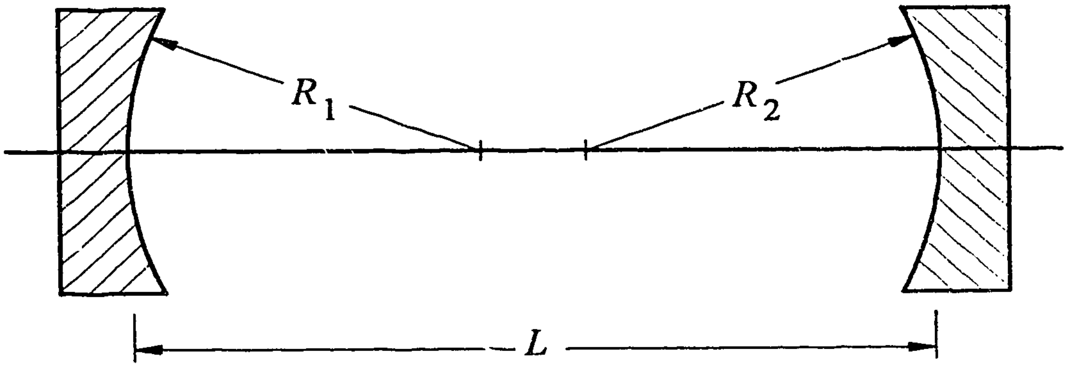


Fig. 1.2. Basic laser resonator parameters.

High power laser systems generally cannot utilize partially transmissive optics due to the enormous thermal loads placed on them. Consequently, totally reflective optics, fabricated with metals and water-cooling, are employed. In order to extract a beam from such a resonator, one of the mirrors is made convex (negative radius of curvature) with a diameter smaller than that of the concave. Thus, as the beam expands towards the edges of the concave mirror, it eventually travels past the outer edges of the convex, producing a annular-shaped output beam (shown in Fig. 1.3). Such resonators are generally unstable and provide for easy alignment, high mode filling factors, and superior lowest order modes.

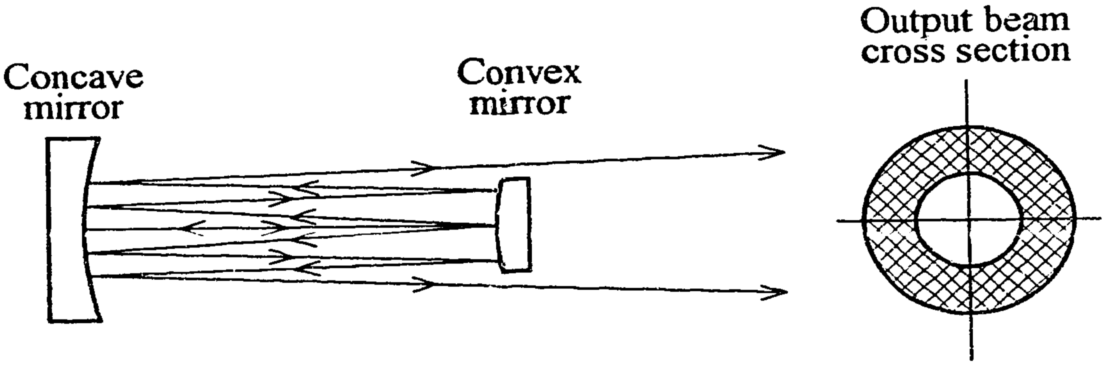
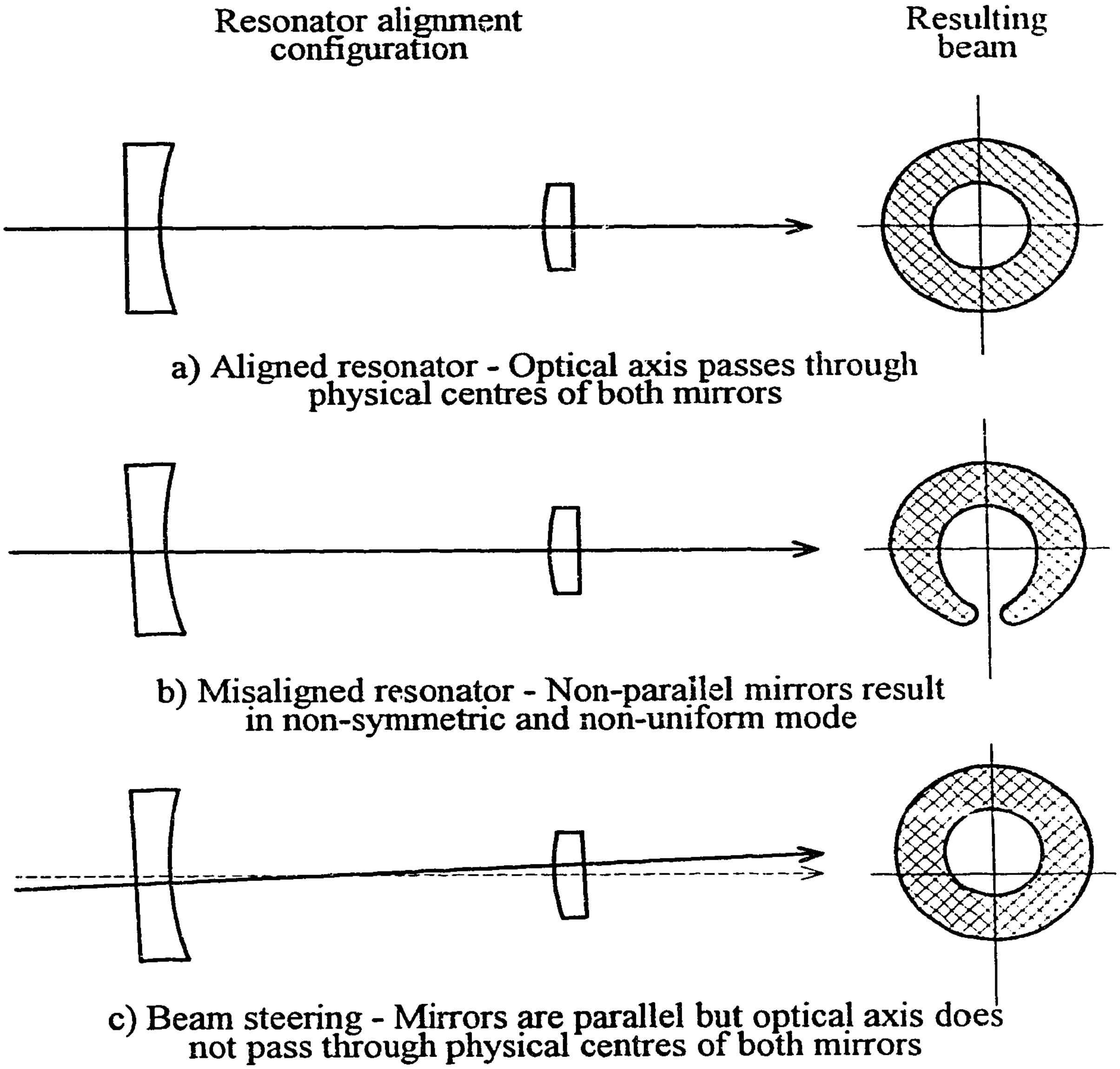


Fig. 1.3. Unstable resonator utilizing totally reflective optics.

Three distinct alignment configurations of an unstable laser resonator and the resulting cross sections of the lowest order output modes are presented in Fig. 1.4. The first condition depicts proper alignment, in which the two mirrors are parallel to each other, and perpendicular to the optical axis linking their centres. The associated output beam is relatively symmetric and uniform in power distribution. The second case illustrates a misaligned resonator, in which the mirrors are not mutually parallel, resulting in a non-uniform and partial beam. Such beams further increase the thermal stresses placed on the mirrors and may seriously damage sensitive optical components. The final condition reflects the concept of beam steering. Both



a) Aligned resonator - Optical axis passes through physical centres of both mirrors

b) Misaligned resonator - Non-parallel mirrors result in non-symmetric and non-uniform mode

c) Beam steering - Mirrors are parallel but optical axis does not pass through physical centres of both mirrors

Fig. 1.4. Resonator alignment conditions and the associated beam cross section.

resonator mirrors are parallel, but they are not perpendicular to the axis connecting their centres. Although the direction of beam propagation is not parallel to the optical axis, the mode remains relatively symmetric, provided the diameters of both mirrors are greater than $\sqrt{\lambda L}$, where λ is the wavelength of radiation, and the propagational axis is confined within this boundary. Depending on the length of the beam delivery system and the type of application involved, such a minute divergence could have numerous adverse effects on the final quality of the product.

1.3 Overview of Beam Control and Diagnostics

A number of external beam parameters may be monitored to analyse the beam's quality and uniformity for a given application. These include the power and energy densities and distributions in the beam, the angular divergence and beam spot size, the output wavelength, and the coherence and polarization of the beam.[22] Of these parameters, observing the power distribution and intensity profile is crucial to maintaining a symmetric and uniform beam with maximum intensity.

Many complex systems have been developed to accurately measure the quality and uniformity of the output beam.[23-29] These range from two-dimensional photodiode arrays[30] to acoustic detectors which measure signals emanating from the laser optics.[31,32] For CO_2 lasers, which produce an invisible infrared beam with a wavelength of 10.6 μm, pyroelectric detectors[33,34] and thermistor arrays[35] have been used to examine the quality of the output. One of the more recent and versatile methods of beam monitoring is to utilize an imaging system and employ image analysis techniques to process the data.[36,37] Imaging systems used for laser diagnostics genera! examine a sampled version of the beam's cross section. This cross-sectional image is then analysed and an intensity profile of the beam is produced. At this point, human interpretation of the data is required to make the necessary modifications to the laser control system. Due to the diversity and variability in laser operating parameters, and differences in operator expertise, obtaining optimal and repeatable results in this manner is both difficult and tedious.

Eliminating the need for human intervention requires a merging of both the diagnostic and control systems. Currently, low-power laser systems (especially those with wavelengths in the visible range) do exist with a crude forms of automatic resonator alignment and power control capabilities.[38] However, high power lasers are generally limited to feedback control of only the total output power,[39] necessitating manual alignment of the resonator optics to preserve the mode quality.

A typical schematic diagram of a vision system for automatic laser resonator alignment and power stabilization is shown in Fig. 1.5. The system operates by first sampling the output beam, which permits on-line utilization, and then feeding a cross section of this sampled beam into a video or CCD camera. Depending on the power level and wavelength of the laser, it may be necessary to project the sampled beam onto a visibly reflective screen, or to employ a camera suitable for the particular wavelength (such as a far-infrared camera for CO_2 lasers). The image of the beam cross section is then digitized by a frame-grabber and stored as a two-dimensional array in a computer. Elements of the image array consist of a series of gray values, each of which corresponds to the intensity of the beam at a given point. Thus, an intensity profile of the beam may be extracted from the image, and the total output

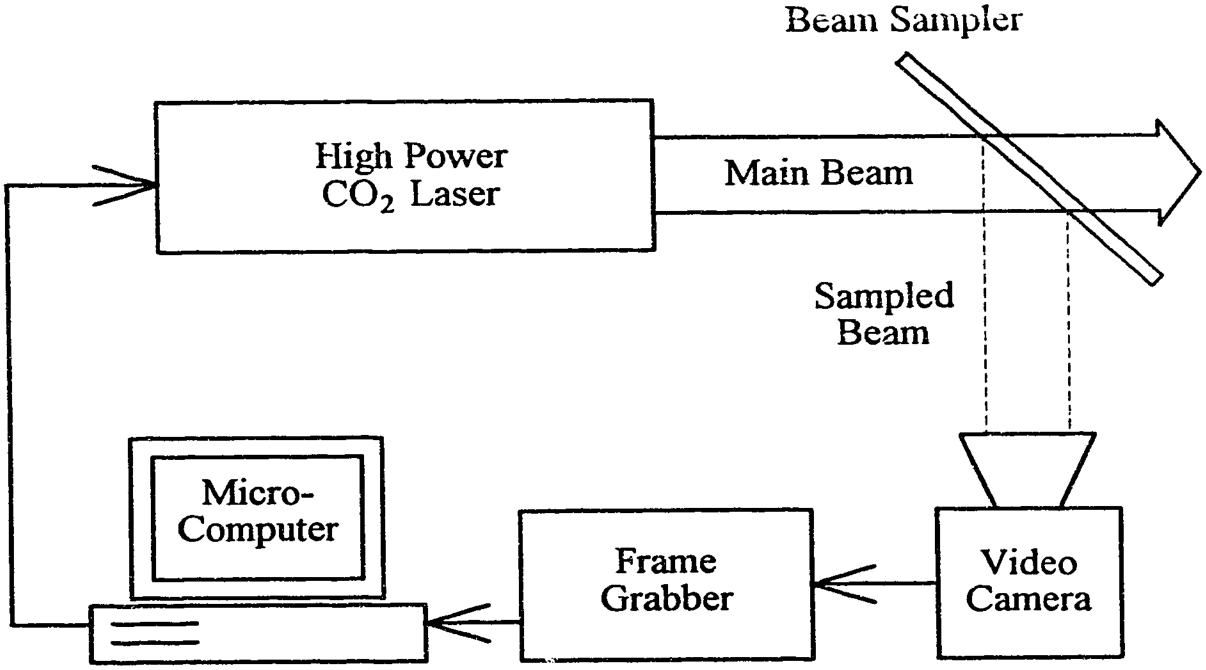


Fig. 1.5. Schematic diagram of a computer vision system for on-line laser control.

power may be calculated. This cross-sectional image may be further analysed , using image processing techniques, to obtain measures for uniform power and energy distribution. Based on these measures, the system can apply the necessary controls to the laser system to optimize the output beam and operate at the desired power level. This process is repeated continuously while the laser is in operation to maintain maximum beam uniformity and laser efficiency.

For operation in a real-time environment, the digitizing and image processing systems must possess sufficient speed, and the response time of the optical alignment servos must be adequate. However, since high power laser thermal time constants are usually large, the processing speeds need not be excessively high, thereby enabling the use of a standard microcomputer.

This computer vision system for monitoring and control of a laser's output parameters has a number of advantages over other conventional techniques. The system is extremely versatile and completely external to the laser, making it adaptable to virtually any type of laser system, with no internal modifications. Since the sensory system is based on visual data, no additional mechanical or electrical process related signature on which to base control decisions is necessary. Many of the laser's constants may be modelled in software, enabling effortless transportability between different laser systems. The laser itself may be operated in either CW or pulsed mode. If operated in pulsed mode, some synchronization is required between the imaging system and the laser, which can be easily included.

1.4 Experimental Research Strategy

In order to develop the proposed vision-based feedback control system for on-line laser supervision, an unstable resonator arrangement, employing a low power HeNe laser, was constructed. This system was used to initially design the image analysis software and alignment algorithms for control of a single resonator mirror.

Upon development of a suitable imaging system for detection of the invisible infrared CO_2 laser beam and fabrication of the necessary hardware interfaces, the diagnostic and control software was tested on the high power PIE-3 CO_2 laser.* The performance improvement of the laser system was thus evaluated.[40] Control algorithms were subsequently expanded to include power stabilization[41] and manipulation of the second resonator optic. An effective control strategy was ascertained to eliminate fluctuations in beam uniformity, direction of propagation, and output power. The performance characteristics of the system were then assessed on the higher power PIE-4 laser, at which time a mathematical model for the laser and control systems was derived to examine their stability and limitations.

1.5 Summary of Research Objectives

1. Develop a vision-based system which automatically aligns a single resonator optic of a high power CO_2 laser.

2. Develop sufficient beam diagnostic capabilities, such as 2D and 3D intensity profile generation, to aid in analysing and characterizing resonator designs.

3. Expand the system to incorporate automatic output power stabilization.

4. Further expand the alignment control system to include control o:̃ the second resonator optic, and develop an effective control strategy.

* The PIE-3 laser is a 10 kW CO_2 laser utilizing the PIE (Photo-initiated Impulse-enhanced Electrically-excited) discharge technology,[42-43] which was developed at the University of Alberta in the late 1970's. The PIE-4 CO_2 laser is of similar design, but is capable of output powers in excess of 25 kW.

CHAPTER 2

SYSTEM HARDWARE DESIGN

In order to expeditiously evaluate the laser cavity feedback control concept, a low power unstable resonator configuration, utilizing a HeNe laser, was constructed to simulate a high power laser cavity (illustrated in Fig. 2.1). This low power model facilitated preliminary investigation of control strategies and unstable resonator behavior, without jeopardizing costly, high power laser components. The HeNe output beam was passed through a minute opening in the centre of the concave mirror. Following repeated reflections within the cavity, the typical annular-shaped output of an unstable resonator was produced. Since the HeNe beam was in the visible range of the spectrum, it was reflected directly into the video camera. The video signal was then digitized, the image information was processed, and the

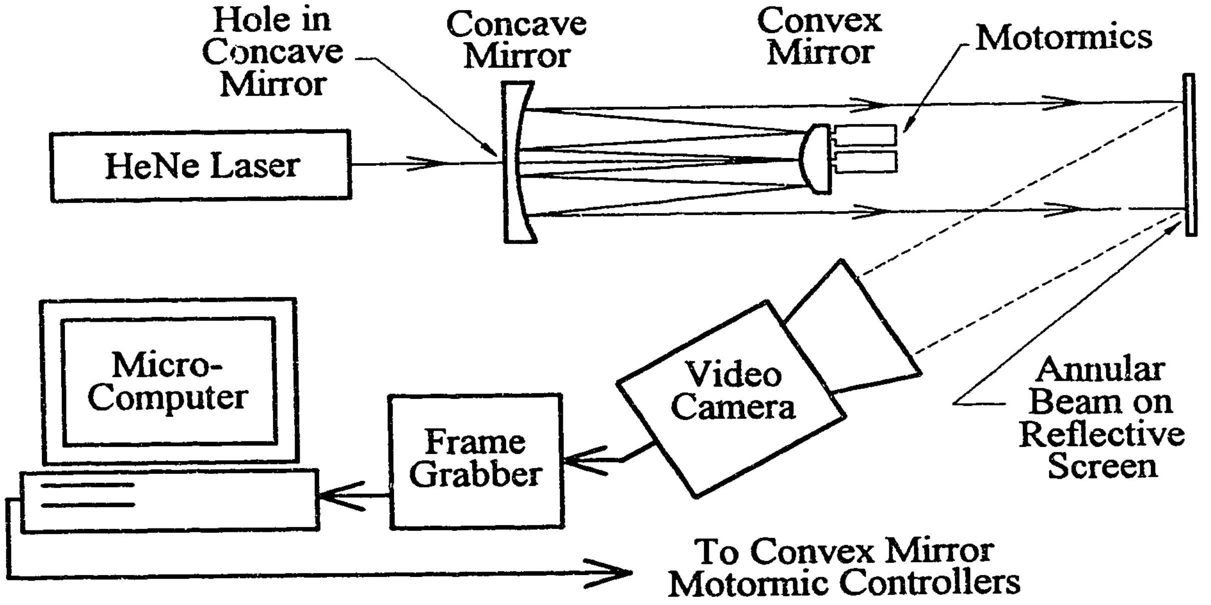


Fig. 2.1. Schematic of a low power control simulation utilizing a HeNe laser.

necessary control signals were sent to two motor-driven micrometers (motormics), causing angular motion of the convex mirror about the X and Y axes. With the use of this model, the required software and control strategies were developed and tested, after which the system was installed on the PIE-3 CO_2 laser. After further performance evaluation, software enhancement, and expansion of control to the second resonator optic, the system was transferred to the higher power PIE-4 laser.

A detailed block diagram of the laser control and diagnostic system designed for the PIE-3 and PIE-4 lasers is shown in Fig. 2.2. The system consists of six main subsystems: the high power PIE CO_2 laser, the beam sampler, the infrared CO_2 beam

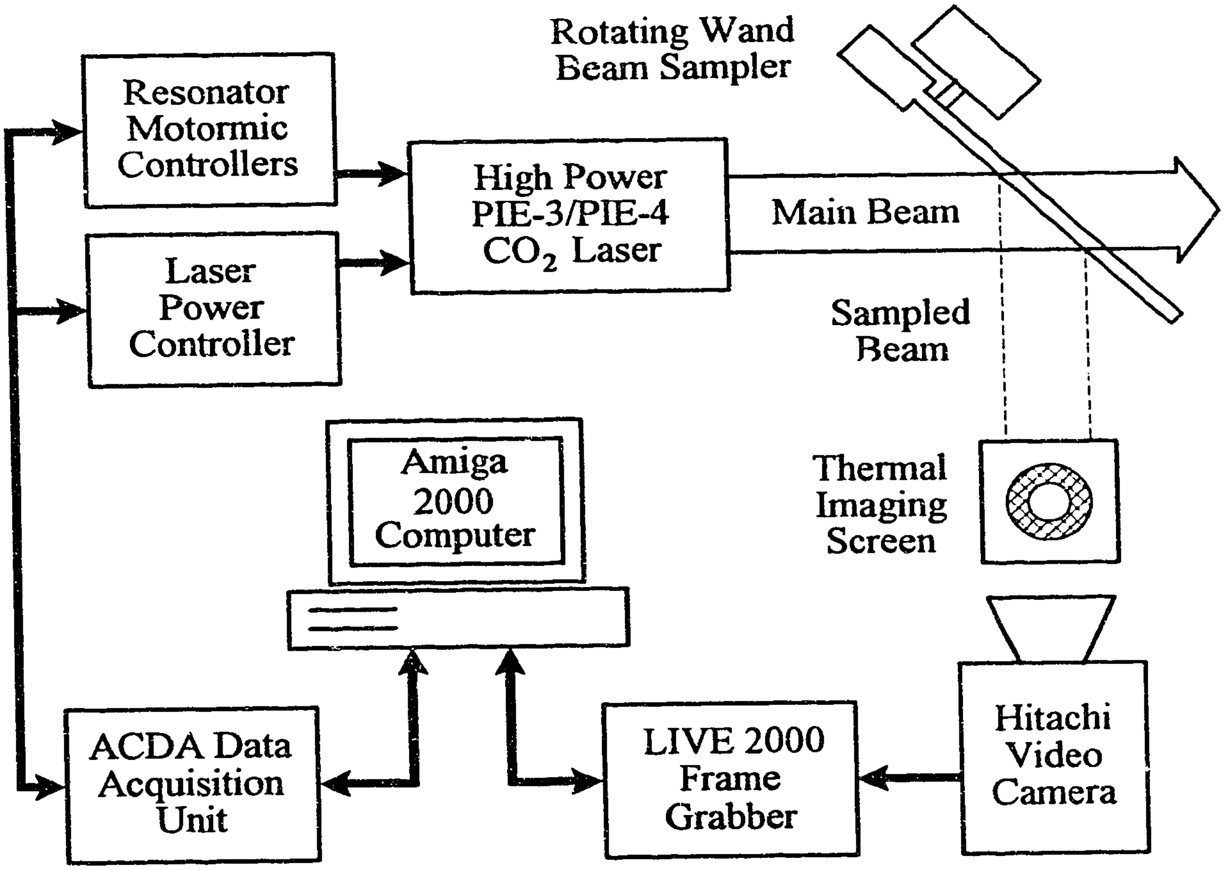


Fig. 2.2. Detailed schematic diagram of the vision-based control and diagnostic system for the PIE-3 and PIE-4 CO_2 lasers.

detector, the image digitizing and analysis system, the resonator alignment scheme, and the laser input power regulation system.

2.1 High Power PIE CO_2 Laser Systems

PIE CO_2 lasers[44-46] are capable of output powers in the multikilowatt range and contain the folded-path resonator configuration illustrated in Fig. 2.3.[47] The main advantage of using a two-pass folded resonator is that it effectively doubles the volume of the gain medium, and is equivalent to doubling the length of the resonator. Furthermore, asymmetries in the discharge, which may produce noticeable non-uniformities in the output of a single-pass cavity, tend to cancel out on the return path in a two-pass system, thereby yielding a more uniform output beam. As evident in Fig. 2.3, the beam oscillates between the two resonator mirrors, with the 90° roof prism acting as the folding element. The beam is then extracted from the resonator cavity with a skimmer and reflected out through a ZnSe window (transparent to far-

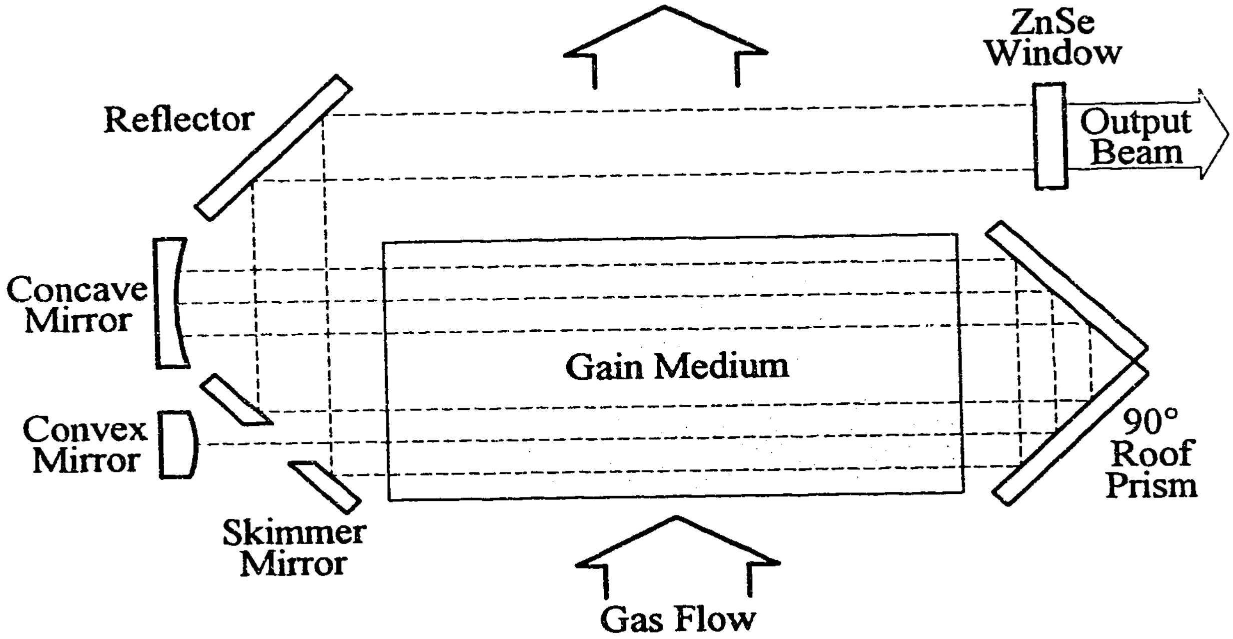


Fig. 2.3. Diagram of folded-path resonator employed in PIE-3 and PIE-4 lasers.

infrared radiation). All reflective optics are water-cooled and constructed of aluminum, with micro-machined surfaces (absorption coefficient of approximately 2%). The resonator itself may operate in either a stable or unstable mode, depending on the radii of curvature of the optics.

2.1.1 PIE-3 CO_2 Laser

The PIE-3 laser, capable of output powers not exceeding 10 kW, has a discharge cross section and length of 18 cm by 9 cm and 3 m respectively, which corresponds to a discharge volume of 40 L. The radii of the convex and concave mirrors are -20.8 m and 38 m respectively, with a cavity length of 8.6 m (marginally unstable resonator). The output beam is annular in shape, with outer and inner diameters of approximately 9 cm and 4.5 cm. With the exception of water-cooled optics, no other inherent mode stabilization systems are installed.

2.1.2 PIE-4 CO_2 Laser

The PIE-4 laser permits power levels exceeding 25 kW and is similar in design to the PIE-3 laser, although it incorporates more sophisticated electronic and temperature stabilization systems.[48] It possesses a discharge volume of 81 L (10 cm × 18 cm × 4.5 m), with a cavity length of 9.2 m and mirror curvature radii of -40.5 m and 44.65 m. These dimensions yielded a marginally stable resonator, which produced a better quality output mode for this particular laser than the unstable configuration. Inner and outer diameters of the output beam measure approximately 10 cm and 5 cm respectively. Temperature of the optical components are maintained between 22 and 24 °C using water cooling, while the mirror mounts are fastened to an optical bench, thermally stable to within 0.1 °C. This prevents the large thermal loads on the laser tank from significantly affecting the alignment of the resonator.

2.2 Beam Sampling Mechanisms

Conventional methods of beam sampling, such as beam splitting using partially transmissive optics, is not possible at high power levels due to the large thermal loads which are placed on the optics. Furthermore, very few materials are transparent to the far-infrared beam of CO_2 lasers. Since high power lasers with large diameter annular beams require optical windows of equivalent or greater size, the use of optics fabricated from these materials, such as ZnSe, are not cost effective. Thus, a sampling system using entirely reflective optics was investigated. Two such systems are the rotating wand beam sampler[49] and the hole grating beam sampler.[50] The hole grating beam sampler is inappropriate for this application since it samples only specific points in the beam, resulting in the loss of valuable intensity profile information. As a result, the rotating wand beam sampler, which produces a good quality sample without significantly disturbing the output , was used in this system.

2.2.1 Double-Ended Rotating Wand

The wand is usually double-ended for ease of balancing, and composed of two, small pie-shaped aluminum mirrors which have been machined on a microsurface lathe. The specifications of a single end of the wand are displayed in Fig. 2.4 (note that the wand is symmetric about the horizontal axis and contains an identical opposite end). A small, two-pole induction motor rotates the wand through the main output beam at approximately 1500 RPM. The sampled beam is reflected perpendicular to the main beam and directed onto an infrared detector, producing an annular-shaped image of the beam's cross section.

Average power coupled by a double-ended wand from the main output beam is given by the ratio:

$$\frac{P_{\text{sampled}}}{P_{\text{main}}} = \frac{2\phi}{360^\circ} \tag{2.1}$$

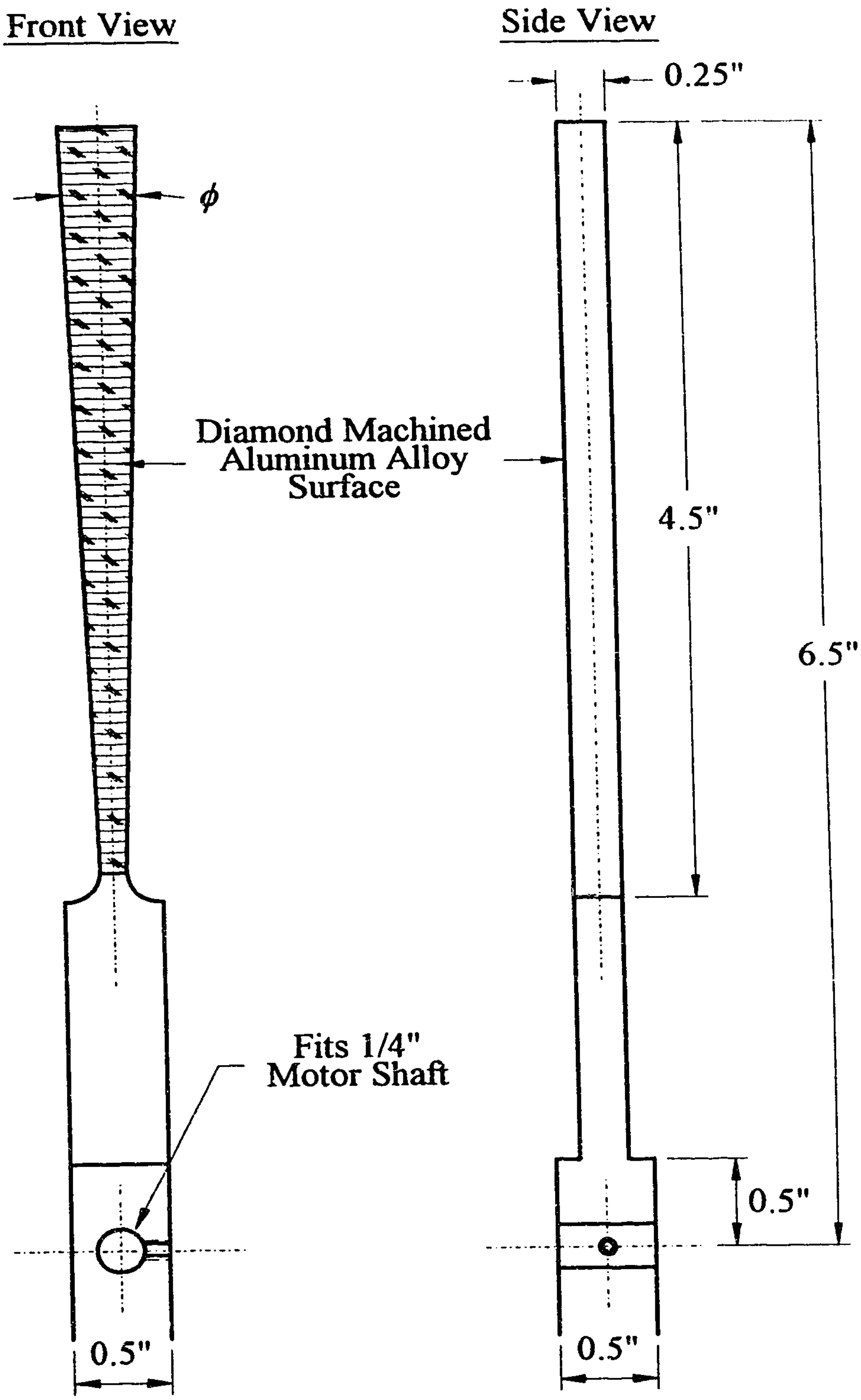


Fig. 2.4. Specifications of rotating wand beam sampler.

where ϕ (specified in degrees and defined in Fig. 2.4) is the angular width of the wand and P is the total average output power. Increasing or decreasing the angle ϕ samples a larger or smaller percentage of the main beam respectively. For use with the PIE-3 laser, an angle ϕ of 3.6° was chosen, which resulted in coupling 2% of the output. At higher power levels, this sampling percentage proved to be too large, necessitating masking of one end of the wand (1% coupling). The rotational speed of the wand has no effect on the overall power of the sampled beam. However, it must be sufficiently fast to prevent any degradation in the quality of the main output.

2.2.2 Variable Coupling Rotating Wand

Since the average operating range of the PIE-4 laser for extended periods of time varies from 1 to 15 kW, utilization of a constant percentage sampling wand would require the beam detector to have a large dynamic range and high damage threshold. Generally, far-infrared detectors which have sufficient resolution for imaging systems do not posses a large dynamic range, becoming either saturated at the upper limit or losing valuable intensity information at the lower.[51] Consequently, a variable coupling device, which can be adjusted on-line to alter the sampling percentage to correspond to the output power level, would ensure operating the beam detector in its optimal range.

A variable coupling rotating wand was developed specifically for this vision-based system and the PIE-4 laser, and consists of a standard reflective wand fitted with a cover. The two components are assembled together with a system of two sliding weights and four springs, as depicted in Fig. 2.5 (only half of the symmetric, double-ended system is shown). Centripetal forces exerted on the weights by the rotational speed of the system are counterbalanced by the spring force. The equilibrium position reached by the weights depends on the rotational speed, and this causes the cover to either open or close, since its slot is at an angle different from

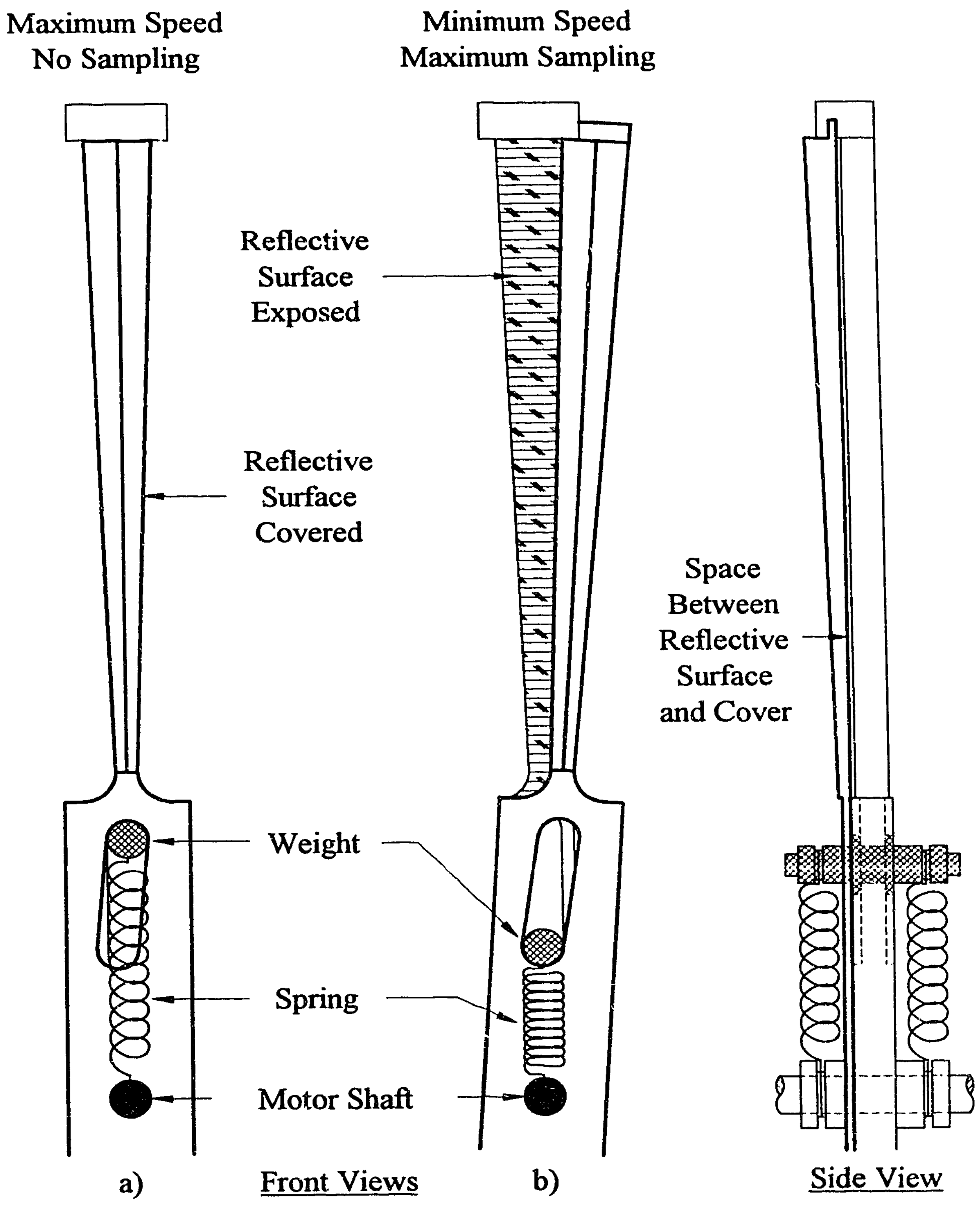


Fig. 2.5. Basic operation and assembly of variable coupling wand.

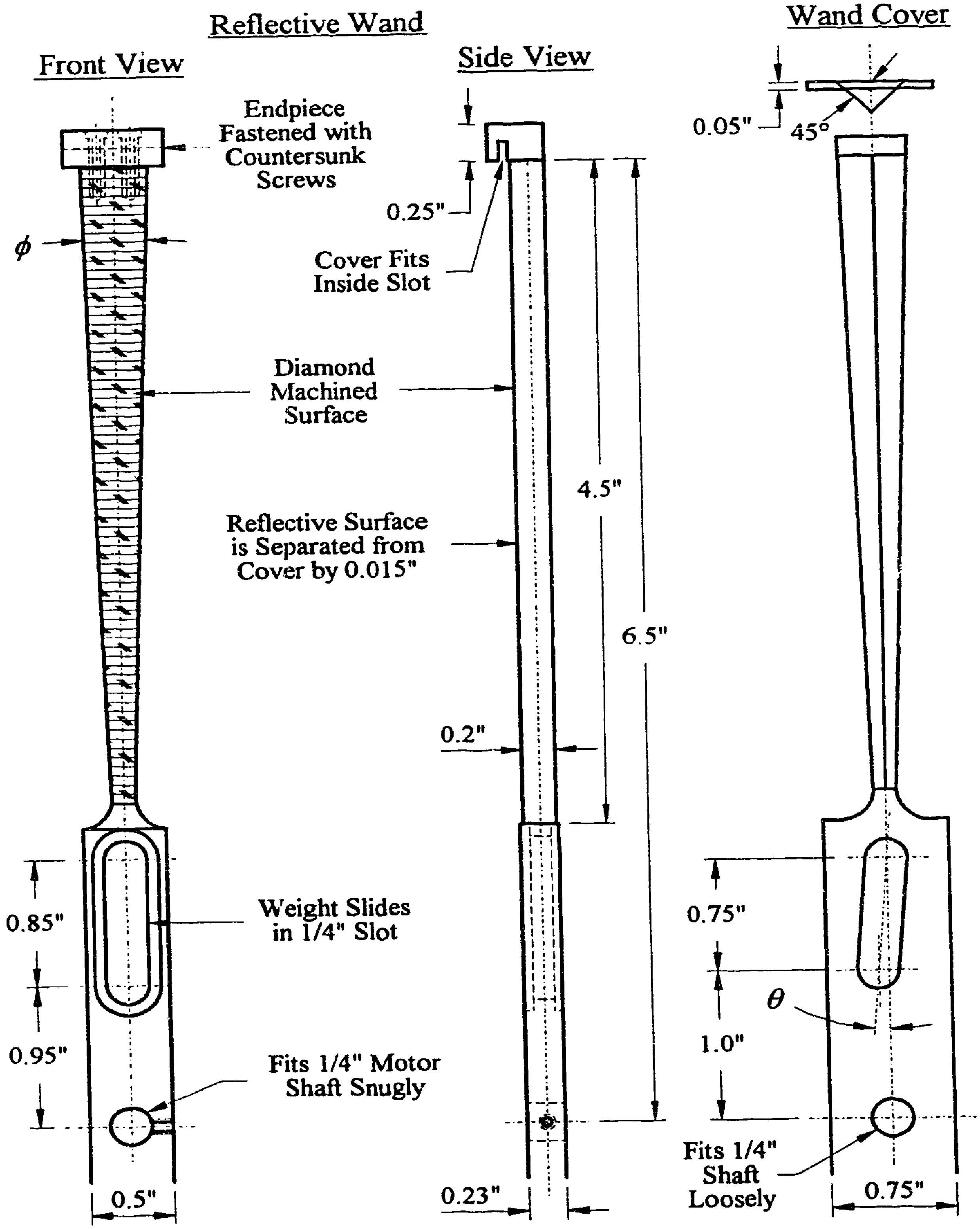


Fig. 2.6. Specifications of reflective wand and cover for variable coupler.

that of the reflective wand. At maximum speed, the weight travels to the outer end of the slots, resulting in the reflective surface being completely hidden (no sampling). Similarly, at minimum speed, the weights travel to the inner limits of the slots, coupling the maximum possible energy from the output beam. A small gap between the wand and cover prevents damage to the reflective surface as a result of their relative motion.

Specifications for the reflective wand and cover are illustrated in Fig. 2.6. The angle of the cover's slot, θ, is 4.8°, which corresponds to the ϕ value of 3.6°, and can be computed from the associated dimensions as follows:

$$\theta = \tan^{-1}\left(\frac{\tan\phi}{0.75}\right). \qquad (2.2)$$

The cover's surface was sandblasted to disperse the blocked radiation in random directions away from the imaging screen. The cross section of the cover is an isosceles triangle, with the congruent angles being 45°. This is a design requirement, since the angle of the plane of rotation with respect to the beam, in order to obtain a perpendicularly propagating sample, is 45°. Fig. 2.7 graphically presents the relationship between the wand's plane of the rotation, β, and the cross-sectional design of the cover. As depicted, the congruent angles of the cover must not exceed β. In fact, it is desirable to construct the angles of the cover about 10° less than β, to avoid problems with grazing incidence.

The system was designed to operate in a speed range of 500 to 750 RPM, which is sufficiently fast to prevent degradation of the output, yet adequately slow to prevent undesired effects from friction and air resistance. Based on this requirement, the mass of the sliding weights and spring constants were calculated based on the following equilibrium condition:

$$F_{\text{centripetal}} + F_{\text{spring}} + F_{\text{friction}} = 0. \qquad (2.3)$$

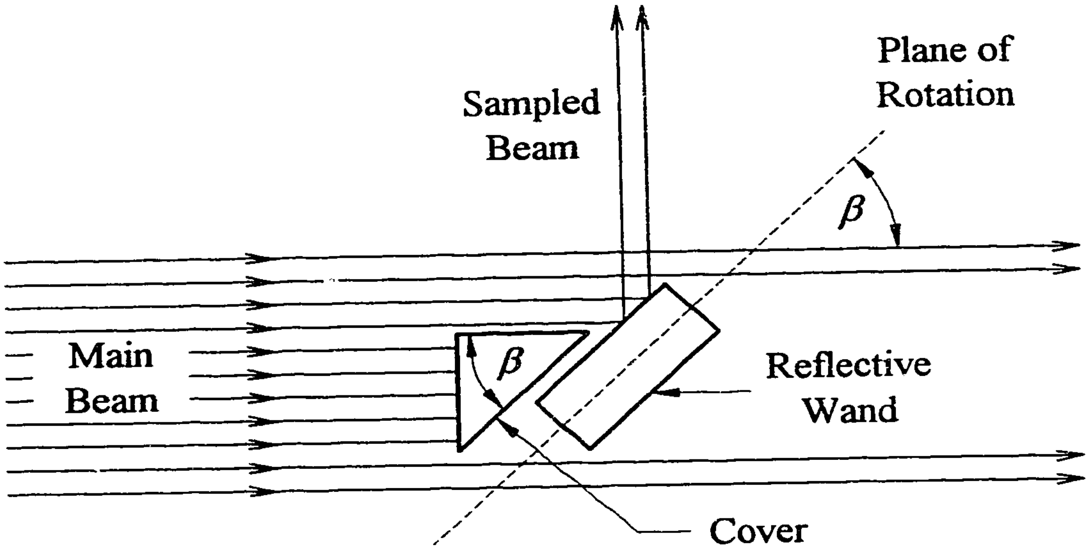


Fig. 2.7. Relationship between plane of rotation of variable coupling wand a'id cross-sectional design of cover.

Neglecting friction and substituting the force formulas, the equation simplifies to:

$$m\omega^2 r = 2k(r - x_0) \tag{2.4}$$

where m is the mass of the a single weight, ω is the rotational speed, r is the distance of the weight from the center of rotation, k is the spring constant, and x_0 is the relaxed length of the spring. Arbitrarily selecting springs with constants of 40 N/m and substituting the design requirements for ω and r, the mass of the weights were computed to be 7.45 g each. Upon construction, the exact mass and spring constant were measured to be 7.25 g and 37.8 N/m respectively. Fig. 2.8 illustrates the resulting dependence between the coupling percentage and rotational speed. As evident, the relationship between the two is linear with a speed range of 540 to 720 RPM, which agrees with the original design predictions. The rotational speed and the corresponding percentage opening were measured with an infrared LED and phototransistor assembly. Pulses from this sensor were monitored by computer, their width and period were measured, and the resulting coupling percentage was

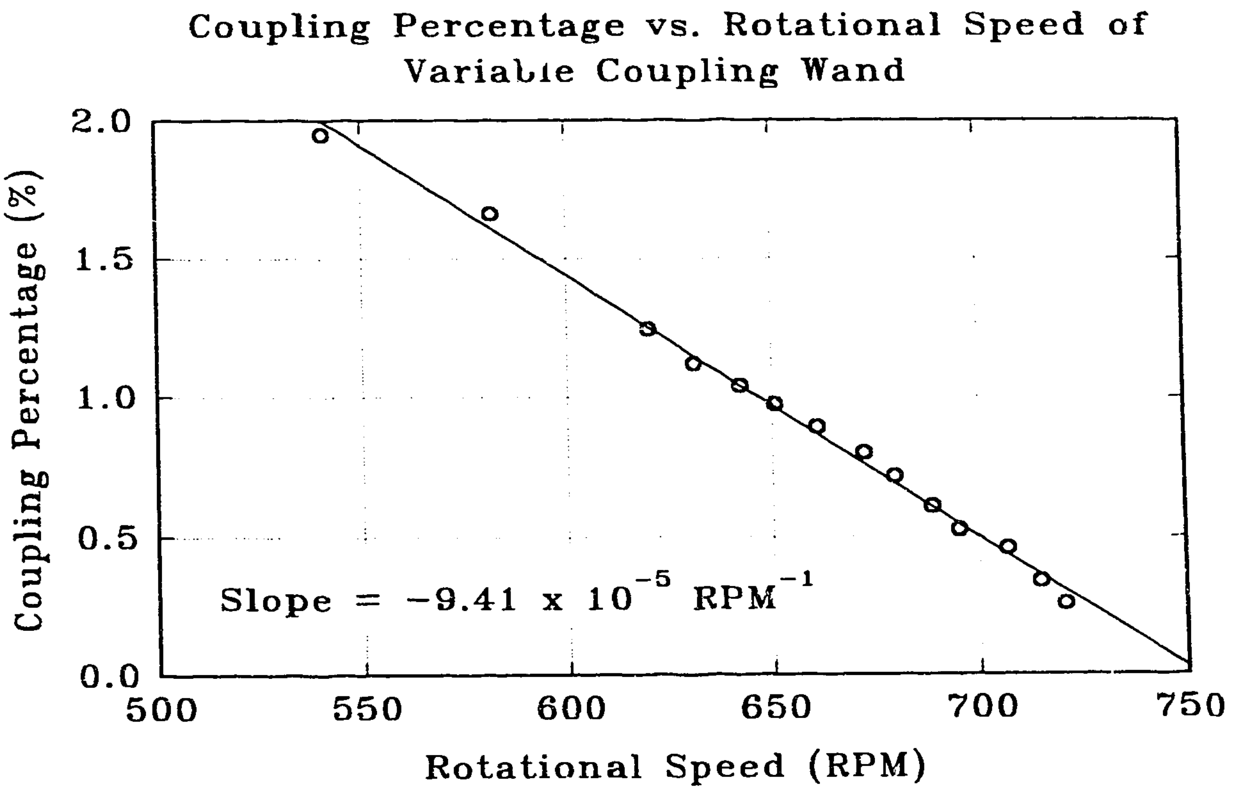


Fig. 2.8. Performance characteristics of variable coupling rotational wand.

calculated. Each data point represents the average of 25 measurements, and the error bars are smaller than the size of the data points (this applies to all following graphs).

The speed regulator for the AC induction motor consisted of a feedback controller, which utilized the pulse from the LED-phototransistor sensor. This signal was amplified, integrated, and compared to a setpoint (from 0 to 15 V) initiated by the computer. Signal error was used to adjust the motor speed, via a variable speed motor driver, to maintain the desired coupling percentage. Fig. 2.9 describes the characteristic linear relationship between the speed controller setpoint and the rotational speed. Detailed circuit schematics are presented in Appendix A.

Controller response was tuned experimentally to be sufficiently robust, with no significant overshoot. Consequently, no measurable delays or disturbances were added to the performance of the vision-based laser controller. A sample closed loop

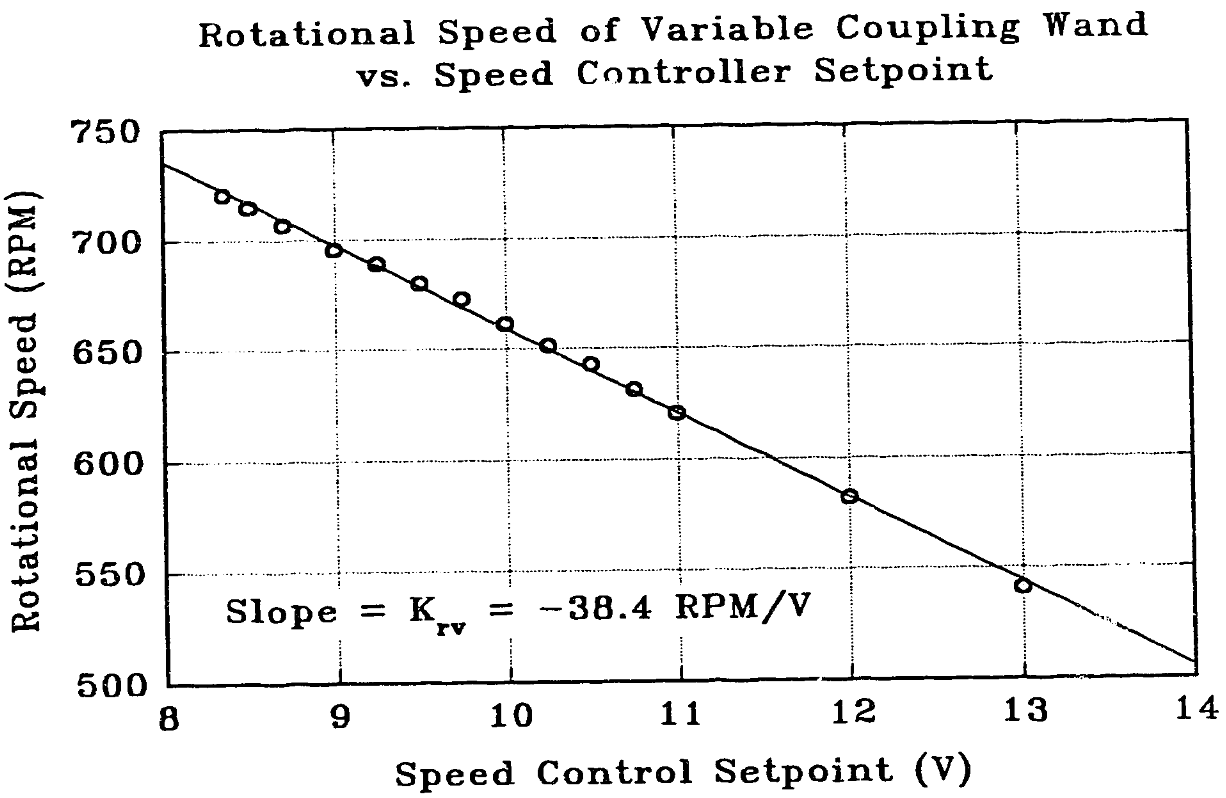


Fig. 2.9. Characteristics of AC induction motor speed controller for variable coupling wand.

step response is presented in Fig. 2.10. A first order model of the closed loop system was developed from the input and output data, sampled at one second intervals, using the batch least squares technique (detailed later in Section 6.4.1).[52] The discrete model was converted to the following continuous-time closed loop transfer function (W is the rotational speed of the wand and U_w is the controller setpoint):

$$G(s) = \frac{W(s)}{U_w(s)} = \frac{K_{rv}e^{-T_d s}}{1+\tau s} = \frac{-38.4e^{-s}}{1+0.141s}, \qquad (2.5)$$

and the resulting model-based data, along with the raw values and setpoint change, are graphed in Fig. 2.10. The raw rotational speed data contains excessive noise

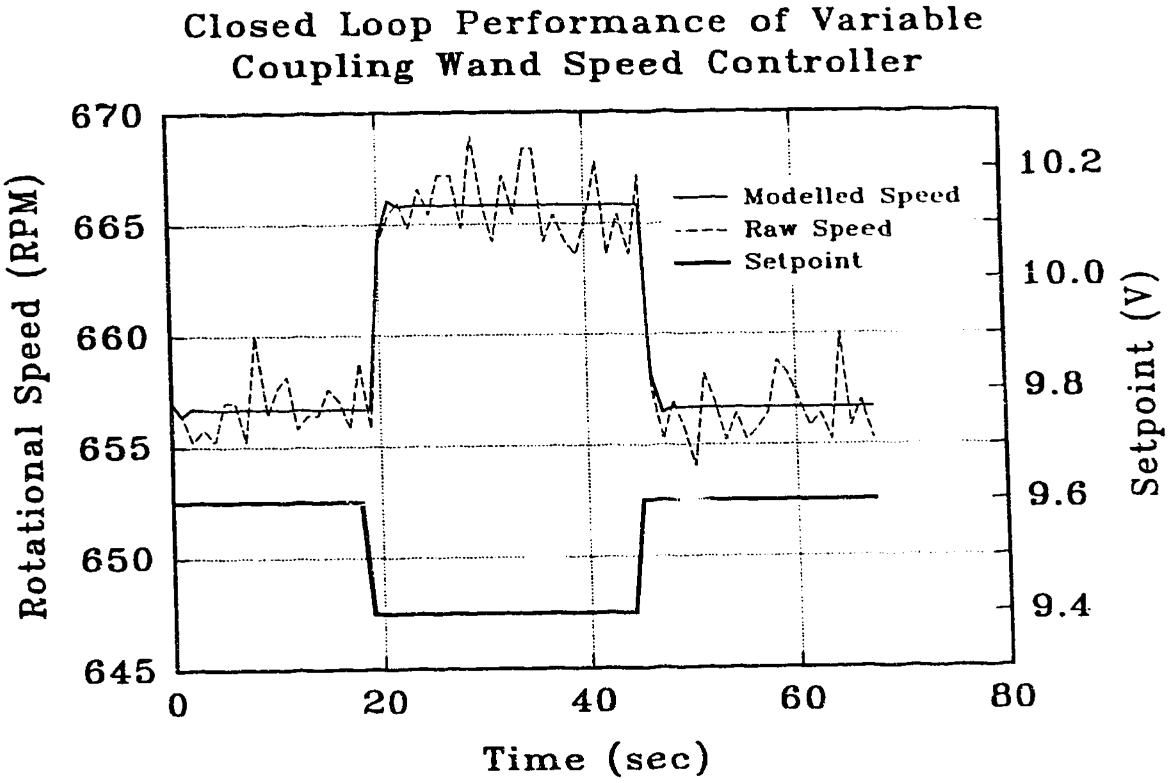


Fig. 2.10. Closed loop step response of rotational speed controller.

since no averaging or filtering was performed. However, as evident from the graph, the modelled output is consistent with the raw data for a 0.2 V setpoint change. The delay (T_d) of 1 second due to sampling and the closed loop time constant (τ) of 0.141 seconds is sufficiently fast for most feedback control systems. Since the infrared detector used in this system (described later in Section 2.3) possesses a time constant of approximately 3.5 seconds for global intensity changes (25 times larger than that of the speed controller), no measurable delays or disturbances were added to the performance of the laser control system.

2.3 Infrared CO_2 Beam Detectors

As specified previously, the wavelength of a CO_2 laser beam is 10.6 μm, which is in the far-infrared region of the electromagnetic spectrum. Thus, before the image of the sampled beam's cross section can be viewed by a standard video camera (which is only sensitive to visible light), it must be detected using an infrared to visible conversion device. Alternately, this external conversion may be bypassed by employing an infrared camera. A number of options were tested, and evaluated based on their time response, linearity, dynamic range, damage threshold, and cost efficiency.

The first technique tested was asbestos cloth, which glows a bright orange colour as it is heated by the laser beam. Only the annular region corresponding to the beam's cross section will be heated, and its intensity profile may be easily observed. However, the minimum power density required for asbestos cloth to illuminate is approximately 32 W/cm^2. Since the average power density of the PIE-3 output beam is about 90 W/cm^2, a 36% sample would be required, resulting in highly inefficient operation of the laser. In addition, asbestos is associated with certain health hazards which deem it unfavourable for general use in an closed environment.

A second and similar method is to use a sheet of liquid crystal which is sensitive to changes in temperature. The sheet changes colour in the region heated by the beam, enabling the beam's shape and power distribution to be determined. Although this approach has its merits, the dynamic range of the sheets is only about 5 °C, insufficient for use with a 1% beam sample. Furthermore, the changes in colour do not correspond to equal changes in gray levels, as seen by the digitizer, leading to an extremely non-linear intensity to power mapping.

The third option is to replace the standard video camera and infrared detection apparatus with an infrared vidicon camera. A pyroelectric camera, with a spectral sensitivity from 2 to 20 μm, a temperature range from 20 to 40 °C, and a sensitivity of 0.2 °C, was tested.[53] Since the damage threshold of the camera was 100 mW/cm^2, a 0.1% sample of the output beam was used. However, the input to the camera required chopping, since pyroelectric devices respond only to changes in temperature. The natural chopping frequency of the sampling wand (between 20 and 50 Hz for a double-ended wand) did not provide adequate recovery time for the pyroelectric crystal. In addition, a uniform and consistent cross-sectional beam image could not be obtained since all portions of the beam could not be sampled instantaneously. Consequently, a fading effect was noted across the image - the section being sampled first being less intense, while the portion sampled last being more intense. An added disadvantage is the relatively high cost and delicacy of this instrument.

The final technique is to utilize the concept of thermal quenching of induced fluorescence.[54] Certain phosphors and fluorescent materials, which fluoresce when irradiated by ultraviolet light, have their fluorescence diminished as the local temperature increases. After testing a number of these compounds, the phosphor used in the thermal imaging screens manufactured by Optical Engineering proved to possess the best features.[55] The particular screen used in these experiments had a sensitivity range between 0.2 and 1.2 W/cm^2, a response time of 0.4 seconds, and a resolution of approximately 50 lines/cm. These specifications are dependent on the amount of cooling employed and the properties of the heat sink. Spatial restrictions necessitated that the video camera be mounted behind the thermal imaging screen, requiring the use of a visibly transparent heat sink capable of withstanding the high thermal stresses. The material best fulfilling these criteria is fused quartz, which has a thermal conductivity of 0.0033 cal$\cdot$cm$^{-1}\cdot$s$^{-1}\cdot$°C^{-1}. In order to obtain the above mentioned characteristics, the imaging screen was stretched (about 0.7 cm on each side) onto a 1/16" thick fused quartz plate and air-cooled with approximately 15 to 20 psi of pressure. The resulting overall screen time constant ranged between 1.5 and 4.0 seconds, depending on the size and type of step change. Step changes which require large modifications to screen intensity (global temperature changes across the entire screen and heat sink), such as those for output power and coupling percentage, exhibit longer time constants than those necessitating smaller, local variations in intensity.

Under these conditions, fluorescence quenching of the bright yellowish-orange screen was found to be directly proportional to the average laser power, provided that the screen was irradiated with power levels ranging from 15 to 40 W for a beam size corresponding to that of the PIE-4 laser (as shown in Fig. 2.11). The total intensity of the image is measured as the sum of gray values (gv) of all pixels which constitute the beam's cross section.

Of the above summarized methods, the thermal imaging screen satisfied all requirements and was employed in the final design for both the PIE-3 and PIE-4

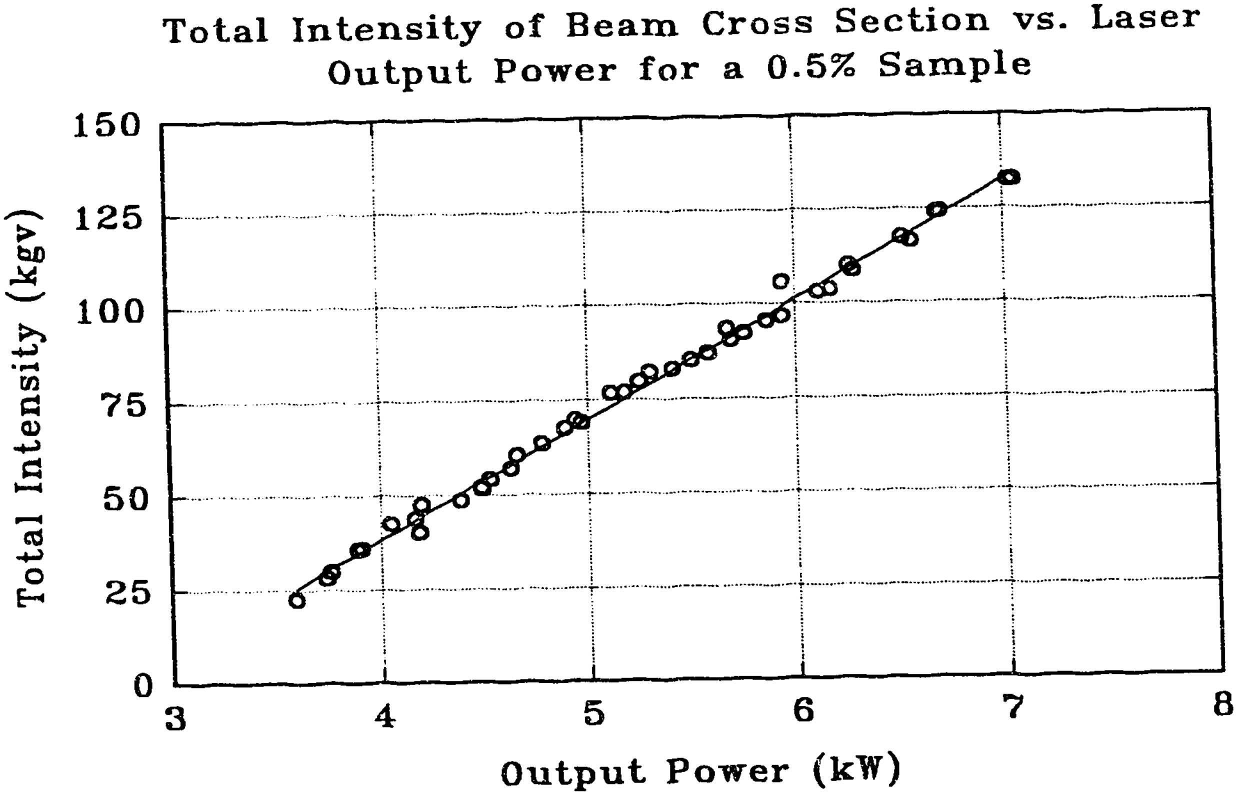


Fig. 2.11. Linearity of thermal imaging screen when utilizing a 0.5% sample of the PIE-4 laser output.

lasers. This resulted in an approximately linear relationship between the laser output power and overall darkness of the beam image, thereby enabling the use of the screen as a means of on-line power measurement.

2.4 Image Digitization and Analysis System

Video imaging of the thermal viewing screen is accomplished with a Hitachi VK-C2000 colour video camera, possessing a horizontal resolution of 260 lines. The video signal so derived is digitized by a LIVE 2000 frame grabber[56] at a speed of 15 frames/s into 320 × 200 pixel images having 16 gray levels. Thresholded binary images can be obtained at a rate of 60 frames/s. The frame grabber possesses two feedthrough camera inputs, with the software switching capability to select either. An Amiga 2000 personal computer (a Motorola 68000 based machine operating at 8 MHz) analyses this digitized beam image data and compares it to preset references. The Amiga 2000 PC was chosen for its multitasking capabilities, its on-board dedicated graphics processing hardware, and its relatively low cost frame-grabbing and imaging accessories. The system was later upgraded with an accelerator card containing a Motorola 68030 main processor and a 68882 floating point unit, operating at 25 MHz.

The external laser signals are monitored and controlled via the ACDA Proto-40K data acquisition unit, which contained 16 A/D inputs lines (12-bit resolution), 16 digital inputs, 16 digital outputs, two 8-bit D/A outputs, and three 16-bit digital timers.[57] Since this device did not possess sufficient capability to interface with all the necessary controls and signals, a supplementary "piggy-back" board was designed to perform additional switching, multiplexing, and processing functions. The physical size of the board was restricted by spatial limitations within the computer. As a result, programmable logic devices were used to limit the number of microchips required and permit future modification of necessary logic.

The final design expands data acquisition capacities to 52 digital outputs and 26 digital inputs. It also incorporates a 16-bit counter which interfaces with the motormic drivers to maintain the position of the micrometers within 0.1 μm. Schematic diagrams and further operational explanations of this unit are supplied in Appendix B.

2.5 Resonator Optics Alignment System

Both convex and concave resonator mirrors can each be positioned using two motormics,[58] which permit angular motion about the horizontal and vertical axes passing through their centres. The motormics possess a built-in digital positional readout, as well as a computer interface, enabling continual tracking of their actual motion. However, the computer interface does not provide positional information in an absolute form. Instead, a digital pulse is produced each time the motormic travels 0.1 μm, with the direction of motion indicated by the state of another digital output. Consequently, the computer must continually monitor the "count" line when a motormic is in motion and compute the actual position, consuming valuable computer processing time. This inefficiency was eliminated with the introduction of the above mentioned "piggy-back" board. The on-board 16-bit counter maintains the position of any one motormic or may be preset to move a motormic a given distance. Thus, the counter could be read directly in one memory operation, and the absolute position of the motormic may be ascertained. The limitation of this method is that only one motormic may be adjusted at any one time.

The motormics did not possess sufficient torque to directly overcome the spring forces of the optical mounts in the PIE-3 laser. Thus, a differential micrometer, with a gear reduction ratio of 50:1, was introduced. The motormics were used to drive the differential micrometers, which in turn moved the optics. In effect, these micrometers converted the transverse load experienced by the motormics to an axial one, reducing the overall required torque. The additional gear reduction permitted the mirrors to be controlled with much greater resolution. Based on the gear ratios and moment arms, the correlation between the transverse motion of the micrometer and the angular movement of the optics was determined to be 0.282 rad/m. For the PIE-4 laser, the torque of the motormics was sufficient to directly modify the angle of the optics, due to the larger moment arm. However, this direct-drive method increased hysteresis significantly as a result of the larger load on the motormics. Consequently, backlash compensation was incorporated in software by correspondingly increasing movemenıs when changes in direction were required. Since the design of the optical mounts differed from the PIE-3 laser, the angular conversion factors were 0.787 rad/m for both axes of the convex mirror, 1.432 rad/m for the horizontal axis of the concave mirror, and 1.167 rad/m for the corresponding vertical axis. For simplicity, all resonator optic positions will be specified in microns, since this was the directly measured quantity.

2.6 Laser Input Power Controllers

Adjusting the total output power requires modification to the laser excitation level. In the PIE-3 laser, this is accomplished via a motorized variac in its DC power supply. The computer interfaces to the variac via two relays, which rotate the variac in opposite directions. Thus, the input power to the laser can be increased by activating the corresponding relay. The slight backlash encountered while changing the direction of motion of the variac (due to mechanical limitations), was compensated for by the control algorithm. The PIE-4 laser incorporates an SCR controlled power supply, which enables a more rapid adjustment of the excitation level. The output power level is initially set with a manually controlled potentiometer. The computer may vary this level by approximately ±2.5 kW by providing a 0 to 15 V setpoint to the laser power controller.

CHAPTER 3

SYSTEM SOFTWARE DESIGN

The adaptability and expandability of this computer vision control and diagnostic system are its major advantages over conventional electronic controllers. The supervisory and control strategies, entirely software-based, may be altered to satisfy desired modes of operation on a variety of laser systems. Furthermore, on-line modification of the controller gains and associated constants facilitates optimal performance under conditions in which process parameters may vary significantly.

The system software consists of the main laser controller program, 2D and 3D intensity profile generating programs, and several utilities for image conversion, colourization, display, and printing. The majority of the software is written in the C programming language (ANSI standard),[59] with certain memory and computationally intensive routines coded in Assembly for increased execution speed.[60] All built-in Amiga ROM functions,[61-64] required to manage internal devices and utilize the graphical interface capabilities, may be accessed directly using the standard C function calling protocol.

3.1 Laser Controller Algorithm

The control system program may be subdivided into five distinct areas: initialization and input validation, image digitization and conversion, error signal computation and filtering, output control signal generation, and data presentation and storage. A simplified flowchart describing the control algorithm's operation is illustrated in Fig. 3.1. Upon initialization, the presence of a valid video signal is established, the digitization levels are set, and a single frame is digitized. Based on

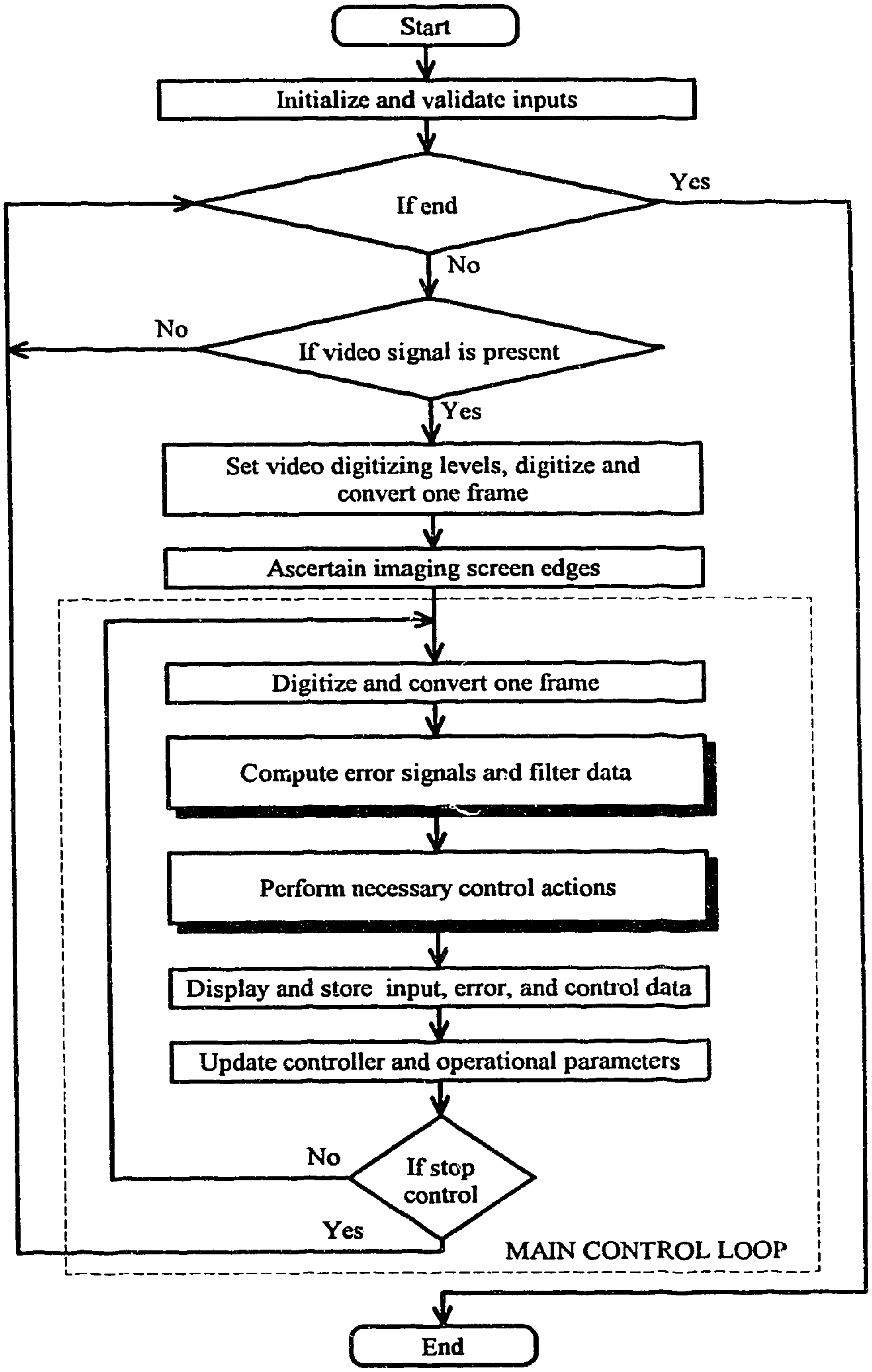

Start
Initialize and validate inputs
If end
Yes
No
No
If video signal is present
Yes
Set video digitizing levels, digitize and convert one frame
Ascertain imaging screen edges
Digitize and convert one frame
Compute error signals and filter data
Perform necessary control actions
Display and store input, error, and control data
Update controller and operational parameters
No
If stop control
Yes
MAIN CONTROL LOOP
End

Fig. 3.1. Simplified flowchart of laser control and diagnostic algorithm.

this image data, the edges of the imaging screen are ascertained and the screen position is determined. The algorithm then enters the control loop, in which a single frame of the cross-sectional beam image is obtained, the necessary error signals are calculated from and filtered (if required), and the prescribed control actions are implemented. The image data, along with the corresponding error values and controller outputs, are displayed in numerical or graphical form and may be stored for later analysis. Any modifications to the operational and controller parameters are updated, and the cycle is repeated until execution is terminated. Expanded flowcharts of the shadowed blocks in Fig. 3.1 are presented later in this chapter in Fig. 3.4 and Fig. 3.5.

3.1.1 Initialization and Input Validation

Following the launch of the control program, the opening sequence initializes pertinent system hardware, including the data acquisition unit, frame grabber, audio processing device, and system timers. All controller output signals are set to their null states to prevent uninitiated control actions from occurring. A customized configuration file provides the program with information necessary to initialize operational and controller parameters. Different configurations files may be created for use under various operating conditions, enabling a greater degree of automation and flexibility. Upon successful validation of all inputs, the algorithm initializes the image memory maps and graphical user interface required to display image and error signal information. Any faults which occur during the initialization phase must be corrected before proceeding with subsequent execution.

3.1.2 Image Digitization and Conversion

The quality and resolution of the digitized image are dependent on the overall brightness and contrast of the video signal (i.e. its amplitude range). An average, medium brightness image does not occupy the entire available amplitude range. Thus, digitizing to the extents of the amplitude limits results in many of the 16 discrete gray levels being empty. The LIVE 2000 permits adjustment of the maximum and minimum values of the amplitude range, known as the ceiling and floor respectively, to ensure all 16 digitization levels contain information. As shown in Fig. 3.2, the ceiling may be shifted from 100% to 25% of the maximum possible signal in 16 steps, while the floor may be changed from 0% to 75% of the maximum attainable voltage in 16 steps. The A/D range between the ceiling and floor is discretized into 16 levels, with voltages above the ceiling interpreted as full white (gray value 15), while those below the floor are defined as full black (gray value 0). Since the colour of the imaging screen is yellowish-orange and not white, the digitizing ceiling must be initially set to the brightness of the screen, providing a uniform, white background. Similarly, the floor is set such that the darkest areas of the beam cross section are digitized as full black, producing a 16 gray level image.

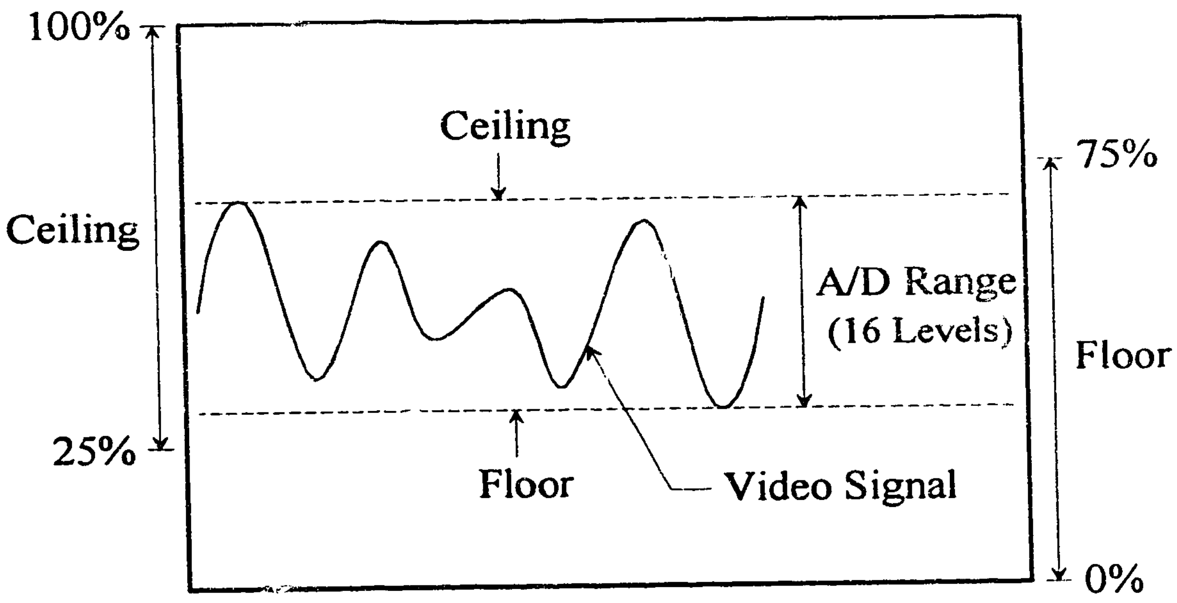


Fig. 3.2. Video signal digitization settings to obtain optimal gray scale distribution.

Once a single video frame has been digitized, the image information is converted to an easily interpretable format to facilitate use of standard image processing functions. The LIVE 2000 utilizes a "cyclic encoding" scheme, which possesses the properties of a standard Grey code,* in order to reduce digitization errors due to noise.[65] It also places the image in a bitplane format, as required for display by the Amiga hardware. Consequently, the image data is first translated to a binary-encoded form, enabling arithmetic operations to be performed on the data, and then converted to raster format, which permits elementary and rapid addressing of individual pixels.

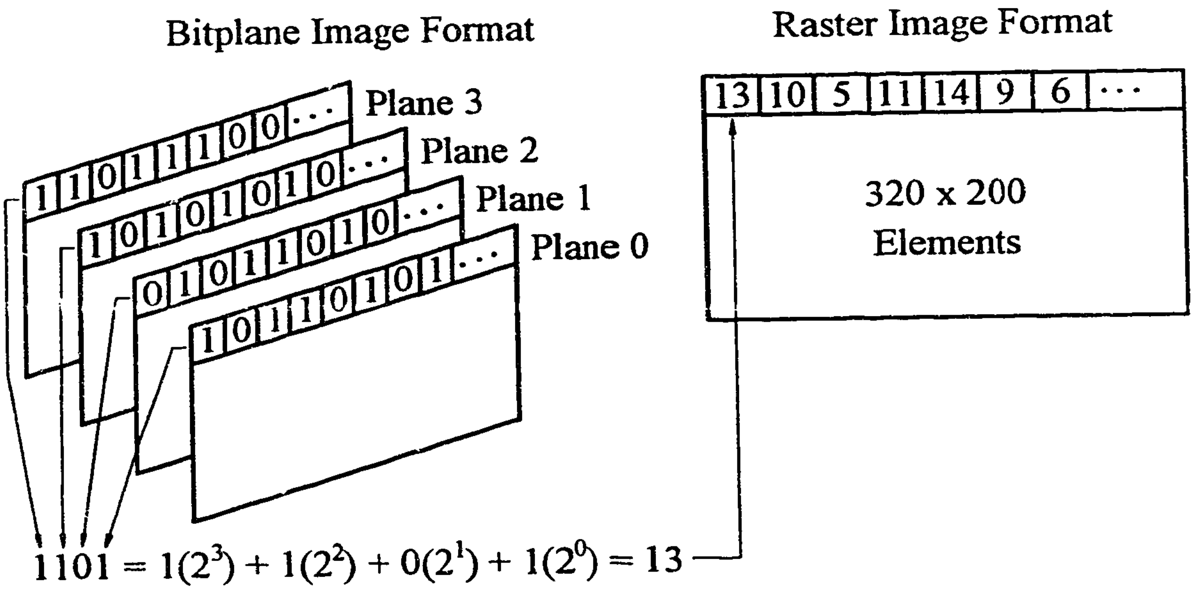


Fig. 3.3. Bitplane to raster image format conversion technique.

The bitplane to raster conversion technique is illustrated in Fig. 3.3. Each bitplane contains data corresponding to a particular bit for the entire image. In order

* Grey codes possess the property that each member of the code differs from the preceding and succeeding numbers by only one bit. Thus, if a bit error occurs due to noise, the resulting encoded value will deviate from the actual by not more than one.

to obtain the intensity of a given pixel, the corresponding bits from each bitplane are multiplied by the bitplane weighting factor (2 to the power of the bitplane number) and summed together. Software to perform this conversion is extremely inefficient if written in C, requiring approximately 21.5 seconds for execution (with the 8 MHz microprocessor). However, re-written in Assembly to optimize internal register usage and minimize external memory access, the code executes in 1.06 seconds; an increase in speed of almost 2000%.

3.1.3 Error Signal Computation

The 320 × 200 pixel, binary-encoded image raster is used to compute all required error signal information. Fig. 3.4 illustrates the error computation routine. Outer edges of the beam are determined using a customized edge detection algorithm. Since the image contains only 16 gray levels, adjacent pixels which differ by more than one gray level generally do not exist. Consequently, conventional edge detection filters do not perform well since the filter threshold must be set at a relatively low level.[66] The edge detector assumes the digitizing ceiling has been set such that the image background (the imaging screen colour) is completely white. It commences at the screen boundaries and searches inward until a non-white pixel is encountered, at which point, it examines the 3 × 3 pixel matrix surrounding this non-white pixel. If the matrix contains less than 6 non-white elements, the pixel is characterized as noise, and the algorithm continues searching. Once the horizontal and vertical outer beam diameters are ascertained, the inner X and Y diameters are determined by commencing at the centre (as defined by the outer diameters) and probing outwards. The midpoints of the outer and inner diameters are averaged to obtain the beam centre. If the resonator is sufficiently misaligned such that only a partial beam exists, the algorithm may be unable to compute diameters, or they may be smaller than preset limits. In such cases, previously valid values (which could be

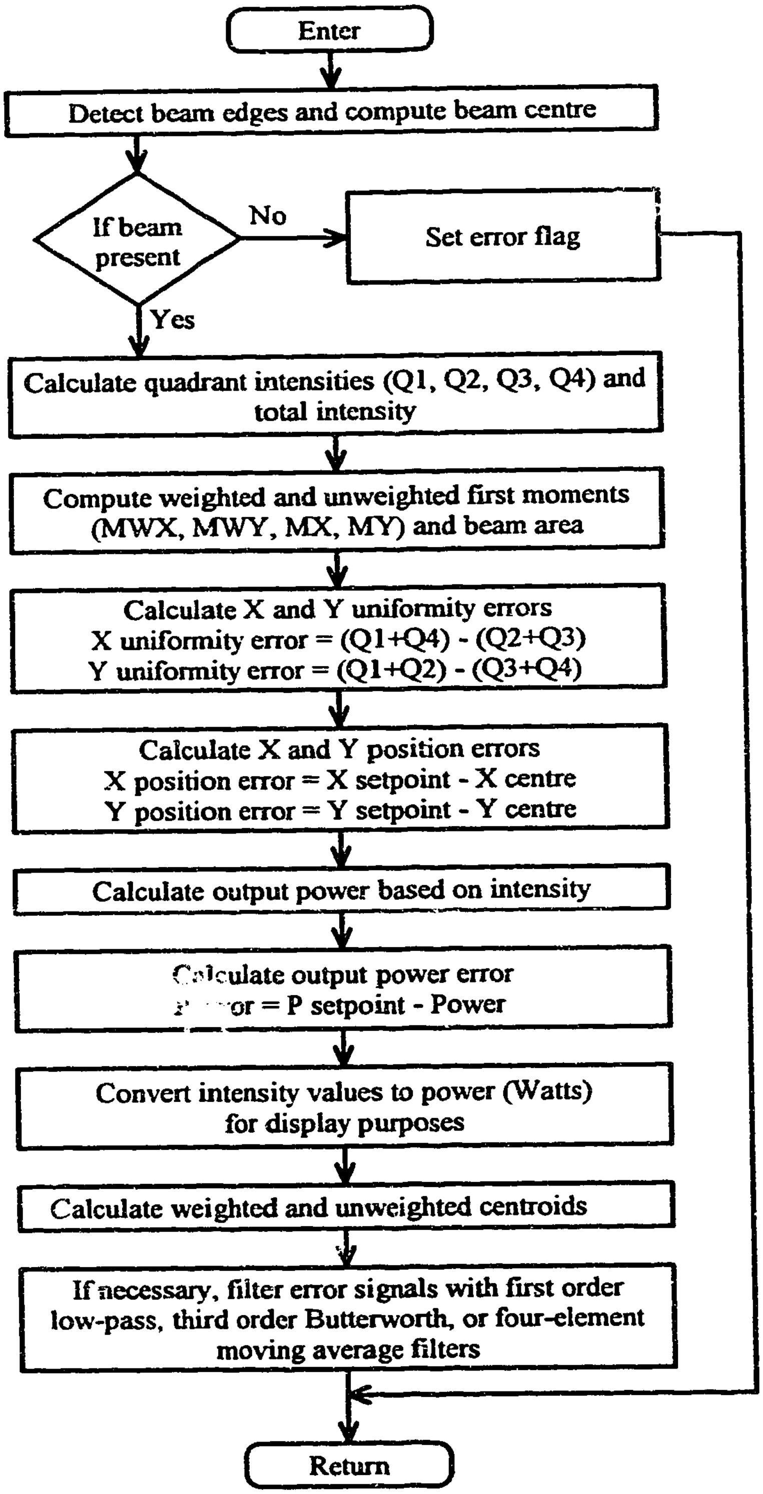

Enter
Detect beam edges and compute beam centre
If beam present
No
Set error flag
Yes
Calculate quadrant intensities (Q1, Q2, Q3, Q4) and total intensity
Compute weighted and unweighted first moments (MWX, MWY, MX, MY) and beam area
Calculate X and Y uniformity errors
X uniformity error = (Q1+Q4) - (Q2+Q3)
Y uniformity error = (Q1+Q2) - (Q3+Q4)
Calculate X and Y position errors
X position error = X setpoint - X centre
Y position error = Y setpoint - Y centre
Calculate output power based on intensity
Calculate output power error
P error = P setpoint - Power
Convert intensity values to power (Watts) for display purposes
Calculate weighted and unweighted centroids
If necessary, filter error signals with first order low-pass, third order Butterworth, or four-element moving average filters
Return

Fig. 3.4. Flowchart of error signal computation algorithm.

the defaults) are used.

The image is then partitioned into four quadrants (Q_1, Q_2, Q_3, and Q_4), with the beam centre at the origin of an X-Y coordinate system. Relative intensities in each quadrant are determined by summing the gray levels of all pixels within the specific quadrant. The routine then computes the weighted and unweighted first moments of inertia (MW_i and M_i respectively) according to the following equations:

$$MW_x = \sum_{x,y} (\text{Intensity}_{xy})(x) \tag{3.1}$$

$$MW_y = \sum_{x,y} (\text{Intensity}_{xy})(y) \tag{3.2}$$

$$M_x = \sum_{x,y} (x \text{ coordinate of all pixels in beam image}) \tag{3.3}$$

$$M_y = \sum_{x,y} (y \text{ coordinate of all pixels in beam image}) \tag{3.4}$$

as well as the total beam area (number of pixels comprising the beam's cross section). These memory intensive computations are coded in Assembly, yielding a 487% increase in execution speed. From these values, the weighted and unweighted centroids, C_i, are calculated as follows:

$$\text{Weighted } C_i = \frac{MW_i}{\text{Total Intensity}} \tag{3.5}$$

$$\text{Unweighted } C_i = \frac{M_i}{\text{Beam Area}}. \tag{3.6}$$

A uniformity error for the X direction is computed by subtracting the intensity of the negative X plane from that of the positive (i.e. $[Q_1+Q_4]$ - $[Q_2+Q_3]$). Similarly, an error for the Y direction is determined by subtracting the intensities of Q_3 and Q_4 from Q_1 and Q_2. Beam positional error is ascertained by calculating the difference between the X and Y coordinates of the beam centre and the desired centre setpoint. The algorithm then computes the error in output power by translating the total intensity of the beam image ($Q_1+Q_2+Q_3+Q_4$) to Watts (based on a predetermined relationship), and comparing it to a preset value. If necessary, higher frequencies of the error signals may be eliminated by applying a first order low pass, third order Butterworth, or four-element moving average filter.[67]

3.1.4 Output Control Signal Generation

A generalized flowchart of the control signal generating program sequence is described in Fig. 3.5. Uniformity errors are converted to micrometer motion (specified in microns) by multiplication with the appropriate controller gain factor. Offsets may be added to the intensity errors to compensate for permanent non-uniformities in the beam. If either of the resulting movements exceeds a preset minimum, the larger of the two is executed by computer activation of the motormic corresponding to the uniformity control mirror (usually the convex). In order to prevent interaction between the two alignment axes, and due to hardware limitations, only a correction for the largest error is made during a single control loop. If both errors are within minimum thresholds, indicating that the output beam is uniform in power distribution, the X and Y positional errors are examined, and the controller compensates for the larger of the two, with appropriate motion of the position control mirror (the concave resonator mirror). However, this compensation is performed in small steps, to prevent significant disturbances to beam uniformity. Thus, a typical control sequence involving both uniformity and positional mirror manipulation would entail complete optimization of uniformity with the first mirror followed by a small adjustment to the second mirror to reduce positional error. This process would be repeated until both errors are eliminated. Since an accurate measure of the beam centre can only be obtained under conditions of optimal uniformity, positional control is disabled if an error in uniformity exists.

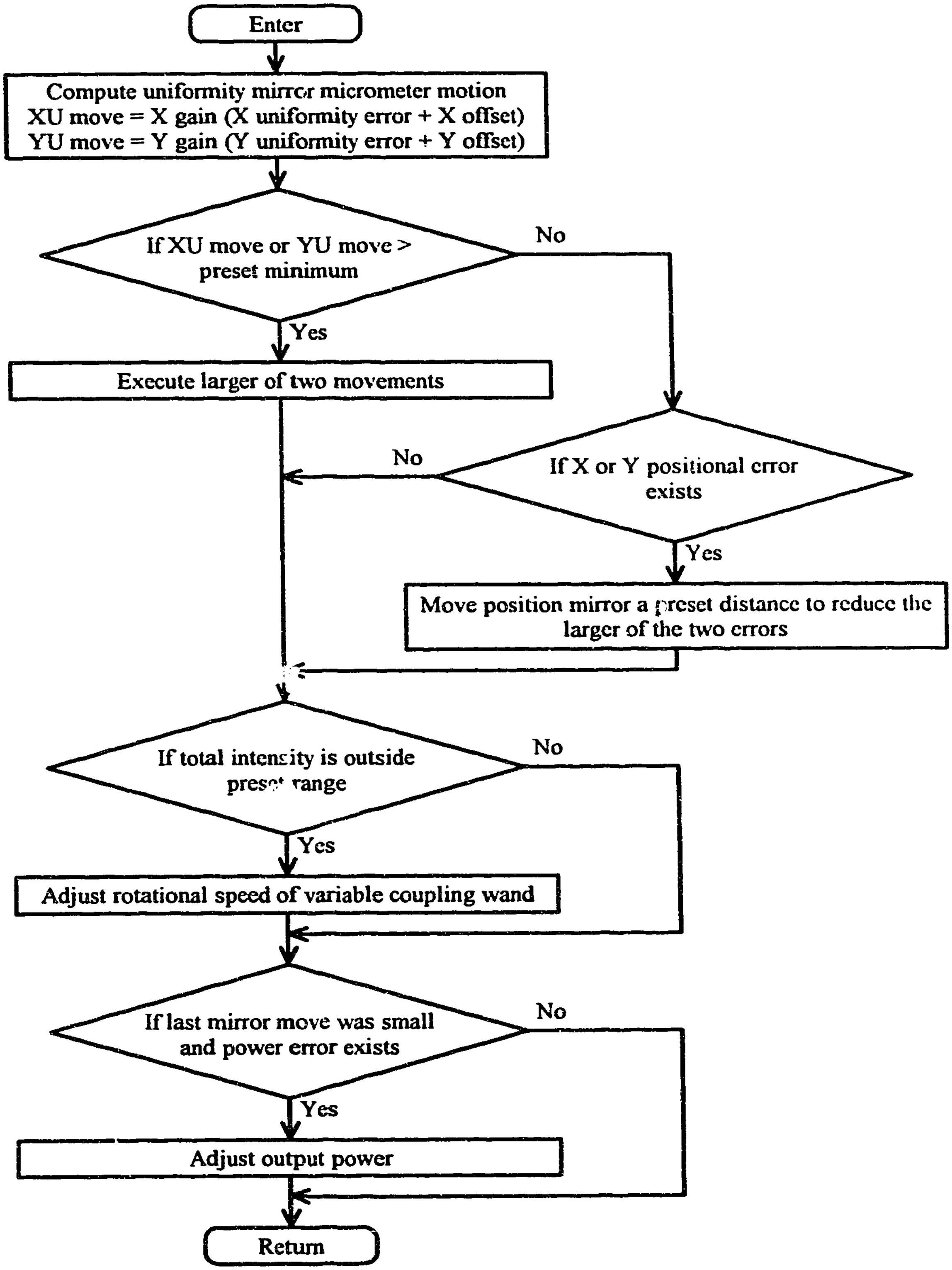

Enter
Compute uniformity mirror micrometer motion
XU move = X gain (X uniformity error + X offset)
YU move = Y gain (Y uniformity error + Y offset)
If XU move or YU move > preset minimum
No
Yes
Execute larger of two movements
If X or Y positional error exists
No
Yes
Move position mirror a preset distance to reduce the larger of the two errors
If total intensity is outside preset range
No
Yes
Adjust rotational speed of variable coupling wand
If last mirror move was small and power error exists
No
Yes
Adjust output power
Return

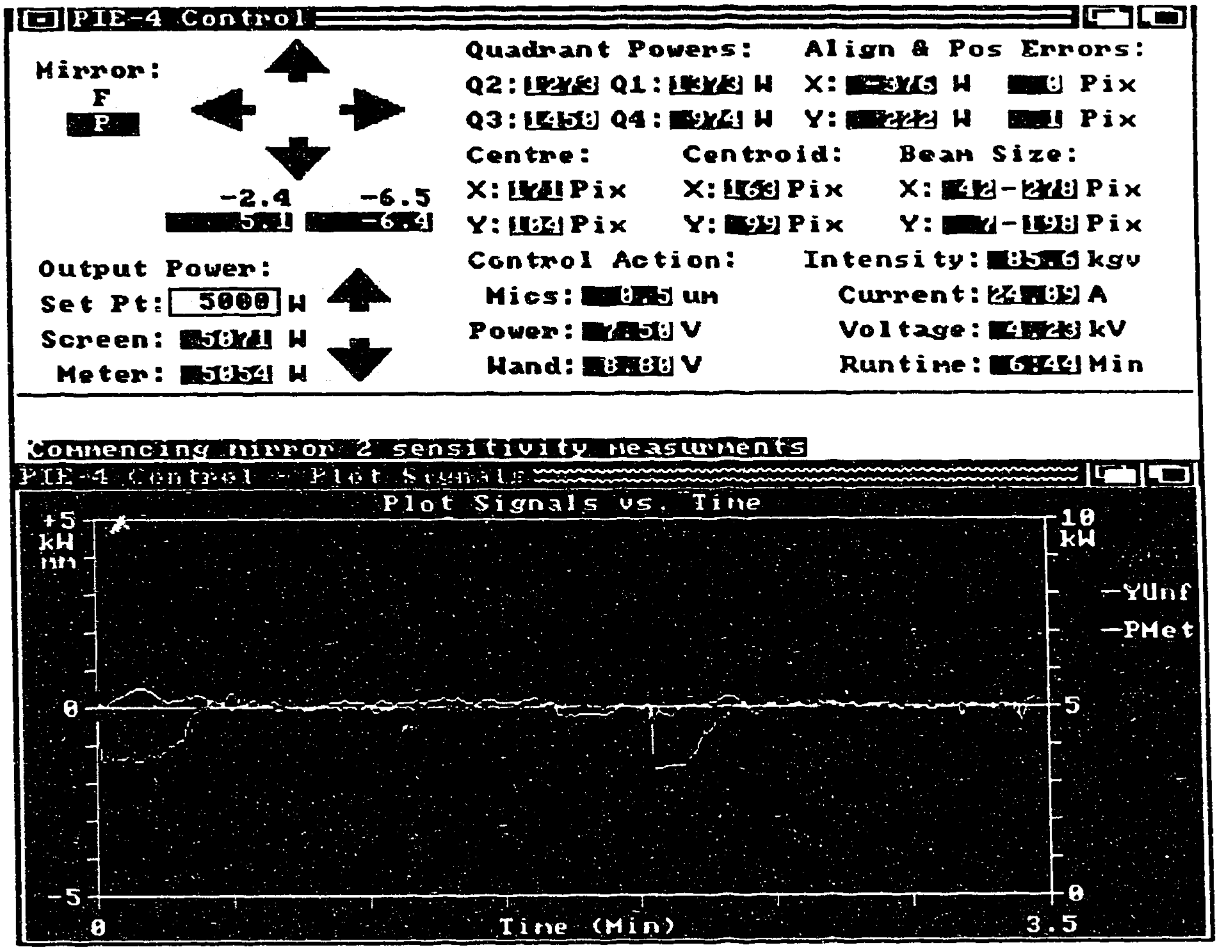
Fig. 3.5. Flowchart of control action generating algorithm.

Following initiation of the optic control actions, the total intensity of the beam is compared to a preset operating range. If the value is not within range, the controller modifies the rotational speed of the variable coupling wand to either increase or decrease the coupling percentage. If the previously executed mirror adjustment is sufficiently small and an error in the total power exists, the computer modifies the input power to the laser to stabilize output power. However, if a significant error in uniformity or position exists, laser excitation is not corrected until a near optimum alignment is re-established. This precaution is taken since an accurate determination of the laser's output power, based on the wand's rotational speed and the overall image intensity, could only be derived when its resonator is near alignment.

3.1.5 Data Presentation and Storage

The controller utilizes an extensive menu-driven, multi-level windowed display and interface, in order to maximize versatility and provide a user-friendly environment. Fig. 3.6 presents an example of the top-level window design. The upper window contains a series of mouse-activated gadgets, which enable manual control of the desired mirror. In addition, the output power setpoint may be modified or the power level may be adjusted manually. The remainder of the window displays any error messages or warnings, pertinent input information, error values, and control actions. This data includes the motormic positions, output power as measured by the imaging screen and laser power meter, quadrant powers, alignment and position errors, beam centre and size, and laser input current and voltage. The lower window graphically displays error signals and power levels for easy visual monitoring of system performance.

All displayed data and other relevant information may be stored in an ASCII data file for off-line analysis. Furthermore, a sequence of images (in raster format)



```
PIE-4 Control
Mirror:        Quadrant Powers:            Align & Pos Errors:
  F            Q2: 1273 Q1: 1373 W         X: -376 W    8 Pix
  P            Q3: 1458 Q4: 974 W          Y: 222 W     1 Pix
               Centre:        Centroid:    Beam Size:
  -2.4  -6.5   X: 171 Pix     X: 163 Pix   X: 42-278 Pix
   5.1  -6.4   Y: 104 Pix     Y: 99 Pix    Y: 7-198 Pix
Output Power:  Control Action:             Intensity: 85.6 kgv
Set Pt: 5000 W   Mics: 0.5 um              Current: 24.89 A
Screen: 5071 W Power: 7.58 V               Voltage: 4.23 kV
 Meter: 5054 W   Wand: 8.88 V              Runtime: 6:44 Min
Commencing mirror 2 sensitivity measurements
```

Fig. 3.6. Top level graphical user interface to laser control and diagnostic program.

can be collected. Both data and image files must initially be written to a RAM drive, to prevent degradation of program execution speed, and may later be transferred to permanent storage media. The total amount of data and image storage is limited by memory availability. Data files require approximately 11 kbytes/min, while each image file necessitates 64 kbytes. The Amiga's 5 Mbytes of main memory (approximately 3.5 Mbytes free for program usage) permits maximum storage of about 5 hours of data or a sequence of 50 images, or a lesser combination of both.

3.2 Execution Times for Controller Algorithm

The execution times for all algorithms is dependent on the speed of the microprocessor and the number of concurrent software programs being executed. Table. 3.1 lists the execution times for various code sections of the control algorithm's main loop (Assembly code is labeled as ASM). The times presented are an average of ten successive iterations of the particular code segment. No other programs were in execution at the time these measurements were obtained. For the 8 MHz processor, the average control loop execution time is approximately

Program Code Segment of Main Control Loop	Execution Time (sec)	
	8 MHz	25 MHz
Frame digitization and binary-encoding	0.104	0.028
Bitplane to raster conversion (ASM)	1.068	0.288
Beam centre computation	0.201	0.078
Quadrant intensity computation (ASM)	0.320	0.015
First moment computation (ASM)	N/A	0.045
Control action generation	0.180	0.082
Data display	0.028	0.013
Storage of image and corresponding data	0.054	0.017
Miscellaneous operations	0.120	0.035
Average loop execution time	2.075	0.601

Table 3.1. Execution times for program code segments of main control loop.

2.1 seconds. This does not include computation of first moments and centroids, or control of the variable coupling wand, which were not implemented prior to upgrading the Amiga PC. The 25 MHz processor executes one complete control cycle in approximately 0.6 seconds. This permits a sampling period of 1 second to be utilized.

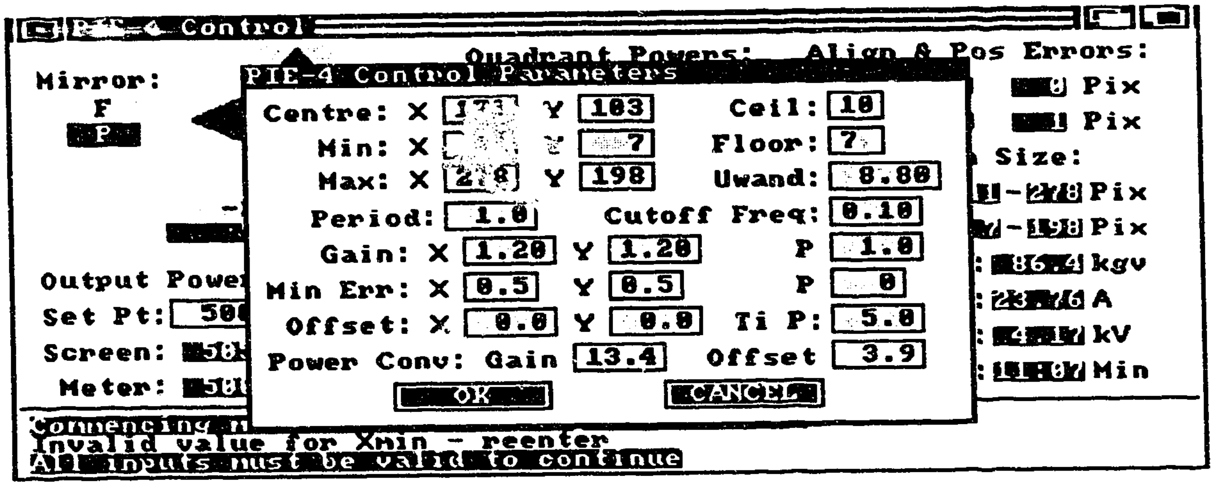
Due to the limited multitasking capabilities of the Amiga 2000, the times listed in Table. 3.1 increase significantly whenever additional software, such as the 3D beam profile program, is executed simultaneously. The exact value of the penalty in speed is dependent on the number of concurrent applications, their complexity, and the processor time slice each occupies. At present, all programs have equal priority. Thus, one additional application increases controller execution times by about 200%. However, these programs share processor time only during their computational phases, generally less than 5 seconds. The residual time is spent in a suspended, interrupt "wait-state", which does not tax the main processor. For instance, if a 3D profile is generated every 100 seconds, the controller software would execute at its normal speed approximately 95% of the time, and at a slower rate for the remainder.

3.3 Operational Features of Software Controller

The software controller possesses a degree of flexibility and decision-making ability unrivaled by comparable hardware-based systems. It incorporates superior error checking and handling capabilities, self-tuning of controller gains, self-calibration of the imaging screen, automated measurement of resonator sensitivities, and on-line modification of system gains, thresholds, and other operational parameters. A detailed description of the controller program's features are presented in the operating manual in Appendix C.

Operation of the system may be in either manual or automatic mode, with each of the separate control functions (uniformity, position, output power, and coupling percentage) enabled independently. Input and error signal filtering, to eliminate excessive noise levels, can be activated if necessary, along with data acquisition, image storage (in overwritten or sequenced mode), and graphical display of user selected signals. In addition, images may be colourized to better illustrate changes in beam intensity, with areas of lower power density appearing as shades of violet and blue, while those of higher concentration are viewed as orange and red. If desired, measures of laser efficiency, and calibration of the intensity-power relationship are obtained by monitoring the input current, input voltage, and output power (from laser power meter) via A/D inputs on the data acquisition unit. All of the above mentioned options may be altered on-line, without interruption of the laser control sequence.

Controller, digitization, and output beam parameters are modified via the lower level input window depicted in Fig. 3.7. The beam centre setpoints and default beam diameters (specified in pixels) may be set to accommodate varying beam positions and sizes. Furthermore, the window permits manual input of the digitization levels (ceiling and floor), wand coupling percentage, controller sampling period, and filter cutoff frequency. Also adjustable are the gains for the horizontal



PIE-4 Control Parameters

Centre: X 173 Y 103 Ceil: 10

Min: X [illegible] Y 7 Floor: 7

Max: X 278 Y 198 Uwand: 0.00

Period: 1.0 Cutoff Freq: 0.10

Gain: X 1.20 Y 1.20 P 1.0

Min Err: X 0.5 Y 0.5 P 0

Offset: X 0.0 Y 0.0 Ti P: 5.0

Power Conv: Gain 13.4 Offset 3.9

```
Commencing n
Invalid value for Xmin - reenter
All inputs must be valid to continue
```

Fig. 3.7. Lower level input window for modification of controller parameters.

uniformity, vertical uniformity, and output power controllers, along with the corresponding minimum error thresholds and offsets. All modified values must fall within preset limits before being accepted.

In addition to performing the essential control functions, the system is capable of automatically acquiring various sensitivity measurements, the data from which may be analysed off-line to quantify system performance and process parameters. The resonator mirror sensitivities are measured by rotating the specified optic about the X alignment axis in both positive and negative directions, storing the data, and repeating the process for the Y axis. From this data, the relationship between micrometer motion and uniformity error is ascertained, along with measures of hysteresis present in the optical mounts. Beam steering sensitivity is attained by rotating the positional control optic about the X and Y axes, while maintaining beam uniformity with the other mirror. A series of beam centre coordinates, at which no uniformity error exists, and the corresponding mirror position are accumulated for future analysis. Output power and coupling percentage relationships to the overall beam image intensity may also be self-generated. This is accomplished by modulation of the laser input power and wand's rotational speed respectively, while simultaneously measuring total intensity at optimum beam uniformity. These relationships may change as the imaging screen cooling levels and equilibrium temperatures deviate from their optimal values. In such cases, if the beam is directed into the laser power meter, the intensity-power conversion references may be automatically recalibrated by comparing actual output power to computed values.

Open and closed loop step responses may be performed and controlled via a supplementary sub-level window. These include step responses for both resonator optics, as well as output power and wand coupling percentage. If desired, the controller is capable of tuning itself by calculating and resetting the uniformity gains. These gains are determined from the initial and final steady-state values of the step responses, and application of an experimentally computed, gain modification factor.

In all modes of program execution, extensive tests are performed for both input and operational errors. If the video signal is disrupted or the total beam intensity decreases below a resolvable limit, all control functions are disabled until the condition is corrected. Warnings are issued whenever error signals are excessively large, and the resulting controller outputs are maintained within safe limits. Since the controller generally operates in an unsupervised mode, an operator not directly viewing the computer screen does not observe any visually displayed system errors and warnings. Thus, all consequential errors are vocally communicated utilizing the Amiga's built-in speech processor.

3.4 Additional Diagnostic Software and Utilities

The software base also includes programs for 2D and 3D beam intensity profile generation, and several utilities for image manipulation. Cross-sectional intensity profiles provide vital information pertaining to beam symmetry and uniformity which is not immediately apparent in the visual image. Fig. 3.8 depicts a typical cross-sectional, gray-scale image of an aligned 5 kW output beam from the PIE-4 laser. The corresponding 3D intensity profile is presented on Fig. 3.9. As evident, the profile graphically accentuates subtle differences in power distribution, and may be viewed from any angle of rotation or elevation. A 2D profile (a single slice of the 3D profile) taken horizontally across the beam image of Fig. 3.8, and through its centre, is illustrated in Fig. 3.10. Such plots, which may be generated along any straight line passing through the beam centre, provide excellent measures of beam symmetry.

Additional utility software also aided in processing and cataloguing of stored image sequences. Images, saved in raster format, may be colourized and converted to compressed bitplane (Amiga IFF) format, which requires about 85% less disk

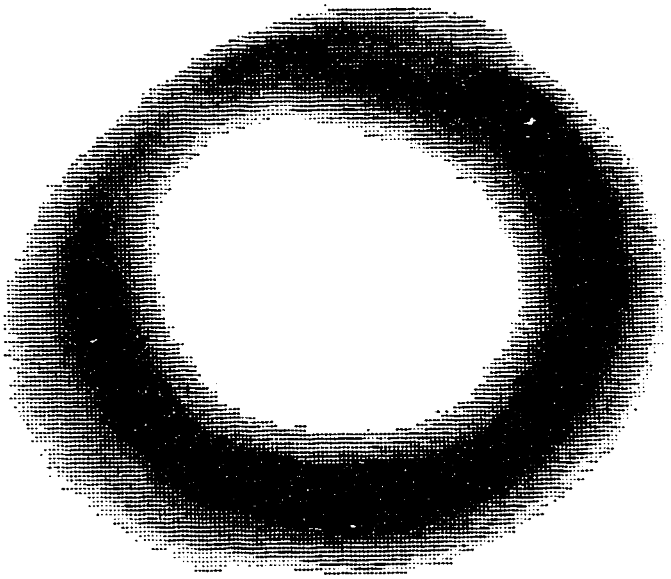


Fig. 3.8. Digitized image of 5 kW PIE-4 output beam cross section.

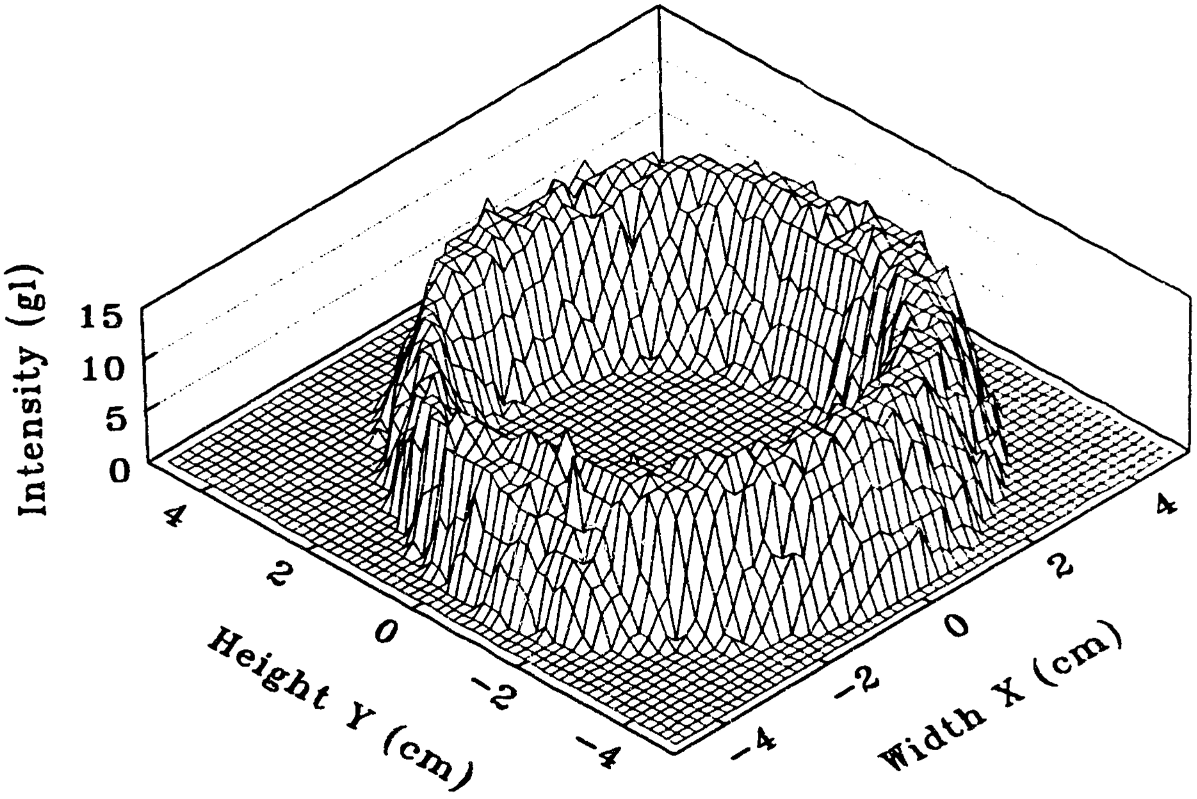


Fig. 3.9. 3D intensity profile of cross-sectional beam image depicted in Fig. 3.8, at viewing angles of rotation and elevation are 45° and 70° respectively.

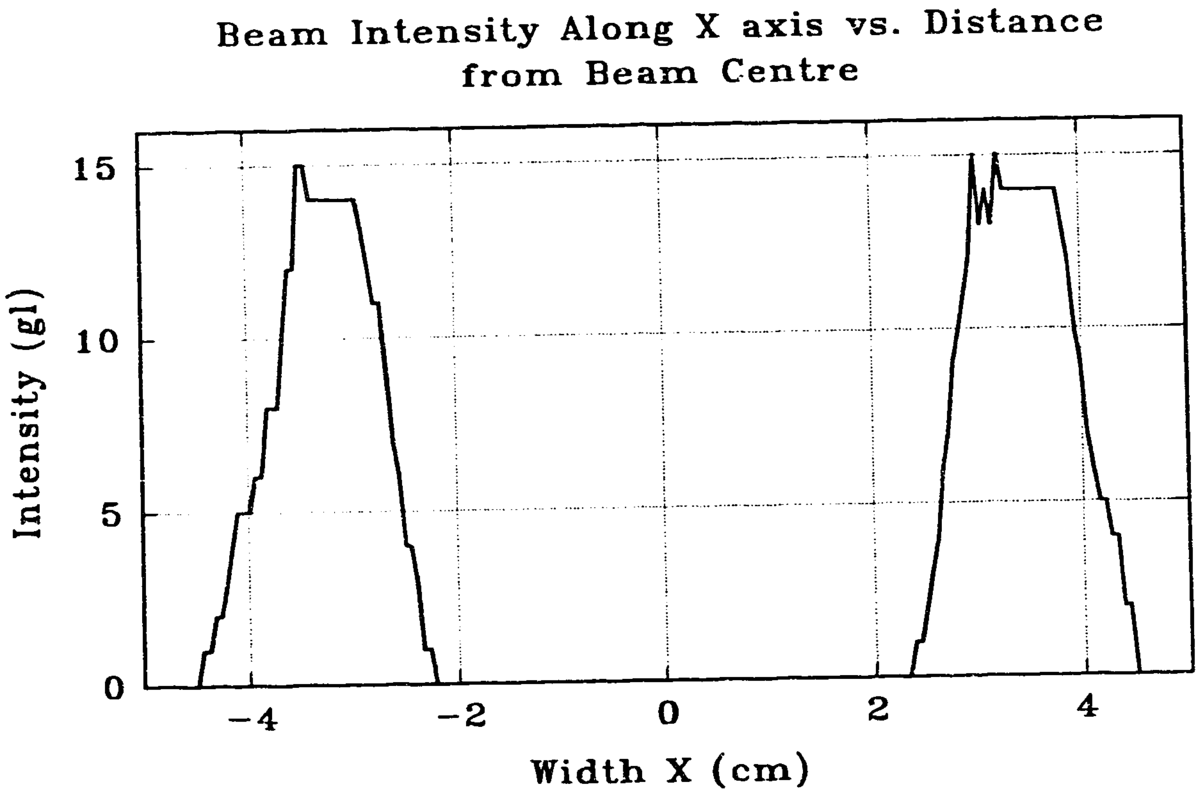


Fig. 3.10. 2D profile of intensity along X axis of beam image shown in Fig. 3.8.

storage space. If necessary, specific sections of an image may be extracted, enhanced, thresholded, and/or complemented. A special image printing routine, for use with laser printers, enables customization of dithering patterns to improve visual quality.

CHAPTER 4

INITIAL SINGLE-OPTIC CONTROL SYSTEM

The performance of the beam uniformity and power stabilization control system was initially evaluated on the PIE-3 CO_2 laser. At this time, hardware and software systems were not completely developed, thus, the results presented do not fully describe the potential capabilities of the control system. Open loop tests were conducted first to determine optimal controller gains, following which, feedback control of the convex resonator optic and input power regulation variac were implemented. The performance of the system was evaluated with a series of transient and steady-state responses. Comparison of these results to data obtained without controller operation revealed a significant improvement in performance. The stability of the control system was analysed experimentally using the continuous cycling method, which indicated a high degree of stability.

4.1 Measurement and Control Apparatus

Preliminary control and monitoring systems did not entail all hardware devices implemented in the final design. Firstly, the central microcomputer operated at a clock speed of only 8 MHz. As such the computationally intensive program code needed to be limited in order to permit a sufficiently fast sampling interval. In addition, the enhancement to the data acquisition unit (the supplementary board described in Appendix B), with its 16-bit counter, was not yet developed. This aspect necessitated micrometer positions to be maintained via software polling of the driver feedback signals. Due to the limited power range of the PIE-3 laser, a constant coupling rotational wand was utilized to sample 1% of the output.

A complete block diagram of the experimental setup and interconnecting signals is illustrated in Fig. 4.1. Laser output power was measured using another beam sample, focused onto a Coherent Model 213 water-cooled power meter.[68] This 5% sample was derived from a second rotating wand positioned after the 1% device, in order to prevent interference between the two wands, thereby improving beam imaging quality. Laser power was controlled by two mutually exclusive digital outputs (PWRUP and PWRDN) which enabled two corresponding relays, and caused the DC power supply variac to rotate in either a clockwise or counter-clockwise direction. The resulting change in input current was monitored with a hall-probe current sensor. Beam uniformity was adjusted via the digital outputs, FWD and

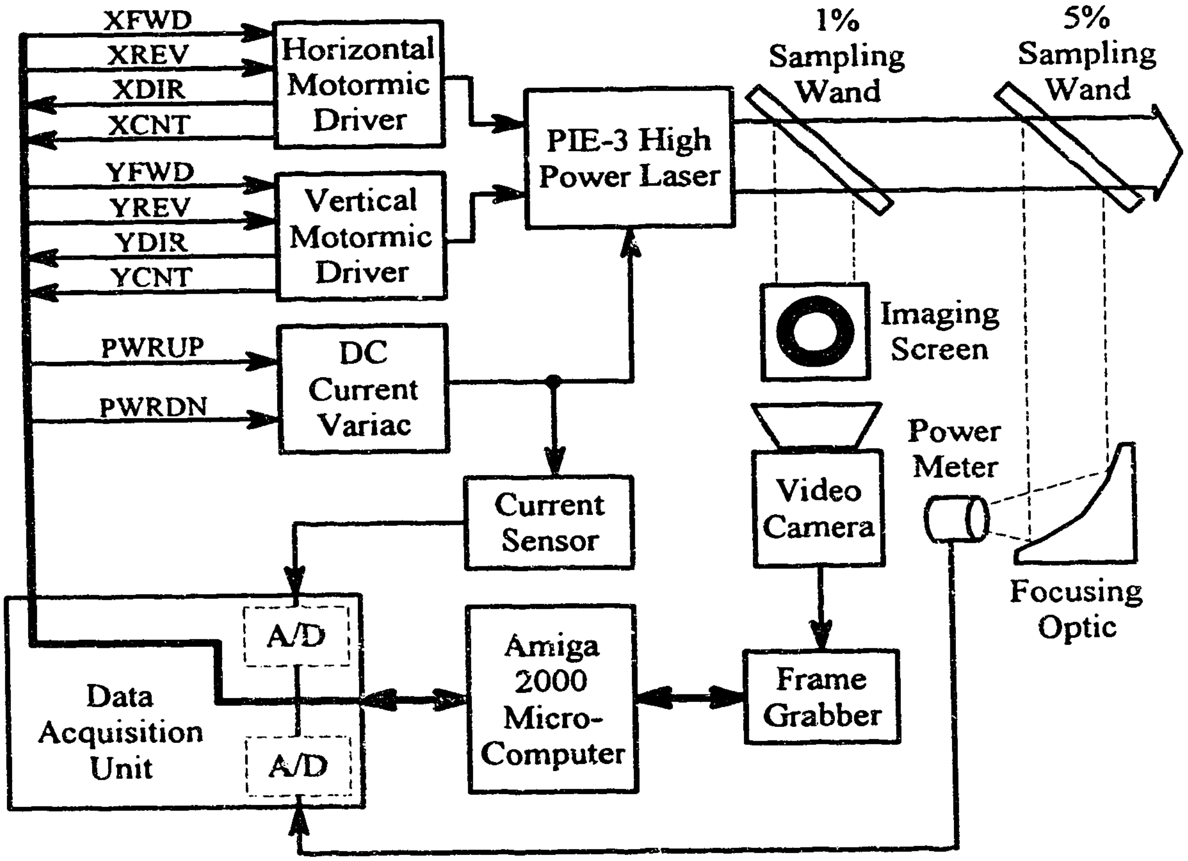


Fig. 4.1. Schematic of experimental setup and interconnecting signals.

REV, which affected the direction of motion of each micrometer, and subsequently, the angular alignment of the convex mirror. The two inputs, DIR and CNT, enabled tracking of micrometer positions.

4.2 Closed Loop System Model

The general practice to characterize multi-input, multi-output (MIMO) systems and implement an effective multivariable control strategy, is to subdivide the system into interdependent sub-systems, and then eliminate the interdependencies. This yields several independent single-input, single-output (SISO) systems which can be controlled individually.[69] The three feedback loops for beam uniformity and power stabilization are presented in Fig. 4.2. The sampling interval, T_s, varies depending on the control actions performed.

The beam uniformity feedback system consists of the two main horizontal and vertical alignment loops. For each of these sub-sections, the motormics are modelled as integrators, with gain parameters K_{xs} and K_{ys} (in µm/s) corresponding to the motor speed setting for the horizontal and vertical micrometers respectively. These parameters are externally set between 10 and 20 µm/s to permit both manual and automatic alignment with a sufficient degree of sensitivity. Since control actions are based on the amount of mirror motion and not speed of motion, modifying these parameters does not require re-tuning of controller gains. Increasing these parameters simply results in control movements being executed more rapidly, hence producing faster closed loop responses. The laser and infrared detection screen are modelled as a single, first-order block. Ideally, the laser may be represented as a separate gain factor, without a time constant or delay, since it responds instantaneously to changes in alignment (shown in Fig. 4.3). However, uniformity error could not be measured directly in kilowatts. Thus, the two blocks were combined, along with the corresponding gains, K_{ul} (in kW/µm) and K_{ud} (in gv/W), to

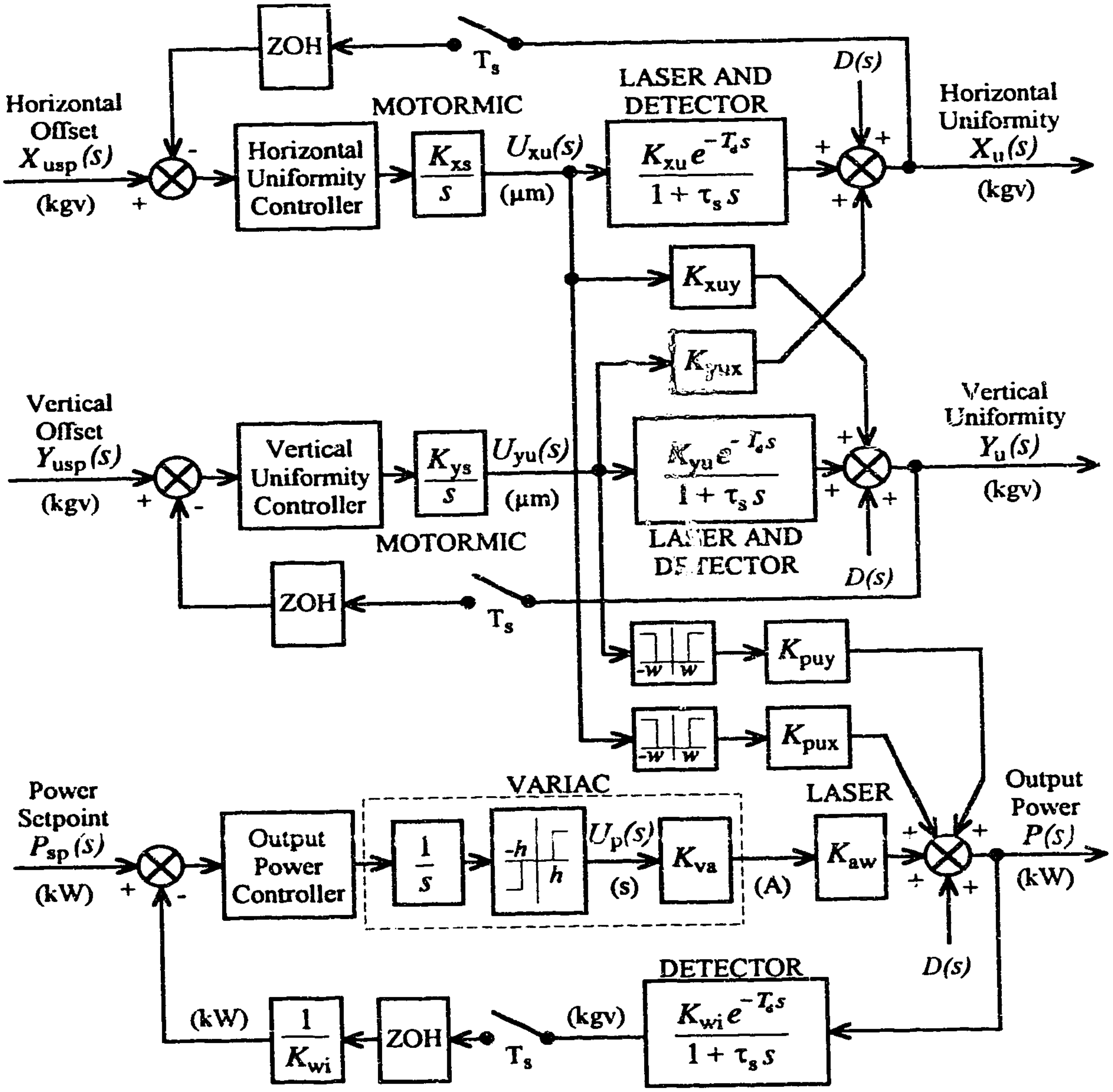


Fig. 4.2. Beam uniformity and output power feedback control system.

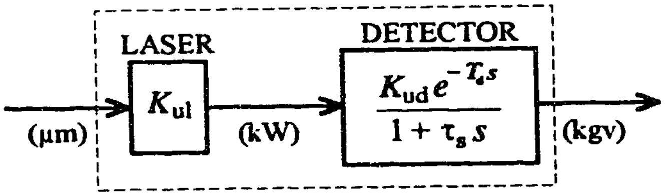


Fig. 4.3. Ideal representation of laser and detector block in Fig. 4.2.

yield the horizontal and vertical uniformity gains, K_{xu} and K_{yu} (in kgv/μm). Since the motormics were driving differential micrometers, they did not exhibit any significant backlash. Consequently, corresponding hysteresis blocks were not included in the model. The two interconnecting blocks, K_{xuy} and K_{yux} (in kgv/μm), describe the effect of the particular micrometer's motion on the uniformity error in the complementary direction. For a properly designed, two-axis optical mount, these interactions should be negligible.

The output power feedback loop consists of the DC current supply variac, laser, and infrared imaging screen. The variac may be modelled as three distinct units - the motor which physically rotates the variac at a constant speed (represented as an integrator), the backlash between the motor and variac shafts due to the gear coupling (the hysteresis block), and a gain factor, K_{va} (in A/s), characterizing the variation in input current when the variac motor is enabled for a given period of time. Output of the laser discharge block, assigned a single gain factor, K_{aw} (in kW/A), is sampled onto the imaging screen detector, thus producing an intensity output in gray levels (kgv). This output is then divided by the detector's power to intensity conversion factor, K_{wi} (in gv/W), to obtain a measure of output power in kilowatts, which may then be compared to the power setpoint. As evident from Fig. 4.2, the output power is also dependent on beam uniformity. Specifically, if resonator alignment drifts outside an error window (specified by $\pm w$), the output power decreases correspondingly. Although this reduction may not be linear, it is modelled as two simple gain blocks, K_{pux} and K_{puy} (in kW/μm), for simplicity.

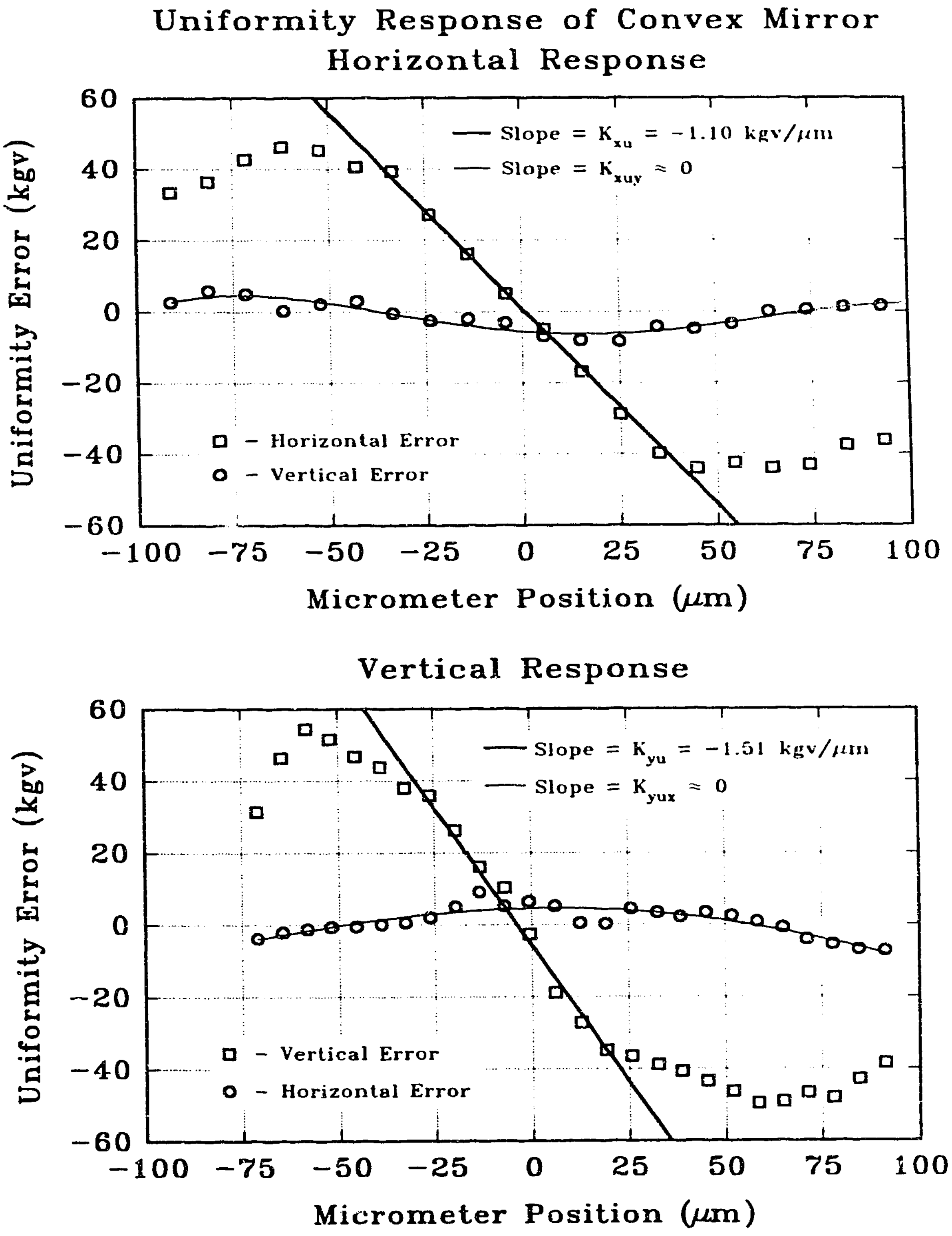
4.3 Model Parameter Estimation

Gain parameters in the system model were obtained by varying the particular inputs and measuring the corresponding steady-state outputs. Fortunately, the imaging screen detector possessed a sufficiently long time constant, τ_s , to adequately filter this data and eliminate excessive noise. However, due to hardware limitations and the relatively large sampling interval required, acquiring open loop step responses to accurately model the detector's time constant and delay was not possible. As a result, the imaging screen's time constant and delay were estimated to be approximately 1.0 second and 0.4 seconds respectively. These values were based on both manufacturer's specifications and visual observations. The error in these values was not critical to system performance, due to the type of controller implemented.

4.3.1 Uniformity Process Parameters

The first step was to determine the process gains relating micrometer motion to uniformity error. This was accomplished by fixing the position of the concave mirror and then varying the alignment of the convex unit along the axis being analysed. The exact procedure included moving the convex mirror in the negative X direction until the uniformity error became saturated, and then stepping in the positive X direction (in steps of about 10 μm) until the opposite saturation condition was reached. The resulting sensitivity data is presented in the first graph of Fig. 4.4. The second graph of Fig. 4.4 illustrates the vertical sensitivity obtained by repeating the same procedure for the Y direction. A similar set of measurements acquired for the concave mirror is depicted in Fig. 4.5.

As evident from these graphs, the curves may be approximated by a straight line for an operation window of about ±25 μm. Also, the slopes representing the complementary gains, K_{xuy} and K_{yux}, are approximately zero, thereby enabling each axis of uniformity error to be controlled independently. For both optics, the horizontal uniformity gain, K_{xu}, is less than the vertical, K_{yu}, possibly due to gain variations in the laser discharge. The magnitude of the process gains, K_{xu} and K_{yu}, for both alignment axes are greater for the concave mirror. This feature indicates that the convex mirror may be regulated with a greater degree of sensitivity, thus


Uniformity Response of Convex Mirror

Horizontal Response

Vertical Response

Fig. 4.4. Sensitivity of uniformity error to changes in convex mirror alignment.

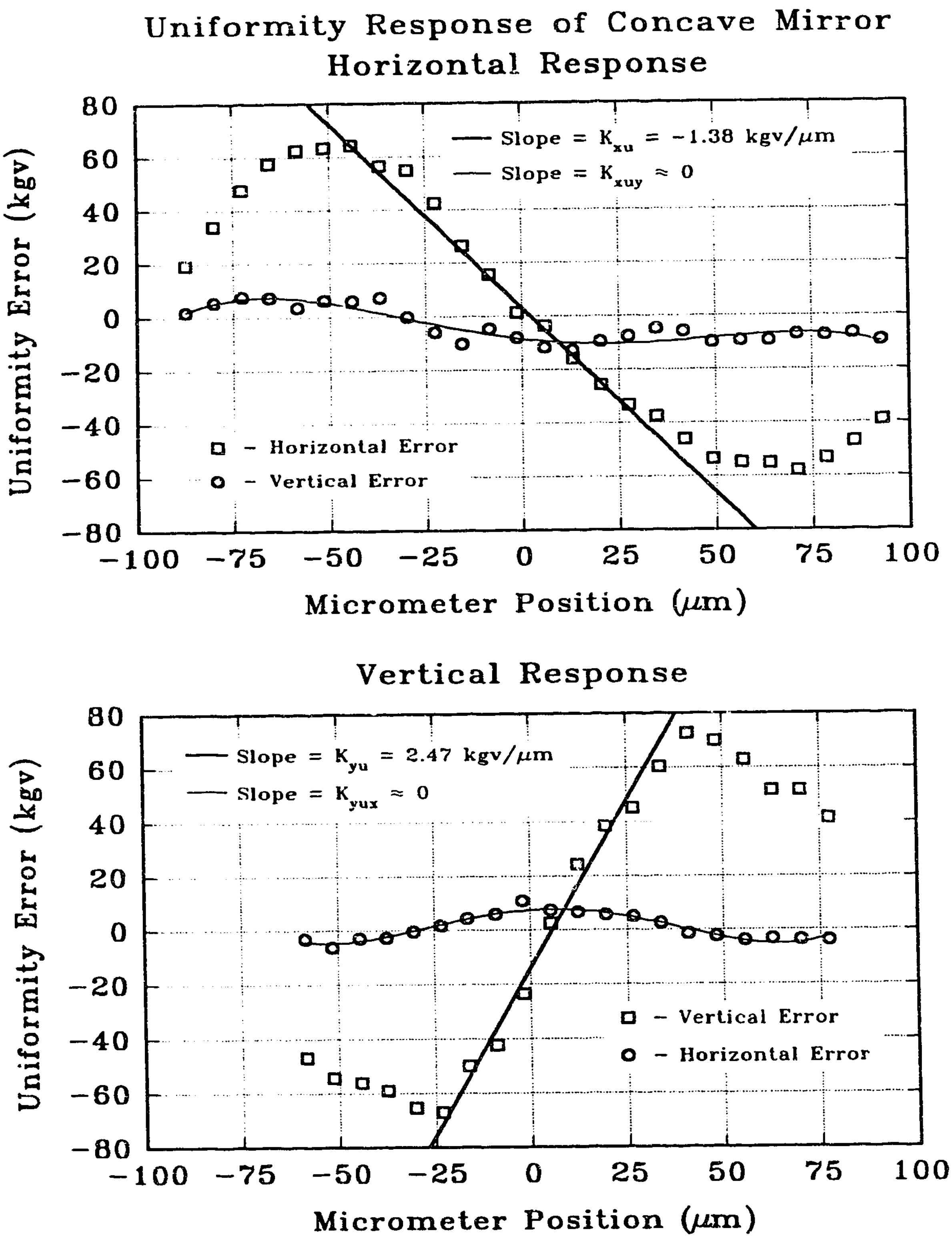


Fig. 4.5. Sensitivity of uniformity error to changes in concave mirror alignment.

supporting the original design decision to control this optic.

4.3.2 Output Power Process Parameters

The output power gain parameters were obtained by varying the DC discharge current and recording the corresponding power levels (measured with the power meter) and screen intensity values. A measure of the variac hysteresis window, $2h$, was determined to be 0.5 seconds by analysing data from positive and negative step responses. Furthermore, tests revealed that the length of the controller output pulse, which activates the variac, must be at least 0.18 seconds in order to initiate a change in current. Hence, the input current could be controlled with a minimum resolution of approximately 0.25 A. Fig. 4.6 depicts the dependence between DC current and variac activation time (i.e. the variac gain, K_{va}). Data points represent measurements taken following 0.18 second pulses to the variac control relays. The laser discharge gain, K_{aw}, was determined to be 0.36 kW/A from the graph of Fig. 4.7, while the

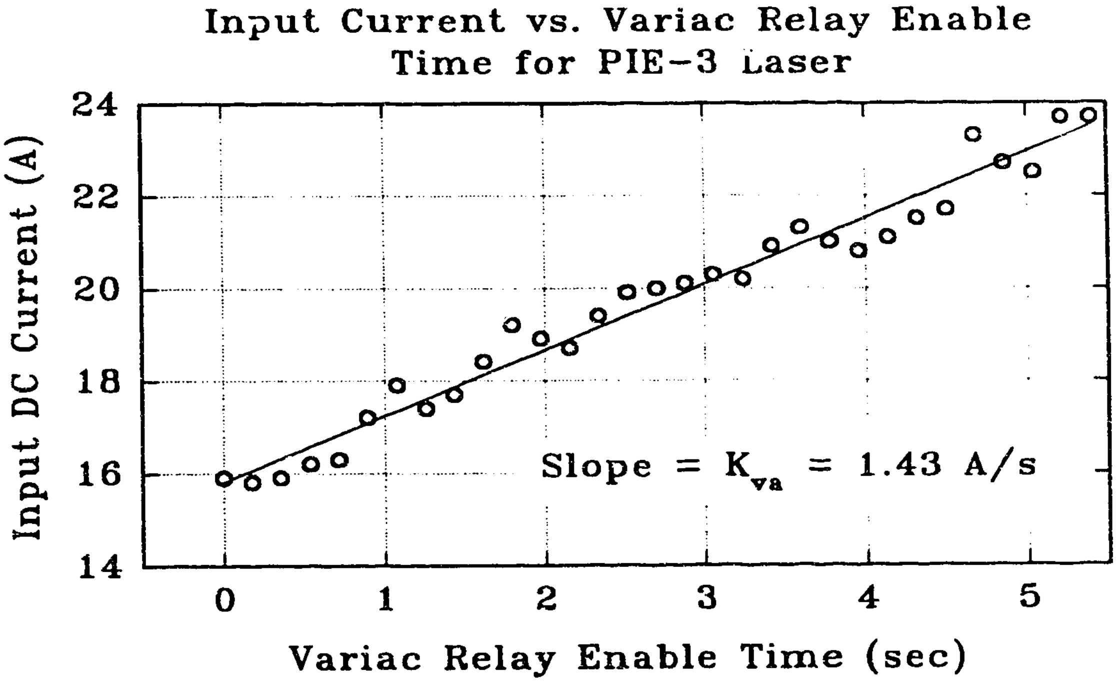


Fig. 4.6. Dependence of DC current on variac activation time.

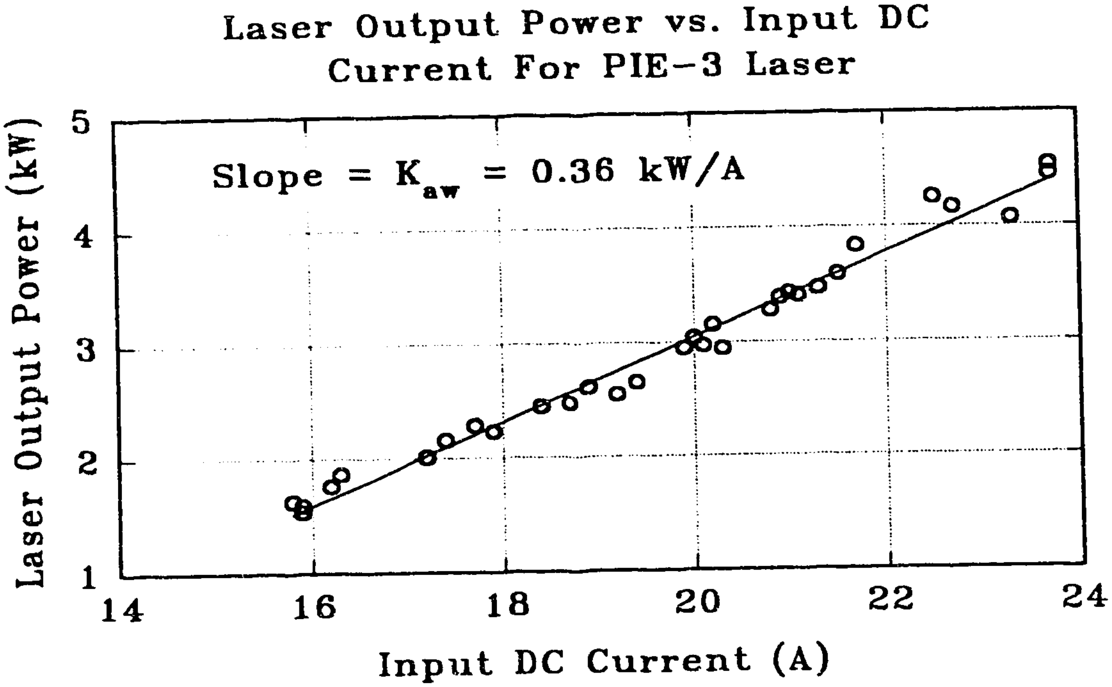


Fig. 4.7. Relationship between output power and DC input current.

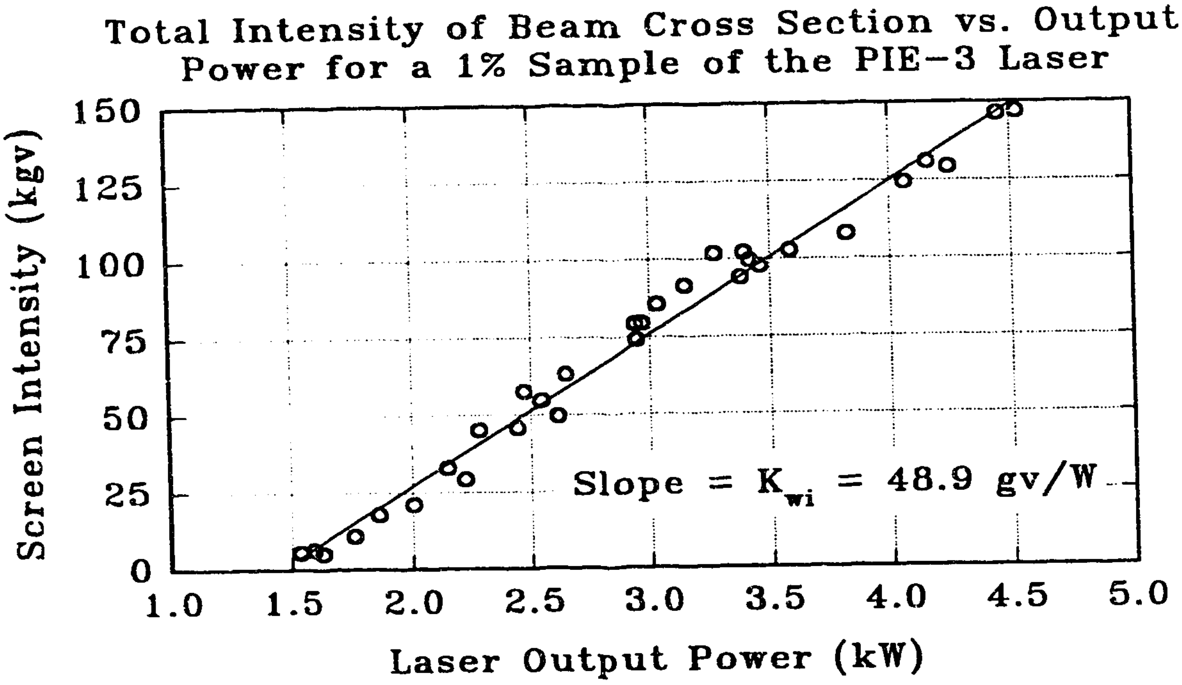


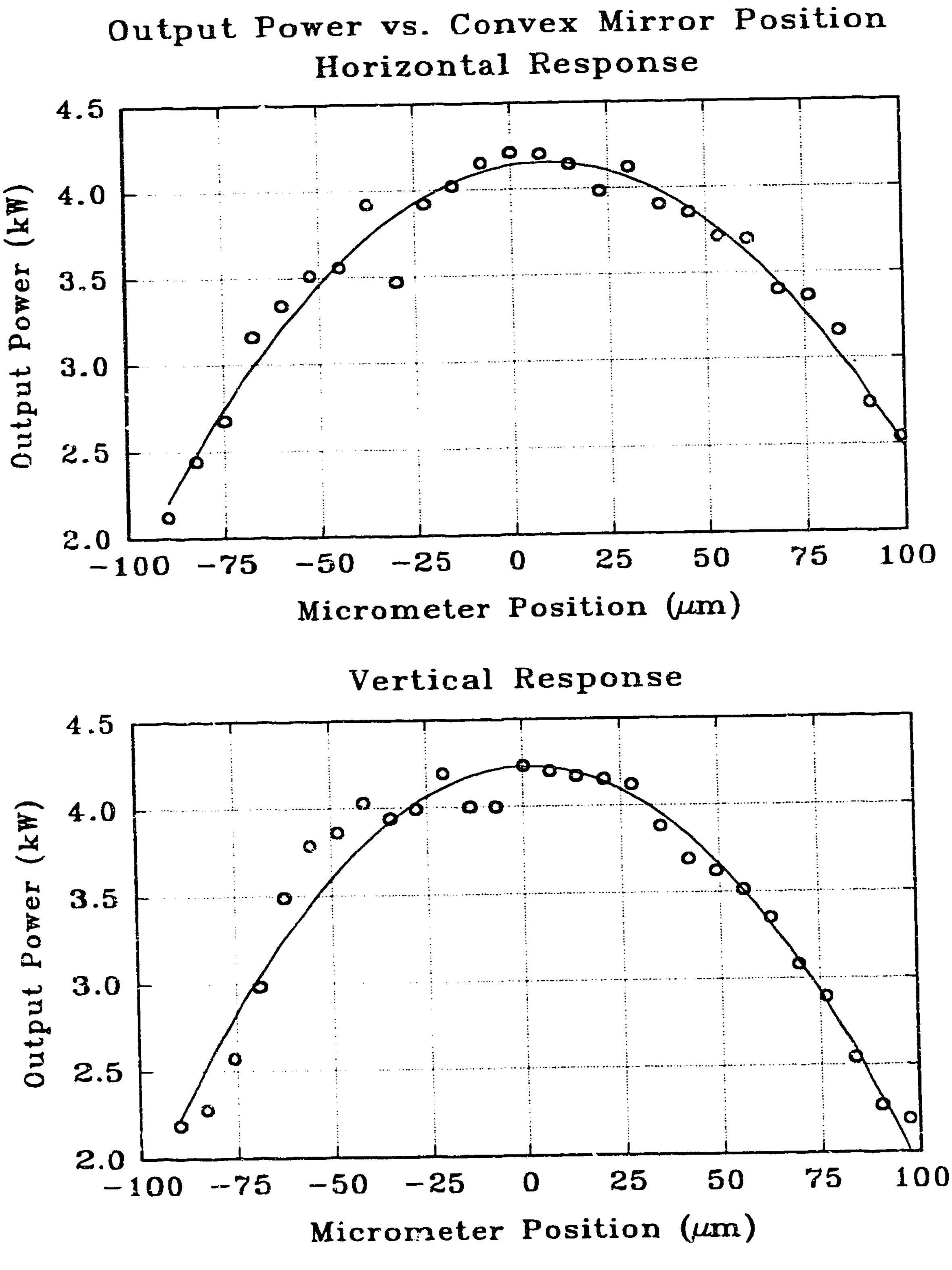
Fig. 4.8. Power to intensity conversion gain for imaging screen.

detector gain, K_{wi}, was obtained from the intensity-power relation shown in Fig. 4.8

In an attempt to model the interdependency between beam uniformity and output power (blocks K_{pux} and K_{puy}), the data in Fig. 4.9 was acquired. The first graph illustrates the decrease in output power as the horizontal micrometer is moved in positive and negative directions from its optimally aligned position. The second graphs presents the similar response for the vertical micrometer. As is evident, the output power is approximately constant within ±15 μm of alignment. Thus, if the power controller is disabled when uniformity control actions greater than 15 μm are

Model Parameter	Value
K_{xs} and K_{ys} for convex and concave optics	10 to 20 μm/s
K_{xu} for convex optic	-1.10 kgv/μm
K_{yu} for convex optic	-1.51 kgv/μm
K_{xu} for concave optic	-1.38 kgv/μm
K_{yu} for concave optic	2.47 kgv/μm
K_{xuy} and K_{yux} for convex and concave optic	~0 kgv/μm
K_{va} for variac	1.43 A/s
K_{aw} for laser	0.36 kW/A
K_{wi} for imaging screen detector	48.9 gv/W
K_{pux} and K_{puy} when $-w < U_{xu}(s)$ and $U_{yu}(s) < +w$	~0 kW/μm
τ_s for imaging screen detector	~1.0 s
T_d for imaging screen	~0.4 s
T_s for all feedback loops	1.8 to 3.6 s
h (hysteresis) for variac	0.25 s
w for power controller deactivation	15 μm

Table 4.1. Estimated parameters for uniformity and power stabilization model.


Output Power vs. Convex Mirror Position

Horizontal Response

Vertical Response

Fig. 4.9. Sensitivity of output power to changes in convex mirror alignment.

required, the interconnecting blocks may be eliminated, thereby yielding three independently controllable processes. Table 4.1 lists est [illegible] alues for all parameters described in the system model of Fig. 4.2.

4.4 Controller Description and Design

Due to hardware limitations and the non-linearity of the system, many conventional control techniques could not be employed in this initial controller design. Since the motormics operate at a manually set constant speed, the use of standard linear controllers is not feasible. Consequently, several non-linear control strategies were investigated.[70]

The most simple and intuitive approach was to adopt a relay-type control scheme. Such a method would continuously sample the output, initiate micrometer motion upon existence of an error condition, and halt movement once error signals are reduced to acceptable limits. However, the minimum possible sampling interval, due to the time consuming digitization and error computation operations, was approximately two seconds. Thus, motormic speeds must be extremely slow to function under this condition, yielding inadequate and inefficient controller performance. Furthermore, micrometer positions could not be measured while computing error signals, since simultaneous monitoring of the driver feedback signals was not possible. As a consequence of these limitations, a form of optimal control was applied, which attempted to restore beam uniformity and output power in a single sampling interval.

A schematic diagram of the beam uniformity and power controllers is shown in Fig. 4.10. The first step in the control sequence is to convert the horizontal and vertical uniformity errors to microns by applying the respective controller gains, K_{cxu} and K_{cyu} (in µm/kgv). If either of these two control actions falls outside the threshold window ($\pm w$), the controller associated with the larger error is enabled.

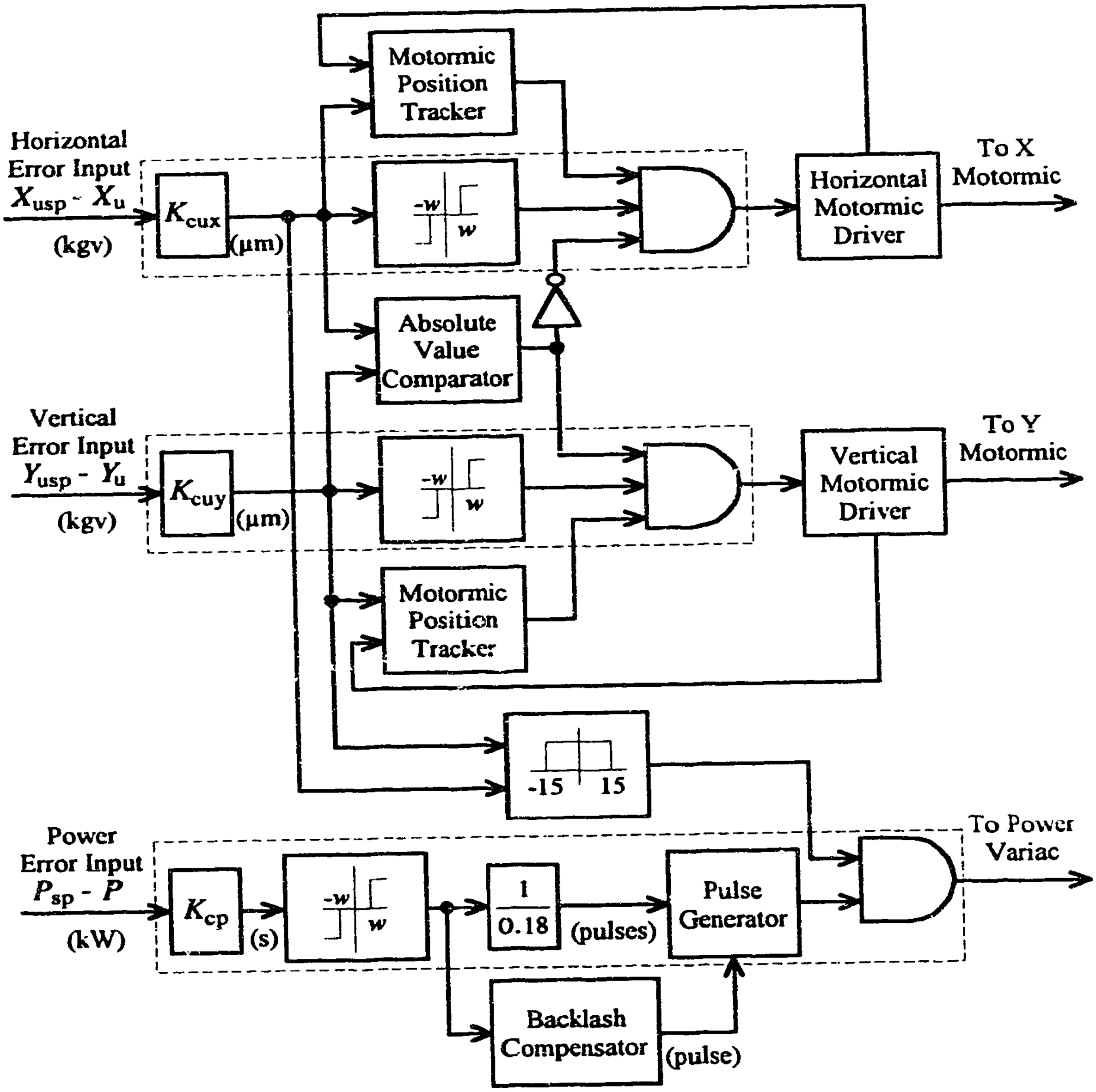


Fig. 4.10. Schematic of uniformity and output power controller.

The corresponding motormic is subsequently activated, and the controller monitors the distance traveled via the driver feedback signals. Upon movement of the required distance, the motormic is halted, a delay of 0.2 seconds is employed for motion to cease completely, and the final position is recorded. At this point, the output power controller is enabled if the uniformity control action was within the error window of ±15 μm.

The output of the power controller is a series of 0.18 second pulses, with a 50% duty cycle. The number of pulses is proportional to the power error, and determined by the power controller gain, K_{cp} (in s/kW). This pulsed control output permitted the variac to be regulated with greater consistency and accuracy (similar to a stepper motor) than utilization of a single pulse with varying length. Following execution of both uniformity and power control actions, a 1 second delay was implemented before obtaining the next sample. This delay allowed the imaging screen to respond and accurately reflect the effect of the control sequence.

Assuming that the system is operating within the linear region of the previously described sensitivity curves, the ideal controller gains to compensate for all errors in a single control action may be computed as follows:

$$K_{cxu} = \frac{1}{K_{xu}} = \frac{1}{-1.10 \text{ kgv / } \mu\text{m}} = -0.91 \ \mu\text{m / kgv} \tag{4.1}$$

$$K_{cyu} = \frac{1}{K_{yu}} = \frac{1}{-1.51 \text{ kgv / } \mu\text{m}} = -0.66 \ \mu\text{m / kgv} \tag{4.2}$$

$$K_{cp} = \frac{1}{K_{va}K_{aw}} = \frac{1}{(1.43 \text{ A / s})(0.36 \text{ kW / A})} = 1.94 \text{ s / kW} \tag{4.3}$$

The system was tested with the above gains and found to perform as expected. However, uniformity errors usually required two control steps to correct, instead of the predicted single step. The cause was attributed to the slight reverse motion of the motormics when they were stopped (i.e. a 10 μm movement actually traveled only about 7.5 μm), due to the axial load. Consequently, the uniformity gains, K_{cxu} and K_{cyu}, were increased by about 30% to -1.2 and -0.9 μm/kgv respectively. When operating outside the linear region, the controller always required more than one sample period (usually three) to restore optimal uniformity for a given alignment axis.

Closed loop response of the output power controller featured significant overshoot for step changes greater than 0.5 kW. This characteristic was a direct result of the imaging screen's time constant. The detector could not respond to large power changes within the 1 second delay allotment (before obtaining the next sample). Thus, the power controller output was limited to 0.9 seconds (5 pulses of 0.18 seconds each) per sample period, producing a 0.45 kW power change. The backlash compensator stores the previous rotational direction of the variac. If the particular power error necessitates variac motion in the opposite direction, a 0.5 second pulse is added to the beginning of the controller pulse train in order to overcome this hysteresis.

4.5 Control System Performance

Performance of the control system was evaluated by examining a series of closed loop responses, and monitoring long-term laser operation with the controller activated and deactivated. Controller gain settings K_{cxu}, K_{cyu}, and K_{cp} [illegible] the following responses were 1.2 μm/kgv, 0.9 μm/kgv, and 1.9 s/A respectively, while the minimum uniformity error thresholds were ±4.0 μm for both alignment axes.

4.5.1 Transient Responses

Sample transient responses for the uniformity controller are illustrated in Fig. 4.11. For the first graph, a horizontal error of approximately 12.5 kgv was introduced with the controller disabled. The control system was then activated at 9 seconds. Immediately, the controller induced a 11.2 μm movement, which eliminated the error within the 3 second sampling interval. The average (filtered) error signal is shown for clarity and was not used for any control purpose. The second graph depicts a similar response for the vertical controller, which was activated at 11 seconds following the introduction of a 14.2 kgv error. As expected,

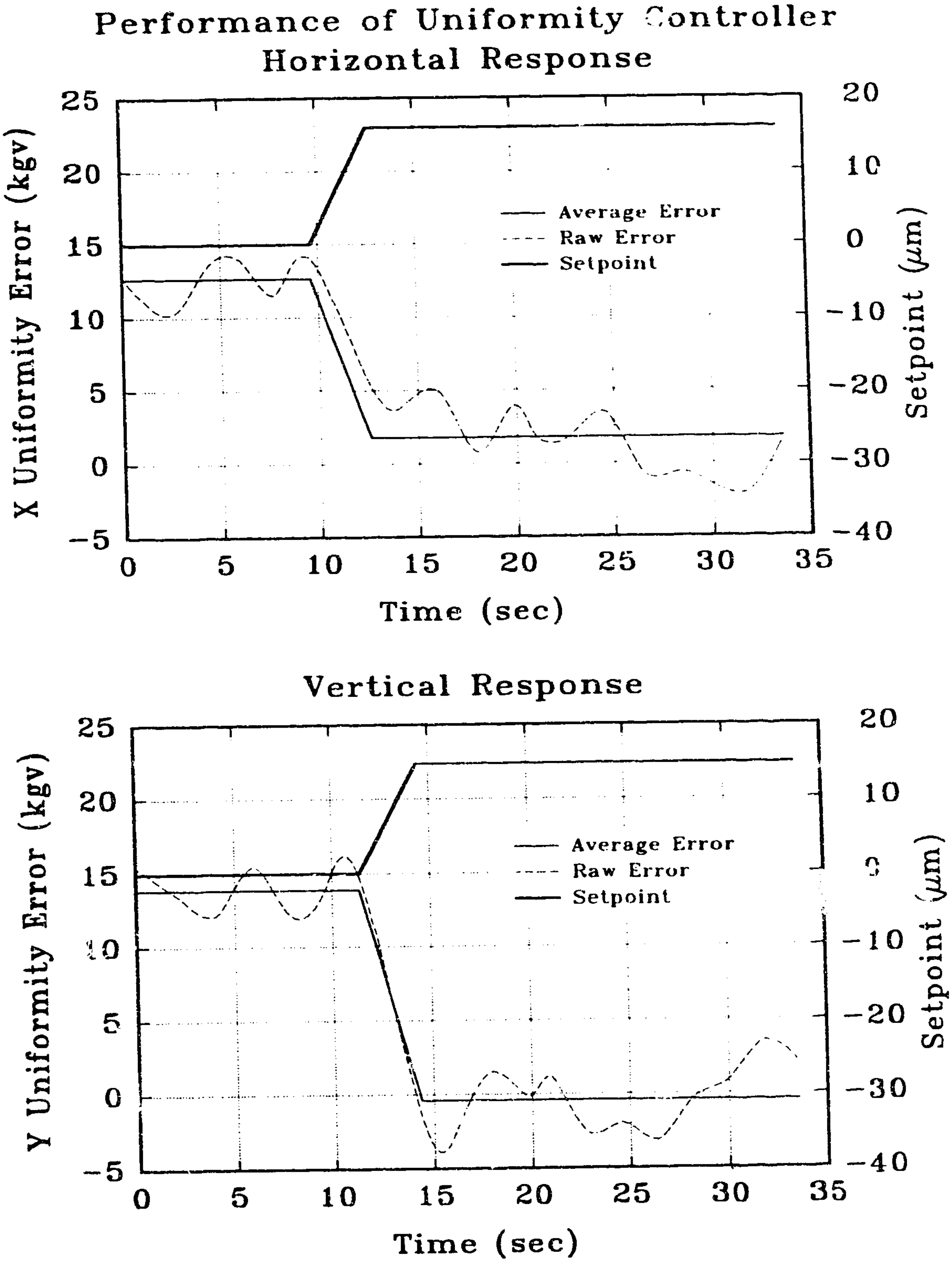


Fig. 4.11. Transient response of uniformity controller.

a single 10.8 µm control action reduced the error to within preset limits in 3 seconds.

The transient output power response is presented in Fig. 4.12. The laser power level was initially set to 2.5 kW, following which the setpoint was changed to 4 kW (at 0 seconds). The controller subsequently increased the input current until the output power reached the setpoint, in approximately 12 seconds. This level was maintained for the remaining duration with minor adjustments to the input current.

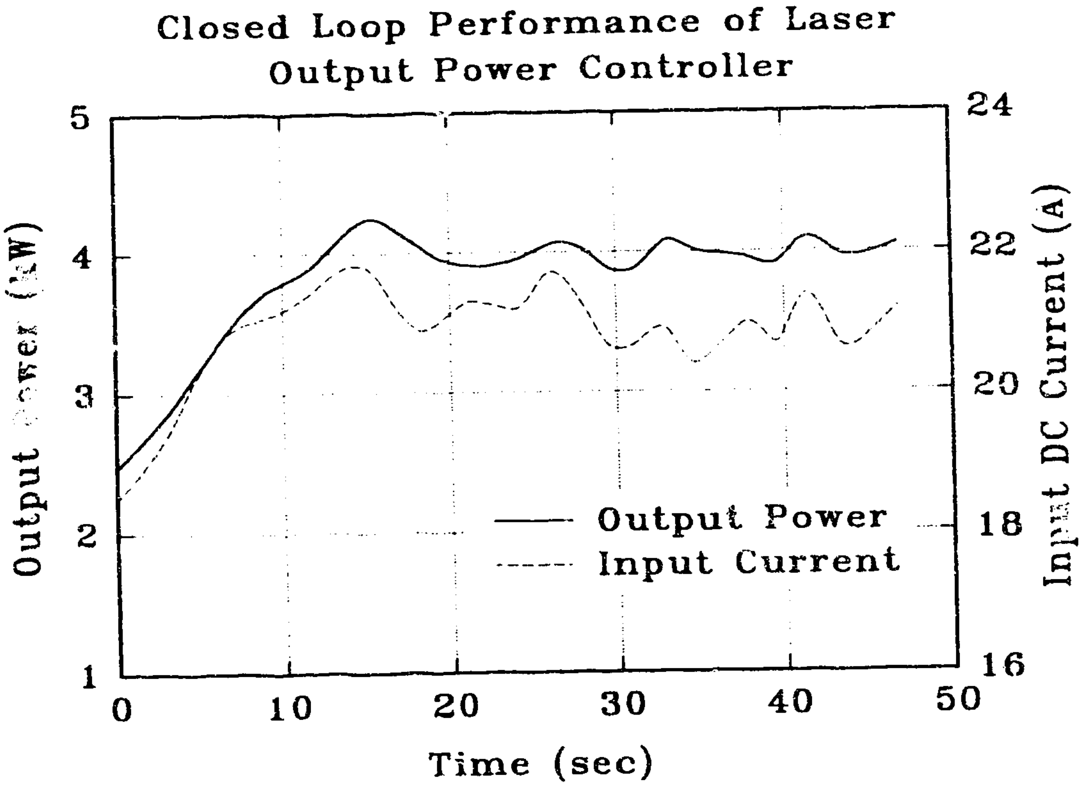


Fig. 4.12. Transient response of output power controller.

4.5.2 Steady-State Responses

The steady-state response of the system and overall improvement in laser operation were investigated by observing long-term performance, both with and without feedback control. It was initially observed that resonator alignment and output power degraded rapidly with time. This aspect is evident from the graph of Fig. 4.13, which reveals that laser performance (measured by the relative and total output powers of each quadrant) decreased continually in the absence of any measure of control or operator intervention during the first 9 minutes. However, when the alignment and power control system was activated at 9 minutes, the power distribution and total power of the beam recovered to their original values in approximately 45 seconds. In the graph of Fig. 4.14, only the uniformity control portion of the system was enabled. Thus, the relative powers in each quadrant were kept constant, although the absolute power of each decreased with time due to an overall degradation of laser output. This significant reduction in power was caused by accelerated lasing gas contamination from a number of small water leaks and inadequate discharge cooling. When power control was activated after a 10 minute interval, the system again recovered to its initial status in about 45 seconds. The relatively long recovery times may be attributed to large uniformity and power errors which are not within the linear operating range of the system.

The graph of Fig. 4.15 reveals operation with both alignment and power control features activated. As is evident, both the output power of each quadrant and their relative intensities were now stabilized to within 5%. Ideally, the power in each quadrant should be identical for a perfectly symmetric beam. However, due to minor distortion and degradation of the aging resonator optics, the data in Fig. 4.15 reflects the best possible beam obtainable from this laser.

Time response of the power control system can be perceived from the graph of Fig. 4.16. When operated under uniformity and power control, the total laser output power was maintained within 4% of the specified value (4 kW). This 4% stability factor is a direct result of the minimum effective step size of the input current variac and the speed of the variac motor. When the power control algorithm was deactivated, total power decreased by approximately 40% in the first 10 minutes.

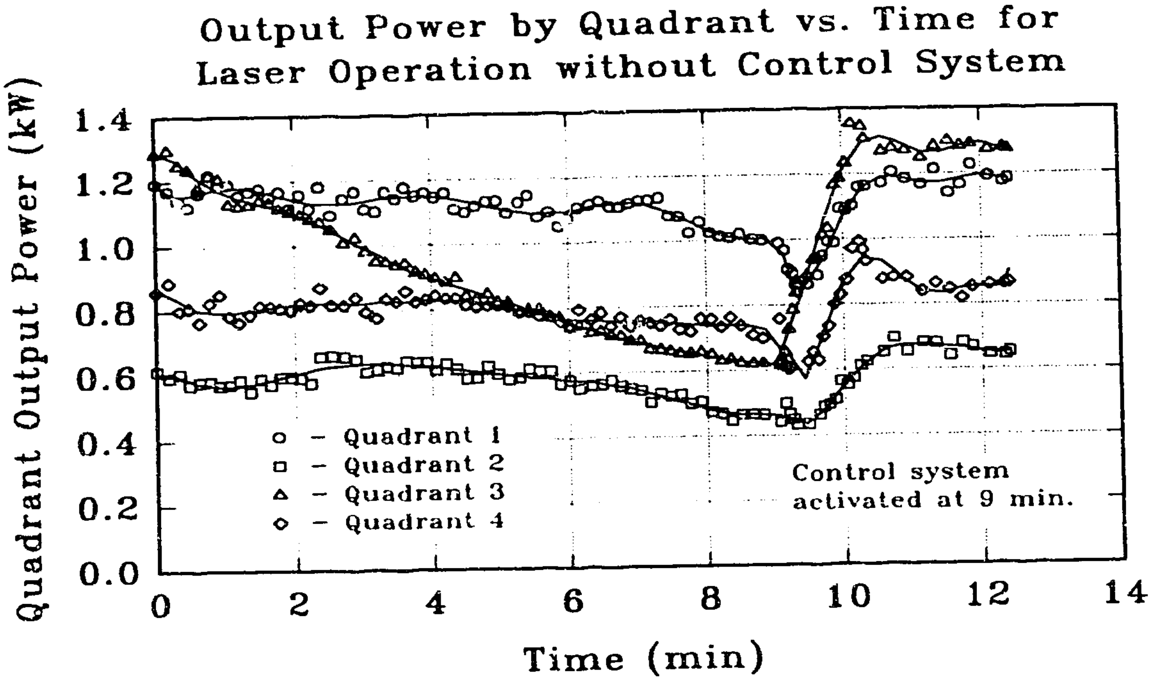


Fig. 4.13. Performance of system with uniformity and power control deactivated.

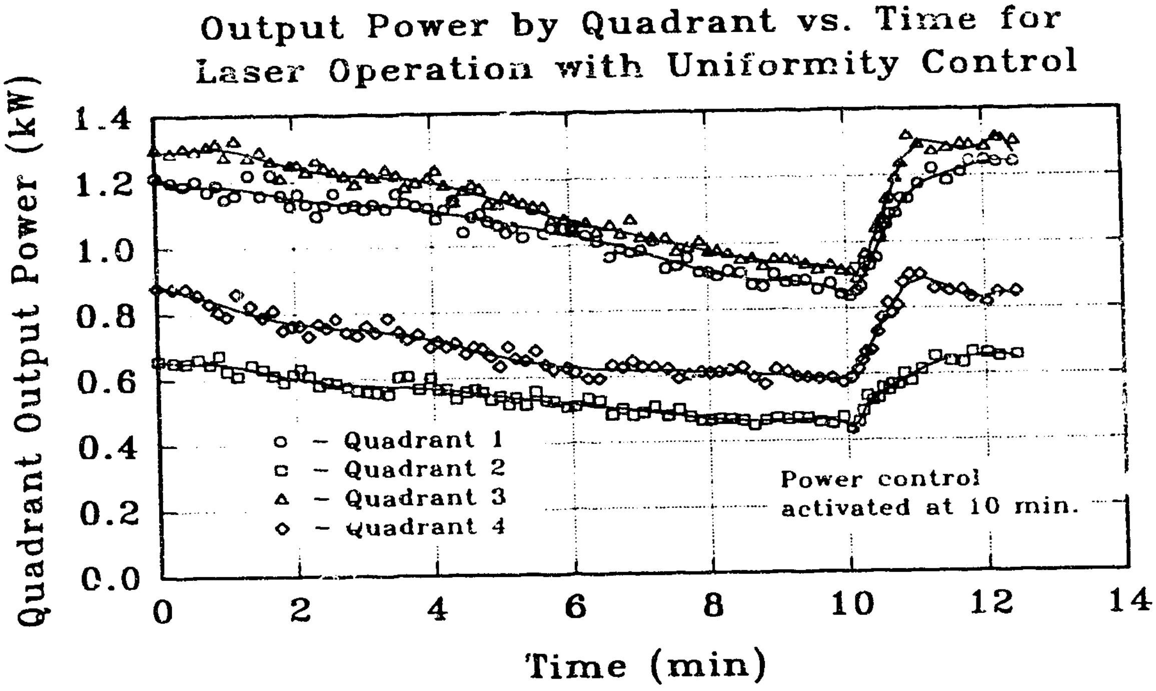


Fig. 4.14. Performance of system with only uniformity control activated

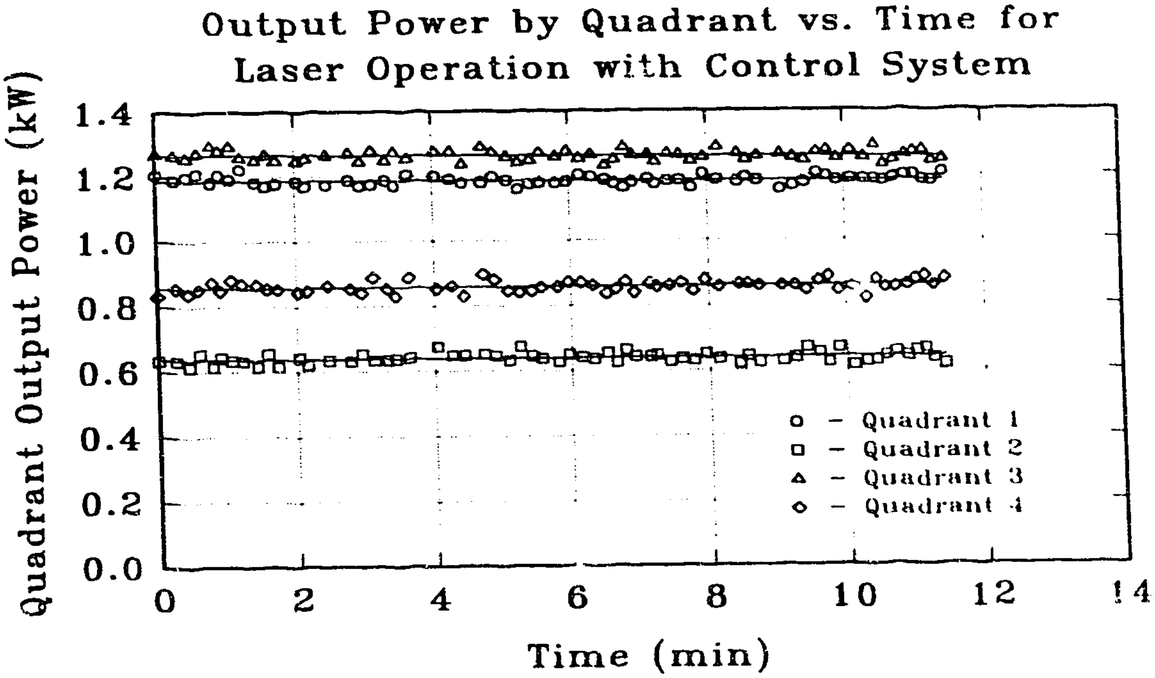


Fig. 4.15. Performance of system with uniformity and power control ac:.

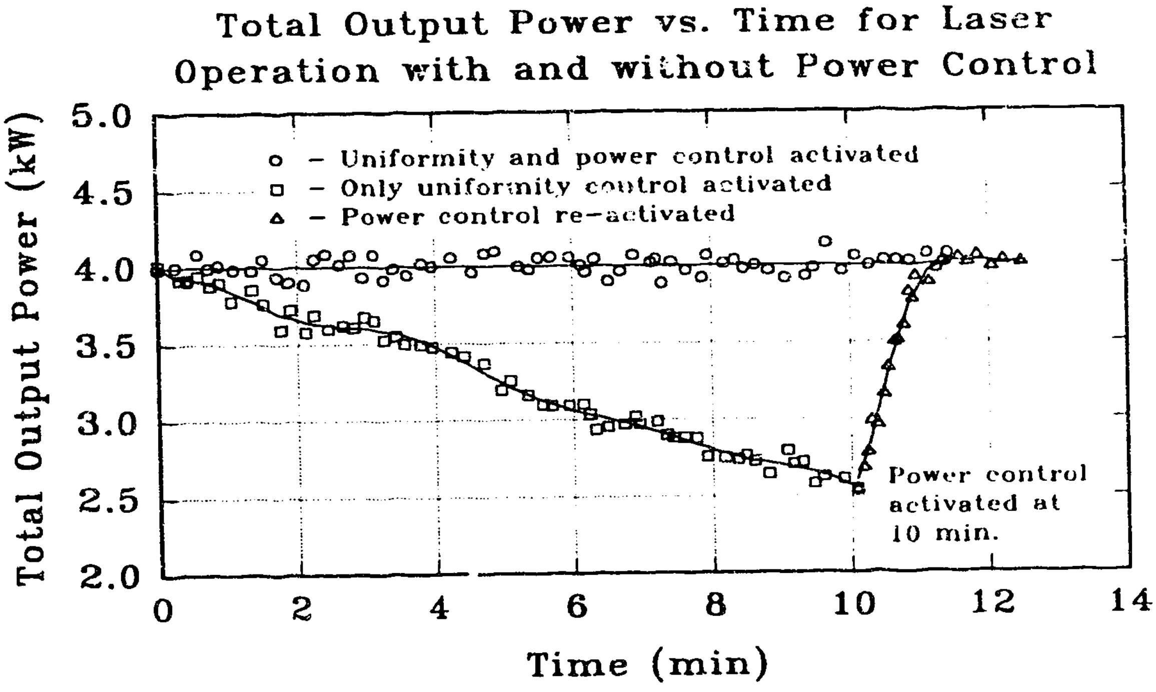


Fig. 4.16. Performance of power control system.

Once re-activated, the laser output power recovered to its original value within 1 minute, with no significant overshoot.

4.6 Controller Stability Analysis

The stability of any controller is critical to the safe and reliable operation of the associated system. Systems which exhibit unstable oscillations may be seriously damaged, requiring replacement of sensitive and expensive components. Certain systems, with large time constants and limited moving parts, are designed to operate with stable oscillations. However, systems such as the one described here, which require motor-driven micrometer and variac motion to stabilize the outputs, do not perform well with an oscillatory control strategy. These oscillations generally produce excessive wear on the motors and gear couplings, especially those with significant hysteresis. Thus, it is essential to ensure that closed loop oscillations are minimized, and preferably eliminated.

Determining the stability of non-linear controllers is a complex task. Many techniques, both analytical and numerical, have been devised, each possessing its own merits.[71] The use of simulation software and analytical methods to accurately model this system was complicated, due to the complex controller design, variable sampling period, and possible error in the imaging screen time constant and delay. Consequently, an experimental approach was utilized to determine the critical gain (gain for which system oscillates continually) of the system. The three controller gains were increased independently of each other until continuous cycling occurred for the corresponding output after the introduction of a small setpoint change. It was found that the critical gains for the horizontal and vertical uniformity controllers were approximately 2.8 µm/kgv and 1.9 µm/kgv. Since the gains being used are less than half the critical gains, the uniformity controller is stable. A similar investigation for the output power controller revealed that the gain could be increased to 10 s/A (a factor of 5) without occurrence of oscillations. This result, due to the limit of 5 control pulses (maximum 0.45 kW power change) per sample interval placed on the controller, indicates power controller stability. In addition to the low gains, many software safety features were implemented to prevent execution of large control actions, and inform the operator if error signals became excessively large.

4.7 Conclusions and Possible Enhancements

The experimental results presented herein clearly demonstrate that highly stable performance can be obtained from a non-sophisticated laser when using this vision-based computer control system. Considering the relatively sluggish motion of the power supply variac and motormics, the delay introduced by the imaging screen's time constant, and the relatively slow computational speed of the Amiga 2000, the controller performed admirably. Even with the non-optimum electrical components identified above, overall controller response should be adequate for most industrial processing situations.

Improvements to the control system could be made in six main areas. Firstly, extension of feedback control to the second (concave) resonator optic could result in a more stable output beam, since this mirror experiences similar thermal loads and al distortions. Secondly, performance could be enhanced dramatically by increasing computational speed with the addition of a Motorola 68030 accelerator board operating at 25 MHz. Furthermore, implementation of a micrometer position tracker in hardware would enable utilization of more conventional control techniques, while freeing computer time and resources for more useful activities. It should also be feasible, though considerably more expensive, to eliminate the time constant of the thermoluminescent imaging screen by utilizing a suitable infrared camera for direct conversion of the laser's profile into a video format. In addition, development of software to better analyse image data and obtain more accurate measures of uniformity and position would yield improved transient and steady-state responses. Finally, if a SCR controlled DC supply is used instead of the motor-driven variac, a much improved power stability would be expected. In this case, the stability limitation would be determined by the sensitivities of the SCR controller and the thermal imaging screen. Since the thermal screen can respond to power fluctuations of about 0.5%, an overall stability better than 1% is conceivable. Five of these possible improvements (all except the infrared camera) were implemented in the final design discussed in Chapter 6.

CHAPTER 5

CONTROL OF SECOND RESONATOR OPTIC

Following successful evaluation of the single-mirror uniformity and power stabilization systems described in Chapter 4, the feasibility of extending control to the second (concave) resonator optic was investigated. The motivation for this study arose from the fact that the single-optic controller attempts to compensate for distortions and misalignments of both cavity mirrors with only one of the optics. Although this method provided a substantial increase in performance, it is intuitive that expanding feedback control to include the second mirror should yield superior results. The development of an effective control strategy to encompass regulation of both optics requires generation of supplementary error information on which to base control decisions. Such data should be sufficiently sensitive to variations in concave mirror position, and exhibit a general pattern which may facilitate an operative alignment scheme.

As a result of the relatively slow clock speed of the Amiga, and its 16-bit architecture, performing additional computations and data manipulation would increase the already lengthy sampling interval. Thus, an accelerator board, containing a 25 MHz Motorola 68030 processor, a Motorola 68882 floating point unit, and 4 Mbytes of expansion RAM,[72] was added at this time. The card's faster clock rate and 32-bit design could increase execution speeds by 300% to 700%, depending on the complexity of the program code. The minimum possible sampling period was subsequently reduced from approximately 2 seconds to 0.6 seconds, although the average interval (including motormic movements and settling time delays) was only reduced to about 1.8 seconds.

5.1 Investigation of Control Strategies

Determining an effective control procedure required examination of a number of error measures. At first, the concept that automatic alignment of the concave optic would further increase beam uniformity appeared practical. Consequently, an overall uniformity error measure, E_u, based on the sum of absolute differences between quadrant intensities, was derived as follows:

$$E_u = |Q_1 - Q_2| + |Q_1 - Q_3| + |Q_1 - Q_4| + |Q_2 - Q_3| + |Q_2 - Q_4| + |Q_3 - Q_4|. \qquad (5.1)$$

The concave mirror position was then varied in a X-Y grid pattern, while the convex mirror maintained beam uniformity. Each resulting grid point, depicted in Fig. 5.1,

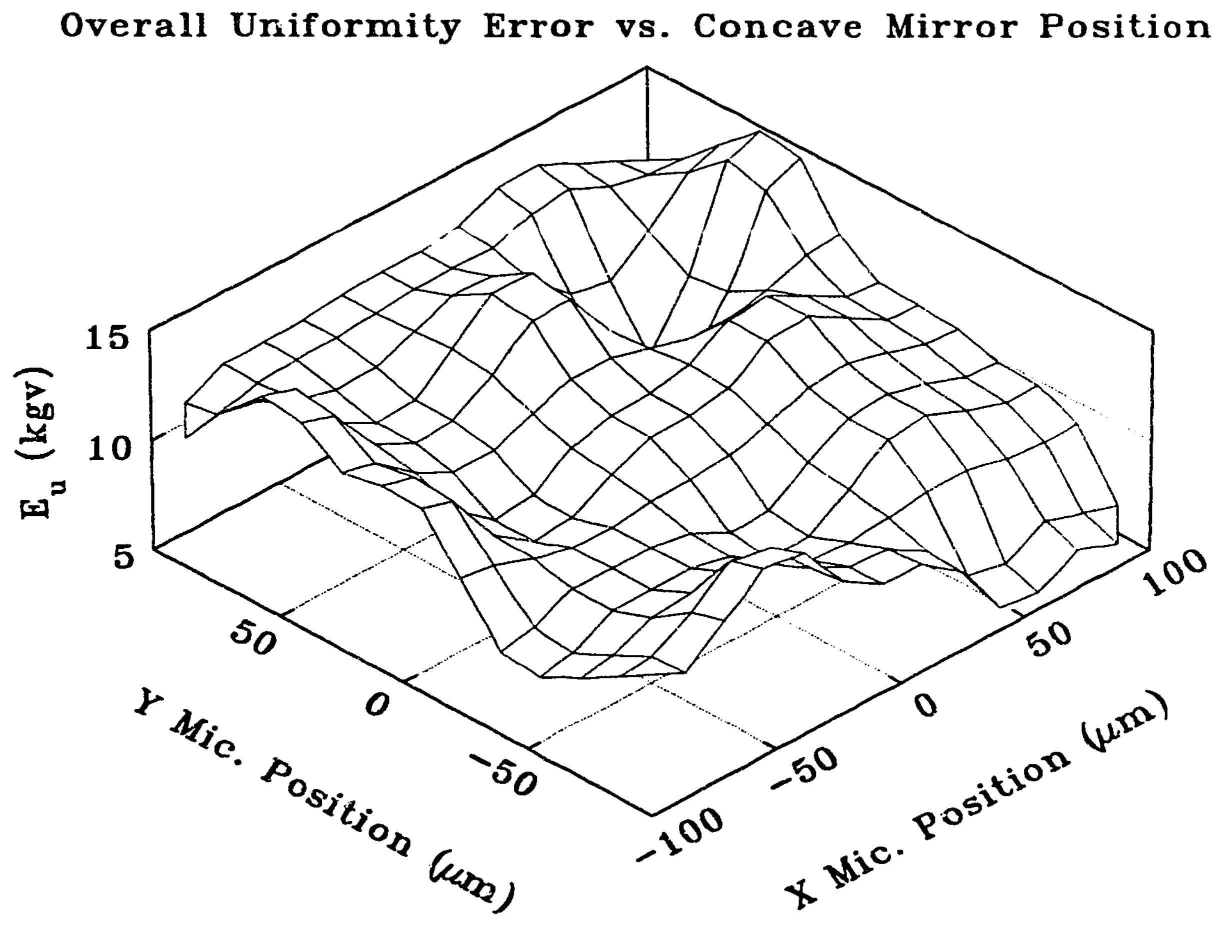


Fig. 5.1. Overall uniformity error as a function of concave mirror position.

represents the overall uniformity error, E_u, under conditions of optimal alignment. If the shape of this error profile was similar to an inverted gaussian surface, with a unique and distinct minimum, the error, E_u, could be used to control the concave mirror. Since such a minimum does not exist, it was concluded that the overall beam uniformity cannot be significantly improved with a two-mirror alignment system. The total beam intensity (proportional to the total output power) was also monitored at each of the grid points to determine if the concave mirror position effects laser efficiency. However, the intensity data did not disclose any observable patterns since the levels, including output power, remained approximately constant.

Further analysis of the data revealed the expected phenomenon of beam steering, illustrated in Fig. 5.2. When the concave mirror is positioned such that it is not perpendicular to the resonator axis, the controller compensates by misaligning the convex mirror to maintain parallelism of the optics. This yields a relatively uniform and symmetric beam, provided the following conditions are satisfied:

$$D_{cx}, D_{cv} > \sqrt{\lambda L} \quad \text{and} \quad \frac{L \tan\alpha}{2} < \sqrt{\lambda L}. \tag{5.2}$$

The data suggested that beam position, characterized by its centre, varied proportionally with concave mirror position. Although the beam centre, which was computed based on the inner and outer beam diameters, was severely contaminated

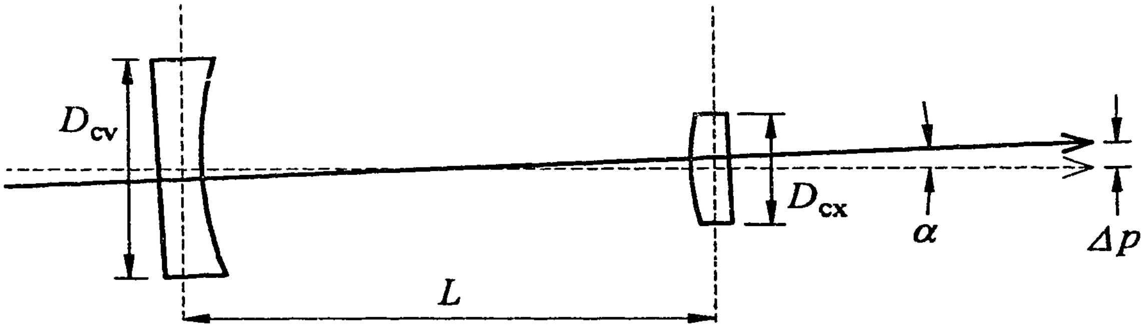


Fig. 5.2. Resonator parameters to characterize beam steering.

with noise, the general trend was clearly evident. Excessive noise was a direct result of the Amiga's insufficient gray scales (16 levels) and the limited pixel resolution of the digitizer (320 × 200 pixels), thereby, preventing accurate detection of beam edges and diameters. Thus, development of adequate filtering methods and additional measures of beam position quantification were necessary.

At this point, algorithms to compute the weighted and unweighted beam centroids (described in Section 3.1.3) were implemented. Subsequent sensitivity measurements indicated that the susceptibility of these signals to noise was much greater. However, they did provide supplementary information pertaining to beam uniformity. In the aligned state, the absolute difference between the weighted and unweighted centroids was minimal. As the uniformity error increased, the difference became more pronounced, with the weighted centroid drifting further from the actual beam centre. As a result, this error signal could provide an alternate means of uniformity control for systems possessing less noise, superior gray scale resolution, and greater pixel densities.

A final technique to accurately ascertain the beam centre involved comparison of intensity data to a previously acquired image of an optimal beam cross section. This method first required storage of the series of pixel coordinates (the overlay) which comprised the shape of the best possible output cross section. The overlay was then superimposed on the sampled image, and the total intensity of the intersecting area was calculated. Similar intensity values were obtained as the overlay position was varied along the positive and negative X and Y axes, until a maximum was reached. The overlay centre corresponding to this maximum was used to represent the beam centre. Although this technique produced consistent results, with relatively low noise, it increased computation time considerably. Depending on the number of total intensity calculations required (at least five, not more than twenty), the minimum sample time increased to over 3 seconds, with the average sample period (including micrometer movements and delays) being about 5 seconds. Consequently, this method of beam centre determination was discarded, and effort was directed at filtering the original centre signals.

Application of conventional filters to the beam centre error signals was difficult due to the variable sampling time of the system, which prevented a suitable choice of cutoff frequency. Thus, a simple, four-element, moving average filter was employed, which produced acceptable steady-state results. However, if a sufficient uniformity error existed, such that the beam was not near symmetry, the corresponding centre calculation was in error. When uniformity was restored (usually in less than three sample intervals), the centre remained in error due to the phase lag of the filter, and required an additional three or four sample periods to accurately reflect the centre position. Since this phase lag could be accommodated with appropriate controller design, no further investigation was conducted.

5.2 Beam Steer System Model

Similar to the uniformity control system, the beam steer reduction system may be subdivided into the horizontal and vertical feedback loops, depicted in Fig. 5.3. As described previously, motormic speed gains, K_{xs} and K_{ys}, are set between 10 and 20 μm/s, while the motormic itself is modelled as an integrator. The laser and infrared detector are combined into a first-order block, with gain parameters, K_{xp} and K_{yp}, relating micrometer motion to beam centre. The two interconnecting gains, K_{xpy} and K_{ypx}, represent variation in beam centre when the complementary micrometer is adjusted. Additional interdependencies, K_{xup} and K_{yup}, model the disturbance to the centre measurement when uniformity errors exist (i.e. beam cross section is not symmetric). Signals, U_{xu} and U_{yu}, emanate from the uniformity feedback control system illustrated in Fig. 4.2. Interconnections between the output power and beam position control systems do not exist, since power levels are not dependent on concave mirror position, but only on beam uniformity.

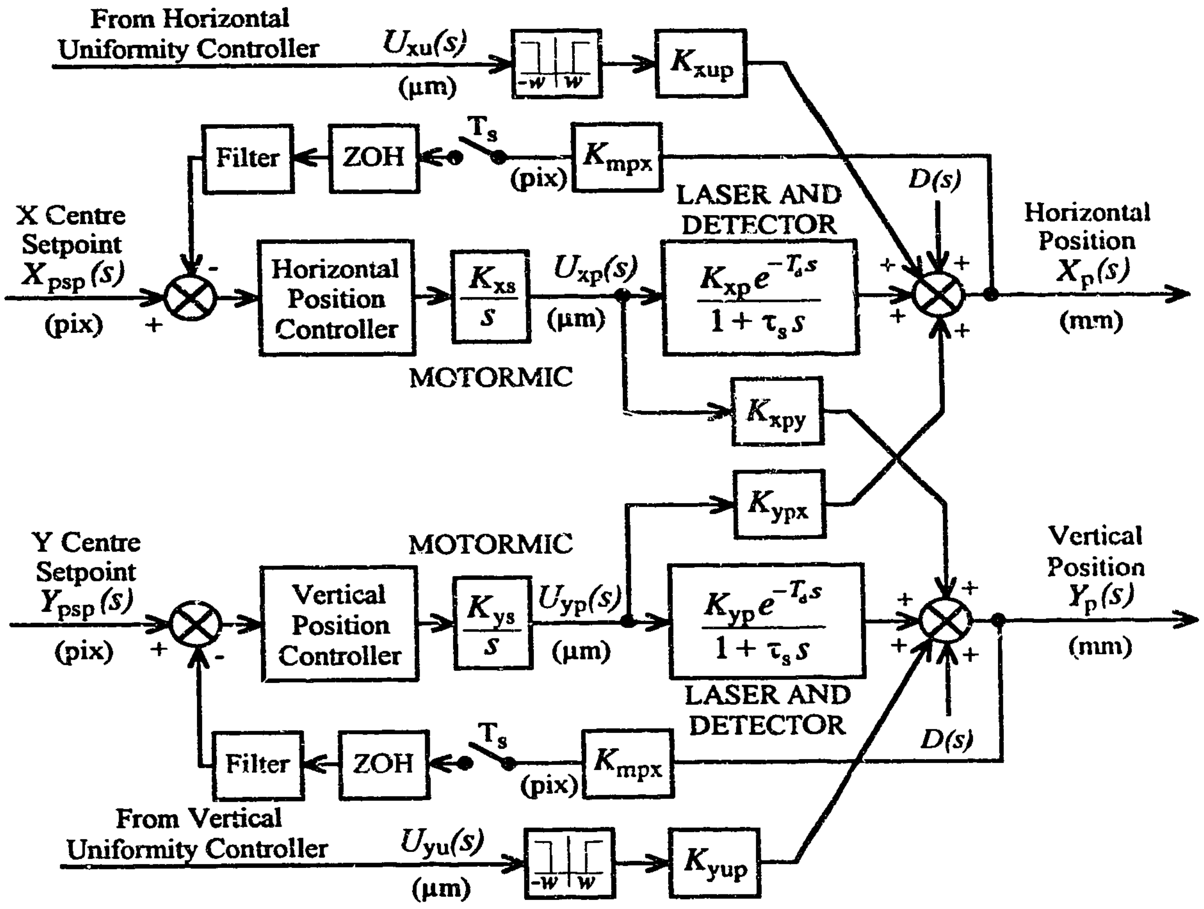


Fig. 5.3. Model of feedback control system for beam steer compensation.

5.2.1 Model Parameter Estimation

The process gains, K_{xp} and K_{yp}, were determined by varying the position of the concave mirror along the associated alignment axis, with uniformity control of the convex optic activated. After each concave mirror movement, uniformity was restored with the convex unit, and the position of the beam centre was recorded. This data presented in Fig. 5.4. To overcome the phase lag of the moving average filter, centre coordinates were stored only if uniformity error was within the

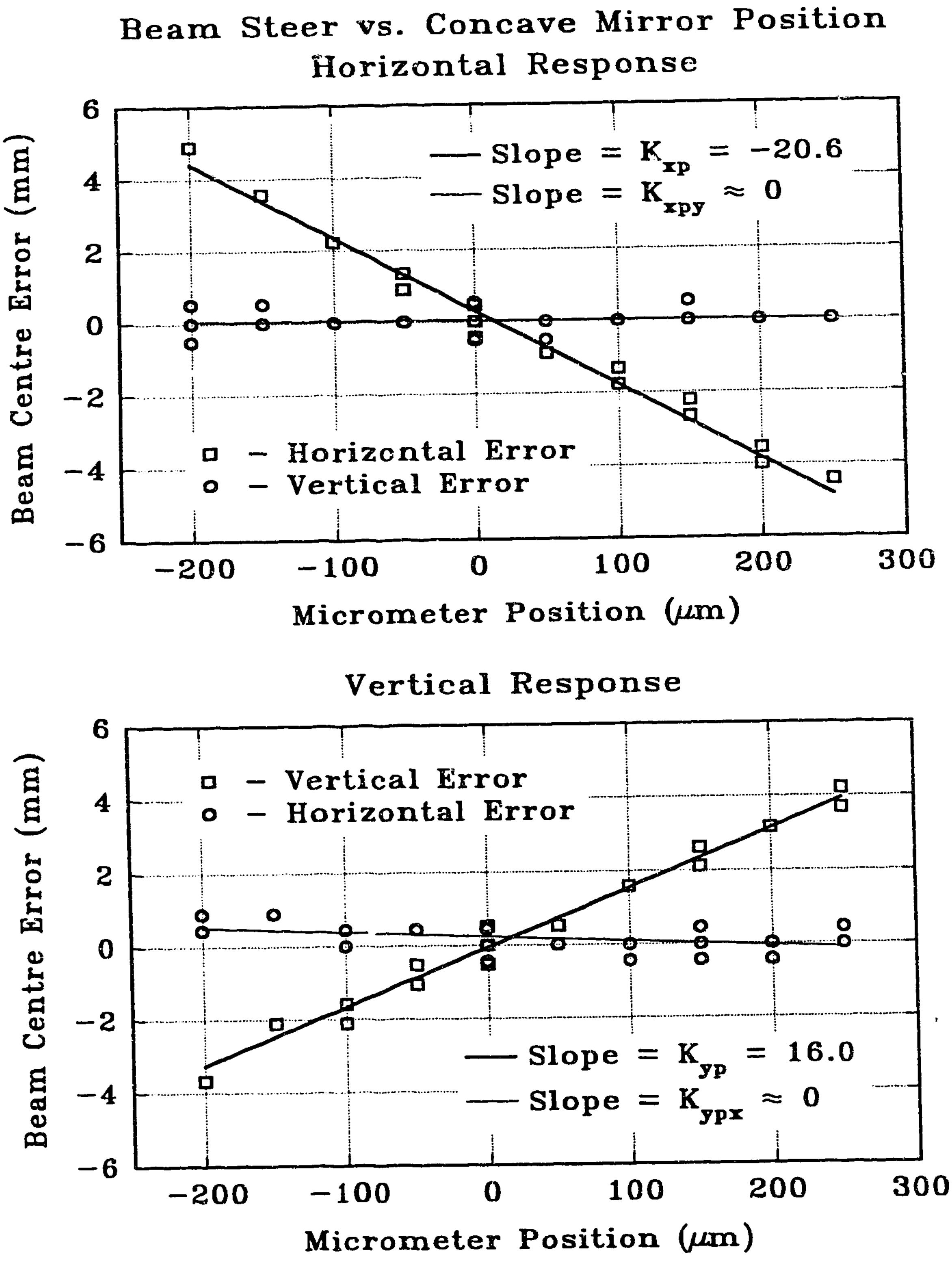


Fig. 5.4. Sensitivity of beam centre to changes in concave mirror position.

acceptable threshold window for at least four sampling intervals, thereby, ensuring an accurate centre measurement, without noise or phase lag. Results indicate that the responses are linear, [illegible] noise margins of about ±0.5 mm (±1 pixel). The corresponding angular deviation, α, was determined to be approximately 54 µrad based on the distance of the imaging screen to the centre of the resonator cavity (about 9.3 m). Gains, K_{xp} and K_{yp}, were computed from the slopes to be -20.6 and 16.0 respectively, while the interconnecting gains. K_{xpy} and K_{ypx}, were approximately zero. The error is expressed in millimeters (for better comprehension), but is actually controlled in pixels. The associated conversion factor may be calculated from the size of the computer monitor (26.7 × 19.7 cm), the magnification factor of the video camera (1.88), and the digitizer's pixel resolution (320 × 200). Thus, the conversion gains, K_{mpx} and K_{mpy}, were computed to be 2.25

Model Parameter	Value
K_{xs} and K_{ys} for concave optic	10 to 20 µm/s
K_{xp} process gain	-20.6
K_{yp} process gain	16.0
K_{mpx} conversion gain	2.25 pix/mm
K_{mpy} conversion gain	1.91 pix/mm
K_{xpy} and K_{ypx} interconnecting gains	~0
K_{xup} and K_{yup} when $-w < U_{xu}(s)$ and $U_{yu}(s) < +w$	~0
τ_s for imaging screen detector	~1.0 s
T_d for imaging screen	~0.4 s
T_s for all feedback loops	0.6 to 2.2 s
w for position controller deactivation	4 µm

Table 5.1. Estimated parameters for beam position stabilization model.

and 1.91 pix/mm respectively. A summary of model parameters is given in Table 5.1.

Interdependencies between the uniformity controller and the beam steer system, K_{xup} and K_{yup}, may be eliminated in a manner similar to that between the uniformity and output power control systems. When the uniformity controller outputs, U_{xu} and U_{yu}, are outside the threshold window ($\pm w$), the beam position controller may be disabled, permitting the feedback loops to be modelled as two, independently controlled, SISO systems. The value of the threshold, w, was chosen to be 4 μm, equivalent to the minimum error margin for the uniformity controller.

5.3 Beam Position Controller Design

As with the design of the uniformity controller, a type of optimal control was employed for beam position stabilization, to overcome hardware limitations and system non-linearities. A schematic diagram of the position control system is illustrated in Fig. 5.5. Horizontal and vertical error inputs are first converted to microns by applying the controller gains, K_{cpx} and K_{cpy} (in μm/pix). The signals X_p' and Y_p' (specified in pixels) denote the filtered outputs X_p and Y_p. Since only one micrometer may be moved at a given time, the X and Y errors are compared, and the controller corresponding to the larger is enabled, providing the error is not within the minimum threshold window ($\pm w$). However, the control action is performed only if a uniformity error did not exist for the previous four sample intervals, thus, guaranteeing noise-free centre measurements with no phase lag. Following execution of the control action, the necessary delays of 0.2 seconds for micrometer deceleration and 1.0 seconds for imaging screen response were implemented, and the next sample was obtained. This sample generally contained a significant uniformity error which automatically activated the uniformity controller to regulate convex mirror position.

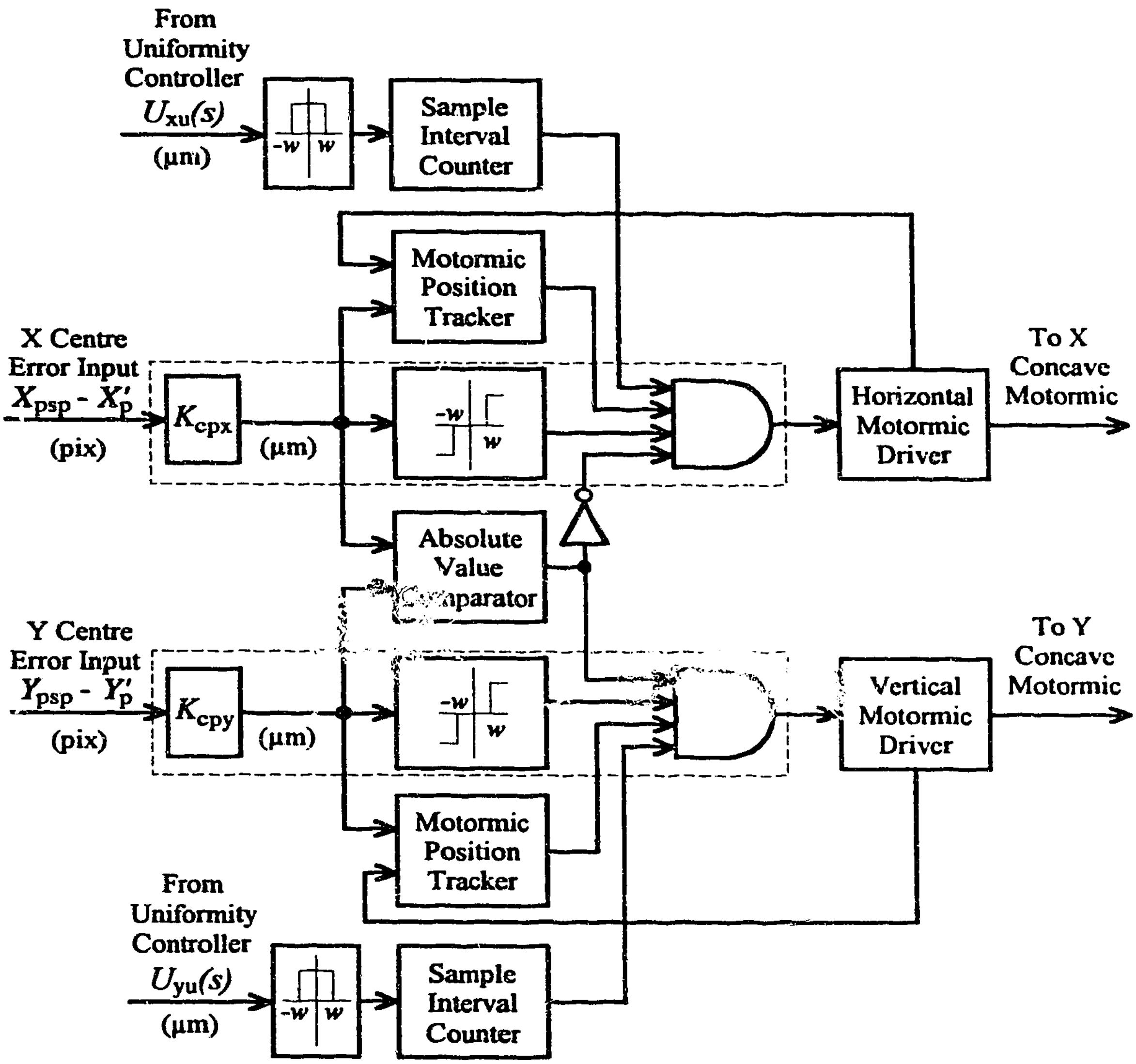


Fig. 5.5. Schematic of beam position control system tor concave mirror.

In order to restore beam position in a single control step, the controller gains should be the reciprocal of the process gains, and may be calculated as follows:

$$K_{cxp} = \frac{1}{K_{xp}K_{mpx}} = \frac{(10^3\ \mu m\ /\ mm)}{(-20.6)(2.25\ pix\ /\ mm)} = -21.6\ \mu m\ /\ pix \qquad (5.3)$$

$$K_{cyp} = \frac{1}{K_{yp}K_{mpy}} = \frac{(10^3\ \mu m\ /\ mm)}{(16.0)(1.91\ pix\ /\ mm)} = 32.7\ \mu m\ /\ pix. \qquad (5.4)$$

After performing closed loop evaluations using the above values, gain modifications were necessary. Firstly, the gains needed to be increased by about 15% (to 25 and 38 μm/pix respectively) to compensate for the reverse rotation of the motormics when they are halted (explained in Section 4.4). Since the minimum error resolution is only 1 pixel, the possible concave motormic movements are in multiples of 25 μm for the horizontal direction and 38 μm for the vertical. However, as discussed previously, concave movements greater than 25 μm result in large disturbances to the uniformity error, which cause controller operation outside the modelled linear region. Thus, the optimal nature of the uniformity controller diminishes, producing slower transient responses. In addition, large uniformity errors at high power levels create areas of increased power density, which may seriously damage optical components. Consequently, the position controller was restricted to movements in 25 μm steps, which limited disturbances to beam uniformity and permitted operation within the linear region. The position error window was set to ±1 pixel (approximately ±0.5 mm) for accommodation of the noise margin.

5.4 Positional Control System Performance

Performance of the beam position control system was evaluated based on its transient and steady-state responses, and by comparison to data obtained with only uniformity and power control enabled. The following responses were recorded at an output power level of 4 kW.

5.4.1 Transient Responses

Closed loop transient responses for both the X and Y position controllers are presented in Fig. 5.6. The first graph illustrates the horizontal response, in which the system was initially operated with only uniformity control, and a horizontal position error of 1.33 mm (3 pixels). At 0.8 minutes, the position controller was activated,

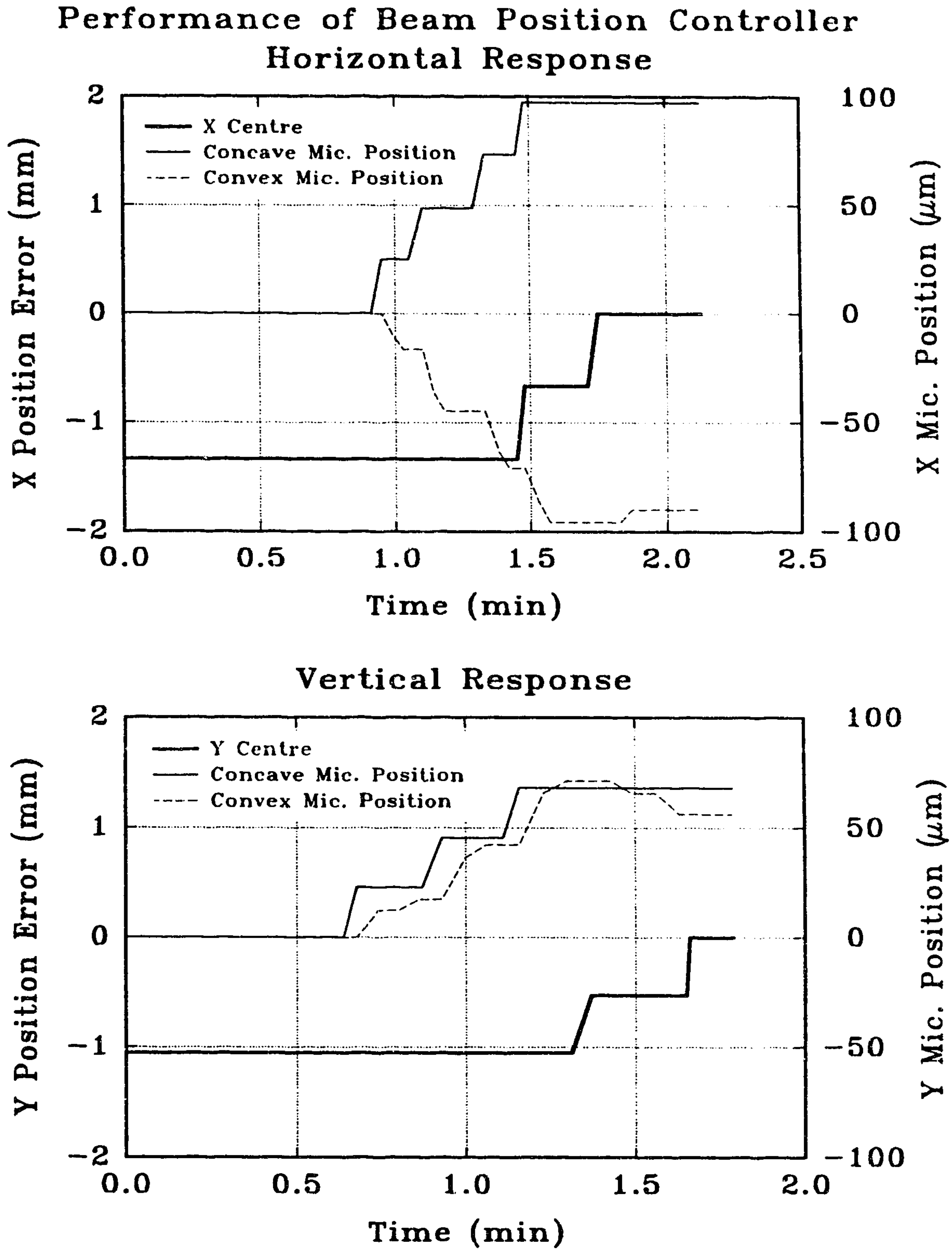


Fig. 5.6. Transient response of beam position controller.

causing an immediate +25 μm movement of the concave mirror. Upon digitization of the next sample, the uniformity controller initiated motion of the convex mirror in the opposite direction to compensate for deterioration in beam symmetry. Once uniformity was restored for four consecutive sample periods, the position controller initiated another +25 μm concave movement, and the process repeated until the position error was corrected. Although the error was eliminated completely (due to continued drifts in alignment), the position controller ceased regulation when the error was within ±1 pixel window (at about 1.4 minutes). Thus, the error was stabilized in approximately 50 seconds.

A similar analysis of the vertical position controller's performance (shown in the second graph) revealed that a Y centre error of 1.05 mm (2 pixels) was corrected about 75 seconds after the controller was activated (at 0.7 minutes). The longer response time was due to the 25 μm movement limit, which is 1.5 times less than the preferred controller gain, K_{cpy}, of 38 μm/pix. The plot also indicates that the uniformity controller compensates with the convex mirror in the same direction as the concave, instead of in an opposite direction as for the horizontal response. However, the vertical motormics for each mirror are mounted on opposite ends (i.e. possess opposing moment arms), thereby causing opposite angular motion for equivalent transverse motion.

5.4.2 Steady-State Responses

Overall performance improvement and steady-state response was examined by observing laser operation, both with and without beam position control. Fig. 5.7 illustrates system performance with the position controller deactivated, while uniformity and power stabilization remained enabled. Although the beam centre originates within the error window of ±0.5 mm, thermal stresses on the optics, mounts, and laser tank, cause the centre to drift by almost 2 mm after 9 minutes of

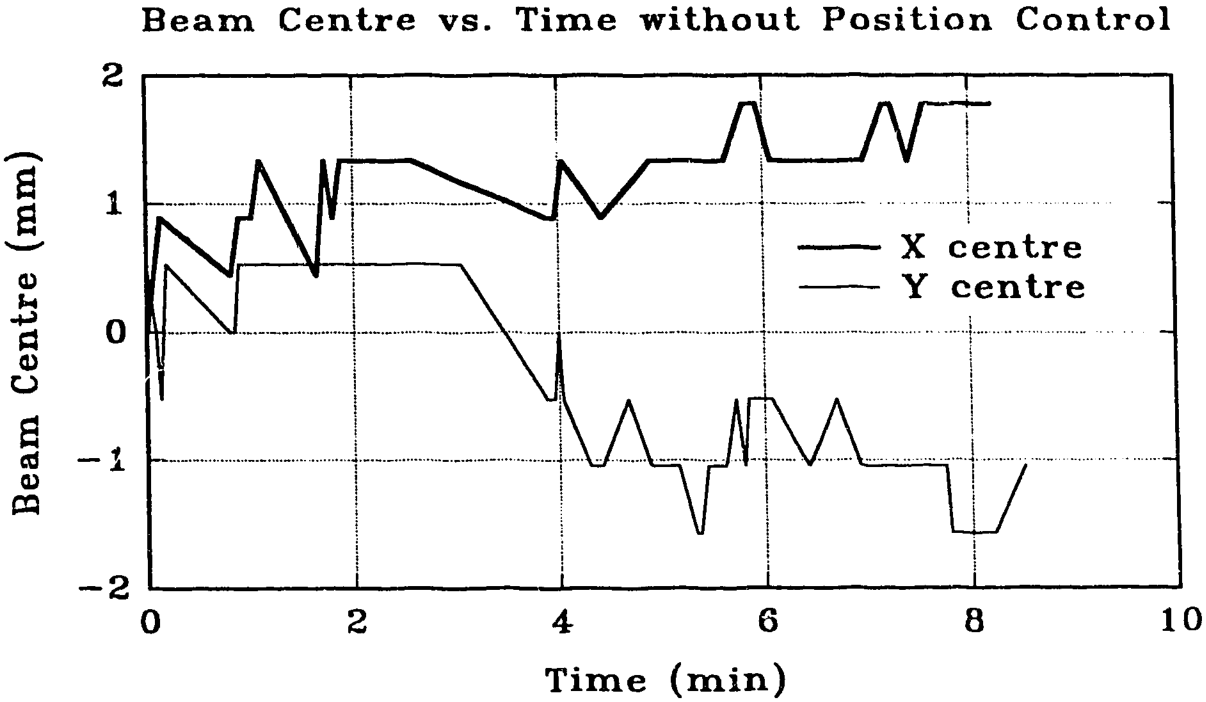


Fig. 5.7. Beam steering of system when position controller is deactivated.

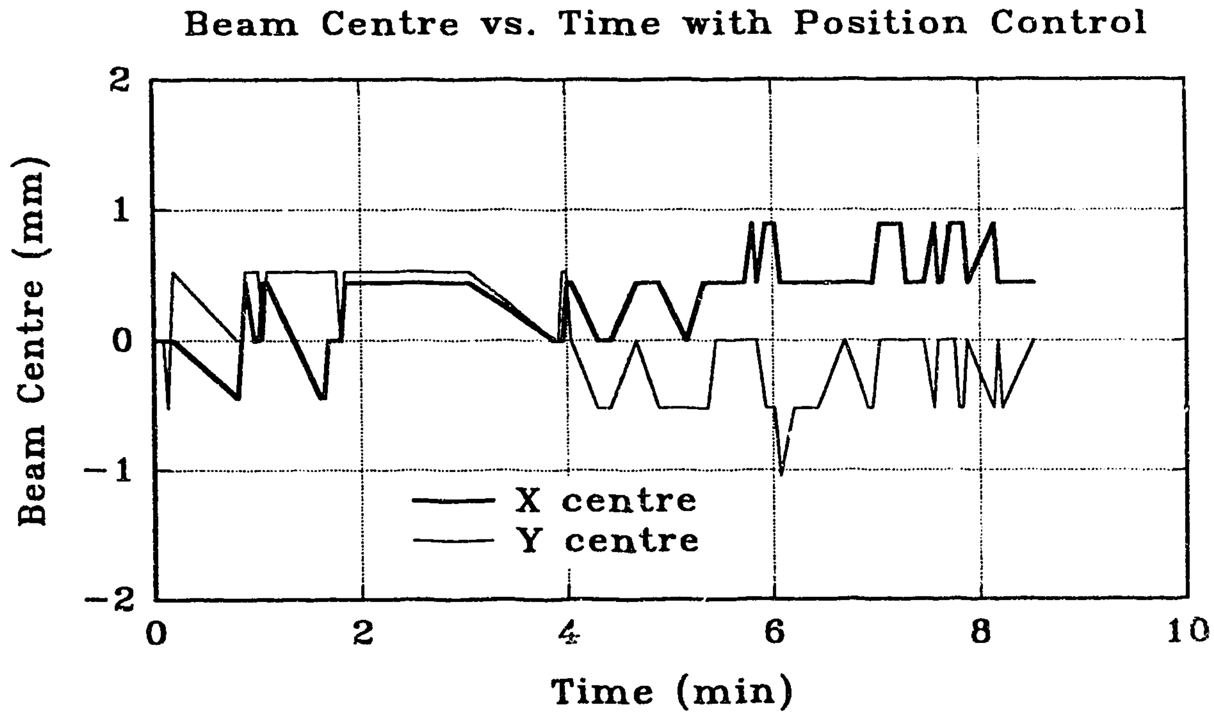


Fig. 5.8. Reduction of beam steering when position controller is activated.

operation. This corresponds to a beam steering angle of approximately 0.2 mrad. Increases in lasing gas contamination after 15 to 20 minutes resulted in discharge instabilities, and prevented investigation of steering effects over longer periods of time. Fig. 5.8 depicts system performance with position control enabled. As evident, the controller attempts to maintain the beam centre within ±0.5 mm. If the error drifts outside this window, the controller regulates the concave optic to maintain beam position, followed by the convex mirror to restore uniformity. The results show that the beam centre was regulated within ±2 pixels (±0.89 mm horizontally and ±1.05 mm vertically). During these control sequences, the output power remained stabilized.

5.5 Discussion and Conclusions

The results obtained indicate that beam steering may be reduced significantly with the vision-based position controller described herein. This controller maintained the beam position within ±2 pixels, which corresponds to an angular deviation of approximately 0.11 mrad. Transient response times to restore a 1 mm positional error were approximately 50 seconds for the horizontal direction and 75 seconds for the vertical.

Stability of the system is ensured since concave mirror control movements are restricted to 25 μm steps. Since micrometer motion of this step size does not effect the beam centre by more than 1 pixel (half the width of the error threshold window), oscillatory responses are not possible. Theoretically, the critical gains for this type of control scheme are twice the reciprocal of the process gains, or -43.2 and 65.4 μm/pix respectively for the horizontal and vertical controllers. Thus, even if the system was operated without a minimum error window, single pixel errors would not cause oscillations since the 25 μm movement step size is almost half the critical gains.

The relatively slow transient response of the system could be improved considerably if simultaneous movement of more than one motormic was possible. Such a system could incorporate feed-forward control to compensate for uniformity disturbances (using the convex optic) while stabilizing beam position. Since this technique would minimize loss of beam symmetry due to positional adjustments, the delay of four sample periods to accommodate the filter phase lag may not be necessary. In addition, modification of the system to utilize a constant sampling period would permit use of more efficient filters, which may further increase performance.

CHAPTER 6

FINAL DUAL-OPTIC CONTROL SYSTEM

The performance of the output beam control system developed to this point has been evaluated exclusively on the PIE-3 CO_2 laser. Although this device was capable of output powers approaching 10 kW, long-term operation at these levels was not possible due to an inadequate cooling system and contamination of the lasing gases. In fact, to achieve continuous operation (from startup) for 15 minutes, the power level could not exceed 4 kW. Following each of these relatively short runs, a down-time of approximately 3 hours was necessary for evacuation of the laser tank and refilling of gases. Consequently, testing modifications to the software-based controller was a laborious and time consuming task . For instance, the time required to isolate and correct a simple program error generally necessitated system shut-down. As a result, the majority of software debugging was performed using video-taped results from outputs of previous runs.

In addition to limited continuous operation, the relatively low power level prevented controller evaluation at the higher output powers used by many current laser material processing systems. Certain specialized applications, such as those which utilize materials of varying thicknesses and composition, require processing over a wide range of power levels, at increased speeds. Therefore, controller testing under these conditions is essential to fully evaluate performance. Furthermore, incorporation of the control system into an industrial quality laser, capable of high power levels and long-term operation, would further validate results obtained to this point Also, this transition from a research to a more industrial environment would demonstrate the versatility and flexibility of the system.

The newly acquired PIE-4 laser (described in Section 2.1.2) proved ideal for this task. Its resonator arrangement was virtually identical to that of PIE-3, enabling the optic alignment servos (motormics) to be installed without .uch modification. Its industrial quality design possessed many enhancements, including a chilled water(~6 °C) gas cooling system and a superior vacuum tight cavity, which permitted continuous operation at high power levels for up to 8 hours. Therefore, the control system was transferred to this laser and a complete performance evaluation was performed.

6.1 System Modifications and Enhancements

In order to implement the system on the PIE-4 laser, certain hardware and software modifications were necessary. Firstly, the imaging screen was enlarged to accommodate the increase in beam size. Next, optical mounts were altered for motormic installation. Since spatial restrictions did not permit usage of the differential micrometers as an intermediate drive stage, the motormics were mounted such that they directly adjusted optical position. Fortunately, the larger moment arms of the optical mounts reduced loads sufficiently to facilitate this modification. Unlike the motor-driven power supply variac of the PIE-3 laser, PIE-4 incorporated an SCR controlled supply. This system required an input voltage setpoint between 0 and 15 V, which corresponded to an approximate power range of 0 to 25 kW. Since the data acquisition unit possessed two 8-bit D/A converter, one of these outputs was used to control laser power level. To improve sensitivity of the 8-bit control signal and to prevent large changes in output power (for safety reasons), a combination of manual and automatic adjustment was implemented. The manual setpoint determines the mean of the desired operating power range, while the computer-controlled setpoint allows changes about this value. The resolution of the 8-bit D/A setpoint permitted minimum power variations of about 0.025 kW, up to a maximum deviation of about ±2.5 kW.

Modifications to the control system software were minimal. The increased load on the optic alignment motormics, due to the elimination of the differential micrometer drive stage, introduced a significant amount of hysteresis. Consequently, backlash compensation routines were developed, similar to that for the PIE-3 power supply variac. The only other major change was to the relay-type power control algorithm, which was converted to standard PI (proportional-integral) controller.

6.1.1 Improvements to PIE-4 Control System

In addition to the required modifications, a number of enhancements (recommended in Chapters 4 and 5 based on results obtained) were incorporated. The most significant of these was the improvement to the data acquisition unit (described in Appendix B), to include a motormic position tracker and more digital input/output capabilities for better interfacing with external devices. The board was designed to fit directly onto the unit card (i.e. "piggy-back" style) within the computer. Thus, the entire data acquisition and control system was self-contained and well shielded from noise generated by the laser's high voltage pulser. All connecting cables were shielded, with separate shield and signal grounds, to prevent noise from entering the computer. Due to spatial restrictions within the computer, and the available programmable logic devices, only one 16-bit counter could be implemented to record the positional displacement of any one motormic. This counter could be preset to the desired movement value before activation of the motormic. The logic was such that the motormic was halted when the counter reached zero. Therefore, the control algorithm could continue to acquire beam image samples and compute error signals while the motormic was in motion. Its displacement from the starting position could be read directly from the counter, and an absolute position could be calculated and correlated to error signal data. As a result of this improvement, the controller was altered to utilize a constant sampling period, which in turn permitted design of more efficient filters for use with the beam position controller.

A further software enhancement was developed to overcome the reverse motion of the motormics, which occurs after they are halted. Due to the increased load, this effect was more pronounced in the PIE-4 system, and was observed to vary non-linearly with motormic speed. Thus, an adaptive compensation technique was employed to monitor the actual and specified movements, and increase control actions by the necessary factor to eliminate any difference. This gain factor was continually updated each sample period, thereby insensitizing controller performance to motormic speed settings.

The higher output power levels and wider range of operation could not be accommodated by the infrared detection screen, under conditions of constant percentage sampling. Thus, the variable coupling rotating wand, summarized in Section 2.2.2, was developed to ensure sensor operation within its optimal dynamic range. The second D/A output of the data acquisition unit was utilized to issue a 0 to 15 V setpoint to the associated rotational speed controller (detailed in Appendix A).

6.2 System Overview and Apparatus

A schematic diagram of the complete measurement and control system for the PIE-4 laser is illustrated in Fig. 6.1. The laser output was sampled with the variable coupling wand and directed onto the imaging screen. The resulting intensity profile information was then digitized and analysed using the image processing system described previously. Each of the resonator optics was controlled with the alignment apparatus utilized in the PIE-3 systems. This consisted of two motormic drivers for the convex and concave mirrors, with each driver controlling two motormics (the horizontal and vertical). The 16-bit counter added to the data acquisition unit monitored the direction and displacement signals (DIR and CNT) for the active motormic, to maintain relative position. The unit's two 8-bit D/A outputs acted as the setpoints for the rotational wand speed controller and the input power supply. Input current was monitored using a current shunt and an isolation amplifier, while a resistive divider network provided input voltage data. Each was a 0 to 5 V signal, linearly proportional to the actual values. Monitoring of the input current and voltage permitted on-line calculation of overall laser efficiency.

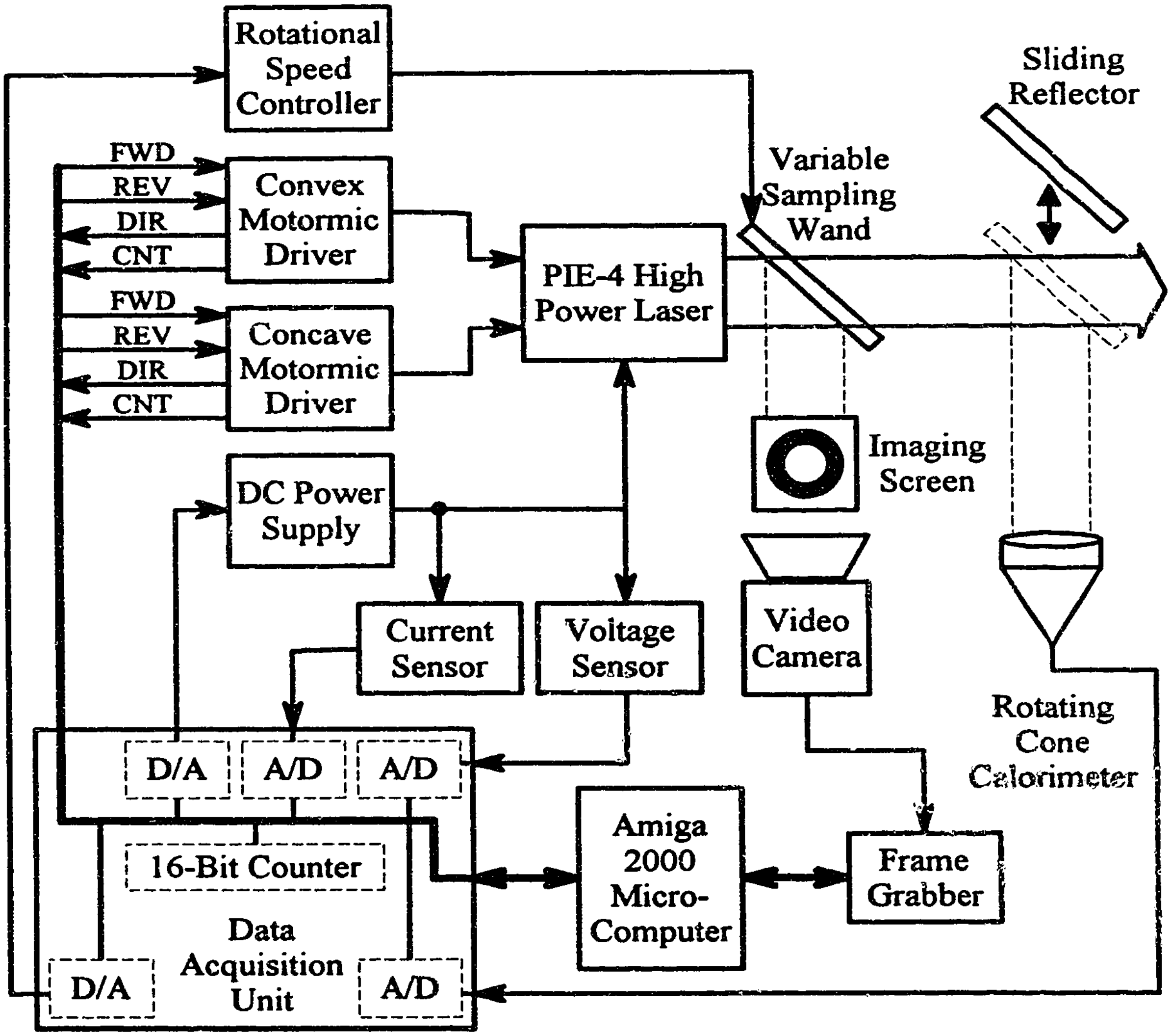


Fig. 6.1. Schematic diagram of experimental setup and apparatus for PIE-4 system.

The PIE-4 laser did not possess a sampled power measurement system similar to the PIE-3 device. Instead, a pneumatically controlled sliding mirror reflected the output beam into a rotating cone calorimeter[73] (shown by dashed lines in Fig. 6.1). Thus, direct power measurement could not be performed without interruption of the laser application, further emphasizing the need for an on-line power monitoring technique. The output from the calorimeter (0 to 5 V) was also linearly proportional to the laser power level. This signal was only utilized to calibrate the infrared detection screen and obtain an intensity-power gain factor. Once calibration was complete, the beam could be returned to the laser process, while power stabilization and measurement was performed based on imaging screen intensity information.

6.3 Complete System Model

The final system model consists of the beam uniformity, beam position, and output power control systems, discussed in Chapters 4 and 5, and the coupling percentage controller used to maintain optimal sampling levels. Since all interdependencies among feedback control loops were eliminated on the PIE-3 system, with appropriate controller design, the corresponding interconnecting blocks have been removed from the overall system model. Thus, the final MIMO system may be subdivided into six individually controllable SISO systems - the horizontal and vertical beam uniformity control system, the horizontal and vertical beam steer reduction system, the output power regulation system, and the sampling percentage control system.

6.3.1 Alignment Control Systems

The four feedback control loops of the uniformity and beam position systems are depicted in Fig. 6.2. System models are virtually identical to those of PIE-3, except for inclusion of motormic hysteresis blocks due to the increased loads. The

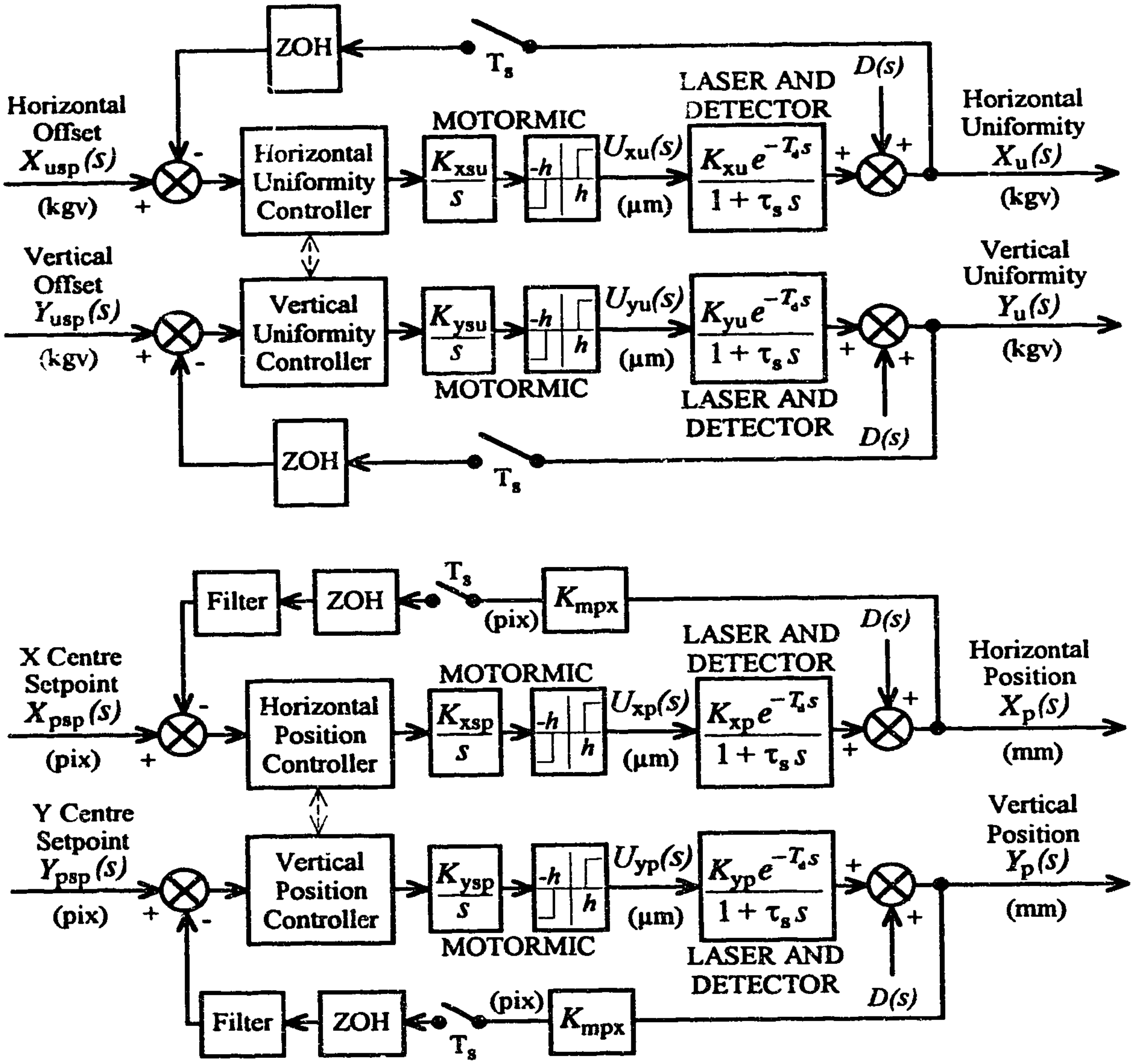


Fig. 6.2. Model of complete resonator alignment control system.

gain oarameters K_{xsu}, K_{ysu}, K_{xsp}, and K_{ysp}, represent motormic speeds, which are manually set between 5 and 10 μm/s. These settings are considerably slower than those used during testing of the PIE-3 system, as a direct result of the elimination of the intermediate micrometer drive stage. Increasing these values above 10 μm/s prevented execution of small motormic adjustments, thereby degrading system performance. The laser and infrared detector are again combined into a single block, with the uniformity process gains, K_{xu} and K_{yu} (in kgv/μm), relating uniformity error to motormic displacement, while the beam steer process gains K_{xp} and K_{yp}, perform a similar function for the position error. Gain blocks K_{mpx} and K_{mpy}, in the beam position feedback loops, convert error signals from millimeters to pixels for comparison to the setpoint. As a result of the increased size of the PIE-4 beam cross section, these factors decreased to 1.89 pix/mm for the horizontal direction and 1.60 pix/mm for the vertical. The change to a constant sampling period facilitated selection of the position error filter type from one of the following three: four element moving average, first order low-pass, or third order Butterworth.

6.3.2 Power and Intensity Control Systems

Closed loop models for the output power and coupling percentage regulation systems are illustrated in Fig. 6.3. The first loop details the sampling percentage control system. The wand's rotational speed controller is represented as a first order block with delay, and requires an input setpoint, U_w, in volts. Its, gain factor, K_{rv}, produces a change in rotational speed (in RPM) depending on the voltage setpoint variation. The speed output of this block is converted to a coupling factor via the gain K_{rn} (in RPM^{-1}), and is finally expressed as a screen intensity value following multiplication with the detector gain K_{ni} (in kgv).

The output power stabilization loop comprises two gain blocks, K_{va} (in A/V) and K_{aw} (kW/A), which model the input current controller and laser discharge respectively. The detector gain, K_{wi}, performs the power to intensity conversion, which must later be reversed (after sampling) to obtain a power measure in kilowatts. Although this gain is expressed as a constant for simplicity, it is actually a function of the wand speed controller setpoint, U_w. Any modifications to the coupling percentage produce a change in total screen intensity, which the power controller perceives as a variation in output power. Consequently, modifications to the rotational speed controller setpoint must be accompanied by simultaneous changes to the gain factor K_{wi}, in order to compensate for any variation in screen intensity.

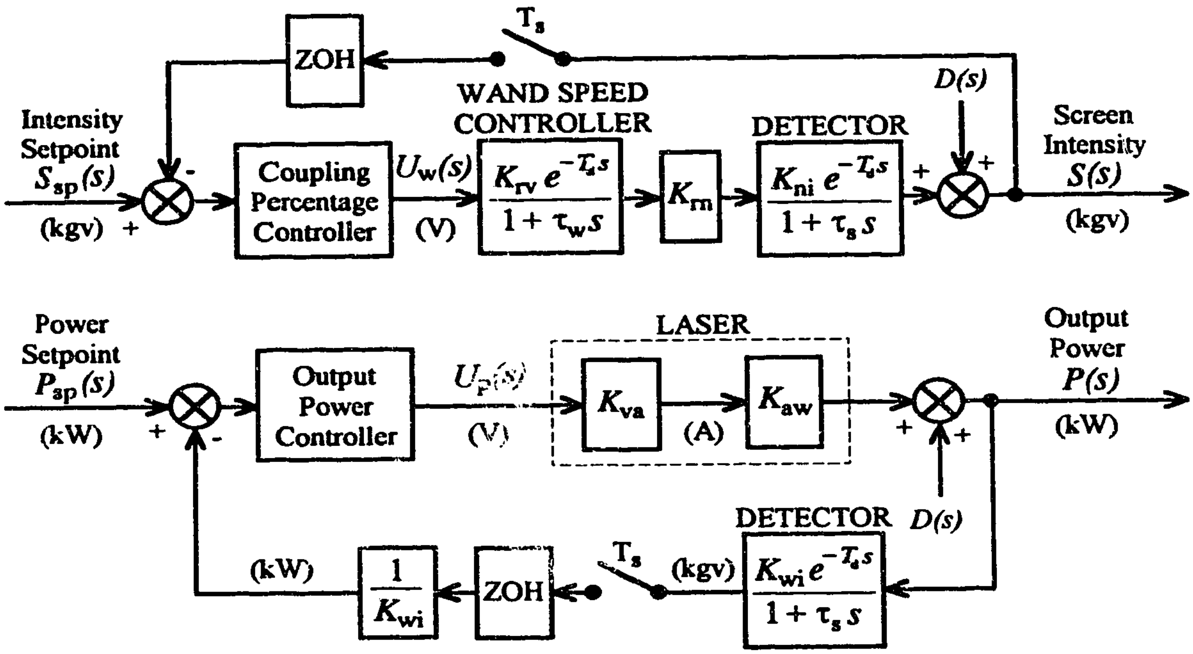


Fig. 6.3. Model of output power and variable coupling wand control systems.

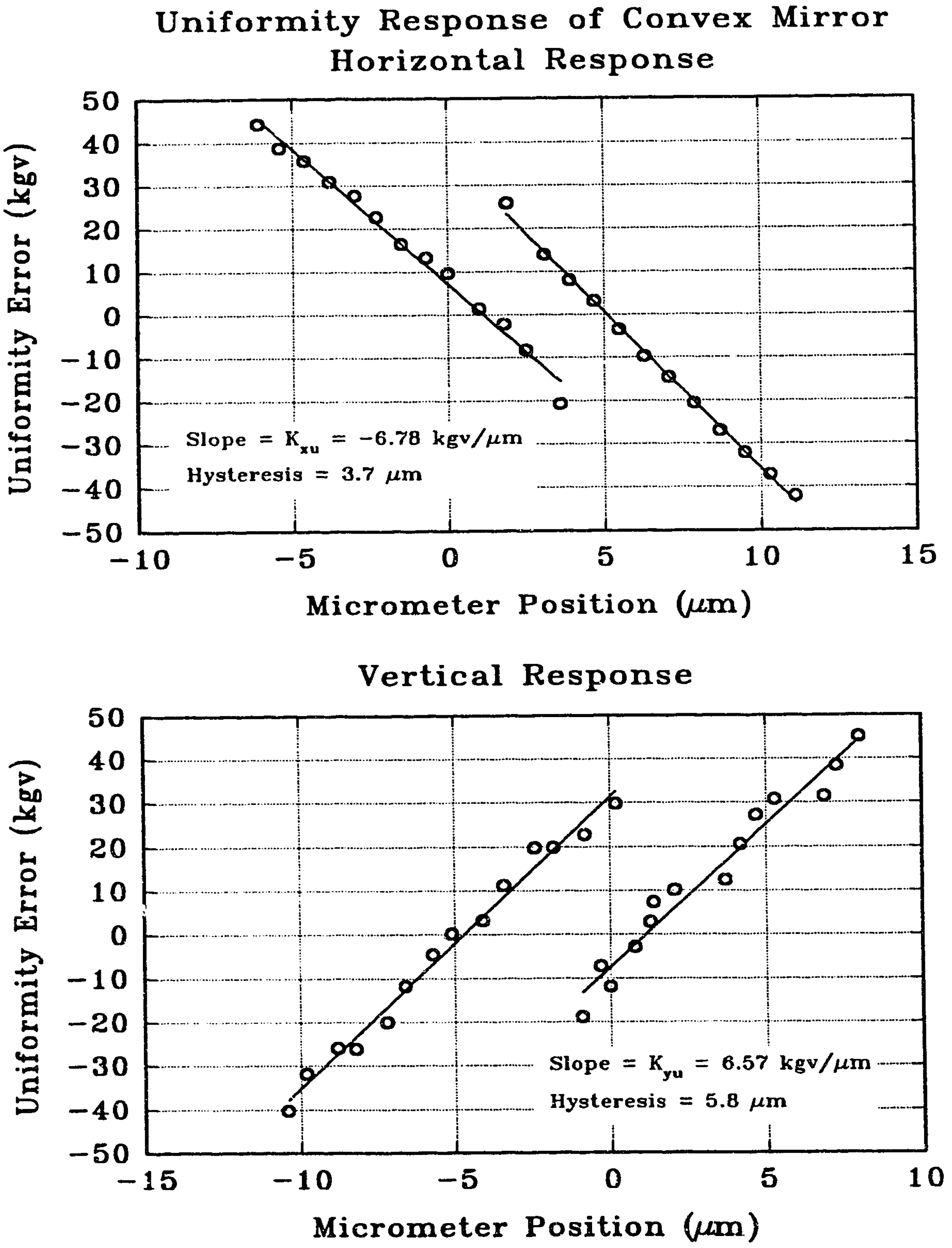
6.4 Model Parameter Estimation

The gains, delays, and time constants for all modelled processes were obtained in a similar manner to that described in Section 4.3 (for the PIE-3 system). The only significant improvement was the more accurate determination of the infrared detector's time constant and delay. This was accomplished with the use of the MATLAB matrix manipulation software's system identification functions,[74,75] and the necessary open loop step responses. Since the minimum execution time for one complete loop of the error computation and control algorithms was approximately 0.6 seconds (assuming exclusive access to the microprocessor), a constant sampling period, T_s, of 1 second was employed for all following performance responses. The extra 0.4 seconds between samples permitted concurrent execution of additional programs, such as the 2D and 3D beam profile software, without lengthening the control loop execution time over 1 second.

6.4.1 Uniformity Process Parameters

Uniformity control system parameters were ascertained by performing open loop step responses along both alignment axes. In order to obtain more accurate gain measurements and determine motormic hysteresis, the position of each resonator optic was varied in both X and Y directions and the corresponding uniformity error was recorded. Results are presented in the graphs of Fig. 6.4 (for the convex mirror) and Fig. 6.5 (for the concave optic). The two distinct plots of each graph represent data obtained by micrometer activation in the positive direction, followed by movement in the negative direction. The average horizontal separation between the two data sets designates the size of the hysteresis window ($2h$). Subsequent testing revealed slight overshoots when these exact hysteresis values were utilized by the compensation algorithm. Hence, all backlash values were reduced by 0.5 μm.

Contrary to the PIE-3 resonator system, both concave optic uniformity gains, K_{xu} and K_{yu}, are less than those for the convex mirror. This feature indicates that finer adjustments to overall uniformity is possible utilizing the concave optic, instead of the previously used convex unit. The controller design was such that the user selected mirror (which could be changed on-line) regulated the uniformity error, while the deselected optic controlled beam position. Consequently, the control strategies for the two mirrors were reversed to incorporate uniformity error stabilization with the concave optic and beam position control with the convex mirror. The controller effectively compensated for motormic backlash by storing the direction of the last control movement. If the next control action required motion in the opposite direction, the specified movement for the particular motormic was



Uniformity Response of Convex Mirror

Horizontal Response

Vertical Response

Fig. 6.4. Uniformity error sensitivity of convex mirror and associated hysteresis.

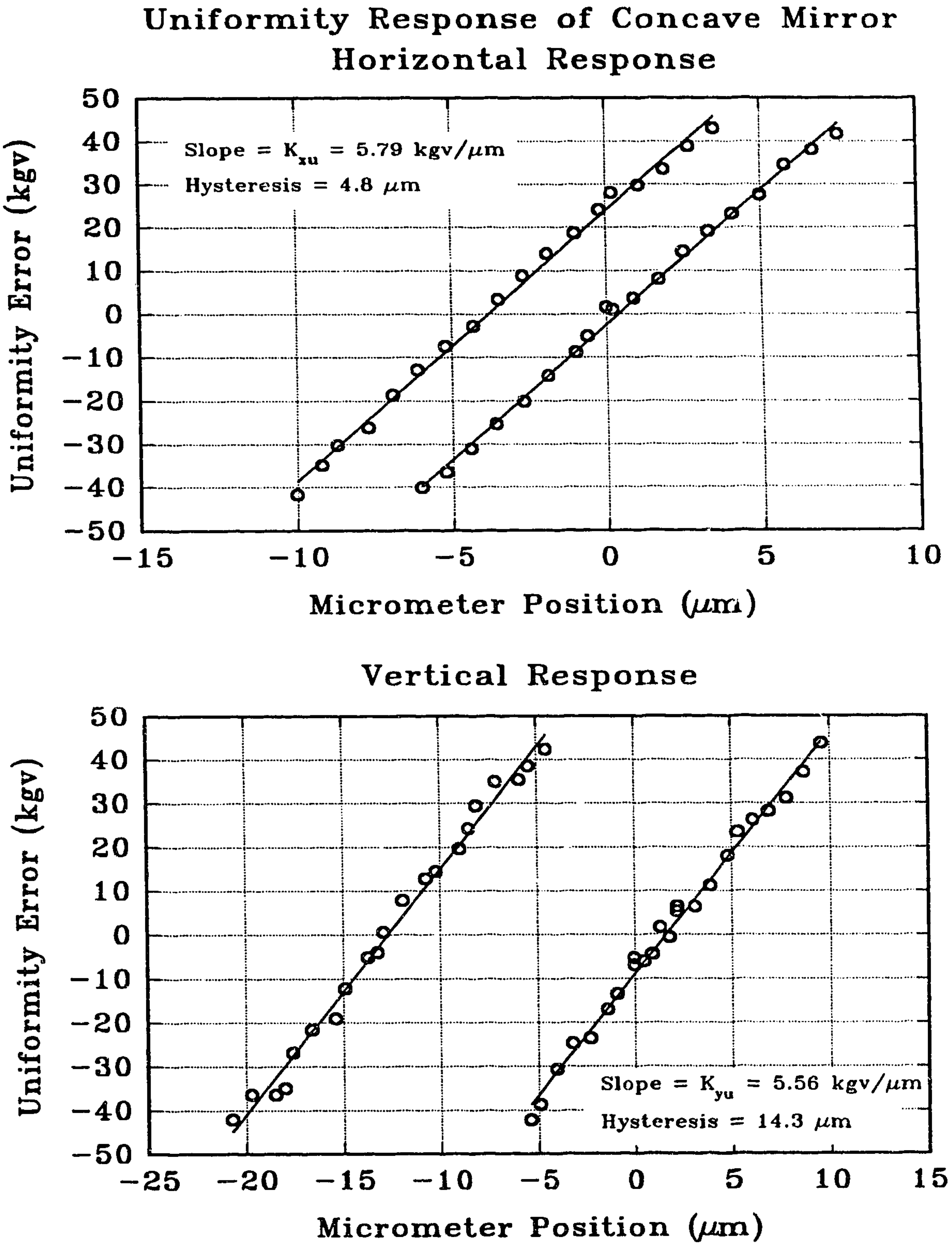


Fig. 6.5. Uniformity error sensitivity of concave mirror and associated hysteresis.

increased by its hysteresis value.

Since the concave mirror was now being employed for uniformity control, open loop step responses were performed on this optic to ascertain the imaging screen's time constant and delay. Data was acquired by first stabilizing beam uniformity to an optimal condition, then deactivating all control systems, and initiating the desired positive and negative step changes.

The discrete and continuous models of the horizontal uniformity process were obtained using the MATLAB program. First, the backlash was removed from the micrometer position data. Next, all data was offset such that they possessed a mean value of approximately zero, since the system identification routines yielded more accurate results when the model is expressed in terms of perturbation variables. From examination of the data, it was evident that the X uniformity error responds to setpoint changes after one or two sample periods, depending on hysteresis effects. Thus, the process delay T_d, was assumed to be about 1.5 seconds (an averaged value). Since this delay is not an exact multiple of the sampling period, the modified Z-transform approach was used to derive the following first order, discrete-time transfer function:

$$G(z^{-1}) = \frac{X_u(z^{-1})}{U_{xu}(z^{-1})} = \frac{(b_1 + b_2 z^{-1})z^{-d}}{1 - a_1 z^{-1}}, \tag{6.1}$$

where X_u is the horizontal uniformity error, U_{xu} is the horizontal motormic movement, and a_1, b_1, and b_2 are the model parameters to be estimated. The process delay, d (in number of sample periods), is defined as:

$$d = \text{integer portion of} \left(\frac{T_d}{T_s} \right), \tag{6.2}$$

and was calculated to be 1 sample interval. The input-output DARMA (Deterministic Auto-Regressive Moving-Average) form corresponding to the discrete model, expressed as:

$$X_u(n) = a_1 X_u(n-1) + b_1 U_{xu}(n-d) + b_2 U_{xu}(n-d-1), \tag{6.3}$$

was used in conjunction with the batch least squares technique to estimate model parameters. The least squares estimate of the parameter vector, $\mathbf{\Theta}$, is given by the following equation:

$$\mathbf{\Theta} = \left(\mathbf{\Phi}^{\mathrm{T}}\mathbf{\Phi}\right)^{-1}\mathbf{\Phi}^{\mathrm{T}}\mathbf{X}_u. \tag{6.4}$$

Since the derivation of this equation is well documented,[52] only the final result is presented. The parameter vector, $\mathbf{\Theta}$, input-output (I/O) matrix, $\mathbf{\Phi}$, and output vector, $\mathbf{X}_u$, are structured as follows:

$$\mathbf{\Theta} = \begin{bmatrix} a_1 \\ b_1 \\ b_2 \end{bmatrix}, \quad \mathbf{\Phi} = \begin{bmatrix} X_u(d+2) & U_{xu}(2) & U_{xu}(1) \\ X_u(d+3) & U_{xu}(3) & U_{xu}(2) \\ X_u(d+4) & U_{xu}(4) & U_{xu}(3) \\ \vdots & \vdots & \vdots \\ X_u(N-1) & U_{xu}(N-d-1) & U_{xu}(N-d-2) \end{bmatrix}, \quad \mathbf{X}_u = \begin{bmatrix} X_u(d+3) \\ X_u(d+4) \\ X_u(d+5) \\ \vdots \\ X_u(N) \end{bmatrix},$$

where N is the total number of samples. From the discrete-time parameters, a continuous domain model of the form:

$$G_{xu}(s) = \frac{X_u(s)}{U_{xu}(s)} = \frac{K_{xu}e^{-T_d s}}{1 + \tau_s s} \tag{6.5}$$

may be derived, where the process gain and time constant are defined as:

$$\tau_s = \frac{-T_s}{\ln(a_1)} \tag{6.6}$$

$$K_{xu} = \frac{b_1 + b_2}{1 - a_1}. \tag{6.7}$$

Equation 6.4 was solved using the MATLAB system identification functions to yield the following discrete model for the horizontal uniformity process:

$$G(z^{-1})=\frac{X_u(z^{-1})}{U_{xu}(z^{-1})}=\frac{(1.41+0.646z^{-1})z^{-d}}{1-0.658z^{-1}}. \tag{6.8}$$

The analogous continuous domain transfer function for a sampling period of 1 second was:

$$G_{xu}(s)=\frac{X_u(s)}{U_{xu}(s)}=\frac{5.99e^{-1.5s}}{1+2.39s}. \tag{6.9}$$

Since discrete models do not enable easy interpretation of their parameters (in terms of gains and time constants), all following models will be presented only in continuous form. The response of the model of equation 6.9 is depicted in Fig. 6.6,

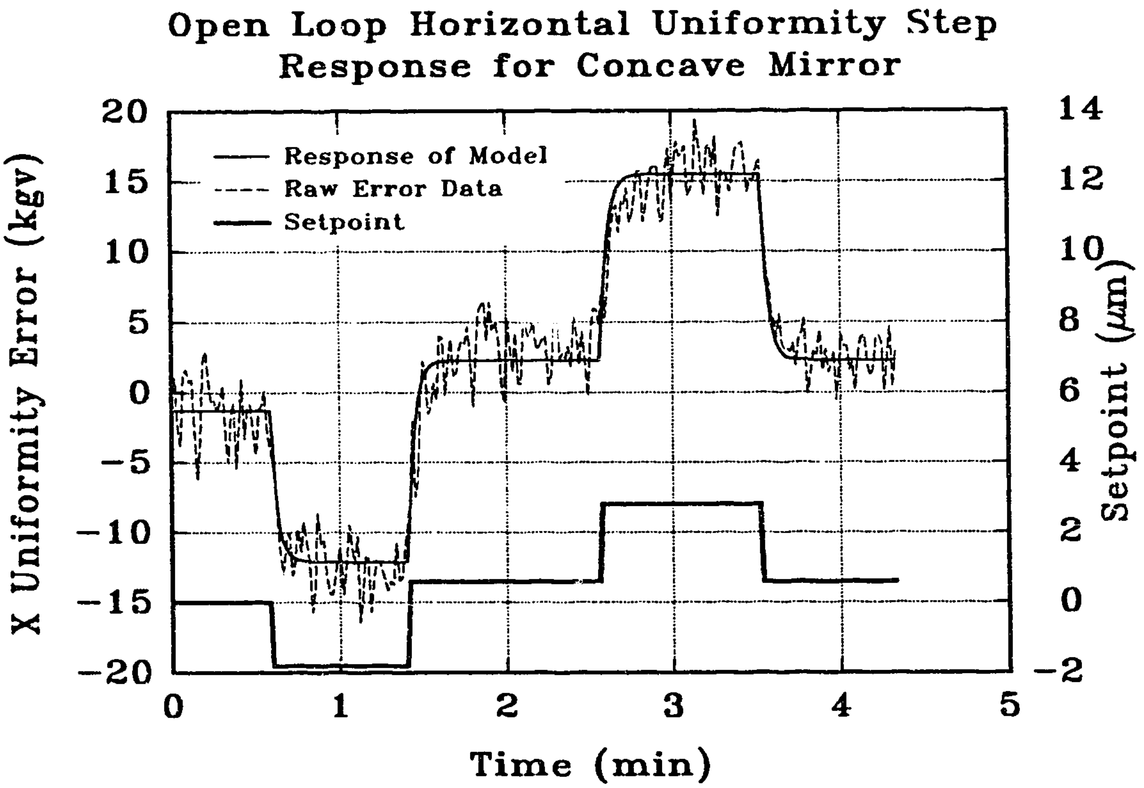


Fig. 6.6. Open loop step response to estimate horizontal uniformity parameters for concave mirror.

along with the corresponding raw data and setpoint change of about ±2 μm. The value of K_{xu} computed from the step response (5.99 kgv/μm) agrees with that from the sensitivity data of Fig. 6.5 (5.79 kgv/μm). Since the gains computed in Fig. 6.5 were obtained from an average of data points over a wider operating range, these values will be used for subsequent computations and analysis. The time constant for the imaging screen detector, as derived from the model, was 2.39 seconds.

A similar analysis was performed for the vertical uniformity process. The resulting continuous transfer function, computed from the open loop I/O data using the batch least squares technique, was:

$$G_{yu}(s) = \frac{Y_u(s)}{U_{yu}(s)} = \frac{5.13e^{-1.5s}}{1 + 2.69s}. \tag{6.10}$$

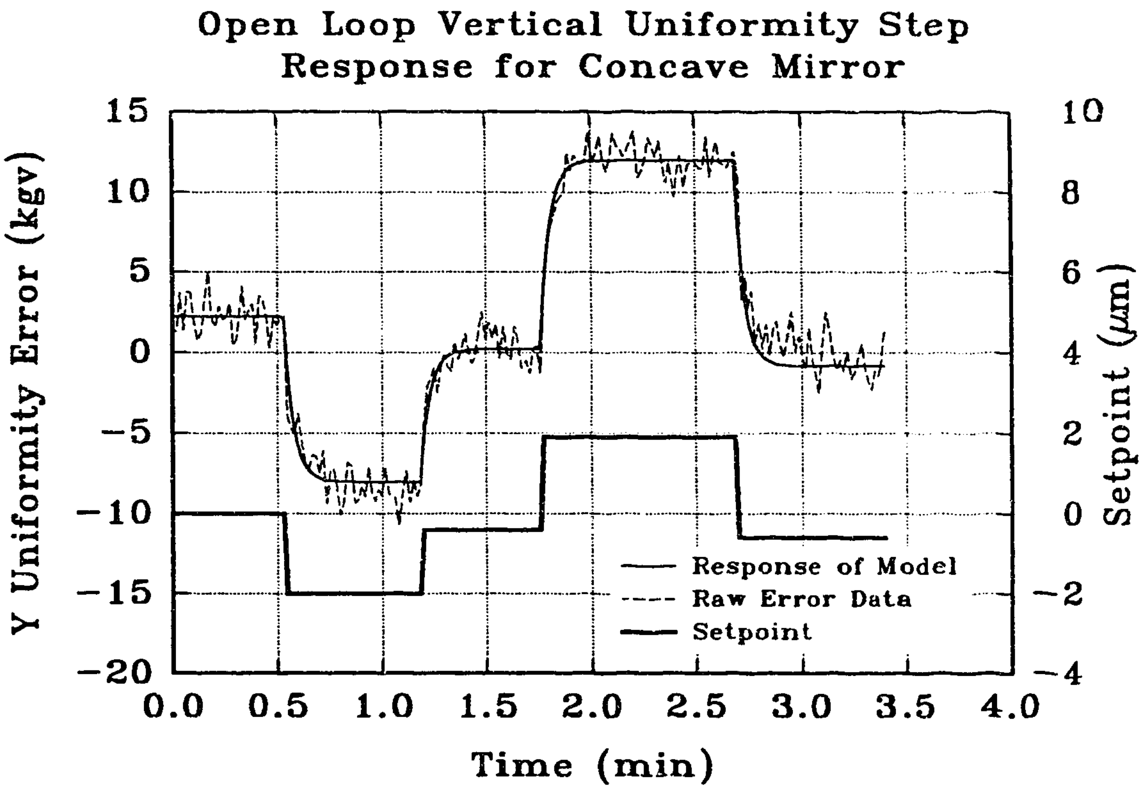


Fig. 6.7. Open loop step response to estimate vertical uniformity parameters for concave mirror.

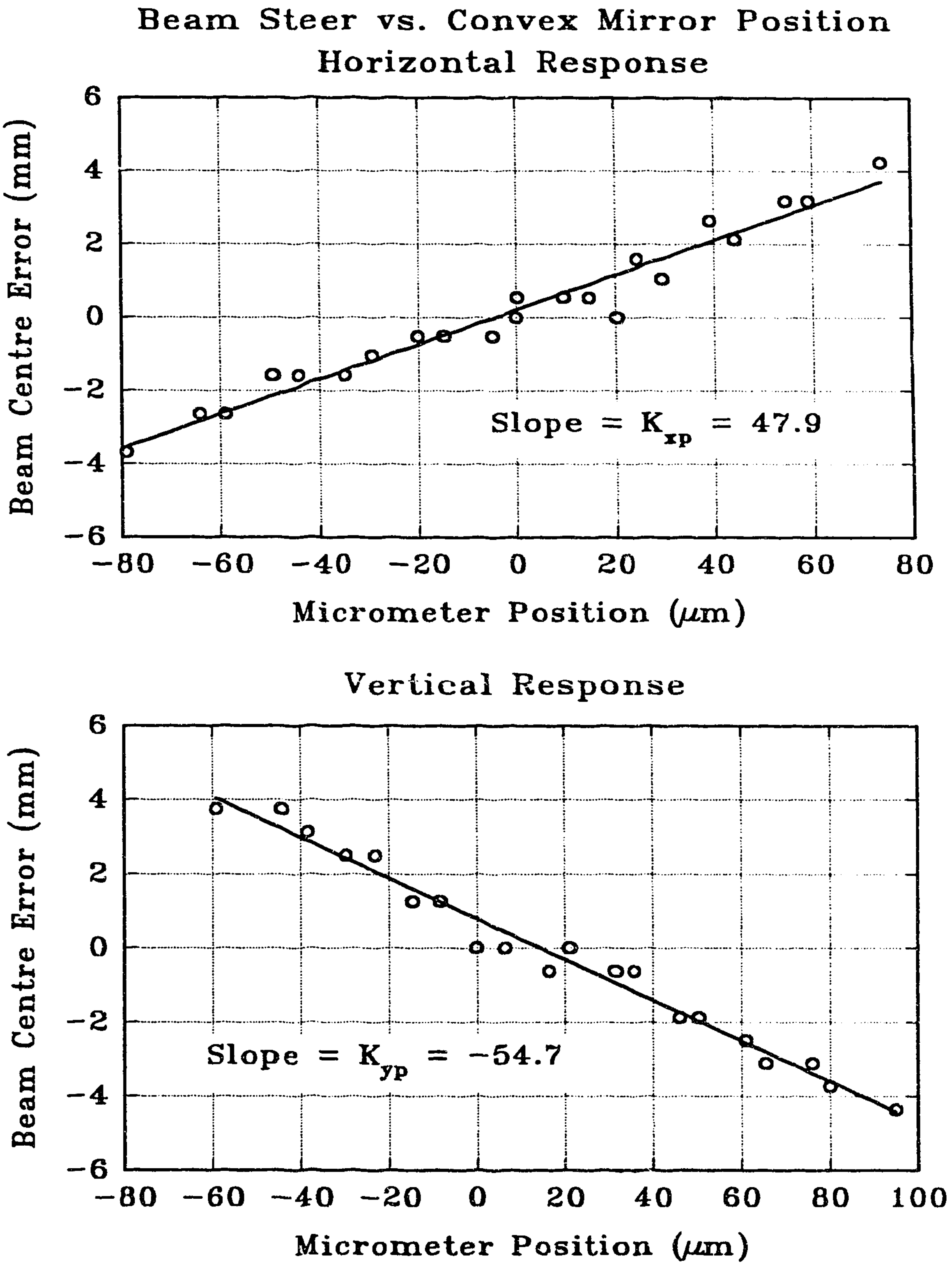
The simulated model response and the associated raw data is illustrated in Fig. 6.7. The process gain, $K_{yu,}$, of 5.13 kgv/µm is again comparable with that obtained from the steady-state data of Fig. 6.5, while the time constant of 2.69 seconds also agrees with that computed from horizontal response data. For the purpose of simplicity, an average value of 2.54 seconds will be assumed for both X and Y uniformity processes.

6.4.2 Beam Steer Process Parameters

Certain parameters of the beam position control system, such as the motormic hysteresis, imaging screen's time constant, and process delay, have already been determined from analysis of the uniformity system responses. Thus, the only model parameters remaining to be estimated are the process gains, K_{xp} and K_{yp}, and the millimeter to pixel conversions, K_{mpx} and K_{mpy}.

Process gains were ascertained in a similar manner to those for the PIE-3 system (described in Section 5.2.1), except for the reversal of the functions of the two resonator optics. The position of the convex mirror was varied along both alignment axes, while the concave optic maintained beam uniformity. Following each convex movement, a set of positional data was stored once beam uniformity was restored for four consecutive sample periods. This eliminated any phase lag from the moving average filter, and provided an accurate beam centre measurement. The sensitivity data obtained is depicted in Fig. 6.8. As evident, beam centre is linearly proportional to micrometer position, with a noise margin of about ±0.6 mm (±1 pixel).

The conversion gains, K_{mpx} and K_{mpy}, were calculated from the computer monitor dimensions, camera magnification factor, and pixel resolution of the digitizer. The only difference for the PIE-4 system was a smaller magnification factor of 1.58 (due to the larger beam cross section), which resulted in horizontal and


Beam Steer vs. Convex Mirror Position

Horizontal Response

Vertical Response

Fig. 6.8. Sensitivity of beam centre to changes in convex mirror position.

vertical conversion gains of 1.89 pix/mm and 1.60 pix/mm respectively. Based on these values, and the distance from the detection screen to the midpoint of the resonator (about 9.9 m), the minimum resolvable angular deviation of beam propagation direction is approximately 60 μrad.

6.4.3 Variable Sampling Process Parameters

Much of the data necessary for estimation of the variable coupling percentage system has been presented in Section 2.2.2. The open loop response of the wand speed controller, previously illustrated in Fig. 2.10, revealed a controller time constant, τ_w, of 0.141 seconds and a process delay of 1 second. Gain parameters, K_{rv} and K_{m}, were computed to be -38.4 RPM/V and -9.41 × 10^{-5} respectively, from the slopes of Fig. 2.9 and Fig. 2.8. These gains may vary due to increases in AC motor temperature and changes in frictional forces between wand components. However, the speed controller bases its output on the width of the wand's feedback pulse (i.e. amount of reflective surface which is exposed). Thus, the product of the two gains remains constant, and any individual variations do not affect system performance.

The sensor gain, K_{ni}, was obtained by varying the sampling percentage and recording the corresponding screen intensity data, while maintaining optimal beam uniformity with the concave mirror. Measurements of coupling percentage represented an average of 25 data values, in order to reduce excessive contamination with noise. This data, depicted in Fig. 6.9, indicates a gain, K_{ni}, of 2.28 × 10^{-4} kgv. The time constant of the model, τ_s, was determined from a least squares analysis of an open loop step response. The simulated response of this model, given by:

$$G_w(s) = \frac{S(s)}{U_w(s)} = \frac{89.8e^{-1.5s}}{1+5.23s}, \tag{6.11}$$

and the actual open loop step data, are presented in Fig. 6.10. The wand speed

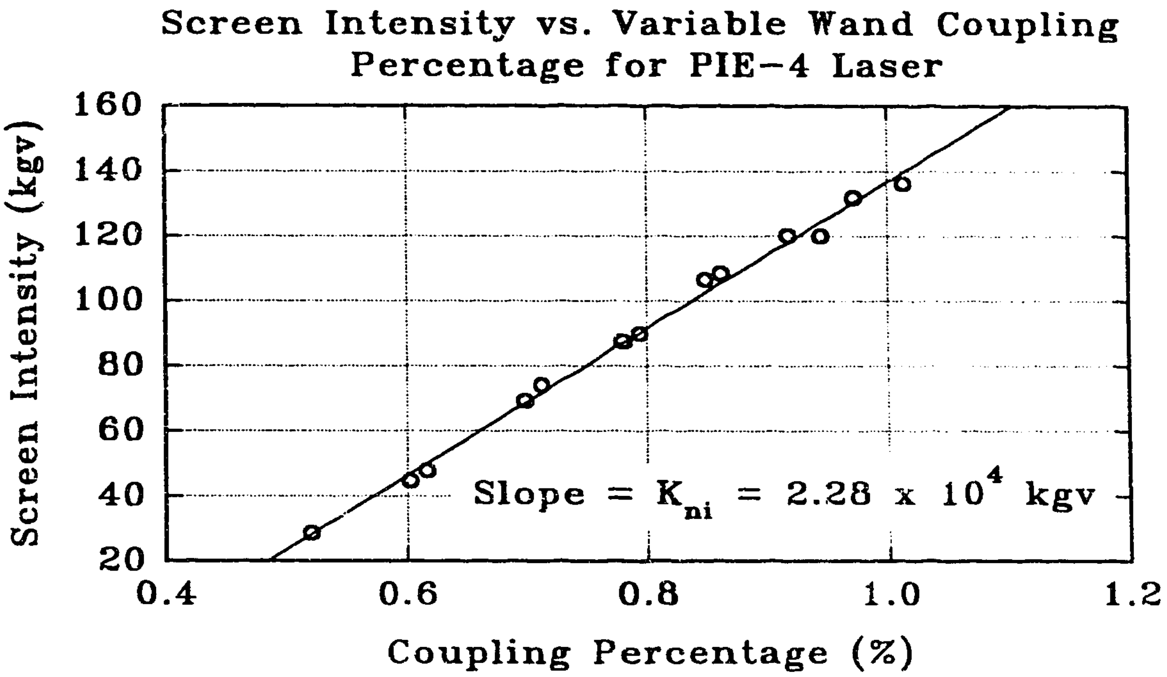


Fig. 6.9. Sensitivity of total screen intensity to changes in sampling percentage.

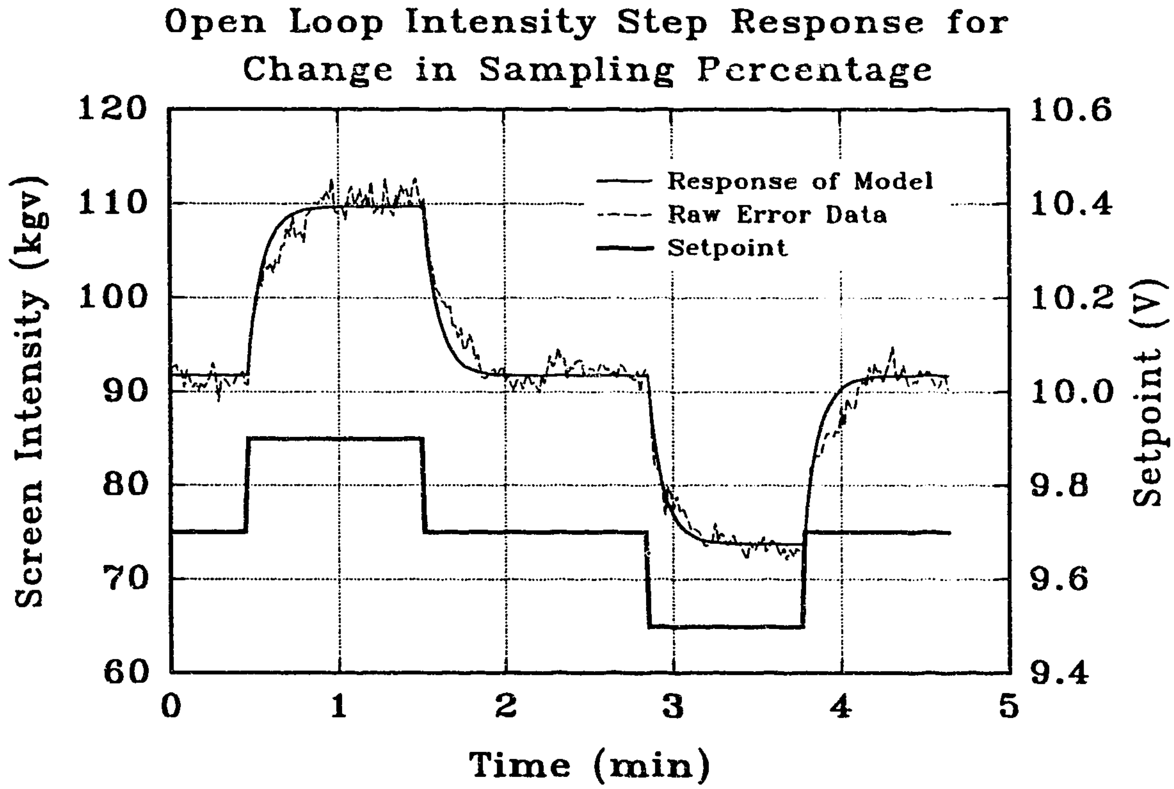


Fig. 6.10. Open loop screen intensity response to a wand controller setpoint change.

controller setpoint change of ±0.2 V corresponded to a sampling percentage variation of approximately ±0.075%. Overall model gain of 89.8 kgv/V compares well with the value of 82.3 kgv/V, which is the product of the three individual process gains, K_{rv} (-38.4 RPM/V), K_{rn} (-9.41 × 10^{-5}), and K_{ni} (2.28 × 10^{-4} kgv). The time constant of 5.23 seconds is considerably longer than the previously computed values, due to the relatively large global change in screen intensity (about 20 kgv).

6.4.4 Output Power Process Parameters

As a result of its SCR controlled power supply, the output power process was modelled as two simple gain blocks. The value of the first gain parameter, K_{va}, was obtained by varying the discharge current setpoint and observing subsequent changes to the current level. The resulting linear response, with a slope of 0.534 A/V, is presented in Fig. 6.11. The second gain value, K_{aw}, was determined in a similar manner by varying the input current and monitoring corresponding output power.

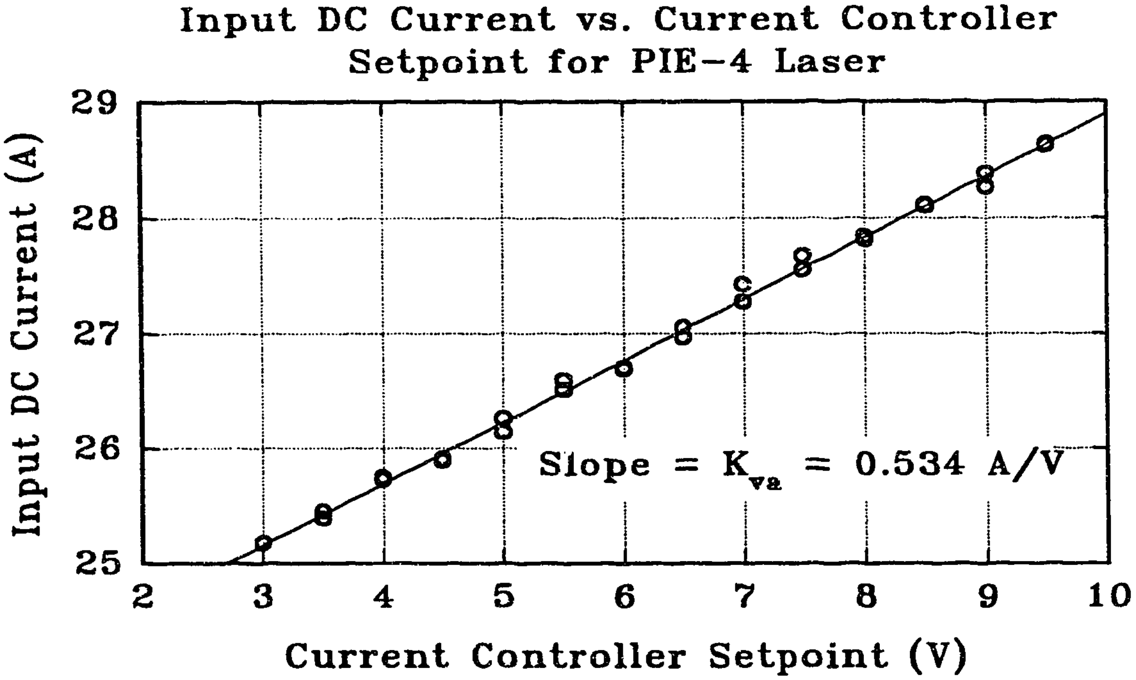


Fig. 6.11. Relation between input DC current and current controller setpoint.

The recorded data, depicted in Fig. 6.12, indicates a relatively linear relationship, with a slope of 0.696 kW/A. As evident, the response deviates slightly from its linear behavior for power levels above 9 kW. Since this research limits testing to output powers under 10 kW, the constant gain approximation is valid. However, if operation at higher power levels is desired, the software based controller can be easily modified to accommodate a non-linear relationship.

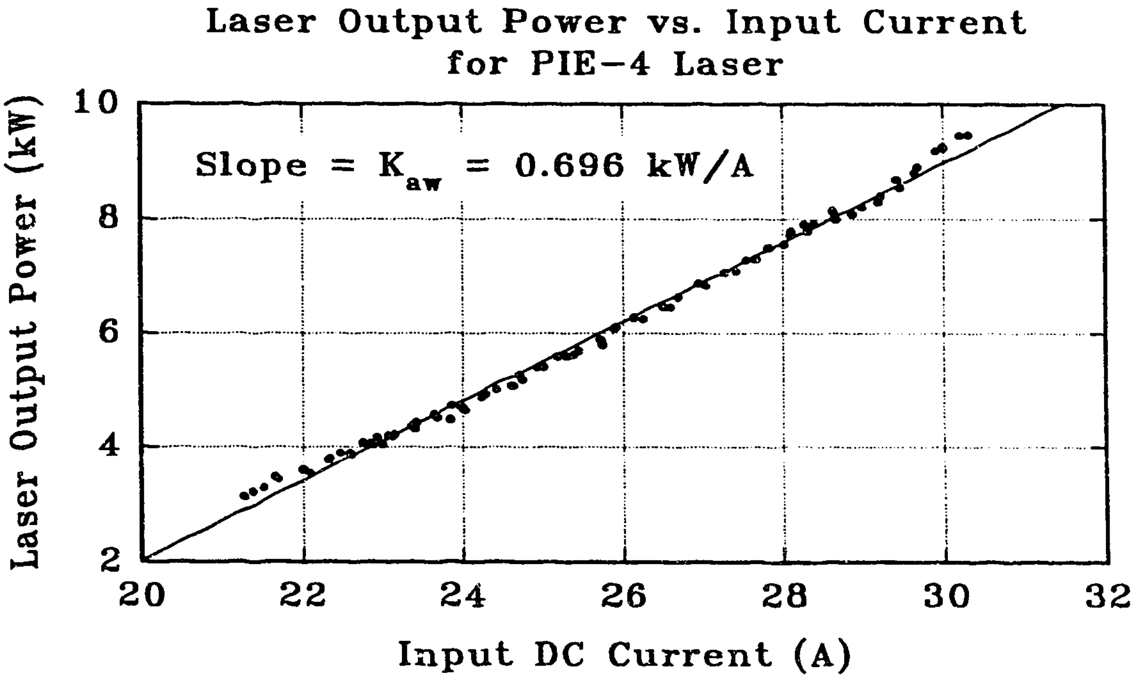


Fig. 6.12. Dependence of output power on input DC current.

Due to the introduction of the variable coupling wand, the gain parameter for the infrared detector, $K_{wi,}$, could not be estimated as a constant. Instead, it must be modelled as a function of the wand speed controller setpoint, U_w. Consequently, a series of data sets relating total screen intensity to output power, for varying wand controller setpoints, was obtained. This data, illustrated in Fig. 6.13, indicates an increase in detector gain (slope of each plot) as the speed controller setpoint decreases

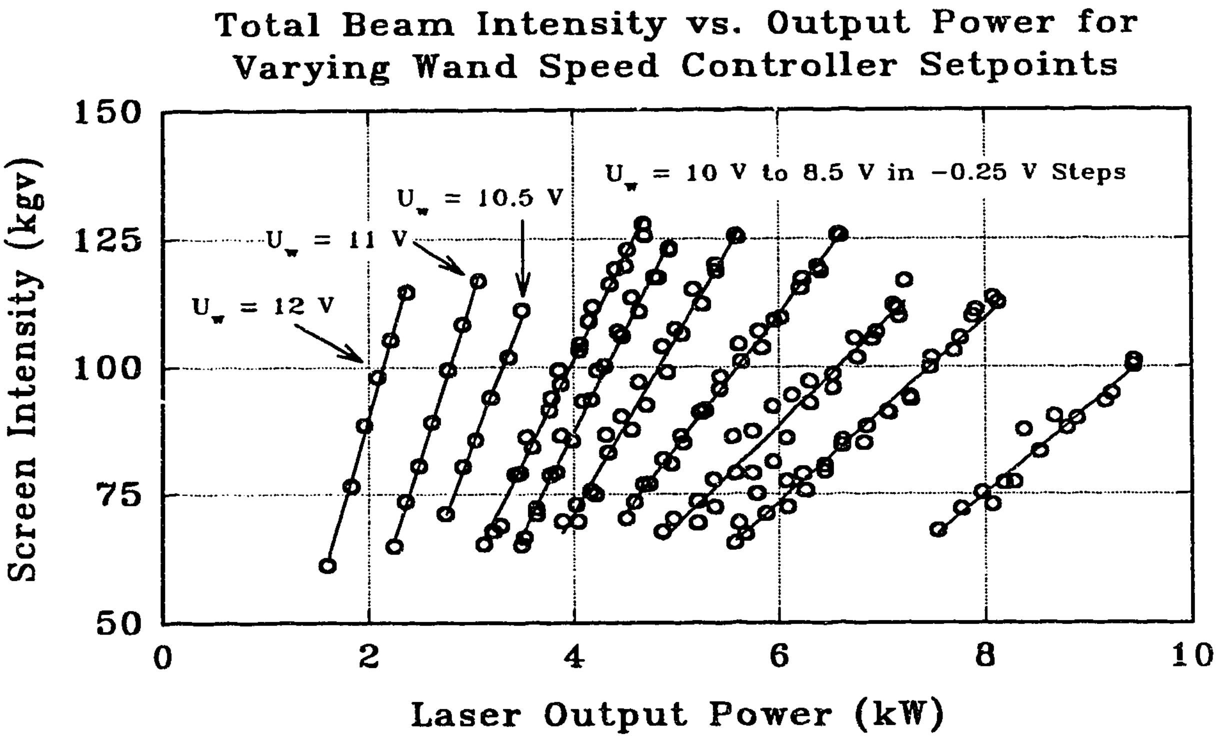


Fig. 6.13. Intensity-power relation for varying wand speed controller setpoint.

(i.e. coupling percentage increases). This result is expected since the optimal range of power with which the imaging screen may be irradiated remains constant (25 to 35 W). Thus, an increase in sampling percentage must be accompanied by a reduction in the maximum possible variation in output power of the laser, in order to maintain imaging screen power within the 25 to 35 W limit. Each of the gain values calculated from Fig. 6.13 is plotted in Fig. 6.14 as a function of the speed controller setpoint. Provided that the system operation is restricted to wand setpoints between 8.5 and 11 V (which corresponds to an acceptable output power range of about 2 to 10 kW), the resulting relation may be linearly approximated as:

$$K_{wi}(U_w) = 20.97U_w - 166.4, \tag{6.12}$$

where K_{wi} is specified in gv/W. If operation at power levels outside these limits is necessary, the response in Fig. 6.14 may be approximated as a cubic (indicated by

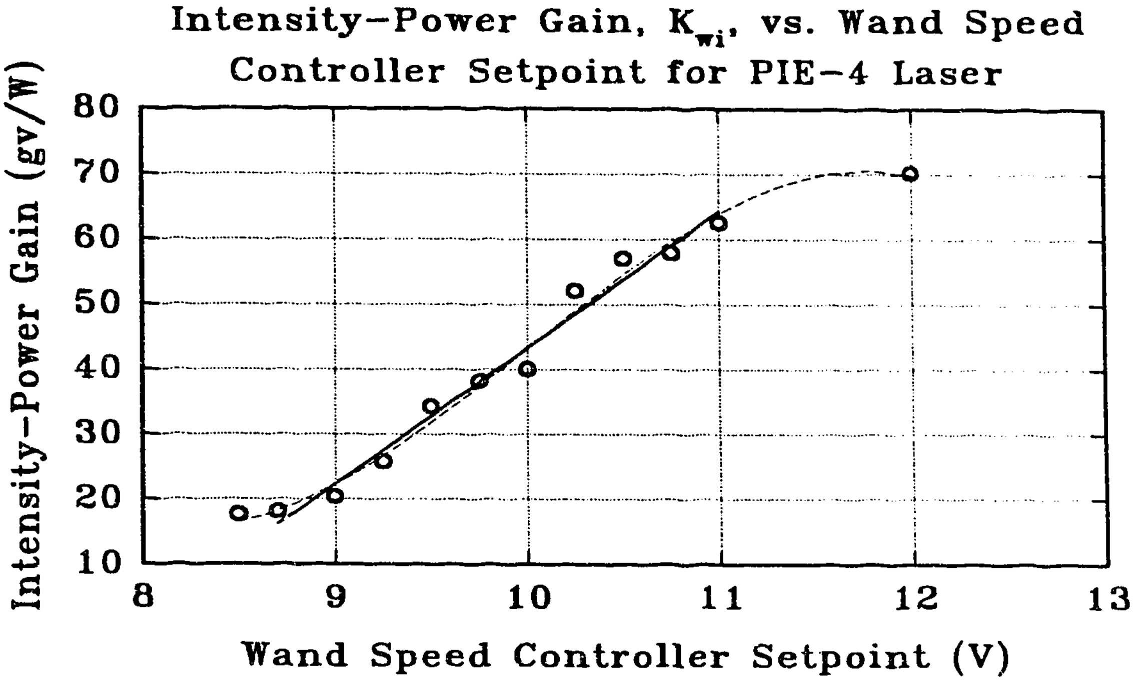


Fig. 6.14. Imaging screen gain as a function of the wand speed controller setpoint.

the dashed line) and expressed as:

$$K_{wi}(U_w) = -2.480U_w^3 + 74.34U_w^2 - 719.4U_w + 2283 \tag{6.13}$$

The offset (zero order term) for each of these equations may be recalibrated on-line if process parameters have changed significantly.

Since modifications to output power necessitate a global change to total screen intensity, rather than the local variations caused by uniformity errors, an open loop step response was performed to determine the associated infrared sensor time constant and delay. Analysis of data from a ±0.5 kW step change (current controller setpoint change of ±1.5 V) revealed a process delay between 1 and 2 sample periods. Thus, an average value of 1.5 seconds was assumed. A least squares estimate of the I/O data (similar to that described in Section 6.4.1) yielded the following first order continuous model:

$$G_p(s) = \frac{P(s)}{U_p(s)} = \frac{0.376e^{-1.5s}}{1 + 3.04s}. \qquad (6.14)$$

This model response and the associated raw data is illustrated in Fig. 6.15. Output power was monitored directly with the calorimetric power meter and obtained from screen intensity data. The process gain of 0.376 kW/V agrees with the expected value of 0.373 kW/V, derived from the product of K_{va} and K_{aw}. Since the data was obtained for a wand controller setpoint of 9.2 V, a conversion gain, K_{wi}, of 26.5 gv/kW was evaluated from equation 6.12, and utilized to determine output power from total intensity measurements. The detector time constant of 3.04 seconds is approximately 15% greater than that for the uniformity process, indicating that the

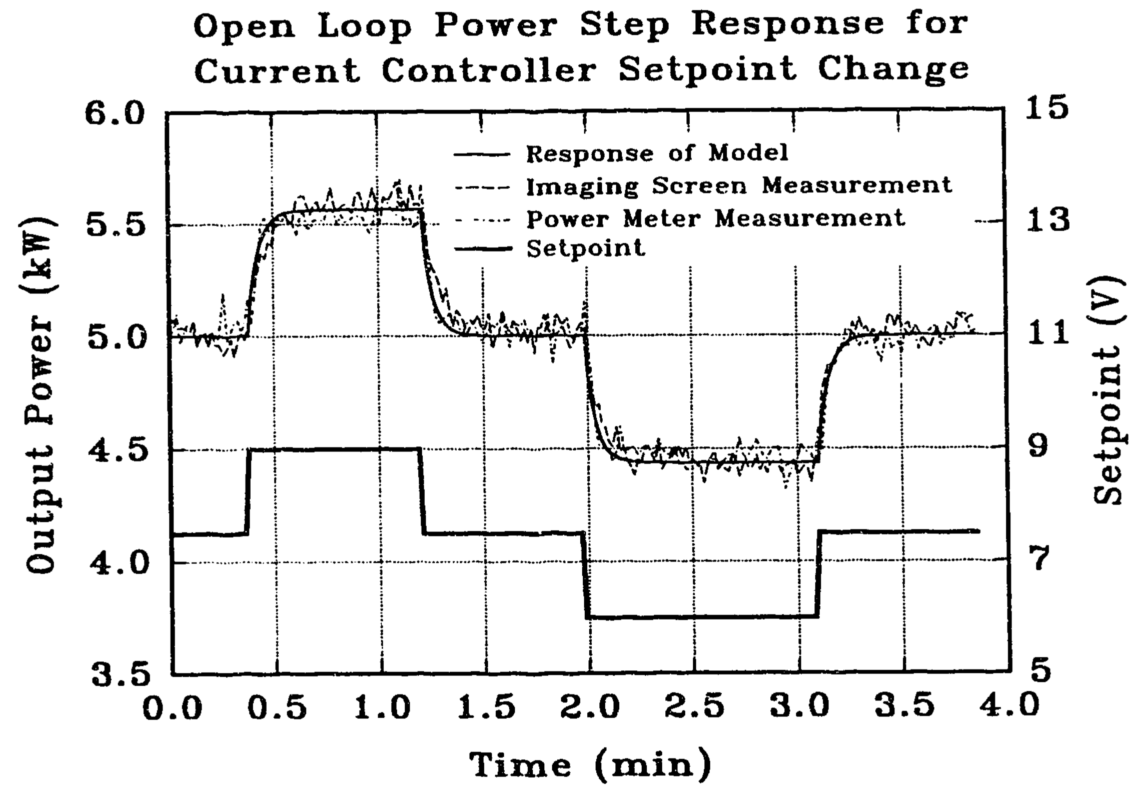


Fig. 6.15. Open loop response of output power to a change in current controller setpoint.

imaging screen requires a longer time period to respond to global intensity changes. Tables 6.1 and 6.2 summarize all estimated system model parameters.

Model Parameter	Value
General System Parameters	
T_s for all feedback loops	1.0 s
T_d for imaging screen	1.5 s
Uniformity Process Parameters	
K_{xsu} and K_{ysu} for concave optic	5 to 10 μm/s
K_{xu} for concave optic	5.79 kgv/μm
K_{yu} for concave optic	5.56 kgv/μm
K_{xu} for convex optic	-6.78 kgv/μm
K_{yu} for convex optic	6.57 kgv/μm
τ_s for imaging screen	2.54 s
h (hysteresis) for horizontal concave motormic	4.3 μm
h (hysteresis) for vertical concave motormic	13.8 μm
h (hysteresis) for horizontal convex motormic	3.2 μm
h (hysteresis) for vertical convex motormic	5.3 μm
Beam Steer Process Parameters	
K_{xsp} and K_{ysp} for convex optic	5 to 10 μm/s
K_{xp} process gain	47.9
K_{yp} process gain	-54.7
K_{mpx} conversion gain	1.89 pix/mm
K_{mpy} conversion gain	1.60 pix/mm
τ_s for imaging screen	2.54 s

Table 6.1. Estimated parameters for beam uniformity and position control systems.

Model Parameter	Value
Variable Sampling Process Parameters	
K_{rv} for wand speed controller	-38.4 RPM/V
K_{rn} for variable coupling wand	-9.41×10^{-5} RPM^{-1}
K_{ni} for imaging screen	2.28×10^{-4} kgv
T_d for wand speed controller	~1 s
τ_w for wand speed controller	0.141 s
τ_s for imaging screen	5.23 s
Output Power Process Parameters	
K_{va} for input current controller	0.534 A/s
K_{aw} for laser	0.696 kW/A
K_{wi} for imaging screen detector	$f(U_w)$ gv/W
τ_s for imaging screen	3.04 s

Table 6.2. Estimated parameters for output power and variable sampling systems.

6.5 Controller Operation and Design

Overall controller operation has been discussed in Section 3.1.4. The general flowchart of the controller algorithm, presented in Fig. 3.5, details all aspects of control sequences and the associated conditional logic. Thus, this section will focus mainly on the tuning of all control systems and any enhancements not previously described.

6.5.1 Beam Uniformity Controller

The uniformity controller used in the PIE-4 system is identical to that for PIE-3 laser, described in Section 4.4 and Fig. 4.10. The only improvement is in the design of the motormic position tracker, which has been implemented in hardware. A typical control sequence involves computation of the required motormic movements, by application of the controller gains (K_{cux} for the horizontal error and K_{cuy} for the vertical) to the uniformity errors. The 16-bit counter is set to the larger of the two movements, and the appropriate concave mirror motormic is activated. Upon commencement of the next sample period, required movements are again calculated. However, before execution of the control action, the status of the 16-bit counter is examined. If the previous movement has not yet completed, the counter is read, data values are updated, and no uniformity or position control action is performed. Errors in output power or screen intensity may still be corrected, provided the necessary conditions are satisfied. At motormic speed settings of 5 to 10 μm/s, average uniformity control actions execute in the allotted 1 second sampling interval. The error window was set to ±0.5 μm, which accommodated the process noise margin.

Since controller design was modified to incorporate a constant sampling rate, the necessary delays implemented in the PIE-3 system (for the imaging screen to accurately respond) are not present. Thus, the optimal controller gains (i.e. the reciprocal of the process gains) could not be utilized due to ensuing stable oscillations. Although general operation is that of a relay-type system, controller design actually regulates motormic movements in proportion to corresponding error signals. Consequently, proportional controller tuning techniques could be used to determine initial gain values. A more recent method, based on the integral of time absolute error (ITAE) criterion,[76] given by:

$$K_{\mathrm{c}} = \frac{0.490}{K}\left(\frac{T_{\mathrm{d}}}{\tau_{\mathrm{s}}}\right)^{-1.084}, \tag{6.15}$$

for a simple proportional controller, was used. This yielded a horizontal uniformity gain, K_{cxu}, of 0.135 μm/kgv and a vertical gain, K_{cyu}, of 0.180 μm/kgv. Closed loop performance testing indicated slight oscillations for uniformity step responses. Thus, both X and Y gains were decreased to 0.120 and 0.140 μm/kgv, which resulted in acceptable transient responses with no overshoot.

6.5.2 Beam Position Controller

The beam position control system operates as described in Section 5.3. Upon existence of horizontal and vertical position errors, the convex optic motormic corresponding to the larger of the two is activated, providing the uniformity error is within its threshold window. The additional requirement of optimal beam uniformity for four consecutive sample intervals, to compensate for filter phase lag, was abolished due to the faster sampling rate.

Optimal controller gains, K_{cxp} and K_{cyp}, required to eliminate position errors in a single control action, are given by:

$$K_{cxp} = \frac{1}{K_{xp}K_{mpx}} = \frac{(10^3\,\mu\text{m / mm})}{(47.9)(1.89\text{ pix / mm})} = 11.0\ \mu\text{m / pix} \qquad (6.16)$$

$$K_{cyp} = \frac{1}{K_{yp}K_{mpy}} = \frac{(10^3\,\mu\text{m / mm})}{(-54.7)(1.60\text{ pix / mm})} = -11.4\ \mu\text{m / pix.} \qquad (6.17)$$

However, using these settings introduced large disturbances to beam uniformity, since the minimum possible movement (for an error of 1 pixel) is about 11 μm. Further investigation indicated that maximum convex movements of 5.5 μm produced minimum disturbances to beam uniformity, enabling the concave optic to compensate in a few sample periods. In addition, beam symmetry was sufficiently preserved to prevent erroneous measurement of beam diameters and centres. This aspect, combined with the faster sampling rate, significantly reduced phase lag effects of the filters.

6.5.3 Sampling Percentage Controller

The variable coupling controller acted independently of all other systems to maintain total screen intensity within an optimal range of 70 to 110 kgv. Its relatively simple design was based on the overall system gain of approximately 85 kgv/V. The controller monitored total beam intensity until a value outside the error window of 70 to 110 kgv was obtained. At this point, a wand speed controller setpoint change was issued to restore screen intensity to the centre of the operating range. Following each wand control action, a delay of 5 sample periods was observed before any further changes to the speed controller setpoint were executed. This precaution permitted sufficient time for the imaging screen to respond, due to the relatively large process time constant of 5.23 seconds. The speed controller setpoint was modified in steps of 0.2V which corresponded to an intensity change of approximately 20 kgv. Consequently, if either extreme of the error window was reached, operation was returned to the optimal intensity value of 90 kgv with a single control action.

6.5.4 Output Power Controller

The design of the output power control system was altered from the previous relay-type strategy to a standard PI controller. Integral action was introduced to improve steady-state performance. The general form of a continuous PI controller is as follows:

$$u(t) = K_c \left[\varepsilon(t) + \frac{1}{T_i} \int \varepsilon(t)dt \right], \tag{6.18}$$

where K_c is the controller gain, T_i is the integral or reset time, $u(t)$ is the controller output, and $\varepsilon(t)$ is the error signal. The positional form of the discrete PI algorithm can be derived from equation 6.18 by rectangular approximation of the integral term.

This result is given by:

$$u(n) = K_c\left[\varepsilon(n) + \frac{T_s}{T_i}\sum_{i=0}^{n}\varepsilon(i)\right]. \tag{6.19}$$

Since a differential algorithm is more suitable for computer implementation, the velocity form of equation 6.19 may be obtained by introducing the following change of variable:

$$\Delta u(n) = u(n) - u(n-1). \tag{6.20}$$

The subsequent incremental PI algorithm, given by:[77]

$$\Delta u(n) = K_c\left[\varepsilon(n) - \varepsilon(n-1) + \frac{T_s}{T_i}\varepsilon(n)\right], \tag{6.21}$$

was implemented in software to form the basis of the output power controller.

Initial power controller parameters were calculated using the ITAE method as follows:

$$K_{cp} = \frac{0.859}{K}\left(\frac{T_d}{\tau_s}\right)^{-0.977} \tag{6.22}$$

$$T_i = 1.484\tau_s\left(\frac{T_d}{\tau_s}\right)^{0.680}. \tag{6.23}$$

Based on these criteria, a power controller gain, K_{cp}, of 4.59 V/kW and an integral time, T_i, of 2.79 seconds were used to initially test performance. Analysis of closed loop data revealed fast transient responses with significant overshoots and oscillations. However, further tuning produced acceptable results following a decrease in the proportional gain, K_{cp}, to 1.0 V/kW and an increase in T_i to 5.0 seconds. A summary of all controller parameters is presented in Table 6.3.

Controller Parameter	Value
K_{cxu} for horizontal uniformity controller	0.120 µm/kgv
K_{cyu} for vertical uniformity controller	0.140 µm/kgv
K_{cxp} for horizontal position controller	5.5 µm steps
K_{cxp} for vertical position controller	5.5 µm steps
K_{cw} for variable sampling controller	0.2 V steps
K_{cp} for output power controller	1.0 V/kW
T_{i} for output power controller	5.0 seconds
w (error window) for uniformity controller	±0.5 µm
w (error window) for position controller	±1 pixel
w (error window) for variable sampling controller	±20 kgv
w (error window) for power controller deactivation	±2.0 µm
w (error window) for position controller deactivation	±0.5 µm

Table 6.3. Controller parameters for all feedback systems.

6.6 System Performance

Closed loop performance of all control systems was evaluated from a transient and steady-state perspective. In addition, the sensitivity of output signals to external disturbances was examined, along with overall improvement in laser operation. Controller parameter values listed in Table 6.3 were utilized for all following responses, which were obtained at an average output power level of 5 kW.

6.6.1 Uniformity Controller Performance

Closed loop step responses of the horizontal and vertical uniformity controllers are depicted in Fig. 6.16. Following initial optimization of beam

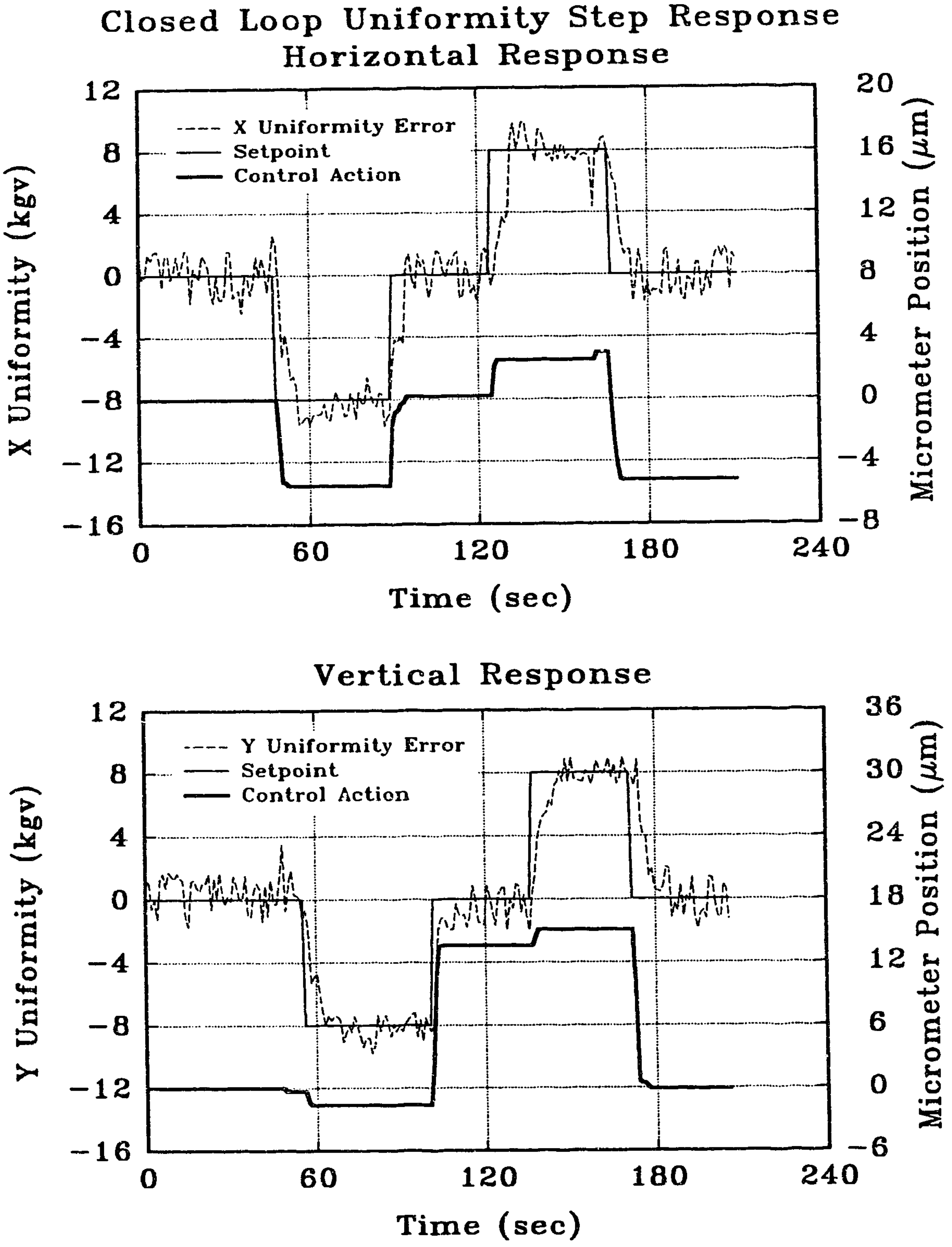


Fig. 6.16. Transient closed loop uniformity response for small setpoint changes.

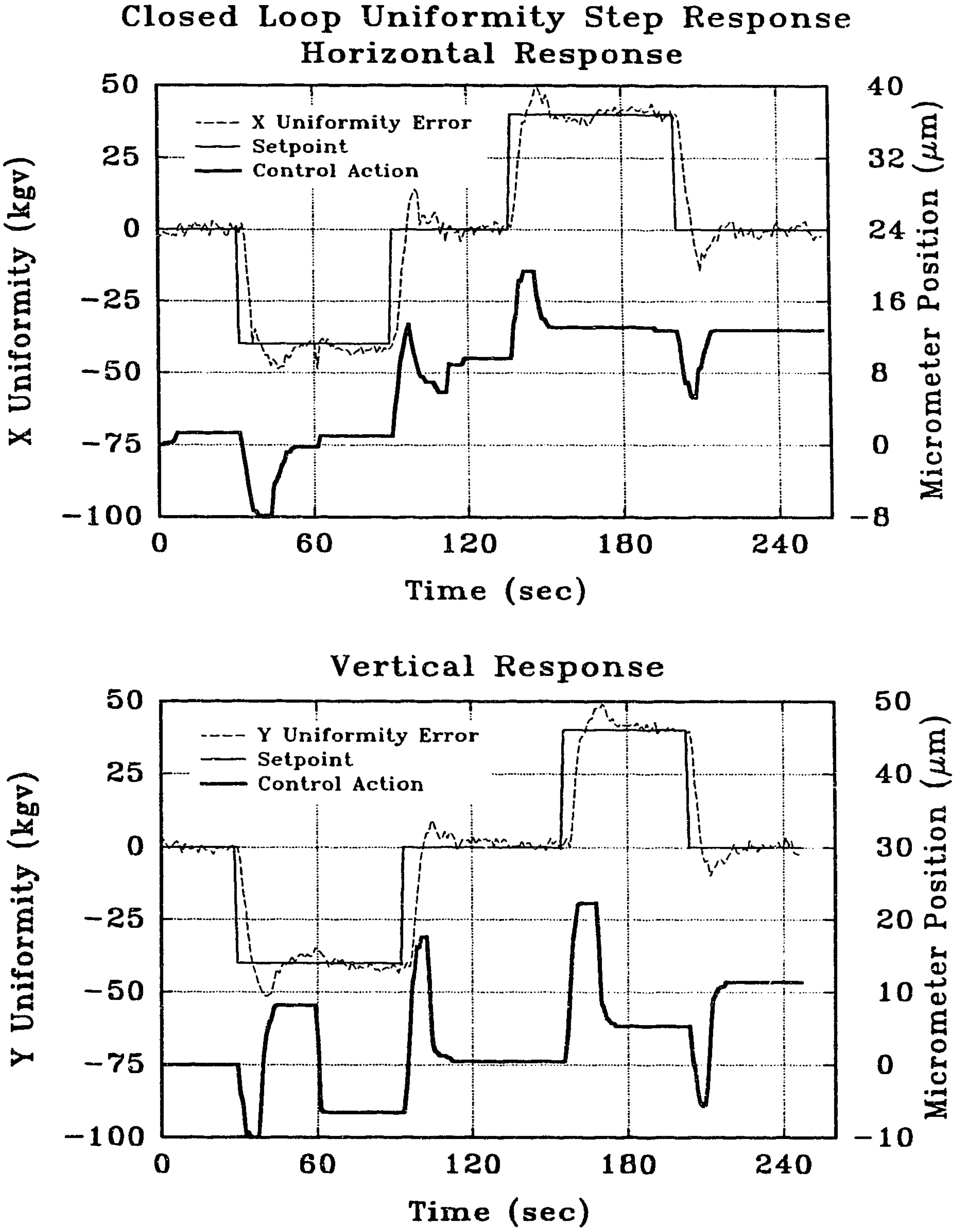


Fig. 6.17. Transient closed loop uniformity response for large setpoint changes.

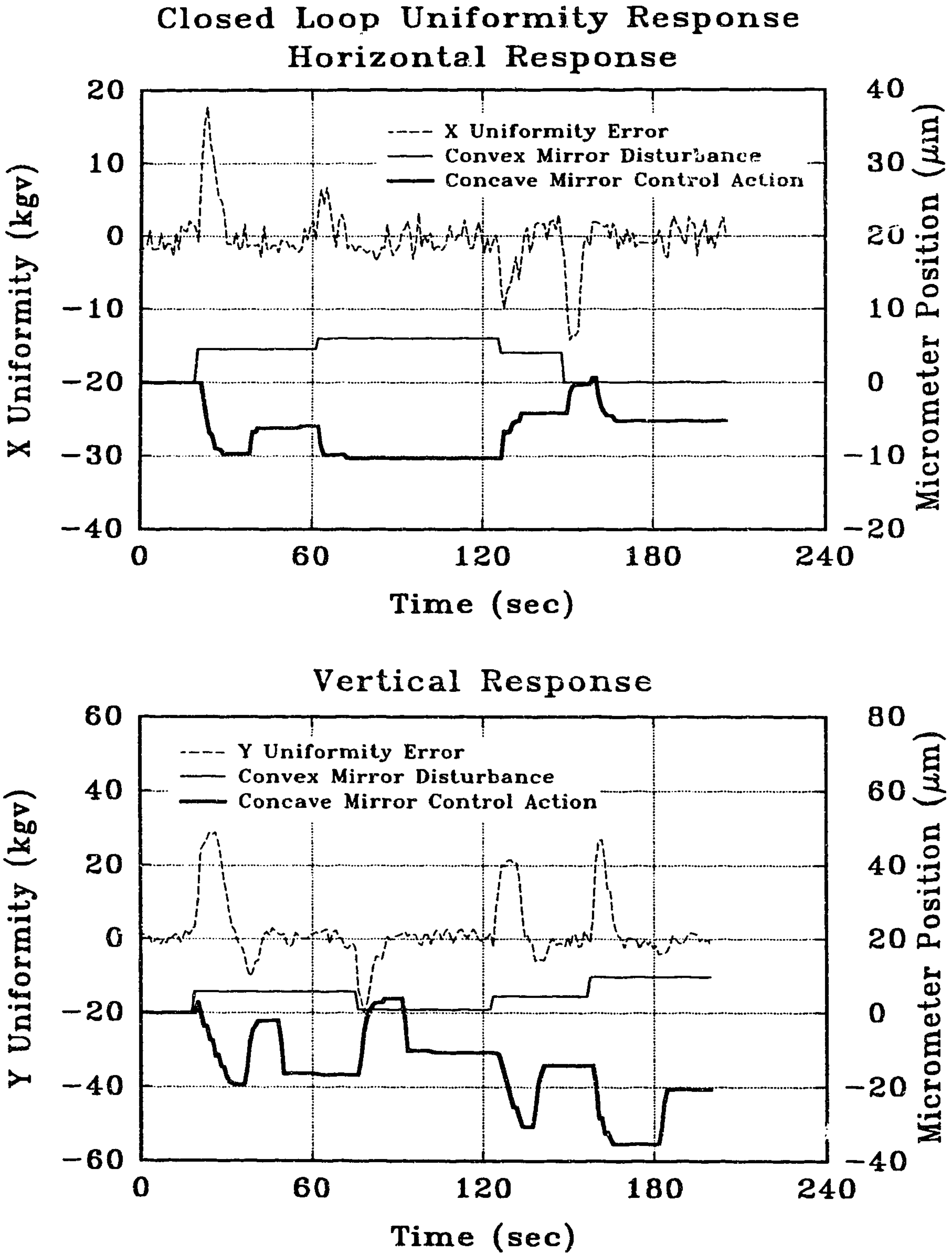


Fig. 6.18. Closed loop response of uniformity controller to external disturbances.

uniformity, the controller offsets were modified by ±8.0 kgv. The resulting response reveals that steady-state values are reached in 4 to 5 seconds. Plots of the corresponding control actions indicate error compensation with no overshoot. Fig. 6.17 illustrates similar responses, but for larger step changes (±40 kgv). Response times are slightly longer (6 to 8 seconds) with a single characteristic overshoot, due to the increased time constant for large (global) intensity variations.

Steady-state uniformity controller performance and its response to external disturbances is depicted in Fig. 6.18. With only the uniformity and power controllers activated, disturbances to beam uniformity were introduced with the convex (position control) optic. The results indicate that uniformity is restored within 4 to 8 seconds, depending on the magnitude of the disturbance. Also evident is the poorer performance of the vertical motormic due to its larger hysteresis v-'ue (13.8 μm as compared to 4.3 μm for the horizontal direction). The performance of the system was also investigated when both X and Y uniformity errors co-exist. Simultaneous changes to the horizontal and vertical offsets required approximately twice the length of time to execute. A similar response was observed for concurrent introduction of X and Y disturbances. This performance characteristic was expected since only a single motormic can be operated at any given time.

6.6.3 Beam Position Controller Performance

Transient responses for the beam position control system are presented in Fig. 6.19. Initially, the horizontal position error resided within 1 pixel of the setpoint, shown in the first graph. Hence, beam uniformity was maintained, with no position control actions being executed. At 55 seconds, the horizontal position setpoint was modified by 1 pixel, which activated the position controller to regulate the convex mirror. Following each convex mirror movement, the resulting disturbance to the uniformity process was corrected with the concave optic. Each

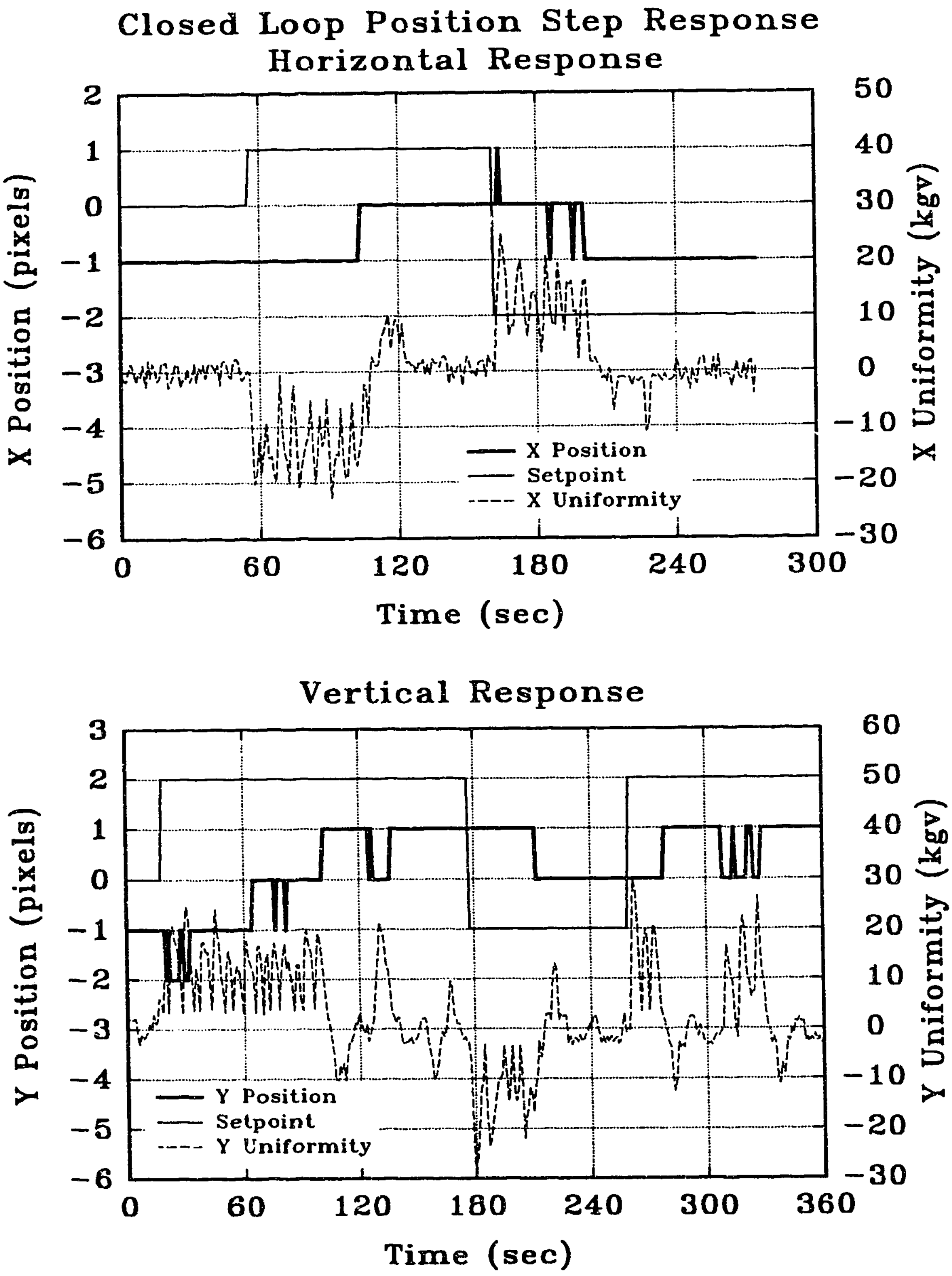


Fig. 6.19. Transient closed loop position controller performance.

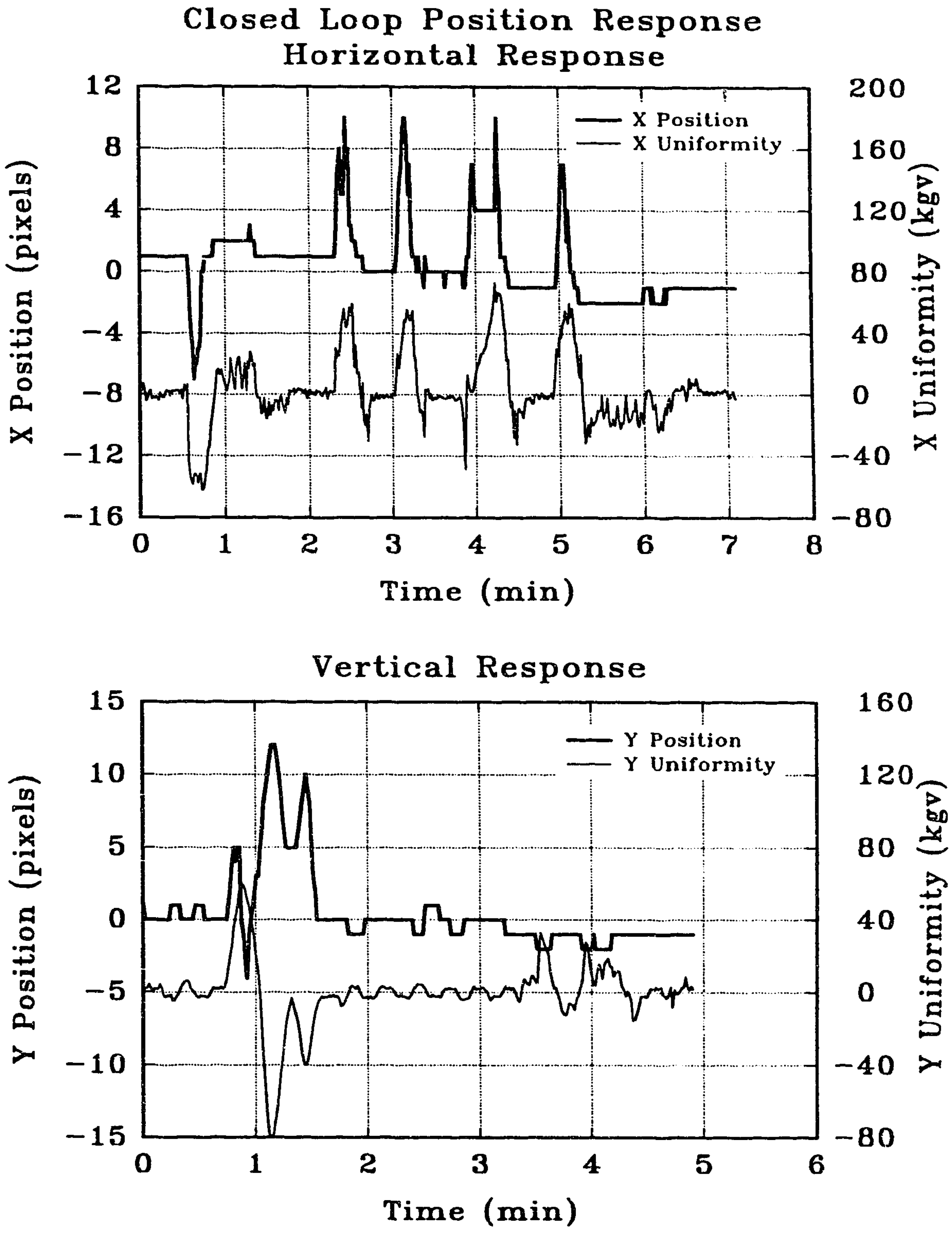


Fig. 6.20. Closed loop response of position controller to external disturbances.

uniformity disturbance peak represents a corresponding convex mirror movement of 5.5 μm. After nine successive steps, beam position was aligned within 1 pixel of the setpoint (in approximately 48 seconds). At 161 seconds, a negative step change was instigated, with similar results. The second graph depicts the vertical position response for both positive and negative step changes. A positive step of 3 pixels was completed in about 2 minutes, while the negative step of 1 pixel was executed in 43 seconds. As noted previously, the inferior performance of the vertical uniformity motormic, due to its large hysteresis window, is readily apparent.

The response of the system to external disturbances is illustrated in Fig. 6.20. During steady-state operation, large movements of the convex optic (about ±30 μm) were introduced by manual misalignment. In all cases, subsequent disturbances to both the position and uniformity errors were corrected in approximately 30 to 45 seconds. Beam position was then maintained within ±1 pixel (±60 μrad) of its specified value.

6.6.3 Sampling Percentage Controller Performance

The closed-loop performance of the coupling percentage controller is illustrated in Fig. 6.19. Since the optimal range of the detector is between 70 and 110 kgv, the system was initially operated at an intensity level of about 85 kgv. The laser power was then increased (at approximately 12 seconds), which resulted in the screen intensity exceeding the upper limit of 110 kgv. Immediately, the coupling percentage is decreased by instigating a -0.2 V wand speed controller setpoint change, which returned total intensity to its previous state. At 45 seconds, the output power was reduced to its initial value. The subsequent intensity change caused the coupling percentage to be increased, and beam intensity was restored to its optimal value. The wand assembly was tested at power levels ranging from 1 to 12 kW, which resulted in sampling from 2% to 0.25% of the output respectively.

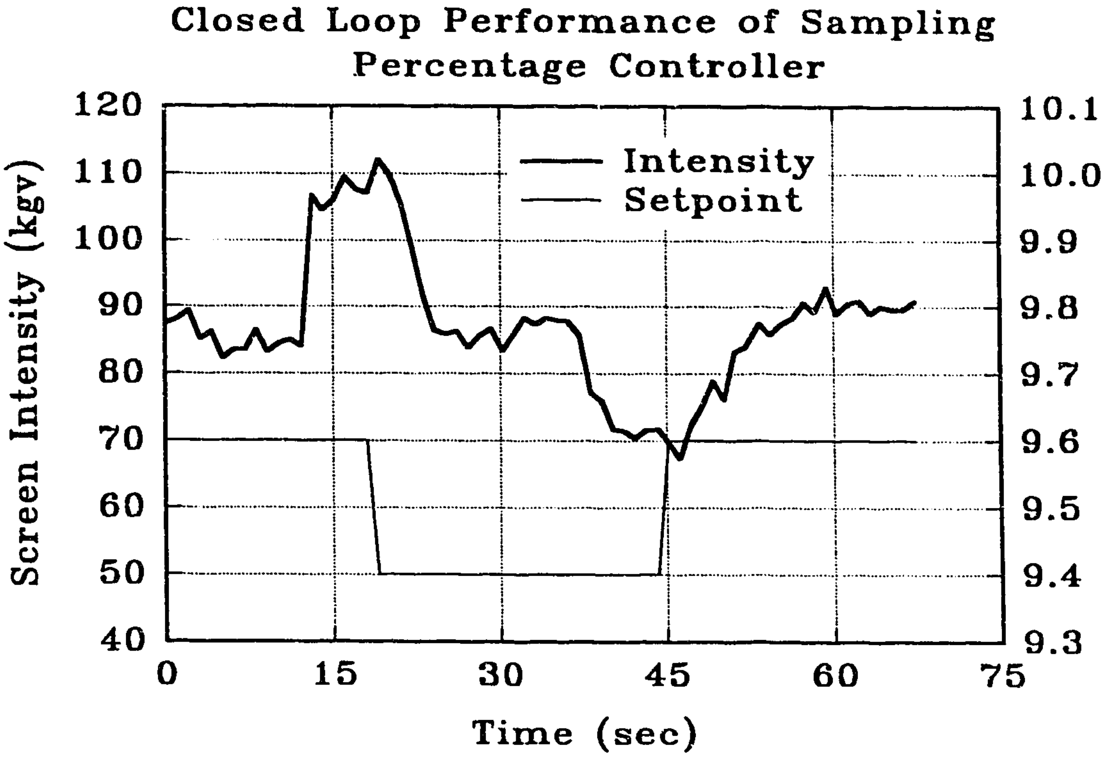


Fig. 6.19. Response of sampling percentage controller to intensity disturbances.

6.6.4 Output Power Controller Performance

The response of the output power PI control system to changes in controller setpoint is depicted in Fig. 6.22. The output power was controlled based on the total screen intensity (using the necessary conversion gains), provided the uniformity process control action was within ±2.0 μm. Power levels were measured directly with the laser's calorimetric power meter. The resulting data revealed that steady-state values were reached 5 to 10 seconds after a setpoint change. The associated controller output (the input to the laser's current controller) data indicates no overshoot or oscillations. Fig. 6.23 depicts closed loop response to disturbances in output power caused by changes to the manually controlled setpoint. In each case, output power level was restored to the computer-controlled setpoint of 5 kW

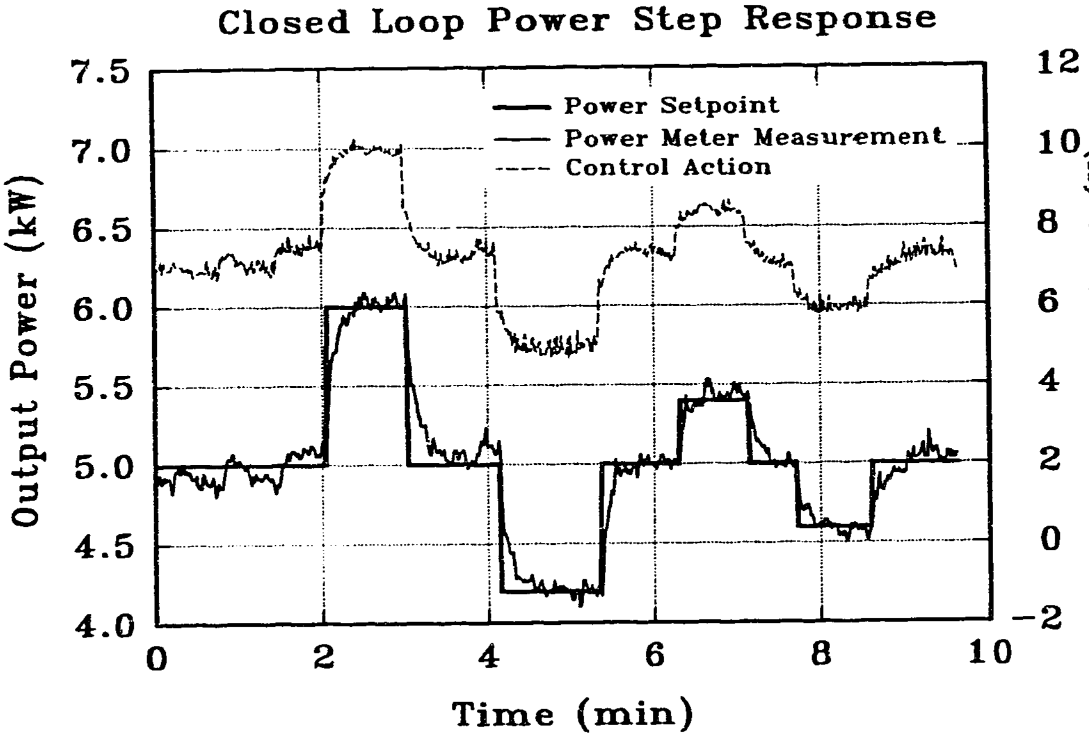


Fig. 6.22. Closed loop response of power controller to setpoint changes.

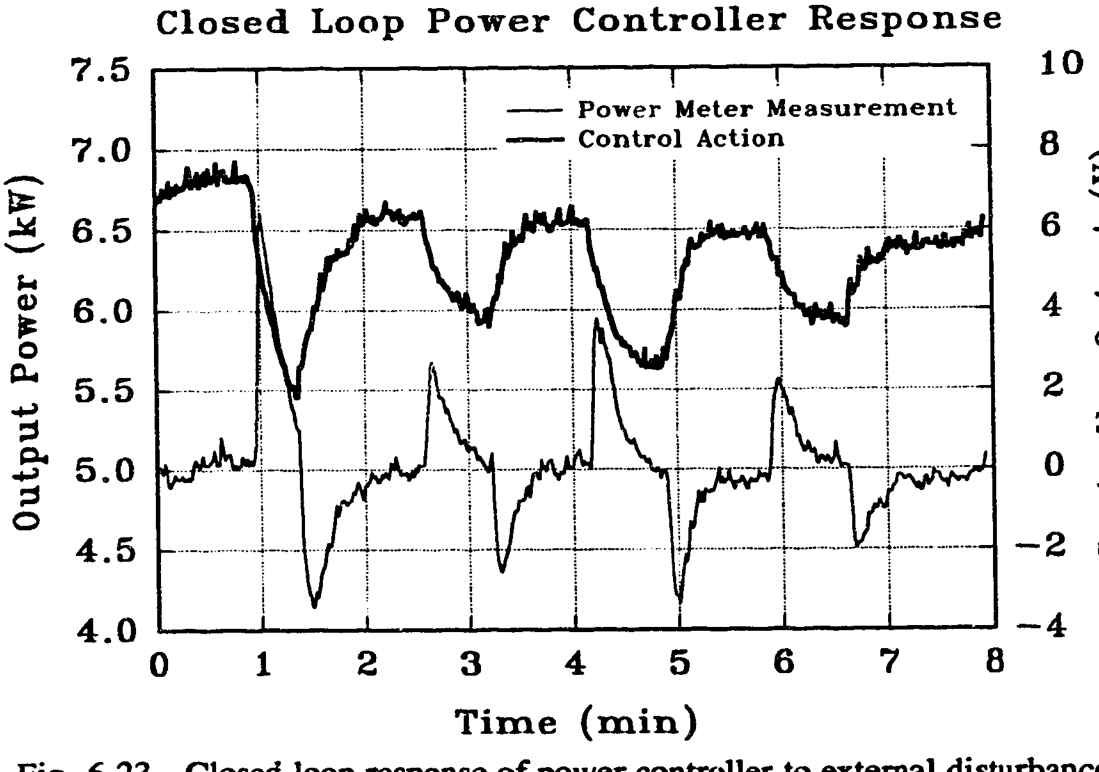


Fig. 6.23. Closed loop response of power controller to external disturbances.

within 30 to 45 seconds.

6.6.5 Overall System Performance

Similar to overall evaluation of the PIE-3 system, operation of the PIE-4 laser was examined with control systems disabled and enabled. Although the PIE-4 beam parameters do not degrade as rapidly, due to its improved design, a considerable enhancement in system performance was still observed.

The three graphs of Fig. 6.24 depict laser operation with the corresponding controllers deactivated. For the top graph, all control systems, with the exception of the sampling percentage controller, were deactivated. As evident, beam uniformity drifts continually in a random pattern. Error signals in this graph have been filtered for better visualization and clarity * The middle graph illustrates the concept of beam steer. With both uniformity and power controllers enabled, beam position was monitored from initial startup. Results indicated that the overall beam position drifted by as much as 2 mm. The bottom graph reveals variations in output power with only the uniformity controller activated. Although power levels remain relatively constant, an immediate improvement in perceived when the power controller was enabled at 14 minutes.

The performance of the laser with all control systems activated is illustrated in Fig. 6.25. The relatively large disturbance to beam uniformity during the first 3 minutes of operation was a result of convex mirror movements initiated by the position controller. However, once initial position and uniformity errors were corrected, steady-state values were maintained. The output power was stabilized from startup to within ±0.2 W. This 4% error is a direct result of the resolution of

* Data was filtered for presentation utilizing a third-order Butterworth, which was implemented using the MATLAB program. Corresponding phase lag was eliminated by applying the specified filter in both the forward and reverse directions.

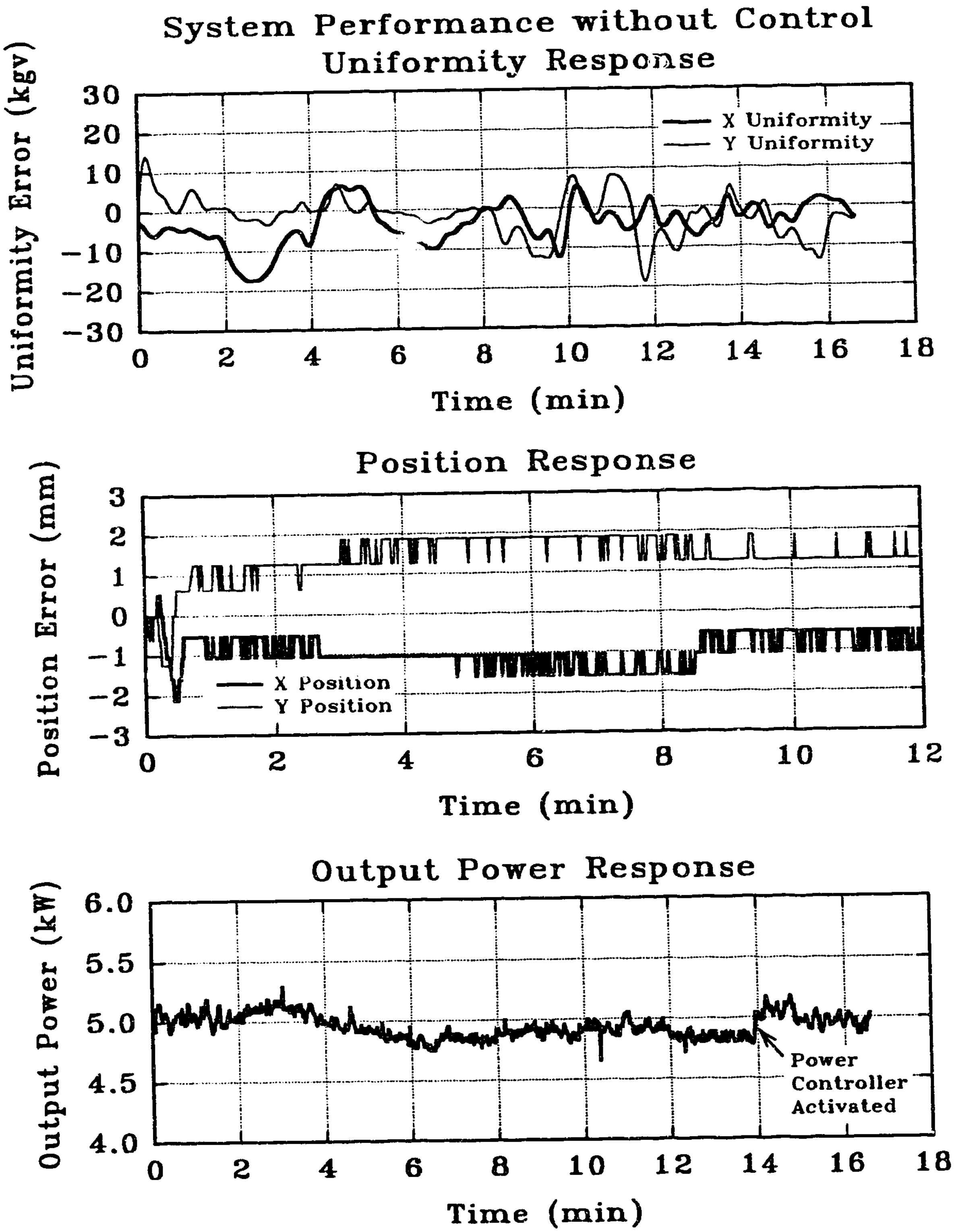
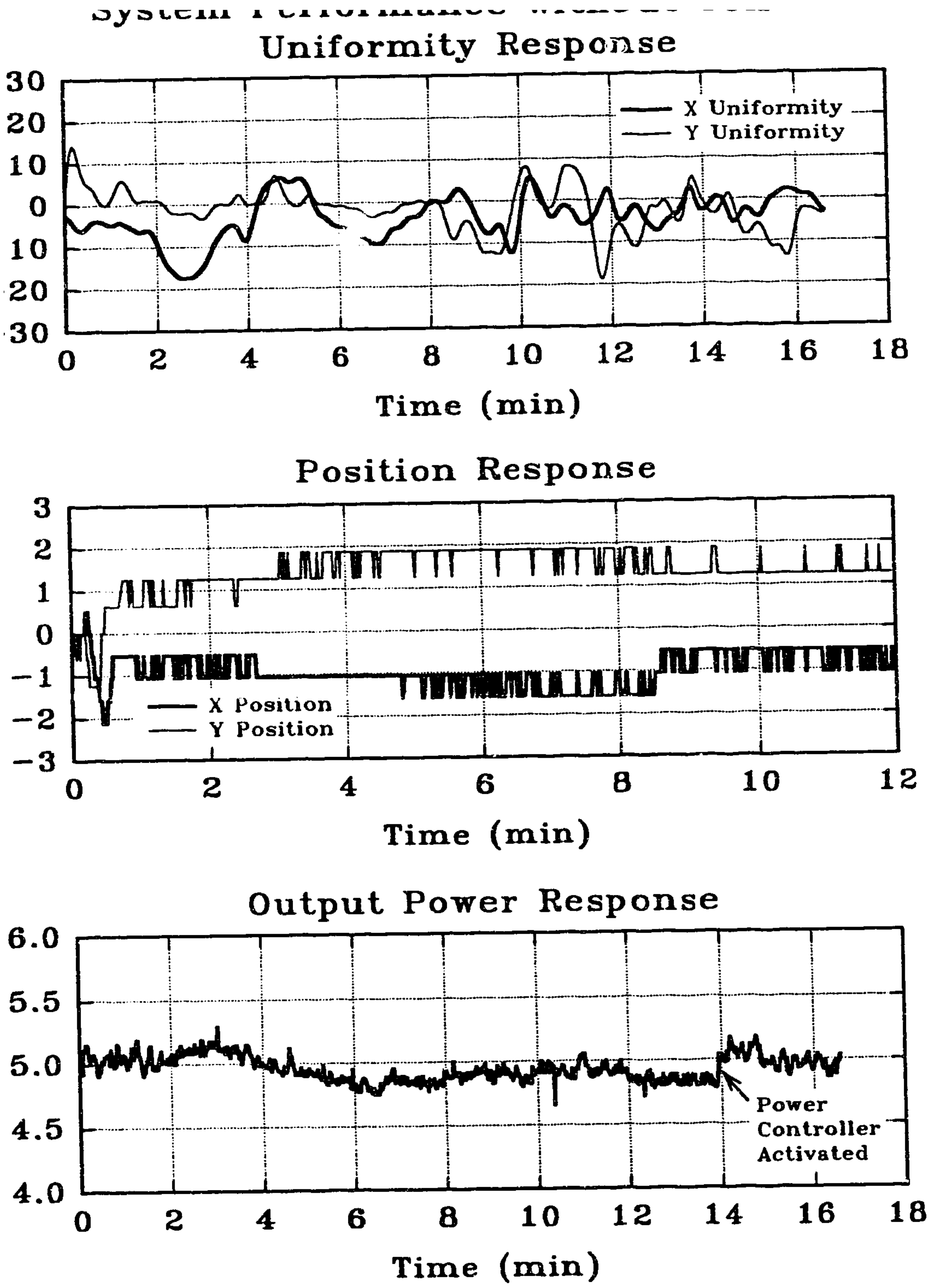


Fig. 6.24. Performance of laser system with control systems deactivated.



ig. 6.24. Performance of laser system with control systems deactivated.

the laser's input current controller. This unit is designed to oscillate within its error window, which in turn produces oscillations of the output power. Attempting to compensate for this condition via the software controller is futile due to the imaging screen's relatively long time constant.

6.7 Control System Stability

The stability analysis for the PIE-3 controller required experimental determination of critical gain values, due to the lack of accurate process models. Although more detailed modelling of the PIE-4 system was performed, non-linear controller design for uniformity stabilization necessitates the use of a similar experimental approach for stability analysis of this system. However, the linear design of the output power controller enables oscillatory behavior to be examined using mathematical techniques. For this system, the MATLAB program and its control system toolbox was used to simulate system performance as controller gains were increased.[78]

The beam position control system is highly stable due to its design, which limits controller outputs to a maximum of ±5.5 µm. Since this amount of motion does not effect the beam centre by more than 1 pixel (process sensitivity is about 11 µm/pix), system stability is guaranteed. This reasoning may also be applied to the sampling percentage control system. As described previously, controller output is restricted to ±0.2 V, which corresponds to an intensity change of approximately ±20 kgv. Since the size of the error threshold window is 40 kgv, and a 5 second delay is implemented for the imaging screen to adequately respond, system oscillations are not possible. Therefore, only the uniformity and power control systems require a detailed stability analysis.

For the case of the horizontal and vertical uniformity control systems, the closed loop performance was monitored as the controller gains were increased.

Fig. 6.26 depicts the horizontal uniformity response for the controller gain, K_{cxu}, of 0.230 µm/kgv. A small disturbance was introduced at 40 seconds. The resulting oscillatory response indicates that the critical gain value is approximately 0.230 µm/kgv. At 150 seconds, the operational controller gain of 0.120 µm/kgv was reinstated, and system stability was restored. A similar analysis for the vertical uniformity controller revealed a critical gain of 0.255 µm/kgv. Since both critical gain values are approximately a factor of two larger than the operational gains, the uniformity control system is stable.

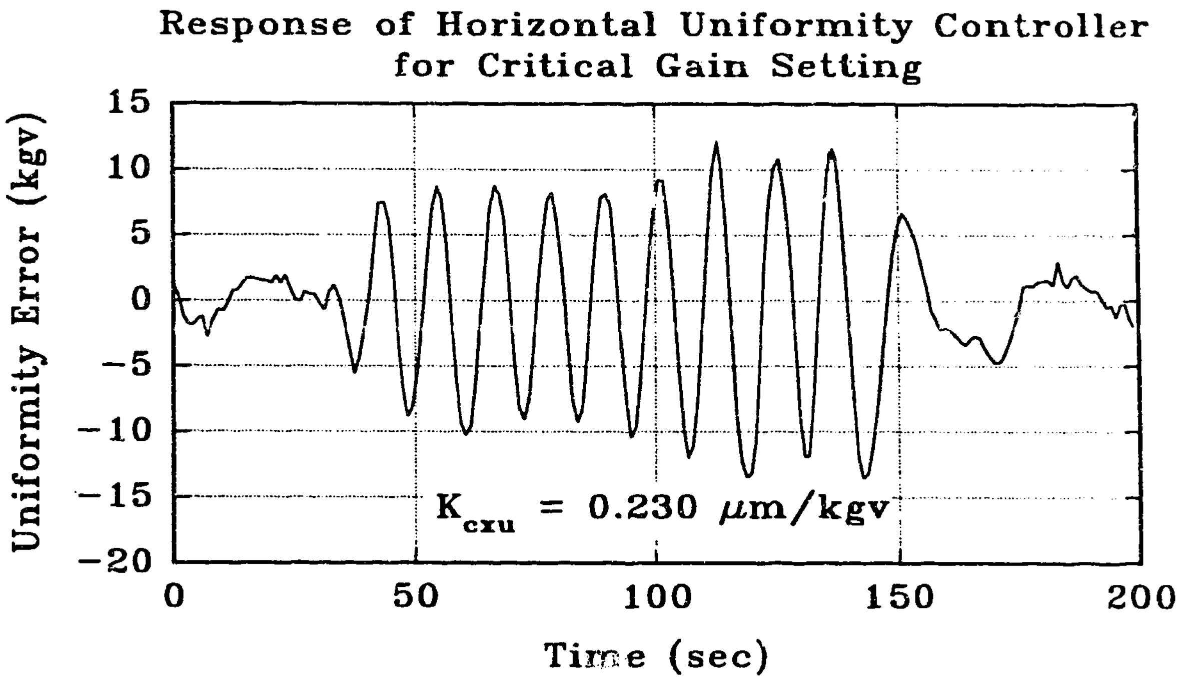


Fig. 6.26. Horizontal uniformity response for critical controller gain setting.

For the output power control system, the closed loop system transfer function is given by:

$$G(s) = \frac{P}{P_{sp}} = \frac{G_p(s)G_{cp}(s)}{1 + G_p(s)G_{cp}(s)}, \tag{6.24}$$

where G_p is defined by equation 6.14 and G_{cp} is the PI controller specified by:

$$G_{cp}(s) = K_{cp}\left(1 + \frac{1}{T_i s}\right). \tag{6.25}$$

The resulting transfer function model was simulated for varying values of K_{cp} and T_i. Fig. 6.27 depicts the simulated response for a controller gain, K_{cp}, of 1.0 V/kW and integral time, T_i, of 5 seconds (used to obtain all performance data). Also shown is the response when the gain increased to 5.0 V/kW and the integral time is decreased to 1 second (a 500% change). Although these modifications to controller parameters cause a significant overshoot, no oscillatory behavior results, indicating that the controller is highly stable.

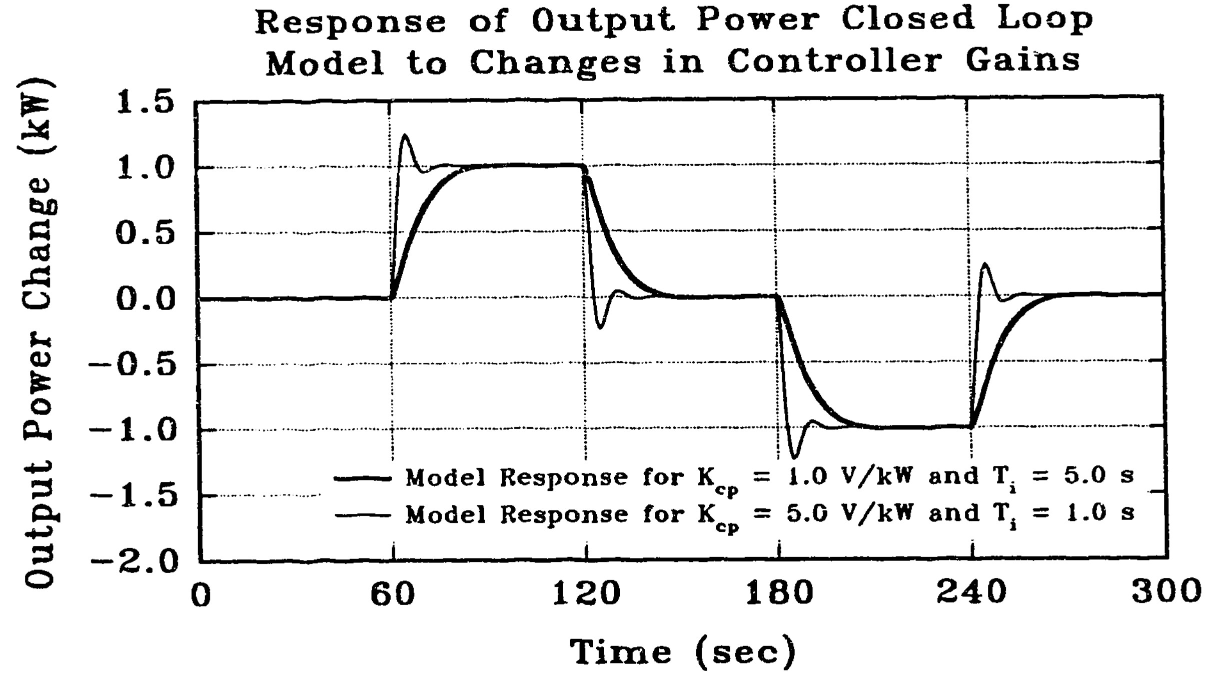


Fig. 6.27. Response of closed loop power control model to modifications in controller parameters.

6.8 Discussion and Conclusions

The original design objectives for the PIE-4 laser beam control system have been achieved. All four individual controllers, beam uniformity, beam position, output power, and sampling percentage, performed as expected, to significantly improve overall laser performance. The relatively effortless transfer of the system to the PIE-4 laser demonstrates the flexibility of this vision-based software controller.

Sensitivity analysis of the resonator optics revealed that the concave mirror was more amenable for uniformity control in this laser, since it permitted finer adjustments to uniformity signals. Thus, the function of each optic was reversed to facilitate uniformity control with the concave mirror and beam position control with the convex unit. Final performance data revealed that optimal beam uniformity was maintained within ±3 kgv, which corresponds to approximately 3.5% of total beam intensity. In addition, output power was stabilized to within 4% of the user defined setpoint. The beam position was maintained within ±1 pixel, which translates to an angular deviation of approximately ±60 μrad from the desired direction of beam propagation. The sampling percentage controller was capable of accurately regulating the speed of the rotating wand to vary the coupling percentage between 0.25% and 2%. This ensured operation within the optimal dynamic range of the infrared imaging screen (70 and 110 kgv), while permitting laser power levels to be modified from 2 to 12 kW.

The transient response times of the controller were relatively slow due to the time constant of the imaging screen and the lengthy sampling period of 1 second. Most uniformity errors were corrected in about 4 to 8 seconds, depending on their magnitudes. Modifications to beam position required approximately 30 to 45 seconds for completion of a minimum resolvable movement of 1 pixel (~0.5 mm). This delayed response was partially due to controller design, which prevented large movements of the convex optic in order to prevent severe disturbances to beam uniformity. Output power setpoint changes necessitated a comparable time of 5 to 10 seconds. However, once initial errors in uniformity, position, and output power were eliminated, steady-state operation was considerably superior. Small aberrations in error signals were usually corrected within one sample period, yielding optimally stabilized output beam parameters for long periods of operation.

CHAPTER 7

CONCLUSIONS AND FUTURE RESEARCH

The computer vision system which has been the subject of this research, provides a means of automatic alignment and power control of a laser system to ensure operation at maximum efficiency, and with maximum beam intensity and uniformity. In addition, various diagnostic functions to measure the overall quality of the output beam have been incorporated to assist in analysing output beam parameters. The system's adaptability and versatility enable it to be utilized with virtually any type of laser and in any mode of operation. Since much of the system is software-based, modifications and expansions may be performed with relative ease.

7.1 Research Summary

The system was initially designed to function on an unstable resonator configuration utilizing a low power HeNe laser. The associated visible beam cross section was directly viewed by the video camera, digitized, and analysed by the microcomputer in order to develop the necessary software algorithms and control strategies. This low power simulation proved useful for testing and debugging the controller software without jeopardizing expensive high power laser components.

The system was then transferred to the high power PIE-3 CO_2 laser. The corresponding far-infrared beam required development of a infrared to visible conversion system for use with a standard video camera. The alternative of employing a thermal-sensing camera did not prove to be cost-effective. Consequently, the thermal imaging screen sensor, which operates on the principle of thermal quenching of induced fluorescence, was utilized following testing of numerous other techniques. Since this enables the control system to operate entirely with visual data, no additional electronic or process related signatures are necessary. Furthermore, due to the software control algorithm's ability to automatically detect beam edges, precise alignment of the detector is not crucial. This imaging method also provides the operator with a means of visually monitoring an otherwise invisible beam.

The experimental results obtained on the PIE-3 unit clearly demonstrated that highly stable performance can be obtained from a non-sophisticated laser when using this vision-based computer control system. The controller maintained the best possible output beam achievable by the laser by aligning the convex resonator optic, while stabilizing output power within 4% of the specified setpoint. Investigations into a dual-optic control strategy revealed that the concave optic could be used to modify beam position, thereby reducing beam steering effects. Initial testing indicated that the system was capable of maintaining beam position within ±1 mm. which corresponded to a beam steer angle of 110 μrad.

Due to inadequate operative capabilities of the PIE-3 laser, the control system was installed on the newly acquired PIE-4 unit, which was capable of output powers exceeding 25 kW. The increased operating range could not be accommodated by the imaging screen detector. This necessitated development of a variable sampling device which could regulate the coupling percentage on-line to maintain detector operation within its optimal dynamic range. Thus, the variable coupling rotating wand was constructed and incorporated into the system, along with an associated feedback control unit to set the sampling percentage based on a computer generated setpoint.

At this point, several modifications were made to the design of the controller. Addition of an accelerator board to increase error computation and program execution speed, permitted use of a constant sampling period of 1 second, instead of a varying

2 to 5 second sample interval. An enhancement to the data acquisition unit enabled motormic positions to be monitored in hardware, thus freeing computer resources for more consequential activities. Finally, the relay-type controller previously utilized for output power control (to drive the motorized variac power supply of the PIE-3 laser) was replaced with a PI controller. This modification was possible due to the PIE-4 laser's SCR controlled power supply.

Performance evaluation of the system revealed that optimal beam uniformity was preserved, while beam position was maintained within ±0.6 mm (±60 μrad divergence of direction of beam propagation). Output power was stabilized within 4% of the desired value. In addition, the sampling percentage controller regulated imaging screen intensity within an optimal range of 70 to 110 kgv.

7.2 Recommendations for Future Research

As a result of the flexibility of this vision-control and diagnostic system, a number of future investigations are possible to further improve performance. From a hardware point of view, three possible enhancements are recommended. First, the imaging screen time constant could be reduced by utilizing a more efficient heat sink. Although the transparent quartz plate permits easy mounting of the video camera behind the detector screen, thereby saving considerably space, utilization of a metal-based heat sink would yield much faster responses. Secondly, the motormics could be replaced with more recent models, possessing more torque and less hysteresis. These units also incorporate special drivers with serial interfaces, specifically designed for computer control. This would enable simultaneous movement of more than one motormic and permit more complex control strategies, such as feed-forward control. Thirdly, the optics themselves could be replaced with deformable mirrors. This would enable utilization of more elaborate image processing routines, further optimizing beam uniformity. Finally, the computer system and digitizer could be upgraded to units with faster speed, more gray levels (at least 256), and greater pixel resolution.

From a software perspective, several improvements deserve examination. Although much of the computationally intensive code has been written in Assembly language, other routines may be converted as well to decrease overall execution time. Also, the 16 gray level limit may be artificially extended to 64 by obtaining four consecutive samples at differing digitization levels, and combining this image data. Control strategies may also be changed to incorporate more sophisticated techniques, such as adaptive and internal-model-control (IMC) based tuning. However, to derive full benefit from these methods, the hardware improvements described above should be implemented.

In addition to monitoring the laser's performance, process parameters may also be examined to ensure superior material processing. Process parameters may vary significantly with respect to the width and depth of a laser cut, the height and width of a cladding bead, or the shape and uniformity of a weld.[79,80] As a result of this diversity and variability in operating parameters, a completely automatic control system, employing feedback control techniques, would ensure a high degree of consistency and repeatability. Optical sensing and feedback control have been incorporated into many conventional material processing systems, such as arc welding, and have produced significant improvements in processing quality and efficiency.[81-83] A vision-based system, utilizing two video cameras, would be able to analyse data obtained from both the laser beam and process monitoring systems. A block diagram of the proposed laser and process control system is illustrated in Fig. 7.1. Data derived from the process camera (indicated with dashed lines) may be used to make the necessary corrections to the laser application, while the original camera signal is utilized to maintain beam parameters. The feasibility and performance of such a system will depend greatly on the capabilities of the image processing system. A more precise frame grabber, with additional gray level (and possibly colour) resolution, would be required to accurately view the laser material process. The current control system possesses the necessary hardware interfaces to support an additional camera input and manipulate a pair of X-Y tables.[84] The software to switch between cameras and control the tables has also been developed.

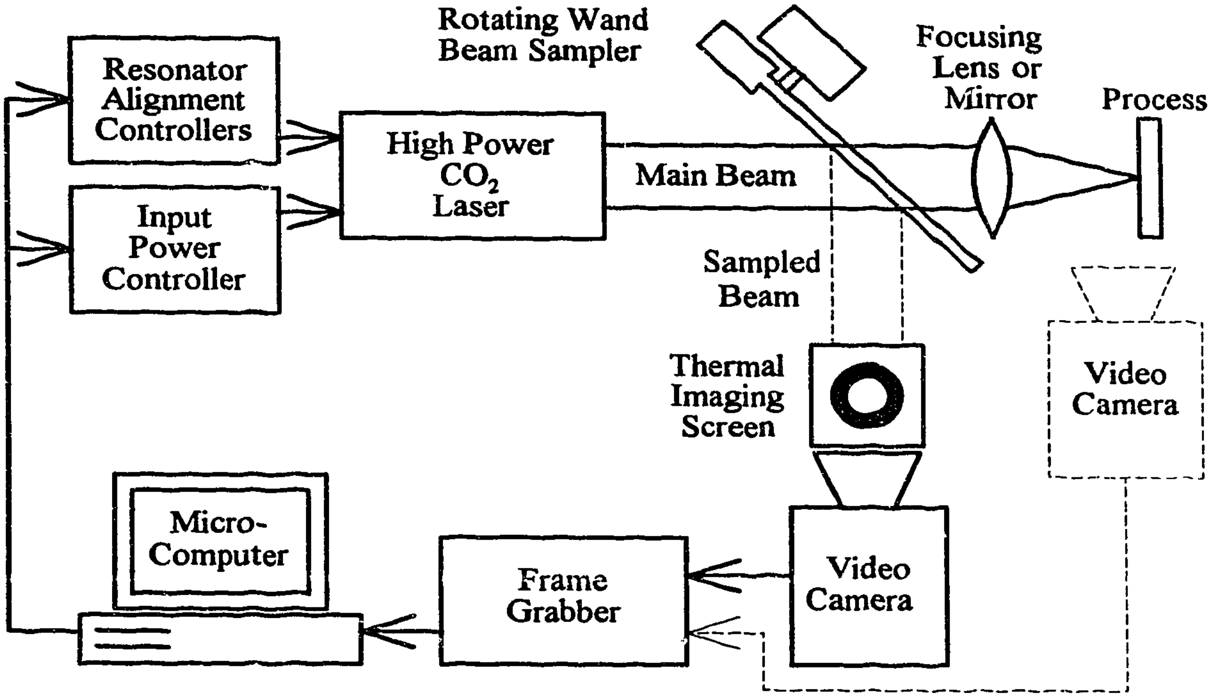


Fig. 7.1. Schematic diagram of proposed laser beam and process control system.

7.3 Conclusion

The versatility and relative simplicity of this computer-vision system, featuring both diagnostics and laser beam control, should permit cost-effective retro-installation into virtually any type of high power laser. Since the operational parameters may be easily modified from the menu-driven console, and much of the system is software-based, it should be feasible to adapt the controller for most industrial situations. The elimination of human intervention or judgement during the control procedure, provides for consistency and repeatability in many applications, and improves the quality control of the process significantly.

The author's contribution to this research included the design of all control, diagnostic, and utility software, along with the necessary control strategies to optimize laser operation. In addition, the designs of the variable sampling rotational wand, the imaging screen assembly, and all electronic circuits described in the Appendices are the original work of the author.

REFERENCES

1. T. H. Maiman, R. H. Hoskins, et al., "Stimulated Emission in Fluorescent solids II. Spectroscopy and Stimulated Emission in Ruby," Phys. Rev. **123**, 1151-1157 (1961).

2. C. K. Patel, "Selective Excitation Through Vibrational Energy Transfer and Optical Maser Action in N_2-CO_2," Phys. Rev. Lett. **13**, 617 (1964).

3. C. K. Patel, P. K. Tien, and J. H. McFee, "CW High-Power CO_2-N_2-He Laser," Appl. Phys. Lett. 7, 290-292 (1965).

4. M. Ohmine, S. Hoshinouchi, et al., "Improved Metal Processing with High Power CO_2 Lasers, Part I - Welding and Cutting," Proc. ICALEO **44**, 253 (1984).

5. E. Nakamuri, S. Kimura, et al. ,"Improved Metal Processing with High Power CO_2 Lasers, Part II - Hardening," Proc. ICALEO **44**, 261 (1984).

6. J. S. Foley, "Survey of Appiications of High Power Lasers in Manufacturing," Report #R86-917261-1, United Technologies Research Center, East Harford, CT (1986).

7. J. W. Davis, "United Technologies Industrial Laser Systems in Production," Seminar Ind. Applications of Lasers, Eindhoven, Netherlands (1984).

8. J. Powell, K. Frass, I. A. Menzies, H. Fuhr, "CO_2 Laser Cutting of Non-Ferrous Metals," Proc. SPIE - High Power CO_2 Laser Systems and Applications, **1020**, 156-163 (1988).

9. G. M. Eboo and A. E. Linemans, "Advances in Laser Cladding Process Technology," Proc. SPIE - Los Angeles (1985).

10. D. B. Snow, E. M. Brienan, et al., "Rapid Solidification Processing of Superalloys Using High Power Lasers," Proc. 4th Int. Symposium on Superalloys, Seven Springs, PA (1980).

11. G. E. Grotke, G. J. Brock, et al., "A Parametric Study of Surface Transformation Hardening With High Power Lasers," Report #84-9D4- SURFC-R1, Westinghouse Electric Corp., Pittsburgh, PA (1984).

12. A. P. Schwarzenbach and U. W. Hunziker, "Industrial CO_2 Laser with High Overall Efficiency," Proc. SPIE - High Power CO_2 Laser Systems and Applications, **1020**, 43-48 (1988).

13. A. Gukelberger, "Industrial Applications of High Power CO_2 Lasers - System Descriptions," Proc. SPIE - HIgh Power Lasers and Their Industrial Applications, **650**, 250-261 (1986).

14. I. J. Spalding, A. C. Selden, M. Hill, et. al., "High Power CO_2 Lasers," Gas Flow and Chemical Lasers, Vienna, 1-8, Aug. (1988).

15. L. Cleemann, "Eureka: Eurolaser," Proc. SPIE - High Power CO_2 Laser Systems and Applications, **1020**, 20-23 (1988).

16. J. P. [illegible] "CO_2 Electric Discharge Lasers: Present Status and Future Applications," Gas Flow and Chemical Lasers, 129-143, (1978).

17. Y. Arata, "Challange to Laser Advanced Materials Processing," Proc. Int. Conf. on Laser Advanced Materials Processing - Science and Applications, Osaka, Japan (1987).

18. "The Importance of Measuring the Spatial Characteristics of Optical Beams," Photon, Inc., Los Gatos, CA (1988).

19. O. S. Heavens, Lasers, (Scribner, New York, 1971).

20. S.S. Charschan (ed.), Lasers in Industry, (Van Nostrand Reinhold Company, Toronto, 1972), pp. 18-20.

21. A. E. Siegman, Lasers, (University of Science Books, Mill Valley, CA, 1986) pp. 558-891.

22. H. G. Heard, Laser Parameter Measurements Handbook, (John Wiley & Sons, Inc., New York, 1968).

23. A. J. B. Travis, "Laser Beam Diagnostic Equipment for On-line Use with Multikilowatt CO_2 Lasers," Proc. 5th Int. Symposium on Gas Flow and Chemical Lasers, 367-372, Aug. (1984).

24. W. P. Latham and J. D. German, "Comparing Analytical and Numerical Laser Mode Predictions to Experimental Data for Large Volume CO_2 Testbeds with Unstable Resonators," Proc SPIE - Laser Diagnostics, **343**, 64-70 (1982).

25. T. J. Ramos, D. R. Lim, and A. C. Lingenfelter, "Low-cost Laser Diagnostic System," Proc. Medecine and Biology Symposium - ICALED '85, 152-157, Nov. (1985).

26. G. C. Lim and W. M. Steen, "Laser Beam Analyser," Proc. 1st Int. Conf. on Lasers in Manufact., Brighton, U. K., Nov. (1983).

27. G. Sepold, P. O. Juptner, and J. Telepski, "Measuring the Quality of High Power Laser Beams," Proc. SPIE - High Power Lasers and Their Industrial Applications, **650**, 167-169 (1986).

28. E. Beyer, G. Herrziger, R. Kramer, and P. Loosen, "A Diagnostic System for Measurement of the Focused Beam Diameter of High Power CO_2 Lasers," Proc. SPIE - High Power Lasers and Their Industrial Applications, **650**, 170-177 (1986).

29. F. H. White, G. A. Needham, "High Energy Laser Diagnostics - A Review," Proc. SPIE - Laser Diagnostics, **343**, 2-15 (1982).

30. J. T. Knudtson and K. L. Ratzlaff, "Laser Beam Spatial Profile Analysis Using a Two- dimensional Photodiode Array," Review of Scientific Instruments, **54**, no. 7, 856-860 (1983).

31. V. M. Weerasinghe and W. M. Steen, "In-process Monitoring of Laser Processes," Proc. Int. Conf. on Applications of Lasers and Electro-optics - ICALED '85, 107-112, Nov. (1985).

32. W. M. Steen and V. M. Weerasinghe, "Monitoring of Laser Material Processes", Proc. SPIE - High Power Lasers and Their Industrial Applications, **650**, 160-165 (1986).

33. "Beamscan: Far-infrared Optical Profiler," Photon, Inc., Los Gatos, CA (1987).

34. "Laser Beam Diagnostic Systems," Spiricon, Inc., Logan, Utah (1988).

35. D. R. Akitt, H. J. Seguin,M. R. Cervenan, and S. K. Nikumb "Electronic Mode and Power Control of a High Power CO_2 Laser," IEEE J. Quantum Electron. **86**, no. 8, 1413-1417 (1990).

36. F. Martin and J. G. Willman, "Video Graphics System Analyzes Laser Beam Profiles," Laser Focus / Electro. Opt. (U.S.A.), **21**, no. 7, 104-111 (1985).

37. G. P. Anderson, "Using a Micro-computer to Measure the Intensity Distribution in a Laser Beam," Central Electricity Generating Board (1987).

38. R. A. Heyler and S. C. Guggenheimer, "Stabilized Beam Position Improves Ion-laser System Performance," Ion-Laser Technology, 107-116 (1989).

39. V. Fantini and G. Incerti, "High Performance 2.5 kW Industrial CO_2 Laser," Proc. SPIE - High Power Lasers and Their Applications, **650**, 36-38 (1986).

40. C. V. Sellathamby, H. J. J. Seguin, and S. K. Nikumb, "Mode Stabilization of a High Power Laser via Computer Vision," Opt. Commun. **78**, no. 1, 47-50 (1990).

41. C. V. Sellathamby, H. J. J. Seguin, and S. K. Nikumb, "Performance Characteristics of a High Power CO_2 Laser with Computer Vision Mode and Power Control," Appl. Opt. **29**, no. 30, 4499-4503 (1990).

42. K. H. Nam, H. J. J. Seguin, J. Tulip, "Operational Characteristics of a PIE CO_2 Laser," IEEE J. Quantum Electron. **QE-15**, no. 1, 44-50 (1979).

43. H. J. J. Seguin, K. H. Nam, J. Tulip, "Gain and Optical Measurements in a Pulser-Sustained CO_2 Laser," IEEE J. Quantum Electron. **QE-15**, no. 1, 50-54 (1979).

44. A. K. Nath, H. J. J. Seguin, and V. A. Seguin, "Optimization Studies of a Multikilowatt CO_2 PIE Laser," IEEE J. Quantum Electron., **QE-22**, 268-727 (1986).

45. S. K. Nikumb, H. J. J. Seguin, V. A. Seguin and H. Reshef, "Gain and Saturation Parameters of a Multikilowatt CO_2 Laser," J. Phys. **20**, 911-916 (1987).

46. S. K. Nikumb, H. J. J. Seguin, V. A. Seguin, R. Willis, and H. Reshef, "High-Average Power-Pulsed Performance of a Multikilowatt PIE Laser," IEEE J. of Quantum Electron., **QE 27**, 1725-1735 (1989).

47. Z. Cheng, H. J. J. Seguin, S. K. Nikumb, V. A. Seguin and H. Reshef, "Annular-coupled Concave-convex Stable Resonator for Large-volume High-quality Energy Extration," Appl. Opt. **27**, no. 5, 836-842 (1988).

48. V. E. Merchant, M. R. Cervenan, and H. J. J. Seguin, "An Industrial Quality 20 kW Infrared Laser," Proc. Int. Conf. on Lasers '85, 642-646 (1985).

49. P. D. Austin, "High Power CO_2 Laser Beam Diagnostics," Proc. SPIE - Laser Processing: Fundamentals, Applications and Controls, **668**, 232-235 (1986).

50. J. E. Harvey and M. L. Scott, "Hole Grating Beam Sampler - Versatile High-energy Laser Diagnostic Tool," Optical Engineering, **20**, no. 6, 881-886 (1981).

51. "Productivity Tools for Optimizing Optical System Performance," Photon, Inc., Los Gatos, CA (1989).

52. L. Ljung, System Identification: Theory for the User, (Prentice-Hall, Toronto, 1987).

53. "ElectroPhysics Pyroviewer - Model 5400Z," Electrophysics Corp., Nutley, NJ (1986).

54. N. C. Kerr, S. E. Clark and D. C. Emmony, "Single Pulse Two-dimensional CO_2 Laser Beam Profiling," Scientific Instruments, **22**, no. 12, 1034-1036 (1989).

55. "Thermal Image Plate for CO_2 and Other Molecular Lasers," Optical Engineering, Inc., Santa Rosa, CA (1986).

56. "Live! 2000," A-Squared Distributions Inc., Oakland, CA (1988)

57. "Proto-40K Operations and Programming Manual," ACDA Corp., Setauket, NY (1989).

58. "Oriel Encoder Micrometer," Oriel Corporation, Stratford, CT (1982).

59. Aztec C for the Amiga V.5.0, (Manx Software Systems, Inc., Shrewsbury, NJ, 1989).

60. Motorola Semiconductor Products Inc., M68000 8-/16-/32-Bit Microprocessors, (Prentice-Hall, Toronto, 1986).

61. Commodore Business Machines, Inc., Amiga Intuition Reference Manual, (Addison-Wesley Pub. Co., Inc., Don Mills, ON, 1988).

62. Commodore Business Machines, Inc., Amiga Hardware Reference Manual, (Addison-Wesley Pub. Co., Inc., Don Mills, ON, 1988).

63. Commodore Business Machines, Inc., Amiga ROM Kernel Reference Manual: Exec, (Addison-Wesley Pub. Co., Inc., Don Mills, ON, 1988).

64. Commodore Business Machines, Inc., Amiga ROM Kernel Reference Manual: Libraries and Devices, (Addison-Wesley Pub. Co., Inc., Don Mills, ON, 1988).

65. A. Abraham, "LIVE 2000 Programmer's Appendixes," A-Squared Distributions Inc., Oakland, CA (1988).

66. D. H. Ballard and C. M. Brown, Computer Vision, (Prentice-Hall Inc., Englewood Cliffs, NJ, 1982) pp. 121-148.

67. D. R. Akitt, "Automatic Resonator Alignment and Power Stabilization of High Power CO_2 Lasers," Thesis, University of Alberta, Dept. of Elec. Engg., 103-104 (1992).

68. "Coherent Laser Instrumentation Catalogue #2", Coherent Component Group, Auborn, CA (1989).

69. D. H. Owens, Multivariable and Optimal Systems, (Academic Press, Toronto, 1981).

70. D. D. Siljak, Nonlinear Systems - The Parameter Analysis and Design, (John Wiley & Sons, Inc., Toronto, 1969).

71. D. P. Atherton, Stability of Nonlinear Systems, (J. Wiley & Sons Ltd., New York, 1981).

72. "Impact 68030 Processor and 4/8 MByte RAM Expansion," Great Valley Products Inc., Paoli, PA (1989).

73. H. J. J. Seguin, V. A. Seguin, J. Dow, and A. K. Nath, "A New Calorimeter for Intense Laser Radiation," Appl. Phys. B 33, 239-241 (1984).

74. "MATLAB User's Guide," The MathWorks, Inc., Natick, MA (1989).

75. L. Ljung, System Identification Toolbox for use with MATLAB, (The MathWorks, Inc., Natick, MA 1986).

76. C. A. Smith and A. B. Corripio, Principles and Practice of Automatic Process Control, (John Wiley & Sons, Inc., New York, 1985).

77. K. J. Astrom and T. Wittenmark, Computer Controlled Systems: Theory and Design, Prentice-Hall, Englewood Cliffs, NJ, 1984).

78. "MATLAB Control System Toolbox," The MathWorks, Inc., Natick, MA (1989).

79. M. Bass, Laser Materials Processing, (North-Holland Publishing Co., New York, NY, 1983).

80. Y. Arata, Plasma, Electron & Laser Beam Technology, (American Society for Metals, Metals Park, OH, 1986).

81. S. Bangs, "Laser Vision Robot Guides Welding Arc," Welding Design and Fabrication, 45-48, Nov. (1984).

82. R. W. Richardson, "Control of Welding Using Optical Sensing," Proc. SPIE - Industrial Optical Sensing, 961, 95-113 (1988).

83. I. Ferrier, "Laser Camera Tracks Zero-gap Weld Joints," Robotics World, 6, n. 8, 22-24 (1988).

84. "Aerotech Stepping Motor Translator 4003/4005 - Instruction Manual for Model d630-1052," Aerotech, Inc., Pittsburg, PA (1978).

APPENDIX A

SAMPLING WAND CONTROLLER DESIGN

The variable coupling percentage rotating wand was designed to operate within a range of 500 to 750 RPM, which corresponds to a sampling percentage between 0% and 2%. The AC motor speed controller, required to maintain the sampling percentage based on a computer generated setpoint, was designed to monitor a pulse from a infrared LED-phototransitor sensor mounted directly behind the wand. Since the width of the gaussian shaped 0 to 5 V pulse from this sensor is proportional to the exposed surface of the wand's reflective area, a feedback controller was developed.

The schematic diagram of the AC motor speed controller circuit is illustrated in Fig. A.1. The input pulse (WTRIG) is first amplified by the operational amplifier U2A and then integrated by U2B. The U2A output is also converted to a 0 to 15 V square pulse (by Q11), the leading edge of which is used to trigger a LM555 timer, which later resets the integrator. The trailing edge of the Q11 output is utilized to trigger a sample-and-hold circuit (Q13 and U2C), which stores the final integrated value. This value is then compared to the setpoint (with U2D) and the driving error signal (CNTRL) is applied to the AC motor speed controller circuit depicted in Fig. A.2. The output of Q11 is also converted to a 0 to 5 V digital pulse (WCMPT) to enable speed monitoring via the computer. The controller setpoint may also be adjusted manually using potentiometer R42. Controller tuning was performed by varying potentiometers R59, R60, R61, and R62. The final values which yielded an acceptable performance were as follows:

R59 = 44.9 Ω, R60 = 39.4 Ω, R61 = 34.7 Ω, R62 = 46.7 Ω.

The variable speed AC motor driver (shown in Fig A.2) receives the controller output (CNTRL), which is optically isolated via an optical coupler. The driver circuit itself is a standard inexpensive speed controller utilizing the bilateral features of a triac.[*] When the triac conducts, the motor receives the normal AC line voltage. However, the firing of the triac may be delayed to a later portion of the voltage cycle, thereby reducing the speed of the motor. The setpoint voltage determines the rate at which capacitor C14 charges (through the phototransistor in the optical coupler). When C14 charges sufficiently to activate the unijuction transistor Q9, the triac is triggered by transformer T2. Thus, by regulating the charging of C14 (via the setpoint), the portion of the AC voltage waveform which is applied to the motor may be varied, resulting in a change of speed.

The system also contains a safety circuit which deactivates laser power if the wand's rotational speed decreases below a preset minimum (about 500 RPM). This circuit, illustrated in Fig. A.3, utilizes a retriggerable timer to short-circuit the laser's input current controller's potentiometer with transistor Q1. The sensor pulse from the wand acts as the trigger to the timer. If the timer is not retriggered within the time determined by resistor R1 and capacitor C1, the J-K flip-flop is set and the laser power is turned off. This circuit prevents damage to the imaging screen detector if the motor speed unexpectedly decreases.

[*] L. J. Reed, "Circuit Applications for the Triac," Motorola Semiconductor Products Inc., AN-466, pp. 6-7.

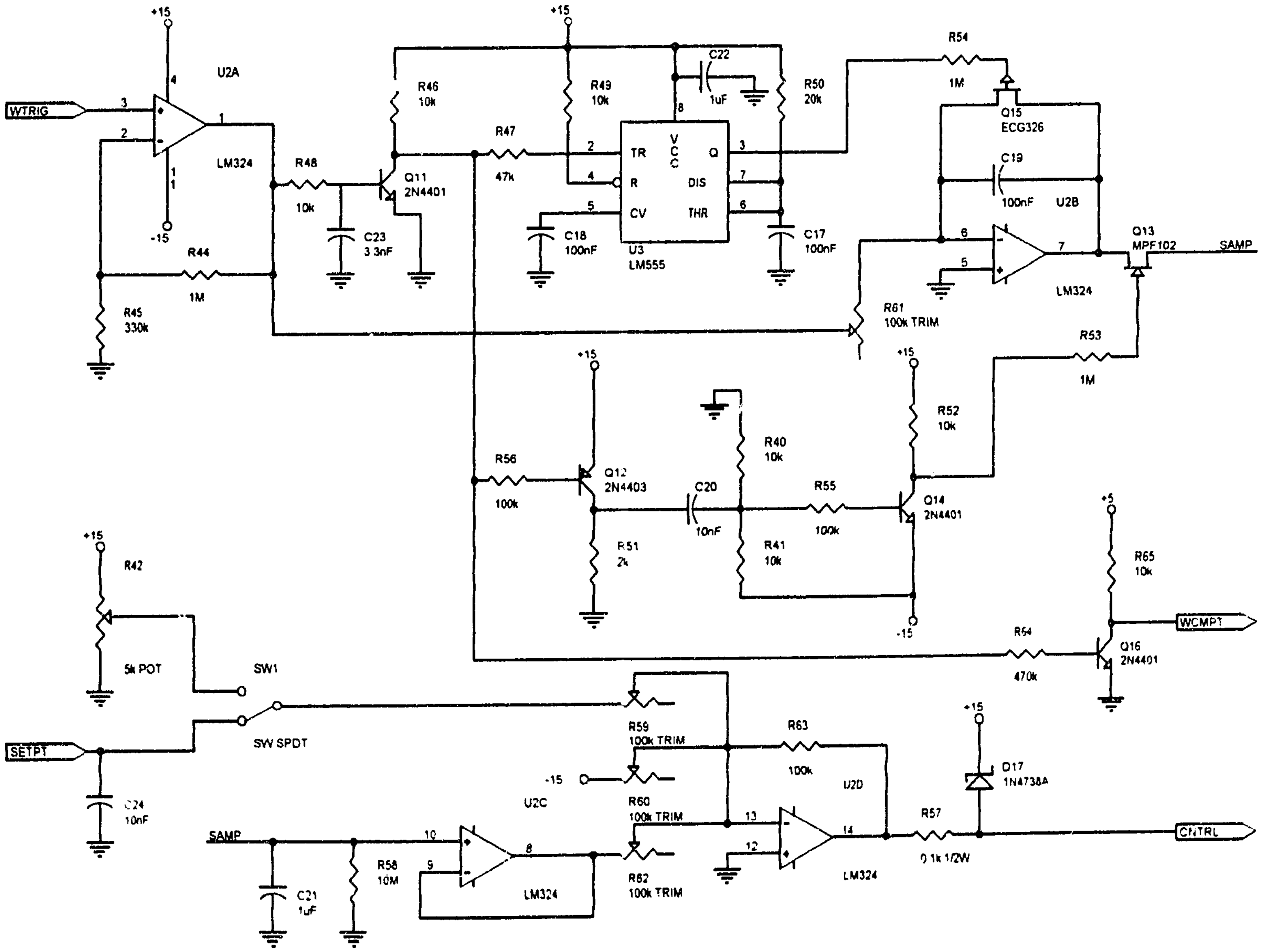


Fig. A.1. Schematic diagram of sampling percentage measurement and control circuit.

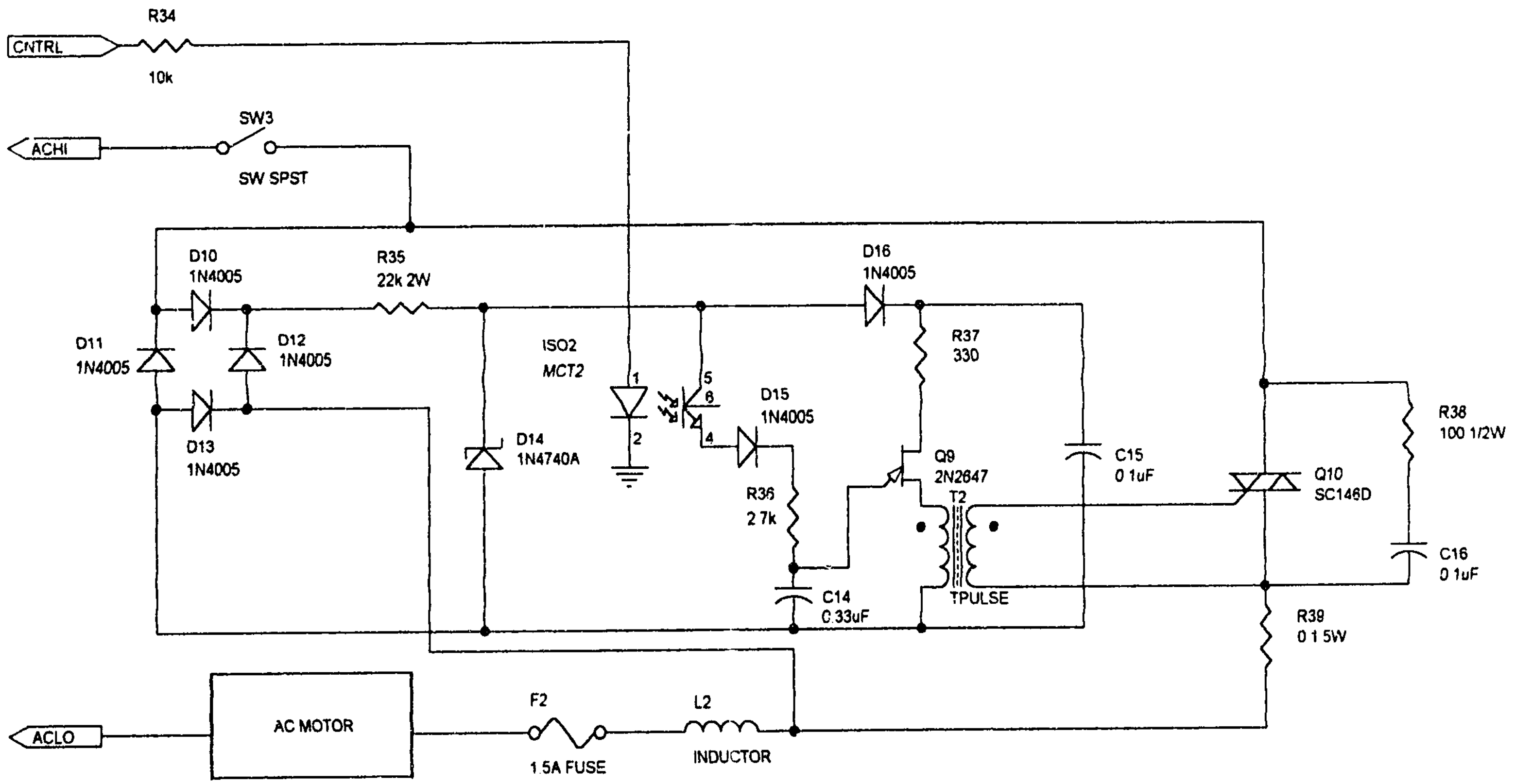


Fig. A.2. Schematic diagram of variable speed AC motor driver circuit.

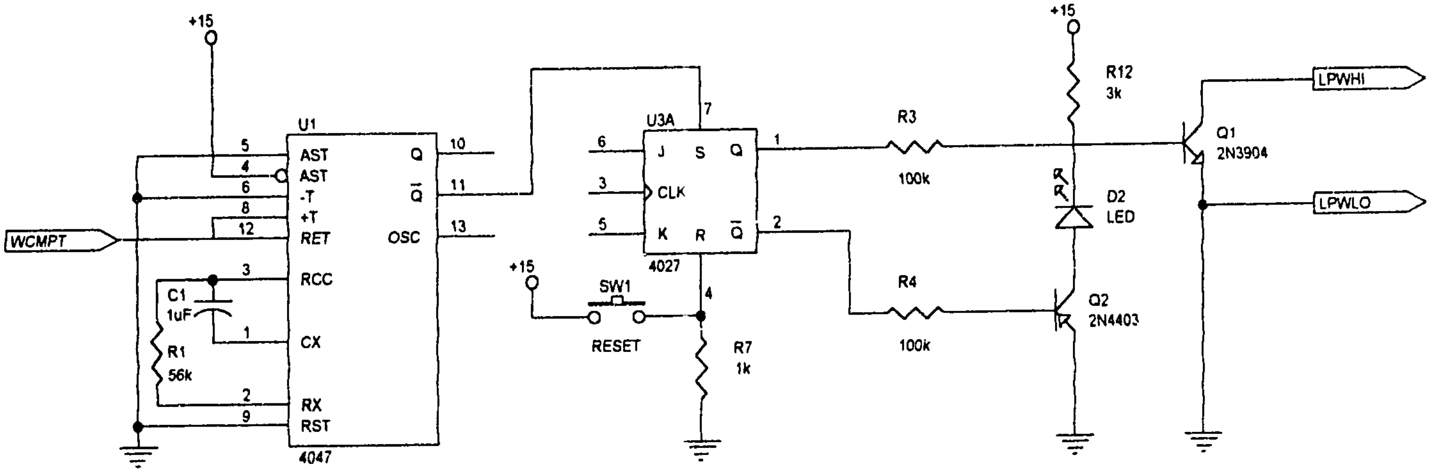


Fig. A.3. Schematic diagram of circuit which switches off laser power if wand speed decreases below minimum limit.

APPENDIX B

SUPPLEMENTARY DATA ACQUISITION BOARD

The supplementary data acquisition board was designed to fit directly onto the original unit via existing header connections, and expand its capability. In addition, the board includes a 16-bit counter and the associated logic to maintain the displacement of a single motormic. The design incorporates three programmable logic devices, along with series of conventional latches.

The schematic of the overall circuit is illustrated in Fig. B.1 and Fig. B.2. Fig. B.1 depicts the connections to the three Altera EPLD's (erasable programmable logic devices), along with the certain input and output header connections. Fig. B.2 depicts the eight latches used to hold the digital output signals which control various external devices. The most significant three bits of the Proto-40K data acquisition unit's digital outputs (PDO13 to PDO15) are used to select and enable the latches. The remaining 13 bits are utilized as data outputs. The unit's 16 digital inputs are used to read the 16-bit counter or 16 other digital inputs via a 2-to-1 multiplexer. The control functions of the output latches are as follows:

U1,U2 - Contain outputs required to control a set of X-Y tables and eight extra outputs for future expansion.

U3,U4 - Contain outputs which select the desired motormic and control the various functions of this expansion board.

U5,U6 - Contain data outputs which load the 16-bit counter with the desired amount of movement.

U7,U8 - Contain 16 digital outputs which may be used to control other external devices.

The internal logic of the 64-pin Altera EP1800J (an equivalent Intel CJ5C180-90 replacement was used) is depicted in Fig. B.3. The device implements four cascaded 4-bit counters and four 4-bit 2-to-1 multiplexers. The EP900 (Intel D5C090 replacement) logic, illustrated in Fig. B.4, consists of a 4-bit decoder to enable each of the four sets of latches, and a dual 4-to-1 multiplexer to select the direction and count inputs from the motormic drivers. Finally, the EP310 (Intel D5C031-40 equivalent) contains logic necessary to halt the active motormic and to control a set of X-Y tables (for future expansion to process control).*

* Altera Progammable Logic Devices, Altera Corp.,Santa Clara, CA (1987).

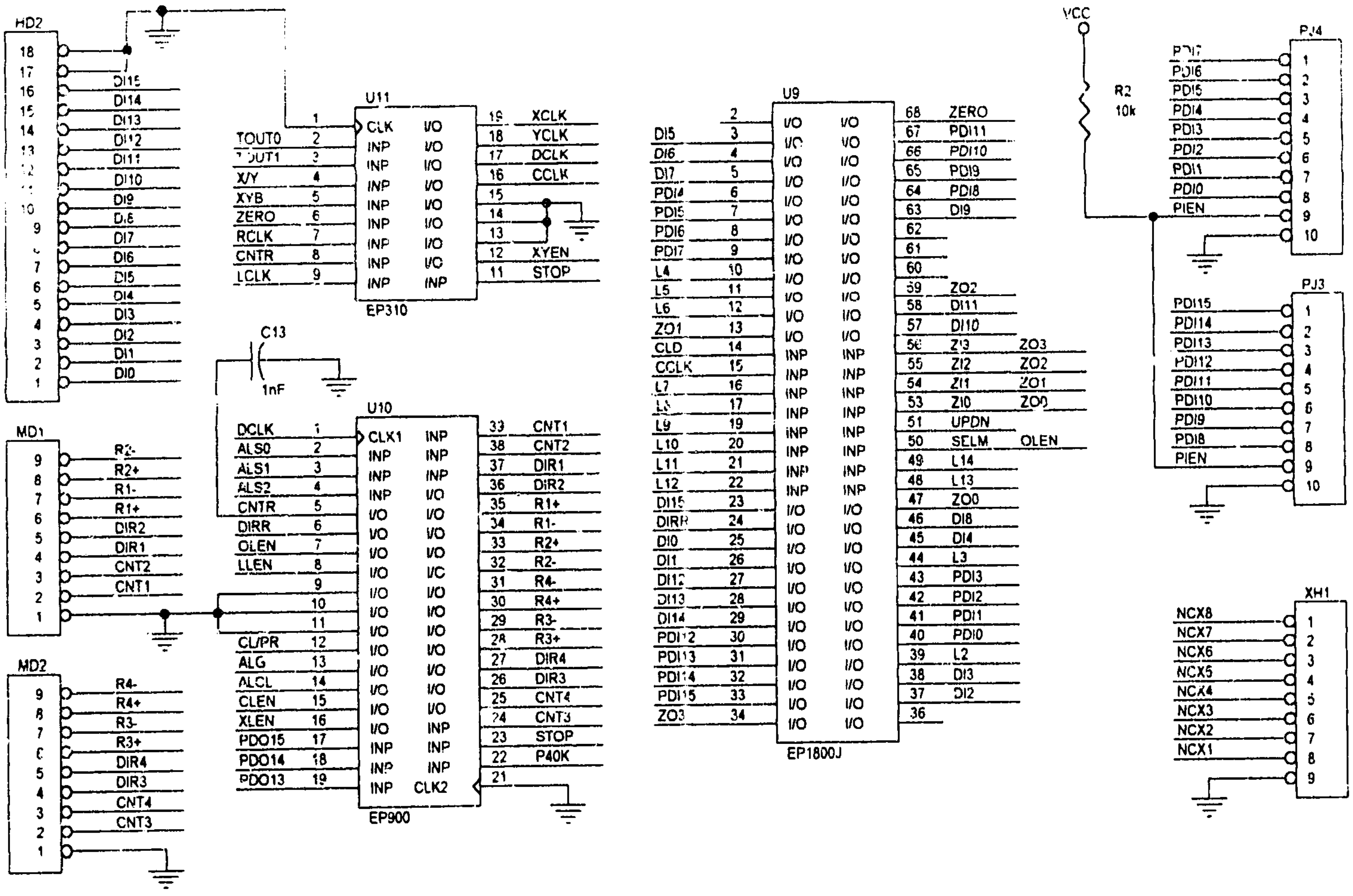


Fig. B.1. Partial schematic of supplementary data acquisition circuit with motormic position tracker.

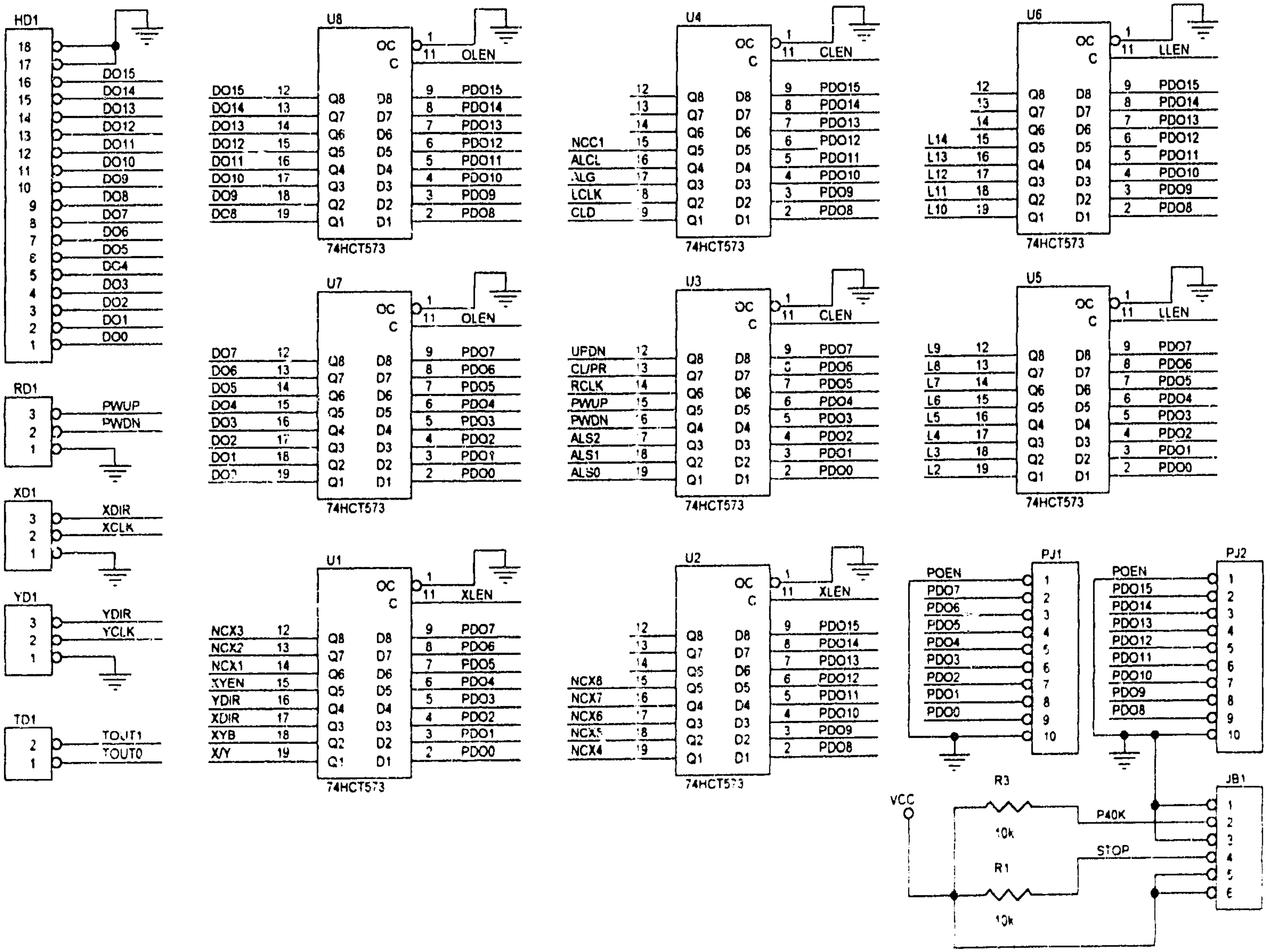


Fig. B.2. Remainder of schematic of supplementary data acquisition circuit with motormic position tracker.

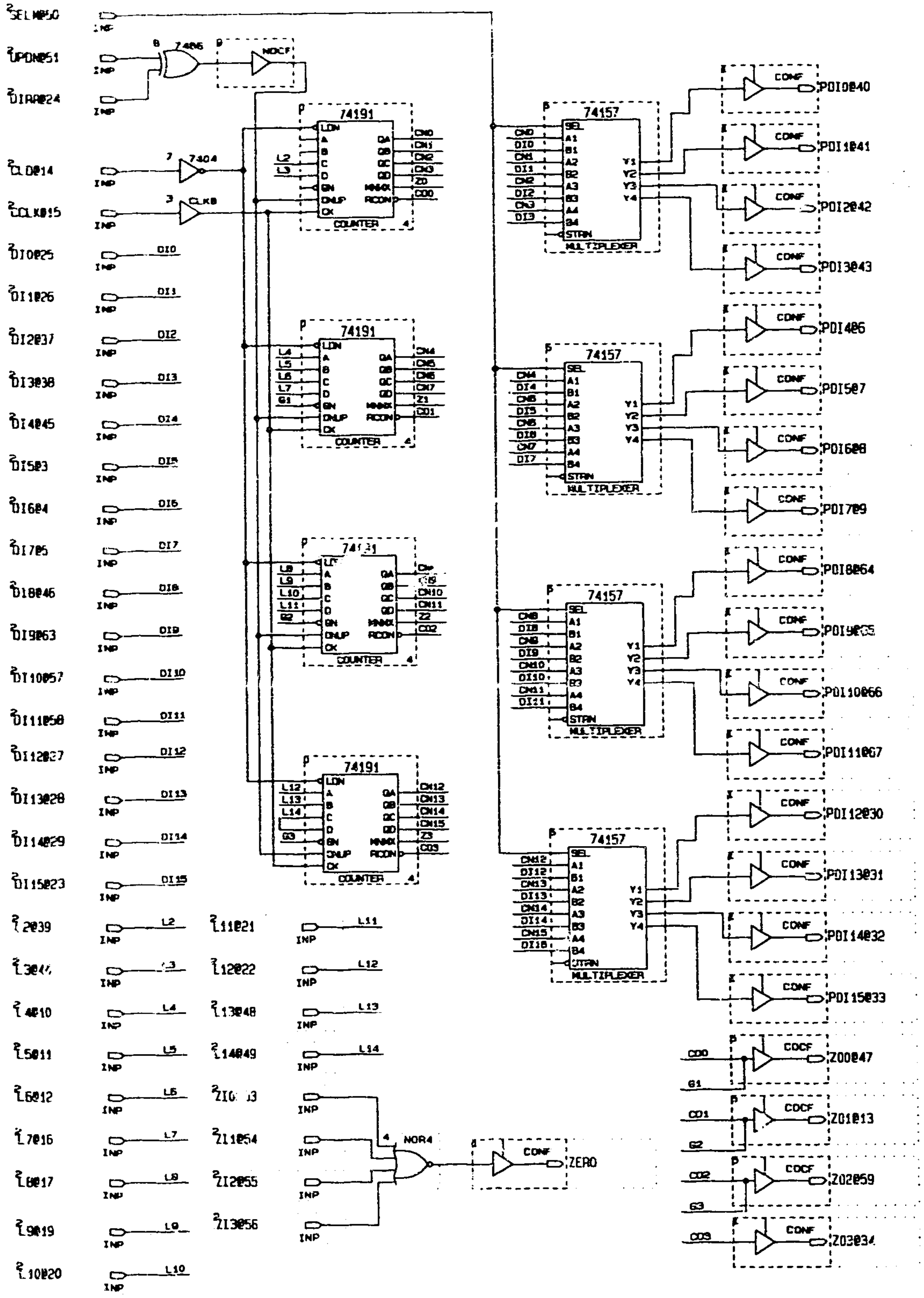


Fig. B.3. Schematic of internal programming logic of Altera EP1800J microchip.

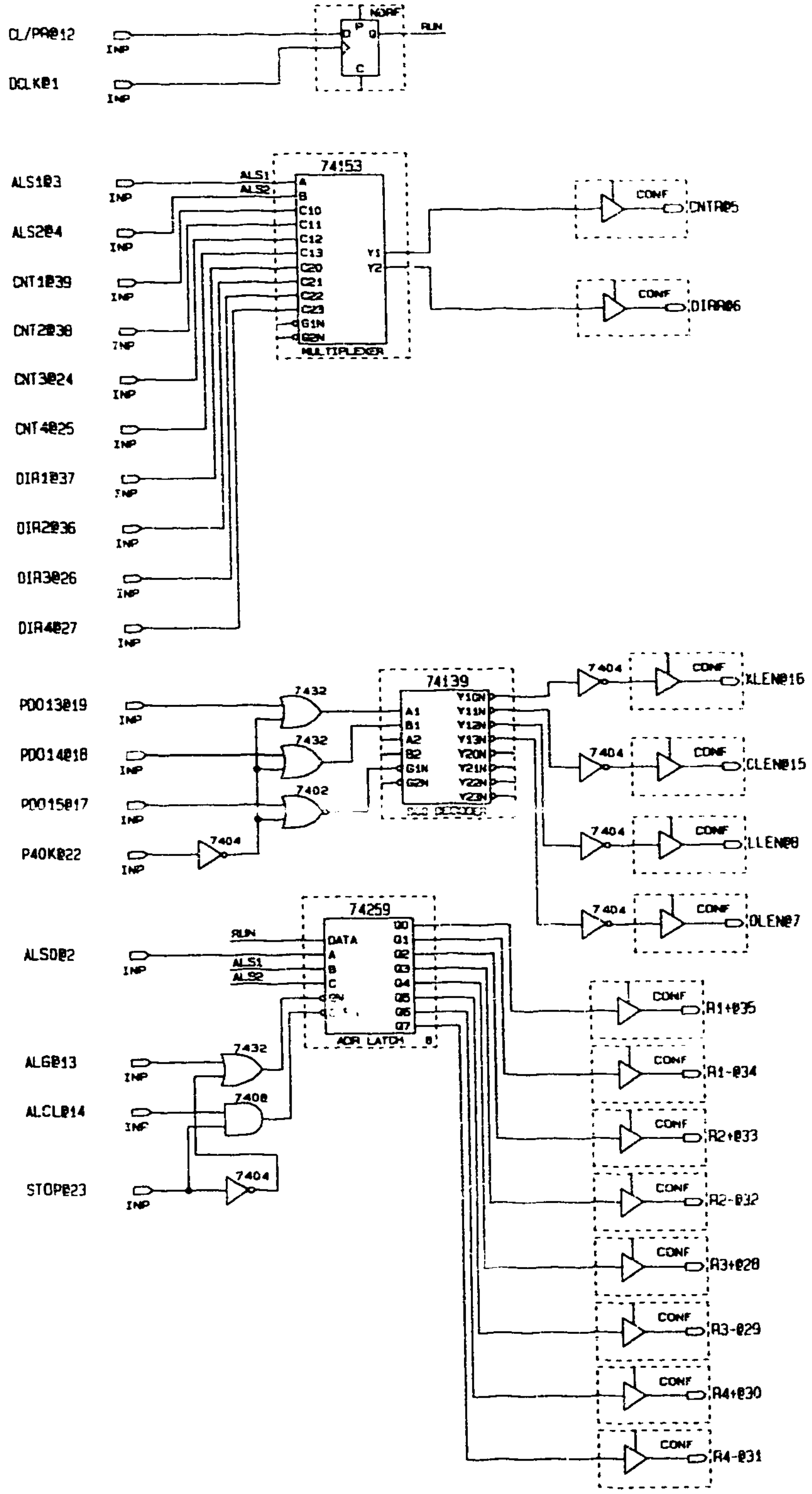


Fig. B.4. Schematic of internal programming logic of Altera EP900 microchip.

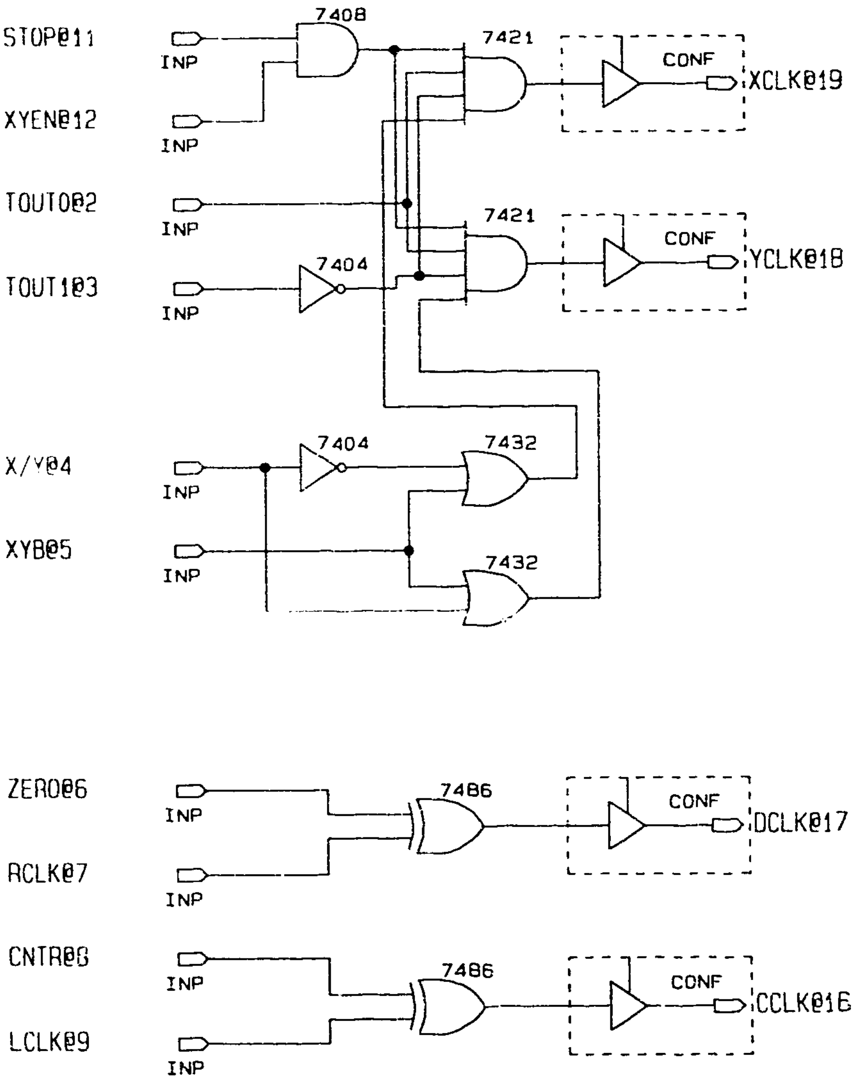


Fig. B.5. Schematic of internal programmirg logic of Altera EP310 microchip.

APPENDIX C

OPERATING MANUAL FOR CONTROL PROGRAM

The automatic alignment and power control program for the PIE-4 laser may be run from a CLI window by typing "CONTROL" or from the Amiga workbench by moving the mouse over the "CONTROL" icon and pressing the left mouse button twice.

C.1 Main Control Window

The Control window contains the gadgets necessary to allow manual operation of the laser, the output power setpoint, and all operating and error signal data. The output power setpoint may be modified by moving the mouse to the its gadget and pressing the left mouse button. The cursor will then appear in the gadget and the setpoint value may be changed using the keyboard. This method applies to all other input gadgets in any sub-windows. The manual laser control gadgets are activated by moving the mouse over the gadget and pressing the left mouse button. As long as the left mouse button remains depressed, the gadget is active.

C.1.1 Control Window Gadgets

Output power setpoint:

Set Pt - The output power level setting in Watts.

Manual control gadgets:

Mirror - Selects between the feedback (convex) and the primary (concave) mirrors. To select a mirror, move the mouse over the desired mirror gadget ("F" for feedback of "P" for primary) and press the left mouse button.

Alignment arrows: The arrows shift the overall intensity of the beam in the specified direction. For example, to make the top half of the beam more intense, place the mouse on the UP arrow and hold down the left mouse button. The micrometer will continue to move until the left mouse button is released.

Power Arrows: The power arrows increase or decrease the output power of the laser by increasing or decreasing the sustainer current. To increase the power, place the mouse over the UP arrow and hold down the left mouse button. The power will continue to increase until the left mouse button is released.

Display information: The window also contains numerical displays of all pertinent error signal and control action data, along with other operating information.

C.2 Main Menu Commands

The main program menu can be obtained by placing the mouse in the program window and holding down the right mouse button. The menu consists of four sub-menus: Project, Options, Command, and Diagnostic. When the mouse is moved over a sub-menu, the options for that particular sub-menu will be displayed and may be chosen by moving the mouse over the option while holding down the right mouse button and then releasing the right mouse button. A number of options may be chosen at one time by holding down the right mouse button, moving over the first desired option and pressing the left mouse button, then moving over the second desired option and pressing the left mouse button, and so on. These methods of selecting options applies to all levels of sub-menus.

C.2.1 Sub-Menu Item: Project

The Project sub-menu contains the commands necessary to start, stop, pause, restart, and exit the program. These options may also be selected from the keyboard by holding down the right Amiga (red A) key and pressing the key corresponding to the command. The menu items and corresponding descriptions are as follows:

Start: Begins execution of the program using the options and parameters which have been chosen. (right Amiga - S from keyboard)

Pause: Pauses the program and allows manual control of the power and alignment. No parameters or options may be changed while the program is paused. (right Amiga - P from keyboard)

Resume: Resumes execution of the program after the program has been paused using the existing parameters and options. (right Amiga - R from keyboard)

End: Stops execution of the program and allows manual power and alignment control. Any parameters and options may be changed at this time. (right Amiga - E from keyboard)

Exit: Stops execution if the program is running, closes all files and libraries, and exits the program. (right Amiga - X from keyboard)

C.2.2 Sub-Menu Item: Options

The Options sub-menu contains the commands necessary to select the various options available when running the program. When the mouse is moved over the option while holding down the right mouse button, the possible choices for the option are displayed, with a check mark beside the one which is currently chosen. The menu items and corresponding descriptions are as follows:

Uniformity:

On - Turns the automatic uniformity control function on.

Off - Turns uniformity control off. The program still computes and displays all data but does not control the chosen resonator optic.

Beam Steering:

On - Turns the automatic beam position control function on.

Off - Turns position control off. The program still computes and displays position data but does not control the second resonator optic.

Power Control:

On - Turns the automatic power control function on. The power is then changed to agree with the setpoint specified in the main window.

Off - Turns power control off. The power data is displayed but the power is not adjusted.

Wand Control:

On - Turns the automatic variable sampling wand control function on. The total screen intensity is maintained within the optimal operating range. When making large external changes to output power, ensure the controller has sufficient time to respond (i.e. do not increase power level too quickly).

Off - Turns variable sampling wand control off.

Save Data:

Yes - Save displayed data to the specified data file.

No - Do not save displayed data.

Save Images:

Discard - Do not save any images.

Overwrite - Save the current image to the image file. The previous image overwritten.

Sequence - Save the current image to a new file. Currently, only a sequence of 15 images can be saved before files are overwritten due to insufficient memory. This number may be increased if additional memory is installed. The filenames consist of the originally specified name with the image sequence number appended to it.

Compute Centre:

Yes - Find the edges of beam and compute the centre. The program searches for the beam in the region bounded by Xmin, Xmax, Ymin and Ymax. Thus, these parameters should be set such that only the beam image is present in this area.

No - Use the specified centre and edge boundary values.

Filter Data:

Yes - Filter data using one of three type of filters (four-element moving average, first-order low-pass, or third-order Butterworth). In order to change the type of filter used for each signal, the flags in the function "filterdata" must be changed and the code recompiled.

No - Do not filter data.

Display Image:

Black & White - Display image in shades of gray levels.

Colourize - Display image using spectrum (rainbow) colour map.

Camera Input:

Top - Use video signal from top camera input.

Bot - Use video signal from bottom camera input. This capability is for future expansion to process control as well.

Plot Signals:

None - Do not graphically plot any signals.

X-Uniformity - Graphically plot X-uniformity error signal.

Y-Uniformity - Graphically plot Y-uniformity error signal.

X-Position - Graphically plot X-position error signal.

Y-Position - Graphically plot Y-position error signal.

Screen Power - Graphically plot power level as measure with screen.

Power Meter - Graphically plot power meter reading.

C.2.3 Sub-Menu Item: Command

The Command sub-menu contains commands which perform various functions on-line, some of which display additional sub-windows for input. Some of these items may also be selected from the keyboard by holding down the right Amiga (red A) key and pressing the key corresponding to the command T: menu items and corresponding descriptions are as follows:

Reset Time: Reset start time to zero. (right Amiga - T from keyboard)

Reset Counters: Reset all motormic position counters to zero.

Reset Vars: Reset certain variables to th ir initial state, such as those which adaptively increment motormic movements due to their reverse motion when halted.

Calibrate: Recalibrate intensity to power conversion offset if imaging screen parameters have changed. The laser power meter must be on-line to perform this function.

Display Coords: Display image with mouse coordinates in bottom right corner. This function is useful to determine beam edges and centres if using manual settings. It also helps in ascertaining coordinate values for use with the 2D and 3D beam profile programs.

Parameters:

Load - Reload parameters from the specified parameter file. (right Amiga - L from keyboard)

Reset - Reset default parameters from the default parameter file "controlpar.def". (right Amiga - O from keyboard)

Modify - This function opens a sub-window containing all operating and controller parameters which may be modified on-line. These parameters are listed below. (right Amiga - M from keyboard)

Centre X - The X centre (in pixels) of the output beam. This value is used when the "Compute Centre" option is disabled to separate the image quadrants. It is also used when the "Beam

Steering" option is enabled as the X centre setpoint of the beam position.

Centre Y - The Y centre (in pixels) of the output beam. Used in a similar manner to "Centre X".

Min X - The left edge (in pixels) of the rectangular box which contains the output beam. If the beam moves outside this box, the information contained in the that part will be lost. (See Fig. C.1)

Max X - The right edge (in pixels) of the rectangular box which contains the output beam.

Min Y - The top edge (in pixels) of the rectangular box which contains the output beam.

Max Y - The bottom edge (in pixels) of the rectangular box which contains the output beam.

Ceiling - A parameter used by the frame grabber which specifies the upper (white) limit of digitizing. Any value above this limit will be considered as white (gray level 15). The ceiling can be moved from 100% to 25% of the video signal in 16 steps (i.e. 0 for 100% and 15 for 25%). (See Fig. C.2)

Floor - A parameter used by the frame grabber which specifies the loser (black) limit of digitizing. Any value below this limit will be considered as black (gray level 0). The floor can be moved from 0% to 75% of the video signal in 16 steps (i.e. 0 for 0% and 15 for 75%). (See Fig. C.2)

Uwand - Variable wand speed controller setpoint (in V) to manual adjustment of sampling percentage. Valid range of settings is from 7.5 (for high power levels) to 12 (for low power levels).

Period - Sampling period setting (in seconds). This value must be greater than 0.6 seconds (minimum control loop execution time).

Cutoff Freq - Cutoff frequency for first-order low pass and third-order Butterworth filters.

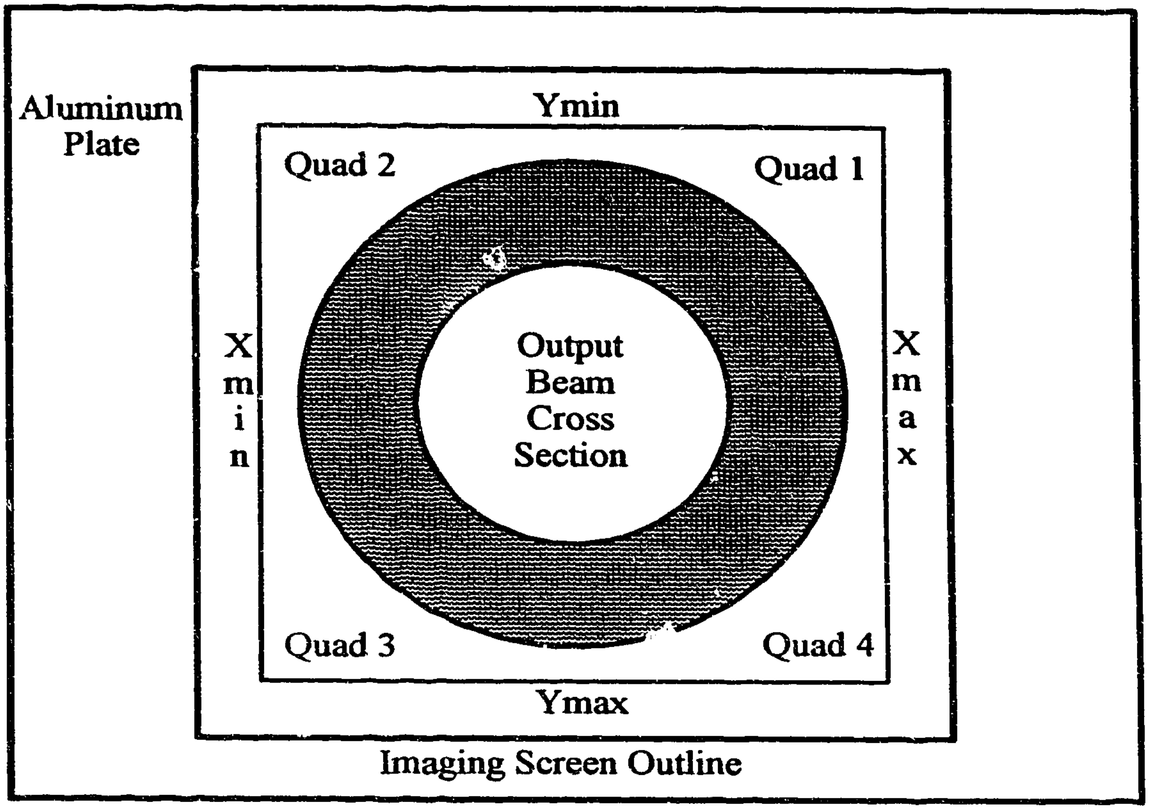


Fig. C.1. Definitions for image boundary settings.

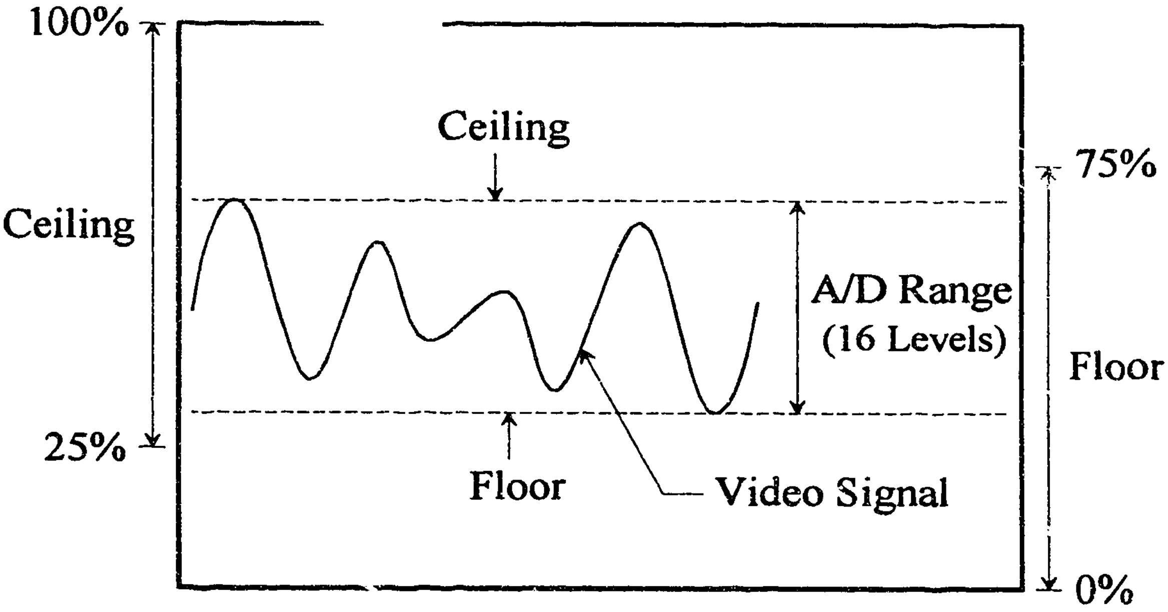


Fig. C.2. Digitizer settings to obtain optimal gray scale distribution.

Gain X - The X uniformity controller gain (specified in 0.1 μm steps per 1000 gray values of intensity). If the gain is set to 1.2, a 1.2 μm movement to the left would be made for a uniformity error of +10,000 gray values of intensity (10 kgv). One unit of motion for each motormic corresponds to one-tenth of a micron.

Gain Y - The Y uniformity controller gain (specified in 0.1 μm steps per 1000 gray values of intensity). If the gain is set to 1.2, a 1.2 μm movement downwards would be made for a uniformity error of +10,000 gray values of intensity (10 kgv).

Gain P - The proportional gain for the PI output power controller (specified in V/kW).

Min Err X - The minimum X uniformity motormic movement (specified in microns). If the required control action is less than this value, no correction to the uniformity control mirror will be made.

Min Err Y - The minimum Y uniformity motormic movement (specified in microns).

Min Err P - The minimum resolvable error in output power(specified in Watts). If the required power control action is less than this value, no correction to the laser's input current controller will be made.

Offset X - The X offset is used as a correction factor for a non-symmetrical beam along the X axis. The control algorithm will continuously attempt to obtain a symmetrical, annular output beam. However, if the best possible beam from the laser is not symmetrical, a offset intensity value may be added to one side to prevent the system from compensating. Setting this value properly may improve the overall uniformity of the beam. A positive value of the X offset will make the left side of the beam more intense.

Offset Y - The Y offset is used as a correction factor for a non-symmetrical beam along the Y axis. A positive value of the Y offset will make the bottom half of the beam more intense.

Ti P - The integral or reset time for the PI output power controller (specified in seconds).

Pow Conv Gain - The intensity to power conversion gain if a constant sampling wand is being used. This gain relates total screen intensity to the laser output power (specified in W/kgv).

Pow Conv Offset - This conversion offset is used to calibrate the intensity to power relationship of the imaging screen. This enables manual calibration of the screen when the power meter is off-line (identical to choosing the "Calibrate" item when the power meter is on-line).

OK - Save the current settings and close the parameter window.

Cancel - Ignore any changes to parameter values and close the parameter window.

Filenames - This function opens a sub-window which permits modification of various filenames.

Image Filename - The name of the image file to which the current image is saved if the "Overwrite" option is used. If the "Sequence" option is used, a file number is appended to this file name. For example, if the file name is "beam", the sequenced file names will be "beam.1", "beam.2", etc.

Data Filename - The name of the file to which all data will be stored. If the file already exists, the new data will be appended to this file.

Parameter Filename - The name of the file from which operating parameters are read, and to which parameters are stored.

OK - Save the current filenames and close the sub-level window.

Cancel - Ignore any changes to the filenames and close the sub-level window.

Save Pars/Opts - This function saves operating parameters and options to the specified parameter file.

C.2.4 Sub-Menu Item: Diagnostic

The Diagnostic sub-menu contains the commands necessary to automatically perform various sensitivity measurements, open loop step responses, and controller tuning. The menu items and corresponding descriptions are as follows:

Sensitivity:

None - Cancel any sensitivity measurements which are being performed.

Feedback - Performs uniformity error vs. motormic position measurements for feedback (convex) mirror.

Primary - Performs uniformity error vs. motormic position measurements for primary (concave) mirror.

Power - Performs output power vs. intensity measurements. The laser power meter must be on-line.

Wand - Performs intensity vs. wand speed controller setpoint measurements. The wand frequency and sampling percentage are also measured.

Position - Performs beam centre vs. motormic position measurements for selected mirror. The uniformity is maintained with the other optic.

O-L Step/Tune: This function opens a sub-window which enables various step responses to be performed. Following a step change, the program computes and displays the new controller gains, which may be used for tuning.

X-move - Performs the specified horizontal motormic movement (movement may be positive or negative).

Y-move - Performs the specified vertical motormic movement (movement may be positive or negative).

Power - Performs the specified setpoint change to the input power controller.

Wand - Performs the specified setpoint change to the variable wand speed controller.

OK - Close the sub-level window and execute the step change specified.

Cancel - Close the sub-level window and return to normal operation.

Monitor A/D:

None - Do not monitor any A/D channels

Power - Monitor the laser power meter reading.

Current - Monitor the laser current sensor reading.

Voltage - Monitor the laser voltage sensor reading.

C.3 Operating Procedure

After choosing the desired options and setting the various parameters, select the "Start" option from the Project Menu. The digitized image of the infrared detection screen will appear on the monitor along with a gadget which allows the ceiling and floor to be modified. Set the ceiling and floor to obtain the maximum range of gray levels for the output beam. However, the edges of the beam should always be distinct and clearly discern- able. If the background colour fades and the beam merges with the edges of the imaging screen, the ceiling and/or floor should be readjusted.

Once a satisfactory image of the output beam is obtained, place the mouse over the "OK" gadget and press the left mouse button. The program will commence and the data obtained will be displayed. The system performs many of the error checks during operation and displays the appropriate error messages. For critical errors, it will also speak out the error message. To terminate the program, select "Exit" from the Project Menu.

C.4 Additional Diagnostic and Utility Programs

The diagnostic and utility programs may all be executed simultaneously (on-line) with the main "CONTROL" program, if desired. A brief description of the programs follows. In each case, executing the program without any parameters, from a CLI window, will display an input window or list the necessary input parameters and their corresponding order. Executing the programs from the Amiga Workbench (double-clicking on the associated icons) will prompt the user for the necessary inputs.

PROFILE-3D: Displays a 3D intensity profile of a given image. The input image must be in raster (pixel) format. The resulting profile may be saved in Amiga IFF (bitplane) format. A profile may generated for any angle of rotation or elevation.

PROFILE-2D: Displays a 2D cross-sectional intensity profile of a given image. The input image must be in raster format. The resulting profile may be saved in Amiga IFF format. A profile may be taken at any angle through the image centre. Specifying an angle of 180° scans through all angles in increments of 1°.

COLORPIX: Colourizes a gray scale image. The input image must be in raster format, while resulting output is a colourized Amiga IFF image.

BPTOPIX: Converts an image from bitplane (Amiga IFF) to pixel (raster) format.

PIXTOBP: Converts an image from pixel (raster) to bitplane (Amiga IFF) format.

COORDS: Displays an Amiga IFF image with the mouse coordinates in the lower right corner. This program is useful for obtaining image boundaries and centres for non-standard beam cross sections.

SEEPIX: Displays a pixel (raster) format image on the screen.

SEEIFF: Displays an Amiga IFF format image on the screen.

COMPLEMENT: Complements and enhances any selected portion of an image.

INITPORT: Initializes all outputs of the data acquisition unit upon startup. This program may also be executed following startup to re-initialize the unit. It requires no input parameters.

IMPRINT: An IBM compatible program for printing images on a HP Laserjet (PCL) compatible printer. The program utilizes a special dithering technique which improves the print quality of beam cross sections.

APPENDIX D

SOFTWARE LISTINGS

The software for the on-line laser beam control and diagnostic system consisted of seven main program files, three assembly files, and several associated header files. All code was written in the ANSI standard C programming language and compiled with the Aztec C-68K compiler (for the Amiga). Only the files associated with the main control program are listed. Additional utility and beam profiling software are not listed due to the length of the code. The listed program files are as follows:

CONTROL.C: Main program which performs all error computation an control functions.

CONTGADG.C: Amiga specific code which controls all menu, window, and user input functions.

GRABFRAM.C: Functions to initialize and control the frame grabber.

DISPLAY.C: Performs all numerical and graphical display functions.

FILTER.C: Performs all data filtering.

P40.C: Functions necessary for initialization and control of the Proto-40K data acquisition unit.

TALK.C: Functions to access the Amiga's speech processor for verbal communication of error messages.

BPTOPIX.ASM: Bitplane to raster image conversion function.

QINTCENT.ASM: Computes quadrant intensities, centroids, and total beam area.

SETCAM.ASM: Sets desired camera input.

CONTROL.H: Header file which defines most constants and error limits.

CONTGADG.H: Header file which defines all window, menu, gadget, and Amiga specific structures (not listed due to its length).

DISPLAY.H: Header file which defines constants for the graphical output window.

LIVEBRARY.H: Header file supplied with the LIVE 2000 frame grabber for interfacing with the device (not listed).

MAKEFILE: File to compile and link all program files.

The diagnostic and utility programs are not listed due to the excessive length of the code. The files associated with each of these programs is summarized below:

PROFILE-3D.C: Main file for 3D intensity profile generating program.

PLOT3D.C: Functions for computation of 3D profile plot coordinates.

PROF3DGADG.C: Gadget defintions for 3D profile program.

PROFILE-2D.C: Main file for 2D cross-sectional profile generating program.

PLOT2D.C: Functions for computation of 2D profile plot coordinates.

PROF2DGADG.C: Gadget definitions for 2D profile program.

DISPW.C: Window display functions for 2D and 3D profiling programs.

WIND.C: Window definitions and interface functions for 2D and 3D profiling programs.

PIXTOBP.ASM: Pixel to bitplane conversion function.

COMPLEMENT.C: Main file to complement and enhance selected sections of images.

COMPGADG.C: Window display and gadget control functions for image complementing and enhancing program.

UTILSUBS.C: Various image processing utility functions used by all the above programs.

PIXTOIFF.C: Main file to convert images from pixel to Amiga IFF format.

IFFTOPIX.C: Main file to convert images from Amiga IFF to pixel format.

COORDS.C: Main file to display an Amiga IFF file with mouse coordinates in lower right corner.

COLORPIX.C: Main file to colourize a gray scale image in pixel format.

MAKEFILE: File to compile and link all diagnostic and utility program files.

D.1 CONTROL.C

```
/***************************************************************************
This program performs automatic alignment, beam positioning, automatic
power control, automatic sampling percentage, and performance monitoring
of the PIE-4 laser.

Filename: "CONTROL.C", Version: 2.1, Date: May 5, 1992

Note: Most variables are made global to decrease the execution time of
  the program for real-time operation.

***************************************************************************/

#include <fcntl.h>
#include <functions.h>
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <exec/types.h>
#include "control.h"

/* Variables and subroutines defined in "contgadg.c" */
extern void  cleanup(),errcleanup(),ignoremsg(),initgadgbufs(),
     dirhighlight(),powhighlight(short pwdir),setmirror(short mirror),
     setoptions(),updategadgbufs();
extern short dispargs(),checkargs(),checkmessage();

/* Variables and subroutines defined in "grabgram.c" */
extern short initlive(),chkvlevels(),getimage(),chkvideo(),
     *gotlive,colorflg;
extern long  camera;

/* Variables and subroutines defined in other source files */
extern short initaudio(UWORD rate,UWORD pitch,UWORD mode,UWORD
                         sex,UWORD sampfq),
     speakmsg(char *str);                /* in "talk.c" */

extern short  *p40_doutput,*p40_dinput;        /* in "p40.c" */
extern float  readADchan(short chan);
extern void   setDAchan(short chan, float volts),
     settime();

extern short  opensystimer();
extern double gettime();

extern void  quadintens();             /* in "quadint.asm" */
extern void  quadintcent();            /* in "qintcent.asm" */
extern void  quadintwcent();           /* in "qintwcen.asm" */
extern void  quadintucent();           /* in "qintucen.asm" */

extern void  dispcount(),dispdata(),    /* in "display.c" */
     printm(char *str),plotsignals();
extern short plotflg[];

extern float fcutoff;                 /* in "filter.c" */
extern void calcfilters(),filterdata(),
     calcperfilts(),permfiltdata();
extern short filtype;

/* Array of directions for displaying on screen (not used) */
char dirarray[8] = { 'U','D','L','R','U','D','R','L' };

/* Array of directions to define movement grid for uniformity measures */
short dirgrid[POSGRID] = { DOWNCV,DOWNCV,DOWNCV,DOWNCV,DOWNCV,
           UPCV,UPCV,UPCV,UPCV,UPCV,
           UPCV,UPCV,UPCV,UPCV,UPCV,
           DOWNCV,DOWNCV,DOWNCV,DOWNCV,DOWNCV,
           RIGHTCV,RIGHTCV,RIGHTCV,RIGHTCV,RIGHTCV,
           LEFTCV,LEFTCV,LEFTCV,LEFTCV,LEFTCV,
           LEFTCV,LEFTCV,LEFTCV,LEFTCV,LEFTCV,
           RIGHTCV,RIGHTCV,RIGHTCV,RIGHTCV,RIGHTCV };
/* these values control the direction of motion of the micrometers */
short  DOWN,      /* set values depending on mirror used */
     UP,
     RIGHT,     /* set values depending on mirror used */
     LEFT;

/* Array to compensate for opposite motion when mic is stopped.
  Amount of opposite motion depends on mic speed setting. The
  values are originally set to zero, but the values will be
  adaptively changed by the program depending on the speed of
  operation. */
short spoff[4] = { 0,0,0,0 };

/* Array to compensate for backlash in mics The corresponding
  value of motion is added to each mic if its direction is changed.
```

```
   Values are specified in tenths of microns */
short bklash[4] = { 48, 32, 137, 27 };

/* Array of coefficients relating the wand control signal to the total
   output power. The relationship is modelled as a cubic polynomial.
   Use first set for cubic approximation or second set for linear
   approximation. */
/* float cpwintgain[4] = { 2283.4, -719.4, 74.34, -2.480 }; */
float cpwintgain[4] = { -166.4, 20.97, 0.0, 0.0 };


/* Subroutines defined in this source file */
void  loadcounter(short data),counterwait(),addcounter(),initialize(),
      backcomp(),movemic(),printstart(),printheader(),
      readparamdata(),printdata(),startmicrun(),stopmicrun(),
      mirsenscontrol(),pwsenscontrol(),controlwand(),tunecontrol(),
      storedata(),possenscontrol(),controlsteer(),pauserun(),
      readcounter(),errormove(),checksteer(),convertADchan(),
      controlpower(),updatecount(),controlalign(),cleanbreak(),
      wandsenscontrol(),stepresponse(),measwandspeed();
short  control(),putimage(),chkcentre(),chknoise(),autocent(),
      saveparameters(),loadparameters(),resetparameters(),screenedge(),
      checkedge(),calcerror();

/* Variables defined in this source file */
static short  i,j,m,n;  /* temporary loop counter variables  */
unsigned char *image,  /* buffer to hold image raster */
       maxdev,       /* gray level ceiling of digitized image */
       minlev;       /* gray level floor of digitized image */
char  imagefile[40],  /* name of file to save image */
      datafile[40],   /* name of file to save data */
      parfile[40],    /* name of file to save or load parameters */
      filename[40],   /* name of file when saving sequences of images */
      tempbuf[80];    /* temporary buffer to create messages */
double runtime,        /* time since current run started in seconds */
      tunetime;        /* time since tuning step response was started */
float powmeter,  /* actual power of laser read from power meter (chan 0) */
      voltage,   /* voltage read from A/D channel 2 */
      current,   /* current read from current sensor (chan 1) */
      tsamp,     /* sampling period for pwoer and alignment control */
      eaxoft,    /* X-alignment controller error signal e(t) */
      eayoft,    /* Y-alignment controller error signal e(t) */
      upwr,      /* control signal output to laser power controller (D/A 1) */
      pwrgain,   /* power controller gain */
```

```
      wandgain,  /* wand controller gain */
      tipwr,     /* integral time constant for power controller */
      epoft,     /* power controller error signal e(t) */
      epoftm1,   /* power controller error signal e(t-1) */
      xalggain,  /* reciprocal of horizontal (X) gain for alignment */
      yalggain,  /* reciprocal of vertical (Y) gain for alignment */
      pwintgain, /* power vs. intensity gain for conversion for variable wand */
      pconvgain, /* power vs. intensity gain for conversion */
      tautime,   /* elapsed time since last control action for screen to settle */
      pwstepset, /* power setpoint change for open loop step responses */
      wnstepset, /* wand setpoint change for open loop step responses */
      initpwstep, /* initial steady-state power meter value for tuning step */
      finpwstep, /* final steady-state power meter value for tuning step */
      uwand,     /* control signal output to wand speed controller (D/A 2) */
      wandopen,  /* timer variable for measuring wand opening */
      wandper;   /* timer variable for measuring wand period */
short powerset,  /* desired power level at which to operate */
      actpower,  /* actual power of laser as computed from screen intensity */
      orgpower,  /* power when mirror and power sensitivity option started */
      q1power,   /* actual power in quadrant 1 */
      q2power,   /* actual power in quadrant 2 */
      q3power,   /* actual power in quadrant 3 */
      q4power,   /* actual power in quadrant 4 */
      ualgx,     /* control move to P-40 counter of amount to move X-mic. */
      ualgy,     /* control move to P-40 counter of amount to move Y-mic. */
      yalgperr,  /* actual power error in vertical (Y) direction */
      xalgperr,  /* actual power error in horizontal (X) direction */
      pmin,      /* minimum error in power before correction */
      xalgmin,   /* minimum X error in alignment before correction */
      yalgmin,   /* minimum Y error in alignment before correction */
      xcenth,    /* desired x-centre pixel of beam (original value) */
      ycenth,    /* desired y-centre pixel of beam (original value) */
      xcentd,    /* x-centre pixel of beam based on midpoint of diameters  */
      ycentd,    /* y-centre pixel of beam based on midpoint of diameters  */
      xcentp,    /* x-centre coordinate of previously grabbed image */
      ycentp,    /* y-centre coordinate of previously grabbed image */
      xcentr,    /* x-centroid pixel of annular laser beam image */
      ycentr,    /* y-centroid pixel of annular laser beam image */
      xcentg,    /* geometric (unweighted) x-centroid of beam image */
      ycentg,    /* geometric (unweightee) y-centroid of beam image */
      yalgdir,   /* vertical (Y) direction to move alignment micrometer */
      xalgdir,   /* horizontal (X) direction to move alignment micrometer */
      yalgdlast, /* last vertical (Y) direction moved of alignment micrometer */
      xalgdlast, /* last horizontal (X) direction moved of alignment micrometer */
```

```
yposdir,   /* vertical (Y) direction to move position micrometer */
xposdir,   /* horizontal (X) direction to move position micrometer */
yposdlast, /* last vertical (Y) direction moved of align micrometer */
xposdlast, /* last horizontal (X) direction moved of align micrometer */
micdir,    /* direction to move micrometer based on align. error */
mirdir,    /* mirror direction when obtaining sensitivity data */
micr,      /* specifies which micrometer of which mirror to move
              bit 2 - mirror, bit 1 - horiz. mic, bit 0 - vert. mic */
resmirror, /* resonator mirror which is being controlled */
centflg,   /* set flag to cause program to compute centre of beam */
imageflg,  /* set to store images to file (1=overwrite, 2=sequence) */
contflg,   /* set flag to enable automatic alignment */
steerflg,  /* set flag to enable automatic beam steering  */
runflg,    /* set flag to indicate a micrometer is in motion  (1=align 2=steer) */
powflg,    /* set flag to enable automatic power control */
wandflg,   /* set flag to enable automatic wand speed control */
dataflg,   /* set flag to store data to datafile */
filtflg,   /* set flag if data is to be filtered */
backflg,   /* indcates if backlash compensated for in last movement */
pwmetflg,  /* set flag if power meter connected */
sensflg,   /* set flag (1,2,3,4) to run corresponding diagnostic
              1 - convex mirror sensitivity measurements
              2 - concave mirror sensitivity measurements
              3 - power sensitivity measurements
              4 - beam uniformity measurements */
strmvflg,  /* set flag when beam steer optic is beaing moved */
smodflg,   /* flag to indicate stage of various sensitivity meas. */
curflg,    /* set flag if current sensor connected */
voltflg,   /* set flag if voltage sensor connected */
aligncnt,  /* number of consequetive samples with no alignment error */
wandcnt,   /* # of sample periods to wait after a wand control action */
stop,      /* flag to stop automatic control, and display menu */
finish,    /* flag to exit from program */
WB,        /* set flag to indicate program was started from workbench */
merit,     /* used by chknoise() as temporary variable */
xmin,      /* current left edge of laser beam image */
xmax,      /* current right edge of laser beam image */
ymin,      /* current top edge of laser beam image */
ymax,      /* current bottom edge of laser beam image */
xminh,     /* variable used to store original xmin */
xmaxh,     /* variable used to store original xmax */
yminh,     /* variable used to store original ymin */
ymaxh,     /* variable used to store original ymax */
xminp,     /* variable used to store previous xmin */
```

```
     xmaxp,      /* variable used to store previous xmax */
     yminp,      /* variable used to store previous ymin */
     ymaxp,      /* variable used to store previous ymax */
     xmini,      /* variable used to store inner diameter xmin */
     xmaxi,      /* variable used to store inner diameter xmax */
     ymini,      /* variable used to store inner diameter ymin */
     ymaxi,      /* variable used to store inner diameter ymax */
     gotedge,    /* temporary flag used by autocent() indicates edge found */
     filenum,    /* file number when saving a sequence of images */
     linecnt,    /* number of lines printed or displayed since headings */
     xylatch,    /* control word to be written to x-y table latch */
     cwlatch,    /* control word to be written to latch */
     ldlatch,    /* data word to be written to load latch */
     p40latch,   /* data word to be written to p40 output latch */
     micmove,    /* amount to move micrometers */
     xmoveset,   /* amount to move X-mic. for open loop step responses */
     ymoveset,   /* amount to move Y-mic. for open loop step responses */
     counter,    /* variable to read counter data */
     counterp,   /* variable to read hold previous counter data */
     initstep,   /* initial steady-state step response value for tuning */
     finalstep,  /* final steady-state step response value for tuning */
     tempcnt;    /* temporary variable to hold counter data */
long numbytes,   /* number of bytes written or read */
     imsize,     /* size of image in bytes (HEIGHT * WIDTH) */
     intensity,  /* sum of gray values of all pixels in image */
     beamarea,   /* number of all pixels in entire image */
     intensold,  /* intensity of image when centre last computed */
     intenslast, /* intensity of last image */
     intenserr,  /* error between actual and optimum intensity */
     quad1,      /* sum of all pixels in first quadrant (top-right) */
     quad2,      /* sum of all pixels in second quadrant (top-left) */
     quad3,      /* sum of all pixels in third quadrant (bottom-left) */
     quad4,      /* sum of all pixels in fourth quadrant (bottom-right) */
     yalgerr,    /* vertical intensity error (i.e. quad1+quad2-quad3-quad4) */
     xalgerr,    /* horizontal intensity error (i.e. quad1+quad4-quad3-quad2) */
     uniformity, /* sum of absolute differences of quadrant powers */
     momentx,    /* moment of intensity about x-axis */
     momenty,    /* moment of intensity about y-axis */
     unwmomx,    /* unweighted X-moment assuming uniform intensity */
     unwmomy,    /* unweighted Y-moment assuming uniform intensity */
     pwerr,      /* error in power */
     xalgoff,    /* compensation (offset) to subtract from right half */
     yalgoff,    /* compensation (offset) to subtract from bottom half */
     pconvoff,   /* power conversion offset (for auto-calibrate) */
```

```
    countvtx,   /* count of vertical mic of convex mirror */
    counthzx,   /* count of horizontal mic of convex mirror */
    countvtv,   /* count of vertical mic of concave mirror */
    counthzv,   /* count of horizontal mic of concave mirror */
    countvtxh,  /* starting vertical mic cnt of convex  for manual motion */
    counthzxh,  /* starting horizontal mic cnt of convex for manual motion */
    countvtvh,  /* starting vertical mic cnt of concave for manual motion */
    counthzvh,  /* starting horiz. mic cnt of concave for manual motion */
    countvto,   /* starting count of vertical mic when meas. sensitivity */
    counthzo,   /* starting count of horizontal mic when meas. sensitivity */
    countvtxchk, /* vertical convex mic count since centre last computed */
    counthzxchk, /* horizontal convex mic count since centre last computed */
    countvtvchk, /* vertical concave mic count since centre last computed */
    counthzvchk, /* horiz. concave mic count since centre last computed */
    file1;      /* file descriptor for image file (unbuffered) */
FILE *file2,    /* file pointer for data file (buffered) */
    *file3;     /* file pointer for parameter file (buffered) */

void main(int argc, char *argv[])
  {
  if (p40_init() == 0)
    { printm("Cannot initialize Proto-40K board"); exit(0); }
  initialize();

  if (argc == 0) WB = 1; else WB = 0;
  if (*argv[1] == '?')
    { printm("Help messages not yet implemented"); exit(0); }

  /* display and initialize control and plotting windows */
  if (dispargs()) { cleanup(1); exit(0); }

  /* initialize audio handler */
  if (initaudio((UWORD)RATE,(UWORD)PITCH,(UWORD)MODE,
                    (UWORD)SEX,(UWORD)SAMPFRQ))
    { cleanup(1); exit(0); }

  /* initialize LIVE hardware */
  if (initlive()) { cleanup(1); exit(0); }

  /* open system timer device */
  if (opensystimer()) { cleanup(1); exit(0); }

  while (control()) errcleanup();
  cleanup(0);
```

```
  exit(0);
  }

void initialize(void)
  {
  cwlatch = CLDEFLT;        /* set control word to default setting */
  *p40_doutput = cwlatch;
  cwlatch |= ENABLT;
  *p40_doutput = cwlatch;
  cwlatch |= DCLOCKH;
  *p40_doutput = cwlatch;
  cwlatch &= DCLOCKL;
  *p40_doutput = cwlatch;
  cwlatch &= DISBLT;
  *p40_doutput = cwlatch;
  runflg = 0;               /* no micrometers are currently running */
  loadcounter((short)0);    /* load mic. controller counter with zero */
  xylatch = XLATCH;         /* initialize default latch settings */
  p40latch = PLATCH;
  gotlive = NULL;           /* indicates LIVE hardware not initialized */
  file1 = 0L;               /* indicates output image file not opened */
  file2 = NULL;             /* indicates output data file not opened */
  image = NULL;             /* indicates memory not allocated for image */
  filenum = 0;              /* starting file number for image sequencing */
  strcpy(parfile,"pie:controlpar.new");  /* set parameter filename */

  /* The following variable are stored in the parameter file.  If the
     parameter file is not present, these are the values are values.   */
  xalggain = 1.0; yalggain = 1.0; pwrgain = 1.0; tipwr = 5.0;
  xalgmin = 10; yalgmin = 10; pmin = 50;
  xalgoff = 0; yalgoff = 0; pconvoff = 0;
  pconvgain = 15.0;
  powerset = 2000;                /* power in Watts */
  maxlev = 8; minlev = 6;
  xminh = 64;  xmaxh = 255;       /* beam size of image */
  yminh = 4; ymaxh = 195;
  xcenth = 160;             /* assume beam centre is at centre of screen */
  ycenth = 100;
  strcpy(imagefile,"ram:beam");       /* set default filenames */
  strcpy(datafile,"ram:rundata");
  contflg = steerflg = powflg = 0;  /* do not control beam or power */
  wandflg = 0;                  /* do not control wand */
  dataflg = imageflg = 0;       /* do not save images or data */
  centflg = 0;                  /* do not compute centre */
```

```
colorflg = 1;                  /* set LIVE screen to color mode */
pwmetflg = 0;                    /* assume power meter not connected */
curflg = 0;                    /* assume current sensor not connected */
voltflg = 0;                   /* assume voltage sensor not connected */
filtflg = 0;                   /* assume no filtering */
for (i=0; i<=NUMSIGS; i++) plotflg[i] = 0;   /* do not plot signals */
camera = TOPCAM;

xmin = xminh;  xmax = xmaxh; /* set actual beam size of image to default */
ymin = yminh; ymax = ymaxh;
xcentd = xcenth; ycentd = ycenth; /* set actual centre to desired centre */

sensflg = smodflg = 0;   /* disable sensitivity diagnostic measurements */

countvtx = counthzx = 0;          /* set mic. counts to zero */
countvtv = counthzv = 0;
micr &= CONVEX;                 /* assume convex mirror being used */
DOWN = DOWNCX; UP = UPCX;
LEFT = LEFTCX; RIGHT = RIGHTCX;
yalgdlast = INVALID;   /* set last direction to invalid number so */
xalgdlast = INVALID;   /* no backlash compensation on first move */
yposdlast = INVALID;
xposdlast = INVALID;

pwintgain = 47.0;
actpower = q1power = q2power = q3power = q4power = 0;
powmeter = current = voltage = 0.0;
runtime = 0.0;

tsamp = 1.0;
ualgx = ualgy = 0;
upwr = PWRSETD; uwand = WNDSETD;
wandopen = 0.0; wandper = 1.0; wandcnt = 0;
epoft = 0.0; epoftm1 = 0.0;

loadparameters();              /* load parameters from file */
setDAchan((short)POWCHAN,upwr);
setDAchan((short)WNDCHAN,uwand);
filtype = MOVEAVG;
calcpermfilts();
if (filtflg) { calcfilters(); filtflg = 1; }
xmoveset = ymoveset = 0;
}
```

```
/* This function controls the operation of the program */
short control(void)
  {
  if (checkargs()) return(0);
  imsize = HEIGHT * WIDTH;    /* allocate memnory for image raster */
  image = (unsigned char *) malloc(imsize);
  if (image == NULL)
    { printm("Not enough memory for image"); return(1); }
  if (chkvideo()) return(1);        /* no video signal present */
  settime();             /* set starting time for run */
  runtime = gettime() + tsamp;  /* set so data sampled immediately */
  counter = *p40_dinput;       /* set counter to current value */
  finish = 0;          /* break out of outer loop when set to 1 */

  do     /* outer loop controls initialization and error checking */
    {
    stop = 0; /* break out of inner loop when 1, both loops when 2 */

    if (chkvlevels()) return(1);   /* set frame grabber levels */
    else updategadgbufs();
    if (getimage(image))        /* and grab one image */
      { printm("No video signal present");
        speakmsg("no video signal"); cleanbreak(); return(1); }

    if (screenedge())   /* check if beam touching screen edges */
      {
      printm("Restoring default screen boundaries");
      xminh = xminp; xmaxh = xmaxp;   /* reset previous values */
      yminh = yminp; ymaxh = ymaxp;
      }
    countvtxchk = countvtx + MAXMOVEY;  /* set so autocent() called */
    counthzxchk = counthzx + MAXMOVEX;
    intensold = 0;
    if (!centflg)   /* set beam edges if centre not to be computed */
      {
      xmin = xminh; xmax = xmaxh;
      ymin = yminh; ymax = ymaxh;
      }
    linecnt = MAXLINE + 1;    /* print column titles initially */

    do    /* inner loop grabs images and performs control actions */
      {
      if (dataflg && !file2)   /* if not open, open data file */
        {
```

```
    file2 = fopen(datafile,"a");
    if (file2 == 0)
      { printm("Cannot open output data file");
        cleanbreak(); return(1); }
    printstart();     /* print header and column titles*/
    }
  if (imageflg == 1 && !file1)    /* open output image file*/
    {
    file1 = open(imagefile,O_WRONLY+O_CREAT);
    if (file1 == -1)
      { printm("Cannot open output image file");
        file1 = 0; cleanbreak(); return(1); }
    }
  while ((gettime() - runtime) < tsamp);
  runtime = gettime();    /* compute time data was sampled */
  if (sensflg == 6) stepresponse();
  counter = *p40_dinput;
  convertADchan();    /* read and convert A/D voltages */
  if (getimage(image)) { cleanbreak(); break; }

  if (centflg)    /* if centre to be computed, check if */
    if (chkcentre())   /* new centre needs to be computed */
      if (autocent())
        { if (!stop) stop = checkmessage(); continue; }

  if (calcerror())       /* calc. intensities and errors */
    { if (!stop) stop = checkmessage(); continue; }
  permfiltdata();
  if (filtflg) filterdata();
  if (plotflg[NUMSIGS]) plotsignals();

  if (linecnt >= MAXLINE) printheader();

  /* Perform normal alignment and power control (if required) */
  if (!sensflg) {
    controlalign(); controlwand(); controlpower();
    measwandspeed();
    storedata(); }
  if (sensflg == 1 || sensflg == 2) mirsenscontrol();
  if (sensflg == 3) pwsenscontrol();
  if (sensflg == 4) wandsenscontrol();
  if (sensflg == 5) possenscontrol();
  if (sensflg == 7) tunecontrol();
```

```
      printdata();         /* print data in CLI window */
      dispdata();  /* display data and plot signals in windows */

      if (imageflg)         /* store image if required */
        if (putimage()) { cleanbreak(); break; }
      linecnt++;
      if (!stop) stop = checkmessage();
      }
    while (!stop);  /* end of inner loop */

    if (file2) { fclose(file2); file2 = 0; }
    if (file1) { close(file1); file1 = 0; }
    if (stop > 1) return(0);
    finish = checkargs();       /* wait for message from window */
    }
  while (!finish);  /* end of outer loop */
  return(0);
  }

void cleanbreak(void)
  {
  if (runflg) pauserun();
  ignoremsg();
  }

/* This function converts the A/D channel voltages to the desired units */
void convertADchan(void)
  {
  if (pwmetflg) powmeter = (readADchan((short)PWMCHAN) * PWSCALE) + PWOFF;
  if (curflg) current = readADchan((short)CURCHAN) * CRSCALE + CROFF;
  if (voltflg) voltage = readADchan((short)VLTCHAN) * VLSCALE + VLOFF;
  }

/* This function stores an image raster to a file. If the sequence flag is
   set, the function appends a number to the file name. Otherwise, the
   image file is overwritten. */
short putimage(void)
  {
  if (imageflg == 2)
    {
    if (filenum >= MAXFILE) filenum = 0;
    filenum++;
    sprintf(filename,"%s.%d",imagefile,filenum);
    file1 = open(filename,O_WRONLY+O_CREAT);
    if (file1 == -1)
      { printm("Cannot open output image file"); file1 = 0; return(1); }
    }
  else if (lseek(file1,0,0)) { printm("File seek error"); return(1); }
  numbytes = write(file1,image,imsize);
  if (imageflg == 2) { close(file1); file1 = 0; }
  if (numbytes != imsize)
    { printm("File write error - out of memory/disk space"); return(1); }
  return(0);
  }

/* This function checks if the mics. have been moved by more than a given
   amount, or the power has been increased by more than a certain amount.
   If so, the function returns (1) so the beam centre will be recomputed */
short chkcentre(void)
  {
  /* Reinstate this code to for the function to operate */
  /*****************************************************
  if ((abs(countvtx-countvtxchk) > SAFEVER) ||
     (abs(counthzx-counthzxchk) > SAFEHER) ||
     (abs(intensity-intensold) > DINTENS))
    {
    countvtxchk = countvtx;
    counthzxchk = counthzx;
    intensold = intensity;
    return(1);
    }
  else return(0);
  *****************************************************/
  return(1);              /* centre always computed */
  }

/* This function computes the quadrant intensities and centroids, and
   performs the necessary intensity to power conversions. */
short calcerror(void)
  {
  /* The function quadintcent() computes the quadrant intensities and
     calculates the X and Y moments needed to compute the weighted and
     unweighted centroids of the beam. The function quadintwcent()
     computes the quadrant intensities and calculates the only the
     X and Y moments needed to compute weighted centroid of the beam.
     The function quadintwcent() computes the quadrant intensities and
     calculates the only the X and Y moments needed to compute unweighted
     centroid of the beam. If either or both of the centroids is not
     required, use the appropriate function to same execution time. */
  /**********
  quadintens();
  quadintwcent();
  quadintucent();
  **********/
  quadintcent();
  intensity = quad1 + quad2 + quad3 + quad4;
  if (intensity < NOBEAM) {
    printm("Beam image not present or too weak - program halted");
    speakmsg("no beam present"); return(1); }

  intenserr = OPTINTN - intensity;
  yalgerr = quad1 + quad2 - quad3 - quad4;  /* compute alignment errors */
  xalgerr = quad1 + quad4 - quad2 - quad3;
  uniformity = abs(quad1-quad2) + abs(quad1-quad3) + abs(quad1-quad4)
       + abs(quad2-quad3) + abs(quad2-quad4) + abs(quad3-quad4);

  /* convert intensity to power (in Watts) */
  if (wandflg) {
    pwintgain = cpwintgain[3]*uwand*uwand*uwand + cpwintgain[2]*uwand*uwand
           + cpwintgain[1]*uwand + cpwintgain[0];
    pwintgain = 1.0/pwintgain; }
  else pwintgain = pconvgain;
  actpower = ((float)(intensity))*pwintgain + pconvoff + 0.5;
  if (actpower < 0) actpower = 0;
  q1power = ((float)quad1 / intensity) * actpower;
  q2power = ((float)quad2 / intensity) * actpower;
  q3power = ((float)quad3 / intensity) * actpower;
  q4power = ((float)quad4 / intensity) * actpower;
  xalgperr = q1power+q4power-q2power-q3power;    /* compute X and Y */
  yalgperr = q1power+q2power-q3power-q4power;    /* alignment errors */
  pwerr = powerset - actpower;          /* compute power error */
  xcentr = momentx / intensity + 0.5;    /* compute weighted moments */
  ycentr = momenty / intensity + 0.5;    /* comment out if not needed */
  xcentg = unwmomx / beamarea + 0.5;    /* compute unweighted moments */
  ycentg = unwmomy / beamarea + 0.5;    /* comment out if not needed */
  return(0);
  }

/* This function performs the backlash compensation for the micrometers */
void backcomp(void)
  {
  if (micr&CONCAVE) {
```

```
  if ((micdir == xalgdir) && (xalgdir != xalgdlast))
  if (xalgdlast == INVALID) xalgdlast = xalgdir;
  else {
    micmove += bklash[micr>>1];
    xalgdlast = xalgdir;
    backflg = (micr>>1) + 1; }
  else if ((micdir == yalgdir) && (yalgdir != yalgdlast))
  if (yalgdlast == INVALID) yalgdlast = yalgdir;
  else {
    micmove += bklash[micr>>1];
    yalgdlast = yalgdir;
    backflg = (micr>>1) + 1; } }
else {
  if ((micdir == xposdir) && (xposdir != xposdlast))
  if (xposdlast == INVALID) xposdlast = xposdir;
  else {
    micmove += bklash[micr>>1];
    xposdlast = xposdir;
    backflg = (micr>>1) + 1; }
  else if ((micdir == yposdir) && (yposdir != yposdlast))
  if (yposdlast == INVALID) yposdlast = yposdir;
  else {
    micmove += bklash[micr>>1];
    yposdlast = yposdir;
    backflg = (micr>>1) + 1; } }
}

/* This function sets the mic. driver controller counter to a specified
   value and starts the specified mic. running. */
void movemic(void)
{
backcomp();
if ((micmove+spoff[micr>>1]) < MINCNT) spoff[micr>>1] = 0;
loadcounter((short)(micmove + spoff[micr>>1]));
counterp = (micmove + spoff[micr>>1]) & 0xfffc;
cwlatch &= CLRMIC & CLRMIR;
if (micdir & RUNNEG) cwlatch |= CNTDOWN;
else cwlatch &= CNTUP;
cwlatch |= (DPRESET | micdir | (micr & CONCAVE));
*p40_doutput = cwlatch;
dirhighlight();
cwlatch |= ENABLT; *p40_doutput = cwlatch;
cwlatch |= DCLOCKH; *p40_doutput = cwlatch;
cwlatch &= DCLOCKL; *p40_doutput = cwlatch;
```

```
 cwlatch &= DCLEAR; *p40_doutput = cwlatch;
 cwlatch &= DISBLT; *p40_doutput = cwlatch;
 }

short chknoise(void)
 {
 merit = 0;
 for (m=-1; m<=1; m++)
   for (n=-1; n<=1; n++)
   if (*(image+(i+m)*WIDTH+j+n) < MAXLEV) merit++;
 if (merit >= BTHRESH) return(1);
 else return(0);
 }

short autocent(void)
 {
 xminp = xmin; xmaxp = xmax;  /* save previous values */
 yminp = ymin; ymaxp = ymax;
 xcentp = xcentd; ycentp = ycentd;
 xmin = xminh; xmax = xmaxh;  /* set to original boundary values */
 ymin = yminh; ymax = ymaxh;
 gotedge = 0;
 i = ymin + 1;
 while ((!gotedge) && (i < ymax))
   {
   j = xmin + 1;
   while (!gotedge && (j < xmax))
     {
     if (*(image+i*WIDTH+j) < MAXLEV) gotedge = chknoise();
     j++;
     }
   i++;
   }
 if (gotedge) ymin = i - 1;
 else if (!sensflg)
   { printm("No beam present - check possible causes");
     speakmsg("no beam present"); return(1); }
 if (ymin < yminh) ymin = yminh;
 gotedge = 0;
 i = ymax - 1;
 while (!gotedge && (i > ymin))
   {
   j = xmin;
   while (!gotedge && (j < xmax))
```

```
      {
      if (*(image+i*WIDTH+j) < MAXLEV) gotedge = chknoise();
      j++;
      }
    i–;
    }
  if (gotedge) ymax = i + 1;
  else if (!sensflg)
    { printm("No beam present - check possible causes");
      speakmsg("no beam present"); return(1); }
  if (ymax > ymaxh) ymax = ymaxh;
  gotedge = 0;
  j = xmin + 1;
  while (!gotedge && (j < xmax))
    {
    i = ymin;
    while (!gotedge && (i < ymax))
      {
      if (*(image+i*WIDTH+j) < MAXLEV) gotedge = chknoise();
      i++;
      }
    j++;
    }
  if (gotedge) xmin = j - 1;
  else if (!sensflg)
    { printm("No beam present - check possible causes");
      speakmsg("no beam present"); return(1); }
  if (xmin < xminh) xmin = xminh;
  gotedge = 0;
  j = xmax - 1;
  while (!gotedge && (j > xmin))
    {
    i = ymin;
    while (!gotedge && (i < ymax))
      {
      if (*(image+i*WIDTH+j) < MAXLEV) gotedge = chknoise();
      i++;
      }
    j–;
    }
  if (gotedge) xmax = j + 1;
  else if (!sensflg)
    { printm("No beam present - check po  ble causes");
      speakmsg("no beam present"); return(1); }
```

```
  if (xmax > xmaxh) xmax = xmaxh;

  xcentd = (xmax - xmin) / 2.0 + xmin + 0.5;
  ycentd = (ymax - ymin) / 2.0 + ymin + 0.5;

  if ((xmax - xmin) < XDIAM)
    {
    xcentd = xcentp;
    xmin = xminp;
    xmax = xmaxp;
    printm("X-diameter too small - X-centre set to previous value");
    }
  else if (abs(xcentd - xcenth) > CENTXER)
    {
    xcentd = xcentp;
    xmin = xminp;
    xmax = xmaxp;
    printm("X-centre too far from midpoint - set to previous value");
    }
  if ((ymax - ymin) < YDIAM)
    {
    ycentd = ycentp;
    ymin = yminp;
    ymax = ymaxp;
    printm("Y-diameter too small - Y-centre set to previous value");
    }
  else if (abs(ycentd - ycenth) > CENTYER)
    {
    ycentd = ycentp;
    ymin = yminp;
    ymax = ymaxp;
    printm("Y-centre too far from midpoint - set to previous value");
    }

  gotedge = 0;
  i = ycentd;
  j = xcentd;
  while ((!gotedge) && (i < ymax))
    {
    if (*(image+i*WIDTH+xcentd) < MAXLEV) gotedge = chknoise(;
    i++;
    }
  if (gotedge)
    {
```

```
    ymaxi = i - 1;
    gotedge = 0;
    i = ycentd;
    while ((!gotedge) && (i > ymin))
      {
      if (*(image+i*WIDTH+xcentd) < MAXLEV) gotedge = chknoise();
      i–;
      }
    if (gotedge) ymini = i + 1;
    else { ymini = ymin; ymaxi = ymax; }
    }
  else { ymaxi = ymax; ymini = ymin; }
  gotedge = 0;
  i = ycentd;
  j = xcentd;
  while ((!gotedge) && (j < xmax))
    {
    if (*(image+ycentd*WIDTH+j) < MAXLEV) gotedge = chknoise();
    j++;
    }
  if (gotedge)
    {
    xmaxi = j - 1;
    gotedge = 0;
    j = xcentd;
    while ((!gotedge) && (j > xmin))
      {
      if (*(image+ycentd*WIDTH+j) < MAXLEV) gotedge = chknoise();
      j–;
      }
    if (gotedge) xmini = j + 1;
    else { xmini = xmin; xmaxi = xmax; }
    }
  else { xmaxi = xmax; xmini = xmin; }
  xcentd = (((xmaxi - xmini) / 2.0 + xmini) + xcentd) / 2.0 + 0.5;
  ycentd = (((ymaxi - ymini) / 2.0 + ymini) + ycentd) / 2.0 + 0.5;
  updategadgbufs();
  return(0);
  }

void printstart(void)
  {
  printf("****** New Parameters ******\n\n");
  printf("Vgain=%6.2f  Hgain=%6.2f  Pgain=%6.1f  Xcent= %4d  Ycent= %4d\n",
                yalggain,xalggain,pwrgain,xcenth,ycenth);
  printf("Vmin =%6.1f  Hmin =%6.1f  Pmin =%6d  Ceil = %4d  Floor= %4d\n",
                yalgmin/10.0,xalgmin/10.0,pmin,maxlev,minlev);
  printf("Voff =%6d  Hoff =%6d  Ti P =%6.1f  Pcnvg= %4.1f  Pcnvo=%5d\n",
              yalgoff,xalgoff,tipwr,pconvgain*KILO,pconvoff);
  if (dataflg) {
    fprintf(file2,"\r\n****** New Parameters ******\r\n\r\n");
    fprintf(file2,"Vgain=%6.2f  Hgain=%6.2f  Pgain=%6.1f  Xcent= %4d  Ycent= %4d\r\n",
                yalggain,xalggain,pwrgain,xcenth,ycenth);
    fprintf(file2,"Vmin =%6.1f  Hmin =%6.1f  Pmin =%6d  Ceil = %4d  Floor= %4d\r\n",
                yalgmin/10.0,xalgmin/10.0,pmin,maxlev,minlev);
    fprintf(file2,"Voff =%6d  Hoff =%6d  Ti P =%6.1f  Fcnvg= %4.1f  Pcnvo=%5d\r\n",
                yalgoff,xalgoff,tipwr,pconvgain*KILO,pconvoff);
    fprintf(file2,"\r\n Time   Y1count X1count Y2count X2count Quad1 Quad2 Quad3
                    Quad4 Intens Yerror Xerror Y-range ");
    fprintf(file2,"X-range Ycn Xcn Ycr Xcr Ycg Xcg MicMv D Upowr Uwand Power Curm
                    Volts Wfreq Wopen Bk Unifrm\r\n");
    fprintf(file2,"----- ----- ----- ----- ----- ---- ---- ---- ---- ----- ----- ----- ----- ");
    fprintf(file2,"----- -- -- -- -- -- -- ---- - ---- ---- ---- ---- ---- ---- ---- - -----\r\n");
                    }
  }

void printheader(void)
  {
  printf("\nTime  Xmove   Ymove  Wfreq Wopen Xerror Yerror Fn\n");
  printf(  "---- ------ ------ ---- ---- ----- ----- --\n");
  linecnt = 0;
  }

void printdata(void)
  {
  printf("%4.2f ",(gettime()-runtime));
  if (micr&CONCAVE)
    printf("%5.1f %c %5.1f %1c ",ualgx/10.0,dirarray[(micr&CONCAVE)|xalgdir],
              ualgy/10.0,dirarray[(micr&CONCAVE)|yalgdir]);
  else
    printf("%5.1f %c %5.1f %1c ",ualgx/10.0,dirarray[(micr&CONCAVE)|xposdir],
              ualgy/10.0,dirarray[(micr&CONCAVE)|yposdir]);
  printf("%5.2f %5.2f %6d %6d %2d\n",1.0/wandper,wandopen,xalgerr,yalgerr,filenum);
  }

void storedata(void)
  {
  if (!dataflg) { backflg = 0; return; }
```

```
fprintf(file2,"%7.1f %7.1f %7.1f %7.1f %7.1f ",runtime,
    countvtx/10.0,counthzx/10.0,countvtv/10.0,counthzv/10.0);
fprintf(file2,"%5d %5d %5d %5d %6d ",quad1,quad2,quad3,quad4,intensity);
fprintf(file2,"%6d %6d ",yalgerr,xalgerr);
fprintf(file2,"%3d %3d %3d %3d %3d %3d %3d %3d ",ymin,ymax,
        xmin,xmax,ycentd,xcentd,ycentr,xcentr);
fprintf(file2,"%3d %3d %5.1f %1c
                 ",ycentg,xcentg,micmove/10.0,dirarray[(micr&CONCAVE)|micdir]);
fprintf(file2,"%5.2f %5.2f ",upwr,uwand);
fprintf(file2,"%5.2f %5.2f %5.2f %5.2f %5.2f %2d %6d\r\n",powmeter,
      current,voltage,1.0/wandper,wandopen,backflg,uniformity);
backflg = 0;
}

short saveparameters(void)
 {
 file3 = fopen(parfile,"w");
 if (file3 == NULL)
   { printm("Cannot open parameter file"); ignoremsg(); return(1); }
 fprintf(file3,"%4.2f %4.2f %7.5f %3d %3d %3d %6d %6d %6d\n",
       xalggain,yalggain,pconvgain,xalgmin,yalgmin,pmin,xalgoff,yalgoff,pconvoff);
 fprintf(file3,"%5d %2d %2d\n",
          powerset,maxdev,minlev);
 fprintf(file3,"%3d %3d %3d %3d %3d %3d\n",
           xminh,xmaxh,yminh,ymaxh,xcenth,ycenth);
 fprintf(file3,"%s %s\n",imagefile,datafile);
 fprintf(file3,"%1d %1d %1d %1d %1d %1d %1d %1d %1d %1d %1d\n",
      contflg,steerflg,powflg,wandflg,dataflg,imageflg,centflg,
      colorflg,micr&CONCAVE,camera,filtflg);
 for (i=0; i<=NUMSIGS; i++)
   fprintf(file3,"%1d ",plotflg[i]);
 fprintf(file3,"\n");
 fprintf(file3,"%1d %1d %1d\n",pwmetflg,curflg,voltflg);
 fprintf(file3,"%4.1f %6.2f %6.2f %5.2f %5.2f
                      %4.2f",tsamp,pwrgain,tipwr,upwr,uwand,fcutoff);
 fclose(file3);
 return(0);
 }

short loadparameters(void)
 {
 file3 = fopen(parfile,"r");
 if (file3 == NULL)
   { printm("Cannot open parameter file"); ignoremsg(); return(1); }
```

```
 readparamdata();
 fclose(file3);
 return(0);
 }

short resetparameters(void)
 {
 file3 = fopen("pie:controlpar.def","r");
 if (file3 == NULL)
   { printm("Cannot open parameter file"); ignoremsg(); return(1); }
 readparamdata();
 fclose(file3);
 return(0);
 }

void readparamdata(void)
 {
 long check;
 short tmp1,tmp2;

 check = fscanf(file3,"%f %f %f %hd %hd %hd %ld %ld %ld",

                 &xalggain,&yalggain,&pconvgain,&xalgmin,&yalgmin,&pmin,&xalgoff,
                 &yalgoff,&pconvoff);
 if (check != 9) printm("Error reading parameters - check new values");
 check = fscanf(file3,"%hd %hd %hd",
          &powerset,&tmp1,&tmp2);
 if (check != 3) printm("Error reading parameters - check new values");
 maxdev = (char)tmp1; minlev = (char)tmp2;
 check = fscanf(file3,"%hd %hd %hd %hd %hd %hd",&xminh,&xmaxh,
            &yminh,&ymaxh,&xcenth,&ycenth);
 if (check != 6) printm("3Error reading parameters - check new values");
 check = fscanf(file3,"%s %s",imagefile,datafile);
 if (check != 2) printm("Error reading parameters - check new values");
 check = fscanf(file3,"%hd %hd %hd %hd %hd %hd %hd %hd %hd %ld %hd",
      &contflg,&steerflg,&powflg,&dataflg,&wandflg,&imageflg,&centflg,
      &colorflg,&tmp1,&camera,&filtflg);
 if (check != 11) printm("Error reading parameters - check new values");
 micr |= tmp1&CONCAVE;
 for (i=0; i<=NUMSIGS; i++)
   {
   check = fscanf(file3,"%hd ",&plotflg[i]);
   if (check != 1)
     printm("Error reading parameters - check new values");
   }
  check = fscanf(file3,"%hd %hd %hd",&pwmetflg,&curflg,&voltflg);
  if (check != 3) printm("Error reading parameters - check new values");
  check = fscanf(file3,"%f %f %f %f %f %f",&tsamp,
          &pwrgain,&tipwr,&upwr,&uwand,&fcutoff);
  if (check != 6) printm("Error reading parameters - check new values");
  initgadgbufs();
  setoptions();
  }

void counterwait(void)
  {
  counter = *p40_dinput;
  Delay(CNTWAIT);
  while (counter != *p40_dinput)
    {
    counter = *p40_dinput; Delay(CNTWAIT);
    }
  }

void addcounter(void)
  {
  switch (micr)
    {
    case VTMICX : countvtx += counter;
          break;
    case HZMICX : counthzx += counter;
          break;
    case VTMICV : countvtv += counter;
          break;
    case HZMICV : counthzv += counter;
          break;
    }
  }

void loadcounter(short data)
  {
  if ((data < MINCNT) && runflg)
    {
    printm("Attempt to load counter with invalid count");
    printm("Counter set to minimum value");
    data = MINCNT;
    }
  ldlatch = (data >> 2) | LLATCH; *p40_doutput = ldlatch;
```

```
  ldlatch |= ENABLT; *p40_doutput = ldlatch;
  ldlatch &= DISBLT; *p40_doutput = ldlatch;
  cwlatch |= CNTLOAD; *p40_doutput = cwlatch;
  cwlatch |= ENABLT; *p40_doutput = cwlatch;
  cwlatch |= CNTCLKH; *p40_doutput = cwlatch;
  cwlatch &= CNTCLKL; *p40_doutput = cwlatch;
  cwlatch &= CNTENAB; *p40_doutput = cwlatch;
  cwlatch &= DISBLT; *p40_doutput = cwlatch;
  }

short screenedge(void)
  {
  xminp = xminh; xmaxp = xmaxh;  /* save previous values */
  yminp = yminh; ymaxp = ymaxh;
  i = 0;
  gotedge = 0;
  while ((i < HEIGHT) && !gotedge)
    {
    j = 0;
    while ((j < WIDTH) && !gotedge)
      {
      if (*(image+i*WIDTH+j) == MAXLEV) gotedge = checkedge();
      j++;
      }
    i++;
    }
  if (!gotedge)
    { printm("Could not find top edge of screen"); yminh = yminp; }
  else yminh = i;
  if (yminh < MINTOP) yminh = MINTOP;
  i = HEIGHT - 1;
  gotedge = 0;
  while ((i >= yminh) && !gotedge)
    {
    j = 0;
    while ((j < WIDTH) && !gotedge)
      {
      if (*(image+i*WIDTH+j) == MAXLEV) gotedge = checkedge();
      j++;
      }
    i–;
    }
  if (!gotedge)
    { printm("Could not find bottom edge of screen"); ymaxh = ymaxp; }
```

```
else ymaxh = i;
if (ymaxh > MAXBOT) ymaxh = MAXBOT;
j = 0;
gotedge = 0;
while ((j < W!DTH) && !gotedge)
  {
  i = yminh;
  while ((i <= ymaxh) && !gotedge)
    {
    if (*(image+i*WIDTH+j) == MAXLEV) gotedge = checkedge();
    i++;
    }
  j++;
  }
if (!gotedge)
  { printm("Could not find left edge of screen"); yminh = yminp; }
else xminh = j;
if (xminh < MINLEFT) xminh = MINLEFT;
j = WIDTH - 1;
gotedge = 0;
while ((j >= xminh) && !gotedge)
  {
  i = yminh;
  while ((i <= ymaxh) && !gotedge)
    {
    if (*(image+i*WIDTH+j) == MAXLEV) gotedge = checkedge();
    i++;
    }
  j–;
  }
if (!gotedge)
  { printm("Could not find right edge of screen"); xmaxh = xmaxp; }
else xmaxh = j;
if (xmaxh > MAXRGHT) xmaxh = MAXRGHT;


gotedge = 0;
m = 0;      /* temporary variable to count edge misses */
while (!gotedge)
  {
  gotedge = 1;
  i = yminh;
  for (j=xminh; j<=xmaxh; j++)
    if (*(image+i*WIDTH+j) != MAXLEV)
```

```
      {
        xminh++;
        xmaxh–;
        printf("i=%d j=%d I(i,j)=%d\n",i,j,*(image+i*WIDTH+j));
        gotedge = 0;
        m++;
        break;
      }
    if (!gotedge) yminh++;
    if (m > EDGEMIS)
      { printm("Beam image touching top edge of screen"); return(0); }
    }
  gotedge = 0;
  m = 0;      /* temporary variable to count edge misses */
  while (!gotedge)
    {
    gotedge = 1;
    i = ymaxh;
    for (j=xminh; j<=xmaxh; j++)
      if (*(image+i*WIDTH+j) != MAXLEV)
      {
        xminh++;
        xmaxh–;
        printf("i=%d j=%d I(i,j)=%d\n",i,j,*(image+i*WIDTH+j));
        gotedge = 0;
        m++;
        break;
      }
    if (!gotedge) ymaxh–;
    if (m > EDGEMIS)
      { printm("Beam image touching bottom edge of screen"); return(0); }
    }
  gotedge = 0;
  m = 0;      /* temporary variable to count edge misses */
  while (!gotedge)
    {
    gotedge = 1;
    j = xminh;
    for (i=yminh; i<=ymaxh; i++)
      if (*(image+i*WIDTH+j) != MAXLEV)
      {
        printf("i=%d j=%d I(i,j)=%d\n",i,j,*(image+i*WIDTH+j));
        gotedge = 0;
        m++;
```

```
          break;
        }
      if (!gotedge) xminh++;
      if (m > EDGEMIS)
        { printm("Beam image touching left edge of screen"); return(0); }
      }
    gotedge = 0;
    m = 0;      /* temporary variable to count edge misses */
    while (!gotedge)
      {
      gotedge = 1;
      j = xmaxh;
      for (i=yminh; i<=ymaxh; i++)
        if (*(image+i*WIDTH+j) != MAXLEV)
          {
          printf("i=%d j=%d I(i,j)=%d\n",i,j,*(image+i*WIDTH+j));
          gotedge = 0;
          m++;
          break;
          }
      if (!gotedge) xmaxh--;
      if (m > EDGEMIS)
        { printm("Beam image touching right edge of screen"); return(0); }
      }
    printf("xmin = %d xmax = %d ymin = %d ymax = %d\n",xminh,xmaxh,yminh,ymaxh);
    if ((xmaxh-xminh) <= 0 || (ymaxh-yminh) <= 0)
      { printm("Could not find screen edges"); return(0); }
    return(0);
    }

short checkedge(void)
  {
  merit = 0;
  for (m=-2; m<=2; m++)
    for (n=-2; n<=2; n++)
      if (*(image+(i+m)*WIDTH+j+n) == MAXLEV) merit++;
  if (merit >= STHRESH) return(1);
  else return(0);
  }

void startmicrun(void)
  {
  loadcounter((short)0);
  cwlatch &= CLRMIC & CLRMIR;
  cwlatch |= (CNTNORM | micdir | (micr & CONCAVE) | DPRESET);
  *p40_doutput = cwlatch;
  cwlatch |= ENABLT; *p40_doutput = cwlatch;
  cwlatch |= DCLOCKH; *p40_doutput = cwlatch;
  cwlatch &= DCLOCKL; *p40_doutput = cwlatch;
  counterp = 0;
  countvtxh = countvtx; counthzxh = counthzx;
  countvtvh = countvtv; counthzvh = counthzv;
  if (micr&CONCAVE) { xposdlast = yposdlast = INVALID; }
  else { xalgdlast = yalgdlast = INVALID; }
  }

void stopmicrun(void)
  {
  cwlatch &= DCLEAR; *p40_doutput = cwlatch;
  cwlatch |= DCLOCKH; *p40_doutput = cwlatch;
  cwlatch &= DCLOCKL; *p40_doutput = cwlatch;
  cwlatch &= DISBLT; *p40_doutput = cwlatch;
  counterwait();
  countvtx = countvtxh; counthzx = counthzxh;
  countvtv = countvtvh; counthzv = counthzvh;
  counter = *p40_dinput;
  addcounter();
  dispcount();
  if (gotlive) dispimage();
  }

void pauserun(void)
  {
  cwlatch &= DCLEAR; *p40_doutput = cwlatch;
  cwlatch |= ENABLT; *p40_doutput = cwlatch;
  cwlatch |= DCLOCKH; *p40_doutput = cwlatch;
  cwlatch &= DCLOCKL; *p40_doutput = cwlatch;
  cwlatch &= DISBLT; *p40_doutput = cwlatch;
  counterwait();
  counter = *p40_dinput;
  counter = ((micmove+spoff[micr>>1]) & 0xfffc) - counter;
  if (micdir & RUNNEG) counter = -counter;
  dirhighlight();
  addcounter();
  dispcount();
  runflg = 0;
  }
```

```
void errormove(void)
  {
  countvtxh = countvtx; counthzxh = counthzx;
  countvtvh = countvtv; counthzvh = counthzv;
  runflg = 1;
  movemic();
  aligncnt = 0;
  }

void updatecount(void)
  {
  countvtx = countvtxh; counthzx = counthzxh;
  countvtv = countvtvh; counthzv = counthzvh;
  counter = *p40_dinput;
  counter = ((micmove+spoff[micr>>1]) & 0xfffc) - counter;
  spoff[micr>>1] += (micmove & 0xfffc) - counter;
  if (micdir & RUNNEG) counter = -counter;
  dirhighlight();
  addcounter();
  dispcount();
  runflg = 0;
  /**********************
  cwlatch |= ENABLT;
  *p40_doutput = cwlatch;
  cwlatch |= DCLOCKH;
  *p40_doutput = cwlatch;
  cwlatch &= DCLOCKL;
  *p40_doutput = cwlatch;
  cwlatch &= DISBLT;
  *p40_doutput = cwlatch;
  **********************/
  }

void checksteer(void)
  {
  micmove = 0;      /* do not move mics */
  if (!pwerr) {
    if (!sensflg && contflg && steerflg) controlsteer();
    if (sensflg == 5) { smodflg++; dataflg = 1; setoptions(); }
    if (sensflg == 1 || sensflg == 2)
      { smodflg++; dataflg = 1; setoptions(); } }
  if (sensflg == 3) pwsenscontrol();
  }
```

```
void mirsenscontrol(void)
  {
  if (smodflg == 0 || smodflg == 5)
    {
    if (runflg) pauserun();
    if (!filtflg) { calcfilters(); filtflg = 1; setoptions(); }
    controlalign();
    controlpower();
    if (smodflg == 0 || smodflg == 5) return;
    if (pwerr > ERRPOWER) { smodflg--; return; }
    tautime = runtime - TAUSCRN;
    }
  if (smodflg == 1)
    {
    fprintf(file2,"Commencing mirror %1d sensitivity measurments\r\n",sensflg);
    sprintf(tempbuf,"Commencing mirror %1d sensitivity measurments",sensflg);
    printm(tempbuf);
    if (centflg)
      {
      centflg = 0; setoptions();
      xmin = xminh; xmax = xmaxh;
      ymin = yminh; ymax = ymaxh;
      }
    smodflg++;
    }
  if (runflg)
    if (counter != *p40_dinput) { readcounter(); return; }
    else updatecount();
  if ((runtime - tautime) < TAUSCRN) return;
  else tautime = runtime;
  storedata();
  if (smodflg == 2)
    if (yalgerr < MAXYERR)
    {
      micmove = (short)(ALGSTEP*10);
      micr &= CLRMIC;
      if (micr&CONCAVE) { yalgdir = UP; micdir = yalgdir; }
      else { yposdir = UP; micdir = yposdir; }
      micr |= VTMIC;
      errormove();
    }
    else { smodflg++; }
  if (smodflg == 3)
    if (yalgerr > -MAXYERR)
  {
    micmove = (short)(ALGSTEP*10);
    micr &= CLRMIC;
    if (micr&CONCAVE) { yalgdir = DOWN; micdir = yalgdir; }
    else { yposdir = DOWN; micdir = yposdir; }
    micr |= VTMIC;
    errormove();
  }
  else { smodflg++; }
if (smodflg == 4)
  if (yalgerr < 0)
  {
    micmove = (short)(ALGSTEP*10);
    micr &= CLRMIC;
    if (micr&CONCAVE) { yalgdir = UP; micdir = yalgdir; }
    else { yposdir = UP; micdir = yposdir; }
    micr |= VTMIC;
    errormove();
  }
  else { smodflg++; }
if (smodflg == 6)
  if (xalgerr < MAXXERR)
  {
    micmove = (short)(ALGSTEP*10);
    micr &= CLRMIC;
    if (micr&CONCAVE) { xalgdir = RIGHT; micdir = xalgdir; }
    else { xposdir = RIGHT; micdir = xposdir; }
    micr |= HZMIC;
    errormove();
  }
  else { smodflg++; }
if (smodflg == 7)
  if (xalgerr > -MAXXERR)
  {
    micmove = (short)(ALGSTEP*10);
    micr &= CLRMIC;
    if (micr&CONCAVE) { xalgdir = LEFT; micdir = xalgdir; }
    else { xposdir = LEFT; micdir = xposdir; }
    micr |= HZMIC;
    errormove();
  }
  else { smodflg++; }
if (smodflg == 8)
  if (xalgerr < 0)
```

```
    {
      micmove = (short)(ALGSTEP*10);
      micr &= CLRMIC;
      if (micr&CONCAVE) { xalgdir = RIGHT; micdir = xalgdir; }
      else { xposdir = RIGHT; micdir = xposdir; }
      micr |= HZMIC;
      errormove();
    }
    else { smodflg++; }
  if (smodflg == 9)          /* all directions completed */
    {
    fprintf(file2,"End of mirror %1d sensitivity measurments\r\n",sensflg);
    sprintf(tempbuf,"End of mirror %1d sensitivity measurments",sensflg);
    printm(tempbuf);
    smodflg = 0; sensflg = 0; dataflg = 1;
    contflg = 1; powflg = 0; wandflg = 0; centflg = 1;
    setoptions();
    }
  }

void pwsenscontrol(void)
  {
  if (smodflg == 0)
    {
    if (!filtflg) { calcfilters(); filtflg = 1; setoptions(); }
    upwr = PWRSETD;
    if (upwr < 0) upwr = 0;
    if (upwr > UPWRMAX) upwr = UPWRMAX;
    setDAchan((short)POWCHAN,upwr);
    intenslast = intensity;
    smodflg++;
    tautime = runtime;
    }
  if (smodflg == 1 || smodflg == 2)
    {
    if ((intenslast > intensity) || ((runtime - tautime) < TAUSCRN))
      {
      intenslast = intensity;
      smodflg = 1;
      controlalign();
      return;
      }
    fprintf(file2,"Commencing power vs. intensity measurments\r\n");
    printm("Commencing power vs. intensity measurments");
    tautime = runtime - TAUSCRN;
    smodflg = 3;
    }
  if ((abs(intenslast-intensity) > ERRINTN) || ((runtime - tautime) < TAUSCRN))
    {
    intenslast = intensity;
    i = smodflg;
    controlalign();
    smodflg = i;
    return;
    }
  else
    {
    i = smodflg;
    controlalign();
    if (smodflg == i) return;
    else { smodflg = i; tautime = runtime; }
    }
  /* step to minimum intensity value twice as fast */
  if (smodflg == 3)
    if (intensity < MININTN || upwr < 0.0) smodflg++;
    else upwr -= (2.0*PWRSTEP);
  if (smodflg > 3) storedata();
  if (smodflg == 4) {
    upwr += PWRSTEP;
    if (intensity > MAXINTN || upwr > UPWRMAX) smodflg++; }
  if (smodflg == 5) {
    upwr -= PWRSTEP;
    if (intensity < MININTN || upwr < 0.0) smodflg++; }
  if (smodflg >= 6)
    {
    fprintf(file2,"End of power vs. intensity measurments\r\n");
    printm("End of power vs. intensity measurments");
    smodflg = 0; sensflg = 0; dataflg = 1;
    contflg = 1; powflg = 0; wandflg = 0;
    setoptions();
    upwr = PWRSETD;
    }
  if (upwr < 0) upwr = 0;
  if (upwr > UPWRMAX) upwr = UPWRMAX;
  setDAchan((short)POWCHAN,upwr);
  }

void wandsenscontrol(void)
```

```
  {
  measwandspeed();
  if (smodflg == 0)
    {
    if (!filtflg) { calcfilters(); filtflg = 1; setoptions(); }
    uwand = WNDSETD;
    if (uwand < 0) uwand = 0;
    if (uwand > UWNDMAX) uwand = UWNDMAX;
    setDAchan((short)WNDCHAN,uwand);
    intenslast = intensity;
    smodflg++;
    tautime = runtime;
    }
  if (smodflg == 1 || smodflg == 2)
    {
    if ((intenslast > intensity) || ((runtime - tautime) < TAUSCRN))
      {
      intenslast = intensity;
      smodflg = 1;
      controlalign();
      return;
      }
    fprintf(file2,"Commencing wand sensitivity measurments\n");
    printm("Commencing wand sensitivity measurments");
    tautime = runtime - TAUSCRN;
    smodflg = 3;
    }
  if ((abs(intenslast-intensity) > ERRINTN) || ((runtime - tautime) < TAUSCRN))
    {
    intenslast = intensity;
    i = smodflg;
    controlalign();
    smodflg = i;
    return;
    }
  else
    {
    i = smodflg;
    controlalign();
    if (smodflg == i) return;
    else { smodflg = i; tautime = runtime; }
    }
  /* step to minimum intensity value twice as fast */
  if (smodflg == 3)
```

```
    if (intensity < MININTN || uwand < 0.0) smodflg++;
    else uwand -= (2.0*WNDSTEP);
  if (smodflg > 3) storedata();
  if (smodflg == 4) {
    uwand += WNDSTEP;
    if (intensity > MAXINTN || uwand > UWNDMAX) smodflg++; }
  if (smodflg == 5) {
    uwand -= WNDSTEP;
    if (intensity < MININTN || uwand < 0.0) smodflg++; }
  if (smodflg >= 6)
    {
    fprintf(file2,"End of wand sensitivity measurments\n");
    printm("End of wand sensitivity measurments");
    smodflg = 0; sensflg = 0; dataflg = 1;
    contflg = 1; powflg = 0; wandflg = 0;
    setoptions();
    uwand = WNDSETD;
    }
  if (uwand < 0) uwand = 0;
  if (uwand > UWNDMAX) uwand = UWNDMAX;
  setDAchan((short)WNDCHAN,uwand);
  }

void stepresponse(void)
  {
  if (abs((int)xmoveset) >= MINCNT)
    {
    fprintf(file2,"Commencing X-direction step response\r\n");
    printm("Commencing X-direction step response");
    if (centflg)
      {
      centflg = 0; setoptions();
      xmin = xminh; xmax = xmaxh;
      ymin = yminh; ymax = ymaxh;
      }
    if (xmoveset < 0) {
      if (micr&CONCAVE) { xalgdir = LEFT; micdir = xalgdir; }
      else { xposdir = LEFT; micdir = xposdir; }
      if (xmoveset > MAXMOVEX) xmoveset = MAXMOVEX; }
    else {
      if (micr&CONCAVE) { xalgdir = RIGHT; micdir = xalgdir; }
      else { xposdir = RIGHT; micdir = xposdir; }
      if (abs((int)xmoveset) > MAXMOVEX) xmoveset = -MAXMOVEX; }
    micmove = abs((int)xmoveset);
    micr &= CLRMIC; micr |= HZMIC;
    initstep = xalgerr;
    sprintf(tempbuf,"Initial X uniformity value = %6d gv",initstep);
    printm(tempbuf);
    errormove();
    tunetime = gettime();
    }
  else if (abs((int)ymoveset) >= MINCNT)
    {
    fprintf(file2,"Commencing Y-direction step response\r\n");
    printm("Commencing Y-direction step response");
    if (centflg)
      {
      centflg = 0; setoptions();
      xmin = xminh; xmax = xmaxh;
      ymin = yminh; ymax = ymaxh;
      }
    if (ymoveset < 0) {
      if (micr&CONCAVE) { yalgdir = DOWN; micdir = yalgdir; }
      else { yposdir = DOWN; micdir = yposdir; }
      if (ymoveset > MAXMOVEY) ymoveset = MAXMOVEY; }
    else {
      if (micr&CONCAVE) { yalgdir = UP; micdir = yalgdir; }
      else { yposdir = UP; micdir = yposdir; }
      if (abs((int)ymoveset) > MAXMOVEY) ymoveset = -MAXMOVEY; }
    micmove = abs((int)ymoveset);
    micr &= CLRMIC; micr |= VTMIC;
    initstep = yalgerr;
    sprintf(tempbuf,"Initial Y uniformity value = %6d gv",initstep);
    printm(tempbuf);
    errormove();
    tunetime = gettime();
    }
  else if (pwstepset > ERRFLOAT)
    {
    fprintf(file2,"Commencing power step response\r\n");
    printm("Commencing power step response");
    if (upwr < 0) upwr = 0;
    if (upwr > UPWRMAX) upwr = UPWRMAX;
    setDAchan((short)POWCHAN,upwr);
    initstep = intensity;
    initpwstep = powmeter;
    sprintf(tempbuf,"Initial intensity value = %6d gv",initstep);
    printm(tempbuf);
    tunetime = gettime();
    }
  else if (wnstepset > ERRFLOAT)
    {
    fprintf(file2,"Commencing wand step response\r\n");
    printm("Commencing wand step response");
    if (uwand < 0) uwand = 0;
    if (uwand > UWNDMAX) uwand = UWNDMAX;
    setDAchan((short)WNDCHAN,uwand);
    initstep = intensity;
    sprintf(tempbuf,"Initial intensity value = %6d gv",initstep);
    printm(tempbuf);
    tunetime = gettime();
    }
  sensflg = 7;
  }

void tunecontrol(void)
  {
  controlalign(); controlwand(); controlpower();
  measwandspeed();
  storedata();
  if ((gettime() - tunetime) > TAUSCRN) {
    if (abs((int)xmoveset) >= MINCNT) {
      finalstep = xalgerr;
      sprintf(tempbuf,"Final X uniformity value = %6d gv",finalstep);
      printm(tempbuf);
      sprintf(tempbuf,"Computed X uniformity gain value = %5.2f",
                fabs((1.0*finalstep-initstep)*XGAINMOD/xmoveset));
      printm(tempbuf);
      xmoveset = 0; }
    else if (abs((int)ymoveset) >= MINCNT) {
      finalstep = yalgerr;
      sprintf(tempbuf,"Final Y uniformity value = %6d gv",finalstep);
      printm(tempbuf);
      sprintf(tempbuf,"Computed Y uniformity gain value = %5.2f",
                fabs((1.0*finalstep-initstep)*YGAINMOD/ymoveset));
      printm(tempbuf);
      ymoveset = 0; }
    else if (pwstepset > ERRFLOAT) {
      finalstep = intensity;
      finpwstep = powmeter;
      sprintf(tempbuf,"Final intensity value = %6d gv",finalstep);
      printm(tempbuf);
      sprintf(tempbuf,"Computed power controller gain value = %5.2f",
                (finpwstep-initpwstep)*PGAINMOD/pwstepset);
      printm(tempbuf);
      sprintf(tempbuf,"Computed power vs. intensity gain value = %5.2f",
                (finpwstep-initpwstep)/(finalstep-initstep));
      printm(tempbuf);
      pwstepset = 0.0; }
    else if (wnstepset > ERRFLOAT) {
      finalstep = intensity;
      finpwstep = powmeter;
      sprintf(tempbuf,"Final intensity value = %6d gv",finalstep);
      printm(tempbuf);
      sprintf(tempbuf,"Computed intensity vs. Uwand gain value = %5.2f",
                (finalstep-initstep)/wnstepset);
      printm(tempbuf);
      wnstepset = 0.0; }
    sensflg = 0; }
  }

void possenscontrol(void)
  {
  if (smodflg == 0 || (smodflg%2) == 1)
    {
    if (dataflg) { dataflg = 0; setoptions(); }
    if (!filtflg) { calcfilters(); filtflg = 1; setoptions(); }
    controlalign();
    controlpower();
    if ((smodflg == 0 || (smodflg%2) == 1) && smodflg != 1) return;
    if (pwerr > ERRPOWER) { smodflg--; return; }
    if (aligncnt < 4) { smodflg--; return; }
    if (!dataflg) { dataflg = 1; setoptions(); }
    }
  if (smodflg == 1)
    {
    fprintf(file2,"Commencing position sensitivity measurments\r\n");
    printm("Commencing position sensitivity measurments");
    smodflg++;
    }
  storedata();
  if ((smodflg/2) <= POSGRID)
    {
    setmirror((short)(resmirror^0xffff));
    micmove = (short)(POSSTEP*10);
    micr &= CLRMIC;
    micdir = dirgrid[(smodflg/2-1)];
    if (micdir == DOWN || micdir == UP)
      { micr |= VTMIC; yposdir = micdir; }
    else { micr |= HZMIC; xposdir = micdir; }
    errormove();
    runflg = 2;
    }
  else
    {
    sensflg = 0; dataflg = 1;
    contflg = 1; powflg = 0; wandflg = 0;
    setoptions();
    fprintf(file2,"End of position sensitivity measurments\n");
    printm("End of position sensitivity measurments");
    }
  smodflg++;
  }

void readcounter(void)
  {
  if (runflg)
    {
    tempcnt = counter;
    counter = counterp - tempcnt;
    counterp = tempcnt;
    if (micdir & RUNNEG) counter = -counter;
    addcounter();
    dispcount();
    }
  else
    {
    tempcnt = *p40_dinput;
    counter = tempcnt - counterp;
    addcounter();
    dispcount();
    counterp = tempcnt;
    }
  }

/* This function implements a standard discrete proportional controller to
   adjust the output power level of the laser. The controller uses the
   intensity data from the imaging screen as feedback and adjusts the
   sustainer current via the Proto-40K D/A interface. */
void controlpower (void)
```

```
  {
  epoft = (float)pwerr/1000.0;    /* convert power error to kW */
  if (powflg && contflg && (micmove < MAXMOVE) && (abs(pwerr) > pmin))
    {
    upwr += pwrgain * (epoft - epoftm1 + tsamp/tipwr*epoft);
    if (upwr < 0) upwr = 0;
    if (upwr > UPWRMAX) upwr = UPWRMAX;
    setDAchan((short)POWCHAN,upwr);
    }
  epoftm1 = epoft;
  }

/* This function implements a standard discrete proportional controller to
   adjust the alignment of the laser beam in the X and Y directions. The
   controller only adjusts the micrometer which controls the larger of the
   X and Y errors. */
void controlalign(void)
  {
  if (runflg == 2)
    {
    if (counter != *p40_dinput) { readcounter(); return; }
    else
      {
      updatecount();
      if (strmvflg) setmirror((short)(resmirror^0xffff));
      strmvflg = 0;
      }
    }
  eaxoft = (float)(xalgerr + xalgoff)/KILO; /* convert X-alg error kInt */
  ualgx = (short)(xalggain * eaxoft + 0.5);
  if (abs((int)ualgx) < MINCNT) ualgx = 0;
  if (ualgx > 0) {
    if (micr&CONCAVE) xalgdir = LEFT;
    else xposdir = LEFT;
    if (ualgx > MAXMOVEX) ualgx = MAXMOVEX; }
  if (ualgx < 0) {
    if (micr&CONCAVE) xalgdir = RIGHT;
    else xposdir = RIGHT;
    if (abs((int)ualgx) > MAXMOVEX) ualgx = -MAXMOVEX; }

  eayoft = (float)(yalgerr + yalgoff)/KILO; /* convert Y-alg error to kInt */
  ualgy = (short)(yalggain * eayoft + 0.5);
  if (abs((int)ualgy) < MINCNT) ualgy = 0;
  if (ualgy > 0) {
    if (micr&CONCAVE) yalgdir = DOWN;
    else yposdir = DOWN;
    if (ualgy > MAXMOVEY) ualgy = MAXMOVEY; }
  if (ualgy < 0) {
    if (micr&CONCAVE) yalgdir = UP;
    else yposdir = UP;
    if (abs((int)ualgy) > MAXMOVEY) ualgy = -MAXMOVEY; }

  if (runflg)
    {
    if (counter != *p40_dinput) { readcounter(); return; }
    else updatecount();
    }
  if (contflg) {
    if ((abs((int)ualgx) >= xalgmin) || (abs((int)ualgy) >= yalgmin))
    if (abs((int)ualgy) > abs((int)ualgx)) {
      micmove = abs((int)ualgy);
      if (micr&CONCAVE) micdir = yalgdir;
      else micdir = yposdir;
      micr &= CLRMIC; micr |= VTMIC;
      errormove(); }
    else {
      micmove = abs((int)ualgx);
      if (micr&CONCAVE) micdir = xalgdir;
      else micdir = xposdir;
      micr &= CLRMIC; micr |= HZMIC;
      errormove(); }
    else {
    micmove = 0;
    aligncnt++;
    if (sensflg) smodflg++;
    if (steerflg) controlsteer(); } }
  }

/* This function implements a standard discrete proportional controller to
  adjust the position of the laser beam in the X and Y directions. The
  controller only adjusts the micrometer which controls the larger of the
  X and Y errors. */
void controlsteer(void)
  {
  if ((abs(xcentd-xcenth) <= XCNOISE) && (abs(ycentd-ycenth) <= YCNOISE))
    return;

  setmirror((short)(resmirror^0xffff));
```

```
  strmvflg = 1;
  micr &= CLRMIC;
  if (abs(xcentd-xcenth) > abs(ycentd-ycenth))
    {
    if (xcentd < xcenth) micdir = LEFT;
    else micdir = RIGHT;
    micmove = XCMOVE;
    micr |= HZMIC; xposdir = micdir;
    }
  else
    {
    if (ycentd < ycenth) micdir = UP;
    else micdir = DOWN;
    micmove = YCMOVE;
    micr |= VTMIC; yposdir = micdir;
    }
  errormove();
  runflg = 2;
  }

/* This function controls the speed of the variable coupling wand. */
void controlwand(void)
  {
  if (wandflg)
    {
    if (intensity < MININTN && wandcnt == 0)
      { uwand += WNDSTEP; wandcnt = WNDELAY; }
    if (intensity > MAXINTN && wandcnt == 0)
      { uwand -= WNDSTEP; wandcnt = WNDELAY; }
    if (uwand < 0) uwand = 0;
    if (uwand > UWNDMAX) uwand = UWNDMAX;
    setDAchan((short)WNDCHAN,uwand);
    if (wandcnt > 0) wandcnt--;
    }
  }

/* This function measures the speed of the variable coupling wand. */
void measwandspeed(void)
  {
  short numavg;
  long cnt,sumcntper,sumcntopen;

  p40latch = PLATCH;
  *p40_doutput = p40latch;
```

```
p40latch |= ENABLT;
*p40_doutput = p40latch;
wandper = gettime();
while (((*p40_dinput & WNDINPUT) == WNDINPUT) &&
    ((gettime() - wandper) < MINWPER));
if ((gettime() - wandper) >= MINWPER) {
  printm("Wand not operating"); return; }
wandper = gettime();
while (((*p40_dinput & WNDINPUT) == 0) &&
    ((gettime() - wandper) < MINWPER));
if ((gettime() - wandper) >= MINWPER) {
  printm("Wand not operating"); return; }
sumcntopen = 0;
sumcntper = 0;
if (sensflg == 4) numavg = NUMSPAVG;
else numavg = 1;
for (i=0; i<numavg; i++)
  {
  cnt=0;
  while ((*p40_dinput & WNDINPUT) == WNDINPUT) cnt++;
  sumcntopen += cnt;
  while ((*p40_dinput & WNDINPUT) == 0) cnt++;
  sumcntper += cnt;
  }
wandper = ((float)sumcntper/(float)numavg) * WPERFACT + WPEROFF;
if (wandper < 0.05) wandper = 0.05;
wandopen = (float)sumcntopen/(float)sumcntper * 200.0 * WNDSCALE; /* in % */
if (wandopen > 2.0) wandopen = 2.0;
p40latch &= DISBLT;
*p40_doutput = p40latch;
}
```

D.2 CONTGADG.C

```
/************************************************************************
This source file contains the variables and functions required to open all
windows and interface with the intuition library. It processes all messages
from the menus and gadgets.

Filename: "CONTGADG.C", Version: 2.1, Date: May 5, 1992

***************************** ********************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <exec/exec.h>
#include <intuition/intuition.h>
#include <intuition/intuitionbase.h>
#include <graphics/gfxbase.h>
#include <workbench/workbench.h>
#include <exec/interrupts.h>
#include <hardware/intbits.h>
#include <functions.h>
#include "control.h"
#include "display.h"
#include "contgadg.h"

/* Variables and functions defined in "control.c" */
extern short DOWN, UP, RIGHT, LEFT;
extern double runtime;
extern float powmeter,current,voltage,uwand,upwr,tsamp,pwrgain,pconvgain,
      xalggain,yalggain,pwintgain,tipwr,pwstepset,wnstepset;
extern long  xalgoff,yalgoff,pconvoff,countvtx,counthzx,countvtv,counthzv,
      file1,intensity;
extern short powerset,xcenth,ycenth,xminh,xmaxh,yminh,ymaxh,
      xalgmin,yalgmin,pmin,runflg,centflg,imageflg,contflg,steerflg,wandflg,
      powflg,WB,dataflg,colorflg,micr,sensflg,senscnt,xmoveset,ymoveset,
      micdir,pwmetflg,curflg,smodflg,voltflg,spoff[],filtflg,xmin,xmax,
      ymin,ymax,wandcnt,filenum,resmirror;
extern char imagefile[],datafile[],parfile[],dirarray[];
extern unsigned char *image,maxlev,minlev;
extern FILE *file2;
extern void startmicrun(),stopmicrun(),adjustpower(),pauserun(),
      printstart(),readcounter(),loadparameters(),saveparameters(),
      resetparameters(),convertADchan();

/* Variables and functions defined in other source files */
extern void audiocleanup();           /* in "talk.c", for cleanup() */

extern short dispimage(),*gotlive;           /* in "grabfram.c" */
extern long camera;
extern void livecleanup(),dispcoords(),setcolormap();

extern void dispwtext(),dispdata(),printm(char *str); /* in "display.c" */
extern void dispcount(),displot(),plotlegend();
extern short plotflg[];

extern void setcam(long camera);           /* in "setcam.asm" */

extern void setDAchan(short chan, float volts),     /* in "p40.c" */
     closesystimer(),settime();
extern double gettime();

extern float fcutoff;                     /* in "filter.c" */
extern void calcfilters();
extern short filtype;

/* Variables and functions defined in this source file */
/* Structures to recover messages and open window */
struct IntuitionBase *IntuitionBase = NULL;
struct GfxBase *GfxBase = NULL;
struct Window *ControlWindow = NULL; /* pointer to main control window */
struct Window *PlotWindow = NULL;   /* pointer to plo t window */
struct IntuiMessage *controlmsg;   /* pointer to intuition message */
struct MenuItem *msgitem;          /* pointer to selected menu item */
ULONG msgclass;                   /* structure to hold message */
USHORT msgcode;              /* variable to hold needed code information */
APTR msgaddress;             /* address of selected gadget */

long op;       /* temporary variable used when calling RemoveGadget() */
static short i;  /* temporary variable used by routines */
short done,      /* set to 1 to stop control, 2 to exit program */
     pauseflg,  /* set to indicate control paused temporarily */
     menuflg;  /* set when menu has been installed (for cleanup() */
short dispargs(),checkargs(),checkmessage(),chkplotflg(),
     initimages(),validinputs();
void ignoremsg(),menumessage(USHORT code),gdownmessage(APTR address),
```

```
     gadgetmessage(APTR address),freeimages(),cleanup(short wait),
     errcleanup(),EnableRunGadg(),DisableRunGadg(),
     dirhighlight(),powhighlight(short pwdir),setmirror(short mirror),setoptions(),
     openplotwind(),closeplotwind(),initgadgbufs(),updategadgbufs();
/* This function opens the necessary libraries and windows, and initializes
   the window text and images. */
short dispargs(void)
  {
  pauseflg = 1;          /* set to enable manual control */
  menuflg = 0;             /* menu not yet installed */
  if (GfxBase == NULL)
    GfxBase = OpenLibrary("graphics.library",LIBRARY_MINIMUM);
  if (GfxBase == NULL)
    { printm("Cannot open graphics library"); return(1); }
  if (IntuitionBase == NULL)
    IntuitionBase = OpenLibrary("intuition.library",LIBRARY_MINIMUM);
  if (IntuitionBase == NULL)
    { printm("Cannot open intuition library"); return(1); }
  if (ControlWindow == NULL)
    if ((ControlWindow = OpenWindow(&NewControlWindow)) == NULL)
    { printm("Cannot open window"); return(1); }

  /* allocate memory accessable by the chips for images */
  if (initimages() != TRUE)
    { printm("Not enough memory for window"); return(1); };

  SetAPen(ControlWindow->RPort,BLACK);
  RectFill(ControlWindow->RPort,0,0,(long)ControlWindow->GZZWidth,
              (long)ControlWindow->GZZHeight);
  SetMenuStrip(ControlWindow,&Project);
  menuflg = 1;           /* indicates menu is installed */
  dispwtext();              /* display window text */
  if (plotflg[NUMSIGS]) openplotwind();     /* open plot window */
  setmirror((short)(micr&CONCAVE));
  return(0);
  }

/* This function checks for waits for messages from the Control window and
   calls the appropriate functions to process them. */
short checkargs(void)
  {
  sprintf(ceilbuf,"%d",maxlev);
  sprintf(floorbuf,"%d",minlev);
  RefreshGadgets(&powerGadget,ControlWindow,NULL);
```

```
dispcount();            /* display micrometer count */
dispdata();          /* display computed and other values */
DisableRunGadg();          /* disable invalid menus and gadgets */
done = 0;
while (!done)
  if (WaitPort(ControlWindow->UserPort)) checkmessage();
if (done > 1) return(1);
else { EnableRunGadg(); return(0); } /* enable valid gadgets and menus */
}

/* This function processes messages from the Control window. */
short checkmessage(void)
 {
 done = 0;
 while (controlmsg = (struct IntuiMessage *)
        GetMsg(ControlWindow->UserPort))
 {
   msgclass = controlmsg->Class;
   msgcode = controlmsg->Code;
   msgaddress = controlmsg->IAddress;
   ReplyMsg(&controlmsg->ExecMessage);
   switch (msgclass)
   {
     case GADGETUP    : gadgetmessage(msgaddress);
            break;
     case GADGETDOWN  : gdownmessage(msgaddress);
            break;
     case MENUPICK    : menumessage(msgcode);
            break;
     case CLOSEWINDOW : ignoremsg();
            if (runflg) pauserun();
            done = 2;
            break;
     default       : break;
     }
 }
 return(done);
 }

/* This function replies any unprocessed messages (i.e. ignores them) */
void ignoremsg(void)
 {
 if (ControlWindow != NULL)
   while (controlmsg = (struct IntuiMessage *)
        GetMsg(ControlWindow->UserPort))
   ReplyMsg(&controlmsg->ExecMessage);
 }

/* This function process all messages from the menus. */
void menumessage(USHORT code)
 {
 while (code != MENUNULL)
   {
   msgitem = ItemAddress(&Project,(long)code);
   if (msgitem == &StartItem)
     { if (validinputs()) { ignoremsg(); done = 1; } }
   else if (msgitem == &PauseItem)
     {
     if (runflg) pauserun();
     pauseflg = 1;
     if (sensflg) { sensflg = 0; smodflg = 0; setoptions(); }
     OffMenu(ControlWindow,0x0020);     /* pause item off */
     OnMenu(ControlWindow,0x0040);      /* resume item on*/
     OnGadget(&convexGadget,ControlWindow,NULL);
     OnGadget(&concavGadget,ControlWindow,NULL);
     while (pauseflg && !done)
       if (WaitPort(ControlWindow->UserPort)) checkmessage();
     }
   else if (msgitem == &ResumeItem)
     {
     pauseflg = 0;
     OffMenu(ControlWindow,0x0040);    /* resume item off */
     OnMenu(ControlWindow,0x0020);      /* pause item on */
     OffGadget(&convexGadget,ControlWindow,NULL);
     OffGadget(&concavGadget,ControlWindow,NULL);
     }
   else if (msgitem == &StopItem)
     {
     ignoremsg();
     if (runflg) pauserun();
     done = 1;
     if (sensflg) { sensflg = 0; smodflg = 0; setoptions(); }
     }
   else if (msgitem == &ExitItem)
     {
     while (msgitem->NextSelect != MENUNULL)
       {
       code = msgitem->NextSelect;
```

```
      msgitem = ItemAddress(&Project,(long)code);
      }
    ignoremsg();
    if (runflg) pauserun();
    done = 2;
    }
  else if (msgitem == &contOffSub) contflg = 0;
  else if (msgitem == &contOnSub) contflg = 1;
  else if (msgitem == &steerOffSub) steerflg = 0;
  else if (msgitem == &steerOnSub) steerflg = 1;
  else if (msgitem == &powrOffSub) powflg = 0;
  else if (msgitem == &powrOnSub) powflg = 1;
  else if (msgitem == &wandOffSub) { wandflg = 0; wandcnt = 0; }
  else if (msgitem == &wandOnSub) { wandflg = 1; wandcnt = 0; }
  else if (msgitem == &dataNoSub) dataflg = 0;
  else if (msgitem == &dataYesSub) dataflg = 1;
  else if (msgitem == &imageNoSub) imageflg = 0;
  else if (msgitem == &imageOvSub) imageflg = 1;
  else if (msgitem == &imageSqSub) imageflg = 2;
  else if (msgitem == &centYesSub) centflg = 1;
  else if (msgitem == &centNoSub)
    {
    centflg = 0;
    xmin = xminh; xmax = xmaxh;
    ymin = yminh; ymax = ymaxh;
    }
  else if (msgitem == &colBWSub) { colorflg = 0; setcolormap(); }
  else if (msgitem == &colColSub) { colorflg = 1; setcolormap(); }
  else if (msgitem == &filtYesSub)
    { filtype = MOVEAVG; calcfilters(); filtflg = 1; }
  else if (msgitem == &filtNoSub) filtflg = 0;
  else if (msgitem == &topcamSub)
    { camera = TOPCAM; setcam(camera); Delay(CAMDLAY); }
  else if (msgitem == &botcamSub)
    { camera = BOTCAM; setcam(camera); Delay(CAMDLAY); }
  else if (msgitem == &plotXerrSub)
    if (!plotflg[XALG]) { plotflg[XALG] = 1; openplotwind(); }
    else {
      plotXerrSub.Flags &= (CHECKED ^ 0xffff);
      plotflg[XALG] = 0;
      if (!chkplotflg()) closeplotwind();
      else plotlegend(); }
  else if (msgitem == &plotYerrSub)
    if (!plotflg[YALG]) {plotflg[YALG] = 1; openplotwind(); }
```

```
      else {
        plotYerrSub.Flags &= (CHECKED ^ 0xffff);
        plotflg[YALG] = 0;
        if (!chkplotflg()) closeplotwind();
        else plotlegend(); }
    else if (msgitem == &plotXposSub)
      if (!plotflg[XPOS]) { plotflg[XPOS] = 1; openplotwind(); }
      else {
        plotXposSub.Flags &= (CHECKED ^ 0xffff);
        plotflg[XPOS] = 0;
        if (!chkplotflg()) closeplotwind();
        else plotlegend(); }
    else if (msgitem == &plotYposSub)
      if (!plotflg[YPOS]) {plotflg[YPOS] = 1; openplotwind(); }
        else {
        plotYposSub.Flags &= (CHECKED ^ 0xffff);
        plotflg[YPOS] = 0;
        if (!chkplotflg()) closeplotwind();
        else plotlegend(); }
    else if (msgitem == &plotSpwrSub)
      if (!plotflg[SPWR]) {plotflg[SPWR] = 1; openplotwind(); }
        else {
        plotSpwrSub.Flags &= (CHECKED ^ 0xffff);
        plotflg[SPWR] = 0;
        if (!chkplotflg()) closeplotwind();
        else plotlegend(); }
    else if (msgitem == &plotMpwrSub)
      if (!plotflg[MPWR]) {plotflg[MPWR] = 1; openplotwind(); }
      else {
        plotMpwrSub.Flags &= (CHECKED ^ 0xffff);
        plotflg[MPWR] = 0;
        if (!chkplotflg()) closeplotwind();
        else plotlegend(); }
    else if (msgitem == &plotNoneSub)
      { for (i=0; i<NUMSIGS; i++) plotflg[i] = 0; closeplotwind(); }
    else if (msgitem == &time0Item)
      { settime(); runtime = gettime() + tsamp; }
    else if (msgitem == &resetcntItem)
      {
      countvtx = 0; counthzx = 0; countvtv = 0; counthzv = 0;
      dispcount();
      }
    else if (msgitem == &calibItem)
      {
```

```
if (powmeter > 0.5 && pwmetflg) {
  pconvoff = powmeter*KILO - (intensity*pwintgain) + 0.5;
  sprintf(pconvoffbuf,"%4.1f",(float)pconvoff/KILO); }
 }
else if (msgitem == &resetvaritem)
 {
 spoff[0] = spoff[1] = spoff[2] = spoff[3] = 0;
 filenum = 0;
 }
else if (msgitem == &coordsItem) dispcoords();
else if (msgitem == &loadparSub)
 {
 sprintf(reqTextbuf,"\"%s\"?",parfile);
 if (AutoRequest(ControlWindow,&loadreqText,&YesreqText,
            &NoreqText,0,0,260,70)) loadparameters();
 RefreshGadgets(&powerGadget, ControlWindow, NULL);
 }
else if (msgitem == &resetparSub)
 {
 sprintf(reqTextbuf,"\"pie:controlpar.org\"?");
 if (AutoRequest(ControlWindow,&resetreqText,&YesreqText,
            &NoreqText,0,0,260,70)) resetparameters();
 RefreshGadgets(&powerGadget, ControlWindow, NULL);
 }
else if (msgitem == &modparSub)
 {
 sprintf(uwandbuf,"%5.2f",uwand);
 if (!Request(&ParamRequest,ControlWindow))
   printm("Error opening requester window");
 }
else if (msgitem == &filenameItem)
 {
 if (!Request(&FileRequest,ControlWindow))
   printm("Error opening requester window");
 }
else if (msgitem == &saveparItem)
 {
 sprintf(reqTextbuf,"\"%s\"?",parfile);
 if (AutoRequest(ControlWindow,&savereqText,&YesreqText,
            &NoreqText,0,0,260,70)) saveparameters();
 }
else if (msgitem == &nosenSub)
 { sensflg = 0; smodflg = 0; setoptions(); }
else if (msgitem == &cxsenSub)
 { sensflg = 1; smodflg = 0; dataflg = 1;
  contflg = 1; powflg = 0; wandflg = 0; steerflg = 0;
  setoptions();
  setmirror((short)CONVEX); }
else if (msgitem == &cvsenSub)
 { sensflg = 2; smodflg = 0; dataflg = 1;
  contflg = 1; powflg = 0; wandflg = 0; steerflg = 0;
  setoptions();
  setmirror((short)CONCAVE); }
else if (msgitem == &pwsenSub)
 { sensflg = 3; smodflg = 0; dataflg = 1;
  contflg = 1; powflg = 0; wandflg = 0; steerflg = 0;
  setoptions(); }
else if (msgitem == &wandsenSub)
 { sensflg = 4; smodflg = 0; dataflg = 1;
  contflg = 1; powflg = 0; wandflg = 0; steerflg = 0;
  setoptions(); }
else if (msgitem == &possenSub)
 { sensflg = 5; smodflg = 0; dataflg = 1;
  contflg = 1; powflg = 0; wandflg = 0; steerflg = 0;
  setoptions(); }
else if (msgitem == &steprespItem)
 {
 sprintf(xmovebuf,"%5.1f",xmoveset/10.0);
 sprintf(ymovebuf,"%5.1f",ymoveset/10.0);
 sprintf(upwrbuf,"%5.2f",upwr);
 sprintf(uwandsrbuf,"%5.2f",uwand);
 if (!Request(&StepRespRequest,ControlWindow))
   printm("Error opening requester window");
 }
else if (msgitem == &monNoneSub)
 { pwmetflg = curflg = voltflg = 0;
  powmeter = current = voltage = 0.0; }
else if (msgitem == &pwmetSub)
 if (!pwmetflg) pwmetflg = 1;
 else {
   pwmetSub.Flags &= (CHECKED ^ 0xffff);
   pwmetflg = 0; powmeter = 0.0;
   if (!pwmetflg && !curflg && !voltflg)
     monNoneSub.Flags |= CHECKED; }
else if (msgitem == &crmetSub)
 if (!curflg) curflg = 1;
 else {
   crmetSub.Flags &= (CHECKED ^ 0xffff);
```

```
      curflg = 0; current = 0.0;
      if (!pwmetflg && !curflg && !voltflg)
        monNoneSub.Flags |= CHECKED; }
    else if (msgitem == &voltsSub)
      if (!voltflg) voltflg = 1;
      else {
        voltsSub.Flags &= (CHECKED ^ 0xffff);
        voltflg = 0; voltage = 0.0;
        if (!pwmetflg && !curflg && !voltflg)
          monNoneSub.Flags |= CHECKED; }
    code = msgitem->NextSelect;
    }
  }

/* This function processes messages from the gadgets which are active only
   when the are held down (i.e. micrometer movement gadgets). */
void gdownmessage(APTR address)
  {
  if (!pauseflg) return;
  if (address == (APTR)&upGadget)
    {
    micr &= CLRMIC; micr |= VTMIC; micdir = UP;
    startmicrun();
    while (upGadget.Flags & SELECTED) readcounter();
    stopmicrun();
    }
  else if (address == (APTR)&downGadget)
    {
    micr &= CLRMIC; micr |= VTMIC; micdir = DOWN;
    startmicrun();
    while (downGadget.Flags & SELECTED) readcounter();
    stopmicrun();
    }
  else if (address == (APTR)&leftGadget)
    {
    micr &= CLRMIC; micr |= HZMIC; micdir = LEFT;
    startmicrun();
    while (leftGadget.Flags & SELECTED) readcounter();
    stopmicrun();
    }
  else if (address == (APTR)&rightGadget)
    {
    micr &= CLRMIC; micr |= HZMIC; micdir = RIGHT;
    startmicrun();
    while (rightGadget.Flags & SELECTED) readcounter();
    stopmicrun();
    }
  else if (address == (APTR)&pwupGadget)
    {
    while (pwupGadget.Flags & SELECTED)
      {
      upwr += 1.0 / 255.0;
      if (upwr < 0) upwr = 0;
      if (upwr > UPWRMAX) upwr = UPWRMAX;
      setDAchan((short)POWCHAN,upwr);
      convertADchan();
      dispdata();
      }
    }
  else if (address == (APTR)&pwdownGadget)
    {
    while (pwdownGadget.Flags & SELECTED)
      {
      upwr -= 1.0 / 255.0;
      if (upwr < 0) upwr = 0;
      if (upwr > UPWRMAX) upwr = UPWRMAX;
      setDAchan((short)POWCHAN,upwr);
      convertADchan();
      dispdata();
      }
    }
  }

/* This function processes messages from gadgets which must be released
   before their messages can be processed. */
void gadgetmessage(APTR address)
  {
  if (address == (APTR)&convexGadget)
    {
    if (convexGadget.Flags & SELECTED) {
      micr &= CONVEX;
      DOWN = DOWNCX; UP = UPCX; LEFT = LEFTCX; RIGHT = RIGHTCX; }
    else {
      micr |= CONCAVE;
      DOWN = DOWNCV; UP = UPCV; LEFT = LEFTCV; RIGHT = RIGHTCV; }
    op = RemoveGadget(ControlWindow, &concavGadget);
    concavGadget.Flags ^= SELECTED;
    AddGadget(ControlWindow  &concavGadget, op);
    RefreshGadgets(&concavGadget, ControlWindow, NULL);
    dispcount();
    }
  else if (address == (APTR)&concavGadget)
    {
    if (concavGadget.Flags & SELECTED) {
      micr |= CONCAVE;
      DOWN = DOWNCV; UP = UPCV; LEFT = LEFTCV; RIGHT = RIGHTCV; }
    else {
      micr &= CONVEX;
      DOWN = DOWNCX; UP = UPCX; LEFT = LEFTCX; RIGHT = RIGHTCX; }
    op = RemoveGadget(ControlWindow, &convexGadget);
    convexGadget.Flags ^= SELECTED;
    AddGadget(ControlWindow, &convexGadget, op);
    RefreshGadgets(&concavGadget, ControlWindow, NULL);
    dispcount();
    }
  else if (address == (APTR)&powerGadget)
    {
    if (atoi(powerbuf) < MINPWR || atoi(powerbuf) > MAXPWR)
      {
      printm("Invalid power setting - power setting unchanged");
      sprintf(powerbuf,"%5d",powerset);
      }
    else powerset = atoi(powerbuf);
    }
  else if (address == (APTR)&OKparGadget)
    if (validinputs()) {
    EndRequest(&ParamRequest,ControlWindow);
    if (uwand < 0.0) uwand = 0.0;
    if (uwand > UWNDMAX) uwand = UWNDMAX;
    setDAchan((short)WNDCHAN,uwand);
    printstart(); }
    else printm("All inputs must be valid to continue");
  else if (address == (APTR)&CancelparGadget) initgadgbufs();
  else if (address == (APTR)&OKfnGadget)
    {
    strcpy(imagefile,imgfilebuf);
    strcpy(datafile,datfilebuf);
    strcpy(parfile,parfilebuf);
    }
  else if (address == (APTR)&CancelfnGadget)
    {
    strcpy(imgfilebuf,imagefile);
    strcpy(datfilebuf,datafile);
    strcpy(parfilebuf,parfile);
    }
  else if (address == (APTR)&OKsrGadget)
    {
    sensflg = 6; dataflg = 1;
    xmoveset = (short)(atof(xmovebuf)*10.0);
    ymoveset = (short)(atof(ymovebuf)*10.0);
    if (xmoveset || ymoveset)
      { if (runflg) pauserun(); contflg = 0; }
    else { powflg = 0; wandflg = 0; }
    pwstepset = ((float)atof(upwrbuf)) - upwr;
    upwr = (float)atof(upwrbuf);
    wnstepset = ((float)atof(uwandsrbuf)) - uwand;
    uwand = (float)atof(uwandsrbuf);
    setoptions();
    }
  else if (address == (APTR)&CancelsrGadget)
    {
    sprintf(xmovebuf,"%5.1f",xmoveset/10.0);
    sprintf(ymovebuf,"%5.1f",ymoveset/10.0);
    sprintf(upwrbuf,"%5.2f",upwr);
    sprintf(uwandsrbuf,"%5.2f",uwand);
    sprintf(uwandbuf,"%5.2f",uwand);
    }
  }

/* This function selects the desired mirror to control and updates the
  corresponding gadgets. */
void setmirror(short mirror)
  {
  if (mirror == CONCAVE)
    {
    resmirror = CONCAVE;
    micr |= CONCAVE;
    op = RemoveGadget(ControlWindow, &concavGadget);
    concavGadget.Flags |= SELECTED;
    AddGadget(ControlWindow, &concavGadget, op);
    op = RemoveGadget(ControlWindow, &convexGadget);
    convexGadget.Flags &= (SELECTED ^ 0xffff);
    AddGadget(ControlWindow, &convexGadget, op);
    RefreshGadgets(&concavGadget, ControlWindow, NULL);
    DOWN = DOWNCV; UP = UPCV; LEFT = LEFTCV; RIGHT = RIGHTCV;
    }
  else
    {
    resmirror = CONVEX;
    micr &= CONVEX;
    op = RemoveGadget(ControlWindow, &convexGadget);
    convexGadget.Flags |= SELECTED;
    AddGadget(ControlWindow, &convexGadget, op);
    op = RemoveGadget(ControlWindow, &concavGadget);
    concavGadget.Flags &= (SELECTED ^ 0xffff);
    AddGadget(ControlWindow, &concavGadget, op);
    RefreshGadgets(&concavGadget, ControlWindow, NULL);
    DOWN = DOWNCX; UP = UPCX; LEFT = LEFTCX; RIGHT = RIGHTCX;
    }
  dispcount();
  }

/* Thid function intializes the gadget images and allocates chip memory for
  them. */
short initimages(void)
  {
  short i;

  if ((convexIData_chip = (USHORT *)
    AllocMem(convexDsize*2,MEMF_CHIP)) == 0) return(FALSE);
  if ((concavIData_chip = (USHORT *)
    AllocMem(concavDsize*2,MEMF_CHIP)) == 0) return(FALSE);
  if ((upIData_chip = (USHORT *)
    AllocMem(upDsize*2,MEMF_CHIP)) == 0) return(FALSE);
  if ((downIData_chip = (USHORT *)
    AllocMem(downDsize*2,MEMF_CHIP)) == 0) return(FALSE);
  if ((leftIData_chip = (USHORT *)
    AllocMem(leftDsize*2,MEMF_CHIP)) == 0) return(FALSE);
  if ((rightIData_chip = (USHORT *)
    AllocMem(rightDsize*2,MEMF_CHIP)) == 0) return(FALSE);
  if ((pwupIData_chip = (USHORT *)
    AllocMem(upDsize*2,MEMF_CHIP)) == 0) return(FALSE);
  if ((pwdownIData_chip = (USHORT *)
    AllocMem(downDsize*2,MEMF_CHIP)) == 0) return(FALSE);

  for (i=0; i<convexDsize; i++) convexIData_chip[i] = 0;
  for (i=0; i<concavDsize; i++) concavIData_chip[i] = 0;
  for (i=0; i<upDsize; i++) upIData_chip[i] = upData[i];
  for (i=0; i<downDsize; i++) downIData_chip[i] = downData[i];
  for (i=0; i<leftDsize; i++) leftIData_chip[i] = leftData[i];
  for (i=0; i<rightDsize; i++) rightIData_chip[i] = rightData[i];
  for (i=0; i<upDsize; i++) pwupIData_chip[i] = upData[i];
  for (i=0; i<downDsize; i++) pwdownIData_chip[i] = downData[i];

  convexImage.ImageData = convexIData_chip;
  concavImage.ImageData = concavIData_chip;
  upImage.ImageData = upIData_chip;
  downImage.ImageData = downIData_chip;
  leftImage.ImageData = leftIData_chip;
  rightImage.ImageData = rightIData_chip;
  pwupImage.ImageData = pwupIData_chip;
  pwdownImage.ImageData = pwdownIData_chip;
  return(TRUE);
  }

/* Deallocate the memory that was used for gadget images */
void freeimages(void)
  {
  if (convexIData_chip != 0)
    { FreeMem(convexIData_chip,2*convexDsize); convexIData_chip = 0; }
  if (concavIData_chip != 0)
    { FreeMem(concavIData_chip,2*concavDsize); concavIData_chip = 0; }
  if (upIData_chip != 0)
    { FreeMem(upIData_chip,2*upDsize); upIData_chip = 0; }
  if (downIData_chip != 0)
    { FreeMem(downIData_chip,2*downDsize); downIData_chip = 0; }
  if (leftIData_chip != 0)
    { FreeMem(leftIData_chip,2*leftDsize); leftIData_chip = 0; }
  if (rightIData_chip != 0)
    { FreeMem(rightIData_chip,2*rightDsize); rightIData_chip = 0; }
  if (pwupIData_chip != 0)
    { FreeMem(pwupIData_chip,2*upDsize); pwupIData_chip = 0; }
  if (pwdownIData_chip != 0)
    { FreeMem(pwdownIData_chip,2*downDsize); pwdownIData_chip = 0; }
  }

/* This function closes all windows and libraries. and deallocates memory. */
void cleanup(short wait)
  {
  livecleanup();
  audiocleanup();
  closesystimer();
  if (PlotWindow != NULL) { CloseWindow(PlotWindow); PlotWindow = NULL; }
  if (menuflg) { ClearMenuStrip(ControlWindow); menuflg = 0; }
  if (ControlWindow != NULL) { CloseWindow(ControlWindow); ControlWindow = NULL;
                        }
  if (wait && WB) Delay(200);
  if (image != NULL) { free(image); image = NULL; }
  freeimages();
  if (GfxBase != NULL) { CloseLibrary(GfxBase); GfxBase = NULL; }
  if (IntuitionBase != NULL)
    { CloseLibrary(IntuitionBase); IntuitionBase = NULL; }
  }

/* This function performs the cleanup if an error occurs within the main
  control routine while allocating memory and opening files. */
void errcleanup(void)
  {
  if (image != NULL) { free(image); image = NULL; }
  if (file1) { close(file1); file1 = 0; }
  if (file2 != NULL) { fclose(file2); file2 = NULL; }
  }

/* This function enables gadgets and menu items which become valid when the
  program is running (i.e. grabbing images) and disables the others. */
void EnableRunGadg(void)
  {
  pauseflg = 0;
  OffMenu(ControlWindow,0x0000);          /* start item off */
  OffMenu(ControlWindow,0x0040);         /* resume item off */
  OffGadget(&convexGadget,ControlWindow,NULL);  /* mirror gadgets off */
  OffGadget(&concavGadget,ControlWindow,NULL);
  OnMenu(ControlWindow,0x0020);           /* pause item on */
  OnMenu(ControlWindow,0x0060);          /* stop item on */
  }

/* This function disables gadgets and menu items which become invalid when
  the program is not running (i.e. not grabbing images) and enables the
  others. */
void DisableRunGadg(void)
  {
  pauseflg = 1;
  OffMenu(ControlWindow,0x0020);        /* pause item off */
  OffMenu(ControlWindow,0x0040);       /* resume item off */
  OffMenu(ControlWindow,0x0060);         /* stop item off */
  OnMenu(ControlWindow,0x0000);          /* start item on */
  OnGadget(&convexGadget,ControlWindow,NULL);  /* mirror gadgets on */
  OnGadget(&concavGadget,ControlWindow,NULL);
```

```
  }

/* This function initializes the text buffers of string gadgets to their
  correct value. */
void initgadgbufs(void)
  {
  sprintf(yalggainbuf,"%4.2f",yalggain);
  sprintf(xalggainbuf,"%4.2f",xalggain);
  sprintf(powerbuf,"%5d",powerset);
  sprintf(yalgminbuf,"%3.1f",yalgmin/10.0);
  sprintf(xalgminbuf,"%3.1f",xalgmin/10.0);
  sprintf(pminbuf,"%3d",pmin);
  sprintf(yalgoffbuf,"%4.1f",((float)yalgoff)/KILO);
  sprintf(xalgoffbuf,"%4.1f",((float)xalgoff)/KILO);
  sprintf(pconvoffbuf,"%4.1f",((float)pconvoff)/KILO);
  sprintf(pwrgainbuf,"%4.1f",pwrgain);
  sprintf(tipwrbuf,"%4.1f",tipwr);
  sprintf(pconvgainbuf,"%4.1f",pconvgain*KILO);
  sprintf(xminbuf,"%3d",xminh);
  sprintf(xmaxbuf,"%3d",xmaxh);
  sprintf(yminbuf,"%3d",yminh);
  sprintf(ymaxbuf,"%3d",ymaxh);
  sprintf(xcentbuf,"%3d",xcenth);
  sprintf(ycentbuf,"%3d",ycenth);
  sprintf(ceilbuf,"%2d",maxlev);
  sprintf(floorbuf,"%2d",minlev);
  sprintf(tsampbuf,"%4.1f",tsamp);
  sprintf(uwandbuf,"%5.2f",uwand);
  sprintf(fcutbuf,"%4.2f",fcutoff);
  strcpy(imgfilebuf,imagefile);
  strcpy(datfilebuf,datafile);
  }

void updategadgbufs(void)
  {
  sprintf(ceilbuf,"%2d",maxlev);
  sprintf(floorbuf,"%2d",minlev);
  }

/* This function checks if the input values to the string gadgets are
  within acceptable ranges. */
short validinputs(void)
  {
  short error;
```

```
error = 1;
if (atof(yalggainbuf) < MINVGN || atof(yalggainbuf) > MAXVGN)
  { printm("Invalid vertical (Y) gain - reenter"); error = 0; }
else yalggain = atof(yalggainbuf);
if (atof(xalggainbuf) < MINHGN || atof(xalggainbuf) > MAXHGN)
  { printm("Invalid horizontal (X) gain - reenter"); error = 0; }
else xalggain = atof(xalggainbuf);
if (atof(pconvgainbuf) > MAXPCONV || atof(pconvgainbuf) < MINPCONV)
  { printm("Invalid power conversion gain - reenter"); error = 0; }
else pconvgain = atof(pconvgainbuf)/KILO;
if (atof(pwrgainbuf) > MAXPGAIN || atof(pwrgainbuf) < MINPGAIN)
  { printm("Invalid power (P) gain - reenter"); error = 0; }
else pwrgain = atof(pwrgainbuf);
if (atof(tipwrbuf) > MAXTIPWR || atof(tipwrbuf) < tsamp)
  { printm("Invalid power (P) gain - reenter"); error = 0; }
else tipwr = atof(tipwrbuf);
if (atoi(ceilbuf) > MAXCEIL || atoi(ceilbuf) < MINCEIL)
  { printm("Invalid ceiling for digitizer - reenter"); error = 0; }
else maxlev = (unsigned char) atoi(ceilbuf);
if (atoi(floorbuf) > MAXFLR || atoi(floorbuf) < MINFLR)
  { printm("Invalid floor for digitizer - reenter"); error = 0; }
else minlev = (unsigned char) atoi(floorbuf);
if (atoi(xminbuf) < 0 || atoi(xminbuf) >= (WIDTH/2))
  { printm("Invalid value for Xmin - reenter"); error = 0; }
else if (atoi(xmaxbuf) < (WIDTH/2) || atoi(xmaxbuf) >= WIDTH)
  { printm("Invalid value for Xmax - reenter"); error = 0; }
else if ((atoi(xmaxbuf) - atoi(xminbuf)) < MINBEAMW)
  { printm("Invalid Xmin or Xmax - reenter"); error = 0; }
else { xmaxh = atoi(xmaxbuf); xminh = atoi(xminbuf); }
if (atoi(yminbuf) < 0 || atoi(yminbuf) >= (HEIGHT/2))
  { printm("Invalid value for Ymin - reenter"); error = 0; }
else if (atoi(ymaxbuf) < (HEIGHT/2) || atoi(ymaxbuf) >= HEIGHT)
  { printm("Invalid value for Ymax - reenter"); error = 0; }
else if ((atoi(ymaxbuf) - atoi(yminbuf)) < MINBEAMH)
  { printm("Invalid Ymin or Ymax - reenter"); error = 0; }
else { ymaxh = atoi(ymaxbuf); yminh = atoi(yminbuf); }
if (atoi(xcentbuf) < (xminh + MINBEAMW/2) || atoi(xcentbuf) > (xmaxh -
                      MINBEAMW/2))
  { printm("Invalid value for X-centre - reenter"); error = 0; }
else xcenth = atoi(xcentbuf);
if (atoi(ycentbuf) < (yminh + MINBEAMH/2) || atoi(ycentbuf) > (ymaxh + MINBEAMH/2))
  { printm("Invalid value for Y-centre - reenter"); error = 0; }
else ycenth = atoi(ycentbuf);
```

```
 if (atof(yalgminbuf) < (MINCNT/10.0) || atof(yalgminbuf) > MAXVMIN)
   { printm("Invalid minimum Y-error - reenter"); error = 0; }
 else yalgmin = (short)(atof(yalgminbuf) * 10);
 if (atof(xalgminbuf) < (MINCNT/10.0) || atof(xalgminbuf) > MAXHMIN)
   { printm("Invalid minimum X-error - reenter"); error = 0; }
 else xalgmin = (short)(atof(xalgminbuf) * 10);
 if (atoi(pminbuf) < 0 || atoi(pminbuf) > MAXPMIN)
   { printm("Invalid minimum P-error - reenter"); error = 0; }
 else pmin = atoi(pminbuf);
 if ((long)(fabs(atof(yalgoffbuf)) * KILO) > MAXVOFF)
   { printm("Invalid Y-offset - reenter"); error = 0; }
 else yalgoff = atof(yalgoffbuf) * KILO + 0.5;
 if ((long)(fabs(atof(xalgoffbuf)) * KILO) > MAXHOFF)
   { printm("Invalid X-offset - reenter"); error = 0; }
 else xalgoff = atof(xalgoffbuf) * KILO + 0.5;
 if ((long)(fabs(atof(pconvoffbuf)) * KILO) > MAXPOFF)
   { printm("Invalid P-conversion offset - reenter"); error = 0; }
 else pconvoff = (long)(atof(pconvoffbuf) * KILO + 0.5);
 if (atoi(powerbuf) < MINPWR || atoi(powerbuf) > MAXPWR)
   { printm("Invalid power setting - reenter"); error = 0; }
 else powerset = atoi(powerbuf);
 if (atof(tsampbuf) < MINSAMP || atof(tsampbuf) > MAXSAMP)
   { printm("Invalid samplig period - reenter"); error = 0; }
 else tsamp = atof(tsampbuf);
 if (atof(uwandbuf) < 0.0 || atof(uwandbuf) > UWNDMAX)
   { printm("Invalid wand controller setpoint - reenter"); error = 0; }
 else uwand = atof(uwandbuf);
 if (atof(fcutbuf) < FCUTMIN || atof(fcutbuf) > FCUTMAX)
   { printm("Invalid cutoff frequency - reenter"); error = 0; }
 else fcutoff = atof(fcutbuf);
 return(error);
 }

/* This function highlights the appropriate micrometer direction gadget when
   a control movement is being performed. */
void dirhighlight(void)
 {
 if (micdir == UP) {
   op = RemoveGadget(ControlWindow, &upGadget);
   upGadget.Flags ^= SELECTED;
   AddGadget(ControlWindow, &upGadget, op);
   RefreshGadgets(&upGadget, ControlWindow, NULL); }
 else if (micdir == DOWN) {
   op = RemoveGadget(ControlWindow, &downGadget);
```

```
    downGadget.Flags ^= SELECTED;
    AddGadget(ControlWindow, &downGadget, op);
    RefreshGadgets(&downGadget, ControlWindow, NULL); }
  else if (micdir == LEFT) {
    op = RemoveGadget(ControlWindow, &leftGadget);
    leftGadget.Flags ^= SELECTED;
    AddGadget(ControlWindow, &leftGadget, op);
    RefreshGadgets(&leftGadget, ControlWindow, NULL); }
  else if (micdir == RIGHT) {
    op = RemoveGadget(ControlWindow, &rightGadget);
    rightGadget.Flags ^= SELECTED;
    AddGadget(ControlWindow, &rightGadget, op);
    RefreshGadgets(&rightGadget, ControlWindow, NULL); }
  }

/* This function highlights the appropriate power direction gadget when
   a control movement is being performed. */
void powhighlight(short pwdir)
  {
  switch (pwdir)
    {
    case PWUP  : op = RemoveGadget(ControlWindow, &pwupGadget);
         pwupGadget.Flags ^= SELECTED;
         AddGadget(ControlWindow, &pwupGadget, op);
         RefreshGadgets(&pwupGadget, ControlWindow, NULL);
         break;
    case PWDOWN: op = RemoveGadget(ControlWindow, &pwdownGadget);
         pwdownGadget.Flags ^= SELECTED;
         AddGadget(ControlWindow, &pwdownGadget, op);
         RefreshGadgets(&pwdownGadget, ControlWindow, NULL);
         break;
    }
  }

/* This function sets the correct options for the menu items based on data
   read from the parameter file. */
void setoptions(void)
  {
  if (camera == TOPCAM) {
    topcamSub.Flags |= CHECKED;
    botcamSub.Flags &= (CHECKED ^ 0xffff); }
  else {
    botcamSub.Flags |= CHECKED;
    topcamSub.Flags &= (CHECKED ^ 0xffff); }
```

```
  if (contflg) {
    contOnSub.Flags |= CHECKED;
    contOffSub.Flags &= (CHECKED ^ 0xffff); }
  else {
    contOffSub.Flags |= CHECKED;
    contOnSub.Flags &= (CHECKED ^ 0xffff); }
  if (steerflg) {
    steerOnSub.Flags |= CHECKED;
    steerOffSub.Flags &= (CHECKED ^ 0xffff); }
  else {
    steerOffSub.Flags |= CHECKED;
    steerOnSub.Flags &= (CHECKED ^ 0xffff); }
  if (powflg) {
    powrOnSub.Flags |= CHECKED;
    powrOffSub.Flags &= (CHECKED ^ 0xffff); }
  else {
    powrOffSub.Flags |= CHECKED;
    powrOnSub.Flags &= (CHECKED ^ 0xffff); }
  if (wandflg) {
    wandOnSub.Flags |= CHECKED;
    wandOffSub.Flags &= (CHECKED ^ 0xffff); }
  else {
    wandOffSub.Flags |= CHECKED;
    wandOnSub.Flags &= (CHECKED ^ 0xffff); }
  if (dataflg) {
    dataYesSub.Flags |= CHECKED;
    dataNoSub.Flags &= (CHECKED ^ 0xffff); }
  else {
    dataNoSub.Flags |= CHECKED;
    dataYesSub.Flags &= (CHECKED ^ 0xffff); }
  if (imageflg == 1) {
    imageNoSub.Flags &= (CHECKED ^ 0xffff);
    imageOvSub.Flags |= CHECKED;
    imageSqSub.Flags &= (CHECKED ^ 0xffff); }
  else if (imageflg == 2) {
    imageNoSub.Flags &= (CHECKED ^ 0xffff);
    imageOvSub.Flags &= (CHECKED ^ 0xffff);
    imageSqSub.Flags |= CHECKED; }
  else {
    imageNoSub.Flags |= CHECKED;
    imageOvSub.Flags &= (CHECKED ^ 0xffff);
    imageSqSub.Flags &= (CHECKED ^ 0xffff); }
  if (centflg) {
    centYesSub.Flags |= CHECKED;
  centNoSub.Flags &= (CHECKED ^ 0xffff); }
 else {
  centNoSub.Flags |= CHECKED;
  centYesSub.Flags &= (CHECKED ^ 0xffff); }
 if (colorflg) {
  colColSub.Flags |= CHECKED;
  colBWSub.Flags &= (CHECKED ^ 0xffff); }
 else {
  colBWSub.Flags |= CHECKED;
  colColSub.Flags &= (CHECKED ^ 0xffff); }
 if (filtflg) {
  filtYesSub.Flags |= CHECKED;
  filtNoSub.Flags &= (CHECKED ^ 0xffff); }
 else {
  filtNoSub.Flags |= CHECKED;
  filtYesSub.Flags &= (CHECKED ^ 0xffff); }
 if (plotflg[NUMSIGS]) {
  if (plotflg[XALG]) {
  plotXerrSub.Flags |= CHECKED;
  plotNoneSub.Flags &= (CHECKED ^ 0xffff); }
  else plotXerrSub.Flags &= (CHECKED ^ 0xffff);
  if (plotflg[YALG]) {
  plotYerrSub.Flags |= CHECKED;
  plotNoneSub.Flags &= (CHECKED ^ 0xffff); }
  else plotYerrSub.Flags &= (CHECKED ^ 0xffff);
  if (plotflg[XPOS]) {
  plotXposSub.Flags |= CHECKED;
  plotNoneSub.Flags &= (CHECKED ^ 0xffff); }
  else plotXposSub.Flags &= (CHECKED ^ 0xffff);
  if (plotflg[YPOS]) {
  plotYposSub.Flags |= CHECKED;
  plotNoneSub.Flags &= (CHECKED ^ 0xffff); }
  else plotYposSub.Flags &= (CHECKED ^ 0xffff);
  if (plotflg[SPWR]) {
  plotSpwrSub.Flags |= CHECKED;
  plotNoneSub.Flags &= (CHECKED ^ 0xffff); }
  else plotSpwrSub.Flags &= (CHECKED ^ 0xffff);
  if (plotflg[MPWR]) {
  plotMpwrSub.Flags |= CHECKED;
  plotNoneSub.Flags &= (CHECKED ^ 0xffff); }
  else plotMpwrSub.Flags &= (CHECKED ^ 0xffff); }
 else {
  plotNoneSub.Flags |= CHECKED;
  plotXerrSub.Flags &= (CHECKED ^ 0xffff);
```

```
  plotYerrSub.Flags &= (CHECKED ^ 0xffff);
  plotXposSub.Flags &= (CHECKED ^ 0xffff);
  plotYposSub.Flags &= (CHECKED ^ 0xffff);
  plotSpwrSub.Flags &= (CHECKED ^ 0xffff);
  plotMpwrSub.Flags &= (CHECKED ^ 0xffff); }
 if (pwmetflg || curflg || voltflg) {
  if (pwmetflg) {
  pwmetSub.Flags |= CHECKED;
  monNoneSub.Flags &= (CHECKED ^ 0xffff); }
  else pwmetSub.Flags &= (CHECKED ^ 0xffff);
  if (curflg) {
  crmetSub.Flags |= CHECKED;
  monNoneSub.Flags &= (CHECKED ^ 0xffff); }
  else crmetSub.Flags &= (CHECKED ^ 0xffff);
  if (voltflg) {
  voltsSub.Flags |= CHECKED;
  monNoneSub.Flags &= (CHECKED ^ 0xffff); }
  else voltsSub.Flags &= (CHECKED ^ 0xffff); }
 else {
  monNoneSub.Flags |= CHECKED;
  pwmetSub.Flags &= (CHECKED ^ 0xffff);
  crmetSub.Flags &= (CHECKED ^ 0xffff);
  voltsSub.Flags &= (CHECKED ^ 0xffff); }
 if (!sensflg) {
  nosenSub.Flags |= CHECKED;
  cxsenSub.Flags &= (CHECKED ^ 0xffff);
  cvsenSub.Flags &= (CHECKED ^ 0xffff);
  pwsenSub.Flags &= (CHECKED ^ 0xffff);
  wandsenSub.Flags &= (CHECKED ^ 0xffff);
  possenSub.Flags &= (CHECKED ^ 0xffff); }
 }

/* This function opens the plot window for plotting signals. */
void openplotwind(void)
 {
 plotflg[NUMSIGS] = 1;
 if (PlotWindow == NULL)
   if ((PlotWindow = OpenWindow(&NewPlotWindow)) == NULL)
   { printm("Cannot open plot window"); plotflg[NUMSIGS] = 0; }
   else { displot(); plotlegend(); }
 else plotlegend();
 }

/* This function closes the plot window when no signals are to be plotted. */
```

```
void closeplotwind(void)
  {
  plotflg[NUMSIGS] = 0;
  if (PlotWindow) { CloseWindow(PlotWindow); PlotWindow = 0; }
  }

/* This function checks the plotflg array to check if any signals are to be
  plotted. */
short chkplotflg(void)
  {
  for (i=0; i<NUMSIGS; i++) if (plotflg[i]) return(1);
  return(0);
  }
```

D.3 DISPLAY.C

```
/*************************************************************************
This source file contains the variables and functions required to display
the computed data and plot the selected signals.

Filename: "DISPLAY.C", Version: 2.1, Date: May 5, 1992

*************************************************************************/

#include <stdlib.h>
#include <exec/exec.h>
#include <intuition/intuition.h>
#include <intuition/intuitionbase.h>
#include <graphics/gfxbase.h>
#include <functions.h>
#include <math.h>
#include "control.h"
#include "display.h"

/* Variables and functions defined in "control.c" */
extern long countvtx,counthzx,countvtv,counthzv,intensity;
extern short micr,actpower,q1power,q2power,q3power,q4power,xcentd,ycentd,
      xcentr,ycentr,xmin,xmax,ymin,ymax,pwcount,xalgperr,yalgperr,
      xcenth,ycenth,micmove;
extern double runtime;
extern float powmeter,current,voltage,upwr,uwand;

/* Variables and functions defined in "contgadc.c" */
extern struct Window *ControlWindow;
extern struct Window *PlotWindow;

/* Variables and functions defined in this file */
void dispcount(),dispdata(),dispwtext(),displot(),printm(char *str),
    plotsignals(),setsigvals(),scrollplot(),plotlegend(),clearmsgarea();
static short i,      /* temporary loop counter variable */
      msgpen;    /* pen for dislaying messages */
static char countstr[9],   /* string to display micr. counts on screen */
      tempstr[9];    /* string to display power and other values */
double runmin,runsec;   /* minutes and seconds of total run time */
short plotflg[NUMSIGS+1],   /* flags to determine which signals to plot */
            /* if plotflg[NUMSIGS]==0, no signals plotted */
```

```
  plotval[NUMSIGS],    /* array of signal values to be plotted */
  lastplotval[NUMSIGS]; /* array of last signal values plotted */
short plotxlast,     /* last x-value of plot */
  plotxval,      /* current x-value of plot */
  plotcnt;       /* temporary count of # of signals plotted */

/* Arrays of pen colours, line types, and legends for plotted signals */
short plotpen[NUMSIGS] = { WHITE, BLUE, BLACK, WHITE };
short plotline[NUMSIGS] = { -1, -1, -1, 0xaaaa, 0xaaaa, 0xaaaa };
char *plotsignames[NUMSIGS] = { "XUnf","YUnf","XPos","YPos","SPwr","PMet" };

/* Array of coordinates which define outline of plotting window */
short graphwind[8] =
                    {GWXMAX,GWYMIN,GWXMAX,GWYMAX,GWXMIN,GWYMAX,G
                    WXMIN,GWYMIN};

/* This function displays the micrometer counts. */
void dispcount(void)
  {
  if (micr & CONCAVE) {
    SetAPen(ControlWindow->RPort,WHITE);
    SetBPen(ControlWindow->RPort,BLUE); }
  else {
    SetAPen(ControlWindow->RPort,BLACK);
    SetBPen(ControlWindow->RPort,RED); }
  SetDrMd(ControlWindow->RPort,JAM2);
  Move(ControlWindow->RPort,66,73);
  sprintf(countstr,"%7.1f",countvtx/10.0);
  Text(ControlWindow->RPort,countstr,7);
  Move(ControlWindow->RPort,128,73);
  sprintf(countstr,"%7.1f",counthzx/10.0);
  Text(ControlWindow->RPort,countstr,7);
  if (micr & CONCAVE) {
    SetAPen(ControlWindow->RPort,BLACK);
    SetBPen(ControlWindow->RPort,RED); }
  else {
    SetAPen(ControlWindow->RPort,WHITE);
    SetBPen(ControlWindow->RPort,BLUE); }
  SetDrMd(ControlWindow->RPort,JAM2);
  Move(ControlWindow->RPort,66,83);
  sprintf(countstr,"%7.1f",countvtv/10.0);
  Text(ControlWindow->RPort,countstr,7);
  Move(ControlWindow->RPort,128,83);
  sprintf(countstr,"%7.1f",counthzv/10.0);
  Text(ControlWindow->RPort,countstr,7);
  }

/* This function displays all important data and control actions. */
void dispdata(void)
  {
  SetAPen(ControlWindow->RPort,BLACK);
  SetBPen(ControlWindow->RPort,RED);
  SetDrMd(ControlWindow->RPort,JAM2);
  Move(ControlWindow->RPort,72,133);
  sprintf(tempstr,"%5d",actpower);
  Text(ControlWindow->RPort,tempstr,5);
  Move(ControlWindow->RPort,72,148);
  sprintf(tempstr,"%5d",(int)(powmeter*KILO));
  Text(ControlWindow->RPort,tempstr,5);
  Move(ControlWindow->RPort,226,25);
  sprintf(tempstr,"%4d",q2power);
  Text(ControlWindow->RPort,tempstr,4);
  Move(ControlWindow->RPort,290,25);
  sprintf(tempstr,"%4d",q1power);
  Text(ControlWindow->RPort,tempstr,4);
  Move(ControlWindow->RPort,226,40);
  sprintf(tempstr,"%4d",q3power);
  Text(ControlWindow->RPort,tempstr,4);
  Move(ControlWindow->RPort,290,40);
  sprintf(tempstr,"%4d",q4power);
  Text(ControlWindow->RPort,tempstr,4);
  Move(ControlWindow->RPort,368,25);
  sprintf(tempstr,"%5d",xalgperr);
  Text(ControlWindow->RPort,tempstr,5);
  Move(ControlWindow->RPort,368,40);
  sprintf(tempstr,"%5d",yalgperr);
  Text(ControlWindow->RPort,tempstr,5);
  Move(ControlWindow->RPort,440,25);
  sprintf(tempstr,"%3d",(xcenth-xcentd));
  Text(ControlWindow->RPort,tempstr,3);
  Move(ControlWindow->RPort,440,40);
  sprintf(tempstr,"%3d",(ycentd-ycenth));
  Text(ControlWindow->RPort,tempstr,3);

  SetAPen(ControlWindow->RPort,BLACK);
  SetBPen(ControlWindow->RPort,RED);
  SetDrMd(ControlWindow->RPort,JAM2);
  Move(ControlWindow->RPort,218,70);
```

```
sprintf(tempstr,"%3d",xcentd);
Text(ControlWindow->RPort,tempstr,3);
Move(ControlWindow->RPort,218,85);
sprintf(tempstr,"%3d",ycentd);
Text(ControlWindow->RPort,tempstr,3);
Move(ControlWindow->RPort,314,70);
sprintf(tempstr,"%3d",xcentr);
Text(ControlWindow->RPort,tempstr,3);
Move(ControlWindow->RPort,314,85);
sprintf(tempstr,"%3d",ycentr);
Text(ControlWindow->RPort,tempstr,3);
Move(ControlWindow->RPort,410,70);
sprintf(tempstr,"%3d",xmin);
Text(ControlWindow->RPort,tempstr,3);
Move(ControlWindow->RPort,410,85);
sprintf(tempstr,"%3d",ymin);
Text(ControlWindow->RPort,tempstr,3);
Move(ControlWindow->RPort,446,70);
sprintf(tempstr,"%3d",xmax);
Text(ControlWindow->RPort,tempstr,3);
Move(ControlWindow->RPort,446,85);
sprintf(tempstr,"%3d",ymax);
Text(ControlWindow->RPort,tempstr,3);

Move(ControlWindow->RPort,250,115);
sprintf(tempstr,"%5.1f",micmove/10.0);
Text(ControlWindow->RPort,tempstr,5);
Move(ControlWindow->RPort,250,130);
sprintf(tempstr,"%5.2f",upwr);
Text(ControlWindow->RPort,tempstr,5);
Move(ControlWindow->RPort,250,145);
sprintf(tempstr,"%5.2f",uwand);
Text(ControlWindow->RPort,tempstr,5);
Move(ControlWindow->RPort,430,100);
sprintf(tempstr,"%5.1f",1.0*intensity/KILO);
Text(ControlWindow->RPort,tempstr,5);
Move(ControlWindow->RPort,430,115);
sprintf(tempstr,"%5.2f",current);
Text(ControlWindow->RPort,tempstr,5);
Move(ControlWindow->RPort,430,130);
sprintf(tempstr,"%5.2f",voltage);
Text(ControlWindow->RPort,tempstr,5);
Move(ControlWindow->RPort,430,145);
runsec = modf(runtime/60.0,&runmin) * 60.0;
```

```
  sprintf(tempstr,"%2d:%02d",(short)runmin,(short)runsec),
  Text(ControlWindow->RPort,tempstr,5);
  }

/* This function displays any text which is unchanging */
void dispwtext(void)
  {
  SetAPen(ControlWindow->RPort,WHITE);
  SetDrMd(ControlWindow->RPort,JAM1);
  Move(ControlWindow->RPort,10,103);
  Text(ControlWindow->RPort,"Output Power:",13);
  Move(ControlWindow->RPort,10,118);
  Text(ControlWindow->RPort,"Set Pt:",7);
  Move(ControlWindow->RPort,10,133);
  Text(ControlWindow->RPort,"Screen:",7);
  Move(ControlWindow->RPort,120,133);
  Text(ControlWindow->RPort,"W",1);
  Move(ControlWindow->RPort,18,148);
  Text(ControlWindow->RPort,"Meter:",6);
  Move(ControlWindow->RPort,120,148);
  Text(ControlWindow->RPort,"W",1);
  Move(ControlWindow->RPort,200,10);
  Text(ControlWindow->RPort,"Quadrant Powers:",16);
  Move(ControlWindow->RPort,200,25);
  Text(ControlWindow->RPort,"Q2:",3);
  Move(ControlWindow->RPort,264,25);
  Text(ControlWindow->RPort,"Q1:",3);
  Move(ControlWindow->RPort,200,40);
  Text(ControlWindow->RPort,"Q3:",3);
  Move(ControlWindow->RPort,264,40);
  Text(ControlWindow->RPort,"Q4:",3);
  Move(ControlWindow->RPort,328,25);
  Text(ControlWindow->RPort,"W",1);
  Move(ControlWindow->RPort,328,40);
  Text(ControlWindow->RPort,"W",1);
  Move(ControlWindow->RPort,350,10);
  Text(ControlWindow->RPort,"Align & Pos Errors:",19);
  Move(ControlWindow->RPort,350,25);
  Text(ControlWindow->RPort,"X:",2);
  Move(ControlWindow->RPort,350,40);
  Text(ControlWindow->RPort,"Y:",2);
  Move(ControlWindow->RPort,416,25);
  Text(ControlWindow->RPort,"W",1);
  Move(ControlWindow->RPort,416,40);
```

```
Text(ControlWindow->RPort,"W",1);
Move(ControlWindow->RPort,472,25);
Text(ControlWindow->RPort,"Pix",3);
Move(ControlWindow->RPort,472,40);
Text(ControlWindow->RPort,"Pix",3);

Move(ControlWindow->RPort,200,55);
Text(ControlWindow->RPort,"Centre:",7);
Move(ControlWindow->RPort,200,70);
Text(ControlWindow->RPort,"X:",2);
Move(ControlWindow->RPort,200,85);
Text(ControlWindow->RPort,"Y:",2);
Move(ControlWindow->RPort,246,70);
Text(ControlWindow->RPort,"Pix",3);
Move(ControlWindow->RPort,246,85);
Text(ControlWindow->RPort,"Pix",3);
Move(ControlWindow->RPort,296,55);
Text(ControlWindow->RPort,"Centroid:",9);
Move(ControlWindow->RPort,296,70);
Text(ControlWindow->RPort,"X:",2);
Move(ControlWindow->RPort,296,85);
Text(ControlWindow->RPort,"Y:",2);
Move(ControlWindow->RPort,342,70);
Text(ControlWindow->RPort,"Pix",3);
Move(ControlWindow->RPort,342,85);
Text(ControlWindow->RPort,"Pix",3);
Move(ControlWindow->RPort,392,55);
Text(ControlWindow->RPort,"Beam Size:",10);
Move(ControlWindow->RPort,392,70);
Text(ControlWindow->RPort,"X:",2);
Move(ControlWindow->RPort,392,85);
Text(ControlWindow->RPort,"Y:",2);
Move(ControlWindow->RPort,436,70);
Text(ControlWindow->RPort,"-",1);
Move(ControlWindow->RPort,436,85);
Text(ControlWindow->RPort,"-",1);
Move(ControlWindow->RPort,474,70);
Text(ControlWindow->RPort,"Pix",3);
Move(ControlWindow->RPort,474,85);
Text(ControlWindow->RPort,"Pix",3);

Move(ControlWindow->RPort,200,100);
Text(ControlWindow->RPort,"Control Action:",15);
Move(ControlWindow->RPort,200,115);
```

```
 Text(ControlWindow->RPort," Mics:",6);
 Move(ControlWindow->RPort,200,130);
 Text(ControlWindow->RPort,"Power:",6);
 Move(ControlWindow->RPort,200,145);
 Text(ControlWindow->RPort," Wand:",6);
 Move(ControlWindow->RPort,294,115);
 Text(ControlWindow->RPort,"um",2);
 Move(ControlWindow->RPort,294,130);
 Text(ControlWindow->RPort,"V",1);
 Move(ControlWindow->RPort,294,145);
 Text(ControlWindow->RPort,"V",1);
 Move(ControlWindow->RPort,348,100);
 Text(ControlWindow->RPort,"Intensity:",10);
 Move(ControlWindow->RPort,474,100);
 Text(ControlWindow->RPort,"kgv",3);
 Move(ControlWindow->RPort,348,115);
 Text(ControlWindow->RPort,"  Current:",10);
 Move(ControlWindow->RPort,474,115);
 Text(ControlWindow->RPort,"A",1);
 Move(ControlWindow->RPort,348,130);
 Text(ControlWindow->RPort,"  Voltage:",10);
 Move(ControlWindow->RPort,474,130);
 Text(ControlWindow->RPort,"kV",2);
 Move(ControlWindow->RPort,348,145);
 Text(ControlWindow->RPort,"  Runtime:",10);
 Move(ControlWindow->RPort,474,145);
 Text(ControlWindow->RPort,"Min",3);

 /* draw line to separate error message display area */
 Move(ControlWindow->RPort,0,154);
 Draw(ControlWindow->RPort,CNWIDTH-8,154);
 msgpen = WHITE;
 }

/* This function displays the plot outline and labels all axes */
void displot(void)
 {
 short i,delta;

 /* set background colour */
 SetAPen(PlotWindow->RPort,RED);
 RectFill(PlotWindow->RPort,0,0,(long)PlotWindow->GZZWidth,
          (long)PlotWindow->GZZHeight);
 SetAPen(PlotWindow->RPort,BLACK);
```

```
SetDrMd(PlotWindow->RPort,JAM1);

/* draw plot window outline and axis ticks */
Move(PlotWindow->RPort,GWXMIN,GWYMIN);
PolyDraw(PlotWindow->RPort,4,&graphwind[0]);
Move(PlotWindow->RPort,GWXMIN,GWYMID);
Draw(PlotWindow->RPort,GWXMAX,GWYMID);
delta = (GWXMAX - GWXMIN) / XTICKS;
for (i=0; i<=XTICKS; i++)
  {
    Move(PlotWindow->RPort,GWXMIN+(i*delta),GWYMAX);
    Draw(PlotWindow->RPort,GWXMIN+(i*delta),GWYMAX+5);
  }
delta = (GWYMAX - GWYMIN) / YTICKS;
for (i=0; i<=YTICKS; i++)
  {
    Move(PlotWindow->RPort,GWXMIN,GWYMIN+(i*delta));
    Draw(PlotWindow->RPort,GWXMIN-5,GWYMIN+(i*delta));
    Move(PlotWindow->RPort,GWXMAX,GWYMIN+(i*delta));
    Draw(PlotWindow->RPort,GWXMAX+5,GWYMIN+(i*delta));
  }

/* print axis and plot titles and scales */
SetAPen(PlotWindow->RPort,BLACK);
SetDrPt(PlotWindow->RPort,-1);
Move(PlotWindow->RPort,GWXMID-82,GWYMIN-4);
Text(PlotWindow->RPort,"Plot Signals vs. Time",21);
Move(PlotWindow->RPort,GWXMID-34,GWYMAX+16);
Text(PlotWindow->RPort,"Time (Min)",10);
Move(PlotWindow->RPort,GWXMIN-31,GWYMIN+3);
sprintf(tempstr,"%+3d",ERSCALE);
Text(PlotWindow->RPort,tempstr,3);
Move(PlotWindow->RPort,GWXMIN-24,GWYMIN+12);
Text(PlotWindow->RPort,"kW",2);
Move(PlotWindow->RPort,GWXMIN-24,GWYMIN+20);
Text(PlotWindow->RPort,"mm",2);
Move(PlotWindow->RPort,GWXMIN-31,GWYMID+3);
Text(PlotWindow->RPort,"  0",3);
Move(PlotWindow->RPort,GWXMIN-31,GWYMAX+3);
sprintf(tempstr,"%+3d",-ERSCALE);
Text(PlotWindow->RPort,tempstr,3);
Move(PlotWindow->RPort,GWXMAX+7,GWYMIN+3);
sprintf(tempstr,"%2d",PMSCALE);
Text(PlotWindow->RPort,tempstr,2);
```

```
  Move(PlotWindow->RPort,GWXMAX+7,GWYMIN+12);
  Text(PlotWindow->RPort,"kW",2);
  Move(PlotWindow->RPort,GWXMAX+7,GWYMID+3);
  sprintf(tempstr,"%-2d",PMSCALE/2);
  Text(PlotWindow->RPort,tempstr,2);
  Move(PlotWindow->RPort,GWXMAX+7,GWYMAX+3);
  Text(PlotWindow->RPort,"0",1);
  Move(PlotWindow->RPort,GWXMIN-4,GWYMAX+16);
  Text(PlotWindow->RPort,"0",1);
  Move(PlotWindow->RPort,GWXMAX-12,GWYMAX+16);
  sprintf(tempstr,"%3.1f",TMSCALE);
  Text(PlotWindow->RPort,tempstr,3);

  /* initialize last values plotted to (0,0) */
  for (i=0; i<NUMSIGS; i++) lastplotval[i] = GWYMID;
  plotxlast = plotxval = GWXMIN;
  }

/* This function scrolls the message area up 1 line and prints a message.
  If the ControlWindow is not open, the message is printed in the CLI. */
void printm(char *msgstr)
  {
  if (ControlWindow == NULL) printf("%s\n",msgstr);
  else {
    SetAPen(ControlWindow->RPort,WHITE);
    SetBPen(ControlWindow->RPort,BLACK);
    SetDrMd(ControlWindow->RPort,JAM1);
    ScrollRaster(ControlWindow->RPort,0,8,0,CNHEIGHT-38,CNWIDTH-8,CNHEIGHT-
                   15);
    if (msgpen == WHITE) {
    msgpen = BLACK;
    SetAPen(ControlWindow->RPort,BLACK);
    SetBPen(ControlWindow->RPort,RED); }
    else {
    msgpen = WHITE;
    SetAPen(ControlWindow->RPort,WHITE);
    SetBPen(ControlWindow->RPort,BLACK); }
    SetDrMd(ControlWindow->RPort,JAM2);
    Move(ControlWindow->RPort,4,180);
    Text(ControlWindow->RPort,msgstr,strlen(msgstr)); }
  }

void clearmsgarea(void)
  {
```

```
  SetAPen(ControlWindow->RPort,BLACK);
  RectFill(ControlWindow->RPort,0,CNHEIGHT-38,CNWIDTH-8,CNHEIGHT-15);
  }

/* This function displays a legend of the chosen signals to be plotted. */
void plotlegend(void)
  {
  short delta;

  plotcnt = 0;
  delta = (GWYMAX - GWYMIN) / YTICKS;
  SetAPen(PlotWindow->RPort,RED);
  RectFill(PlotWindow->RPort,GWXMAX+24,GWYMIN+delta-
                        3,GWXMAX+66,GWYMAX-delta+3);
  for (i=0; i<NUMSIGS; i++)
    if (plotflg[i])
    {
      SetAPen(PlotWindow->RPort,(long)plotpen[plotcnt]);
      SetBPen(PlotWindow->RPort,RED);
      SetDrMd(PlotWindow->RPort,JAM2);
      SetDrPt(PlotWindow->RPort,(long)plotline[plotcnt]);
      Move(PlotWindow->RPort,GWXMAX+34,GWYMIN+((plotcnt+1)*delta)+3);
      Text(PlotWindow->RPort,plotsignames[i],4);
      Move(PlotWindow->RPort,GWXMAX+24,GWYMIN+((plotcnt+1)*delta));
      Draw(PlotWindow->RPort,GWXMAX+32,GWYMIN+((plotcnt+1)*delta));
      plotcnt++;
    }
  }

/* This function plots the selected signals in the plot window. */
void plotsignals(void)
  {
  plotxval += PLTXINC;     /* increment x-value (i.e. time) */
  plotcnt = 0;
  setsigvals();            /* set signal values */
  for (i=0; i<NUMSIGS; i++)
    {
    if (plotflg[i])    /* if signal selected to be plotted */
      {
      /* clip signals if outside plotting area */
      if (plotval[i] >= GWYMAX) plotval[i] = GWYMAX - 1;
      if (plotval[i] <= GWYMIN) plotval[i] = GWYMIN + 1;
      /* scroll plot to the left if end of plot is reached */
      if (plotxval >= GWXMAX) scrollplot();
      /* set pen colour and line type for each signal */
      SetAPen(PlotWindow->RPort,(long)plotpen[plotcnt]);
      SetDrPt(PlotWindow->RPort,(long)plotline[plotcnt]);
      Move(PlotWindow->RPort,(long)plotxlast,(long)lastplotval[i]);
      Draw(PlotWindow->RPort,(long)plotxval,(long)plotval[i]);
      lastplotval[i] = plotval[i];
      plotcnt++;
      }
    }
  plotxlast = plotxval;
  }

/* This function scales and converts the signal values to pixel values for
   plotting. */
void setsigvals(void)
  {
  plotval[XALG] = (short)(-0.001 * xalgperr / ERSCALE * GWYLEN + 0.5) + GWYMID;
  plotval[YALG] = (short)(-0.001 * yalgperr / ERSCALE * GWYLEN + 0.5) + GWYMID;
  plotval[XPOS] = (short)((xcentd-xcenth) * PIXTOMM / ERSCALE * GWYLEN + 0.5) +
                        GWYMID;
  plotval[YPOS] = (short)((ycentd-ycenth) * PIXTOMM / ERSCALE * GWYLEN + 0.5) +
                        GWYMID;
  plotval[SPWR] = (short)(-0.001 * actpower / PMSCALE * 2*GWYLEN + 0.5) +
                        GWYMAX;
  plotval[MPWR] = (short)(-powmeter / PMSCALE * 2*GWYLEN + 0.5) + GWYMAX;
  }

/* This function scrolls the plot to the left by half its total length. */
void scrollplot(void)
  {
  SetAPen(PlotWindow->RPort,BLACK);
  SetBPen(PlotWindow->RPort,RED);
  SetDrMd(PlotWindow->RPort,JAM2);
  SetDrPt(PlotWindow->RPort,-1);
  ScrollRaster(PlotWindow->RPort,GWXLEN/2,0,GWXMIN,GWYMIN+1,GWXMAX-
                        1,GWYMAX-1);
  Move(PlotWindow->RPort,GWXMIN,GWYMIN);
  Draw(PlotWindow->RPort,GWXMIN,GWYMAX);
  Move(PlotWindow->RPort,GWXMID,GWYMID);
  Draw(PlotWindow->RPort,GWXMAX,GWYMID);
  plotxlast -= GWXLEN/2;
  plotxval = plotxlast + PLTXINC;
  }
```

D.4 GRABFRAM.C

```
/*************************************************************************
This source file contains the variables and functions required to
initialize the LIVE hardware and perform the frame grabbing.

Filename: "GRABFRAM.C", Version: 2.1, Date: May 5, 1992

*************************************************************************/

#include <exec/types.h>
#include <fcntl.h>
#include <graphics/gfx.h>
#include <intuition/intuition.h>
#include <functions.h>
#include "LIVEbrary.h"
#include "control.h"

#define DEPTH   5  /* depth of image screen (i.e. 5 bitplanes) */
#define FMT   1  /* format of image (1:low-res, 4:high-res) */
#define NCOLORS 32  /* number of colors in colormap */

/* Variables and functions defined in other dource files */
extern short *LIVEInitialize();         /* in "livebrary.o" */

extern struct IntuitionBase *IntuitionBase;     /* in "contgadg.c" */
extern struct GfxBase *GfxBase;
extern struct TextAttr textfont;

extern unsigned char maxlev,minlev;       /* in "control.c" */

extern void speakmsg(char *str);          /* in "talk.c" */
extern void printm(char *str);       /* in "display.c" */
extern void setcam(long camera);          /* in "setcam.asm" */
extern void bptopix(PLANEPTR *bitplane,UBYTE *rast,long fmt,long depth);
                    /* in "bptopix.asm */

/* The main structures to communicate with LIVE.library */
struct LIVEly lively;
struct LIVEhardwareValues hvalues;
struct LIVEvideoLevel vlevels;      /* levels for ceiling and floor */
struct LIVEbase *LIVEbase;

/* Intuition structures for a custom screen and a backdrop window in it */
struct Screen *screen;
struct Window *backdrop;
struct Window *display;        /* window to display coordinates */
struct IntuiMessage *backdropmsg;       /* window message structure */
ULONG msgclass;           /* variable to store message class */
USHORT msgcode;            /* variable to store message code */

short *gotlive,     /* point to LIVE! hardware if it is initialized */
  endcoords,     /* flag to end coordinate display on window */
  colorflg;     /* set if colorized colormap is to be used */
long camera;     /* camera input to grab (1 or 2) */
static char tempstr[20];   /* text string to write mouse coordinates */
short init_screen(),initlive(),chkvlevels(),getimage(UBYTE *buffer),
  dispimage(),chkvideo();
void livecleanup(),dispcoords(),checkcoords(),showcoords(),replybackdrop(),
   setcolormap();

UWORD bwcompcmap[32] =  /* black and white complemented colormap */
  {
  0xfff, 0xeee, 0xddd, 0xccc, 0xbbb, 0xaaa, 0x999, 0x888,
  0x777, 0x666, 0x555, 0x444, 0x333, 0x222, 0x111, 0x000,
  0xfff, 0xeee, 0xddd, 0xccc, 0xbbb, 0xaaa, 0x999, 0x888,
  0x777, 0x666, 0x555, 0x444, 0x333, 0x222, 0x111, 0x000
  };
UWORD bwcmap[32] =   /* black and white colormap */
  {
  0x000, 0x111, 0x222, 0x333, 0x444, 0x555, 0x666, 0x777,
  0x888, 0x999, 0xaaa, 0xbbb, 0xccc, 0xddd, 0xeee, 0xfff,
  0x000, 0x111, 0x222, 0x333, 0x444, 0x555, 0x666, 0x777,
  0x888, 0x999, 0xaaa, 0xbbb, 0xccc, 0xddd, 0xeee, 0xfff
  };
UWORD colcompcmap[32] =  /* complemented colorized color map */
  {
  0xfff, 0xf00, 0xf99, 0xf90, 0xff0, 0xdd0, 0x8f8, 0x0f0,
  0x0c0, 0x0a0, 0x99f, 0x00f, 0x00c, 0xb0c, 0x80c, 0x000,
  0xfff, 0xf00, 0xf99, 0xf90, 0xff0, 0xdd0, 0x8f8, 0x0f0,
  0x0c0, 0x0a0, 0x99f, 0x00f, 0x00c, 0xb0c, 0x80c, 0x000
  };
UWORD colcmap[32] =  /* colorized color map */
  {
  0x000, 0x80c, 0xb0c, 0x00c, 0x00f, 0x99f, 0x0a0, 0x0c0,
  0x0f0, 0x8f8, 0xdd0, 0xff0, 0xf90, 0xf99, 0xf00, 0xfff,
  0x000, 0x80c, 0xb0c, 0x00c, 0x00f, 0x99f, 0x0a0, 0x0c0,
  0x0f0, 0x8f8, 0xdd0, 0xff0, 0xf90, 0xf99, 0xf00, 0xfff
  };

struct NewScreen livescr =    /* screen intialization parameters */
  {
  0, 0, 320, 200,     /* top-left corner, width, height */
  5,          /* depth */
  0, 1,         /* drawing pens */
  0,          /* viewmodes */
  CUSTOMSCREEN,     /* sreen type */
  &textfont,      /* screen font */
  (UBYTE *)"Output Beam",  /* screen title */
  NULL, NULL
  };
struct NewWindow coordwd =    /* window to display coordinates */
  {
  212, 191, 108, 9,    /* top-left corner, width, height */
  -1, -1,        /* use default pens */
  (ULONG)0L,       /* IDCMP flags */
  SMART_REFRESH,      /* window flags */
  NULL, NULL,       /* pointer to first gadget and checkmark */
  NULL,         /* pointer to window title */
  NULL, NULL,       /* pointer to screen and bitmap */
  10, 10, 320, 200,    /* minimum and maximum dimensions */
  CUSTOMSCREEN      /* screen type which window is in */
  };
struct NewWindow livewd =  /* window for LIVE image */
  {
  0, 0, 320, 200,      /* top-left corner, width, height */
  0, 1,         /* drawing pens */
  MOUSEBUTTONS|MOUSEMOVE,  /* IDCMP flags */
  BACKDROP|SMART_REFRESH|  /* window flags */
  ACTIVATE|BORDERLESS|
  REPORTMOUSE,
  NULL, NULL,       /* pointer to first gadget and checkmark */
  NULL,         /* pointer to window title */
  NULL, NULL,       /* pointer to screen and bitmap */
  10, 10, 320, 200,    /* minimum and maximum dimensions */
  CUSTOMSCREEN      /* screen type which window is in */
  };

/* This function initializes the LIVE screen and backdrop window to store
  the grabbed image. */
```

```
short init_screen(void)
  {
  livewd.Width = livescr.Width = WIDTH;
  livewd.Height = livescr.Height = HEIGHT;
  livescr.Depth = DEPTH;
  livescr.DetailPen = (1<<DEPTH) - 1;
  livescr.BlockPen = 0;
  if ((screen = OpenScreen(&livescr)) == NULL) return(1);
  ShowTitle(screen, FALSE); /* LIVEgetFrame() will overwrite title bar */
  livewd.Screen = screen;
  if ((backdrop = (struct Window *)OpenWindow(&livewd)) == NULL)
    return(1);
  else return(0);
  }

/* This function initializes the the LIVE hardware and obtains the address of
  the LIVE board. */
short initlive(void)
  {
  if (GfxBase == NULL)
    GfxBase = OpenLibrary("graphics.library", 0);
  if (GfxBase == NULL)
    { printm("Can't open graphics.library"); return(1); }
  if (IntuitionBase == NULL)
    IntuitionBase = OpenLibrary("intuition.library", 0);
  if (IntuitionBase == NULL)
    { printm("Can't open intuition.library"); return(1); }
  if ((LIVEbase = OpenLibrary("live.library", 0)) == NULL)
    { printm("Can't open live.library"); return(1); }
  if (init_screen())
    { printm("Can't open screen or window for LIVE"); return(1); }
  if ((gotlive = LIVEinitialize(screen,&lively,NULL)) == NULL)
    { printm("Can't initialize LIVE hardware"); return(1); }
  lively.grabmode = SINGLEFRAME;
  lively.colormode = INBW;
  LIVEloadBW(screen, &lively);
  hvalues.videolevels = (struct LIVEvideoLevel *) &vlevels;
  lively.usethese = (struct LIVEhardwareValues *) &hvalues;
  lively.inuse = (struct LIVEhardwareValues *) &hvalues;
  replybackdrop();
  ScreenToBack(screen);
  setcam(camera);          /* choose camera video input */
  setcolormap();
  return(0);
  }

/* This function checks for the presence of a video signal */
short chkvideo(void)
  {
  LIVEintuitionOFF(backdrop);
  if (!LIVEgetFrame(screen, &lively, NULL, NULL))
    {
    printm("No video signal - check possible causes");
    LIVEintuitionON(backdrop);
    return(1);
    }
  LIVEintuitionON(backdrop);
  replybackdrop();
  return(0);
  }

/* This function displays the LIVE video level setting window and waits for
  the desired digitizing level to be chosen. */
short chkvlevels(void)
  {
  if (chkvideo()) return(1);
  ShowTitle(screen, FALSE);       /* do not show screen title */
  vlevels.bw_hi = maxlev;       /* levels to chosen values */
  vlevels.bw_lo = minlev;
  if (colorflg) LIVEloadColorReg(screen,&lively,colcompcmap,NULL);
  else LIVEloadColorReg(screen,&lively,bwcmap,NULL);
  ScreenToFront(screen);      /* move LIVE screen to the front */
  LIVEintuitionOFF(backdrop);
  if (!LIVEaskVideoLevel(screen,&lively,0))
    {
    printm("Video level error - check video signal");
    LIVEintuitionON(backdrop);
    return(1);
    }
  LIVEintuitionON(backdrop);
  replybackdrop();
  ScreenToBack(screen);          /* move screen to the back */
  setcolormap();
  maxlev = vlevels.bw_hi;        /* set new digitizing values */
  minlev = vlevels.bw_lo;
  return(0);
  }
```

```
/* This function sets the color map for the LIVE screen */
void setcolormap(void)
  {
  if (colorflg) LoadRGB4(&screen->ViewPort,colcompcmap,NCOLORS);
  else LoadRGB4(&screen->ViewPort,bwcmap,NCOLORS);
  }

/* This function captures one image and contvers if to raster format */
short getimage(unsigned char *buffer)
  {
  ShowTitle(screen, FALSE);
  LIVEintuitionOFF(backdrop);
  if (!LIVEgetFrame(screen, &lively, NULL, NULL))  /* check for signal */
    {
    LIVEintuitionON(backdrop);
    printm("Video signal interrupted - program halted");
    speakmsg("no video signal");
    return(1);
    }
  LIVEintuitionON(backdrop);
  LIVEdeCyclic(screen);
  /* convert bitmap image to rasterformat */
  bptopix(screen->BitMap.Planes,buffer,(long)FMT,(long)DEPTH);
  ShowTitle(screen, TRUE);
  replybackdrop();
  return(0);
  }

/* This function captures one image and and displays it */
short dispimage(void)
  {
  LIVEintuitionOFF(backdrop);        /* turn off interrupts */
  if (!LIVEgetFrame(screen, &lively, NULL, NULL))  /* check for signal */
    { LIVEintuitionON(backdrop);
     printm("No video signal - check possible causes");
     return(1); }
  LIVEintuitionON(backdrop);
  LIVEdeCyclic(screen);
  replybackdrop();
  return(0);
  }

/* This function closes the LIVE hardware and releases its memory */
void livecleanup(void)
```

```
  {
  if (gotlive) { LIVEexit(&lively); gotlive = NULL; }
  replybackdrop();
  if (backdrop) { CloseWindow(backdrop); backdrop = NULL; }
  if (screen) { CloseScreen(screen); screen = NULL; }
  if (LIVEbase) { CloseLibrary(LIVEbase); LIVEbase = NULL; }
  }

/* This function replys any messages from the LIVE backdrop window */
void replybackdrop(void)
  {
  if (backdrop == NULL) return;
  while (backdropmsg = (struct IntuiMessage *)
           GetMsg(backdrop->UserPort))
    ReplyMsg(&backdropmsg->ExecMessage);
  }

/* This function displays the coordinates of the mouse pointer within the
  backdrop window. */
void dispcoords(void)
  {
  if (gotlive == NULL)
    { printm("LIVE hardware not initialized"); return; }
  coordwd.Screen = screen;
  if ((display = (struct Window *)OpenWindow(&coordwd)) == NULL)
    {
    printm("Could not open coordinate display window");
    return;
    }
  replybackdrop();
  endcoords = 0;
  ScreenToFront(screen);
  ShowTitle(screen, FALSE);
  while (!endcoords)
    if (WaitPort(backdrop->UserPort)) checkcoords();
  CloseWindow(display);
  ShowTitle(screen, TRUE);
  replybackdrop();
  ScreenToBack(screen);
  }

/* This function processes messages from the backdrop window. */
void checkcoords(void)
  {
```

```
  while (backdropmsg = (struct IntuiMessage *)
           GetMsg(backdrop->UserPort))
    {
    msgclass = backdropmsg->Class;
    msgcode = backdropmsg->Code;
    ReplyMsg(&backdropmsg->ExecMessage);
    switch (msgclass)
      {
      case MOUSEMOVE    : showcoords();
               break;
      case MOUSEBUTTONS : if ((msgcode == SELECTDOWN) &&
               (backdrop->MouseX < 10) &&
               (backdrop->MouseY < 10))
                 endcoords = 1;
              break;
      default       : printm("Unknown IDCMP message");
      }
    }
  }

/* This function prints the mouse coordinates in the display window */
void showcoords(void)
  {
  sprintf(tempstr,"X: %3d Y: %3d",backdrop->MouseX,backdrop->MouseY);
  Move(display->RPort,2,7);
  Text(display->RPort,tempstr,strlen(tempstr));
  }
```

D.5 FILTER.C

```
/*********************************************************************
This source file contains the functions required to implement a 3rd order
Butterworth low-pass filter with a given cutoff frequency or a simple,
first order, low-pass filter.

Filename: "FILTER.C", Version: 2.1, Date: May 5, 1992

*********************************************************************/

#include "control.h"

extern float tsamp;        /* in "control.c" */
extern long intensity,xalgerr,yalgerr;
extern short filtflg,xcentd,ycentd;
extern float powmeter,current,voltage;
extern void printm(char *str);            /* in "display.h" */

float digfilter(float input, short fn);
void calcfiltcoeffs(float fc, short type, short fn);
void updatefiltindex(),filterdata(),initfiltvals(),calcfilters(),
    calcperfilts(),permfiltdata(),initpermfilts();

float fcutoff;            /* cutoff frequencies for filters */
float a1[NUMFILTS],a2[NUMFILTS],a3[NUMFILTS],a4[NUMFILTS];
float b1[NUMFILTS],b2[NUMFILTS],b3[NUMFILTS],b4[NUMFILTS];
float xf[NUMFILTS][MAXORDER+1],yf[NUMFILTS][MAXORDER+1];
short n,nm1,nm2,nm3,filtype;

/* This function calculates the filter coefficients for a 3rd order
  Butterworth low-pass filter, a simple, first order low-pass filter,
  or a four-element moving average filter. */
void calcfiltcoeffs(float fc, short type, short fn)
  {
  float wc,wc2,wc3,ts,ts2,ts3;

  if (fc < FCUTMIN) {
    printm("Cutoff frequncy too low - frequency set to minimum value");
    fc = FCUTMIN; }

  if (type == BUTWORTH)
  {
  wc = 2 * PI * fc;
  wc2 = wc * wc;
  wc3 = wc2 * wc;
  ts = tsamp;
  ts2 = ts * ts;
  ts3 = ts2 * ts;

  a1[fn] = wc3;
  a2[fn] = 3 * wc3;
  a3[fn] = 3 * wc3;
  a4[fn] = wc3;

  b1[fn] = 8/ts3 + 8*wc/ts2 + 4*wc2/ts + wc3;
  b2[fn] = -24/ts3 - 8*wc/ts2 + 4*wc2/ts + 3*wc3;
  b3[fn] = 24/ts3 - 8*wc/ts2 - 4*wc2/ts + 3*wc3;
  b4[fn] = -8/ts3 + 8*wc/ts2 - 4*wc2/ts + wc3;

  a1[fn] = a1[fn]/b1[fn];
  a2[fn] = a2[fn]/b1[fn];
  a3[fn] = a3[fn]/b1[fn];
  a4[fn] = a4[fn]/b1[fn];

  b2[fn] = b2[fn]/b1[fn];
  b3[fn] = b3[fn]/b1[fn];
  b4[fn] = b4[fn]/b1[fn];
  b1[fn] = 1.0;
  }
else if (type == FIRSTORD)
  {
  wc = 2 * PI * fc;
  ts = tsamp;

  b1[fn] = 1.0 + 2.0/(wc*ts);
  b2[fn] = 1.0 - 2.0/(wc*ts);
  b3[fn] = 0.0;
  b4[fn] = 0.0;

  a1[fn] = 1/b1[fn];
  a2[fn] = 1/b1[fn];
  a3[fn] = 0.0;
  a4[fn] = 0.0;

  b2[fn] = b2[fn]/b1[fn];
```

```
    b1[fn] = 1.0;
    }
  else if (type == MOVEAVG)
    {
    b1[fn] = 0.0;
    b2[fn] = 0.0;
    b3[fn] = 0.0;
    b4[fn] = 0.0;

    a1[fn] = 0.25;
    a2[fn] = 0.25;
    a3[fn] = 0.25;
    a4[fn] = 0.25;
    }
  n = MAXORDER;   /* initialize filter indeces */
  nm1 = n - 1;
  nm2 = n - 2;
  nm3 = n - 3;
  }

/* This function chooses the type of filter to implement for filtering the
   desired data values, and computes the filter coefficients. */
void calcfilters(void)
  {
  calcfiltcoeffs(fcutoff,filtype,(short)INTNFILT);
  calcfiltcoeffs(fcutoff,filtype,(short)POWFILT);
  calcfiltcoeffs(fcutoff,filtype,(short)CURFILT);
  calcfiltcoeffs(fcutoff,filtype,(short)VOLTFILT);
  calcfiltcoeffs(fcutoff,filtype,(short)XERRFILT);
  calcfiltcoeffs(fcutoff,filtype,(short)YERRFILT);
  }

/* This function chooses the type of filter to implement for filtering the
   data which is always filtered, and computes the filter coefficients. */
void calcpermfilts(void)
  {
  calcfiltcoeffs(fcutoff,filtype,(short)XCENFILT);
  calcfiltcoeffs(fcutoff,filtype,(short)YCENFILT);
  }

/* This function filters the specified data variables. */
void filterdata(void)
  {
  if (filtflg == 1) { initfiltvals(); filtflg = 2; }
  intensity = (long)(digfilter((float)intensity,(short)INTNFILT) + 0.5);
  powmeter = digfilter(powmeter,(short)POWFILT);
  current = digfilter(current,(short)CURFILT);
  voltage = digfilter(voltage,(short)VOLTFILT);
  updatefiltindex();
  xalgerr = (long)(digfilter((float)xalgerr,(short)XERRFILT) + 0.5);
  yalgerr = (long)(digfilter((float)yalgerr,(short)YERRFILT) + 0.5);
  }

/* This function filters the data variableswhich are always filtered. */
void permfiltdata(void)
  {
  if (xf[XCENFILT][nm1] < ERRFLOAT) initpermfilts();
  xcentd = (short)(digfilter((float)xcentd,(short)XCENFILT) + 0.5);
  ycentd = (short)(digfilter((float)ycentd,(short)YCENFILT) + 0.5);
  }

/* This function implements a digital filter defined by the cutoff frequency
   and filter coefficients calculated by calfiltcoeffs(). */
float digfilter(float input, short fn)
  {
  xf[fn][n] = input;
  yf[fn][n] = a1[fn]*xf[fn][n] + a2[fn]*xf[fn][nm1] + a3[fn]*xf[fn][nm2] +
      a4[fn]*xf[fn][nm3] - b2[fn]*yf[fn][nm1] - b3[fn]*yf[fn][nm2] -
      b4[fn]*yf[fn][nm3];
  return(yf[fn][n]);
  }

/* This function increments the filter indeces for digfilter(). */
void updatefiltindex(void)
  {
  n++;    /* n  */
  nm1++;  /* n-1 */
  nm2++;  /* n-2 */
  nm3++;  /* n-3 */

  if (n > MAXORDER) n = 0;
  if (nm1 > MAXORDER) nm1 = 0;
  if (nm2 > MAXORDER) nm2 = 0;
  if (nm3 > MAXORDER) nm3 = 0;
  }

void initpermfilts(void)
  {
```

```
  xf[XCENFILT][nm1] = xcentd;
  xf[XCENFILT][nm2] = xcentd;
  xf[XCENFILT][nm3] = xcentd;
  yf[XCENFILT][nm1] = xcentd;
  yf[XCENFILT][nm2] = xcentd;
  yf[XCENFILT][nm3] = xcentd;
  xf[YCENFILT][nm1] = ycentd;
  xf[YCENFILT][nm2] = ycentd;
  xf[YCENFILT][nm3] = ycentd;
  yf[YCENFILT][nm1] = ycentd;
  yf[YCENFILT][nm2] = ycentd;
  yf[YCENFILT][nm3] = ycentd;
  }

void initfiltvals(void)
  {
  xf[INTNFILT][nm1] = intensity;
  xf[INTNFILT][nm2] = intensity;
  xf[INTNFILT][nm3] = intensity;
  yf[INTNFILT][nm1] = intensity;
  yf[INTNFILT][nm2] = intensity;
  yf[INTNFILT][nm3] = intensity;
  xf[POWFILT][nm1] = powmeter;
  xf[POWFILT][nm2] = powmeter;
  xf[POWFILT][nm3] = powmeter;
  yf[POWFILT][nm1] = powmeter;
  yf[POWFILT][nm2] = powmeter;
  yf[POWFILT][nm3] = powmeter;
  xf[CURFILT][nm1] = current;
  xf[CURFILT][nm2] = current;
  xf[CURFILT][nm3] = current;
  yf[CURFILT][nm1] = current;
  yf[CURFILT][nm2] = current;
  yf[CURFILT][nm3] = current;
  xf[VOLTFILT][nm1] = voltage;
  xf[VOLTFILT][nm2] = voltage;
  xf[VOLTFILT][nm3] = voltage;
  yf[VOLTFILT][nm1] = voltage;
  yf[VOLTFILT][nm2] = voltage;
  yf[VOLTFILT][nm3] = voltage;
  xf[XERRFILT][nm1] = xalgerr;
  xf[XERRFILT][nm2] = xalgerr;
  xf[XERRFILT][nm3] = xalgerr;
  yf[XERRFILT][nm1] = xalgerr;
  yf[XERRFILT][nm2] = xalgerr;
  yf[XERRFILT][nm3] = xalgerr;
  xf[YERRFILT][nm1] = yalgerr;
  xf[YERRFILT][nm2] = yalgerr;
  xf[YERRFILT][nm3] = yalgerr;
  yf[YERRFILT][nm1] = yalgerr;
  yf[YERRFILT][nm2] = yalgerr;
  yf[YERRFILT][nm3] = yalgerr;
  }
```

D.6 P40.C

```
/*****************************************************************
 * This source file contains the funtions necessary to initialize and
 * interface with the Proto-40k data acquisition board.
 * This is a modified version of the original file "p40.c V1.1".
 *
 * Filename: "P40.C", Version: 2.1, Date: May 5, 1992
 *
 *****************************************************************
 * ACDA Proto-40k Rev B. Suggested Interface code.
 *****************************************************************
 *
 * 16 A/D channels (0-10 V) or (-10 - +10 V) input range
 *   Programmable gain input amplifier
 * 2 D/A channels (0-14 V swing)
 * 3 Fully programmable timer channels
 * 16 digital Input bits
 * 16 digiatl Output bits
 *
 *****************************************************************/

#include <exec/types.h>
#include <libraries/expansion.h>
#include <libraries/configregs.h>
#include <libraries/configvars.h>
#include <functions.h>

#define ACDA_CORPORATION   2070L /* Manufacturer number */
#define ACDA_PROTO40K      4L    /* Product number */
#define AVGREAD            20    /* # of values to average for a read */

extern void printm(char *str);   /* defined in "display.c" */

/* Proto-40k Registers ********************************************/
short *p40_data;    /* data area */
short *p40_mux;     /* A/D Multiplexer (MUX) address */
short *p40_gain;    /* for programmable gain option (set 0,1,2,3) */
short *p40_strobe;  /* read or write to start conversion */
short *p40_end;     /* read for end of conversion */
short *p40_da1;     /* write byte for D/A, last val stays latched */
short *p40_da2;     /* write byte for D/A, last val stays latched */
short *p40_dinput;  /* read digital input bits */
short *p40_doutput;   /* write digital output bits */
short *p40_modeselect; /* "random"=0, "sequential"=2 A/D addressing */
short *p40_time0;   /* 8254 timer channel 0 (139 nsec resolution */
short *p40_time1;   /* 8254 timer channel 1 (278 nsec resolution */
short *p40_time2;   /* 8254 timer channel 2 (139 nsec resolution */
short *p40_timecontrol; /* 8254 timer control register */

struct ConfigDev *FindConfigDev();
struct ExpansionBase *ExpansionBase;

/* This function returns the base address of the Proto-40k board. */
short *config(long manu, long prod)
 {
 struct ConfigDev *board;

 ExpansionBase = OpenLibrary(EXPANSIONNAME,0L);
 if (ExpansionBase == NULL) {
   printm("Could not open Expansion Library");
   return ((short *)0); }

 board = 0L;
 if((board = (struct ConfigDev *)FindConfigDev(NULL,manu,prod))==NULL)
   return((short *) 0);
 return((short *) board->cd_BoardAddr);
 }

/* This function initializes the registers of the Proto-40k board. */
short p40_init(void)
 {
 short *base;

 if ((base = config(ACDA_CORPORATION, ACDA_PROTO40K)) == NULL)
   return(0);
 /*  Addressing format for the Proto-40k
  -----------------------------
  Register    =   Auto-config base + Offset */

 p40_time0      = (short *) ((long) base + 0x0000);
 p40_time1      = (short *) ((long) base + 0x0004);
 p40_time2      = (short *) ((long) base + 0x0002);
 p40_timecontrol = (short *) ((long) base + 0x0006);
 p40_data       = (short *) ((long) base + 0x1000);
 p40_mux        = (short *) ((long) base + 0x2000);
 p40_gain       = (short *) ((long) base + 0x3000);
 p40_strobe     = (short *) ((long) base + 0x4000);
 p40_end        = (short *) ((long) base + 0x5000);
 p40_da1        = (short *) ((long) base + 0x7000);
 p40_da2        = (short *) ((long) base + 0x7002);
 p40_dinput     = (short *) ((long) base + 0x8000);
 p40_doutput    = (short *) ((long) base + 0x8002);
 p40_modeselect  = (short *) ((long) base + 0x9000);
 return(1);
 }

/* This function returns the value of the specified A/D channel.
  It replaces the function p40_atod() supplied by the manufacturer. */
float readADchan(short chan)
 {
 short temp,num;
 float sum=0.0;

 *p40_modeselect = 0;
 *p40_gain = 0;
 for (num=0; num<AVGREAD; num++)
   {
   *p40_mux = chan<<2;
   *p40_strobe = 0;
   while (*p40_end >=0);
   temp = (*p40_data & 0x0fff);
   sum += (float)((temp - 2048)/204.8);
   }
 sum = sum / AVGREAD;
 return(sum);
 }

/* This function sets the specified D/A channel to the specified voltage.
  It replaces the function p40_dtoa() supplied by the manufacturer. */
void setDAchan(short chan, float volts)
 {
 UBYTE temp;

 temp = (UBYTE)(volts/15.0*255.0 + 0.5);
 if (temp > 255) temp = 255;
 if (temp < 0) temp = 0;
 switch (chan)
   {
   case 1  : *p40_da1 = temp; break;
```

```
   case 2  : *p40_da2 = temp; break;
   default : printm("Invalid D/A channel");
   }
 }

/*******************************************************************
 * Amiga System Timer Functions:
 *******************************************************************/

#include <devices/timer.h>

struct timeval starttime, curtime;
struct timerequest treq;

short opensystimer(void)
  {
  if (OpenDevice(TIMERNAME, UNIT_MICROHZ, &treq.tr_node, 0L)) {
    printf("Cannot initialize timer device\n");
    return(1); }
  treq.tr_node.io_Message.mn_Node.ln_Type = NT_MESSAGE;
  treq.tr_node.io_Message.mn_Node.ln_Pri = 0;
  treq.tr_node.io_Message.mn_Node.ln_Name = NULL;
  treq.tr_node.io_Message.mn_ReplyPort = NULL;
  return(0);
  }

void closesystimer(void)
  {
  if (&treq.tr_node != NULL) CloseDevice(&treq.tr_node);
  }

void settime(void)
  {
  treq.tr_node.io_Command = TR_GETSYSTIME;
  DoIO(&treq.tr_node);
  starttime.tv_secs = treq.tr_time.tv_secs;
  starttime.tv_micro = treq.tr_time.tv_micro;
  }

double gettime(void)
  {
  treq.tr_node.io_Command = TR_GETSYSTIME;
  DoIO(&treq.tr_node);
  curtime.tv_secs = treq.tr_time.tv_secs;
curtime.tv_micro = treq.tr_time.tv_micro;
return(((long)curtime.tv_secs - (long)starttime.tv_secs) +
    ((long)curtime.tv_micro - (long)starttime.tv_micro)/1000000.0);
}
```

D.7 TALK.C

```
/***************************************************************************
This source file contains the variables and functions required to
initialize the audio handler and perform any audio output.

Filename: "TALK.C", Version: 2.1, Date: May 5, 1992

***************************************************************************/

#include <exec/types.h>
#include <functions.h>
#include <exec/exec.h>
#include <devices/narrator.h>
#include <libraries/translator.h>

#define PSTRINGSIZE 512       /* phonemes are longer than english */

extern void printm(char *str);         /* defined in "display.c" */

/* Which audio channels to use */
UBYTE audio_chan[4] = {3, 5, 10, 12};

struct Library *TranslatorBase = NULL;
struct MsgPort *talk_port = NULL;     /* pointers to talk structures */
struct narrator_rb *voice_io = NULL;

short openerror = -1,    /* temporary variables to hold return values */
  rtncode;
short speakmsg(),
  initaudio(UWORD rate,UWORD pitch,UWORD mode,UWORD sex,UWORD sampfq);
char PhonBuffer[PSTRINGSIZE];    /* buffer to hold translated speech */
void audiocleanup();

/* This function initializes the audio port based on the following values:
   sex = 0(male) or 1(female); pitch = 65 to 320; rate = 40 to 400;
   mode = 0(natural) or 1(monotone); sampfq = 5000 to 40000 (Hz);   */
short initaudio(UWORD rate,UWORD pitch,UWORD mode,UWORD sex,UWORD
                         sampfq)
 {
 /* Open the libraries that the program uses directly */
 if (TranslatorBase == NULL)
```

```
  TranslatorBase = (struct Library *)OpenLibrary("translator.library", 33L);
 if (TranslatorBase == NULL)
  { printm("Can't open the translator library"); return(1); }

 if (talk_port == NULL) talk_port = CreatePort(0,0);
 if (talk_port == NULL)
  { printm("Cannot open write port"); return(1); }
 if (voice_io == NULL)
  voice_io = (struct narrator_rb *)CreateExtIO(talk_port,sizeof(struct narrator_rb));
 if (voice_io == NULL)
  { printm("Cannot open narrator port"); return(1); }

 /* Set up the voice channel information */
 voice_io->ch_masks = audio_chan;
 voice_io->nm_masks = sizeof(audio_chan);
 voice_io->message.io_Data = (APTR)PhonBuffer;
 voice_io->message.io_Length = strlen(PhonBuffer);
 voice_io->mouths = 0;
 voice_io->message.io_Command = CMD_WRITE;
 voice_io->pitch = pitch;
 voice_io->sex = sex;
 voice_io->rate = rate;
 voice_io->mode = mode;
 voice_io->sampfreq = sampfq;

 /* Open the narrator device */
 if ((openerror = OpenDevice("narrator.device",0,
     (struct IORequest *)&voice_io->message, 0)) != 0)
  { printm("Can't open the narrator device"); return(1); }
 return(0);
 }

/* This function speaks out the specified message. */
short speakmsg(char *str)
 {
 rtncode = Translate(str,strlen(str),PhonBuffer,PSTRINGSIZE);
 if (rtncode)
  { printm("Cannot translate message"); return(1); }
 voice_io->message.io_Length = strlen(PhonBuffer);
 openerror = DoIO((struct IORequest *)&voice_io->message);
 if (openerror)
  { printm("Cannot perform audio i/o"); return(1); }
 else return(0);
 }
```

```
/* This routine closes the audio handler and associated device. */
void audiocleanup(void)
 {
 if (voice_io)
  {
  CloseDevice((struct IORequest *)&voice_io->message);
  DeleteExtIO((struct IORequest *)&voice_io->message);
  voice_io = NULL;
  }
 if (talk_port) { DeletePort(talk_port); talk_port = NULL; }
 if (TranslatorBase)
  { CloseLibrary(TranslatorBase); TranslatorBase = NULL; }
 }
```

D.8 BPTOPIX.ASM

```
; This function converts an image from bitplane (raw) format to raster
; (pixel) format in order to quickly obtain the intensity of any pixel.

; Filename: "BPTOPIX.ASM", Version: 2.1, Date: May 5, 1992

; Call format: bp2pix(bitmap,raster,format,depth)

; Arguments: bitmap - pointer to array containing addresses of bitplanes
;         raster - 2-dimensional array of unsigned char (HEIGHT * WIDTH)
;         format - image format:          1 - (320 x 200), 2 - (640 x 200),
;                                         3 - (320 x 400), 4 - (640 x 400)
;         depth  - number of bitplanes

bpsizel         EQU     8000    ;size of bitplane for format 1
bpsizem         EQU     16000   ;size of bitplane for format 2 and 3
bpsizeh         EQU     32000   ;size of bitplane for format 4

                CSEG
                public _bptopix
_bptopix:
                MOVEA.L 4(SP),A0 ;move ptr to bitmap address array to A0
                MOVEA.L 8(SP),A1 ;move address of raster to A1
                MOVE.L  12(SP),D0;move format to D0
                MOVE.L  16(SP),D1 ;move depth to D1
                MOVEM.L A2-A6/D2-D7,-(SP) ;save other registers on stack
                MOVEA.L A1,A5
                CMP.W   #1,D0     ;if (format == 1)
                BNE     FMTM1
                MOVE.L  #bpsizel,D6 ;  D6 <- bpsizel
                BRA     CHKBP
FMTM1:          CMP.W   #2,D0     ;else if (format == 2)
                BNE     FMTM2
                MOVE.L  #bpsizem,D6 ;  D6 <- bpsizem
                BRA     CHKBP
FMTM2:          CMP.W   #3,D0     ;else if (format == 3)
                BNE     FMTH
                MOVE.L  #bpsizem,D6 ;  D6 <- bpsizem
                BRA     CHKBP
FMTH:           MOVE.L  #bpsizeh,D6 ;else D6 <- bpsizeh
CHKBP:          CMP.W   #5,D1
```

```
                BEQ     BPLANE5;check depth and
                CMP.W   #4,D1   ;branch to corresponding section
                BEQ     BPLANE4
                CMP.W   #3,D1
                BEQ     BPLANE3
                CMP.W   #2,D1
                BEQ     BPLANE2
                CMP.W   #1,D1
                BEQ     BPLANE1
                BRA     RETURN
BPLANE5:        NOP             ;bitplane #5 not used (only 16 levels)
BPLANE4:        MOVEA.L 4(A0),A1 ;move ptr to address of bitplane #2 to A1
                MOVEA.L 8(A0),A2 ;move ptr to address of bitplane #3 to A2
                MOVEA.L 12(A0),A3           ;move ptr to address of bitplane #4 to
                A3
                MOVEA.L (A0),A4 ;move ptr to address of bitplane #1 to A0
                MOVEA.L A4,A0
                DIVU    #4,D6   ;set up D6 is loop counter
                SUBQ.L  #1,D6   ;moves are 32 bits not 8 (divide by 4)
LOOP41:         MOVEQ   #31,D5  ;set up d5 to count bits
                MOVE.L  (A0)+,D0 ;move 32 bits from each bitplane into
                MOVE.L  (A1)+,D1 ;data registers
                MOVE.L  (A2)+,D2
                MOVE.L  (A3)+,D3
LOOP42:         CLR.L   D4
                BTST    D5,D3   ;test bit #D5 of the data from each
                BEQ     ZERO43  ;bitplane is a 1
                ORI.B   #1,D4   ;if it is a 1, set the LSB of D4 to 1 and
                LSL.B   #1,D4   ;shift it to the left
ZERO43:         BTST    D5,D2   ;thus D4 assumes the value of intensity
                BEQ     ZERO42  ;e.g. D3 = 10100010101... (bit 3 of D4)
                ORI.B   #1,D4   ;    D2 = 11010001011... (bit 2 of D4)
ZERO42:         LSL.B   #1,D4   ;    D1 = 01010100010... (bit 1 of D4)
                BTST    D5,D1   ;    D0 = 10010100101... (bit 0 of D4)
                BEQ     ZERO41  ;    for the first loop D4 = 1101
                ORI.B   #1,D4   ;    for the second loop D4 = 0110, etc.
ZERO41:         LSL.B   #1,D4
                BTST    D5,D0
                BEQ     ZERO40
                ORI.B   #1,D4
ZERO40:         MOVE.B  D4,(A5)+ ;write D4 to raster
                TST.B   D5
                DBEQ    D5,LOOP42           ;repeat 32 times
                TST.L   D6
```

```
              DBEQ     D6,LOOP41        ;repeat until entire image is read
              BRA      RETURN
BPLANE3:      NOP               ;no images with 3 bitplanes
              BRA      RETURN
BPLANE2:      NOP               ;no images with 2 bitplanes
              BRA      RETURN
BPLANE1:      MOVEA.L (A0),A4   ;move ptr to address of bitplane to A0
              MOVEA.L A4,A0
              DIVU     #4,D6    ;set up D6 as loop counter (read 32 bits)
              SUBQ.L #1,D6
LOOP11:       MOVEQ  #31,D5     ;set up D5 as bit counter
              MOVE.L (A0)+,D0   ;read 32 bits of data
LOOP12:       CLR.L    D4
              BTST     D5,D0    ;set bit #24, #16, #8, and #0 of D4 to
              BEQ      ZERO13   ;the first 4 bits of D0
              ORI.L    #$1000000,D4
ZERO13:       SUBQ.B #1,D5      ;e.g.
              BTST     D5,D0    ;if D0 = 1111010001... then
              BEQ      ZERO12 ;  D4 = 0000000100000001
                                      0000000100000001
              ORI.L    #$10000,D4
ZERO12:       SUBQ.B #1,D5
              BTST     D5,D0
              BEQ      ZERO11
              ORI.L    #$100,D4
ZERO11:       SUBQ.B #1,D5
              BTST     D5,D0
              BEQ      ZERO10
              ORI.L    #1,D4
ZERO10:       MOVE.L D4,(A5)+   ;write D4 to raster
              TST.B    D5
              DBEQ     D5,LOOP12        ;repeat 8 times
              TST.L    D6
              DBEQ     D6,LOOP11        ;repeat until entire image is read
RETURN:       MOVEM.L (SP)+,A2-A6/D2-D7 ;restore registers
              RTS
              END
```

D.9 QINTCENT.ASM

```
;*******************************************************************
;
; This file contains the routines quadintcent(), quadintwcent(),
; quadintucent() and quadintens() for computing the quadrant intensities
; and the desired moments of intertia.
; Filename: "QINTCENT.ASM", Version: 2.1, Date: May 5, 1992
;*******************************************************************
;
WIDTH          EQU          320     ;width of image

               DSEG
public _xmin,_xmax,_ymin,_ymax,_xcentd,_ycentd
               public _quad1,_quad2,_quad3,_quad4
               public _image
               public _momentx,_momenty,_unwmomx,_unwmomy,_beamarea

               CSEG
               public _quadintcent,_quadintwcent,_quadintucent,_quadintens

;*******************************************************************
;
; This function calculates the total intensity of each quadrant of an image,
; and the weighted and unweighted X and Y moments needed to compute the
; centroids of the image.
;*******************************************************************
;
; Call format: quadintcent()
; Register usage:  D0 - temporary register for intermediate calculations
;                        D1 - outer loop counter
;                        D2 - inner loop counter
;                        D3 - maximum value of outer loop counter
;                        D4 - maximum value of inner loop counter
;                        D5 - used to store pixel intensity value
;                        D6 - used to store and compute weighted X-moment
;                        D7 - used to store and compute weighted Y-moment
;                        A0 - address of image pixel to be accessed
;                        A1 - starting address of image raster
;                        A2 - used to store and compute quadrant intensity
;                        A3 - width of image (in pixels)
;                        A4 - used to store and compute unweighted X-moment
;                        A5 - used to store and compute unweighted Y-moment
;                        A6 - used to store and compute beam area

_quadintcent:
```

```
            MOVEM.L A0-A6/D0-D7,-(SP) ;save registers on stack
            CLR.L    D0          ;clear data registers
            CLR.L    D1
            CLR.L    D2
            CLR.L    D3
            CLR.L    D4
            CLR.L    D5
            CLR.L    D6
            CLR.L    D7
            MOVEA.L _image,A1          ;set A1 to address of image
            MOVEA.L #0,A2     ;use A2 to store quadrant intensity
            MOVEA.W #WIDTH,A3         ;use A3 to store width of image
            MOVEA.L #0,A4     ;clear A4 (used to store unimomx)
            MOVEA.L #0,A5     ;clear A5 (used to store unimomy)
            MOVEA.L #0,A6     ;clear A6 (used to store beamarea)
            MOVE.W  _ycentd,D3         ;set maximum limits for quad2 to
            D3,D4
            MOVE.W  _xcentd,D4
            MOVE.W  _ymin,D1 ;initialize outer loop counter D1
            MOVE.W  A3,D0     ;move width of image to D0
            MULS.W  D1,D0     ;compute starting row address quad2
            MOVEA.L A1,A0     ;  i.e. *(image+ymin*width)
            ADDA.L  D0,A0     ;move starting row address to A0
QUAD21C:    MOVE.W  _xmin,D2 ;initialize inner loop counter D2
QUAD22C:    MOVEQ.L #15,D5    ;adjust intensity for complemented colours
            SUB.B    (A0,D2),D5         ; i.e. pixel = maxintensity - pixel
            ADDA.L  D5,A2     ;add pixel value (D5) to total intensity
            MOVE.W  D2,D0     ;compute X-moment contribution of pixel
            MULS.W  D5,D0
            ADD.L    D0,D6     ;add Xpixel*Xposition to total X-moment D6
            MOVE.W  D1,D0     ;compute Y-moment contribution of pixel
            MULS.W  D5,D0
            ADD.L    D0,D7     ;add Ypixel*Yposition to total Y-moment D7
            CMPI.W  #0,D5     ;if (pixel value == 0)
            BEQ      NOVALC2            goto NOVALC2
            ADDA.L  D2,A4     ;add X-position of pixel to unwmomx A4
            ADDA.L  D1,A5     ;add Y-position of pixel to unwmomy A5
            ADDA.L  #1,A6     ;increment beam area by 1
NOVALC2:    ADDQ.W  #1,D2     ;increment inner loop counter D2
            CMP.W    D2,D4     ;if (D2 > xcent) break out of inner loop
            BGT      QUAD22C
            ADDA.L  A3,A0     ;move to next row of image raster
            ADDQ.W  #1,D1     ;increment outer loop counter D1
            CMP.W    D1,D3     ;if (D1 > ycent) break out of outer loop
```

```
            BGT      QUAD21C
            MOVE.L  A2,_quad2;store quadrant intensity
            MOVEA.L #0,A2     ;use A2 to store quadrant intensity
            MOVE.W  _ycentd,D3         ;set maximum limits for quad1 to
            D3,D4
            MOVE.W  _xmax,D4
            MOVE.W  _ymin,D1 ;initialize outer loop counter D1
            MOVE.W  A3,D0     ;move width of image to D0
            MULS.W  D1,D0     ;compute starting row address quad1
            MOVEA.L A1,A0     ;  i.e. *(image+ymin*width)
            ADDA.L  D0,A0     ;move starting row address to A0
QUAD11C:    MOVE.W  _xcentd,D2         ;initialize inner loop counter D2
QUAD12C:    MOVEQ.L #15,D5    ;adjust intensity for complemented colours
            SUB.B    (A0,D2),D5         ; i.e. pixel = maxintensity - pixel
            ADDA.L  D5,A2     ;add pixel value (D5) to total intensity
            MOVE.W  D2,D0     ;compute X-moment contribution of pixel
            MULS.W  D5,D0
            ADD.L    D0,D6     ;add Xpixel*Xposition to total X-moment D6
            MOVE.W  D1,D0     ;compute Y-moment contribution of pixel
            MULS.W  D5,D0
            ADD.L    D0,D7     ;add Ypixel*Yposition to total Y-moment D7
            CMPI.W  #0,D5     ;if (pixel value == 0)
            BEQ      NOVALC1            goto NOVALC1
            ADDA.L  D2,A4     ;add X-position of pixel to unwmomx A4
            ADDA.L  D1,A5     ;add Y-position of pixel to unwmomy A5
            ADDA.L  #1,A6     ;increment beam area by 1
NOVALC1:    ADDQ.W  #1,D2     ;increment inner loop counter D2
            CMP.W    D2,D4     ;if (D2 > xmax) break out of inner loop
            BGE      QUAD12C
            ADDA.L  A3,A0     ;move to next row of image raster
            ADDQ.W  #1,D1     ;increment outer loop counter D1
            CMP.W    D1,D3     ;if (D1 > ycent) break out of outer loop
            BGT      QUAD11C
            MOVE.L  A2,_quad1;store quadrant intensity
            MOVEA.L #0,A2     ;use A2 to store quadrant intensity
            MOVE.W  _ymax,D3           ;set maximum limits for quad4 to
            D3,D4
            MOVE.W  _xmax,D4
            MOVE.W  _ycentd,D1         ;initialize outer loop counter D1
            MOVE.W  A3,D0     ;move width of image to D0
            MULS.W  D1,D0     ;compute starting row address quad4
            MOVEA.L A1,A0     ;  i.e. *(image+ycent*width)
            ADDA.L  D0,A0     ;move starting row address to A0
QUAD41C:    MOVE.W  _xcentd,D2         ;initialize inner loop counter D2
```

```
QUAD42C:        MOVEQ.L #15,D5    ;adjust intensity for complemented colours
                SUB.B   (A0,D2),D5          ; i.e. pixel = maxintensity - pixel
                ADDA.L  D5,A2     ;add pixel value (D5) to total intensity
                MOVE.W  D2,D0     ;compute X-moment contribution of pixel
                MULS.W  D5,D0
                ADD.L   D0,D6     ;add Xpixel*Xposition to total X-moment D6
                MOVE.W  D1,D0     ;compute Y-moment contribution of pixel
                MULS.W  D5,D0
                ADD.L   D0,D7     ;add Ypixel*Yposition to total Y-moment D7
                CMPI.W  #0,D5     ;if (pixel value == 0)
                BEQ     NOVALC4             goto NOVALC4
                ADDA.L  D2,A4     ;add X-position of pixel to unwmomx A4
                ADDA.L  D1,A5     ;add Y-position of pixel to unwmomy A5
                ADDA.L  #1,A6     ;increment beam area by 1
NOVALC4:        ADDQ.W  #1,D2     ;increment inner loop counter D2
                CMP.W   D2,D4     ;if (D2 > xmax) break out of inner loop
                BGE     QUAD42C
                ADDA.L  A3,A0     ;move to next row of image raster
                ADDQ.W  #1,D1     ;increment outer loop counter D1
                CMP.W   D1,D3     ;if (D1 > ymax) break out of outer loop
                BGE     QUAD41C
                MOVE.L  A2,_quad4;store quadrant intensity
                MOVEA.L #0,A2     ;use A2 to store quadrant intensity
                MOVE.W  _ymax,D3            ;set maximum limits for quad3 to
                D3,D4
                MOVE.W  _xcentd,D4
                MOVE.W  _ycentd,D1          ;initialize outer loop counter D1
                MOVE.W  A3,D0     ;move width of image to D0
                MULS.W  D1,D0     ;compute starting row address quad3
                MOVEA.L A1,A0     ; i.e. *(image+ycent*width)
                ADDA.L  D0,A0     ;move starting row address to A0
QUAD31C:        MOVE.W  _xmin,D2 ;initialize inner loop counter D2
QUAD32C:        MOVEQ.L #15,D5    ;adjust intensity for complemented colours
                SUB.B   (A0,D2),D5          ; i.e. pixel = maxintensity - pixel
                ADDA.L  D5,A2     ;add pixel value (D5) to total intensity
                MOVE.W  D2,D0     ;compute X-moment contribution of pixel
                MULS.W  D5,D0
                ADD.L   D0,D6     ;add Xpixel*Xposition to total X-moment D6
                MOVE.W  D1,D0     ;compute Y-moment contribution of pixel
                MULS.W  D5,D0
                ADD.L   D0,D7     ;add Ypixel*Yposition to total Y-moment D7
                CMPI.W  #0,D5     ;if (pixel value == 0)
                BEQ     NOVALC3             goto NOVALC3
                ADDA.L  D2,A4     ;add X-position of pixel to unwmomx A4
                ADDA.L  D1,A5     ;add Y-position of pixel to unwmomy A5
                ADDA.L  #1,A6     ;increment beam area by 1
NOVALC3:        ADDQ.W  #1,D2     ;increment inner loop counter D2
                CMP.L   D2,D4     ;if (D2 > xcent) break out of inner loop
                BGT     QUAD32C
                ADDA.L  A3,A0     ;move to next row of image raster
                ADDQ.W  #1,D1     ;increment outer loop counter D1
                CMP.L   D1,D3     ;if (D1 > ymax) break out of outer loop
                BGE     QUAD31C
                MOVE.L  A2,_quad3;store quadrant intensity
                MOVE.L  D6,_momentx         ;store weighted X-moment
                MOVE.L  D7,_momenty         ;store weighted Y-moment
                MOVE.L  A4,_unwmomx         ;store unweighted X-moment
                MOVE.L  A5,_unwmomy         ;store unweighted Y-moment
                MOVE.L  A6,_beamarea  ;store beam area
                MOVEM.L (SP)+,A0-A6/D0-D7 ;restore registers
                RTS

;*********************************************************************
;
; This function calculates the total intensity of each quadrant of an image,
; and the weighted X and Y moments needed to compute the centroid of the
; image.
;*********************************************************************
;
; Call format: quadintwcent()
; Register usage:  D0 - temporary register for intermediate calculations
;                  D1 - outer loop counter
;                  D2 - inner loop counter
;                  D3 - maximum value of outer loop counter
;                  D4 - maximum value of inner loop counter
;                  D5 - used to store pixel intensity value
;                  D6 - used to store and compute weighted X-moment
;                  D7 - used to store and compute weighted Y-moment
;                  A0 - address of image pixel to be accessed
;                  A1 - starting address of image raster
;                  A2 - used to store and compute quadrant intensity
;                  A3 - width of image (in pixels)

_quadintwcent:
                MOVEM.L A0-A6/D0-D7,-(SP) ;save registers on stack
                CLR.L   D0          ;clear data registers
                CLR.L   D1
                CLR.L   D2
                CLR.L   D3
                CLR.L   D4
```

```
            CLR.L     D5
            CLR.L     D6
            CLR.L     D7
            MOVEA.L _image,A1           ;set A1 to address of image
            MOVEA.W #WIDTH,A3           ;use A3 to store width of image
            MOVEA.L #0,A2      ;use A2 to store quadrant intensity
            MOVE.W  _ycentd,D3          ;set maximum limits for quad2 to
            D3,D4
            MOVE.W  _xcentd,D4
            MOVE.W  _ymin,D1 ;initialize outer loop counter D1
            MOVE.W  A3,D0      ;move width of image to D0
            MULS.W  D1,D0      ;compute starting row address quad2
            MOVEA.L A1,A0      ;  i.e. *(image+ymin*width)
            ADDA.L  D0,A0      ;move starting row address to A0
QUAD21W:    MOVE.W  _xmin,D2 ;initialize inner loop counter D2
QUAD22W:    MOVEQ.L #15,D5     ;adjust intensity for complemented colours
            SUB.B     (A0,D2),D5           ; i.e. pixel = maxintensity - pixel
            ADDA.L  D5,A2      ;add pixel value (D5) to total intensity
            MOVE.W  D2,D0      ;compute X-moment contribution of pixel
            MULS.W  D5,D0
            ADD.L     D0,D6    ;add Xpixel*Xposition to total X-moment D6
            MOVE.W  D1,D0      ;compute Y-moment contribution of pixel
            MULS.W  D5,D0
            ADD.L     D0,D7    ;add Ypixel*Yposition to total Y-moment D7
            ADDQ.W  #1,D2      ;increment inner loop counter D2
            CMP.W   D2,D4      ;if (D2 > xcent) break out of inner loop
            BGT                QUAD22W
            ADDA.L  A3,A0      ;move to next row of image raster
            ADDQ.W  #1,D1      ;increment outer loop counter D1
            CMP.W   D1,D3      ;if (D1 > ycent) break out of outer loop
            BGT                QUAD21W
            MOVE.L  A2,_quad2;store quadrant intensity
            MOVEA.L #0,A2      ;use A2 to store quadrant intensity
            MOVE.W  _ycentd,D3          ;set maximum limits for quad1 to
            D3,D4
            MOVE.W  _xmax,D4
            MOVE.W  _ymin,D1 ;initialize outer loop counter D1
            MOVE.W  A3,D0      ;move width of image to D0
            MULS.W  D1,D0      ;compute starting row address quad1
            MOVEA.L A1,A0      ;  i.e. *(image+ymin*width)
            ADDA.L  D0,A0      ;move starting row address to A0
QUAD11W:    MOVE.W  _xcentd,D2          ;initialize inner loop counter D2
QUAD12W:    MOVEQ.L #15,D5     ;adjust intensity for complemented colours
            SUB.B     (A0,D2),D5           ; i.e. pixel = maxintensity - pixel
            ADDA.L  D5,A2      ;add pixel value (D5) to total intensity
            MOVE.W  D2,D0      ;compute X-moment contribution of pixel
            MULS.W  D5,D0
            ADD.L     D0,D6    ;add Xpixel*Xposition to total X-moment D6
            MOVE.W  D1,D0      ;compute Y-moment contribution of pixel
            MULS.W  D5,D0
            ADD.L     D0,D7    ;add Ypixel*Yposition to total Y-moment D7
            ADDQ.W  #1,D2      ;increment inner loop counter D2
            CMP.W   D2,D4      ;if (D2 > xmax) break out of inner loop
            BGE       QUAD12W
            ADDA.L  A3,A0      ;move to next row of image raster
            ADDQ.W  #1,D1      ;increment outer loop counter D1
            CMP.W   D1,D3      ;if (D1 > ycent) break out of outer loop
            BGT       QUAD11W
            MOVE.L  A2,_quad1;store quadrant intensity
            MOVEA.L #0,A2      ;use A2 to store quadrant intensity
            MOVE.W  _ymax,D3            ;set maximum limits for quad4 to
            D3,D4
            MOVE.W  _xmax,D4
            MOVE.W  _ycentd,D1          ;initialize outer loop counter D1
            MOVE.W  A3,D0      ;move width of image to D0
            MULS.W  D1,D0      ;compute starting row address quad4
            MOVEA.L A1,A0      ;  i.e. *(image+ycent*width)
            ADDA.L  D0,A0      ;move starting row address to A0
QUAD41W:    MOVE.W  _xcentd,D2          ;initialize inner loop counter D2
QUAD42W:    MOVEQ.L #15,D5     ;adjust intensity for complemented colours
            SUB.B     (A0,D2),D5           ; i.e. pixel = maxintensity - pixel
            ADDA.L  D5,A2      ;add pixel value (D5) to total intensity
            MOVE.W  D2,D0      ;compute X-moment contribution of pixel
            MULS.W  D5,D0
            ADD.L     D0,D6    ;add Xpixel*Xposition to total X-moment D6
            MOVE.W  D1,D0      ;compute Y-moment contribution of pixel
            MULS.W  D5,D0
            ADD.L     D0,D7    ;add Ypixel*Yposition to total Y-moment D7
            ADDQ.W  #1,D2      ;increment inner loop counter D2
            CMP.W   D2,D4      ;if (D2 > xmax) break out of inner loop
            BGE       QUAD42W
            ADDA.L  A3,A0      ;move to next row of image raster
            ADDQ.W  #1,D1      ;increment outer loop counter D1
            CMP.W   D1,D3      ;if (D1 > ymax) break out of outer loop
            BGE       QUAD41W
            MOVE.L  A2,_quad4,store quadrant intensity
            MOVEA.L #0,A2      ;use A2 to store quadrant intensity
            MOVE.W  _ymax,D3            ;set maximum limits for quad3 to
```

```
                D3,D4
                MOVE.W  _xcentd,D4
                MOVE.W  _ycentd,D1          ;initialize outer loop counter D1
                MOVE.W  A3,D0     ;move width of image to D0
                MULS.W  D1,D0     ;compute starting row address quad3
                MOVEA.L A1,A0     ;  i.e. *(image+ycent*width)
                ADDA.L  D0,A0     ;move starting row address to A0
QUAD31W:        MOVE.W  _xmin,D2 ;initialize inner loop counter D2
QUAD32W:        MOVEQ.L #15,D5    ;adjust intensity for complemented colours
                SUB.B   (A0,D2),D5          ; i.e. pixel = maxintensity - pixel
                ADDA.L  D5,A2     ;add pixel value (D5) to total intensity
                MOVE.W  D2,D0     ;compute X-moment contribution of pixel
                MULS.W  D5,D0
                ADD.L   D0,D6     ;add Xpixel*Xposition to total X-moment D6
                MOVE.W  D1,D0     ;compute Y-moment contribution of pixel
                MULS.W  D5,D0
                ADD.L   D0,D7     ;add Ypixel*Yposition to total Y-moment D7
                ADDQ.W  #1,D2     ;increment inner loop counter D2
                CMP.L   D2,D4     ;if (D2 > xcent) break out of inner loop
                BGT               QUAD32W
                ADDA.L  A3,A0     ;move to next row of image raster
                ADDQ.W  #1,D1     ;increment outer loop counter D1
                CMP.L   D1,D3     ;if (D1 > ymax) break out of outer loop
                BGE               QUAD31W
                MOVE.L  A2,_quad3;store quadrant intensity
                MOVE.L  D6,_momentx         ;store weighted X-moment
                MOVE.L  D7,_momenty         ;store weighted Y-moment
MOVEM.L (SP)+,A0-A6/D0-D7 ;restore registers
                RTS

; ************************************************************************
; This function calculates the total intensity of each quadrant of an image,
; and the unweighted X and Y moments needed to compute the centroid of the
; image.
; ************************************************************************
; Call format: quadintucent()
; Register usage:  D0 - temporary register for intermediate calculations
;                       D1 - outer loop counter
;                       D2 - inner loop counter
;                       D3 - maximum value of outer loop counter
;                       D4 - maximum value of inner loop counter
;                       D5 - used to store pixel intensity value
;                       A0 - address of image pixel to be accessed
;                       A1 - starting address of image raster
;                       A2 - used to store and compute quadrant intensity
;                       A3 - width of image (in pixels)
;                       A4 - used to store and compute unweighted X-moment
;                       A5 - used to store and compute unweighted Y-moment
;                       A6 - used to store and compute beam area

_quadintucent:
                MOVEM.L A0-A6/D0-D7,-(SP) ;save registers on stack
                CLR.L   D0          ;clear data registers
                CLR.L   D1
                CLR.L   D2
                CLR.L   D3
                CLR.L   D4
                CLR.L   D5
                CLR.L   D6
                CLR.L   D7
                MOVEA.L _image,A1           ;set A1 to address of image
                MOVEA.L #0,A2     ;use A2 to store quadrant intensity
                MOVEA.W #WIDTH,A3           ;use A3 to store width of image
                MOVEA.L #0,A4     ;clear A4 (used to store unimomx)
                MOVEA.L #0,A5     ;clear A5 (used to store unimomy)
                MOVEA.L #0,A6     ;clear A6 (used to store beamarea)
                MOVE.W  _ycentd,D3          ;set maximum limits for quad2 to
                D3,D4
                MOVE.W  _xcentd,D4
                MOVE.W  _ymin,D1 ;initialize outer loop counter D1
                MOVE.W  A3,D0     ;move width of image to D0
                MULS.W  D1,D0     ;compute starting row address quad2
                MOVEA.L A1,A0     ;  i.e. *(image+ymin*width)
                ADDA.L  D0,A0     ;move starting row address to A0
QUAD21U:        MOVE.W  _xmin,D2 ;initialize inner loop counter D2
QUAD22U:        MOVEQ.L #15,D5    ;adjust intensity for complemented colours
                SUB.B   (A0,D2),D5          ; i.e. pixel = maxintensity - pixel
                ADDA.L  D5,A2     ;add pixel value (D5) to total intensity A2
                CMPI.W  #0,D5     ;if (pixel value == 0)
                BEQ     NOVAL2U             goto NOVAL2U
                ADDA.L  D2,A4     ;add X-position of pixel to unwmomx A4
                ADDA.L  D1,A5     ;add Y-position of pixel to unwmomy A5
                ADDA.L  #1,A6     ;increment beam area by 1
NOVAL2U:        ADDQ.W  #1,D2     ;increment inner loop counter D2
                CMP.W   D2,D4     ;if (D2 > xcent) break out of inner loop
                BGT     QUAD22U
                ADDA.L  A3,A0     ;move to next row of image raster
                ADDQ.W  #1,D1     ;increment outer loop counter D1
```

```
            CMP.W   D1,D3     ;if (D1 > ycent) break out of outer loop
            BGT     QUAD21U
            MOVE.L  A2,_quad2;store quadrant intensity
            MOVEA.L #0,A2     ;use A2 to store quadrant intensity
            MOVE.W  _ycentd,D3          ;set maximum limits for quad1 to
D3,D4
            MOVE.W  _xmax,D4
            MOVE.W  _ymin,D1 ;initialize outer loop counter D1
            MOVE.W  A3,D0     ;move width of image to D0
            MULS.W  D1,D0     ;compute starting row address quad1
            MOVEA.L A1,A0     ;  i.e. *(image+ymin*width)
            ADDA.L  D0,A0     ;move starting row address to A0
QUAD11U:    MOVE.W  _xcentd,D2          ;initialize inner loop counter D2
QUAD12U:    MOVEQ.L #15,D5    ;adjust intensity for complemented colours
            SUB.B   (A0,D2),D5          ;  i.e. pixel = maxintensity - pixel
            ADDA.L  D5,A2     ;add pixel value (D5) to total intensity A2
            CMPI.W  #0,D5     ;if (pixel value == 0)
            BEQ     NOVAL1U             goto NOVAL1U
            ADDA.L  D2,A4     ;add X-position of pixel to unwmomx A4
            ADDA.L  D1,A5     ;add Y-position of pixel to unwmomy A5
            ADDA.L  #1,A6     ;increment beam area by 1
NOVAL1U:    ADDQ.W  #1,D2     ;increment inner loop counter D2
            CMP.W   D2,D4     ;if (D2 > xmax) break out of inner loop
            BGE     QUAD12U
            ADDA.L  A3,A0     ;move to next row of image raster
            ADDQ.W  #1,D1     ;increment outer loop counter D1
            CMP.W   D1,D3     ;if (D1 > ycent) break out of outer loop
            BGT     QUAD11U
            MOVE.L  A2,_quad1 ;store quadrant intensity
            MOVEA.L #0,A2     ;use A2 to store quadrant intensity
            MOVE.W  _ymax,D3            ;set maximum limits for quad4 to
D3,D4
            MOVE.W  _xmax,D4
            MOVE.W  _ycentd,D1          ;initialize outer loop counter D1
            MOVE.W  A3,D0     ;move width of image to D0
            MULS.W  D1,D0     ;compute starting row address quad4
            MOVEA.L A1,A0     ;  i.e. *(image+ycent*width)
            ADDA.L  D0,A0     ;move starting row address to A0
QUAD41U:    MOVE.W  _xcentd,D2          ;initialize inner loop counter D2
QUAD42U:    MOVEQ.L #15,D5    ;adjust intensity for complemented colours
            SUB.B   (A0,D2),D5          ;  i.e. pixel = maxintensity - pixel
            ADDA.L  D5,A2     ;add pixel value (D5) to total intensity A2
            CMPI.W  #0,D5     ;if (pixel value == 0)
            BEQ     NOVAL4U             goto NOVAL4U
```

```
            ADDA.L  D2,A4     ;add X-position of pixel to unwmomx A4
            ADDA.L  D1,A5     ;add Y-position of pixel to unwmomy A5
            ADDA.L  #1,A6     ;increment beam area by 1
NOVAL4U:    ADDQ.W  #1,D2     ;increment inner loop counter D2
            CMP.W   D2,D4     ;if (D2 > xmax) break out of inner loop
            BGE               QUAD42U
            ADDA.L  A3,A0     ;move to next row of image raster
            ADDQ.W  #1,D1     ;increment outer loop counter D1
            CMP.W   D1,D3     ;if (D1 > ymax) break out of outer loop
            BGE               QUAD41U
            MOVE.L  A2,_quad4;store quadrant intensity
            MOVEA.L #0,A2     ;use A2 to store quadrant intensity
            MOVE.W  _ymax,D3            ;set maximum limits for quad3 to
            D3,D4
            MOVE.W  _xcentd,D4
            MOVE.W  _ycentd,D1          ;initialize outer loop counter D1
            MOVE.W  A3,D0     ;move width of image to D0
            MULS.W  D1,D0     ;compute starting row address quad3
            MOVEA.L A1,A0     ;  i.e. *(image+ycent*width)
            ADDA.L  D0,A0     ;move starting row address to A0
QUAD31U:    MOVE.W  _xmin,D2 ;initialize inner loop counter D2
QUAD32U:    MOVEQ.L #15,D5    ;adjust intensity for complemented colours
            SUB.B   (A0,D2),D5          ;  i.e. pixel = maxintensity - pixel
            ADDA.L  D5,A2     ;add pixel value (D5) to total intensity A2
            CMPI.W  #0,D5     ;if (pixel value == 0)
            BEQ               NOVAL3U             goto NOVAL3U
            ADDA.L  D2,A4     ;add X-position of pixel to unwmomx A4
            ADDA.L  D1,A5     ;add Y-position of pixel to unwmomy A5
            ADDA.L  #1,A6     ;increment beam area by 1
NOVAL3U:    ADDQ.W  #1,D2     ;increment inner loop counter D2
            CMP.L   D2,D4     ;if (D2 > xcent) break out of inner loop
            BGT               QUAD32U
            ADDA.L  A3,A0     ;move to next row of image raster
            ADDQ.W  #1,D1     ;increment outer loop counter D1
            CMP.L   D1,D3     ;if (D1 > ymax) break out of outer loop
            BGE               QUAD31U
            MOVE.L  A2,_quad3;store quadrant intensity
            MOVE.L  A4,_unwmomx         ;store unweighted X-moment
            MOVE.L  A5,_unwmomy         ;store unweighted Y-moment
            MOVE.L  A6,_beamarea  ;store beam area
            MOVEM.L (SP)+,A0-A6/D0-D7 ;restore registers
            RTS

;*******************************************************************
```

```
; This function calculates the total intensity of each quadrant of an image.
; ************************************************************************
; Call format: quadintens()
; Register usage:  D0 - temporary register for intermediate calculations
;                  D1 - outer loop counter
;                  D2 - inner loop counter
;                  D3 - maximum value of outer loop counter
;                  D4 - maximum value of inner loop counter
;                  D5 - used to store pixel intensity value
;                  A0 - address of image pixel to be accessed
;                  A1 - starting address of image raster
;                  A2 - used to store and compute quadrant intensity
;                  A3 - width of image (in pixels)

_quadintens:
          MOVEM.L A0-A6/D0-D7,-(SP) ;save registers on stack
          CLR.L     D0            ;clear data registers
          CLR.L     D1
          CLR.L     D2
          CLR.L     D3
          CLR.L     D4
          CLR.L     D5
          CLR.L     D6
          CLR.L     D7
          MOVEA.L _image,A1           ;set A1 to address of image
          MOVEA.L #0,A2      ;use A2 to store quadrant intensity
          MOVEA.W #WIDTH,A3           ;use A3 to store width of image
          MOVE.W  _ycentd,D3          ;set maximum limits for quad2 to
          D3,D4
          MOVE.W  _xcentd,D4
          MOVE.W  _ymin,D1 ;initialize outer loop counter D1
          MOVE.W  A3,D0      ;move width of image to D0
          MULS.W  D1,D0      ;compute starting row address quad2
          MOVEA.L A1,A0      ;  i.e. *(image+ymin*width)
          ADDA.L  D0,A0      ;move starting row address to A0
QUAD21:   MOVE.W  _xmin,D2 ;initialize inner loop counter D2
QUAD22:   MOVEQ.L #15,D5     ;adjust intensity for complemented colours
          SUB.B     (A0,D2),D5         ; i.e. pixel = maxintensity - pixel
          ADDA.L  D5,A2      ;add pixel value (D5) to total intensity
          ADDQ.W  #1,D2      ;increment inner loop counter D2
          CMP.W   D2,D4      ;if (D2 > xcent) break out of inner loop
          BGT       QUAD22
          ADDA.L  A3,A0      ;move to next row of image raster
          ADDQ.W  #1,D1      ;increment outer loop counter D1
```

```
          CMP.W   D1,D3      ;if (D1 > ycent) break out of outer loop
          BGT       QUAD21
          MOVE.L  A2,_quad2;store quadrant intensity
          MOVEA.L #0,A2      ;use A2 to store quadrant intensity
          MOVE.W  _ycentd,D3          ;set maximum limits for quad1 to
          D3,D4
          MOVE.W  _xmax,D4
          MOVE.W  _ymin,D1 ;initialize outer loop counter D1
          MOVE.W  A3,D0      ;move width of image to D0
          MULS.W  D1,D0      ;compute starting row address quad1
          MOVEA.L A1,A0      ;  i.e. *(image+ymin*width)
          ADDA.L  D0,A0      ;move starting row address to A0
QUAD11:   MOVE.W  _xcentd,D2          ;initialize inner loop counter D2
QUAD12:   MOVEQ.L #15,D5   ;adjust intensity for complemented colours
          SUB.B     (A0,D2),D5         ; i.e. pixel = maxintensity - pixel
          ADDA.L  D5,A2      ;add pixel value (D5) to total intensity
          ADDQ.W  #1,D2      ;increment inner loop counter D2
          CMP.W   D2,D4      ;if (D2 > xmax) break out of inner loop
          BGE       QUAD12
          ADDA.L  A3,A0      ;move to next row of image raster
          ADDQ.W  #1,D1      ;increment outer loop counter D1
          CMP.W   D1,D3      ;if (D1 > ycent) break out of outer loop
          BGT       QUAD11
          MOVE.L  A2,_quad1 ;store quadrant intensity
          MOVEA.L #0,A2      ;use A2 to store quadrant intensity
          MOVE.W  _ymax,D3            ;set maximum limits for quad4 to
          D3,D4
          MOVE.W  _xmax,D4
          MOVE.W  _ycentd,D1          ;initialize outer loop counter D1
          MOVE.W  A3,D0      ;move width of image to D0
          MULS.W  D1,D0      ;compute starting row address quad4
          MOVEA.L A1,A0      ;  i.e. *(image+ycent*width)
          ADDA.L  D0,A0      ;move starting row address to A0
QUAD41:   MOVE.W  _xcentd,D2          ;initialize inner loop counter D2
QUAD42:   MOVEQ.L #15,D5   ;adjust intensity for complemented colours
          SUB.B     (A0,D2),D5         ; i.e. pixel = maxintensity - pixel
          ADDA.L  D5,A2      ;add pixel value (D5) to total intensity
          ADDQ.W  #1,D2      ;increment inner loop counter D2
          CMP.W   D2,D4      ;if (D2 > xmax) break out of inner loop
          BGE       QUAD42
          ADDA.L  A3,A0      ;move to next row of image raster
          ADDQ.W  #1,D1      ;increment outer loop counter D1
          CMP.W   D1,D3      ;if (D1 > ymax) break out of outer loop
          BGE       QUAD41
```

```
                MOVE.L  A2,_quad4;store quadrant intensity
                MOVEA.L #0,A2     ;use A2 to store quadrant intensity
                MOVE.W  _ymax,D3            ;set maximum limits for quad3 to
                D3,D4
                MOVE.W  _xcentd,D4
                MOVE.W  _ycentd,D1          ;initialize outer loop counter D1
                MOVE.W  A3,D0     ;move width of image to D0
                MULS.W  D1,D0     ;compute starting row address quad3
                MOVEA.L A1,A0     ;  i.e. *(image+ycent*width)
                ADDA.L  D0,A0     ;move starting row address to A0
QUAD31:         MOVE.W  _xmin,D2 ;initialize inner loop counter D2
QUAD32:         MOVEQ.L #15,D5    ;adjust intensity for complemented colours
                SUB.B   (A0,D2),D5          ; i.e. pixel = maxintensity - pixel
                ADDA.L  D5,A2     ;add pixel value (D5) to total intensity
                ADDQ.W  #1,D2     ;increment inner loop counter D2
                CMP.L   D2,D4     ;if (D2 > xcent) break out of inner loop
                BGT     QUAD32
                ADDA.L  A3,A0     ;move to next row of image raster
                ADDQ.W  #1,D1     ;increment outer loop counter D1
                CMP.L   D1,D3     ;if (D1 > ymax) break out of outer loop
                BGE     QUAD31
                MOVE.L  A2,_quad3;store quadrant intensity
                MOVEM.L (SP)+,A0-A6/D0-D7 ;restore registers
                RTS
                END
```

D.10 SETCAM.ASM

```
; This function selects the camera from which to accept the video signal
; Filename: "SETCAM.ASM", Version: 2.1, Date: May 5, 1992
; Call format: setcam(long cameranumber)

                DSEG
                public  _gotlive

                CSEG
                public  _setcam

_setcam:
                MOVE.L  4(SP),D0  ;move camera number to D0
                MOVEM.L A0-A6/D1-D7,-(SP) ;save registers on stack
                MOVEA.L _gotlive,A0          ;ptr to address of live board
                CLR.L   D1
                CMP.L   #2,D0     ;if (camera == 2)
                BNE     CAM1
                MOVE.W  #$4000,D1            ; *(gotlive+64) = 0x4000
                BRA     RETURN ;else
CAM1:           MOVE.W  #$0000,D1            ; *(gotlive+64) = 0x0000
RETURN:         MOVE.W  D1,64(A0)
                MOVEM.L (SP)+,A0-A6/D1-D7 ;restore registers
                RTS
                END
```

D.11 CONTROL.H

```
/*********************************************************************
This header file contains constant definitions used by all program.

Filename: "CONTROL.H", Version: 2.1, Date: May 5, 1992

*********************************************************************/

#define HEIGHT   200   /* height of image in pixels */
#define WIDTH    320   /* width of image in pixels */

#define MVPLUS   0x0000 /* bit definitions for controlling mics */
#define MVMINUS  0x0001 /* i.e. bits 0 and 1 of control word */
#define MHPLUS   0x0002 /* MV = vertical mic, MH = horizontal mic */
#define MHMINUS  0x0003

#define DOWNCX   MVMINUS /* bits to move up/down mic for convex mirror */
#define UPCX     MVPLUS  /* i.e. bits 0 and 1 of control word */
#define DOWNCV   MVMINUS /* bits to move up/down mics for concave mirror */
#define UPCV     MVPLUS  /* i.e. bits 0 and 1 of control word */
#define RIGHTCX  MHMINUS /* bits to move left/right mic for convex mirror */
#define LEFTCX   MHPLUS
#define RIGHTCV  MHPLUS  /* bits to move left/right mic for concave mirror */
#define LEFTCV   MHMINUS

#define VTMIC    0x0001 /* bit to select vertical mic of either mirror */
#define HZMIC    0x0002 /* bit to select horizontal mic of either mirror */
#define CONVEX   0xfffb /* bit to select mirror (decoder address): bit2 = 0 */
#define CONCAVE  0x0004 /* bit 2 = 1 */
#define VTMICX   0x0001 /* bits indicate mic of convex mirror */
#define HZMICX   0x0002 /* i.e. HZMIC | CONVEX */
#define VTMICV   0x0005 /* bits indicate mic of convex mirror */
#define HZMICV   0x0006 /* i.e. HZMIC | CONCAVE */

#define CLDEFLT  0x2000 /* default control word for no operations */
#define CLRMIC   0xfffc /* word to clear previous mic setting */
#define CLRMIR   0xfffb /* word to clear previous mirror setting */
#define DCLEAR   0xffbf /* bit to enable and disable micrometer movement */
#define DPRESET  0x0040 /* i.e. sets D input to flip-flop (bit 6)*/
#define DCLOCKL  0xffdf /* bit to start/stop micrometer movement (bit 5) */
#define DCLOCKH  0x0020 /* i.e. pulses clock of D flip-flop */
```

```
#define CNTUP    0xff7f /* bit set to 0 when counting up (bit 7) */
#define CNTDOWN  0x0080 /* bit set to 1 when counting down (automatic mode) */
#define CNTNORM  0x0080 /* bit set to 1 for manual operation */
#define RUNNEG   0x0001 /* bit to check if mic running in negative dir. */
#define CNTLOAD  0x0100 /* bit 8 set to 1 to load counters */
#define CNTENAB  0xfeff /* bit 8 set to 0 to enable counters */
#define CNTCLKL  0xfdff /* bit 9 set to 0 to to clock counters */
#define CNTCLKH  0x0200 /* bit 9 set to 1 to to clock counters */
#define CLRLTCH  0x9fff /* bit 14,13 = 00 to clear latch enable bits */
#define XLATCH   0x0000 /* bit 14,13 = 00 to enable x-y table latch */
#define CLATCH   0x2000 /* bit 14,13 = 01 to enable control latch */
#define LLATCH   0x4000 /* bit 14,13 = 10 to enable counter load latch */
#define PLATCH   0x6000 /* bit 14,13 = 11 to p40 inputs and outputs */
#define ADECODE  0xf3ff /* bit 11,10 = 00 sets address latch to decode mode */
#define ENABLT   0x8000 /* bit 15 = 1 enables selected latch */
#define DISBLT   0x7fff /* bit 15 = 0 disables all latches */
#define TOPCAM   1L   /* value to select top camera input of LIVE board */
#define BOTCAM   2L   /* value to select bottom camera input of LIVE board */
#define PWUP     0x0010 /* control bit to increase/decrease power for PIE-3 */
#define PWDOWN   0x0008 /* i.e. bit 3 = decrease, bit 4 = increase PIE-3 */
#define PWCLR    0xffe7 /* word to clear previous power setting PIE-3 */

/* these values are for error checking and determining current micrometer */
#define BTHRESH  6     /* threshold to check for background noise */
#define STHRESH  15    /* threshold to check for edges of screen  */
#define CNTWAIT  10    /* time to wait for counters to settle */
#define MAXYERR  40000 /* maximum possible vertical error for sens. meas. */
#define MAXXERR  40000 /* maximum possible horizontal error for sens. meas.*/
#define MINYERR  3000  /* minimum possible vertical error for alignment */
#define MINXERR  3000  /* minimum possible horizontal error for alignment */
#define MINPWR   500   /* minimum possible desired power setting */
#define MAXPWR   15000 /* maximum possible desired power setting */
#define SAFEVER  500   /* safe errors for alignment; if errors are greater, */
#define SAFEHER  500   /* alignment safety checks are performed */
#define CENTXER  40    /* maximum x deviation from centre of screen */
#define CENTYER  25    /* maximum y deviation from centre of screen */
#define XDIAM    120   /* minimum x-diameter of beam in pixels */
#define YDIAM    100   /* minimum x-diameter of beam in pixels */
#define MAXLEV   15    /* gray level of background */
#define MAXMOVE  10    /* maximum mic move after which power is changed */
#define MAXMOVEX 250   /* maximum possible X-mic. move for one control action */
#define MAXMOVEY 250   /* maximum possible Y-mic. move for one control action */
#define NOBEAM   5000  /* m inimum intensity value required for control */
#define MAXFILE  30    /* maximum number of image files sequenced */
```

```
#define MAXLINE  20   /* maximum number of lines before printing header */
#define CAMDLAY  22   /* time to pause after switching cameras (1/50 secs) */
#define EDGEMIS  10   /*  number of rows/cols missed when finding screen edges */
#define MINTOP   6    /* minimum pixels from top to top edge of screen */
#define MAXBOT   198  /* maximum pixels from bottom to bottom screen edge */
#define MINLEFT  40   /* minimum pixels from left to left edge of screen */
#define MAXRGHT  279  /* maximum pixels from right to right screen edge */
#define MINCNT   4    /* minimum resolution of counter (in pulses) */
#define DINTENS  8000 /* intensity change required to recheck centre */
#define TAUSCRN  20.0 /* 5 time constants of imaging screen */
#define XGAINMOD 0.2  /* gain modification factor for tuning X-uniformity */
#define YGAINMOD 0.2  /* gain modification factor for tuning Y-uniformity */
#define PGAINMOD 0.1  /* gain modification factor for tuning power controller */

/* these values are for scaling input variables */
#define KILO  1000  /*  scale factor for compensation and offsets */

/* maximum and/or minimum values for input parameters */
#define MINVGN   0.1   /* minimum possible vertical alignment gain */
#define MAXVGN   3.00  /* maximum possible vertical alignment gain */
#define MINHGN   0.1   /* minimum possible horizontal alignment gain */
#define MAXHGN   3.00  /* maximum possible horizontal alignment gain */
#define MINPGAIN 0.1   /* minimum possible power controller gain */
#define MAXPGAIN 5.0   /* maximum possible power controller gain */
#define MAXTIPWR 30.0  /* maximum possible power controller integral time */
#define MINPCONV 1.0   /* minimum possible power conversion gain */
#define MAXPCONV 99.9  /* maximum possible power conversion gain */
#define MAXCEIL  15    /* maximum possible ceiling for digitizer */
#define MINCEIL  0     /* minimum possible ceiling for digitizer */
#define MAXFLR   15    /* maximum possible floor for digitizer */
#define MINFLR   0     /* minimum possible floor for digitizer */
#define MINBEAMW 50    /* minimum possible width of beam image */
#define MINBEAMH 50    /* minimum possible height of beam image */
#define MAXVMIN  9.9   /* maximum possible minimum vertical error */
#define MAXHMIN  9.9   /* maximum possible minimum horizontal error */
#define MAXPMIN  250   /* maximum possible minimum power error */
#define MAXVOFF  40000 /* maximum possible vertical compensation */
#define MAXHOFF  40000 /* maximum possible horizontal compensation */
#define MAXPOFF  99999 /* maximum possible power compensation */
#define MINSAMP  0.75  /* minimum possible sampling period */
#define MAXSAMP  30.0  /* maximum possible sampling period */

/* These are the digital input channels used */
#define WNDINPUT 0x0800 /* channel to minitor wand pulse (bit 11) */
```

```
#define MINWPER   0.5  /* minimum wand period */
#define WPERFACT  2.18e-6 /* conversion factor from count increments to time */
#define WPEROFF   1.33e-4 /* offset factor for count inc. to time conversion */
#define NUMSPAVG  10   /* number of readings to average for speed measurement */

/* These are the A/D channels */
#define CURCHAN   1    /* current sensor channel */
#define CRSCALE   7.20 /* multiplying factor for current */
#define CROFF     0.0  /* offset for current conversion */
#define PWMCHAN   0    /* power meter channel */
#define PWSCALE   -3.33 /* multiplying factor for power meter reading */
#define PWOFF     -0.064 /* offset for power meter conversion */
#define VLTCHAN   2    /* voltage channel */
#define VLSCALE   0.86 /* multiplying factor for voltage reading */
#define VLOFF     0.0  /* offset for voltage conversion */
#define PIXTOMM   0.6  /* pixel to mm conversion factor */
#define MIRSTEP   100  /* mirror motion size for sensitivity measurements */
#define MAXSTEP   99   /* maximum steps of MIRSTEP in 1 dir. for sen. meas. */
#define POWFACT   0.7  /* power factor at which to stop sensitivity meas. */
#define MININTN   60000 /* intensity below which wand speed is decreased */
#define MAXINTN   100000 /* intensity above which wand speed is increased */
#define OPTINTN   80000 /* intensity above which wand speed is increased */
#define XCNOISE   1    /* minimum x-centre error for adjuting beam steer */
#define YCNOISE   1    /* minimum y-centre error for adjuting beam steer */
#define XCMOVE    55   /* x-step size to move concave if steer error exists */
#define YCMOVE    55   /* y-step size to move concave if steer error exists */
#define NUMSIGS   6    /* number of signals that may be plotted at one time */
#define WNDSCALE  1.333 /* scaling factor for wand coupling percentage */

/* Parameters for audio device to speak messages */
#define RATE      200  /* rate of speech (0 to 400)*/
#define PITCH     150  /* pitch of speech (65 to 20) */
#define MODE      0    /* natural (0) or monotone (1) */
#define SEX       0    /* male (0) or female (1) voice */
#define SAMPFRQ   22200 /* sampling frequency (5000 to 40000 Hz) */

#define POWCHAN   1    /* D/A channel to control power */
#define WNDCHAN   2    /* D/A channel to control wand speed */
#define PWRSETD   7.5  /* default power setpoint if no control is desired */
#define UPWRMAX   15.0 /* maximum value of power controller output */
#define WNDSETD   9.0  /* default wand setpoint if no control is desired */
#define UWNDMAX   15.0 /* maximum value of wand contrller controller output */
#define PWRSTEP   0.5  /* minimum step (in Watts) by which to change power */
#define WNDSTEP   0.1  /* minimum step (in Watts) by which to change power */
```

```
#define WNDELAY  5    /* # of sample periods delay after a wand control action */
#define ALGSTEP  1.0  /* number of microns to move for alignment sens. meas. */
#define POSSTEP  15.0 /* number of microns to move for position sens. measurements
                      */
#define POSGRID  40   /* grid size position sensitivity measurements */
#define ERRFLOAT 0.05 /* minimum difference to check if 2 floats are equal */
#define ERRINTN  400  /* minimum intensity difference signifying no int. change */
#define ERRPOWER 100  /* minimum difference to check if power level is correct */
#define INVALID  999  /* value to indicate last mic. direction is invalid */

#define PI       3.141593
#define FCUTMIN  0.001 /* minimum cutoff frequency for filters */
#define FCUTMAX  10.0  /* minimum cutoff frequency for filters */
#define FIRSTORD 1  /* value to choose first order filter */
#define BUTWORTH 2  /* value to choose Butterworth filter */
#define MOVEAVG  3  /* value to choose moving average filter */
#define NUMFILTS 8  /* number of data variables to be filtered */
#define MAXORDER 3  /* maximum order of all filters used */
#define POWFILT  0  /* filter number for power meter */
#define CURFILT  1  /* filter number for current */
#define VOLTFILT 2  /* filter number for voltage */
#define INTNFILT 3  /* filter number for intensity */
#define XERRFILT 4  /* filter number for X-alignment error */
#define YERRFILT 5  /* filter number for Y-alignment error */
#define XCENFILT 6  /* filter number for X-centre */
#define YCENFILT 7  /* filter number for Y-centre */
```

D.12 CONTGADG.H

This header file contains all window, menu, and gadget definitions specific to the Amiga. Due to its length (3189 lines), this file is not listed.

D.13 DISPLAY.H

```
/**********************************************************************
This header file contains constant definitions used by all program.

Filename: "DISPLAY.H", Version: 2.1, Date: May 5, 1992

**********************************************************************/

/* Window dimensions */
#define CNWIDTH  528 /* width of ControlWindow */
#define CNHEIGHT 196 /* height of ControlWindow */
#define PLWIDTH  528 /* width of PlotWindow */
#define PLHEIGHT 204 /* height of PlotWindow */

/* Pen numbers to draw gadget borders/images/text with */
#define RED    3   /* color in register 3 was red */
#define BLACK  2   /* color in register 2 was black */
#define WHITE  1   /* color in register 1 was white */
#define BLUE   0   /* color in register 0 was blue */

/* Graphing window outline */
#define GWXMIN  32
#define GWXMAX  PLWIDTH-76
#define GWYMIN  12
#define GWYMAX  PLHEIGHT-32
#define GWXMID  (GWXMAX-GWXMIN)/2+GWXMIN
#define GWYMID  (GWYMAX-GWYMIN)/2+GWYMIN
#define GWXLEN  (GWXMAX-GWXMIN)     /* length of positive x-axis */
#define GWYLEN  (GWYMAX-GWYMIN)/2   /* length of positive y-axis */

#define YTICKS  10  /* number of divisions for y-axis */
```

```
#define XTICKS 7    /* number of divisions for x-axis */
#define PLTXINC 2   /* number of pixels to move on x-axis for one sample */

#define ERSCALE 5    /* maximum scale for errors (kW,mm) for y-axis */
#define PMSCALE 10   /* maximum scale for power (kW) for y-axis */
#define TMSCALE 3.5  /* maximum number of minutes to plot on x-axis */

#define XALG   0   /* plot array element for x-alignment error */
#define YALG   1   /* plot array element for y-alignment error */
#define XPOS   2   /* plot array element for x-position error */
#define YPOS   3   /* plot array element for y-position error */
#define SPWR   4   /* plot array element for screen power */
#define MPWR   5   /* plot array element for power meter output */
```

D.14 LIVEBRARY.H

This header file is supplied with the LIVE 2000 frame grabber and is not listed.

D.15 MAKEFILE

```
OBJ1 = control.o bptopix.o contgadg.o grabfram.o qintcent.o livebrary.o \
     talk.o p40.o setcam.o display.o filter.o

CFLAGS = -c2 -f8 -mcd -wadoru -hi header.dmp
AFLAGS = -c -d

control: $(OBJ1)
      ln $(OBJ1) -o $@ -lm8l -lcl

control.o: control.h
contgadg.o: control.h contgadg.h display.h
display.o: control.h display.h
grabfram.o: livebrary.h
filter.o: control.h
```