NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

INVESTIGATION OF A CCD CAMERA
FOR MEASUREMENTS OF
OPTICAL ATMOSPHERIC TURBULENCE

by

William Jeffrey Rall

March, 1992

Thesis Advisor: Donald L. Walters

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved OMB No 0704-0188

1a REPORT SECURITY CLASSIFICATION: Unclassified	1b RESTRICTIVE MARKINGS
2a SECURITY CLASSIFICATION AUTHORITY	3 DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
2b DECLASSIFICATION/DOWNGRADING SCHEDULE	
4 PERFORMING ORGANIZATION REPORT NUMBER(S)	5 MONITORING ORGANIZATION REPORT NUMBER(S)
6a NAME OF PERFORMING ORGANIZATION: Naval Postgraduate School; 6b OFFICE SYMBOL (If applicable): Code 33	7a NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000	7b ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000
8a NAME OF FUNDING/SPONSORING ORGANIZATION: Phillips Laboratory Attn: Cpt. A. Slavin; 8b OFFICE SYMBOL (If applicable): PL/LTE	9 PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c ADDRESS (City, State, and ZIP Code): Kirtland Air Force Base Albuquerque, NM 87117	10 SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO; PROJECT NO; TASK NO; WORK UNIT ACCESSION NO

11 TITLE (Include Security Classification)

INVESTIGATION OF A CCD CAMERA FOR MEASUREMENTS OF OPTICAL ATMOSPHERIC TURBULENCE

12 PERSONAL AUTHOR(S)

Rall, William J. in conjunction with Walters, Donald L.

13a TYPE OF REPORT	13b TIME COVERED	14 DATE OF REPORT (Year, Month, Day)	15 PAGE COUNT
Master's Thesis	FROM _____ TO _____	1992 March	88

16 SUPPLEMENTARY NOTATION

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the DOD or US Government.

17 COSATI CODES			18 SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Atmospheric optical turbulence, CCD sensor, Image scintillation, Spatial coherence length

19 ABSTRACT (Continue on reverse if necessary and identify by block number)

Atmospheric turbulence introduces random phase distortions in optical imaging systems. The development of new laser and imaging systems requires information on the spatial and temporal distribution of this atmospheric turbulence. Measurements of the image spread and the jitter induced by the atmosphere on an optical system provide two techniques to quantify these phenomena. This thesis evaluates a Spectra Sources Lynxx PC Plus charge coupled device (CCD) array as an atmospheric turbulence sensor. Data acquisition and processing programs were written to measure the image spread of a point source and centroid jitter of a point source imaged through the atmosphere. Since atmospheric jitter measurements require high image frame rates on the order of 200 images per second, a large portion of this thesis involved measurements of the times for the CCD detector, interface board and IBM compatible computer to perform their

20 DISTRIBUTION/AVAILABILITY OF ABSTRACT	21 ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL: D. L. Walters	22b TELEPHONE (Include Area Code): 408-646-2267; 22c OFFICE SYMBOL: PH/WE

DD Form 1473, JUN 86 — Previous editions are obsolete — SECURITY CLASSIFICATION OF THIS PAGE

S/N 0102-LF-014-6603 — UNCLASSIFIED

19. tasks. Recommendations for higher performance are presented.

Approved for public release; distribution unlimited.

Investigation of a CCD Camera for
Measurements of Optical Atmospheric Turbulence

by

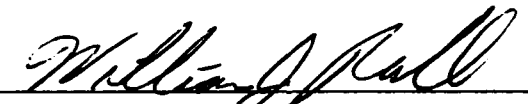
William J. Rall
Lieutenant, United States Coast Guard
B.S., United States Coast Guard Academy, 1984

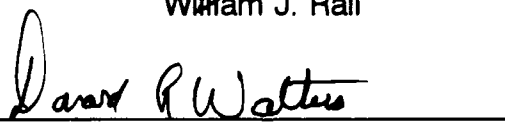
Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN PHYSICS

from the

NAVAL POSTGRADUATE SCHOOL
March 1992

Author: 
William J. Rall

Approved by: 
Donald L. Walters, Thesis Advisor

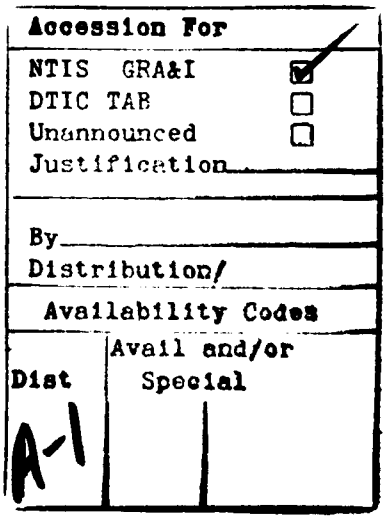

D. Scott Davis, Second Reader


Karlheinz E. Woehler,
Chairman, Department of Physics

ABSTRACT

Atmospheric turbulence introduces random phase distortions in optical imaging systems. The development of new laser and imaging systems requires information on the spatial and temporal distribution of this atmospheric turbulence. Measurements of the image spread and the jitter induced by the atmosphere on an optical system provide two techniques to quantify these phenomena. This thesis evaluates a Spectra Sources Lynxx PC Plus charge coupled device (CCD) array as an atmospheric turbulence sensor. Data acquisition and processing programs were written to measure the image spread of a point source and centroid jitter of a point source imaged through the atmosphere. Since atmospheric jitter measurements require high image frame rates on the order of 200 images per second, a large portion of this thesis involved measurements of the times for the CCD detector, interface board and IBM compatible computer to perform their tasks. Recommendations for higher performance are presented.

TABLE OF CONTENTS



ACKNOWLEDGEMENTS

I would like to express "thanks" to all those who made my tour at NPS both enjoyable and successful. To my thesis advisor, Professor Donald Walters, for his patience and endless enthusiasm. To our technical support personnel, Dale Galarowicz and George Jaksha, with their ever ready rapid and professional assistance. And finally, to my wife, Roblynn, for always being there when needed.

I INTRODUCTION

This thesis considers a method used to measure atmospheric turbulence using a silicon charge coupled device (CCD) camera. Light from a distant star consists of almost perfect plane waves before entering the earth's atmosphere. When the waves traverse the atmosphere, local variations in the index of refraction induce warping of the wave fronts. Therefore, a telescope pointed at a star will produce a distorted image, whose spatial and temporal characteristics depend upon the properties of the atmospheric path followed by the starlight. Typically, the atmosphere introduces both image spread and temporal jitter, so that a CCD array placed at the telescope's focal plane will reveal a highly unstable and complex irradiance pattern. It has been found that both a high frame rate (> 200 per second) and correspondingly short exposure time (< 5 ms) are necessary to avoid under-sampling the dynamic variations of typical images.

The goal of this thesis was to find a way to measure atmospheric turbulence with a CCD camera by optimizing the speed and efficiency of the camera system's software and hardware. The software was written in the C language. Software components that can affect system speed include both the various algorithms used and the overall efficiency of the compiled program code. Three compilers' outputs were compared for executable code speed and utility:

Microsoft Quick C 2.5, Borland C++ 2.0 and Borland Turbo C++ 1.0. Fixed hardware components included a Texas Instruments TC211 CCD image sensor in a Spectra Source PC Lynxx Plus camera attached to an 8 bit IBM PC compatible interface card. The camera system was installed in two IBM compatible personal computers, one using an Intel 80386 and the other a 80486 processor, and the overall speeds were compared. Finally, the camera-computer interface and the CCD image sensor were analyzed to determine whether such a low-cost commercial system would be suitable for turbulence measurements, or whether a customized device would have to be fabricated.

II BACKGROUND

A. ATMOSPHERIC TURBULENCE

Turbulence in the atmosphere causes many problems. In astronomy, it introduces distortion that obscures image detail. Since exoatmospheric use is not practical for most telescopes, knowing the spatial and temporal distribution of turbulence is essential in the planning stages of a new facility or in diagnostic evaluation of test systems. Figure 1 sketches a wave front incident on a turbulent region [Ref. 1]. Random phase fluctuations in the index of refraction field produce scintillations and image blurring. Twinkling or scintillation arises from the interference of starlight that traverses multiple paths through the atmosphere. Image blurring arises from the reduction in spatial coherence of the wave front.

B. PARAMETERS FOR THE MEASUREMENT OF TURBULENCE

The atmospheric index of refraction depends on both temperature and pressure. Since pressure fluctuations disperse rapidly, at the speed of sound, temperature fluctuations are the main contributor to atmospheric optical turbulence. Structure functions provide a way to quantify the statistics of atmospheric turbulence. For propagation through the atmosphere, optical parameters will involve an integral of the refractive structure function along the

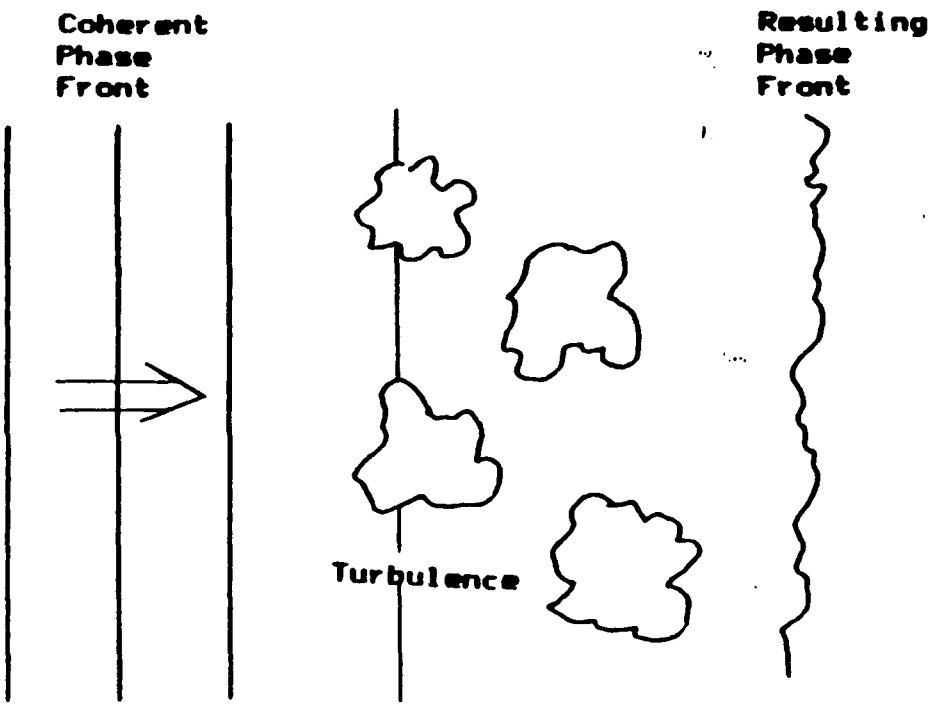


Figure 1. Optical atmospheric turbulence distorts a coherent phase front.

optical path. The primary optical parameter is the spatial coherence length, r_o. It represents the transverse autocorrelation length of the electromagnetic field. A related parameter is the isoplanatic angle, θ_o. It represents the angle between two different paths where the Strehl ratio of an ideal adaptive optics system is within e^{-1} of perfect correction [Ref. 2]. Another parameter, the Greenwood frequency, f_g, represents the electrical bandwidth needed to remove atmospheric phase distortions with an adaptive optical system.

1. Structure Function

Atmospheric turbulence produces localized variations in the index of refraction along an optical path. Tatarski [Ref. 3] defines the structure function by

$$D_x(\bar{r}_2-\bar{r}_1) = \langle [x(\bar{r}_2)-x(\bar{r}_1)]^2 \rangle, \qquad (1)$$

where < > represents an ensemble spatial average, $\bar{r}_1$ and $\bar{r}_2$ represent the location of two points in space, and x represents an atmospheric parameter of interest, such as temperature or index of refraction. Assuming the turbulence is isotropic, homogeneous and incompressible [Ref. 4] the Kolmogorov theory of turbulence [Ref. 5] shows that

$$D_x = C_x^2 r_{12}^{2/3}, \qquad (2)$$

where C_x^2 is a structure parameter and r_{12} is the distance between the two points in space. By combining equations (1) and (2), the thermal structure parameter, C_T^2, which characterizes the mean squared temperature difference between two points in space is [Ref. 3],

$$C_T^2 = \frac{\langle (T_2-T_1)^2 \rangle}{r_{12}^{2/3}}, \qquad (3)$$

where T_2-T_1 is the temperature difference and r_{12} is the distance between the two points in space. There is a similar expression for the index of refraction structure parameter, C_n^2. Taking the partial derivatives of the atmospheric index of refraction [Ref. 3],

$$n-1 = \frac{79\times10^{-6}P}{T}, \qquad (4)$$

where P is the atmospheric pressure in mbar and T is the Kelvin temperature, and assuming isobaric turbulence allows us to write C_n^2 in terms of C_T^2 [Ref. 6],

$$C_n^2 = \left(\frac{79\times10^{-6}P}{T^2}\right)^2 C_T^2. \qquad (5)$$

The pressure is assumed to be constant, since small random pressure differences disperse rapidly.

2. Spatial Coherence Length

The spatial coherence length, r_o, as stated above, measures the transverse autocorrelation length of a wave front. It represents the effective aperture diameter of a diffraction-limited optical system with a similar angular resolution as the system under study. Typical values of r_o range from two or three centimeters for high turbulence to as much as thirty centimeters for low turbulence, when measured from the earth's surface upward. When observing a star from the ground with an optical device, the C_n^2 component of r_o is cumulative along the optical path, even though most of the degradation of the wave front occurs close to the ground. Integrating the optical turbulence $C_n^2(z)$ along the path [Ref. 6],

$$r_o = \left[\frac{2.91}{2} k^2 \sec\phi \int_0^L C_n^2(z)\, W(z)\, dz\right]^{-3/5}, \qquad (6)$$

where k is the wave number($2\pi/\lambda$), ϕ is the zenith angle, L is the vertical path

length and W(z) is a weighting function. Typically W(z) has one of three forms: for a plane wave, W=1; for a spherical diverging wave, $W=(z/L)^{5/3}$; and for a converging wave, $W=(1-z/L)^{5/3}$. The factor of sec ϕ compensates for the slant path through a horizontally stratified atmosphere.

Temperature fluctuations along a vertical path have been measured by two means: by microthermal probes carried by a meteorological balloon and by an echo sounder [Ref. 7 and 8]. Both of these methods provide credible values of r_o. However, there is a more direct method for calculating the coherence length at the earth's surface using an optical point source irradiance distribution [Ref. 9]. The discrete irradiance values from a two dimensional irradiance distribution represent the point spread function, P(x,y). The line spread function for x, L(x), is the summation of all of the y irradiance values from P(x,y) for each x. L(y) is a similar summation of the x values for each y. Then take the Fourier transform of L(x) or L(y) to get the optical transfer function (OTF). Assuming the turbulence is isotropic, that is symmetric with respect to rotation about the path, and that it is laterally stationary, the absolute value of the OTF equals the modulation transfer function (MTF). For an optical system imaging a star the observed quantity, MTF_o, is a product of the MTF's of the source or star, the atmosphere and the instrument [Ref. 9],

$$MTF_o = MTF_s \cdot MTF_a \cdot MTF_i . \qquad (7)$$

MTF_s for a star corresponds to a point source at infinity and is unity. MTF_i is a

measured instrumental transfer function. To determine the atmospheric component, MTF_a, from the observed quantity, MTF_o, we divide MTF_o by the measured instrumental transfer function, MTF_i.

Fried shows that atmospheric MTF_a is a function of the atmospheric wave structure function, which is an integral of the index of refraction structure parameter along the path [Ref 2]. MTF_a reduces to a form [Ref. 9],

$$MTF_a = e^{-3.44\left(\frac{\lambda R \nu}{r_o}\right)^{5/3}}, \qquad (8)$$

where R is the effective focal length, λ is the wavelength of light and ν is the spatial frequency. To determine the atmospheric parameter r_o, we took a one dimensional fast Fourier transform of a line spread function of a star image. Dividing this by the known instrument function gives the spatial spectrum similar to Equation (8). The next step was to determine the e^{-1} frequency for this experimental spatial spectrum from which [Ref. 9],

$$r_o = 2.1\ \lambda\ R\ \nu_e, \qquad (9)$$

where ν_e was the e^{-1} frequency of the MTF_a.

Yet another way to calculate r_o uses image centroid jitter, also called beam wander, by calculating the x and y centroid standard deviation from a sequence of point source images. It is equal to αF, where α is the atmospheric angle of arrival fluctuations and F is the optical system's effective focal length. The angle of arrival is sketched in Figure 2. Mathematically, angle of arrival

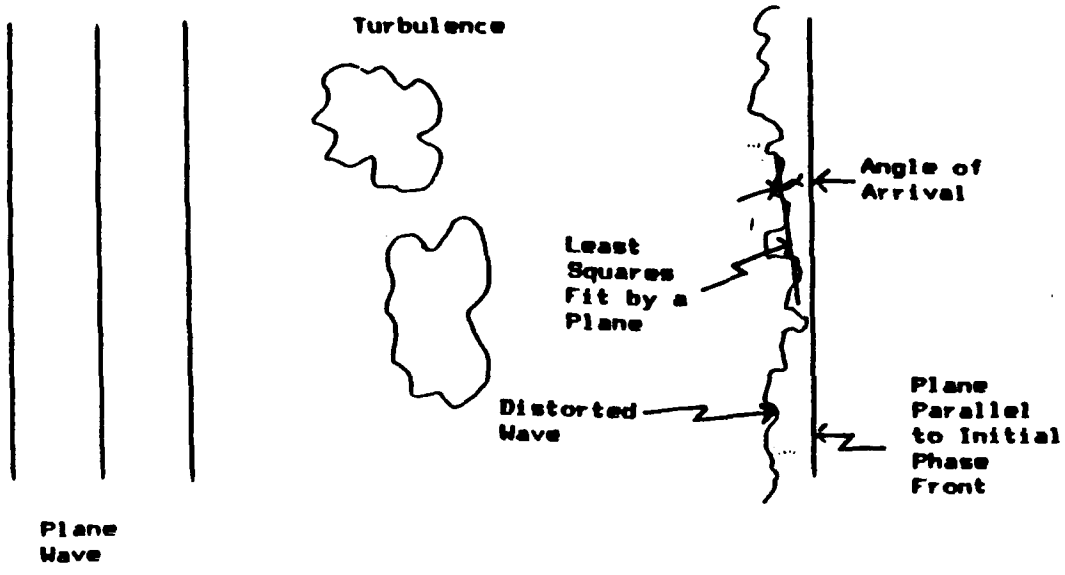


Figure 2. The angle of arrival is the least-squares plane that fits the electric field vector.

fluctuations are equivalent to least-squares planes that fit the electric field across an aperture. In terms of the index of refraction structure parameter [Ref. 5 and 10],

$$\alpha^2 = \frac{1.026}{D^{1/3}} \int_0^L C_n^2(z)\, dz, \qquad (10)$$

where D is the diameter of the aperture and L is the optical path. Combining equation (10) with equation (6) for a plane wave,

$$r_o = [1.418 k^2 \sec\phi\ D^{1/3} \alpha^2]^{-3/5}. \qquad (11)$$

To calculate r_o from the centroid motion requires a very high image frame rate to "freeze" the atmosphere. Characteristically at least 200 frames per second are required in order to avoid aliasing that would otherwise underestimate the true centroid variance.

3. Isoplanatic Angle

The isoplanatic angle measures the angular coherence in the vicinity of an object. It defines a cone that constrains the angles over which an adaptive optical system will provide valid correction. In terms of C_n^2 [Ref. 1],

$$\theta_0 = [2.91 k^2 \int_0^L C_n^2(z)\, z^{5/3}\, dz]^{-3/5}, \qquad (12)$$

where the parameters are the same as those of r_o. Simple instruments exist to measure θ_o [Ref. 11]. For this reason, only the coherence length is addressed in this thesis.

4. Greenwood Frequency

The Greenwood frequency was introduced in an earlier paper by Darryl Greenwood [Ref. 12 and 13]. It is a mean temporal frequency of an adaptive optics system and depends on the wind velocity, V, and C_n^2 along the path. For a Kolmogorov turbulence spectrum, the Greenwood frequency is [Ref. 13],

$$f = 2.31 \lambda^{-6/5} [\int_0^L C_n^2(z)\, V^{5/3}(z)\, dz]^{3/5}, \qquad (13)$$

where λ is the wavelength of light, z is the distance along the propagation path, V(z) is the wind speed and L is the distance from the source to the receiving optical device.

C. CHARGE COUPLED DEVICE (CCD)

An atmospheric coherence length sensor needs an imaging detector to measure the point spread function. The speed, reliability and availability of charge coupled devices (CCD's) makes them worthy of consideration. The basis of a CCD is a metal-oxide-semiconductor (MOS) capacitor forming potential wells and channels that comprise light-sensitive pixels and read-out registers. Figure 3 is a sketch of the MOS capacitor structure [Ref. 14]. Electrons excited by the photoconductive effect are trapped in potential wells formed under the positively

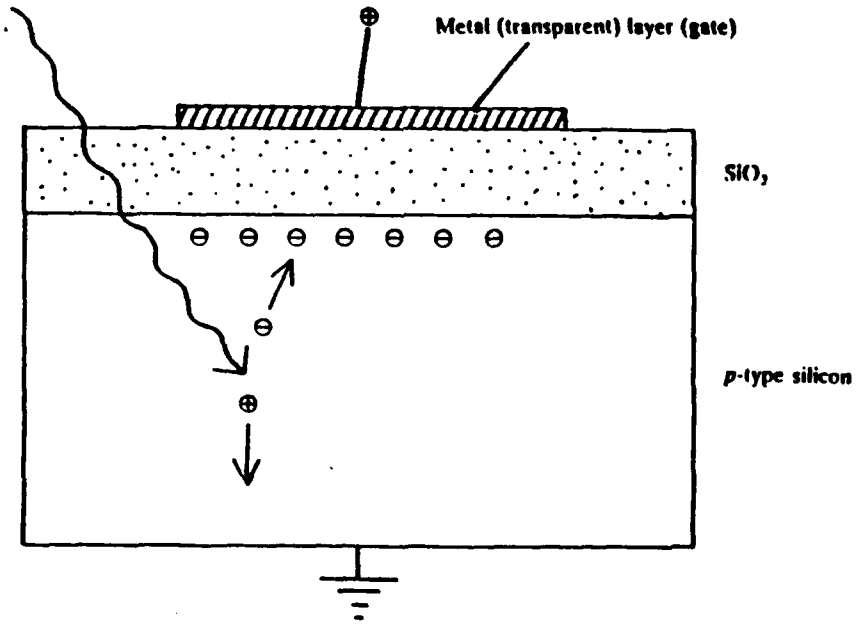


Figure 3. The metal-oxide-semiconductor (MOS) capacitor structure,the basis of the charge coupled device (CCD).

charged metal gate contact. The number of electrons trapped is proportional to the integrated irradiance during an exposure.

There are several methods to read-out the device. Voltages with different phases placed on the gates transfer charge from pixel to pixel or potential well to potential well. A set of electrodes connected together is called a phase. Figure 4 shows an example of a three phase device with three pairs of gates (G) and three lines (L) connecting the electrodes [Ref. 14]. A positive electrical pulse

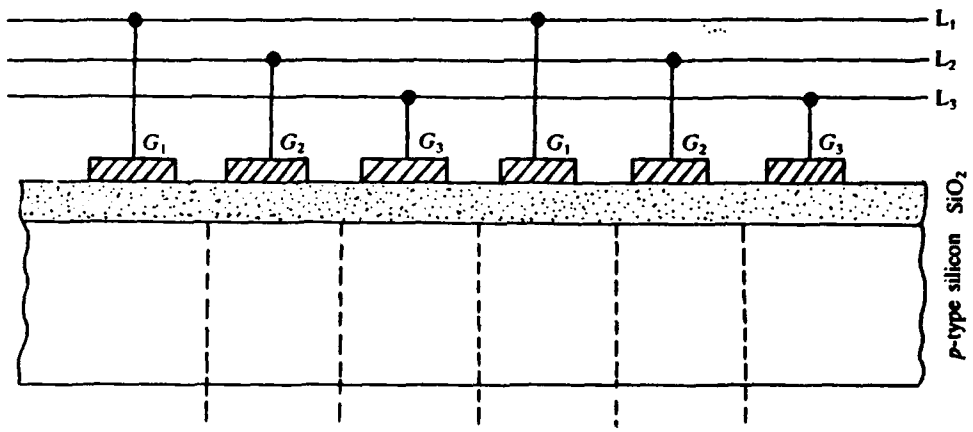


Figure 4. Three pairs of gates (G) are connected with the lines (L) to form three different phases.

from a clock forces a transfer of charge from one phase of pixels to the next. Clocking a phase of cells reduces the barrier between pixel columns allowing charge transfer. A bucket of water running down several steps with boards or barriers on each step is a visual representation of this process. The water flows down to a lower potential each time the board is lifted up. The number of phases used to transfer out charge packets represents the number of steps.

There are two methods used to insure that efficient charge transfer occurs in one direction. The first uses multiple phases to separate the charge packets. The Texas Instruments TC211 [Ref. 15] uses the second method which is to force

an asymmetry in each well using an ion implantation zone between each pixel. The clocking method is the monophase mode, also referred to as a 1+½ phase mode, sketched in Figure 5 [Ref. 16]. One phase has an intermediate voltage level while the voltages applied to the other phase vary on both sides of this level.

The charge packets transfer down channels. There are two types of channels, surface and buried. The water analogy also works here. In a buried channel, there is less loss of charges, similar to a pipe. A surface channel, however, has a higher capacity, like a canal, but this CCD has higher noise from surface imperfections.

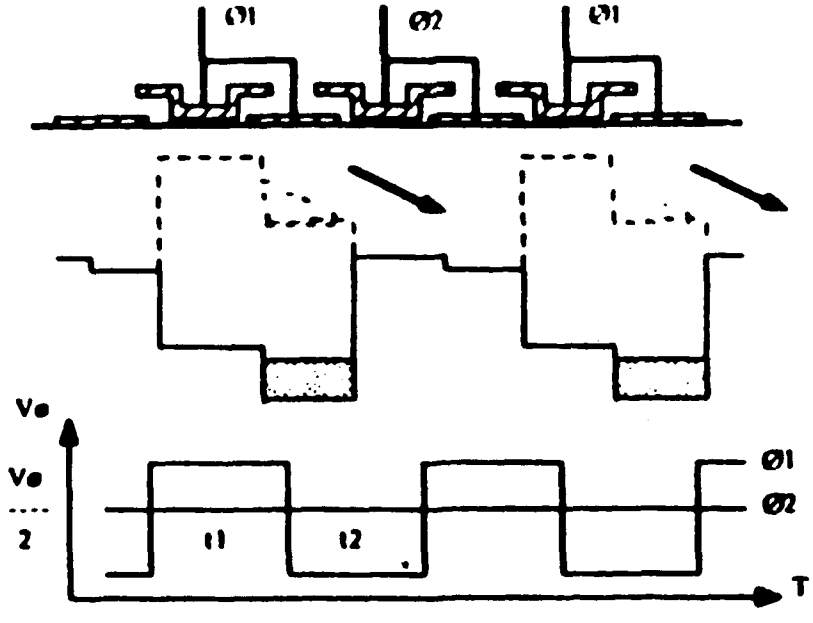


Figure 5. The monophase clocking method holds a phase at an intermediate level while the voltages are applied on either side.

During the read-out of a CCD, image smear will occur since light will still produce charge in the moving CCD rows, unless a shutter is incorporated. Different techniques exist to reduce image smear during read-out. Frame transfer devices overcome this disadvantage by effectively having two arrays, one for image exposure and a second for storage. Moving charges from the image area to an opaque, masked set of pixels quickly allows another exposure to commence while reading the storage area. Figure 6 shows two examples [Ref. 14].

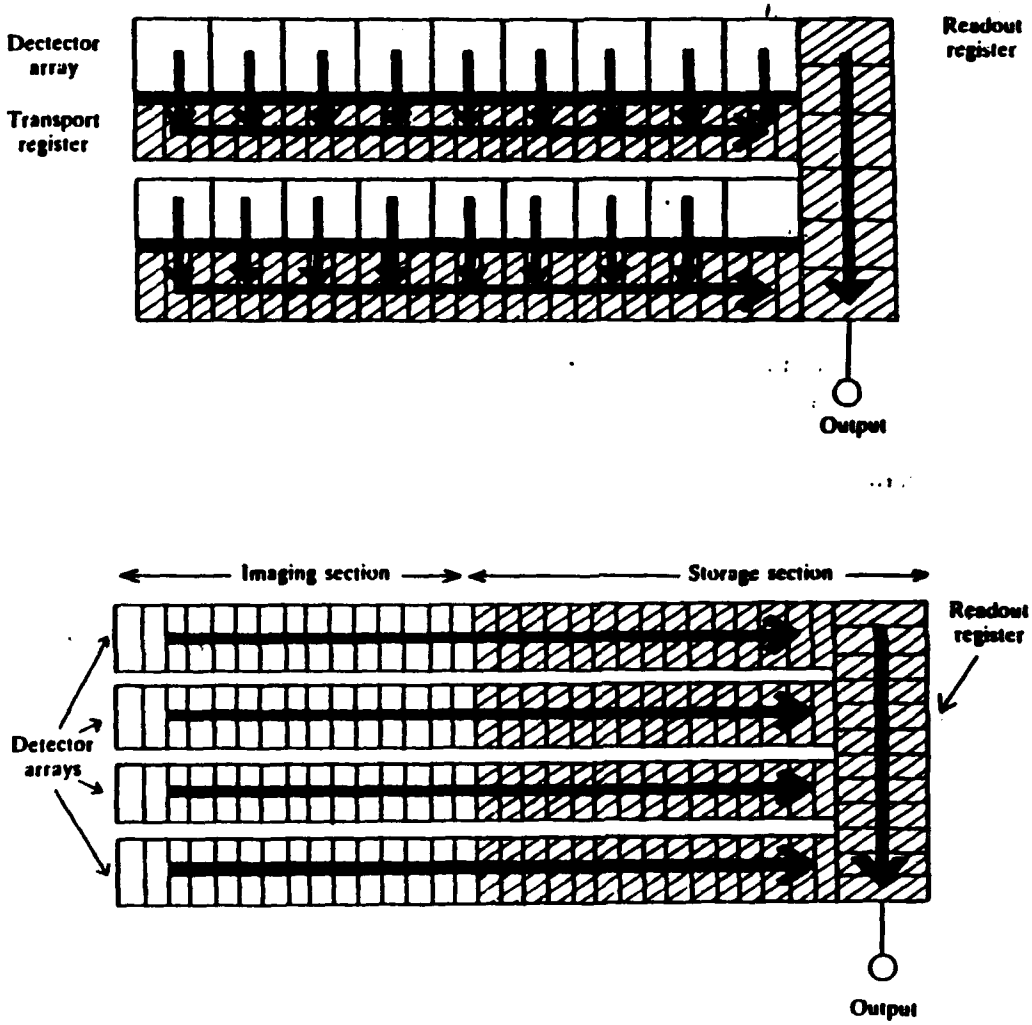


Figure 6. Two schemes for CCD read-out. Top, the interline method; bottom, frame transfer method.

III DISCUSSION

A. EQUIPMENT

This thesis investigated a Spectra Source Lynxx PC Plus CCD Imaging System [Ref. 17 and 18]. The original intent was to use this CCD camera with a telescope and personal computer to develop software to measure the atmospheric coherence length, r_o. During this process it became clear that the off-the-shelf CCD system could not provide sufficient frame rates to achieve the intended goal. Consequently, the task shifted to determine the causes of the sluggish acquisition rates. The CCD detector array, computer interface card and the IBM compatible PC were investigated.

1. CCD Camera

The camera contains a 192 X 165 pixel Texas Instruments TC211 CCD image sensor. Its specifications are in Appendix A [Ref. 19]. Figure 7 is a photograph of the sensor centered inside the camera. The other components inside the camera head translate the clock signals from the interface board and amplify the CCD output.

Figure 8 shows a functional block diagram of a TC211 CCD image sensor [Ref. 19]. Key components include the silicon matrix of pixels and a set of gates to shift the charge. The image area gate (IAG) performs a parallel shift of all rows for each clock pulse. As a row enters the output serial register the

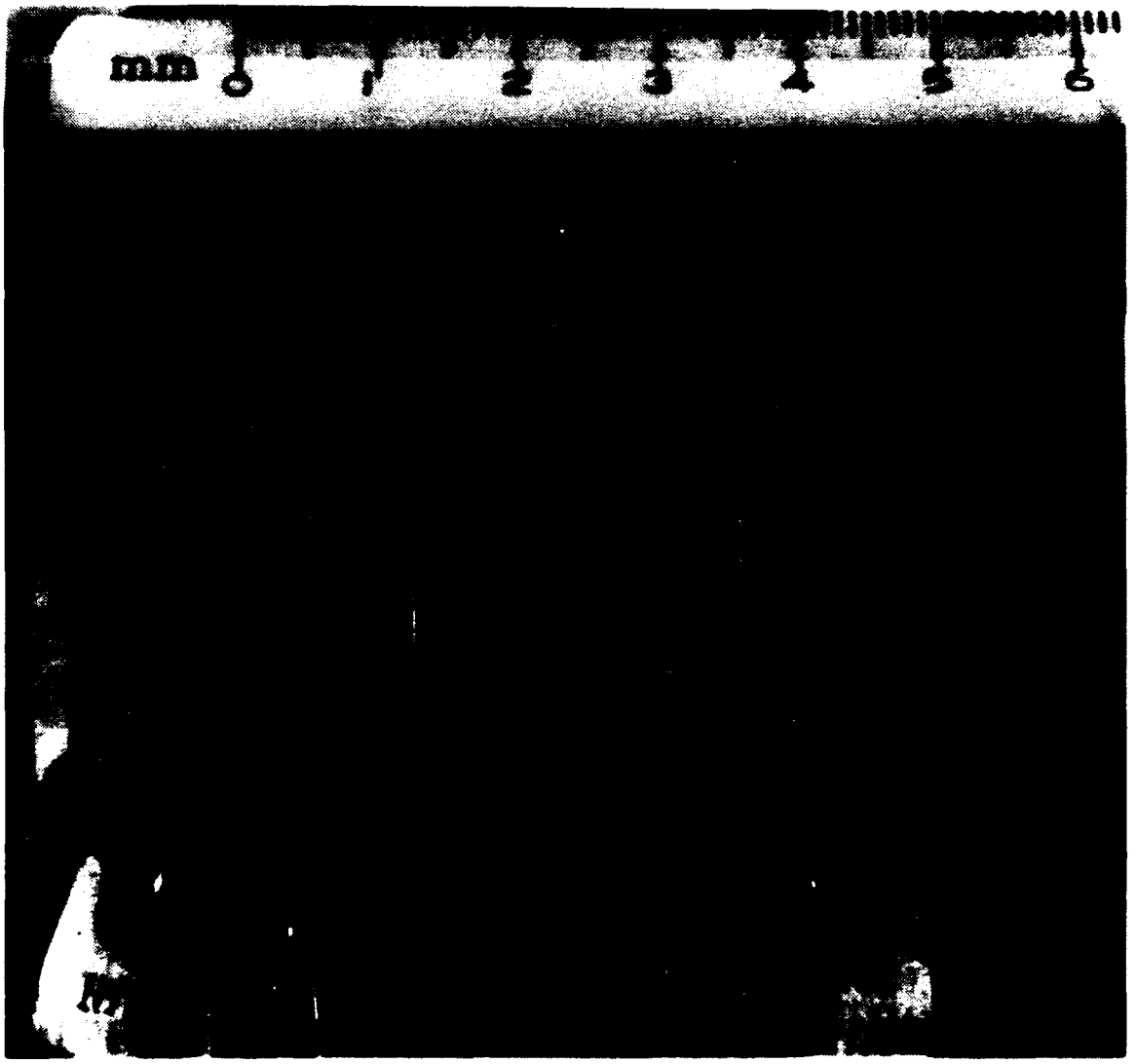


Figure 7. Photograph of inside the CCD camera head of the Spectra Sources Lynxx PC+ system.

serial register gate (SRG) shifts the row to the charge detection amplifier. The TC211 includes an antiblooming gate (ABG) that should not be used since it introduces a severe nonlinear photo response.

The method used to acquire and to read-out an exposure determines the CCD sensor's read-out rate. The electrons collect in potential wells, or pixels

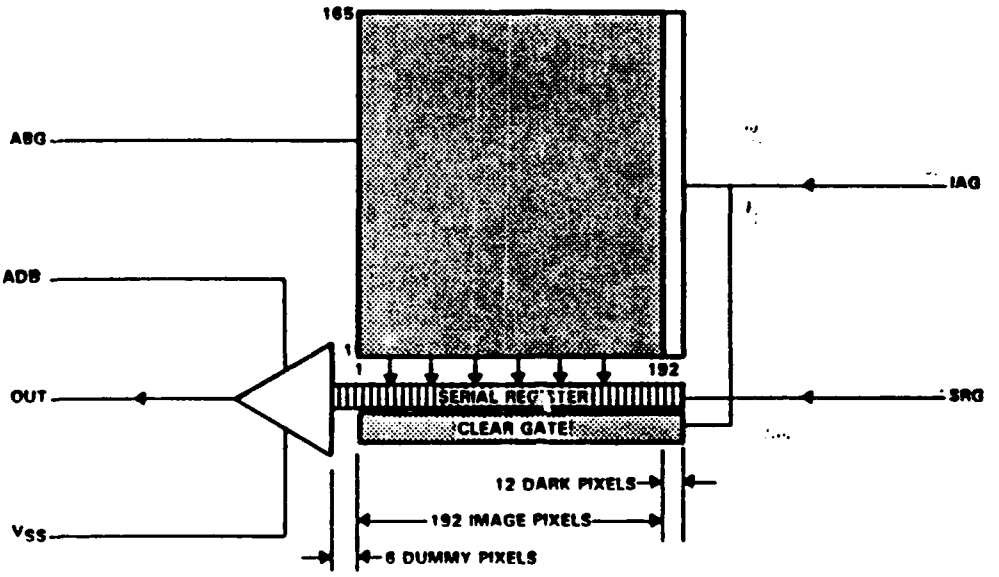


Figure 8. Lynxx PC+ image sensor CCD, Texas Instruments TC211 block diagram.

after exposing the image area to light. The charges then shift down to the output register as rows, through 165 timing cycles. The output register then shifts the charges 210 times for each row. This cycle of 210 shifts includes 12 dark pixels for a dark reference and 6 dummy pixels used to transfer the charges out of the register. Figure 9 shows the charge transfer process [Ref. 19]. Another factor to consider for an image sensor chip is the noise. The noise equivalent signal for the TC211 is 150 electrons. It depends on $(kTC)^{1/2}$ /q, where C is the capacitance of the read-out charge collector, k is the Boltzmann constant, T is the absolute temperature and q is the charge of an electron.

The integration time needed to achieve a particular signal-to-noise ratio depends on both the telescope optics and the object's radiance. It will be a

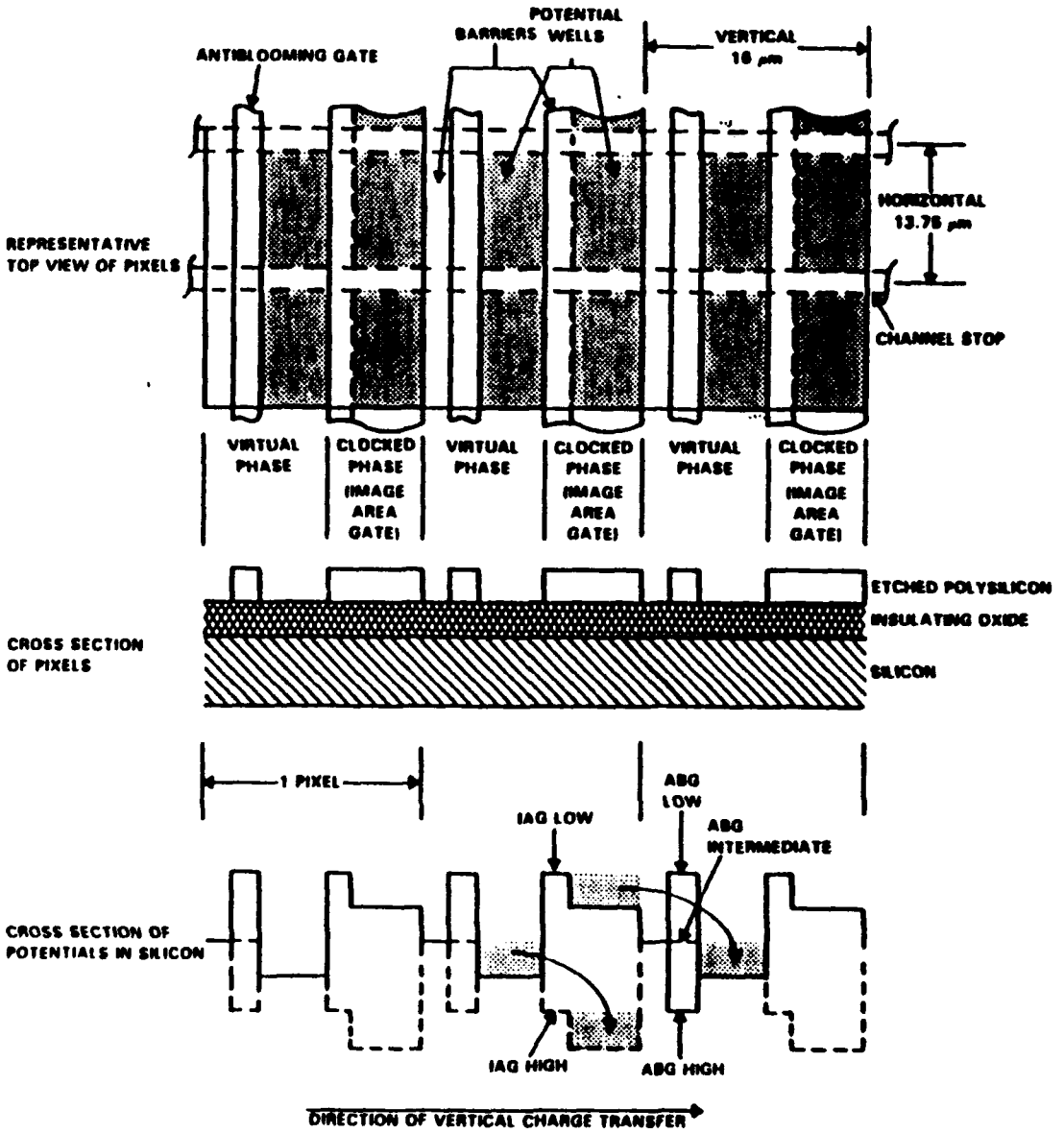


Figure 9. TC211 image area transfer process, uses a virtual phase.

constant time added to the read-out time regardless of the number of pixels used or transfer times. The image area gate (IAG) and serial register gate (SRG) maximum pulse rates determine the sensor read-out rate.

The maximum rate for the IAG parallel row transfer gate is 1.5 MHz. This corresponds to 0.667 microseconds (μs) per row. Clocking the IAG shifts the whole array down by one row. This can be used to clear the CCD. To clear all 165 rows requires only 110 μs. Since an IAG clock pulse clears the serial register, the IAG should not be clocked any faster than the SRG can read-out a row of pixels.

The maximum rate for the SRG single row shift gate is 10 MHz. This corresponds to 0.1 μs per pixel. Figure 8 shows the pixel read-out order. Six dummy pixels will appear before any valid data and the twelve trailing dark pixels can be ignored. At the maximum clock rates, the ideal minimum time required to read an entire row of pixels is 210 pixels X 0.1 μs = 21 μs. The minimum time needed for all 165 rows is 165 rows X 21 μs = 3465 μs.

The total read-out time must include the serial shift time and two parallel array shifts. One set of IAG parallel shifts are needed to move each row into the output shift register and a second set must clear the CCD before the next exposure. Since the output register cannot be read during a row transfer (IAG clocking), using all the pixels, the ideal minimum read-out time is 3.685 ms. Placing the image in the lower left portion of the array can increase the frame rate since only a portion of the array needs to be read-out. After shifting the rows with information to the output register, the IAG clocking can increase to its maximum rate to clear the CCD and to commence another integration. The six leading dummy pixels will always add some time to each serial read-out time, but

the twelve trailing dark pixels could be chopped off. As an example, a 60 X 60 image would take 2 X 60 rows X 0.667 μs + 66 pixels X .1 μs X 60 rows for the serial read-out for a total of 476 μs. There are 66 pixels per row because of the six dummy pixels.

As mentioned earlier, the integration or exposure time adds to the read-out time to produce the total measurement time. The total time could be reduced to only the integration time if the CCD were read-out while integrating the next set of data. This reduces the minimum total time to the longer of these two processes. Frame transfer devices as shown in Figure 6 have a complete extra set of pixels to hold the data for read-out. The interline method is not practical for a star image because the gaps at the transfer pixel rows reduce the exposed area by a factor of two. Each pixel would lose half of its photons. This is unacceptable for this application. Masking off half of the array is a possible solution. It would not work with the TC211 chip. Clocking the IAG parallel row shift register would produce streaking in the upper half exposed pixels while reading the masked pixels.

2. Interface Card

The Spectra Source computer interface card contains the circuitry needed to transfer information from the CCD array to the computer bus. It does an eight bit parallel transfer to an IBM compatible computer. Figure 10 is a block diagram of the components of this card. The CCD sensor was discussed in Section 1. The sample-and-hold (S&H) amplifier chip converts video voltage from

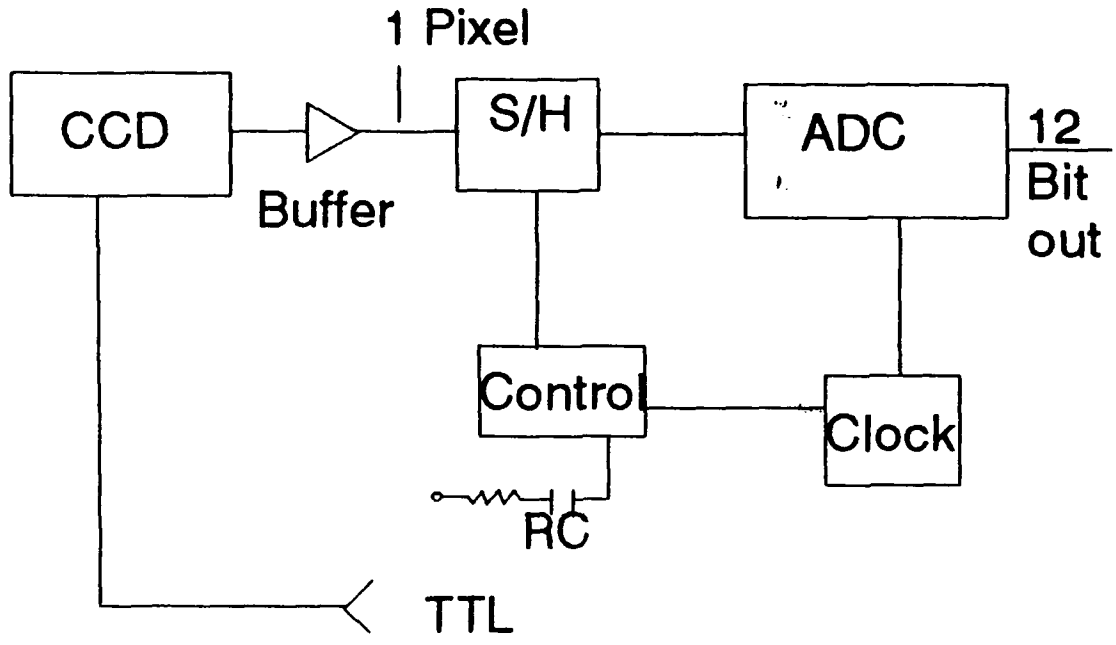


Figure 10. Spectra Sources Lynxx PC+ interface card block diagram.

Section 1. The sample-and-hold (S&H) amplifier chip converts video voltage from one pixel into a voltage pulse. It is a NEC SE/NE5537 [Ref. 20]. The minimum acquisition time is less than 4 μs. It is controlled by pulses from the chip below it, a 74LS123P [Ref. 21], which is a dual retriggerable one-shot. Its pulse switching characteristics are in the nanosecond (ns) range. A 1200 pF capacitor and 12 k resistor determine the present pulse length of 5180 ns. The control chip characteristics did not limit the speed of the S&H chip.

The analog digital converter (ADC) chip converts the voltage to a positive 12 bit integer, which has a maximum value of 4095. This system has a Maxim MX7572 ADC chip [Ref. 22]. Its minimum conversion time is 5 μs. It, as

and-hold and digital conversion time is 9 μs for each pixel. For a 60 X 60 pixel image, the total S&H and ADC time is 32.4 ms.

3. Computers

Two different computers were used to investigate the dependence of the frame rate on computer characteristics. The first was a Compaq 386, 20 megahertz machine. It had a 16 megabyte RAM and a 60 megabyte hard disk. The second was a Dell 486, 33 megahertz machine. It had a 16 megabyte RAM with a 350 megabyte hard disk.

B. SOFTWARE

The Lynxx system came with software programs designed for amateur astronomical observations [Ref. 17]. This thesis used a separate set of data acquisition program modules written for NPS by Spectra Sources [Ref. 18]. It was necessary to modify these modules during this thesis research. The software was written in the C language. Microsoft Quick C 2.5, Borland Turbo C++ 1.0 and Borland C++ 2.0 compilers produced executable code that were tested to compare their relative speeds while collecting and processing data.

IV RESULTS

The first task involved writing the modulation transfer function, centroid jitter and display programs. After it become clear that the Lynxx PC + CCD system was too slow for effective centroid measurements, the second task developed programs to analyze the Lynxx image acquisitions times. The data acquisition functions were timed using different computers and code from different compilers. The manufacturer's maximum speed specifications for both the CCD camera and the interface card were compared to the actual measured times. After analyzing these results, replacements were recommended.

A. PROGRAMS

A modulation transfer function program and a jitter program were written to compute the atmospheric coherence length, r_o. An artificial star program was developed to test the MTF program. The following sections describe these programs.

1. Modulation Transfer Function

This program calculated the coherence length, r_o, from a stellar image intensity distribution. It subtracted the background from the star image, calculated the line spread function, found the centroid and calculated the fast Fourier transform. It then calculated r_o using equation (9). The calculations were

done in the x and y dimensions separately, giving a quality check, since actual image turbulence is nearly always isotropic. With the image centroid data the telescope could operate in an auto track mode, a feature that comes with the Lynxx software package. The program plotted the modulation transfer function as a visual check. The program separates into modules for ease of future revisions. This program is in Appendix B.

2. Artificial Star Program

This program created a perfect exponential star image for the MTF program. This provided a convenient test of the MTF program, since the FFT of an exponential is also an exponential. This program is in Appendix C.

3. Jitter

This program calculated r_o from the centroid jitter. It first removed the background by subtraction. After this, it computed the image centroid jitter using the standard deviation of the image, according to equation (11). This program also calculated the line spread functions and plotted them on the screen so that proper telescope tracking was verified. Appendix D contains this jitter program.

B. TIME MEASUREMENTS

Programs were written or modified to compare frame rates attainable for different subframe sizes. The computer clock measured the time differences. Since the MS-DOS system clock had a 55 ms resolution, many of the times were done for 100 cycles of the function of interest. All the times used were at least

one second long, resulting in an accuracy within 1%. An array of 5 or more trials ensured consistency. Appendix E is an example of the code used for measuring the times.

1. Computer

We found that the type and speed of the computer did not contribute significantly to the camera frame rate. Measurement time included the clearing of the CCD, the integration, and digitization times. With an image size of 60 X 60, there was less than ½ of a frame per second difference between 386 and 486-based computers. Figure 11 shows a comparison of frame rates verses subframe size for the two machines. Including calculations for the line spread functions and backgrounds, the frame rate improvement was about 3½ frames per second for a 60 X 60 image with a 486 computer over that achievable with a 386 machine. This is shown in Figure 12. The mathematical calculations were timed to determine their contributions to the total frame rate. These calculations could have been done separately if they had slowed the frame rate, but they were actually performed in real time.

2. Compiler

It was found that the choice of compiler did not significantly influence the speed of the measurements. Figure 13 shows there was no difference in the frame rate for the code produced by three different compilers to acquire data. Figure 14 shows that after adding the mathematical calculations to the acquisition

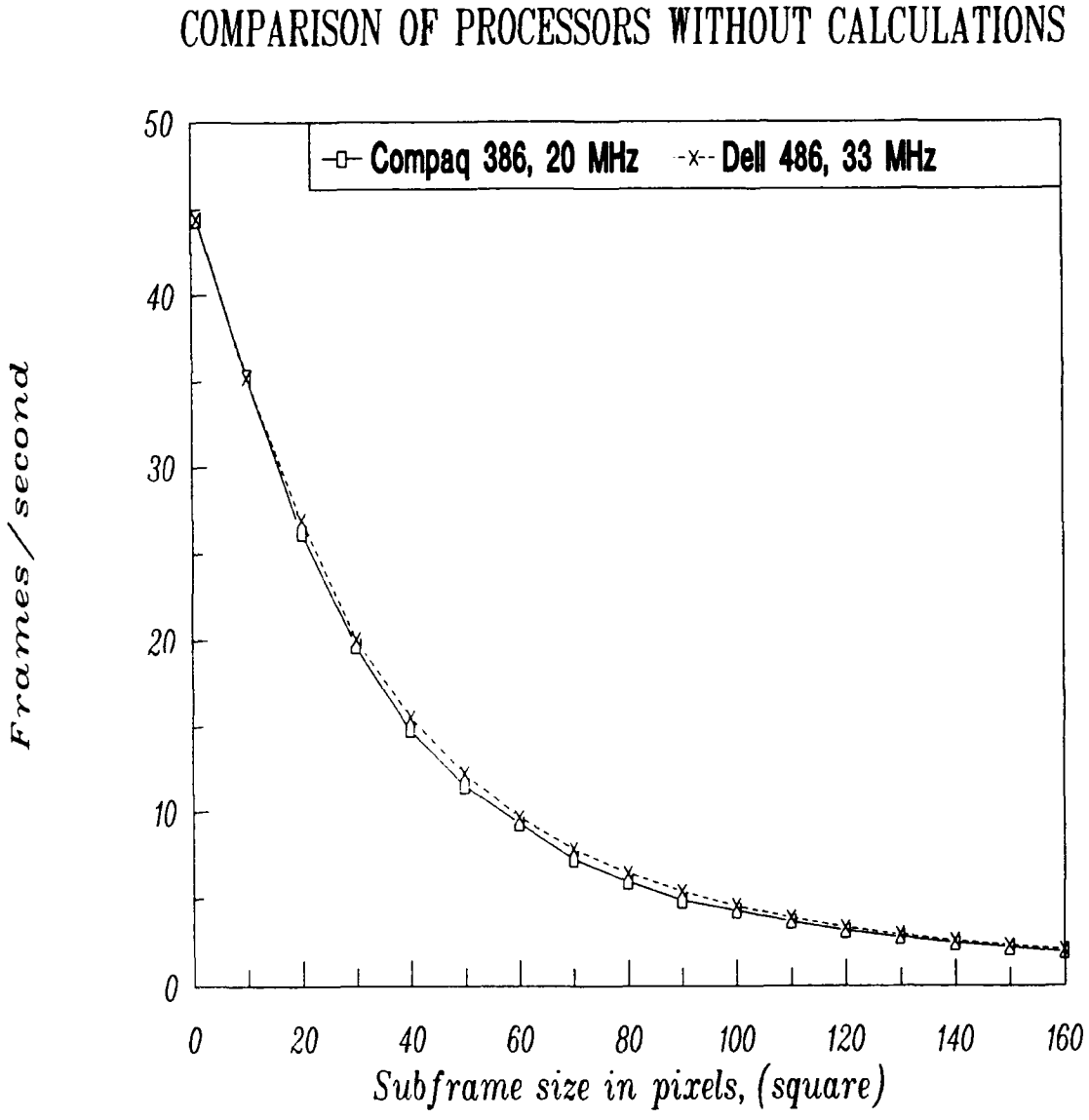


Figure 11. Spectra Sources Lynxx PC+ frame rates using IBM PC 386 and 486 computers. Times include clear, integration and digitization, using code compiled under Turbo C++.

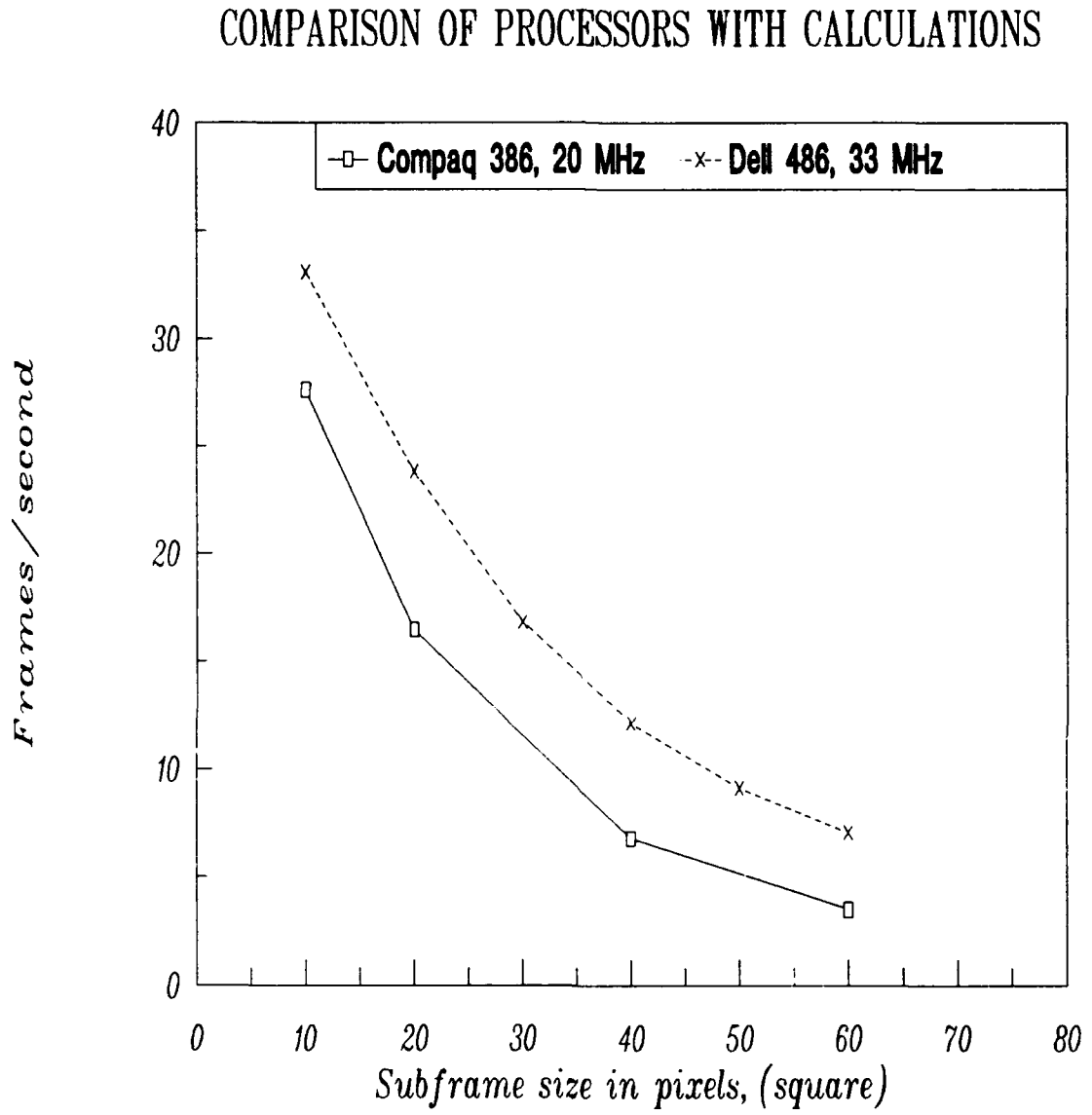


Figure 12. Spectra Sources Lynxx PC+ frame rates using IBM PC 386 and 486 computers. Times include clear, integration, digitization and mathematical calculations, using code compiled under Quick C.

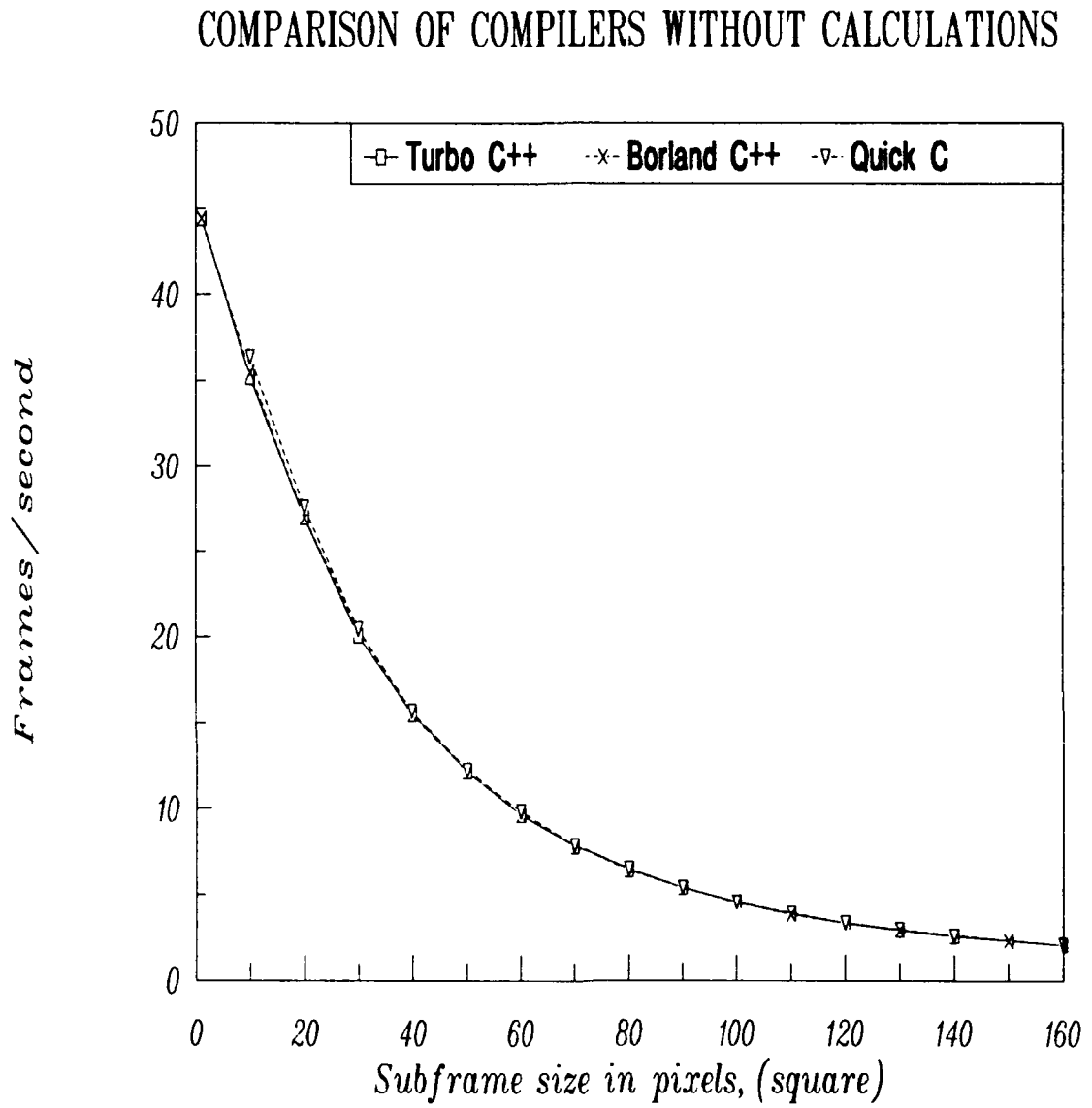


Figure 13. Spectra Sources Lynxx PC+ frame rates using Turbo C++, Borland C++ and Quick C compilers. Includes clear, integration and digitization times, using a Dell 486, 33 MHz PC.

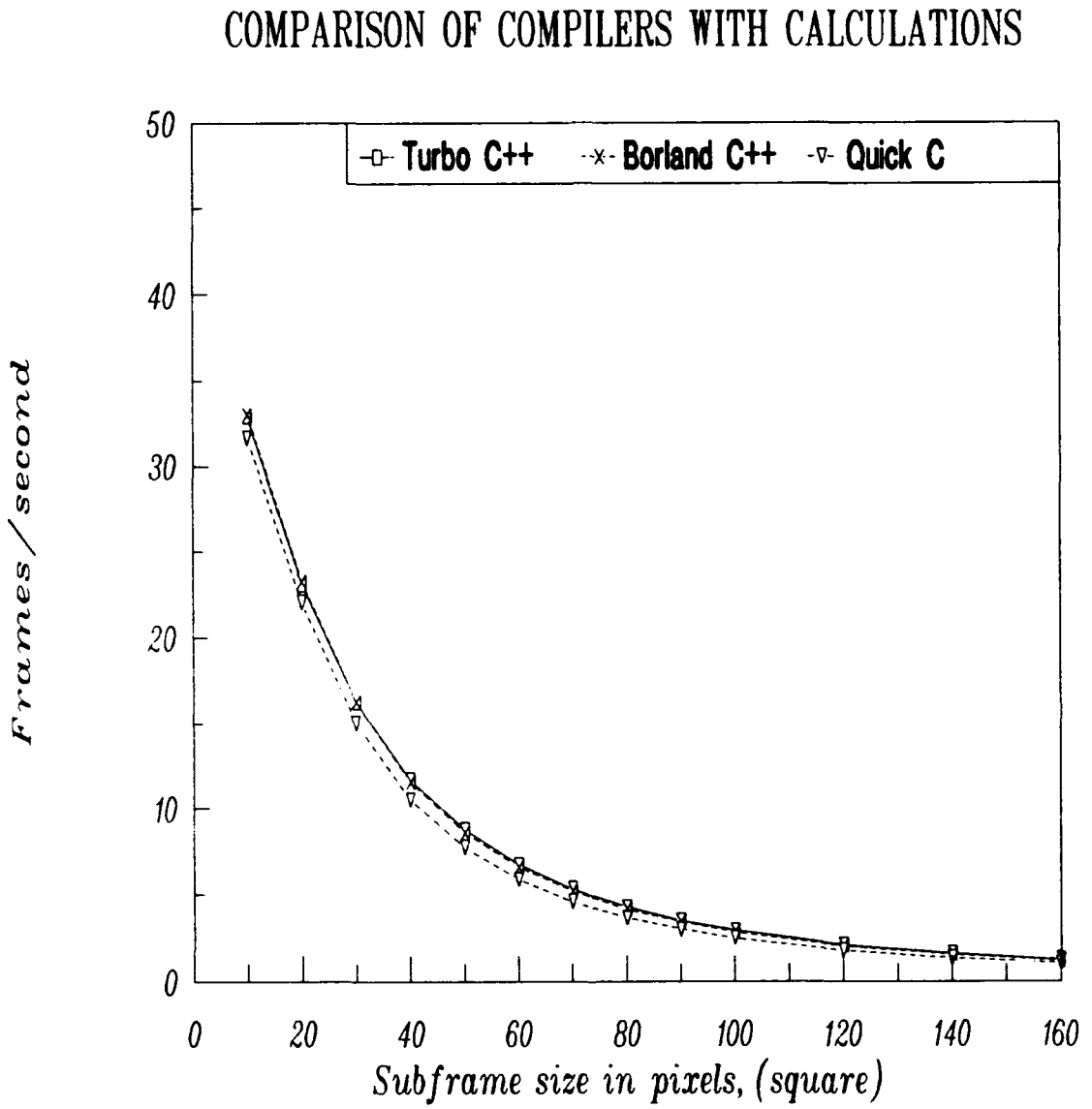


Figure 14. Spectra Sources Lynxx PC+ frame rates using Turbo C++, Borland C++ and Quick C compilers. Includes clear, integration, digitization and mathematical calculation times, using a Dell 486, 33 MHz PC.

time, the code produced by Quick C was about ¾ of a frame/second slower than the code from Borland C++ or Turbo C++ for a 60 X 60 image size.

3. Functions

The times for each measurement were divided into four steps. Table 1 shows these steps.

TABLE 1

Step	Image size	Time
Clear CCD	Must clear entire array	constant 6.45 ms
Integration	Independent of size	used 5.0 ms
Digitization	60 X 60	86.1 ms
Calculations	60 X 60	40.2 ms
Total	60 X 60	137.75 ms

The digitization and calculation times depended on the size of the image, while the clear and integration times were constant for each measurement. This is shown in Figure 15. Clear and digitization times depended on the image sensor and interface card.

C. HARDWARE RATE COMPARISONS

This section compared the maximum design speeds of the chips in both the camera head and interface card to those actually measured. A general time reference was 200 frames/second which corresponded to 5 ms per frame.

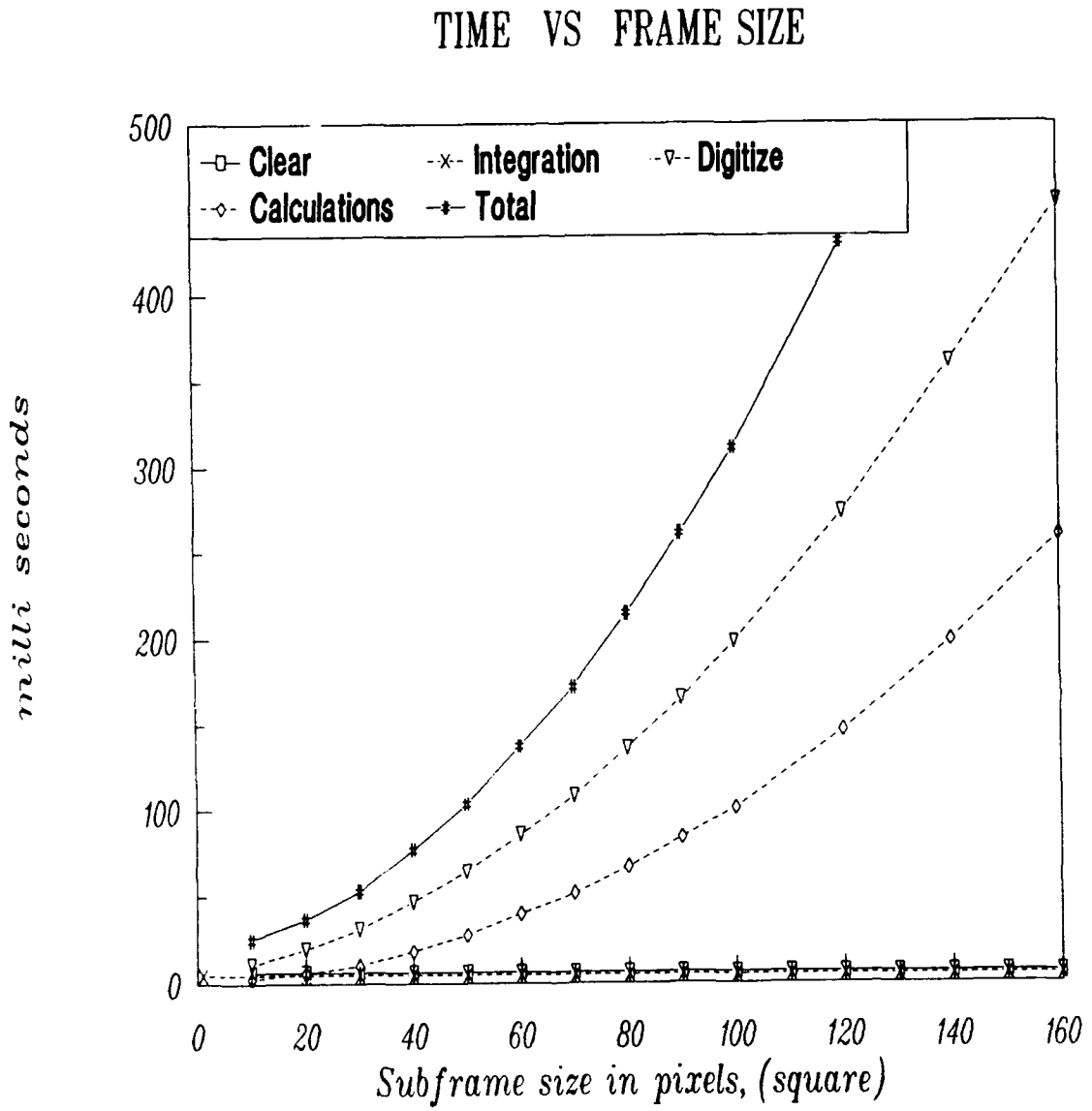


Figure 15. Spectra Sources Lynxx PC+ function times for a Dell 486, 33 MHz PC. Clear and integration times were constant at 6.45 and 5.0 ms respectively. Digitization and calculation times depended on the subframe size.

1. CCD Camera

The measured time for the clear function was 6.45 ms. The present sensor, TC211, is capable of clearing the entire array in 110 μs. All rows need to be cleared even when reading a sub array such as a 60 X 60 image. This function was much slower than the sensor capability. It is not known why the clear times are so slow.

The present sensor could read a 60 X 60 image in 436 μs at maximum design speeds. The measured digitization time, which included the read-out time, sample-and-hold (S&H) and the digital conversion (ADC) times was 86.1 ms. Since this was much larger than 436 μs, the acquisition time depended primarily on the S&H and ADC processes.

2. Interface Card

The interface card controls the S&H and ADC times. Their maximum speed specifications would result in processing a 60 X 60 image in 32.4 ms. This is a significant portion of the observed 86.1 ms digitization time. We therefore concluded that these two chips were the primary contributors to the frame rate and were not operating at their maximum speed specifications. The sensor itself can hypothetically operate at significantly higher speeds.

D. RECOMMENDATIONS

1. CCD Camera

To increase the frame rate requires an image sensor that uses a frame transfer method with separate image and storage arrays. The Texas Instruments TC277, a 735 X 580 pixel CCD image sensor is a proposed replacement sensor [Ref.19]. Figure 16 shows its set-up. The image area is 699 X 288 which is more

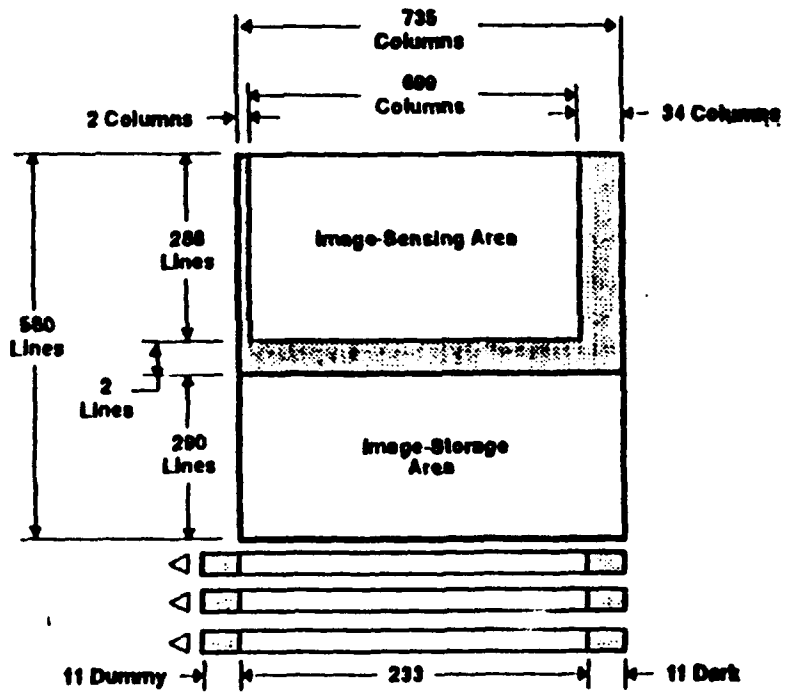


Figure 16. Recommended image sensor CCD, Texas Instruments TC277 block diagram.

than big enough for our applications. The image sensing area and image storage area have different gates. The image area gate (IAG) shifts the rows from the sensing area to the storage area one row per clock pulse. The storage area

gate (SAG) shifts the rows down to the serial register at one row per clock pulse. The three serial register gates (SRG) shift out the pixels in each row.

The storage area gate (SAG) maximum clock rate is 3.34 MHz, which corresponds to 0.299 μs per row. The serial registers gates (SRG) maximum clock rates are 4.46 MHz, which corresponds to 0.224 μs per pixel. For a 60 X 60 image, noting that there are eleven dummy pixels per row, the read-out time is 60 rows X 0.299 μs + 71 pixels X 0.224 μs X 60 rows for a total of 972 μs. This is done in parallel with the integration. Since this is much less than any expected integration time, the integration time and transfer time of data from the sensing area to the storage area become the limiting factors for frame rates.

The image area gate (IAG) maximum clock rate is 3.34 MHz, which corresponds to 0.299 μs per row. The clear time would then be 288 rows X 0.299 μs or 86 μs. This clear time plus the number of rows X 0.299 μs plus the integration time will be the measurement time for each set of data. For an image with 60 rows, the measurement time would be only 86 + 17.9 = 104 μs, plus the integration time. For the TC211, the minimum clear and 60 X 60 image read-out time is 110 + 436 = 546 μs. The TC277 is over 5 times faster.

It should also be noted that the number of pixels in each row of the TC277 can be the maximum available (735) without affecting the speed, due to the parallel read-out and integration times. The noise equivalent signal is only 25 electrons, a 6-fold improvement from the TC211's 150 electrons. Replacing the image sensor would require modifications to the interface board.

Using such an improved sensor, the bulk of the data could be transferred while collecting the next data set. What remains is to get the sampling and digital conversion times fast enough to be completed during the integration time.

2. Interface Card

The first step in the digital conversion is the sample-and-hold (S&H). A proposed replacement chip for this is Analog Devices AD781 [Ref. 23]. It provides a 700 ns conversion time. This is over 5 times faster than the present SE/NE5537 S&H chip's 4 μs. For a 60 X 60 image, the AD781 S&H time reduces to 2.52 ms from 14.4 ms for the present chip. The present control chip, DM74LS123P, for the S&H chip does not need replacement. Changing its capacitor to 55 pF and resistor to 25 k would adjust its pulse width to 700 ns [Ref. 21]. This is the only revision necessary to replace the S&H chip.

A proposed replacement chip for the ADC is a Maxim Max162 [Ref. 22]. This chip is almost identical to the present one, except that its conversion time is 3 μs instead of 5 μs. For a 60 X 60 image, the improved ADC time would be 10.8 ms. The Max162 could replace the Mx7552 with little revision to the rest of the circuit.

There are some faster S&H and ADC chips available if the board were rebuilt. Datel has a S&H chip with a time of only 25 ns and a ADC chip with a conversion time of 350 ns [Ref. 24]. A better option is to have the S&H and ADC processes combined in one chip. Of those available, the Datel ADS118 [Ref. 24]

has a throughput rate of 5.0 Mhz, which corresponds to only 200 ns per pixel. The DM74LS123P control chip will still work by using a 60 pF capacitor with a 5 k resistor [Ref. 21]. Besides providing fast timing, combining these two chips would also simplify the surrounding circuit. For a 60 X 60 image, the S&H and ADC time would be 720 μs for the ADS118.

E. SUMMARY

Table 2 summarizes all of the relevant times. The components are replaced going from left to right across the table. They are in order of the complexity of replacement. Replacing the CCD sensor would be the significant step requiring major modifications to the interface board. There are also parallel times after the CCD sensor is replaced. The processes of clearing, integration and image area gate (IAG) transfer occur parallel to the read-out and digital conversion. The slowest of these times will limit the frame rate. The S&H and ADC times are the limiting factors. For this reason and the necessity to modify the board after replacing the CCD sensor, it is best to build a new system.

TABLE 2

Image size, 60X60	Lynxx time measured	Hardware ideal minimum	New S&H AD781	New ADC Max162	New CCD TC277	New S&H/ADC ADS7818
Integration	5.0 ms	5.0 ms	5.0 ms	5.0 ms	5.0 ms	5.0 ms
Clear	6.5 ms	110 µs	110 µs	110 µs	88 µs	88 µs
Image trans	NA	NA	NA	NA	18 µs	18 µs
Read-out	NA	436 µs	436 µs	436 µs	972 µs	972 µs
S&H	NA	14.4 ms	2.5 ms	2.5 ms	2.5 ms	NA
ADC	NA	18.0 ms	18.0 ms	10.8 ms	10.8 ms	NA
Digitization	86.1 ms	NA	NA	NA	NA	720 µs
Total Time	97.5 ms	37.9 ms	26.0 ms	18.8 ms	5.1 ms[1] 14.0 ms[2]	5.1 ms[1] 1.7 ms[2]
Frames/ second	10.3	26.4	38.5	53.2	[3]196 70	[3]196 592

[1]This time includes the integration, clear and image area gate transfer (IAG).

[2]This time includes the CCD read out, sample and hold (S&H), and digital conversion (ADC).

[3]The lowest of these two rates will limit the system. These two rates reflect the parallel times introduced by the frame transfer CCD.

V CONCLUSIONS AND RECOMMENDATIONS

A CCD camera can measure atmospheric turbulence by measuring the image spread of a point source and by measuring the centroid jitter induced by turbulence. The Spectra Sources Lynxx PC+ system evaluated in this thesis can measure the coherence length using the point source image spread technique, but it is much too slow for jitter measurements. Centroid motion and Greenwood frequency measurements need a sample rate of 200 Hz or more to avoid undersampling the atmospheric dynamics. Achievement of these rates requires a frame transfer CCD, so that parallel image exposure and frame read-out are possible. Data processing can be performed during an exposure if the time needed is sufficiently short, or after digitizing a series of exposures. The Lynxx PC+ system, which provides 12 frames/second, is much too slow for this. Since both the CCD sensor and digital conversion components need replacement, it is best to design and build a new system. The recommended CCD image sensor is a Texas Instruments TC277 [Ref. 19]. The recommended replacement for the sample-and-hold, and analog digital converter chips is a combined sampling analog-to-digital converter chip, Datel ADS118 [Ref. 24]. The atmospheric coherence length programs developed in this thesis should work with a new system.

LIST OF REFERENCES

1. Nelson, D., Atmospheric Turbulence Effects In the Relay Mirror Experiment, M.S. Thesis, Naval Postgraduate School, Monterey, California, June 1988.

2. Fried, D. L., "Optical Resolution Through a Randomly Inhomogeneous Medium for Very Long and Very Short Exposures", Journal of the Optical Society of America, Vol. 56, 1966.

3. Tatarski, V.I., Wave Propagation in a Turbulent Medium, Dover Publications, Inc., New York, 1961.

4. Strohbehn, J.W., "Laser Beam Propagation in the Atmosphere", Topics in Applied Physics, Vol. 25, Springer-Verlag, New York, 1978.

5. Suits, G.H., "Propagation Through Atmospheric Turbulence", The Infrared Handbook, Environmental Research Institute of Michigan, Ann Arbor, Michigan, 1985.

6. Walters, D.L. and Kunkel, K.E., "Atmospheric Modulation Transfer Function for Desert and Mountain Locations: The Atmospheric Effects on r_o", Journal of the Optical Society of America, Vol. 71, April 1981.

7. Hoover, C.R., Investigation of a Single Point Temperature Probe for Measurements of Atmospheric Turbulence, M.S. Thesis, Naval Postgraduate School, Monterey, California, December 1991.

8. Mattingly, T.S., Measurement of Surface Layer Optical Turbulence Above AMOS, M.S. Thesis, Naval Postgraduate School, Monterey, California, December, 1991.

9. Walters, D.L., Favier, D.L. and Hines, J.R., "Vertical Path Atmospheric Modulation Transfer Function Measurements", Journal of the Optical Society of America, Vol. 69, June 1979.

10. Hogge, C.B., "Propagation of High Energy Laser Beams in the Atmosphere", Topics in Applied Physics, Volume 38, Springer-Verlag, New York, 1980.

11. Stevens, K.B., Remote Measurement of the Atmospheric Isoplanatic Angle and Determination of Refractive Turbulence Profiles by Direct Inversion of the Scintillation Amplitude Covariance Function with Tekhonov Regularization, Ph.D. Dissertation, Naval Postgraduate School, Monterey, California, December, 1985.

12. Greenwood, D.P., "Bandwidth Specification for Adaptive Optics Systems", Journal of the Optical Society of America, Vol. 67, pp. 390-393, 1977.

13. Merrill, B.R., Measuring the Greenwood Frequency: Theoretical Considerations, Phillips Laboratory, May 1991.

14. Wilson, J. and Hawkes, J.F.B., Optoelectronics: An Introduction, Prentice Hall International (UK) Ltd., London, England, 1989.

15. Hynecek, J., "Virtual Phase Technology: A New Approach to Fabrication of Large-Area CCD's, IEEE Transactions on Electron Devices, Vol. ED-28, No. 5, May, 1981.

16. Buil, C., CCD Astronomy, Willmann-Bell, Inc., Richmond, Virginia, 1991.

17. SpectraSource Instruments, Lynxx/Lynxx Plus CCD Digital Imaging System User's Manual, SpectraSource Instruments, Agoura Hills, California, 1990.

18. SpectraSource Instruments, PC-Lynxx Runtime Library Manual, SpectraSource Instruments, Westlake Village, California, 1991.

19. Area Array Image Sensor Products Data Manual 1992, Texas Instruments Incorporated, Dallas, Texas, 1991.

20. IC Master, Volume II, Hearst Business Communications, Inc., Santa Clara, California, 1986.

21. LS/S/TTL Logic Databook, National Semiconductor Corporation, Santa Clara, California, 1987.

22. Maxim 1992 New Releases Data Book, Maxim Integrated Products, Inc., Sunnyvale, California, 1991.

23. 1992 Data Converter Reference Manual, Volume II, Analog Devices, Inc., Norwood, Massachusetts, 1992.

24. Datel Data Conversion Components Databook, Volume 1, Datel, Inc., Mansfield, Massachusetts, 1991.

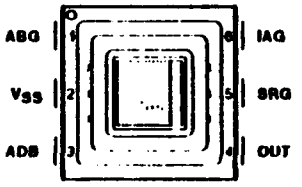
APPENDIX A TEXAS INSTRUMENTS TC211 CCD IMAGE SENSOR

TC211
192 × 165-PIXEL CCD IMAGE SENSOR

D3271, JANUARY 1990

- Full-Frame Operation
- Antiblooming Capability
- Single-Phase Clocking for Horizontal and Vertical Transfers
- Fast Clear Capability
- Dynamic Range . . . 60 dB Typical
- High Blue Response
- High Photoresponse Uniformity
- Solid-State Reliability With No Image Burn-In, Residual Imaging, Image Distortion, Image Lag, or Microphonics
- 6-Pin Dual-In-Line Ceramic Package
- Square Image Area:
  - 2640 μm by 2640 μm
  - 192 Pixels (H) by 165 Pixels (V)
  - Each Pixel 13.75 μm (H) by 16 μm (V)

DUAL-IN-LINE PACKAGE
(TOP VIEW)



description

The TC211 is a full frame charge coupled device (CCD) image sensor designed specifically for industrial applications requiring ruggedness and small size. The image sensing area is configured into 165 horizontal lines each containing 192 pixels. Twelve additional pixels are provided at the end of each line to establish a dark reference and line clamp. The antiblooming feature is activated by supplying clock pulses to the antiblooming gate, an integral part of each image sensing element. The charge is converted to signal voltage at 4 μV per electron by a high performance structure with built in automatic reset and a voltage reference generator. The signal is further buffered by a low noise two stage source follower amplifier to provide high output drive capability.

The TC211 is supplied in a 6 pin dual-in line ceramic package approximately 7,5 mm (0.3 in.) square and is characterized for operation from −10 °C to 45 °C. The glass window can be cleaned using any standard method for cleaning optical assemblies or by wiping the surface with a cotton swab soaked in alcohol.

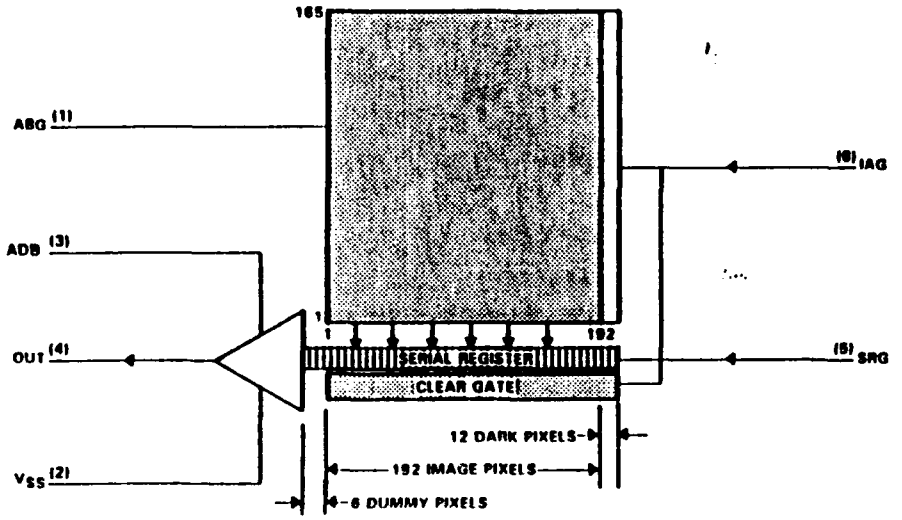
This MOS device contains limited built in gate protection. During storage or handling, the device leads should be shorted together or the device should be placed in conductive foam. In a circuit, unused inputs should always be connected to V_{SS}. Under no circumstances should pin voltages exceed absolute maximum ratings. Avoid shorting OUT to V_{SS} during operation to prevent damage to the amplifier. The device can also be damaged if the output terminals are reverse biased and an excessive current is allowed to flow. Specific guidelines for handling devices of this type are contained in the publication *Guidelines for Handling Electrostatic-Discharge-Sensitive (ESDS) Devices and Assemblies* available from Texas Instruments.

TEXAS INSTRUMENTS
POST OFFICE BOX 655303 • DALLAS, TEXAS 75265

TC211
192 × 165-PIXEL CCD IMAGE SENSOR

D3271, JANUARY 1990

functional block diagram



pin functional description

PIN NAME	PIN NUMBER	DESCRIPTION	INPUT/OUTPUT (I/O)
ABG	1	Antiblooming gate	I
V_{SS}	2	Amplifier ground	
ADB	3	Supply voltage for amplifier drain bias	I
OUT	4	Output signal	O
SRG	5	Serial register gate	I
IAG	6	Image area gate storage	I

functional description

The image sensing area consists of 165 horizontal image lines each containing 192 photosensitive elements (pixels). Each pixel is 13.75 µm (horizontal) by 16.00 µm (vertical). As light enters the silicon in the image sensing area, free electrons are generated and collected in potential wells (see Figure 1). During this time, the antiblooming gate is activated by applying a burst of pulses every horizontal blanking interval. This prevents blooming caused by the spilling of charge from overexposed elements into neighboring elements. The antiblooming gate is typically held at a mid level voltage during readout. The quantity of charge collected in each pixel is a linear function of the incident light and the exposure time. After exposure and under dark conditions, the charge packets are transferred from the image area to the serial register at the rate of one image line per each clock pulse applied to the image area gate. Once an image line has been transferred into the serial register, the serial register gate can be clocked until all of the charge packets are moved out of the serial register to the charge detection node at the amplifier input.

There are 12 dark pixels to the right of the 192 image pixels on each image line. These dark pixels are shielded from incident light and the signal derived from them can be used to generate a dark reference for restoration of the video black level on the next image line.

Each clock pulse applied to the image area gate causes an automatic fast clear of the 192 image pixels and 12 dark pixels of the serial register before the next image line is transferred into the serial register. (Note that the six dummy pixels at the front of the serial register, which are used to transport charge packets from the serial register to the amplifier input, are not cleared by the image area gate clock.) The automatic fast clear feature can be used to initialize the image area by transferring all 165 image lines to the serial register gate under dark conditions without clocking the serial register gate.

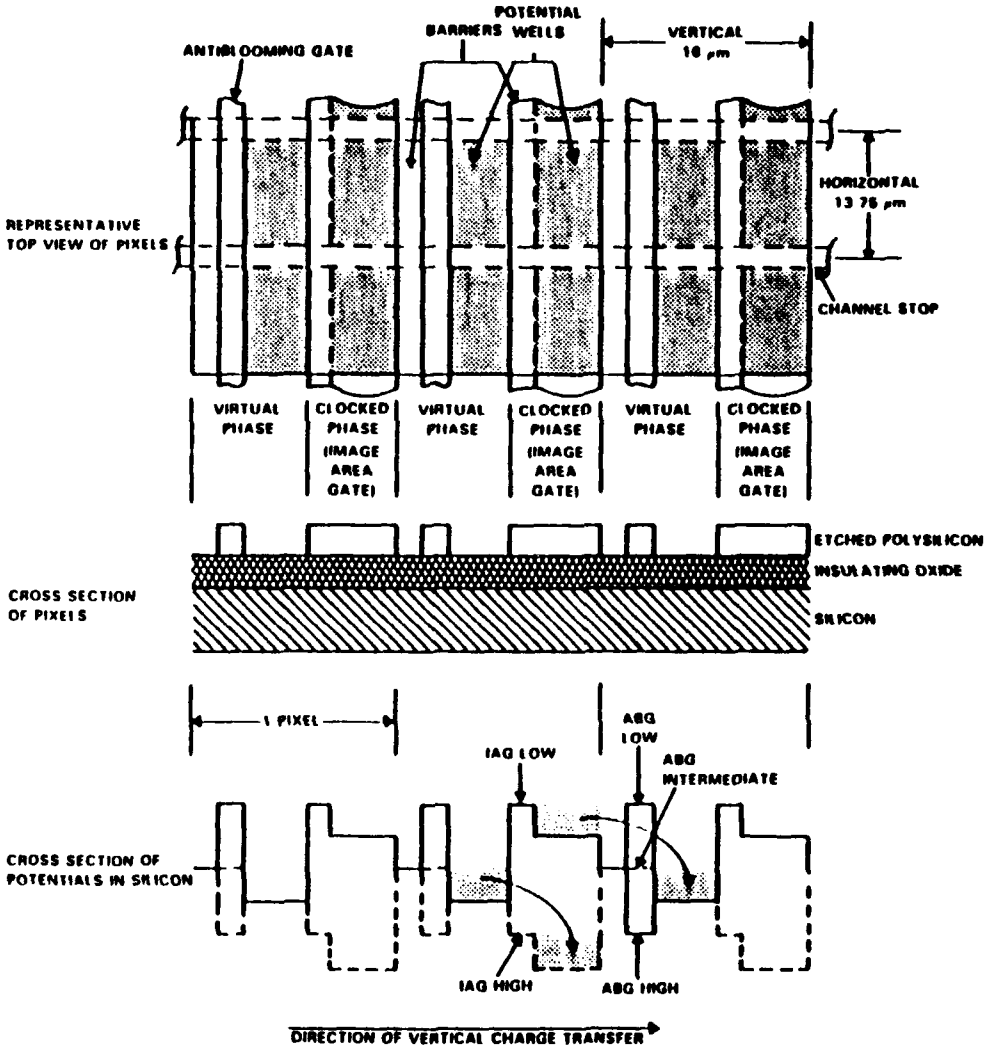


Figure 1. Charge Accumulation and Transfer Process

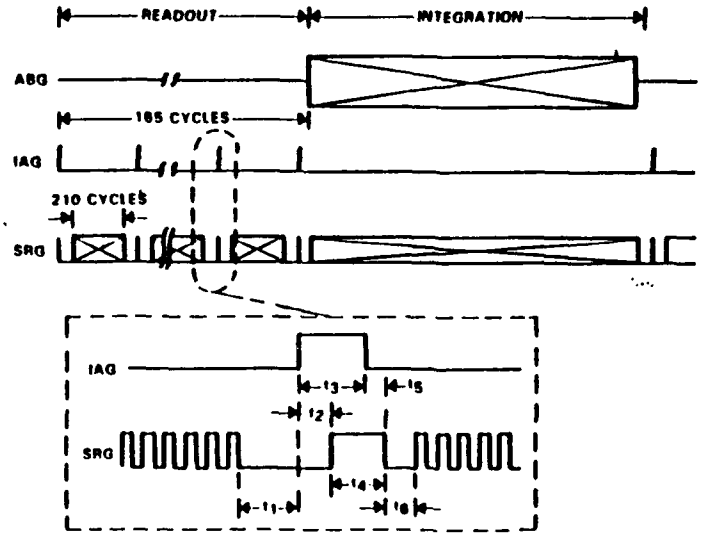


Figure 2. Timing Diagram, Noninterlace Mode

absolute maximum ratings over operating free-air temperature range (unless otherwise noted)

Supply voltage range (ADB) (see Note 1) ..... 0 V to 15 V
Clock voltage range for IAG, SRG, ABG terminals ..... −15 V to 5 V
Operating free-air temperature range ..... −30°C to 85°C
Storage temperature range ..... −30°C to 85°C
Lead temperature 1,6 mm (1/16 inch) from case for 10 seconds ..... 260°C

NOTE 1: All voltage values are with respect to the substrate terminal

recommended operating conditions

			MIN	NOM	MAX	UNIT
Supply voltage, A_{DB}			11	12	13	V
Substrate bias voltage				0		V
Clock voltage†	Image area gate	High level	0	1	2	V
		Intermediate level‡	−10	−5	2	
		Low level	−10	−9.5	−9.5	
	Serial register gate	High level	1.5	2	2.5	
		Low level	−10	−9.5	−9.5	
	Antiblooming gate	High level	2.5	3.5	4	
		Intermediate level‡	−3	−2.5	−2	
		Low level	−8	−7	−6	
Clock rate	IAG				1.5	MHz
	SRG				10	
	ABG				2	
t_1	Time interval, SRG↓ to IAG↑		70			ns
t_2	Time interval, IAG↑ to SRG transfer pulse ↑		0			ns
t_3	Pulse duration, IAG high		350			ns
t_4	Pulse duration, SRG transfer pulse high		350			ns
t_5	Time interval, IAG↓ to SRG transfer pulse ↓		350			ns
t_6	Time interval, SRG transfer pulse ↓ to SRG clock pulse ↑		70			ns
Capacitive load	OUT pin			12		pF
Operating free-air temperature, T_A			−10		45	°C

† The algebraic convention, in which the least positive (most negative) value is designated minimum, is used in this data sheet for clock voltage levels

‡ Adjustment is required for optimal performance

electrical characteristics over recommended operating range of supply voltage, T_A = –10°C to 40°C

PARAMETER		MIN	TYP†	MAX	UNIT
Dynamic range (see Note 2)	Antiblooming disabled (see Note 3)	60			dB
	Antiblooming enabled	57			
Charge conversion factor			4.0		µV/e
Charge transfer efficiency (see Note 4)		0.99990	0.99998		
Signal response delay time, τ (see Note 5 and Figure 5)			25		ns
Gamma (see Note 6)		0.97	0.98	0.99	
Output resistance			700	800	Ω
Noise voltage	1/f noise (5 kHz)		370		nV/√Hz
	Random noise, f = 100 kHz		70		
Noise equivalent signal			150		electrons
Rejection ratio at 7.16 MHz	From ADB to output (see Note 7)		19		dB
	From SRG to output (see Note 8)		37		
Supply current, I_{DD}			5	10	mA
Capacitance	Image area gate		1600		pF
	Serial register gate		25		
	Antiblooming gate		780		

† All typical values are at T_A = 25°C

NOTES: 2 Dynamic range is –20 times the logarithm of the mean noise signal divided by the saturation output signal

3 For this test, the antiblooming gate must be biased at the intermediate level

4 Charge transfer efficiency is one minus the charge loss per transfer in the output register. The test is performed in the dark using an electrical input signal.

5 Signal response delay time is the time between the falling edge of the SRG clock pulse and the output signal valid state.

6 Gamma (γ) is the value of the exponent in the equation below for two points on the linear portion of the transfer function curve (this value represents points near saturation):

$$\left(\frac{\text{Exposure (2)}}{\text{Exposure (1)}}\right)^{\gamma} = \left(\frac{\text{Output signal (2)}}{\text{Output signal (1)}}\right)$$

7 ADB rejection ratio is –20 times the logarithm of the ac amplitude at the output divided by the ac amplitude at ADB

8 SRG rejection ratio is –20 times the logarithm of the ac amplitude at the output divided by the ac amplitude at SRG

optical characteristics, T_A = 25°C (unless otherwise noted)

PARAMETER			MIN	TYP	MAX	UNIT
Sensitivity (see Note 9)	No IR filter	Measured at V_{IJ} (see Note 10)		280		mV/lx
	With IR filter			33		
Saturation signal (see Note 11)		Antiblooming disabled	400	600		mV
		Antiblooming enabled	350	450		
Blooming overload ratio (see Note 12)		Strobe		6		
		Shuttered light		100		
Output signal nonuniformity (1/2 saturation) (see Note 13)				10%	20%	
Image area well capacity				150×10^3		electrons
Dark current		T_A = 21°C		0.027		nA/cm^2
Dark signal (see Note 14)				10	15	mV
Dark signal nonuniformity for entire field (see Note 15)				4	15	mV
Modulation transfer function		Horizontal		50%		
		Vertical		70%		

NOTES: 9 Sensitivity is measured at an integration time of 16.667 ms and a source temperature of 2856 K. A CM 500 filter is used.
10 V_{IJ} is the output voltage that represents the threshold of operation of antiblooming. V_{IJ} ~ 1/2 saturation signal.
11 Saturation is the condition in which further increase in exposure does not lead to further increase in output signal.
12 Blooming overload ratio is the ratio of blooming exposure to saturation exposure.
13 Output signal nonuniformity is the ratio of the maximum pixel to pixel difference in output signal to the mean output signal for exposure adjusted to give 1/2 the saturation output signal.
14 Dark signal level is measured from the dummy pixels.
15 Dark signal nonuniformity is the maximum pixel to pixel difference in a dark condition.

PARAMETER MEASUREMENT INFORMATION

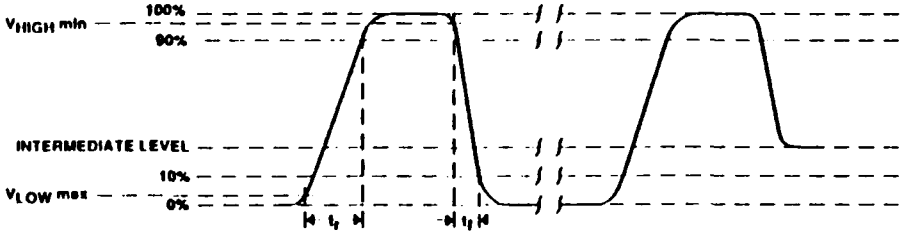
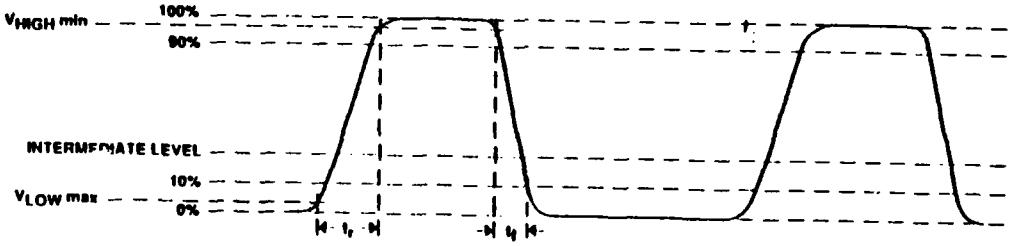


Figure 3. Typical Clock Waveform for IAG and ABG

SLEW RATE BETWEEN 10% and 90% = 70 to 120 V/µs. t_r = 150 ns, t_f = 90 ns

PARAMETER MEASUREMENT INFORMATION



SLEW RATE BETWEEN 10% and 90% = 300 V/μs, $t_r = t_f = 15$ ns.

Figure 4. Typical Clock Waveform for SRG

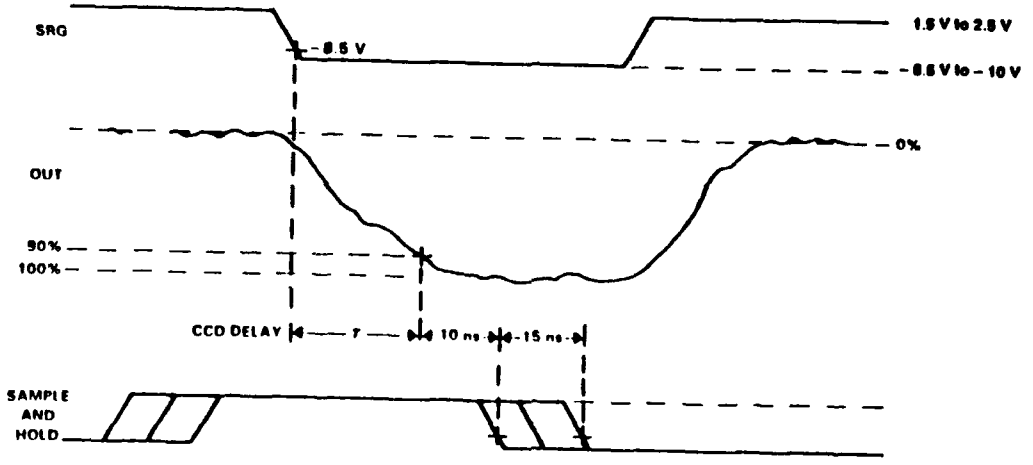


Figure 5. SRG and CCD Output Waveforms

TYPICAL CHARACTERISTICS

HORIZONTAL MODULATION TRANSFER FUNCTION (BARS PARALLEL TO SERIAL REGISTER)

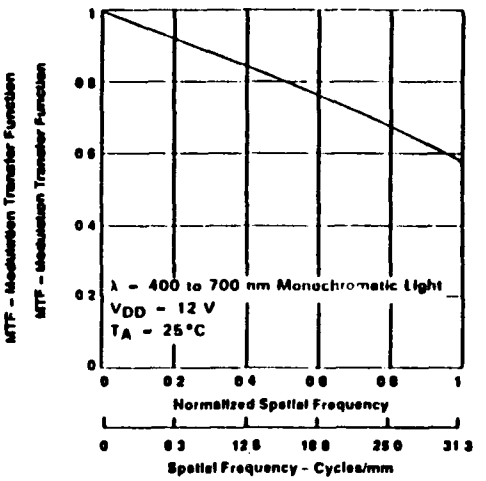


Figure 6

VERTICAL MODULATION TRANSFER FUNCTION (BARS PERPENDICULAR TO SERIAL REGISTER)

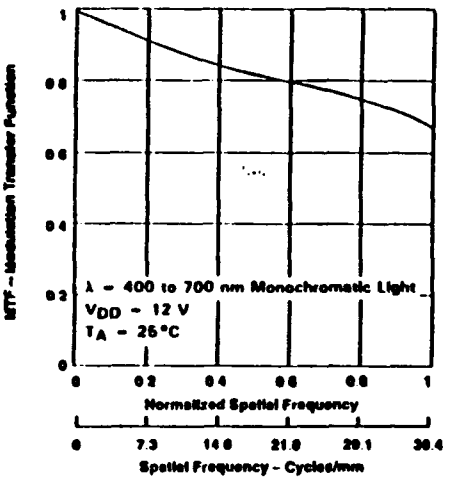


Figure 7

NOISE SPECTRUM OF OUTPUT AMPLIFIER

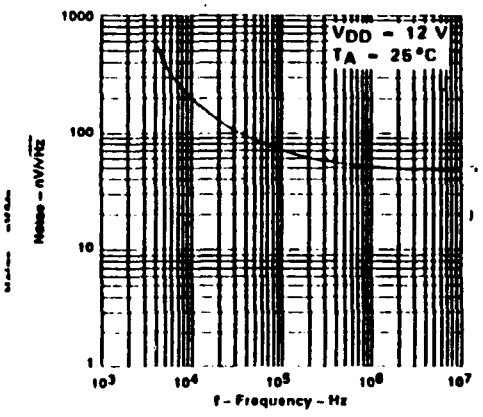


Figure 8

CCD SPECTRAL RESPONSIVITY

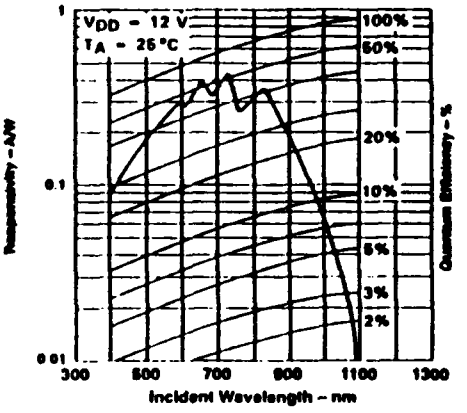
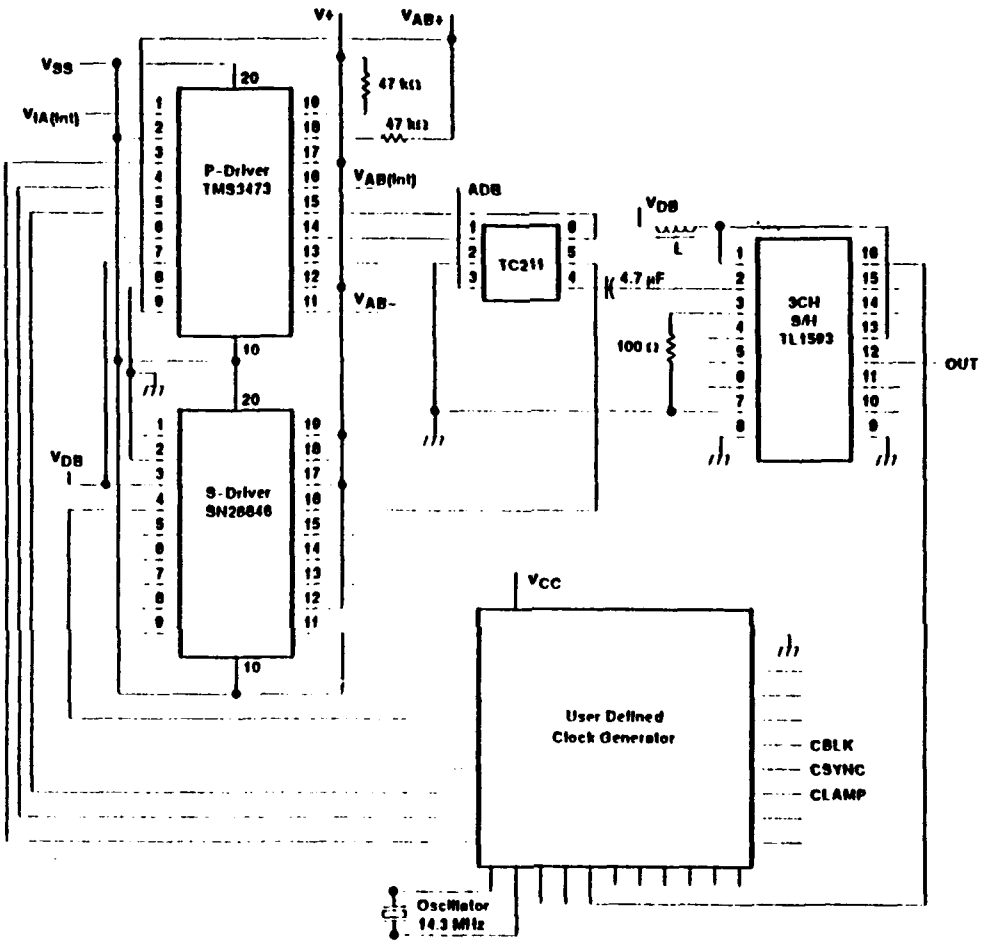


Figure 9

TYPICAL APPLICATION DATA

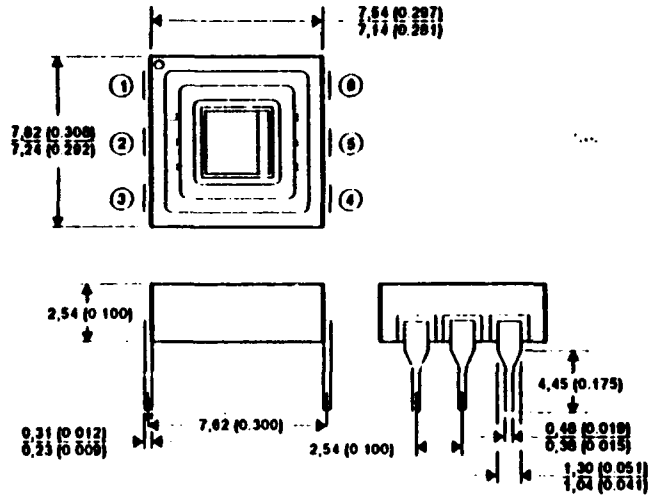


SUPPORT CIRCUITS			
DEVICE	PACKAGE	APPLICATION	FUNCTION
SN28846	20 pin NF	Serial driver	Driver for SRG
TMS3473	20 pin NF	Parallel driver	Driver for IAG, ABG
TL1593	16 pin D	Sample and hold	Three-channel sample-and-hold IC

Figure 10. Typical Application Circuit Diagram

MECHANICAL DATA

The package for the TC211 consists of a ceramic base, glass window, and a 8 lead frame. The glass window is sealed to the package by an epoxy adhesive. The package leads are configured in a dual in line organization and fit into mounting holes with 2,54 mm (0 1 inch) center-to center spacings.



NOTES A Dimensions are in millimeters and parenthetically in inches Single dimensions are nominal
B The center of the package and the center of the image area are not coincident
C The distance from the top of the glass to the image sensor surface is typically 1 mm (0 040 inch) The glass is typically 0 020 inch thick and has an index of refraction of 1 52

APPENDIX B MTF PROGRAM

```
/*MTF.c          find ro and f from the lsf using FFT to the MTF*******/
#define TITLE        "Measure ro, plot the MTF"

#define AUTHOR                  "By W. J. Rall"
          /***********Quick C*********************************/
/*#include <alloc.h>  */
#include <lynxx.h>

#define True 1
#define False 0
#define false 0

#include <stdio.h>
#include <stdlib.h>
#include <graph.h>
#include <math.h>

#include <sys\types.h>

/*int frame_ptr[FRAME_BYTES/2];*/
int size=60, xstart=0, ystart=0;
float lsfx[165],lsfmax=0.0,lsfy[192];

int sign=1, m=8, mm=2, mm2;/*mm is # of points in FFT, mm2=mm/2***/
float mtfx[1024],mtfmax=0.0,mtfy[1024],  re[1024],im[1024],px[512],py[512];
float rocal = 1.0,rox,roy;

void init();
      void subexpose(long tm,int shutter);
        void expose(long tm);
        void long_delay(long tm) ;
        void calc_lsf(unsigned int far *sub_frame);
        void background();
        void fft(int m, int sign, float re[], float im[]);
        void calc_ro();
        void init_mtf();
        void plot_mtf();
        /*to see lsf's, use these two functions instead of mtf ones*/
```

```
        void init_graph();
        void plot_graph();

/*-----------------------main routine----------------------*/
void main()
{
        int i;
        for (i=0; i<m-1; i++) mm = 2*mm;
    init_lynxx();
        init();
    cooler_off();
    _setvideomode( _TEXTC80);
}
/*-------------------Initialize input info-----------------*/
void init()
{
    int cool = 0,shutter=0;
    long tm;
    printf("Expose time in ms.:");
    scanf("%ld",&tm);
    printf("Cooler 0=OFF 1=ON:");
    scanf("%d",&cool);
    if (cool)
            cooler_on();
    printf("Use shutter ? (1=Yes/0=No)");
    scanf("%d", &shutter);
        printf("Subframe size? ");
        scanf("%d", &size);

 /* printf("\nHit any key to quit:");*/

 subexpose(tm,shutter);

}
/*-----------------------subExpose CCD---------------------*/
void subexpose(long tm,int shutter)
{
    unsigned int *sub_frame;
    int sub_frame_bytes;

        float px0,py0;
        int i,j,x,y;
        for (i=0; i<m-1; i++) mm = 2*mm;
```

```
    mm2=(int)((float)(mm)/2.0);

sub_frame_bytes = size*size * sizeof(unsigned int);
sub_frame = (unsigned int*)malloc(sub_frame_bytes);

if (!shutter) open_shutter();

    while ( !kbhit() )
    {
        if (shutter)
        {
            expose(tm);
        }
        else
        {
            clrccd(); /* if not using shutter    *****************/
            long_delay(tm);
        }
        x_param = (unsigned char) xstart;
        y_param = (unsigned char) ystart;
        size_param = (unsigned char) size;
        ptr_param = sub_frame;
        digitize_sub_frame();
        calc_lsf(sub_frame);
        background();

        for (x=xstart+size-1; x>=xstart; x--) re[x] = lsfx[x];
        fft(m, sign, re, im);
        for (x=0; x<mm; x++)  mtfx[x] = re[x];

        /*find the x  power spectrum***************/
        px0 = re[0]*re[0] + im[0]*im[0];
        for (j=0; j<mm2; j++)
        {
          if(mtfx[j] != 0.0 ) if(px0 != 0.0)
              px[j]=sqrt((re[j]*re[j]+im[j]*im[j])/px0)/mtfx[j];
        }
        /*reinitialize the FFT arguements before finding  the Y  FFT*/
        for (i=0; i<1024; i++)
        {
        re[i] = 0.0;
        im[i] = 0.0;
        }
```

```
            for (y=ystart+size-1; y>=ystart; y--) re[y] = lsfy[y];
            fft(m, sign, re, im);
            for (y=0; y<mm; y++)  mtfy[y] = re[y];

            /*find the y  power spectrum***************/
            py0 = re[0]*re[0] + im[0]*im[0];
            for (j=0; j<mm2; j++)
            {
              if(mtfy[j] != 0.0) if(py0 != 0.0)
                  py[j]=sqrt((re[j]*re[j]+im[j]*im[j])/py0)/mtfy[j];
            }
            calc_ro();
            init_mtf();
            plot_mtf();
            /*init_graph();
            plot_graph();    *********    use these to see lsf's       */
        }
free(sub_frame);
close_shutter();
}
/*-----------------------Expose  CCD-----------------------*/
void expose(long tm)
{
    clrccd();
    open_shutter();
    long_delay(tm);
    close_shutter();
}
/*-----------------------long  delay-----------------------*/
void long_delay(long tm)
{
    delay_param = 1000;
    while (tm > 1000)
    {
            delay_ms();
            tm = tm -1000;
    }
    delay_param = (unsigned short)tm;
    delay_ms();
}
 /*--------------------calculate  the line spread functions----------*/
void calc_lsf(unsigned int far *sub_frame)
{
```

```
    unsigned int pixel;
    int x,y;
    float lsfxmax=0.0,lsfymax=0.0;
    for (x=0; x<165; x++) lsfx[x]=0.0;
    for (y=0; y<165; y++) lsfy[y]=0.0;
    for (x=size-1+xstart; x>=xstart; x--)
    {
            for (y=size-1+ystart; y>=ystart; y--)
            {
                pixel=*(sub_frame++);
                lsfx[x] = lsfx[x] + (float)pixel;
                lsfy[y] = lsfy[y] + (float)pixel;
                if(lsfx[x] > lsfxmax) lsfxmax=lsfx[x];
                if(lsfy[y] > lsfymax) lsfymax=lsfy[y];
            }
      }
      lsfmax=lsfxmax;
      if(lsfxmax < lsfymax) lsfmax = lsfymax;
}

/*-------------Subtract off the background----------------*/
void background()
{
    int x,y,edge;
    float sumx=0.0,sumy=0.0,ave_x1,ave_x2,ave_y1,ave_y2,aveback;
    edge=(int)(size/10.0); /*this uses 20% to find background*/
    /*-----Find the average background----------*/
      for (x=xstart; x<xstart+edge; x++) sumx=sumx+lsfx[x];
      ave_x1=sumx/edge;  sumx=0.0;
      for (x=xstart+size-edge-1; x<xstart+size; x++) sumx=sumx+lsfx[x];
      ave_x2=sumx/edge;
      for (y=ystart; y<ystart+edge; y++) sumy=sumy+lsfy[y];
      ave_y1=sumy/edge;  sumy=0.0;
      for (y=ystart+size-edge-1; y<ystart+size; y++) sumy=sumy+lsfy[y];
      ave_y2=sumy/edge;
      aveback =((ave_x1+ave_x2)+(ave_y1+ave_y2))/4.0;
    /*----subtract off the background-------*/
      lsfmax = lsfmax - aveback;
      for (x=xstart+size-1; x>=xstart; x--)
            lsfx[x]=lsfx[x]-((x-xstart)*(ave_x2-ave_x1)/(size-edge)+ave_x1);
      for (y=ystart+size-1; y>=ystart; y--)
            lsfy[y]=lsfy[y]-((y-ystart)*(ave_y2-ave_y1)/(size-edge)+ave_y1);
/*  printf("\nThe average background is %f\n",aveback);*/
```

```
}

/*********************Find the FFT*******************************/
void fft(int m, int sign, float re[], float im[])
{
      int n, j, j1, ndiv2, n1, n2, k, i, ip, npts;
      int le, le0, le1, l;
      float pts, t, ure, uim, wre, wim, tre, tim;
      double ang, pi;

      pi = 4.0*atan(1.0);
      n = (int)(pow(2.0, (double)m)+1.0e-10);
      j = 1;
      j1 = 0;
      n1 = n - 1;
      n2= n - 2;
      ndiv2 = n/(int)2;

      for(i=0; i<=n2; i++)
      {
            if(i < j1)
            {
                  t = re[j1];     /* you might do better here with pointer operations
*/
                  re[j1] = re[i];
                  re[i] = t;
                  t = im[j1];
                  im[j1] = im[i];
                  im[i] = t;
            }
            k = ndiv2;
            while(k < j)
            {
                  j -= k;
                  k /= 2;
            }
            j += k;
            j1=j-1;
      }

      le = 1;
      for(l=1; l<=m; l++)
      {
```

```
            ure = (float)1;
            uim = (float)0;
            ang = pi/(double)le;
            wre = (float)cos(ang);
            wim = (float)sin(ang);
            le0=le;
            le1=le-1;
            le +=le;

            for(j=0; j<=le1; j++)
            {
                for(i=j; i<=n1; i+=le)
                {
                    ip = i + le0;
                    tre = re[ip]*ure - im[ip]*uim;
                    tim = re[ip]*uim + im[ip]*ure;
                    re[ip] = re[i] - tre;
                    im[ip] = im[i] - tim;
                    re[i] += tre;
                    im[i] += tim;
                }
                t = ure*wre - uim*wim;
                uim = ure*wim + uim*wre;
                ure = t;
            }
        }
        if(sign<0)
            {
                pts = 1/(float)n;
                    for(i=0; i<n; i++)
                    {
                    re[i] *= pts;
                    im[i] *= pts;
                    }
            }
}
/*******************calculate ro*********************************/
void calc_ro()
{
        int j;
        float z,zold,pj,slope;

        if (px[1] > 1.0) if (px[1] <= 0.0) printf("\nX DATA POINT BAD");
```

```
        else
        {
            z = log (-log(px[1]));
            for (j=2; j<mm2; j++)
            {
                pj = px[j];
                zold = z;
                if (pj<1.0)
                {
                    z = log(-log(pj));
                    slope = (z-zold)/log((float)(j)/(float)(j-1));
                    if (z > 0.0) goto ro;
                }
            }
        }
 ro:  rox = rocal*exp(-zold/slope + log(j-1));

        if (py[1] > 1.0) if (py[1] <= 0.0) printf("\nY DATA POINT BAD");
        else
        {
            z = log (-log(py[1]));
            for (j=2; j<mm2; j++)
            {
                pj = py[j];
                zold = z;
                if (pj<1.0)
                {
                    z = log(-log(pj));
                    slope = (z-zold)/log((float)(j)/(float)(j-1));
                    if (z > 0.0) goto ro2;
                }
            }
        }
  ro2:   roy = rocal*exp(-zold/slope + log(j-1));
}

/*------------------initialize for plotting--MTF's------------------*/
void init_mtf()
{
        int TRUE=1;

        int x,y;
        for (x=0; x<mm; x++)
```

```
            if(mtfx[x] > mtfmax) mtfmax=mtfx[x];
        for (y=0; y<mm; y++)
            if(mtfy[y] > mtfmax) mtfmax=mtfy[y];

    _setvideomode( _VRES16COLOR);
    _setviewport(0,39,639,439);
        _setwindow(TRUE,0,0,mm+mm,mtfmax);
    _setcolor(15);
    _moveto_w(0.,0.);
}

/*--------------------plot modulation transfer functions----------*/
void plot_mtf()
{
    int x,y;
    for (x=mm2; x<mm; x++)
  {
        _lineto_w((double)(x-mm2),(double)mtfx[x]);
  }
    for (x=0; x<mm2; x++)
    {
        _lineto_w((double)(x+mm2),(double)mtfx[x]);
    }

    for (y=mm2; y<mm; y++)
  {
        _lineto_w((double)(mm2+ y),(double)mtfy[y]);
  }
    for (y=0; y<mm2; y++)
    {
        _lineto_w((double)(mm+ mm2 + y),(double)mtfy[y]);
    }

    _settextposition(1,1);
    printf("%s\n %s",TITLE,__DATE__);

    _settextposition(4,1);
    printf("rox = %f",rox);
    _settextposition(4,38);
    printf("roy = %f",roy);

  _settextposition(40,30);
  _outtext("Hit any key to exit:");
```

```
/*   getch();      there is also a getch in subexpose, only use one*/

}


/*------------------initialize for plotting-lsf's---------------*/
void init_graph()
{
       int TRUE=1;
  _setvideomode( _VRES16COLOR);
  _setviewport(0,39,639,439);
     _setwindow(TRUE,0,0,size+size,lsfmax);
  _setcolor(15);
  _moveto_w(0.,0.);
}
/*--------------------plot line spread functions-----------------*/
void plot_graph()
{
  int x,y;

     for (x=xstart; x<xstart+size; x++)
  {
     _lineto_w((double)(x-xstart),(double)lsfx[x]);
  }
     for (y=ystart; y<ystart+size; y++)
  {
           _lineto_w((double)(size+ y-ystart),(double)lsfy[y]);
  }
     _settextposition(1,1);
      printf("%s\n %s\n\n",TITLE,__DATE__);
  _settextposition(40,30);
  _outtext("Hit any key to exit:");
/*   getch();      there is also a getch in subexpose, only use one*/
 }
```

APPENDIX C ARTIFICIAL STAR PROGRAM

```
 /*Uses an artificial star to test the MTF.c program*has lsf plot also**/
#define TITLE        "Plot an artificial star MTF"

#define AUTHOR                "By W. J. Rall"
          /********Quick C*************************************/

#include <stdio.h>
#include <stdlib.h>
#include <graph.h>
#include <math.h>

#include <sys\types.h>

int xsize=165, ysize=192, xstart=0, ystart=0;
float lsfx[165],lsfmax=0.0,lsfy[192];

int sign=1, m=8, mm=2, mm2;
float mtfx[1024],mtfmax=0.0,mtfy[1024],  re[1024],im[1024],px[512],py[512];
float rocal = 1.0,rox,roy;

void make_star();
void background();
void fft(int m, int sign, float re[], float im[]);
void calc_ro();
void init_mtf();
void plot_mtf();
/*to see lsf's use these two functions instead of mtf ones*/
void init_graph();
void plot_graph();


/*-----------------------main  routine----------------------*/
void main()
{
      float px0,py0;
      int i,j,x,y;
      for (i=0; i<m-1; i++) mm = 2*mm;
      mm2=(int)((float)(mm)/2.0);
```

```
        make_star();
        background();

        for (x=xstart+xsize-1; x>=xstart; x--) re[x] = lsfx[x];
        fft(m, sign, re, im);
        for (x=0; x<mm; x++)  mtfx[x] = re[x];

        /*find the x  power spectrum***************/
         px0 = re[0]*re[0] + im[0]*im[0];
         for (j=0; j<mm2; j++)
         {
              if(mtfx[j] != 0.0 ) if(px0 != 0.0)
              px[j]=sqrt((re[j]*re[j]+im[j]*im[j])/px0)/mtfx[j];
         }
        /*reinitialize the FFT arguements before finding  the Y  FFT*/
         for (i=0; i<1024; i++)
         {
              re[i] = 0.0;
              im[i] = 0.0;
         }

         for (y=ystart+ysize-1; y>=ystart; y--) re[y] = lsfy[y];
         fft(m, sign, re, im);
         for (y=0; y<mm; y++)  mtfy[y] = re[y];

        /*find the y  power spectrum***************/
         py0 = re[0]*re[0] + im[0]*im[0];
         for (j=0; j<mm2; j++)
         {
              if(mtfy[j] != 0.0) if(py0 != 0.0)
              py[j]=sqrt((re[j]*re[j]+im[j]*im[j])/py0)/mtfy[j];
         }

        calc_ro();
        init_mtf();
        plot_mtf();
        /*    init_graph();
       plot_graph();            */

    _setvideomode( _TEXTC80);
}
 /*--------------------make  star-----------------------------------*/
void make_star()
```

```
{
      int x,y;
   float pixel,lsfxmax=0.0,lsfymax=0.0;
      for (x=xstart+xsize-1; x>=xstart; x--)
   {
            for (y=ystart+ysize-1; y>=ystart; y--)
            {
              pixel=4095*exp(-0.1*(float)(((x-82)*(x-82)+(y-96)*(y-96))));
              lsfx[x] = lsfx[x] + pixel;
              lsfy[y] = lsfy[y] + pixel;
              if(lsfx[x] > lsfxmax) lsfxmax=lsfx[x];
              if(lsfy[y] > lsfymax) lsfymax=lsfy[y];
            }
   }
   lsfmax=lsfxmax;
   if(lsfxmax < lsfymax) lsfmax = lsfymax;
}
/*------------Subtract off the background----------------------*/
void background()
{
      int x,y,edge;
   float sumx=0.0,sumy=0.0,ave_x1,ave_x2,ave_y1,ave_y2,aveback;
      edge=(int)(ysize/10.0); /*this uses 20% to find background*/
   /*-----Find the average background--------------*/
     for (x=xstart; x<xstart+edge; x++) sumx=sumx+lsfx[x];
     ave_x1=sumx/edge;  sumx=0.0;
           for(x=xstart+xsize-edge-1; x<xstart+xsize; x++)sumx=sumx+lsfx[x];
     ave_x2=sumx/edge;
     for (y=ystart; y<ystart+edge; y++) sumy=sumy+lsfy[y];
     ave_y1=sumy/edge;  sumy=0.0;
           for   (y=ystart+ysize-edge-1;   y<ystart+ysize;   y++)
sumy=sumy+lsfy[y];
     ave_y2=sumy/edge;
     aveback =((ave_x1+ave_x2)+(ave_y1+ave_y2))/4.0;
   /*----subtract off the background-------*/
     lsfmax = lsfmax - aveback;
           for (x=xstart+xsize-1; x>=xstart; x--)
           lsfx[x]=lsfx[x]-((x-xstart)*(ave_x2-ave_x1)/(xsize-edge)+ave_x1);
           for (y=ystart+ysize-1; y>=ystart; y--)
           lsfy[y]=lsfy[y]-((y-ystart)*(ave_y2-ave_y1)/(ysize-edge)+ave_y1);
/*  printf("\nThe average background is %f\n",aveback);*/
}
```

```
/*********************Find the FFT*******************************/
void fft(int m, int sign, float re[], float im[])
{
      int n, j, j1, ndiv2, n1, n2, k, i, ip, npts;
      int le, le0, le1, l;
      float pts, t, ure, uim, wre, wim, tre, tim;
      double ang, pi;

      pi = 4.0*atan(1.0);
      n = (int)(pow(2.0, (double)m)+1.0e-10);
      j = 1;
      j1 = 0;
      n1 = n - 1;
      n2= n - 2;
      ndiv2 = n/(int)2;

      for(i=0; i<=n2; i++)
      {
            if(i < j1)
            {
                  t = re[j1];    /* you might do better here with pointer operations
*/
                  re[j1] = re[i];
                  re[i] = t;
                  t = im[j1];
                  im[j1] = im[i];
                  im[i] = t;
            }
            k = ndiv2;
            while(k < j)
            {
                  j -= k;
                  k /= 2;
            }
            j += k;
            j1=j-1;
      }

      le = 1;
      for(l=1; l<=m; l++)
      {
            ure = (float)1;
            uim = (float)0;
```

```
            ang = pi/(double)le;
            wre = (float)cos(ang);
            wim = (float)sin(ang);
            le0=le;
            le1=le-1;
            le +=le;

            for(j=0; j<=le1; j++)
            {
                for(i=j; i<=n1; i+=le)
                {
                    ip = i + le0;
                    tre = re[ip]*ure - im[ip]*uim;
                    tim = re[ip]*uim + im[ip]*ure;
                    re[ip] = re[i] - tre;
                    im[ip] = im[i] - tim;
                    re[i] += tre;
                    im[i] += tim;
                }
                t = ure*wre - uim*wim;
                uim = ure*wim + uim*wre;
                ure = t;
            }
        }
        if(sign<0)
            {
                pts = 1/(float)n;
                    for(i=0; i<n; i++)
                    {
                    re[i] *= pts;
                    im[i] *= pts;
                    }
            }
}
/*******************calculate ro*********************************/
void calc_ro()
{
        int j;
        float z,zold,pj,slope;

        if (px[1] > 1.0) if (px[1] <= 0.0) printf("\nX DATA POINT BAD");
        else
        {
```

```
            z = log (-log(px[1]));
            for (j=2; j<mm2; j++)
            {
                pj = px[j];
                zold = z;
                if (pj<1.0)
                {
                    z = log(-log(pj));
                    slope = (z-zold)/log((float)(j)/(float)(j-1));
                    if (z > 0.0) goto ro;
                }
            }
        }
ro:  rox = rocal*exp(-zold/slope + log(j-1));

        if (py[1] > 1.0) if (py[1] <= 0.0) printf("\nY DATA POINT BAD");
        else
        {
            z = log (-log(py[1]));
            for (j=2; j<mm2; j++)
            {
                pj = py[j];
                zold = z;
                if (pj<1.0)
                {
                    z = log(-log(pj));
                    slope = (z-zold)/log((float)(j)/(float)(j-1));
                    if (z > 0.0) goto ro2;
                }
            }
        }
 ro2:   roy = rocal*exp(-zold/slope + log(j-1));
}

/*-----------------initialize for plotting--MTF's-------------*/
void init_mtf()
{
        int TRUE=1;

        int x,y;
        for (x=0; x<mm; x++)
            if(mtfx[x] > mtfmax) mtfmax=mtfx[x];
        for (y=0; y<mm; y++)
```

```
            if(mtfy[y] > mtfmax) mtfmax=mtfy[y];

    _setvideomode( _VRES16COLOR);
    _setviewport(0,39,639,439);
        _setwindow(TRUE,0,0,mm+mm,mtfmax);
    _setcolor(15);
    _moveto_w(0.,0.);
}

/*---------------------plot  modulation transfer functions----------*/
void plot_mtf()
{
        int x,y;
        for (x=mm2; x<mm; x++)
    {
            _lineto_w((double)(x-mm2),(double)mtfx[x]);
    }
        for (x=0; x<mm2; x++)
        {
            _lineto_w((double)(x+mm2),(double)mtfx[x]);
        }

        for (y=mm2; y<mm; y++)
    {
            _lineto_w((double)(mm2+ y),(double)mtfy[y]);
    }
        for (y=0; y<mm2; y++)
        {
            _lineto_w((double)(mm+ mm2 + y),(double)mtfy[y]);
        }

        _settextposition(1,1);
        printf("%s\n %s",TITLE,__DATE__);

        _settextposition(4,1);
        printf("rox = %f",rox);
        _settextposition(4,38);
        printf("roy = %f",roy);

    _settextposition(40,30);
    _outtext("Hit any key to exit:");
        getch();
```

```
}

/*------------------initialize for plotting----LSF's-----------*/
void init_graph()
{
        int TRUE=1;
   _setvideomode( _VRES16COLOR);
   _setviewport(0,39,639,439);
      _setwindow(TRUE,0,0,xsize+ysize,lsfmax);
   _setcolor(15);
   _moveto_w(0.,0.);
}
/*--------------------plot line spread functions----------------*/
void plot_graph()
{
   int x,y;

      for (x=xstart; x<xstart+xsize; x++)
   {
      _lineto_w((double)(x-xstart),(double)lsfx[x]);
   }
      for (y=ystart; y<ystart+ysize; y++)
   {
            _lineto_w((double)(xsize+ y-ystart),(double)lsfy[y]);
   }
      _settextposition(1,1);
       printf("%s\n %s\n\n",TITLE,__DATE__);
   _settextposition(40,30);
   _outtext("Hit any key to exit:");
      getch();
}
```

APPENDIX D JITTER PROGRAM

```
/*jitter.c**********finds the cetroid,jitter, ro and f, plots lsf******/
#define TITLE       "Jitter calculations for ro and f, plots lsf's"

#define AUTHOR              "By W. J. Rall"
          /*******Quick C*************************************/
/*#include <alloc.h>  */
#include <lynxx.h>

#define True 1
#define False 0
#define false 0

#include <stdio.h>
#include <stdlib.h>
#include <graph.h>
#include <math.h>
#include <time.h>

#include <sys\types.h>

/*int frame_ptr[FRAME_BYTES/2];*/
int size=160, xstart=0, ystart=0;
float lsfx[165],lsfmax=0.0,lsfy[192];
float xcsum=0.0, xc2sum=0.0, ycsum=0.0, yc2sum=0.0;  /*for jitter calcs*/

int n=1;   /*number of centroids between each jitter calc*/
float dia=1.0, focal=1000.0, k=12.57E+6;/*wave number, 500nm*/
float xjit,yjit, rocal = 1.0, rox,roy,fx,fy;

void init();
    void subexpose(long tm,int shutter);
            void expose(long tm);
            void long_delay(long tm) ;
            void calc_lsf(unsigned int far *sub_frame);
            void background();
            void centroid();
                void jit();
                void jit_rof();
```

```
                void init_graph();
                void plot_graph();

/*----------------------main routine----------------------*/
void main()
{
   init_lynxx();
      init();
   cooler_off();
   _setvideomode( _TEXTC80);
}

/*----------------------Initialize program info----------------*/
void init()
{
   int cool = 0,shutter=0;
   long tm;

   printf("Expose time in ms.:");
   scanf("%ld",&tm);

   printf("Cooler 0=OFF 1=ON:");
   scanf("%d",&cool);
      if (cool)  cooler_on();

   printf("Use shutter ? (1=Yes/0=No)");
   scanf("%d", &shutter);

   printf("Enter size of subframe in pixels:");
   scanf("%d",&size);

      printf("\nHit any key to quit:");

      subexpose(tm,shutter);
}

/*----------------------subExpose CCD----------------------*/
void subexpose(long tm,int shutter)
{
   unsigned int *sub_frame;
   int sub_frame_bytes, i;
   double deltime,time1,time2;
```

```
    sub_frame_bytes = size*size * sizeof(unsigned int);
    sub_frame = (unsigned int*)malloc(sub_frame_bytes);

    if (!shutter) open_shutter();

    while ( !kbhit() )
    {
        for(i=0; i<n; i++)
        {
            if (shutter)  expose(tm);
            else
            {
              clrccd();  /*if not using shutter*/
              long_delay(tm);
             }
            x_param = (unsigned char) xstart;
            y_param = (unsigned char) ystart;
            size_param = (unsigned char) size;
            ptr_param = sub_frame;
            digitize_sub_frame();

            calc_lsf(sub_frame);
            background();
            centroid();
        }
        jit();
        jit_rof();
        init_graph();
        plot_graph();
    }
  free(sub_frame);
  close_shutter();
}

/*-----------------------Expose  CCD-----------------------*/
void expose(long tm)
{
  clrccd();
  open_shutter();
  long_delay(tm);
  close_shutter();
}
```

```
/*-----------------------long delay-----------------------*/
void long_delay(long tm)
{
    delay_param = 1000;
    while (tm > 1000)
    {
            delay_ms();
            tm = tm -1000;
    }
    delay_param = (unsigned short)tm;
    delay_ms();
}

 /*--------------------calculate the line spread functions-----------*/
void calc_lsf(unsigned int far *sub_frame)
{
   unsigned int pixel;
   int x,y;
   float lsfxmax=0.0,lsfymax=0.0;
   for (x=0; x<165; x++) lsfx[x]=0.0;
   for (y=0; y<165; y++) lsfy[y]=0.0;
   for (x=size-1+xstart; x>=xstart; x--)
   {
       for (y=size-1+ystart; y>=ystart; y--)
       {
         pixel=*(sub_frame++);
         lsfx[x] = lsfx[x] + (float)pixel;
         lsfy[y] = lsfy[y] + (float)pixel;
         if(lsfx[x] > lsfxmax) lsfxmax=lsfx[x];
         if(lsfy[y] > lsfymax) lsfymax=lsfy[y];
       }
      }
      lsfmax=lsfxmax;
      if(lsfxmax < lsfymax) lsfmax = lsfymax;
}

/*------------Subtract off the background----------------------*/
void background()
{
    int x,y,edge;
    float sumx=0.0,sumy=0.0,ave_x1,ave_x2,ave_y1,ave_y2,aveback;
    edge=(int)(size/10.0); /*this uses 20% to find background*/
    /*-----Find the average background--------------*/
```

```
    for (x=xstart; x<xstart+edge; x++) sumx=sumx+lsfx[x];
    ave_x1=sumx/edge;  sumx=0.0;
    for (x=xstart+size-edge-1; x<xstart+size; x++) sumx=sumx+lsfx[x];
    ave_x2=sumx/edge;
    for (y=ystart; y<ystart+edge; y++) sumy=sumy+lsfy[y];
    ave_y1=sumy/edge;  sumy=0.0;
    for (y=ystart+size-edge-1; y<ystart+size; y++) sumy=sumy+lsfy[y];
    ave_y2=sumy/edge;
    aveback =((ave_x1+ave_x2)+(ave_y1+ave_y2))/4.0;
  /*----subtract off the background-------*/
    lsfmax = lsfmax - aveback;
    for (x=xstart+size-1; x>=xstart; x--)
        lsfx[x]=lsfx[x]-((x-xstart)*(ave_x2-ave_x1)/(size-edge)+ave_x1);
    for (y=ystart+size-1; y>=ystart; y--)
        lsfy[y]=lsfy[y]-((y-ystart)*(ave_y2-ave_y1)/(size-edge)+ave_y1);
/*  printf("\nThe average background is %f\n",aveback);*/
}

/*-------------Find the centroids-----------------------------------*/
void centroid()
{
   int x,y;
   float xcent,ycent,sumlx=0.0,sumly=0.0,sumxlx=0.0,sumyly=0.0;

      for (x=xstart+size-1; x>=xstart; x--)
      {
       for (y=ystart+size-1; y>=ystart; y--)
       {
        sumxlx = sumxlx+ lsfx[x]*(float)(x);
        sumlx = sumlx + lsfx[x];
        sumyly = sumyly+ lsfy[y]*(float)(y);
        sumly = sumly + lsfy[y];
       }
      }
      xcent=sumxlx/sumlx; ycent=sumyly/sumly;
      /*printf("\n Centroid is xcent=%f \n ycent=%f",xcent,ycent);*/
      /*    sum centroids for jitter calcs***************/
      xcsum = xcsum + xcent;  xc2sum = xc2sum + xcent*xcent;
      ycsum = ycsum + ycent;  yc2sum = yc2sum + ycent*ycent;
}

/*******************calculate the jitter*****************************/
 void jit()
```

```
 {
       xjit = sqrt((xc2sum-xcsum)/(float)(n));
       yjit = sqrt((yc2sum-ycsum)/(float)(n));
       xcsum=0.0; xc2sum=0.0;  ycsum=0.0; yc2sum=0.0;
       printf("\n Jitter is xjitter=%f   yjitter=%f",xjit,yjit);
 }

/*******************calculate ro and f *from the jitter****************/
void jit_rof()
{
      float a,b;

      a = xjit * k / focal;
      b = 1.418 * pow(dia,0.333) * pow(a,2);
      rox = rocal / pow(b,0.6);

      a = yjit * k / focal;
      b = 1.418 * pow(dia,0.333) * pow(a,2);
      roy = rocal / pow(b,0.6);

      fx = xjit;
      fy = yjit*roy;
}

/*------------------initialize for plotting---------------------*/
void init_graph()
{
       int TRUE=1;
  _setvideomode( _VRES16COLOR);
  _setviewport(0,39,639,439);
  _setwindow(TRUE,0,0,size+size,lsfmax);
  _setcolor(15);
  _moveto_w(0.,0.);
}

/*--------------------plot line spread functions-----------------*/
void plot_graph()
{
      int x,y;

      for (x=xstart; x<xstart+size; x++)
      {
            _lineto_w((double)(x-xstart),(double)lsfx[x]);
```

```
        }
        for (y=ystart; y<ystart+size; y++)
        {
              _lineto_w((double)(size+ y-ystart),(double)lsfy[y]);
        }
        _settextposition(1,1);
         printf("%s\n %s\n\n",TITLE,__DATE__);

         _settextposition(4,1);
         printf("rox = %g\nfx = %g",rox,fx);
         _settextposition(4,38);
         printf("roy = %g",roy);
         _settextposition(5,38);
         printf("fy = %g",fy);

         _settextposition(40,30);
         _outtext("Hit any key to exit:");
/*    getch();   subexpose also has one, only use one*/

}
```

APPENDIX E FRAME RATE MEASUREMENT CODE

```
/*time.c****times digitize,calclsf and background, cetroid,jitter*/
#define TITLE        "Time calculations"

#define AUTHOR                  "By W. J. Rall"
          /**Quick C  ******Sample of time code*******************/
#include <time.h>

   while ( !kbhit() )
   {
         time1 = clock();     /*  start clock  */
         for(i=0; i<128; i++)
         {
              digitize_sub_frame();
              calc_lsf(sub_frame);
              background();
              centroid();
         }
         jit(i);

         time2=clock();     /*     stop clock   */
    deltime = (time2-time1)/CLK_TCK;
         printf("\nTime for 128 frames =%f sec", deltime);
     }
```

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center Cameron Station Alexandria, VA 22304-6145	2
2. Library, Code 52 Naval Postgraduate School Monterey, CA 93943-5002	2
3. Prof. Donald L. Walters Department of Physics (Code 61We) Naval Postgraduate School Monterey, CA 93943-5004	7
4. Cpt. Ann Slavin Code PL/WE Phillips Laboratory Kirtland Air Force Base Albuquerque, NM 87117	1
5. Lt. William J. Rall Superintendent (ds3) U.S. Coast Guard Academy 15 Mohican Ave. New London, CT 06320-4195	3