Design and HCI Applications of a Low-Cost Scanning Laser Rangefinder

by

Joshua Andrew Strickon

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1999

Author .....
Department of Electrical Engineering and Computer Science
Feb 3, 1999

Certified by .....
Joseph Paradiso
Principal Research Scientist
Thesis Supervisor

Accepted by .....
Arthur C. Smith
Chairman, Department Committee on Graduate Students

Design and HCI Applications of a Low-Cost Scanning Laser Rangefinder

by

Joshua Andrew Strickon

Submitted to the Department of Electrical Engineering and Computer Science
on Feb 3, 1999, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

A low-cost scanning laser rangefinder was developed and optimized for applications involving real-time human-computer interaction (HCI). The usable range extended across several meters, enabling the device to be applied to such applications as tracking hands (e.g., as in touchscreens) in front of large displays. The system was implemented as a continuous-wave, quadrature phase-measuring rangefinder. This thesis will examine the current states of laser rangefinders and discuss the design and function of the prototype system, as well as propose and demonstrate new HCI applications of such a low cost device.

Thesis Supervisor: Joseph Paradiso
Title: Principal Research Scientist

Acknowledgments

I'd like to thank Joe Paradiso for advising me on this project for the past few years. We actually made it.

I'd also like to thank Tod Machover and the Opera of the Future group for supporting me during this project.

I'd like to thank Neil Gershenfeld and the Physics and Media Group for help and support.

I would like to thank Pete Rice for his involvement with the SIGGRAPH installation and for providing interesting content to be used with the Rangefinder.

I would like to thank everyone in the new Responsive Environments group for providing a place to setup the rangefinder.

I would like to acknowledge the support of the Things That Think Consortium and the other sponsors of the MIT Media Laboratory.

Finally, I would like to thank my family for supporting me.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As computers are slowly creeping out of their traditional settings, they will require new interfaces[Par98]. For example, there is currently no cheap, easy and universally applicable way of getting accurate real-time measurements of a person's gesture in an immersive environment. More specifically, different technologies have been applied in HCI (Human Computer Interaction) research and development to track the position of hands and objects atop intelligent surfaces. None of them so far have been accurate, cheap, fast and generally reliable enough to warrant their application in tracking hands above large interactive (e.g. display) surfaces. In recent years, laser rangefinders have been a tool of surveyors[Re90], and aid to soldiers[Urb95] and eyes for robot vision[Eve95]. To meet such requirements, scanning laser rangefinders have typically cost several thousands of dollars and produced results of sub-millimeter accuracy and/or attained kilometers of range measurement. This thesis describes the development of much lower cost scanning laser rangefinders with moderate resolution and only several meters of range, well-suited to working as an input device for large screen displays.

1.1 HCI Motivation and Existing Methods

Non-contact user interface design has been addressed in a variety of fields by the use of multiple disciplines. A familiar illusion of a non-contact interface most commonly

Figure 1-1: Chromakey Technology

takes form during the evening weather telecasts (Fig. 1-1), where chromakey technology allows actors to be placed in a scene through the use of a blue screen. By replacing the blue back drop with a video image, a person can be situated anywhere. This technology doesn't allow for much interaction though, as the actor sees only the blue screen, and must observe an off-camera monitor to see himself as immersed.

Early uses of non-contact interfaces date back to the 1920's, with the work of Leon Theremin[Cha97]. His use of capacitance proximity sensing as a musical instrument was the earliest electronic, free gesture device. More recently, other types of non-contact sensing have been applied to HCI applications in *The Brain Opera*[Wil97], a large scale interactive music project developed at the MIT Media Lab. The sensing systems varied from sonars and radars to electric field sensors[Par97]. Electric Field

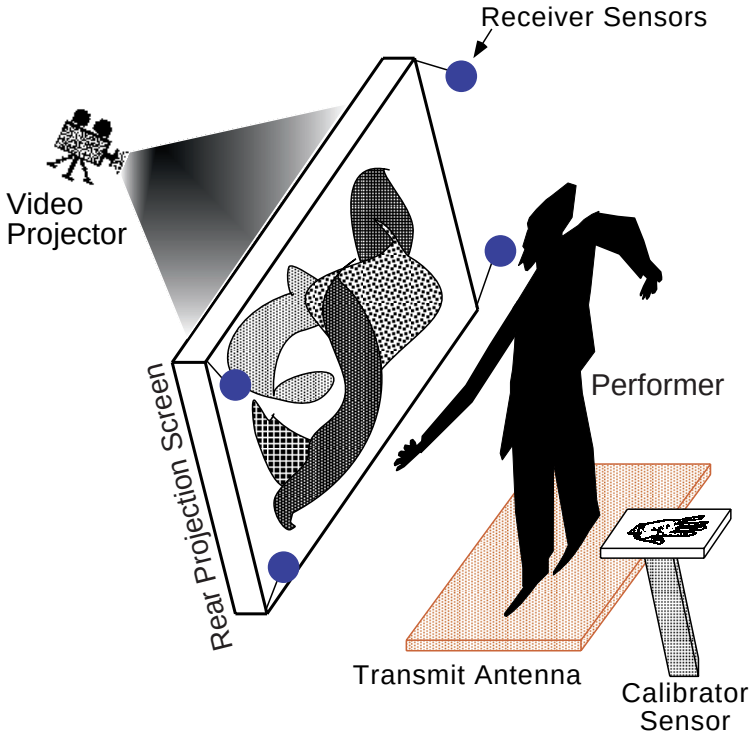

Figure 1-2: Gesture Wall Setup

Sensing has been used to track gesture in one of *The Brain Opera*'s installations, measuring the body's motion in front of projection screen[SWD+98]. This installation was called the *Gesture Wall* (Fig. 1-3). Here, as in Fig. 1-2, the user's body was driven by a weak electric field coupled in through the floor; a set of receive electrodes were placed at the edge of the display perimeter, measuring the body's distance via capacitive coupling. The architecture of this system, however, didn't provide the ability to track repeatably. The ambiguity as to what part of the body the system was tracking put undue constraints on the body's position and posture.

Other approaches have used video systems[Wre97]. Computer vision research seeks to solve some of the same problems, but requires large processing on the part of the host computer. The performance overhead is considerable and these systems generally tend to be very sensitive to background light and clutter, often relying on high contrast of a person's image. They also can require expensive computers to implement the tracking algorithms in real time. They often try to correct for many

Figure 1-3: Gesture Wall In Use

of these problems and derive 3D through the use of multiple cameras[Aya91] as in the SmartDesk installation (see Fig. 1-4)[AP96], although still costly, introducing more ambiguity and processing, and not well-suited to tracking hands just above the screen.

Other commercial optical systems have been released for such interfaces. The Dimension Beam, from Interactive Light[*www.interactivelight.com*], uses the simple intensity of an infrared light source, as reflected off a person's hand, to measure range[FW95]. This system, however is a one-dimensional sensor and is sensitive to material color and background light, as it responds only to unprocessed intensity.

The largest commercial market for 2-D surface tracking devices has been in "smart wallboards", that digitally record handwriting. While many of these systems require contact or pressure to be applied against the sensitive surface (working like a large touchscreen), others detect the position of objects just above the wallboard. Most of latter employ optical sensing, which enables simple, passive reflecting targets to be easily detected in a sensitive plane defined by a collimated light source, such as

Figure 1-4: The Media Lab's SmartDesk, Tracking hands with Multiple Cameras

a diode laser. The requirement of these retro-reflectors detracts from an interface's ability to be transparent to the user and effective in allowing natural motion. Many of these wallboard sensors are not by themselves ranging systems, but rather rely on triangulating multiple angular measurements. These systems then have the drawback of being able to unambiguously track only a single object.

Other technologies are also being used in attempts to create interactive surfaces. *The Holowall*[ROMK98] is an installation, designed by researchers at Sony, that employs IR-sensitive cameras behind a screen with rear projection[RM87]. There are also IR transmitters behind the screen that blast the surface with invisible IR light. The video cameras are sampled by a computer and the amount of IR radiation reflected from people on the front side of the IR-transparent projection screen is measured by the cameras. This method is very sensitive to background IR light and clutter, as well as having the large overhead of the video processing. It also requires an

IR-transparent display screen.

There is essentially no non-contact interface available at the moment that can track a user interacting with a smart wall without introducing considerable constraints or performance compromises. This prompted research into a smarter interface, and encouraged a look at scanning laser rangefinders, which measure both range and angle from a single location.

Chapter 2

Rangefinding

2.1 Advantages of Laser Rangefinders

Laser Rangefinders have the capability of tracking objects in free space without ambiguity. The laser highly-collimated beam provides a unique sample point that enables this method to accurately track particular objects over large ranges in distance. In addition, the amount of time it takes the light to reach the target and return as reflection can be accurately measured, enabling one to use the speed of light as a reference for measuring distance, as incorporated into a variety of laser rangefinders. Sonar can also be used, but the travel time of sound is 6 orders of magnitude longer than that of light. There are also other optical rangefinding techniques that rely on triangulation, but these simply take advantage of the unique sampling point or pattern and do not use the time-of-flight of the beam. Radar and sonar generally have much broader beams, thus cannot be used to easily image the intricacies needed to track precise human hand gesture.

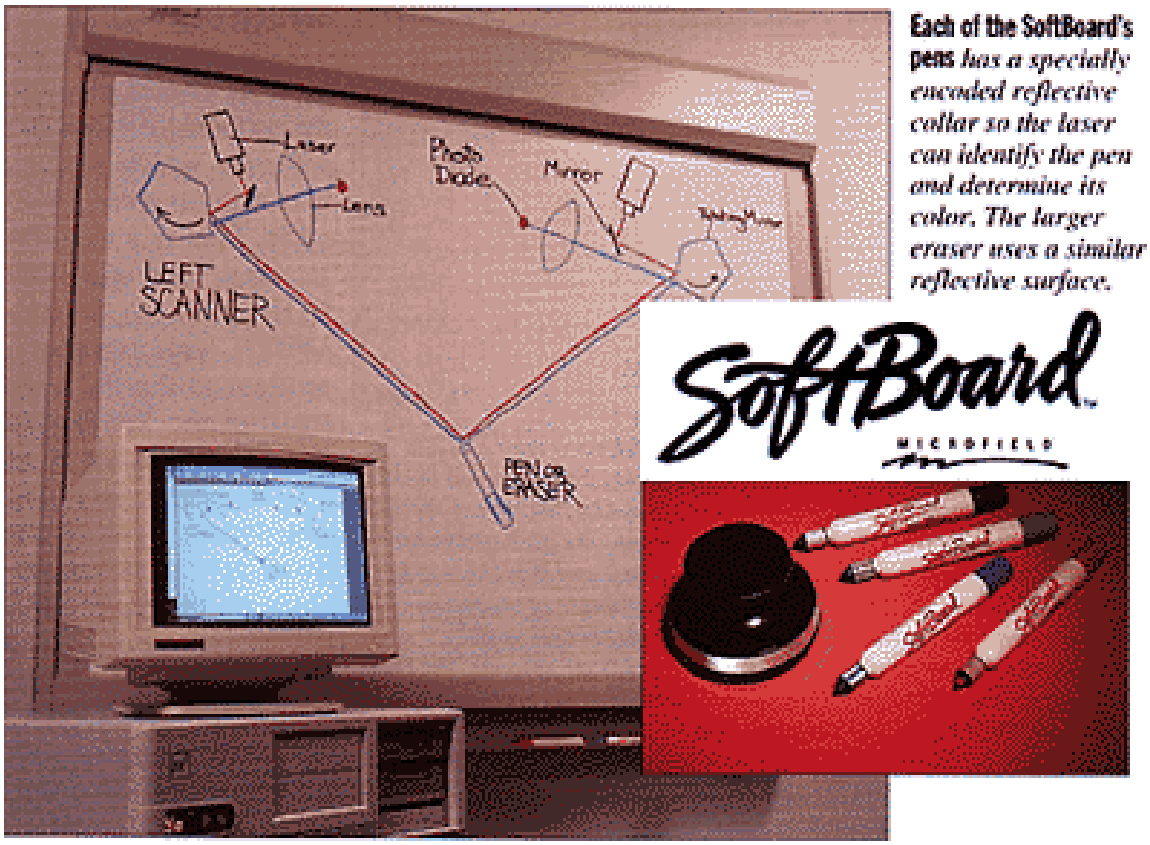

Figure 2-1: The Softboard from Microfield Graphics

2.2 Methods of Rangefinding

2.2.1 Triangulation

The most common laser rangefinders, triangulation devices, come in two varieties. The first employs a single sensor. These devices optically image (via a lens) onto a position-sensitive photodiode or an imaging array. With every change in range, there is an associated change in the incident angle, hence imaged position of the reflected beam. This technique is commonly used in optical proximity sensors[CG95]. This measurement is nonlinear, asymptotic with distance. These devices are often used for profiling surfaces[PTR+98] and typically have short active ranges, well suited to capturing 3D surfaces for computer graphics applications. A compact device can be constructed for a reasonable cost, but it will not meet the required accuracy spec-

ification across several meters[BFB+98]. Another type of triangulation rangefinder is one that uses two incident beams at a fixed angle. The distance between the two beams imaged is thus a function of range. This system is also nonlinear and has the additional problem of not being able to be scanned planarly. This system is often used for surface imaging on a very small scale[Rea97]. The third type of triangulation system uses two scanners and independently calculates the angle of an object relative to each scanner, purely determining its x,y position. A well known example of the system is the "SoftBoard" by Microfield Graphics[Egl94](See Fig. 2-1). This system incorporates the two displaced laser scanners atop a white board, each of which only measures angle. A user is required to write with special pens identified by a barcoded retro reflector on the tip. This system is only used to track a single object due to the requirement of the dual scanners. These types of rangefinders work well in this application, but are clearly not the best solution for a free-gesture, multi-hand, bare-hand input device over a large surface.

2.2.2 Time-of-Flight

Time-of-flight laser rangefinders[Rea97] rely on a simple principle. They function by measuring the time that a pulse of laser light takes to eminate out, reflect off a target and return to a reciever on the rangefinder. This measurement is linear with distance, improving on triangulation techniques. This system employs the least analog hardware, only requiring a pulse generator and a simple high-gain reciever. Since one only requires a logic-level signal from the returning wavefront, a simple fast amplifier circuit can be used. The system relies on a high speed time-to-digital converter(TDC)[Gen91] (potentially a just a fast counter), and a fast photodiode to maintain the timing information. Since this method isn't bound by distance, it is easily scaled for varying ranges. Many methods employ dual phase-locked clocks for varying levels of accuracy[BS91]. These rangefinders are the type that are most often used on military aircraft and vehicles. Theoretically, they could be scaled easily and used most effectively, since they are digital in many aspects.

There are variations on the time-of-flight rangefinder that measure frequency.

The *Acuity* rangefinder uses such technique[*www.acuityresearch.com*]. It measures the frequency of a an oscillating feedback loop composed of a laser diode and the receiver; the transit time of the light to and from the target forms the delay line that defines the oscillator's frequency. In both types of devices, the receive and transmit electronics can be very similar, but the fundamental output is different, one being the period and one being the frequency. The frequency measurement can be done easily using analog or digital technologies. The period can most easily be done digitally. Time-of-flight rangefinders have the problem of being partially sensitive to background noise, since this method doesn't synchronously detect the laser; they generally use a very intense but brief pulse of laser light to readily be discriminated.

2.2.3 Continuous Wave Phase

For our purposes, phase-measurement devices improve on triangulation rangefinders in many ways. The first and most important distinction is that phase is a linear measurement. Phase-measuring devices determine distance by determining the phase delay of a continuous-wave, amplitude-modulated beam. The major limitation to such a system is given by the modulation frequency, as the range measurement will phase-wrap after one wavelength is reached between the laser and photodetector (a half-wavelength from the scanner to the object if the laser and detector are co-located), and the achieved resolution is inversely proportional to the wavelength. Since we are considering the speed of light, even at tens of Megahertz of modulation, the wavelength is still several meters. Because they are synchronously detected, phase-measurement devices are also less sensitive to background light and noise as well. These are often used for high-end measurement applications, (producing sub-millimeter resolution across large ranges)[HHO94], and at today's prices, the cheapest of these systems that respond fast enough for real-time scanning can still cost on the order of $10,000.00, which is too expensive for common HCI usage. The tolerances for user interface applications aren't that strict, however, and the same techniques can be incorporated in a less accurate and lower-range version that can be designed at lower cost.

This, as are of all the devices introduced here, is still an optical system, thus is subject to occlusion of inline objects along a single ray; usually answerable through software tracking filters[RK94] or dual scanners.

Chapter 3

Design of the Rangefinder

Two different rangefinders were developed using two of the methods outlined in Chapter 2. Both a triangulation device and a continuous-wave phase measurement device were constructed. Although the phase device was the obvious winner theoretically, we chose to build the triangulation rangefinder first, as it was electrically much simpler. We found that the non-linearity of the triangulation range measurement was unsatisfactory for a compact user interface over long distances, but were able to pull out some useful data when scanning a small area. After more electronics design, we were able to build the phase-measuring device. Two units were constructed, a prototype using mostly hand-stuffed boards and a second revision, using all printed circuit boards. The first version was also built using expensive off-the-shelf prototyping optics hardware, while the second version was built using simple, custom-built mechanical assemblies. After another modest design simplification, this device should be ready for prompt duplication and limited-run manufacturing.

3.1 Triangulation

A triangulation rangefinder was constructed as a starting ground for the development of low-cost laser hand trackers[PS97]. As this technique can be explored using video cameras or simple, low-frequency photodiode circuits, it is a relatively easy first step to examine the utility of such devices. Our triangulation prototype consisted of two

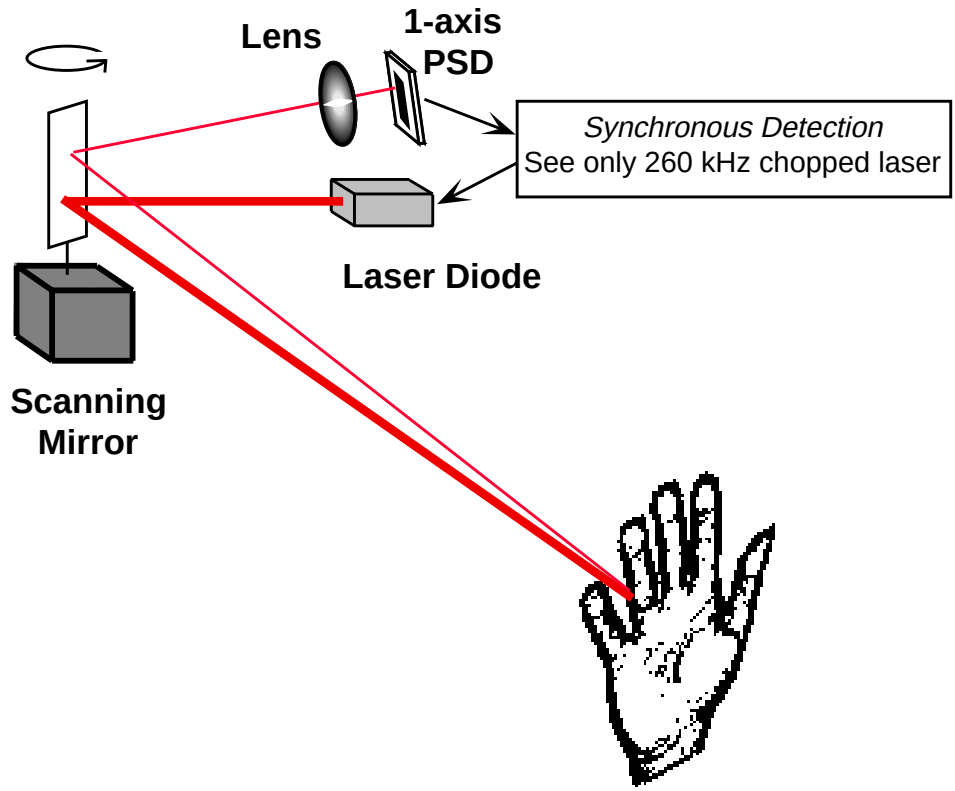

Figure 3-1: Triangulation Rangefinder

components; a scanning head and a synchronous photo-detector.

3.1.1 Lateral-Effect Photodiode Camera and Scanning Assembly

The head-scanning module consists of a few subassemblies. It contains the laser driver, the lateral effect photodiode camera, the start and stop phototransistors and the scanning optics. This arrangement is depicted in Figs. 3-1 and 3-2. The scanning optics are arranged to align the laser in a vertical line with the camera. The scanner itself is a cut piece of mirror, glued into a delron mount, attached to a DC toy motor. The mirror assembly scans the camera view and laser together, such that the camera images laser reflections along a constant vertical line. The photodiode

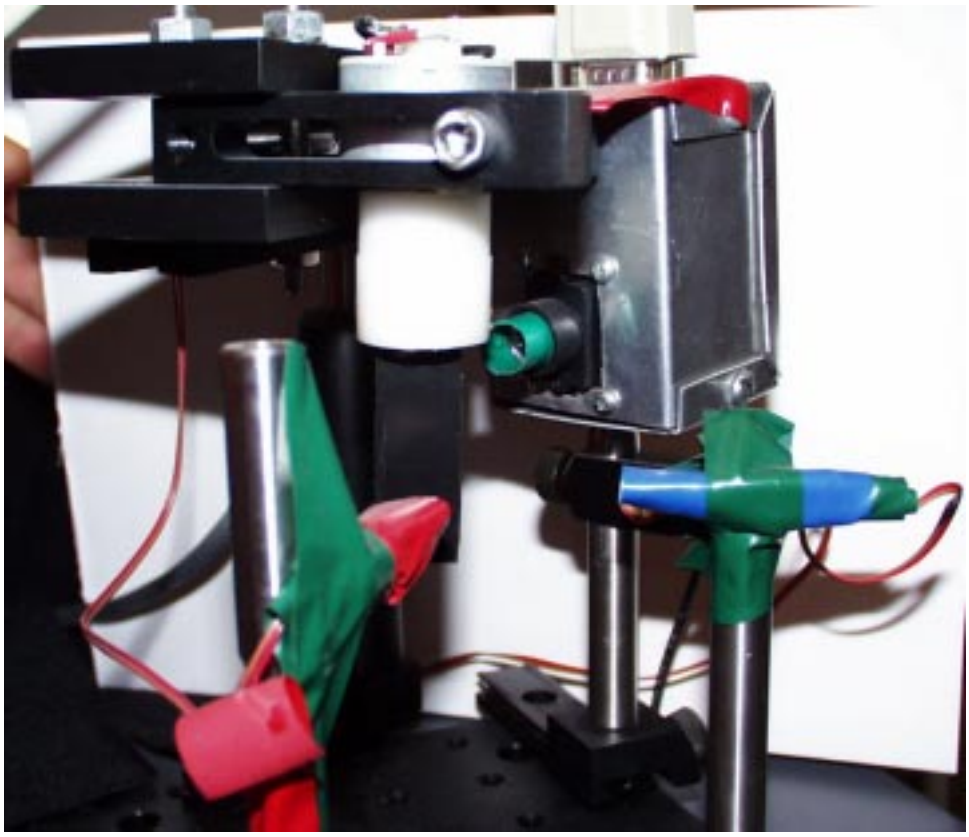

Figure 3-2: Triangulation Rangefinder Scanning Head

camera contains a one-dimensional Hamamatsu lateral effect photodiode and front-end amplifier for the top and bottom signals. The phototransistors are used to trigger the microcontroller to start and stop sampling at the beginning and end of the scan. The two photodidode signals, along with the phototransistor signals, are sent to the main rack unit. Although a video camera could be used to image the reflected laser spot[Bov88], we opted to use the lateral effect photodiode system because of the attenuation of background light from synchronous detection and the lack of any image processing overhead.

3.1.2 Synchronous Detector and Position Computation

The laser is amplitude modulated with 262Khz square wave. The rack unit contains a synchonous detection circuit as well as the sum and difference computations. The

Figure 3-3: Triangulation Position Detection Circuitry

following equations are used to determine the asymptotic range, where *top* and *bottom* are the corresponding amplified photodiode signals. The normalized position of the imaged spot on the photodiode(y) can be calculated via Eq. 3.1. The distance(r) can then be computed with an additional scale factor(s, the baseline distance separating the detector and the laser), the focal length of the photodiode optics(f_o), the resulting magnification factor(m_o) and the angle of the incident beam to the photodode's lens as measured from the baseline(θ).

$$y = \frac{top - bottom}{bottom + top} \tag{3.1}$$

$$r = \frac{s f_o (m_o + 1) \cos(\theta)}{y} \tag{3.2}$$

After the signals are amplified in an AGC amplifier(using the demodulated sum as a reference), and brought to baseband by the demodulator, this computation is done using two simple opamp circuits that are then fed into a multiplier circuit configured as a divider. The phototransistors are used to trigger a ramp circuit, used as an analog timing reference for interpolating the mirror angle between the start and stop pulses. The ramp is built out of a phase-locked-loop, locked to the motor speed. This allows the unit to maintain a continuous measurement of angle, independent of average motor speed. The ramp signal, along with the range signal, is sent to the microcontroller board. An image of the circuit boards appears in Fig. 3-3.

3.1.3 Data Acquisition and Analysis

A data acquisition board based around the Motorola 68HC11 microcontroller is used to interface the rangefinder to the host computer. It performs limited data analysis and reduction. Figure 3-4 shows this system in operation, tracking a hand across a scanned tabletop. However, this system was never used with a more sophisticated program, as it didn't provide enough range for the intended application. The useable range was less than one meter. The asymptotic falloff with $1/r$ (see eq. 3.2) severely limited the dynamic range of the system. This opted for developing the phase-measurement device.

3.2 Continuous Wave Phase

A continuous-wave (CW) phase-measuring rangefinder was constructed[SP98]. Two versions were built, a prototype and more robust version for public installations. For the purpose of the design, we shall describe the second, better-perfected version. The circuits were similar in both cases. The major changes were mostly in data acquisition and software. Both devices were displayed internally at the Media Lab. The second device was shown with a public installation at SIGGRAPH 98[SRP98]. The

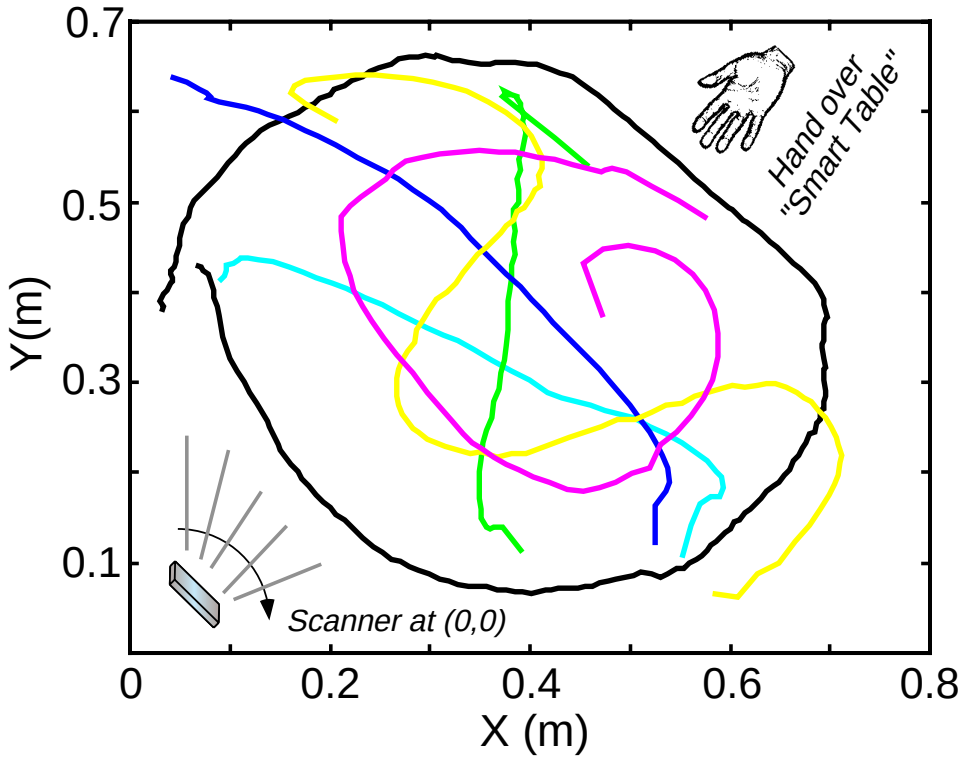

Figure 3-4: Drawing in the air, above a Smart Tabletop, with the Triangulation Rangefinder

continuous-wave device allows for a measurement linear in r. This is important in measuring long distances, allowing the device to return a linear measurement with the dynamic range evenly distributed about the active area. The continuous-wave device measures range by detecting the phase slip of an amplitude modulated laser beam. The phase slip can be measured by a phase detector or by quadrature detection. Quadrature detection allows one to disregard any phase loss in the circuit by calibrating the range after it is setup. In both cases, to avoid phase-wrap and range ambiguity, the measuring range must not exceed one half wavelength of the modulated beam (accounting for the return path as well). Thus the farthest distance that one can unambiguously measure here is one half wavelength. We chose a modulation oscillator frequency(25MHz) appropriately such that the allowed range of phase is within a reasonable amount of the distance we would like to measure. This produces

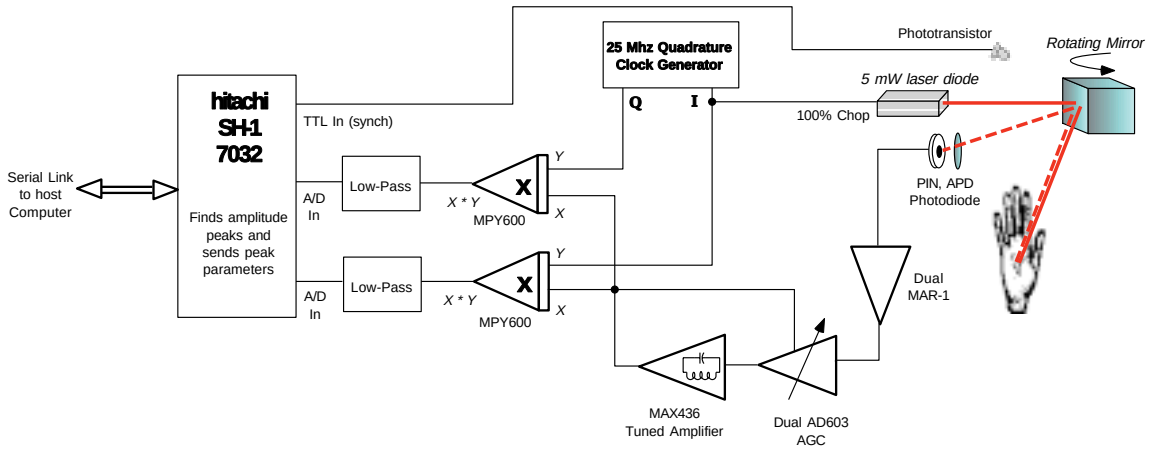

Figure 3-5: Block Diagram of CW Phase-Measuring Rangefinder

the following wavelength.

$$\lambda = \frac{c}{f} = \frac{3x10^8 \frac{m}{s}}{25Mhz} \approxeq 12m \tag{3.3}$$

This $\lambda \approxeq 12$ meters thus allows us to measure up to 6 meters away from the rangefinder, well suited to the projection surfaces that were intended. It is possible to easily change the oscillator frequency along with the demodulation filter cutoff and thereby change the measurement range of the rangefinder for other applications.

The rangefinder was constructed modularly. As with the earlier triangulation rangefinder, the laser and receive optics and electronics were all in one unit. The demodulator was in a separated rack unit along with the microcontroller interface. As the operating frequency of the phase system increased by two orders of magnitude beyond the triangulation unit, many of the components were upgraded to reflect that change. This device was built and satisfied our criterion for such a system. A block diagram is shown in Fig. 3-5 and described below. Schematics of the system are given in Appendix A.

3.2.1 Avalanche Photodiode Camera Imager

The sensor for the system was a EG&G Optoelectronics CA30817E avalanche photodiode (APD). The APD operates in avalanche breakdown mode and creates a high gain junction. In order to operate in the avalanche mode, the photodiode requires a 400 volt back bias. This is accomplished by using a EMCO High Voltage Co. step up inverter. It is coupled to the photodiode by a simple passive low-pass filter stage, to attenuate noise. The photodiode camera contains two circuit boards connected back-to-back. The first board is the first amplification stage. The second board contains the second stage of the front end amplifier, an AGC, a tuned filter, and a line driver. The boards are mounted in a machined block of aluminum. A Thor Labs 1" optics mount is attached to the the block. A delron ring was machined to center the photodiode in the optical assembly. This is important, as the total active area of the photodiode is only $1.5mm^2$, making alignment critical. The first board requires only +12 Volts. The second board runs with +5 and -5 voltage rails. These voltages are regulated on the board by 7805 and 7905 series regulators. The photodiode is directly coupled into a cascade of 2 Minicircuits MAR-2 2 GHz, low-noise, monolithic amplifiers. The MAR-2 stages are coupled together, as well as coupled to the second board by $.1\mu F$ capacitors. The two MAR-2 stages provide about 25dB of gain, each MAR-2 providing approximately 12.5dB. The first component of the second amplifier board is an Active Gain Control (AGC) circuit[AD695]. The AGC is built using two Analog Devices AD603 variable gain amplifiers. They provide a low-noise, high-gain circuit circuit that can run at the frequency we desire. A switch allows this circuit to be flipped between an adjustable constant gain or the AGC mode, with adjustable reference. The AGC amplifier is able to provide up to 83dB of gain. In actual use though, the AGC was always on and the constant gain was never used, although handy for debugging. The feedback loop, following the AGC, is closed after an additional tuned amplifer stage. The tuned amplifier is constructed out of a Maxim MAX436 wideband transconductance amplifier. It is tuned to the modulation frequency using an iron core inductor. This allows the Q of the circuit

to be maximized(providing an additional 30dB of gain). The MAX436 is followed by two Linear Technology's LT1223 100MHz current-feedback amplifiers. They are used as low-gain cable drivers (in actuality each provides an additional 7dB of gain then reduced by a factor of 2 from cable matching). A 50Ω BNC cable follows this stage and connects to the rack unit demodulator. The signal following the last stage is well-conditioned and stable enough to be fed directly into the demodulator. As the gain of both boards and photodiode together approaches 200dB, it was important to appropriately shield the boards and take considerable layout effort to prevent the laser modulation from contaminating the photodiode signal. The boards each have top and bottom ground planes. The aluminum mount is also grounded. The openings of the housing are covered by copper plates. The performance of this unit was optimized in the second prototype by incorporating lessons learned from the first prototype into the PCB board layouts.

3.2.2 Laser Driver

The laser driver board was also attached to the main scanning assembly. It is housed in its own floating, shielded box, isolating it from the receive electronics. It contains a 50Ω line receiver for the 25MHz clock signal. The output of this receiver is connected to the gate of a fast Supertex VN1304 MOSFET. The drain of the MOSFET is wired to a 68Ω resistor, put in series with the laser diode, used to limit the drive current. A 5mW visible laser diode is used. For various tests, depending on the availability and cost, a range of wavelengths were used, varying from 630-670nm. The laser was to be driven at about 60mA averaged current (as the laser is to be modulated with a 50% duty cycle, the peak current can be higher than the recommended setting). The anode of the laser diode is connected to the laser power connection of the board. The power source for the laser diode can be switched on the rack unit. It can be a constant 5 volts or a servoed voltage, that is a function of the demodulated receive signals (the maximum of $|I|$ and $|Q|$). This allows the power of the laser to decrease with reflected intensity; e.g., increasing the measurements range by not saturating the front end amplifier stage when an object is close to the scanner and avoiding eye

damage should someone look into the laser at very close range. The performance of the laser drive circuitry allows the laser to be modulated well above the required 25MHz.

3.2.3 Optical Assemblies

The optical system(Fig. 3-6) had to be as simple as possible, yet allow for alignment adjustments in all axes. This was accomplished by shooting the laser at a right angle to the lens tube of the camera. A mirror was positioned under the lens tube to reflect the laser along the central axis of the scanner. The mirror can be angled up and down as well as left and right. Although the laser is vertically displaced from the camera's view, the mirror is adjusted such that the laser focuses at infinity onto the center of the photodiode. This geometry enables objects to be detected when they are displaced at least 6" from the mirror. A photo of the actual refined scanning assembly appears in Fig. 3-8. One might think that the best solution would be to use a beam splitter, but the loss of optical power on the receive end, together with some difficulty in removing residual reflection, would not be acceptable in this design. An option for future designs(see Fig. 3-7) could employ the trick of drilling a small hole in a mirror, angled at 45 degrees in front of the lens. The laser could then be shot through the hole. As the lens is much bigger than the hole, the lost area of the hole gives insignificant loss in the amount of laser light arriving at the receiver. This configuration would require the least adjustment of all, as the laser and camera views would be colinear.

The lens and the scanning mirror are aligned to scan through approximately 90 degrees. The scanning mirror is a surplus stepper motor fitted with a machined 4-sided mirror mount. Each facet is 1.5" wide by 1" high. This allows the 1" lens to collect a reasonable amount of light across the scan. Two phototransistors are mounted on each end of the scanning range (labeled stop and start in Figs. 3-6 and 3-7). The collectors of the transistors are pulled up to 5 volts with a resistor in parallel with a filtering capacitor. The collector signals are conditioned by a 4050 buffer. The pullups and the buffers are located on the demodulator board. This generates a clean

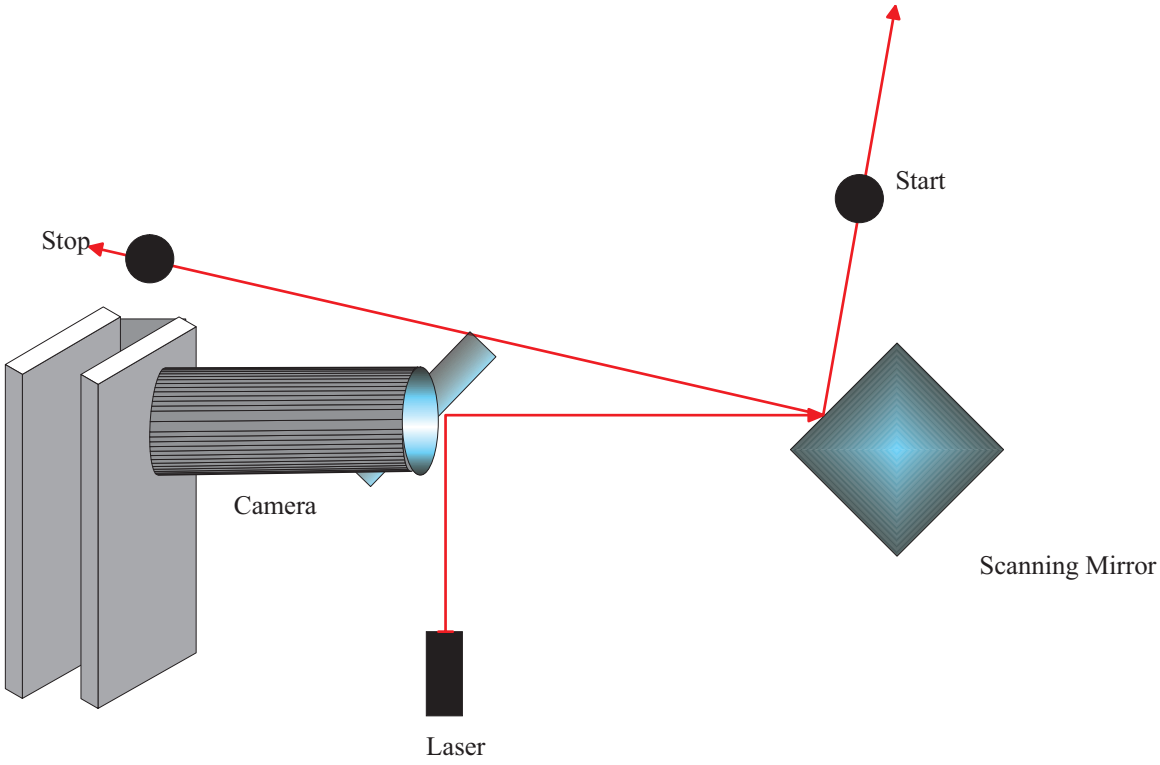

Figure 3-6: Camera Optics, with Camera Vertically Displaced from the Laser

set of timing signals for the digital board. This planar scanning system works well for the application of scanning a smart surface, but could be redesigned to scan other geometries.

3.2.4 Quadrature Clock Generator

The quadrature clock generator is used to generate two clock signals, 90 degrees out of phase. The initial clock signal is generated by a 50 MHz crystal oscillator. The signal is split and fed into two 74F86 XOR gates. One is configured as a buffer and the other as an inverter. This is to keep the propagation delays the same for both signals. These signals are wired to two 74F74 D flip-flops, configured as toggle flip-flops. The clock signals are connected to the clock inputs. The $\overline{D}$ output is connected to the Q input. Since one of the clocks is inverted, one flip-flop is triggered on the rising edge while one is triggered on the falling edge. This creates two 25MHz clock signals, 90

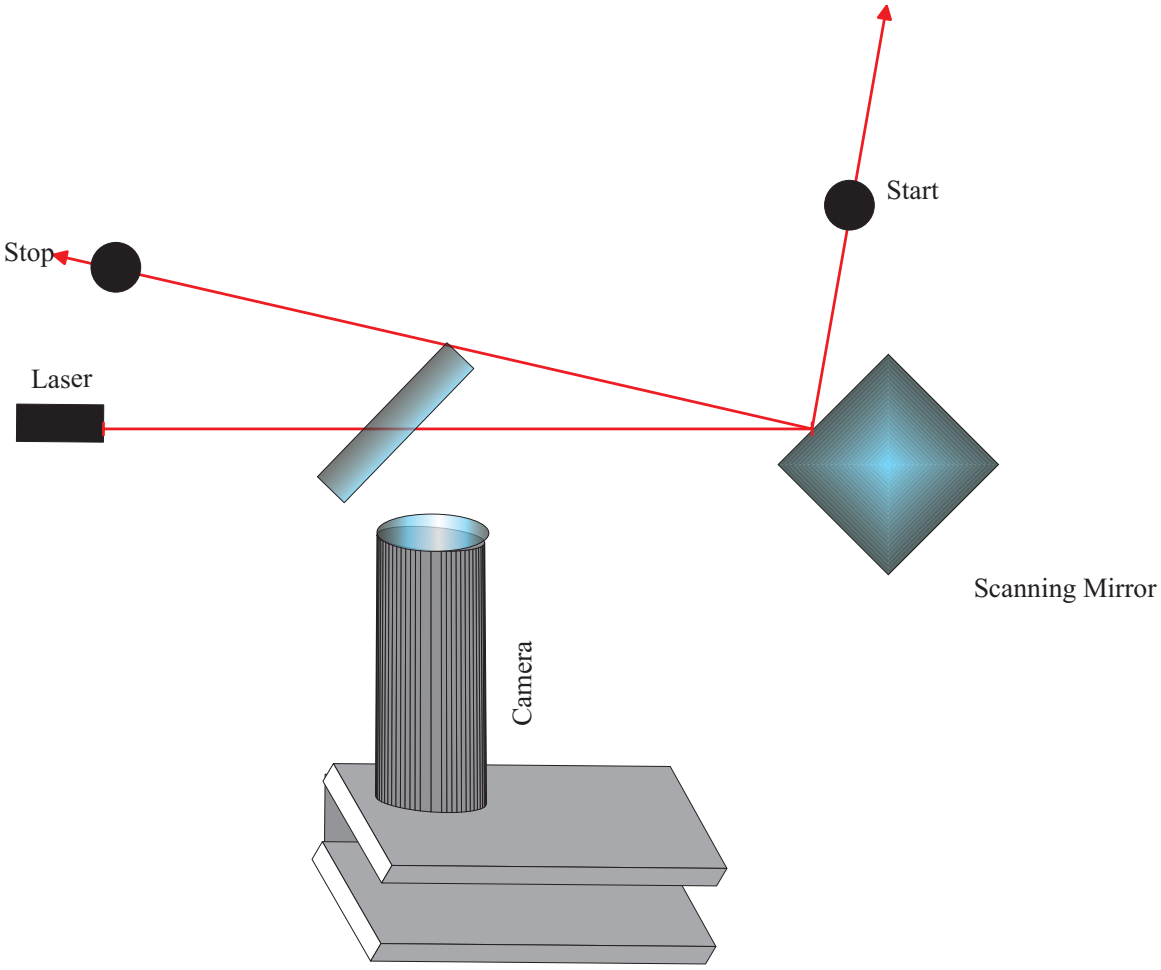

Figure 3-7: Camera Optics with Hole in Mirror and Coplanar Camera and Laser

degrees out of phase, thus providing our I and Q references. The clock signals are then buffered by using the unused XOR gates. One in-phase(I) clock signal is connected to a differential line driver to drive a 50Ω cable to the laser. One in-phase(I) and one quadrature(Q) signal are also coupled to the multipliers for demodulation by two toroidal transformers, avoiding a direct ground connection thus providing the least amount of noise and clock feed through.

3.2.5 Demodulator

The demodulator is the key to the quadrature phase detection circuitry. It is where the high and low frequency sections of the board meet. A photo of the completed

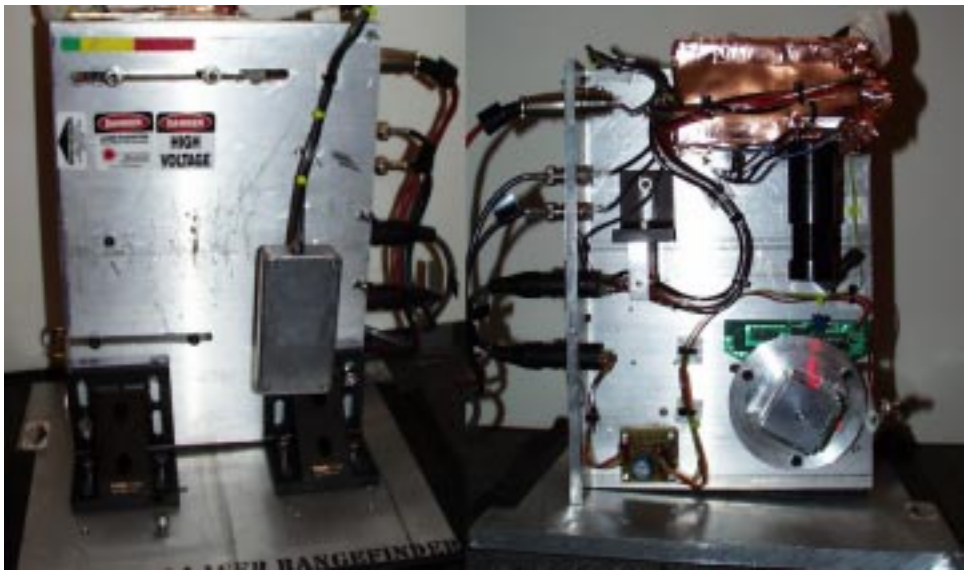

Figure 3-8: Scanning Optics, Front and Back Views

circuit appears in Fig. 3-9. The circuit is relatively simple. It consists of two Burr-Brown MPY600 analog multipliers and a pair of low-pass filters. The in-phase and quadrature reference signals from the clock generator are terminated with 50Ω resistors following the toroidal transformers, as mentioned above. One of the multipliers is fed with I and the other with Q. The signal from the APD camera is also connected to both multipliers. This will modulate the photodiode signal to baseband and twice the fundamental. A low-noise, low-pass filter stage follows the multipliers to select the baseband signal. An Analog Devices OP-297 low-drift amplifier is configured as the low-pass, with a capacitor and resistor in feedback. The positive terminals of the opamps are set by two potentiometers, determining the DC offset of the signal. The cutoff frequency was set fairly low(approximately 500Hz) to pass the lowest amount of noise, but high enough to enable a 30Hz scan rate. In running, this cutoff could probably moved higher, as the optimized layout provided a very clean signal. The photodiode output is directly demodulated to baseband in this design. This allows a simple circuit, but has the added problem of not allowing another AGC to be run at an intermediate frequency. An improvement of the design would demodulated the signal to an intermediate frequency and then again to baseband. The Burr-Brown

Figure 3-9: Demodulation Electronics Rack Unit

multipliers, however, were seen to work well enough in this application to allow the direct-to-baseband mode.

3.2.6 Voltage Controlled Amplifier

The demodulated signals are sent to a voltage-controlled amplifier stage. It is built out of an Analog Devices SSM2164 quad, matched voltage-controlled amplifer. The wiper of a $10K\Omega$ potentiometer is used as the gain control voltage. The SSM2164 outputs current and requires an opamp configured as a transimpedence amplifier on the output to produce a voltage. A Maxim MAX474 is used. The MAX474 is a single supply opamp and can run 0 to 5 volts, making it idea for interfacing to a microcontroller. The VCA allows both I and Q gains to be simultaneously adjusted, preserving their ratio, which determines range.

3.2.7 Microcontroller

A Hitachi SH-1 7032 microcontroller running at 20MHz is used to interface to a host. Its 32-bit architecture, along with its 10-bit analog-to-digital converters and its low cost make it an ideal solution. For ease of intergration, a Hitachi evaluation board was used. It contains circuitry for all of the RS-232 level conversion as well as the reset. The I and Q channels are connected to the microcontroller's A/D inputs, along with the start and stop phototransistor signals as logic levels.

3.2.8 Embedded Code

The embedded code was written in C for the SH-1 micrcontroller. It was compiled and burned into an EPROM. The code is structured in such a way as to utilize many of the functional blocks of the SH-1. The I and Q channels are digitized by configuring the analog-to-digital converter of the SH-1 to scan-convert channels one and two. Once configured and started, the scan converter will sample at the fastest speed, continually updating the registers with the newest results. Upon reset, the microcontroller begins scanning the two analog channels, determining a valid scan interval to be between the "start" and "stop" phototransistor signals. It first takes a calibration scan, averaging all of the samples to determine a baseline for the signals. It then sits in a holding mode, waiting for the configuration information from the host. The controller communicates to the host using RS-232 serial protocol, sending 8 bits without parity, at a 19200 baud rate. The angle is taken as the time on a 16-bit timer that has elapsed since the "start" phototransistor pulse. The timer is reset before each scan.

A 'T' character is sent from the host, signifying the setting of the threshold. Two bytes are then sent for the threshold value, high byte first. A 'W' character is sent from the host to signify that the following 2 bytes determine the wait time. This is the time after a valid start signal before data should be considered. This makes sure that the reflection from the phototransistor housing isn't contrived to be a data point. If a 'D' is sent, the processor enters into a debug mode. This will print all of the

results in ASCII, allowing the terminal program to be used as a debugger. In order to reduce the number of bytes sent, they are normally transmitted in binary. Once all of the desired configuration is set, the host will send an 'R' signifying the run mode. This allows the unit to run continually, analyzing data while scanning, and sending out the serial data when the laser is out of the scan range (after the stop and before the next start). While in the scan range, a simple peak finder and data reduction algorithm is running. The reduced data stream, containing only parameters for valid reflection points, is then packetized and sent out as RS-232 serial data.

Data Reduction and Peak Finding

The rangefinder is normally set up as shown in Fig. 3-10, where a matte-black baffle surrounds the projection area. Very little laser light returns from this surface, allowing hands to readily stand out as amplitude peaks that can be easily discriminated by a fixed threshold(Fig. 3-11). A simple peak finder runs while the scanner is in the scan range. A peak in the data is determined by a Manhattan metric, i.e., the sum of the absolute value of both the in-phase and quadrature signals. When the sum goes above the threshold level, a boolean is set signifying that you are now in a peak. The time of the start of the peak is noted. While in the peak, the inphase and quadrature signals are accumulated. When the Manhattan discrimator drops below the threshold, an out-of-peak state is set. The number of peaks is incremented and the data is stored. The time of the highest point of the peak, the time start of the peak, the ending time of the peak, the maximum values of I and Q, along with the total sum of the I and Q channels are stored. The maximum number of discrete peaks is currently limited to 4 in the firmware, only to reduce the number of bytes that must be sent to the host after each scan (a faster data transfer would enable more peaks). Once all data is sent out, this process repeats. Rather than determine the hand locations from peaks in the reflected amplitude, they could be derived by discontinuities in the detected range sequence. Although this would remove the need for the black baffle around the screen, it would require more complicated embedded code, thus was not pursued in these applications.

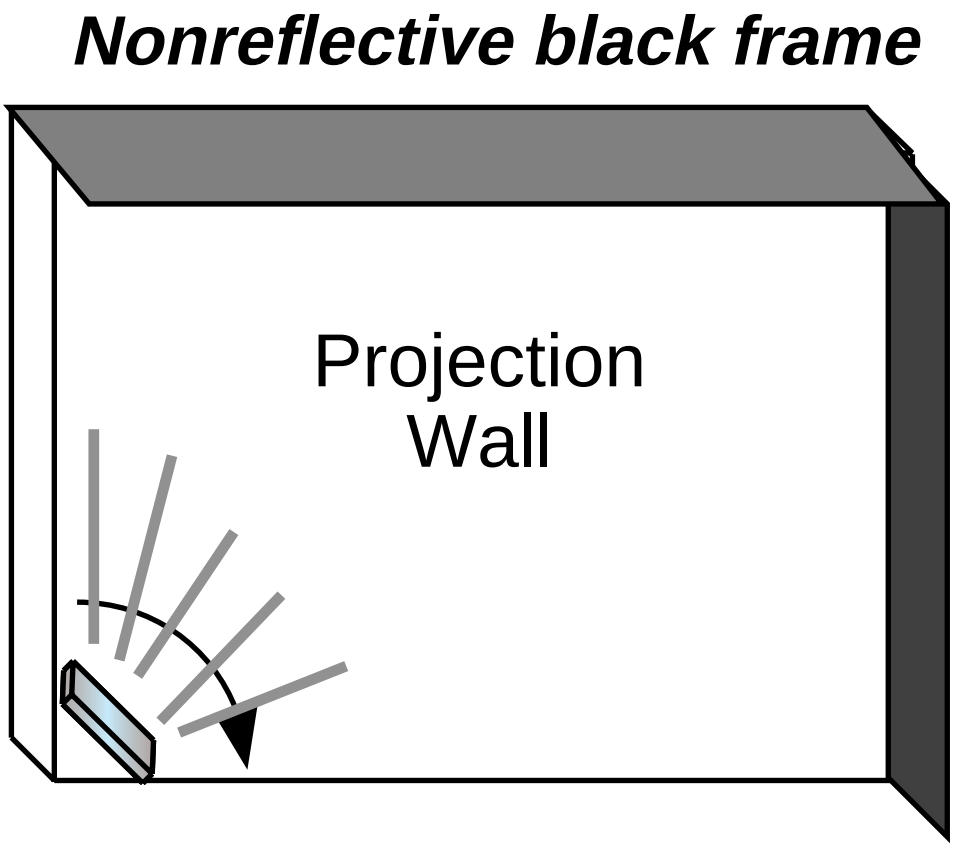

Figure 3-10: Rear-Projection Baffle Setup

Serial Protocol

The device, when running, continually streams data as summarized in Table 3.1. It contains a packet of 73 bytes proceeded by a unique 4-byte start header. As the chances of the unique combination are low, they provide a means of obtaining a reference frame within the data stream. The next byte following the header is the number of peaks, limited to a range of zero through 4. This allows us to only look at the appropriate number of bytes. The data stream contains a constant number of bytes as a simple solution, although a dynamically changing serial stream is also possible. Once the number of peaks is sent, 18 bytes are sent for each peak in the data. The first two are 16-bit quantities values for the maximum values of the I and Q channels in the peak. They are sent high-byte first, as with all of the data. The next three 16-bit values have to do with the timing of the peak. The first time

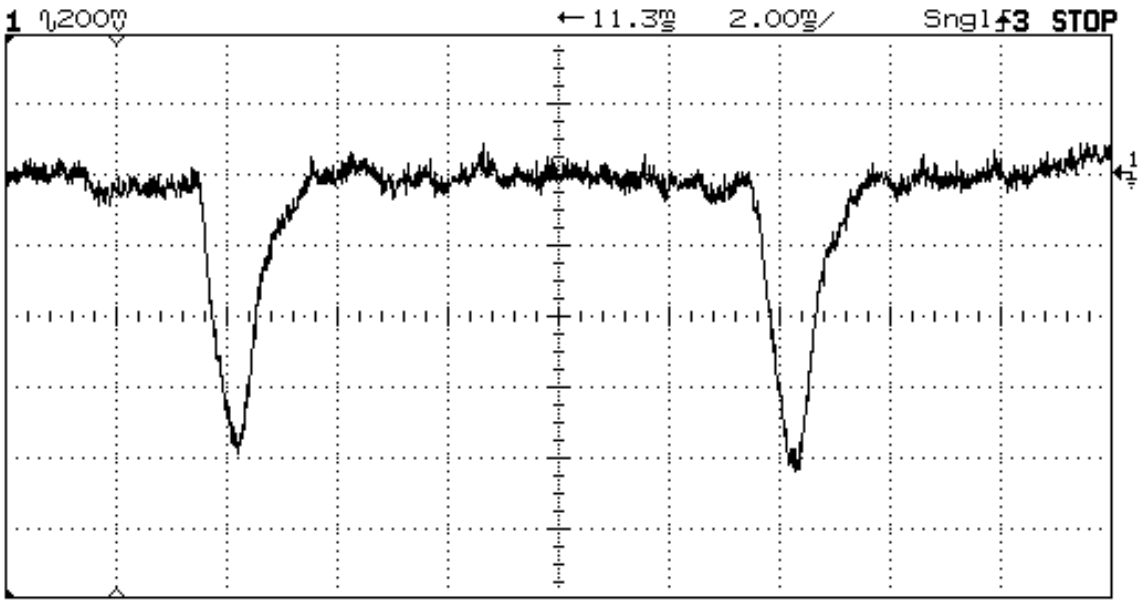

Figure 3-11: Scope Trace of One Range Channel with Two Hands

measurement is the time of the maximum peak height, followed by the start time and finishing with the end time. The start and end times allow one to compute a simple measurement of the width of the peak in the data. This gives a good reference as to the relative size of an object. The following two 32 bit values are the sum of the I and Q channels over the peak in the data. This is what is actually used to determine the range data, as the maximum point can be fairly noisy. There are a total of 4 such 18-byte packets sent to the host. If the number of peaks is less than four, the appropriate sections are ignored in the data stream. All of these bytes are sent as binary. In contrast, the debug mode sends full ASCII, allowing a simple terminal emulator to list the data along with printing various line-feeds and status markers.

3.3 Calibration Program

A graphical calibration program was developed. The purpose of the calibration program was to solve for the coefficients fitting functions that mapped the rangefinder data into graphics raster coordinates. This program calibrates the usable range as

well as any offsets and nonlinearities in r and θ. The program was developed on a PC using Windows NT and OpenGL[WND96][KF97]. A five-by-five grid of data was collected, where a marker was plotted on the screen, a hand placed into the scan beam at that position and data taken at each successive calibration point. For each of these twenty-five points, r and θ were computed from the rangefinder data; r was calculated by the following equation:

$$r = \arctan\left(\frac{\sum_{overpeak} I}{\sum_{overpeak} Q}\right) \tag{3.4}$$

where $\sum_{overpeak}$ is the accumulated data over a given peak, output directly from the rangefinder's serial port, and the signed, quadrature ARCTAN2 routine was used to map through 360° of phase. The averaged data over many samples for the 25 points was written to a file along with the expected screen coordinate values(r_{ex}, θ_{ex}). This file was read into MATLAB[*www.mathworks.com*] and the coefficients for various fitting functions were computed. These were then made available to other programs for mapping the rangefinder's coordinates to screen positions.

3.4 Numerical Methods

3.4.1 Linear Least Squares Fit

The raw data produced by the rangefinder is essentially in polar coordinates, using units of clock time for mirror angle(θ) and quadrature phase angle for range(r). A conversion formula is needed to translate this data into the appropriate screen raster coordinates used by the graphics routines. This can first be accomplished by fitting the measured r_{in} and θ_{in} to expected r_{ex} and θ_{ex} values computed from screen coordinates assuming the scanner at the origin. Using the calibration program described in the last section, a data set was written along with the corresponding expected screen coordinates in polar form. A MATLAB program was written (Lasercal.m), to try various kinds of fits to the data and solve for respective coefficients. The following

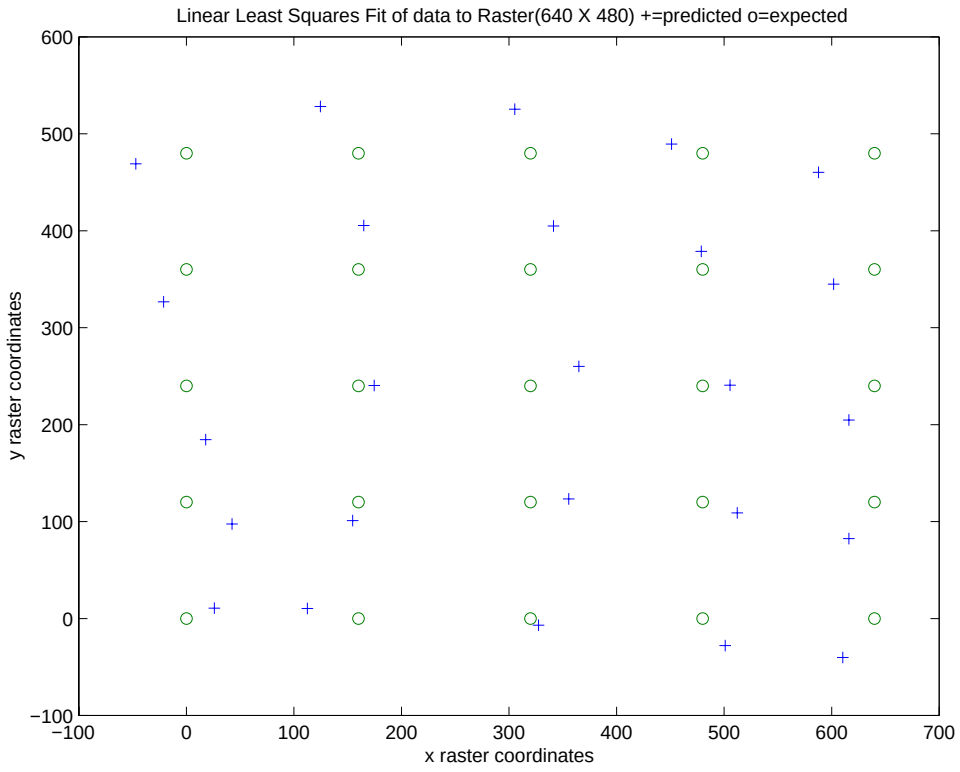

Figure 3-12: Linear Least Squares Fit $\sigma = 41$ VGA pixels across a 8' X 6' screen

equations are used as a basis for an initial linear least squares fit:

$$r_{out} = a_r * r_{in} + b_r * \theta_{in} + c_r \tag{3.5}$$

$$\theta_{out} = a_\theta * r_{in} + b_\theta * \theta_{in} + c_\theta \tag{3.6}$$

$$x = r_{out} * \cos(\theta_{out}) \tag{3.7}$$

$$y = r_{out} * \sin(\theta_{out}) \tag{3.8}$$

The MATLAB least-squares algorithm used this function to compute the best-fit values for r and θ in polar form. The variables a, b and c are solved independently for the r and θ equations. Once the best-fit r and θ were computed, a simple conversion to Cartesian screen space was performed. Fig. 3-12 shows a sample data set fitted to these functions, as taken with the rangefinder across a 8' X 6' screen. The σ value for this fit is 41 pixels, with the data spanning a 640 X 480 VGA grid. There is

some noticeable warp in the agreement from nonlinearities that this fit was unable to describe. Clearly this can be improved.

3.4.2 Nonlinear MATLAB Fit

In order to improve on the linear fit, a more complicated nonlinear fit is tried. This fit attempts to model some of the known sources of error and account for these. The nonlinear fit is accomplished by the following equations.

$$\begin{aligned} x &= [(r_{in} + r_{offset}) * S_r + S_{rq} * (r_{in} + r_{offset})^2] * \\ &\quad \cos[(\theta_{in} + \theta_{offset}) * S_\theta] - x_{offset} \end{aligned} \tag{3.9}$$

$$\begin{aligned} y &= [(r_{in} + r_{offset}) * S_r + S_{rq} * (r_{in} + r_{offset})^2] * \\ &\quad \sin[(\theta_{in} + \theta_{offset}) * S_\theta] - y_{offset} \end{aligned} \tag{3.10}$$

with r_{in} and θ_{in} directly produced by the rangefinder and all other variables adjusted by the fit. The first thing that must be accounted for is the X and Y displacement of the scanner from the screen. This is easily accounted for by solving for an offset (x_{offset}, y_{offset}) in each of these variables. Once the scanner is translated to the origin, we can examine the errors in r and θ. Most of our nonlinear errors will occur in the r value, as the θ values are directly taken from an accurate 16-bit counter on the microcontroller. The θ value must simply be mapped to the appropriate scan angle (e.g., 0 to 90°). This is accomplished by adding an offset and scaling the sum by an appropriate number such the desired range of angles is computed. A similar fit is accomplished with r except for the addition of a second-order term. The hope is that this will remove some of the warping due to quadrature errors of the electronics. The various coefficients were solved for using the MATLAB *fmins* optimization routine and the results are shown in Fig. 3-13. The σ is somewhate less than the linear fit, and can still be improved upon. Additional perturbations arising from amplitude and phase errors in the quadrature demodulation, bias errors, residual crosstalk and unmodelled geometry must be taken into account.

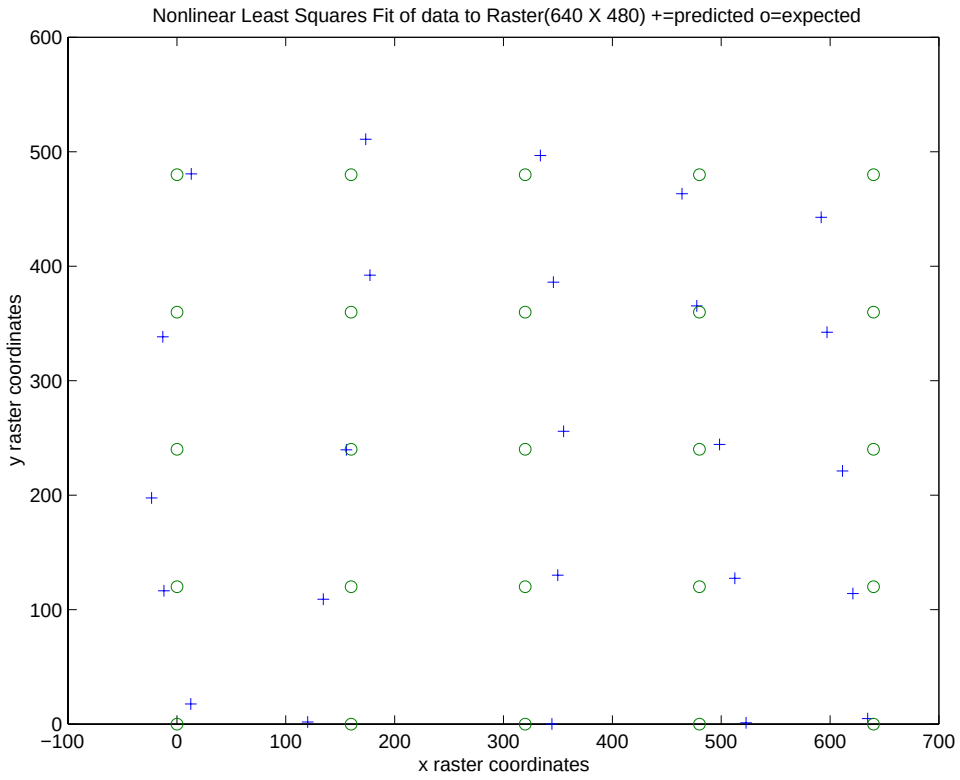

Figure 3-13: Nonlinear Least Squares Fit $\sigma = 32$ VGA pixels across a 8' X 6' screen

3.4.3 Polynomial Fit

As one begins to account for more sources of possible errors, the modeling parameterization grows more complex and one may begin to doubt the ease at which fitting the data will possible. As you cannot explicitly account for every possible variation, one idea is to fit the data to a polynomial. The following equation is the generic form for all of the basis terms for an N-dimensional polynomial of order M.

$$y(\vec{x}) = \sum_{i=1}^{M} f_i(\vec{x}; \vec{a}_i) \tag{3.11}$$

The question then becomes what order is necessary and how to find the coefficients? This problem translates to one of searching. One possible search algorithm is cluster-weighted modeling[Ger99][GSM]. Cluster-weighted modeling(CWM) can be used to fit a probabilistic model to the data. In this case, a 2-dimensional global polynomial

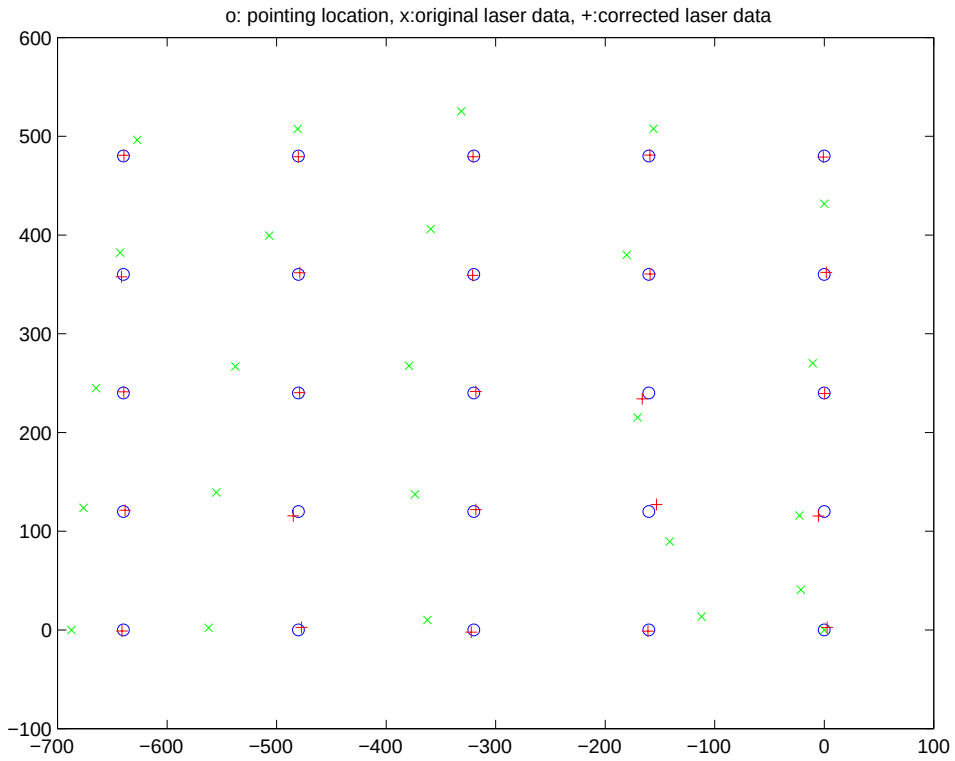

Figure 3-14: Fifth Order Polynomial Fit $\sigma = 4$ VGA pixels across a 8' X 6' screen

is to be fit to the set of data taken as described earlier. A cluster-weighted modeling package, developed by Bernd Schoner at the MIT Media Lab, was used in conjunction with MATLAB to find various orders of polynomials. It was determined that a fifth order polynomial was satisfactory for our data set. The data was fit using one cluster, solving for a global polynomial for the entire parameterized space. The results can be seen in Fig. 3-14, where σ is now 4 VGA pixels, significantly less than the other fits. In order to use CWM, the data must first be transformed to zero mean and unit variance in x and y. The following equations are used to transform the raw data from

the rangefinder:

$$\theta_{scaled} = \frac{(\pi/2) * \theta_{in}}{(\theta_{max} - \theta_{min})} \tag{3.12}$$

$$x_{init} = r * \cos(\theta_{scaled}) \tag{3.13}$$

$$y_{init} = r * \sin(\theta_{scaled}) \tag{3.14}$$

$$x_{pre} = (x_{init} - \overline{x_{data}})/\sigma_{xdata} \tag{3.15}$$

$$y_{pre} = (y_{init} - \overline{y_{data}})/\sigma_{ydata} \tag{3.16}$$

The raw angle is first mapped to the range of 0 to 90° by the first equation, where the difference between the maximum and minimum θ values of the 5-by-5 data set are used to scale the angles. A raw value for x and y can then be computed (Eqs. 3.13 and 3.14) using the raw r_{in} value produced by the rangefinder. The x and y values are then converted to zero mean, unit variance in Eqs. 3.15 and 3.16. The data is now ready for the CWM engine to find the coefficients of the polynomial. For consistency, similar operations are performed on the expected screen position coordinates. The following equations are expanded forms of the generic polynomial equation for a 2-dimensional, fifth-order fit:

$$\begin{aligned} x_{post} = {} & A_x x_{pre}^5 + B_x x_{pre}^4 + C_x x_{pre}^3 + D_x x_{pre}^2 + E_x x_{pre} + F_x + \\ & G_x x_{pre}^4 y_{pre} + H_x x_{pre}^3 y_{pre} + I_x x_{pre}^2 y_{pre} + J_x x_{pre} y_{pre} + K_x y_{pre} + \\ & L_x x_{pre}^3 y_{pre}^2 + M_x x_{pre}^2 y_{pre}^2 + N_x x_{pre} y_{pre}^2 + O_x y_{pre}^2 + \\ & P_x x_{pre}^2 y_{pre}^3 + Q_x x_{pre} y_{pre}^3 + R_x y_{pre}^2 + \\ & T_x x_{pre} y_{pre}^4 + U_x y_{pre}^4 + \\ & V_x y_{pre}^5 \end{aligned} \tag{3.17}$$

$$
\begin{aligned}
y_{post} = {} & A_y x_{pre}^5 + B_y x_{pre}^4 + C_y x_{pre}^3 + D_y x_{pre}^2 + E_y x_{pre} + F_y + \\
& G_y x_{pre}^4 y_{pre} + H_y x_{pre}^3 y_{pre} + I_y x_{pre}^2 y_{pre} + J_y x_{pre} y_{pre} + K_y y_{pre} + \\
& L_y x_{pre}^3 y_{pre}^2 + M_y x_{pre}^2 y_{pre}^2 + N_y x_{pre} y_{pre}^2 + O_y y_{pre}^2 + \\
& P_y x_{pre}^2 y_{pre}^3 + Q_y x_{pre} y_{pre}^3 + R_y y_{pre}^2 + \\
& T_y x_{pre} y_{pre}^4 + U_y y_{pre}^4 + \\
& V_y y_{pre}^5
\end{aligned} \tag{3.18}
$$

The CWM engine has the ability to solve for all 42 coefficients. Once the fit converges, the data can be mapped to the predicted model. As the data is now zero mean, unit variance, this is accomplished by a reverse operation of scaling the data first and then adding the offset, this time by the mean and σ values of the expected output space. The data is now mapped to the screen coordinates by the following equations:

$$x_{screen} = x_{post} * \sigma_{xscreen} + \overline{x_{screen}} \tag{3.19}$$

$$y_{screen} = y_{post} * \sigma_{yscreen} + \overline{y_{screen}} \tag{3.20}$$

This method can aptly produce the best results as it has the mathematical language to describe all of the local maxima and minima of the data set. It potentially has the problem of overfitting, since as order of the polynomial increases, the model can have many more basis terms than data points. The fifth-order fit seems to describe the scanner's response adequately, however. The predicted model is going to be integrated with a new two-handed application that is currently being developed(See Chapter 4).

3.5 Performance

The performance of the rangefinder did achieve our specifications. From the data, we were able to estimate that the overall resolution met our need for millimeter accuracy. The range did extend out to the full 6 m limit imposed by the modulation frequency; in actuallity, the camera, once aligned, was able to see out much further. With changes to the optics and a better mirror, the accuracy could be improved. Wobble

in the scanning optics created apparent noise, which was attenuated by an averaging filter running on the host computer. The scan rate did attain 30Hz, limited by the demodulation filter cutoff along with the size of the packet transmitted after each scan. A faster communications path could have been established to send more data to the host. The currently attainable 30Hz rate was fast enough for our applications, however. It is an interesting problem in deciding how much processing the host must do. It wouldn't be inconceivable to have all of the embedded processing running on the host, requiring only two fast sampling channels.

The radial scan also adds limitations, since the laser beam moves faster as it tracks farther from the source. The level of resolvable detail thus decreases with range, not allowing smaller objects to be detected far away. Although our system could find finger-sized objects at a 3-4 meter radius when scanning at 30Hz, extended range could be attained by slowing the scan rate or raising the demodulation filter cutoff (potentially passing more noise, however).

The parts cost of the system was below $500. This could be reduced further, with additional integration. In addition, more careful design of the front-end electronics and possibly the introduction of an IF gain stage could remove the necessity of the avalanche photodiode (which dominates the parts cost); allowing a much less expensive PIN photodiode to be used in its place.

An important consideration is also the safety of such a system. The 5mW laser diode had a 50% duty cycle. This, along with the active dynamic power control of the laser, should be enough to make it safe for brief periods of eye exposure[Weh97]. The visible laser added the intrinsic ability of being extra careful in keeping it away from the eyes, as the flash is noticeable.

header byte 0	1			
header byte 1	27			
header byte 2	48			
header byte 3	187			
number of peaks	0, 1, 2, 3 or 4			
Inphase Peak 0	byte 1	byte 0		
Quadrature Peak 0	byte 1	byte 0		
Time Peak 0	byte 1	byte 0		
Time Start 0	byte 1	byte 0		
Time End 0	byte 1	byte 0		
Inphase Sum 0	byte 3	byte 2	byte 1	byte 0
Quadrature Sum 0	byte 3	byte 2	byte 1	byte 0
Inphase Peak 1	byte 1	byte 0		
Quadrature Peak 1	byte 1	byte 0		
Time Peak 1	byte 1	byte 0		
Time Start 1	byte 1	byte 0		
Time End 1	byte 1	byte 0		
Inphase Sum 1	byte 3	byte 2	byte 1	byte 0
Quadrature Sum 1	byte 3	byte 2	byte 1	byte 0
Inphase Peak 2	byte 1	byte 0		
Quadrature Peak 2	byte 1	byte 0		
Time Peak 2	byte 1	byte 0		
Time Start 2	byte 1	byte 0		
Time End 2	byte 1	byte 0		
Inphase Sum 2	byte 3	byte 2	byte 1	byte 0
Quadrature Sum 2	byte 3	byte 2	byte 1	byte 0
Inphase Peak 3	byte 1	byte 0		
Quadrature Peak 3	byte 1	byte 0		
Time Peak 3	byte 1	byte 0		
Time Start 3	byte 1	byte 0		
Time End 3	byte 1	byte 0		
Inphase Sum 3	byte 3	byte 2	byte 1	byte 0
Quadrature Sum 3	byte 3	byte 2	byte 1	byte 0

Table 3.1: Serial Data Stream

Chapter 4

Applications

The phase-measurement rangefinders were used in variety of multimedia demonstrations. The second prototype was used in three different applications, most of which were in large, public spaces. It was interesting to see the results of average users interacting with the device. The applications demonstrated that a clear need is being filled by such a system. Although the rangefinder can track multiple nonoccluding hands, in many cases it was used as a mouse substitute, and not exploited to its full capabilities. The system was shown internally at the MIT Media Lab as well as publicly at SIGGRAPH 98[SRP98]. The laser rangefinder was set up in front of a rear-projected screen, as shown in Fig. 4-1. The projector used was an Infocus Light Pro 720 together with a 6' X 8' rear projection screen, and the host computer used was a Pentium II 266 Mhz. The rangefinder camera head was connected to a rackmount unit containing the digital hardware and demodulator board. The rackmount unit sent serial data to the host PC.

4.1 It Works!

The first use of the prototype rangefinder was in a simple multimedia installation(see Fig 4-2)[SP98]. The phase-measuring prototype was used with a Microchip PIC 16C73 microcontroller interface. Although the acquired data had lower resolution than the final design, the system worked very well. In the simple application, rotating

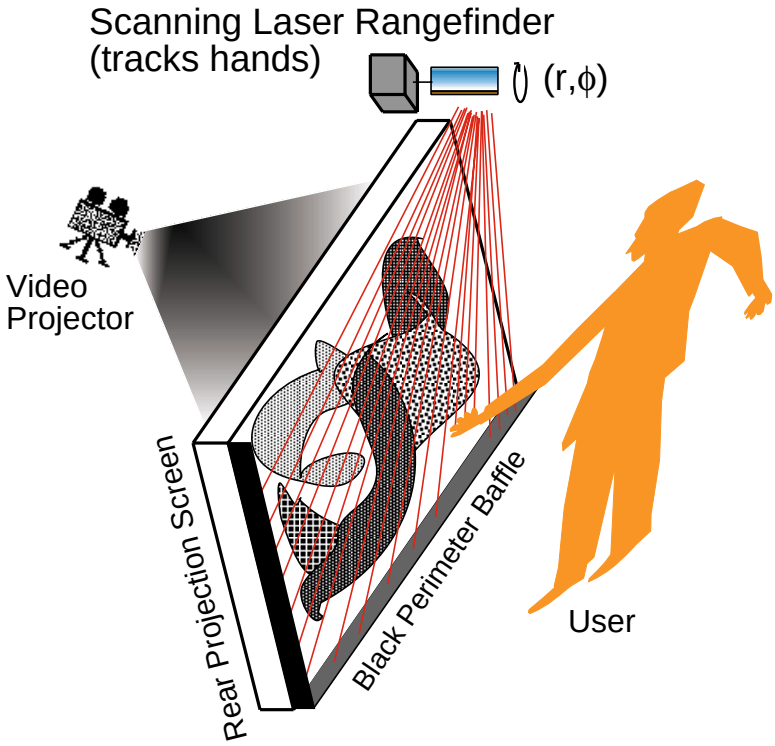

Figure 4-1: Laser Wall Installation Diagram

multicolor squares were plotted at the detected hand positions. This also worked as a music controller for which the screen was divided into four sections. When the presence of a hand was detected in each section, a drum loop was triggered. If more than one hand was detected in the sensing plane, an bass-lead arpeggio was started. The range in pitch of the apeggio would increase with the distance between the hands. This installation was displayed at several Media Lab events and was well received. This first software did very little analysis of the data. Further research followed to develop algorithms for successfully mapping the data points to raster coordinates more accurately.

4.2 Stretchable Music

Stretchable Music [Ric98] is a interactive graphical music program, developed by Peter Rice, at the MIT Media Lab. It allows a user to manipulate a scored piece of music

Figure 4-2: First Musical Installation Tracking Two Hands

by stretching and pulling various parts of graphical objects. A composed piece of music plays while various objects appear on the screen. When the objects are on the screen, they can be manipulated by stretching and pulling on various control points. By manipulating the objects, you can change various parameters of the music. Each object represents a different musical layer in the piece. The system was developed as a tool to allow users to control how a piece of music can be played back. It was initially developed for a mouse but was later integrated with the laser rangefinder. The second phase-measurement prototype was connected with the system. The laser interface allowed for multiple users, but, since this was a simple mouse port, the active cursor was simply the first peak seen in the scan of the data (e.g., the lowest hand, first detected in the scan); additional hands generated a set of twinkling points that tracked as well as added an ambient sound. The installation was setup at SIGGRAPH

Figure 4-3: Stretchable Music Installation

98 for a period of one week[Mah98]. The audience response was extremely positive, although some people initially confused the interface for a touch pressure screen. It had to be explained that one simply must break the beam of the laser with the hand and nothing more. In fact, using a visible red laser as opposed to an invisible IR beam proved to be an advantage, as the users could see the beam on their hands when they were being tracked. As the system attracted people of all races, we were able to attest to its adequate tolerance across skin complexions.

4.3 Mouse Pointer

A mouse-pointer application was developed using the rangefinder to drive the Windows system. It sends virtual mouse commands through a windows system call. The laser mouse program uses the same calibration routine as that of *Stretchable Music*.

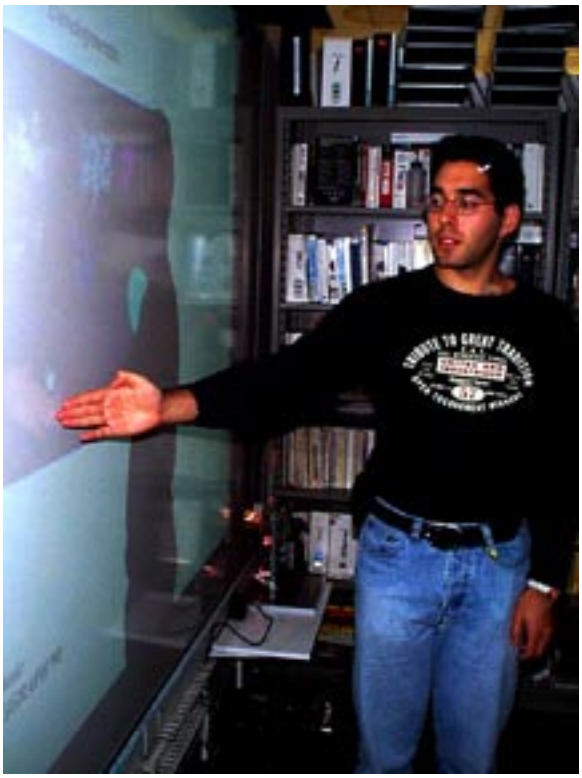

Figure 4-4: Laser Mouse Program

After calibration, the program is set into a run mode, which simulates mouse clicks. For the first application, it was only to be used as a simple interface, e.g. to bring up web pages, drive drawing programs, etc.(Fig 4-4). When a user inserts his or her hand into the path of the laser, a virtual single mouse click is sent to the system. This allows a user to click, or click and drag, but not double click. It is important to consider that this new interface offers other abilities that a mouse is lacking. As it isn't a mouse, gestures like double-clicking are not natural in this environment (although they could be incorporated in a number of ways, e.g. "double-clicking" with the hand, moving in a second hand, etc). The rangefinder can detect the width of the peak in the data. That measurement relates to the size of the object and its range in the scanned plane. This could be used, for instance, for reaching in a 3rd dimension. The user interface of Windows could then incorporate some sense of the 3rd dimension in the desktop. The rangefinder also tracks multiple hands. This

too is also not addressed by the Windows interface. It is like having multiple mice. You could then be allowed to do true multi-tasking by operating on two areas of the screen. Clearly, the rangefinder is not a mouse and will not replace a mouse, but for large screens, it offers an ability to reach out and touch the objects that you are interested in. Also, there is clearly a new area of Human Computer Interface that can be addressed in software as to how these new, immersive, multi-parameter controllers can fully be exploited. Other user interfaces enabled by this device will also give new abilities. The expanded capabilites of the laser rangefinder is being further tested in a application specifically designed for multiple hands, discussed below.

4.4 Two Handed Approaches

The use of two hands is being addressed by the development of a new musical application. Although the application has not been completed, with the addition of the polynomial fit, as well as some new tracking algorithms that manipulate the user to avoid occlusion, the interface will be pushed to its limits. The application will take a form similar to the *Stretchable Music* application; as a rear-projection musical installation. The focus will be likewise that of a musical instrument. The ability to track multiple points will be a primary goal of research. The ability to accurately track the hands will allow the successful use of gesture recognition technology. Planar, large-area gesture can now easily and accurately be tracked, enabling a sensitive surface that can incorporate human emotion in a new electronic musical context.

Chapter 5

Conclusions

Engineers are often are unaware as to the contribution they are truly making to a field until their invention reaches the hands of the public. Such has been the case in developing the laser rangefinder for the user interface community. It wasn't until the rangefinder was demonstrated at the Media Lab for a technical and computer-human interface community that I began to understand the uniqueness of such a device. That alone didn't fully prove its worth. Demonstrating it at SIGGRAPH '98 brought the device out to a more general audience, technically adept in many fields, but new to many of the ideas of HCI design. Computer programers have been plagued by the mouse and keyboard interface that is standard on today's computers. That is changing, and programmers will have to think of how to effectively use the emerging hardware. The laser rangefinder is such a device. It is bringing us closer to a more natural way of interacting with a computer. The currently-designed version is only the first step, and other versions and modifications can be forseen, opening up even more applications.

5.1 Mini Time of Flight

The final rangefinder that was developed was mostly analog in its construction. This required the use of special components rated at a fairly high frequency. The cost of such components is significant. It also requires careful planning in the layout of the

printed circuit boards, to avoid ground loops, crosstalk, oscillation and other effects. A solution to such a problem is to design a device that could be more fully digital, possibly fitting mostly into one application-specific IC or field-programmable gate array. Such a solution could be attained by developing a miniature time-of-flight rangefinder. The time-of-flight rangefinder relies on a simple principle; measure the time it takes a short pulse of laser light to leave the device, hit an object, and return. Most of the hardware could be developed in an FPGA. All that is otherwise needed is a high-gain photodiode receiver and a laser pulser. The laser pulser can simply be a logic line pulsing a fast driver such as the one we already have. The receiver can also be similar to what exists in our current circuit, with the simple addition of a high-pass filter and digital discriminator after the front end amplifier. The only problem that exists is that of resolution. Since light travels at $3x10^8 m/s$, in order to measure with an accuracy of one meter you need to clock the counter with a 150Mhz input(the divide-by-two comes from the path back as well as to the object; measuring a meter in range requires a travel of two meters). If one wanted to clock with centimeter resolution, the clock rate rises to 100 times that frequency, which may be somewhat unreasonable. Many systems that use this technique employ two phase-locked clocks, thus reducing the number of bits necessary for the fast clock[BS91]. You can then use the coarse clock for lower resolution and then add the counts of the fine clock. Other systems use a simple time-stretcher circuit[VKY+97], quickly charging a capacitor during the interval to be measured, then timing how long it takes it to be fully discharged at a much lower current (hence longer interval). These overall methods are simple, but mostly require very fast logic. Such TOF systems are most often used in cases were higher precision isn't necessary, e.g., in military rifle sights[Urb95] or spacecraft remote sensing[Col96].

5.2 Microscanners

The next improvement that could be addressed is that of the scanning and optical assemblies. They can easily be made much smaller. There are two levels of minia-

turization. The first is simply to adopt a smaller, more integrated form, (e.g. in analogy to the optical assembly of a CD player). By having a fully integrated design, the optical path can be colinear. They can also be made in such a way that manual alignment will be minimized. This is critical in fabricating a viable product. It is also important in making the system the simplest to assemble. This miniaturized system could then take advantage of smaller scanners such as galvonmeters. These scanners can only support a small mirror and thus can image a smaller area. This isn't a problem if the mirror is closer to the optics and the optics are smaller(e.g. possible with more sensitive photonics or a more powerful laser). Such a system can be built such that everything fits into a package the size of a camera.

A still further miniaturization may be possible at the chip level. This could perhaps take form of a MEMS device[ML98], consisting of a scanner, laserdiode and photodiode. This system could also interface to a second ASIC that could contain all of the detection and driver circuitry. An installation could then counter the occlusion problem by easily introducing more scanners. These could all be networked in such a way that you can image not just a plane, but an entire room.

As the optical apertures are much smaller than in the device that we have built now, the laser illumination would have to be much brighter. As the pulse is only a few nanoseconds long in a time-of-flight laser rangefinder, this could be attained without damaging the laserdiode or harming eyesight; the average power is still very low.

5.3 3D

Scanning three dimensions is a next possible step. The computer graphics industry is currently using 3D laser rangefinders to scan complicated 3D models[PTR+98]. These systems are not being used for real-time HCI. They are often slow and have limited ranges, as they are mainly triangulation devices. As the components of TOF rangefinders become more integrated and cheaper, it is possible to forsee an inexpensive version that provides the ability to scan spaces. One possibility is simply to fix the optics of the current system such that the laser is colinear with the sensor, as

in Fig. 3-7. One simply attaches an x-y scanning assembly and you can scan 3D. The other possibility to create a network of simpler miniature scanners, as described above, and interpolate the data between them. This would create an installation that you can walk into. To get a sense of a 3D interface, you need not obtain uniform sampling points in space. One such idea is to use a scanning prism and simply wash over a space in some unique pattern. You could then create a sensing cone of light or other desired pattern. Another idea is to integrate the laser rangefinder with a scanning intelligent light, such that the projected spotlight became a sensor. You could then scan the light and thus the sensor around the room. This type of interface could be use for such applications as nightclubs and concerts, as a way of giving the attendees a coordinated method of interacting with the music and visuals.

Although this first-step goal of creating a two-dimensional scanner brought about a multitude of options and new abilities for the user, 3 dimensional scanning is clearly a next step in going further. When multiple users can gesture freely, it will be the duty of the software to make the best use of the data. It is often the case that providing better data alone will not give new capability. Part of the process of developing such an interface is to also develop compatible analysis and interpretation software and applications, as are only beginning to be explored.

Appendix A

Schematics

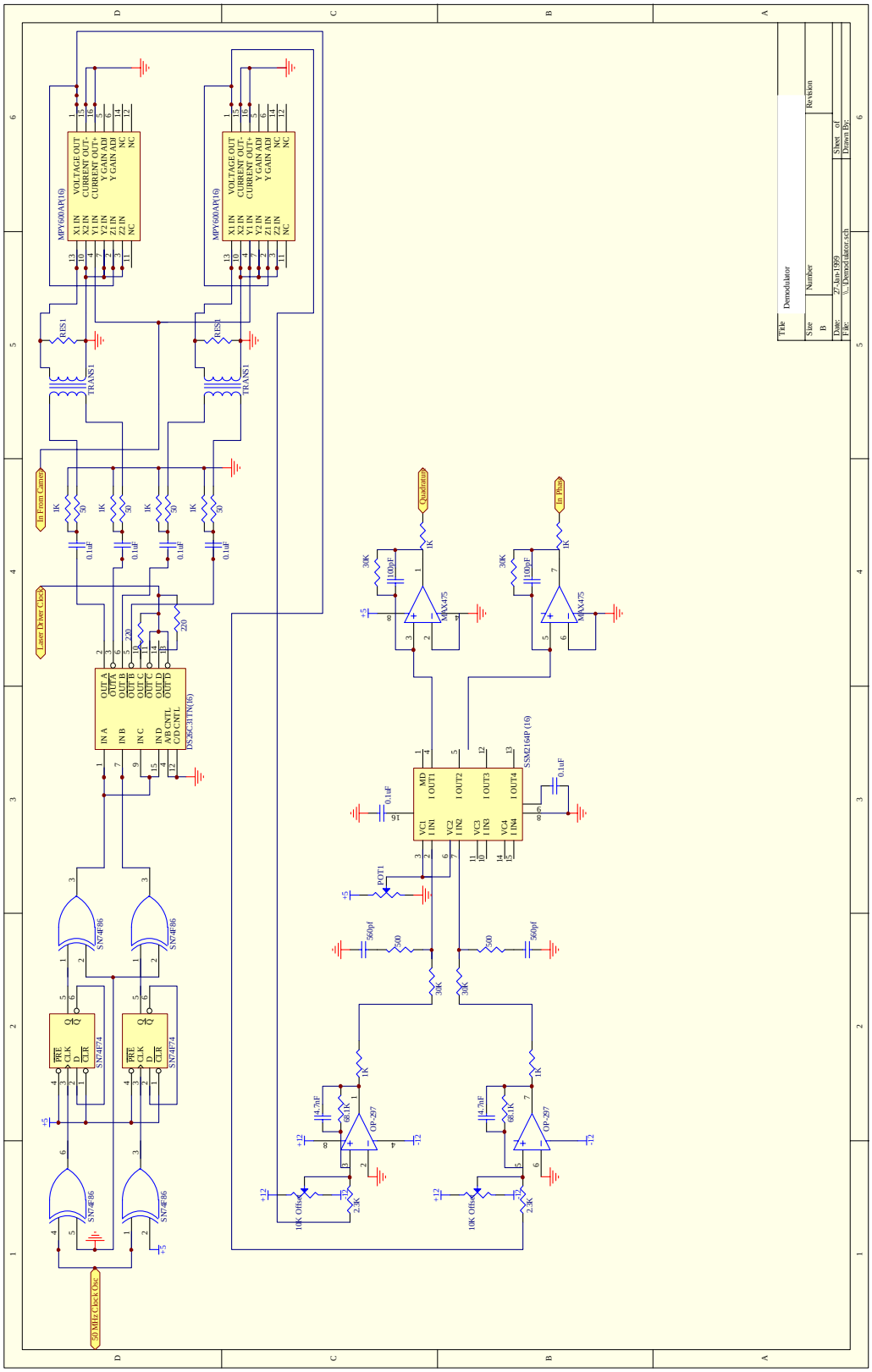

Figure A-1: Demodulator Schematic

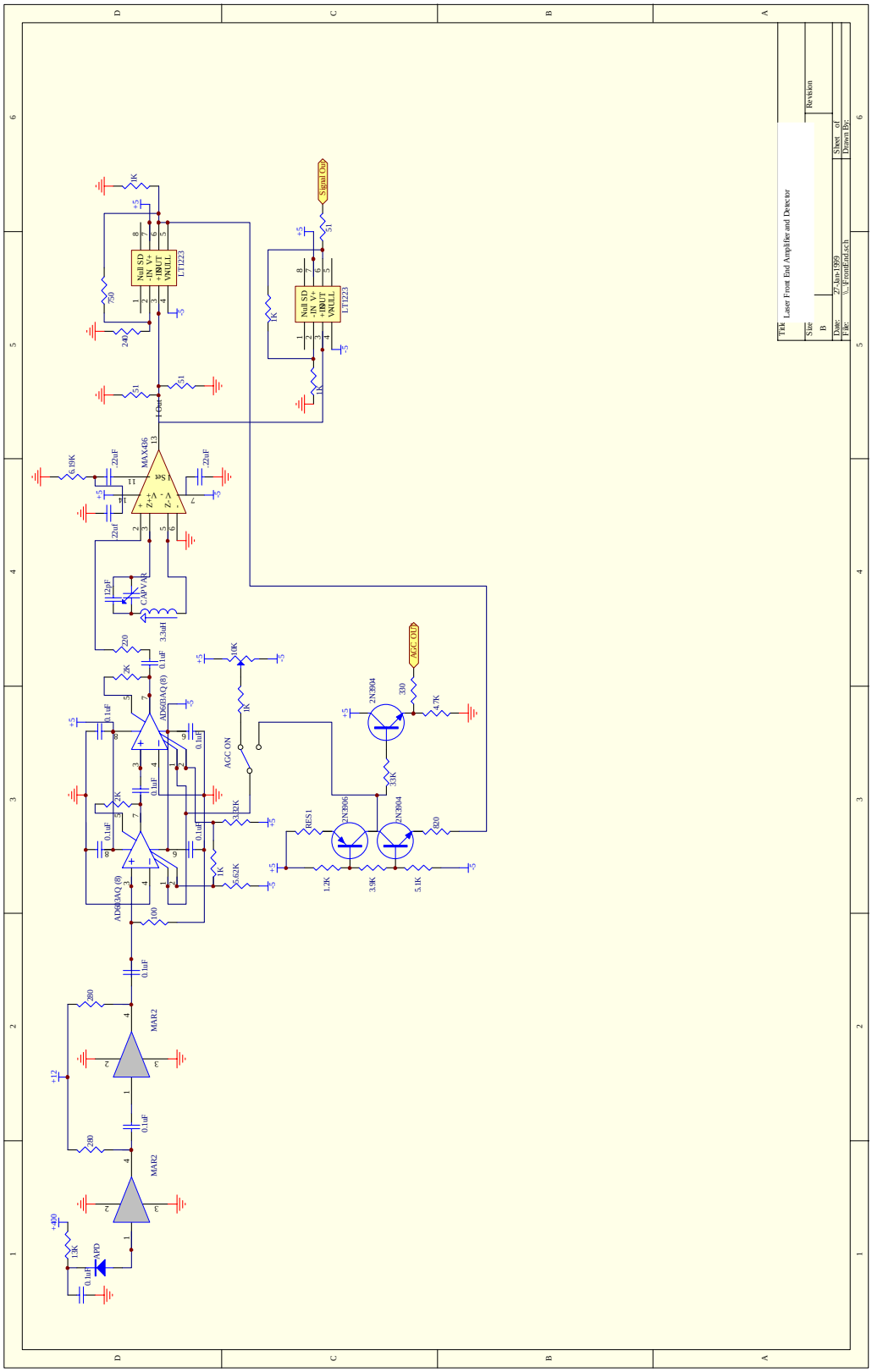

Figure A-2: Front End Amplifier Schematic

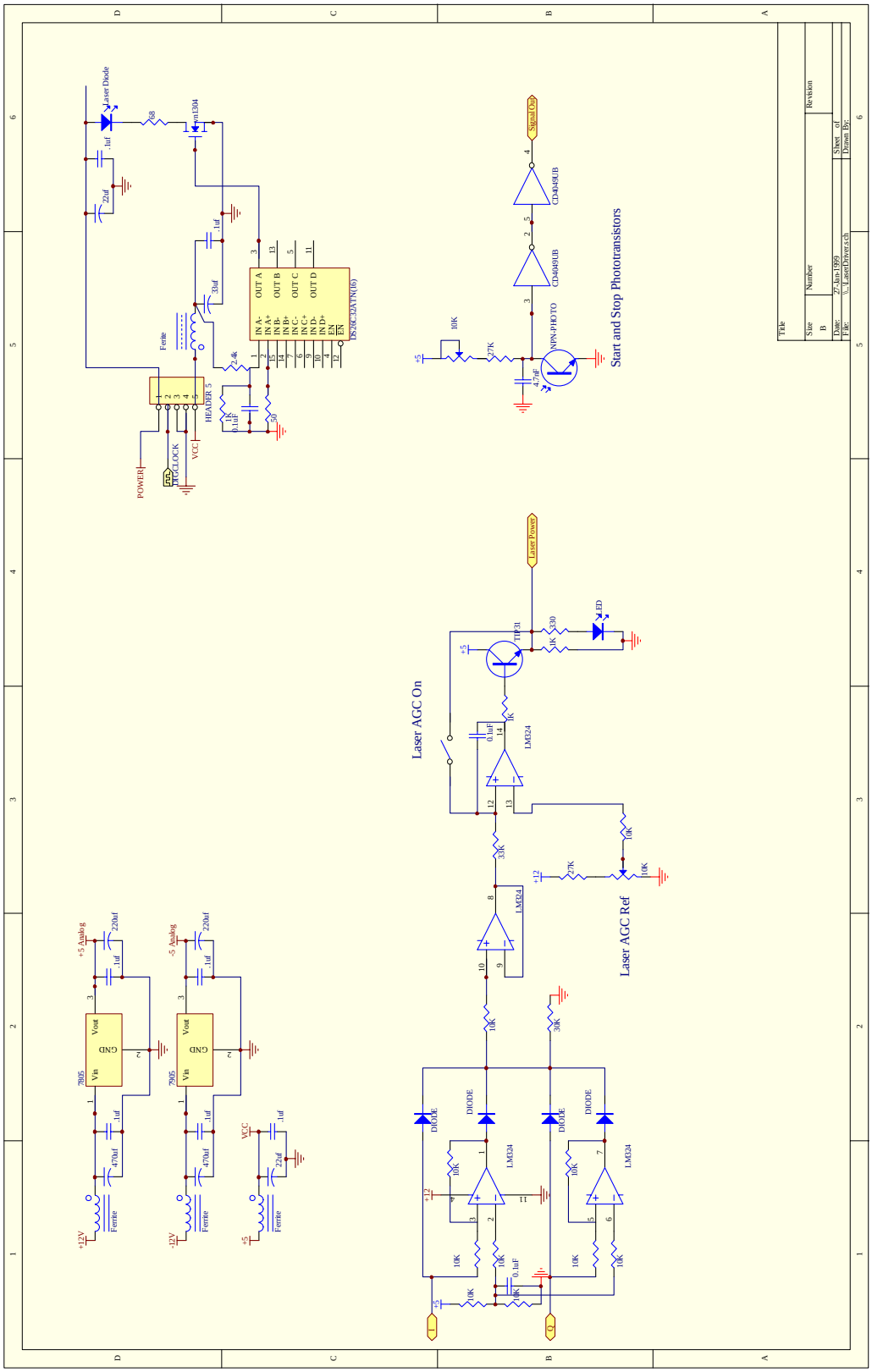

Figure A-3: Laser Driver,Voltage Regulation and Laser Power Control Schematics

Appendix B

Source Code

B.1 Embedded Code

B.1.1 PIC

Lasrun2.c

```
#include <16c73a.h>
#include <stdlib.h>
#include "math.c"

#fuses HS,NOWDT,NOPROTECT,PUT
#use Delay(Clock=16000000)

#define SAMPLE_HOLD PIN_A5
#define START_SCAN PIN_B0
#define END_SCAN PIN_B1
#define IN PIN_A0
#define QUAD PIN_A1

#use RS232(Baud=19200, Parity=N, Xmit=PIN_C6 , Rcv=PIN_C7 )

byte Num_Samples=0;
signed int Ins[6];
signed int Quads[6];
byte Ins_Sum_High[6];
byte Ins_Sum_Low[6];
byte Quads_Sum_High[6];
byte Quads_Sum_Low[6];
byte Time[6];
byte Time_Start[6];
byte Time_End[6];
byte zero=0;
byte sixfour=64;
short In_Peak=0;
int Num_Peaks=0;
signed int I_Temp;
signed int Q_Temp;
byte I_Temp2;
```

```
byte Q_Temp2;
byte Time_Temp;

byte I_Sum_High_Temp=0;
byte I_Sum_Low_Temp=0;
byte Q_Sum_High_Temp=0;
byte Q_Sum_Low_Temp=0;

signed int I_Max;
signed int Q_Max;
byte Time_Max;

byte Time_Start_Temp;


long int Sum_Max;
long int Sum_Temp;
byte Threshold=0;
byte Wait_Time=0;

signed int I_Base_High;
signed int Q_Base_High;

signed int I_Base;
signed int Q_Base;


#int_ext
ext_isr() {
            // External interrupt on B0
}

#int_ad
ad_isr() {
            // A/D Conversion complete
}

#int_tbe
tbe_isr() {
            // Async Serial Data Transmitted
}

#int_rda
rda_isr() {
            // Async Serial Data In
}

#INT_RTCC
clock_isr(){
   ++Time_Temp;
   }

void sample(void){  //sample I and Q Capture Timer
   output_low(SAMPLE_HOLD);
```

```
   delay_us(5);
   set_adc_channel(IN);
   delay_us(20);
   I_Temp=Read_ADC();
   set_adc_channel(QUAD);
   delay_us(20);
   Q_Temp=Read_ADC();
   output_high(SAMPLE_HOLD);

}


void store_baseline(void){  //find Baseline for i and q
   int i;
   I_Base=0;
   Q_Base=0;
   I_Base_High=0;
   Q_Base_High=0;
   I_Temp=0;
   Q_Temp=0;
   while(input(Start_Scan)){}
   while(!input(Start_Scan)){}

    for (i=0; i<64; ++i){
       sample();
       add16( I_Base_High, I_Base, zero, I_Temp);
       add16( Q_Base_High, Q_Base, zero, Q_Temp);

     }

    div16by8(I_Base_High, I_Base, sixfour);

    div16by8(Q_Base_High, Q_Base, sixfour);


}

void set_threshold(void){
   Threshold=getchar();
   printf("\rThreshold set to %2X\n",Threshold);
   }

void set_wait(void){
   Wait_Time=getchar();
   printf("\rWait Time set to %2X\n", Wait_Time);
   }


void output_data(void){
   int i=0;
   putchar(Num_Peaks);
   //printf("Number of Peaks = %2X\r\n",Num_Peaks);
   //printf("Number of Samples = %2X\r\n",Num_Samples);
   for (i=0; i<Num_Peaks; i++){
```

```
      putchar(Ins[i]);
      //printf("I= %2X\r\n",Ins[i]);
      Ins[i]=0;
      putchar(Quads[i]);
      //printf("Q= %2X\r\n", Quads [i]);
      Quads[i]=0;
      putchar(Time[i]);
      //printf("Time= %2x\r\n",Time[i]);
      Time[i]=0;
      putchar(Time_Start[i]);
      //printf("Time_Start= %2X\r\n",Time_Start[i]);
      Time_Start[i]=0;
      putchar(Time_End[i]);
      //printf("Time_End= %2X\r\n", Time_End[i]);
      putchar(Ins_Sum_High[i]);
      //printf("I Sum=%2X%2X\r\n", Ins_Sum_High[i], Ins_Sum_Low[i]);
      putchar(Ins_Sum_Low[i]);
      putchar(Quads_Sum_High[i]);
      //printf("Q Sum=%2X%2X\r\n", Quads_Sum_High[i], Quads_Sum_Low[i]);
      putchar(Quads_Sum_Low[i]);
      }
   Num_Samples=0;
   Num_Peaks=0;
}

void read_data(void){
   while(input(Start_Scan)){}
   while(!input(Start_Scan)){}
   set_rtcc(0);
   Time_Temp=0;
   while(Time_Temp < Wait_Time){}
   while(input(End_Scan))
   {
   sample();
   ++Num_Samples;
   I_Temp2=I_Temp;
   Q_Temp2=Q_Temp;
   I_Temp=I_Temp-I_Base;
   Q_Temp=Q_Temp-Q_Base;
   Sum_Temp=ABS(I_Temp)+ABS(Q_Temp);
   delay_us(5);
   if (Sum_Temp > Threshold)
      {

      Add16(I_Sum_High_Temp, I_Sum_Low_Temp, zero , I_Temp2);
      Sub16(I_Sum_High_Temp, I_Sum_Low_Temp, zero, I_Base);
      Add16(Q_Sum_High_Temp, Q_Sum_Low_Temp, zero, Q_Temp2);
      Sub16(Q_Sum_High_Temp, Q_Sum_Low_Temp, zero, Q_Base);

                              // Check if Above Threshold
        if(In_Peak)
       {
           // Check if Currently in a Peak
           if(Sum_Temp>Sum_Max)
```

```
            {         // Check to see if Value is Greater than Max
               Sum_Max=Sum_Temp;
               I_Max=I_Temp;                  // If it is store it
               Q_Max=Q_Temp;
               Time_Max=Time_Temp;

            }
          }
         else
         {In_Peak=1;                    // Now in peak set flag
          Time_Start_Temp=Time_Temp;
          Sum_Max=Sum_Temp;
          I_Max=I_Temp;
          Q_Max=Q_Temp;
          Time_Max=Time_Temp;
         }
      }
      else                           // sum is less than threshold
         { if(In_Peak)
            {         // out of peak for first time
              In_Peak=0;
              Sum_Max=0;
              if(Num_Peaks<6){
              Ins[Num_Peaks]=I_Max;
              Quads[Num_Peaks]=Q_Max;
              Time[Num_Peaks]=Time_Max;
              Time_Start[Num_Peaks]=Time_Start_Temp;
              Time_End[Num_Peaks]=Time_Temp;
              Ins_Sum_High[Num_Peaks]=I_Sum_High_Temp;
              Ins_Sum_Low[Num_Peaks]=I_Sum_Low_Temp;
              Quads_Sum_High[Num_Peaks]=Q_Sum_High_Temp;
              Quads_Sum_Low[Num_Peaks]=Q_Sum_Low_Temp;
              ++Num_Peaks;
              I_Sum_High_Temp=0;
              I_Sum_Low_Temp=0;
              Q_Sum_High_Temp=0;
              Q_Sum_Low_Temp=0;
              }
             }
          }
   }
         output_data();
         Sum_Max=0;
         I_Sum_High_Temp=0;
         I_Sum_Low_Temp=0;
         Q_Sum_High_Temp=0;
         Q_Sum_Low_Temp=0;
 }



void wait_char(void){
   byte command;
   command=getchar();
```

```
   if (command=='T'){
      set_threshold();}
   if (command=='W'){
      set_wait();}
   if (command=='R'){
      read_data();
      }
}


main() {

   set_tris_a(0x0f);
   set_tris_b(0xff);
   set_tris_c(0x80);
   set_rtcc(0);
setup_counters(RTCC_INTERNAL,RTCC_DIV_2);
setup_port_a(RA0_RA1_RA3_ANALOG);
setup_adc(ADC_CLOCK_DIV_32);
   // enable_interrupts(EXT_INT);
   // enable_interrupts(ADC_DONE);
   // enable_interrupts(INT_TBE);
   // enable_interrupts(INT_RDA);
   enable_interrupts(RTCC_ZERO);
   enable_interrupts(GLOBAL);
   output_high(SAMPLE_HOLD);
   printf("\rLaser Online\n");
   store_baseline();
   printf("I_Base= %2X  Q_Base= %2X\r\n",I_Base,Q_Base);
   while(1){
   wait_char();
   }

}
```

Math.c

```
#inline
void add16( byte & ah, byte & al, byte bh, byte bl) {
                       //// ahal = ahal + bhbl
   #asm
   movf  bl,w
   addwf al,f
   btfsc 3,0
   incf  ah,f
   movf   bh,w
   addwf ah,f
   #endasm
}


void sub16( byte & ah, byte & al, byte & bh, byte & bl) {
                     // ahal = ahal - bhbl
   #asm
   movf  bl,w
   subwf al,f
   btfss 3,0
   decf  ah,f
   movf   bh,w
   subwf ah,f
   #endasm
}


#inline
void div16by16( byte & bh, byte & bl, byte ah, byte al) {
                      //// bhbl = bhbl / ahal
   byte ch,cl,dh,dl,count;

   dl=bl;
   dh=bh;
   bh=0;
   bl=0;
   ch=0;
   cl=0;
   count=16;

   if((dh!=0)||(dl>al)) {
   #asm
loop:
   bcf   3,0
   rlf   dl,f
   rlf   dh,f
   rlf   cl,f
   rlf   ch,f
   movf  ah,w
   subwf ch,w
   btfss 3,2
   goto  nochk
```

```
   movf  al,w
   subwf cl,w
nochk:
   btfss 3,0
   goto  skip
   movf  al,w
   subwf cl,f
   btfss 3,0
   decf  ch,f
   movf  ah,w
   subwf ch,f
   bsf   3,0
skip:
   rlf   bl,f
   rlf   bh,f
   decfsz count,f
   goto  loop
   #endasm
   }
}


#inline
void rem16by8( byte & bh, byte & bl, byte al, byte & rl) {
                      //// bhbl = bhbl / al
                      //// rl = bhbl % al
   byte ch,cl,dh,dl,count,ah;

   dl=bl;
   dh=bh;
   bh=0;
   bl=0;
   ch=0;
   cl=0;
   ah=0;
   count=16;

   #asm
loop:
   bcf   3,0
   rlf   dl,f
   rlf   dh,f
   rlf   cl,f
   rlf   ch,f
   movf  ah,w
   subwf ch,w
   btfss 3,2
   goto  nochk
   movf  al,w
   subwf cl,w
nochk:
   btfss 3,0
   goto  skip
   movf  al,w
```

```
   subwf cl,f
   btfss 3,0
   decf  ch,f
   movf  ah,w
   subwf ch,f
   bsf   3,0
skip:
   rlf   bl,f
   rlf   bh,f
   decfsz count,f
   goto  loop
   #endasm
   rl=cl;
}




#inline
void mul16by8( byte & bh, byte & bl, byte a) {
                      //// bhbl = bhbl * a
   byte ch,cl,dh,dl,count;

   dl=bl;
   dh=bh;
   bh=0;
   bl=0;
   count=16;

   #asm
loop:
   rrf   dh,f
   rrf   dl,f
   btfss 3,0
   goto  skip
   movf  a,w
   addwf bl,f
   btfsc 3,0
   incf  bh,f
skip:
   rrf   bh,f
   rrf   bl,f
   rrf   ch,f
   rrf   cl,f
   decfsz count,f
   goto  loop
   #endasm

   bl=cl;
   bh=ch;
}
```

```
#inline
void div16by8( byte & bh, byte & bl, byte aL) {
                      //// bhbl = bhbl / a
   byte ch,cl,dh,dl,count,ah;

   dl=bl;
   dh=bh;
   bh=0;
   bl=0;
   ch=0;
   cl=0;
   ah=0;
   count=16;

   if((dh!=0)||(dl>al)) {
   #asm
loop:
   bcf   3,0
   rlf   dl,f
   rlf   dh,f
   rlf   cl,f
   rlf   ch,f
   movf  ah,w
   subwf ch,w
   btfss 3,2
   goto  nochk
   movf  al,w
   subwf cl,w
nochk:
   btfss 3,0
   goto  skip
   movf  al,w
   subwf cl,f
   btfss 3,0
   decf  ch,f
   movf  ah,w
   subwf ch,f
   bsf   3,0
skip:
   rlf   bl,f
   rlf   bh,f
   decfsz count,f
   goto  loop
   #endasm
   }
}
```

B.1.2 Hitachi SH-1

IOSH7030.h

```
/* IOSH7030.h   I/O addresses for SH1 series
 *
 * History:
 * 16-Oct-1995  Created by J.D.  (Not all checked)
 * 01-Aug-1996  Bugs in ITU defs fixed by Andrew Huang
 * 19-Apr-1998  rehmi@media.mit.edu: Added SCI and PBCRMD bits
 * 13-May-1998  strickon@media.mit.edu added AD bits
 * 27-May-1998  strickon@media.mit.edu added more itu bytes
 */

/*
 * standard type definitions
 */

#define FALSE   (0==1)
#define TRUE    (!FALSE)
#define NULL    ((void *)0L)

typedef unsigned char   BOOL;
typedef   signed char   BYTE;
typedef   signed short  WORD;
typedef   signed long   LONG;
typedef unsigned char  UBYTE;
typedef unsigned short UWORD;
typedef unsigned long  ULONG;

/*ITU SHARED*/
#define ITU_TSTR  (* (volatile UBYTE  *)(0x05ffff00))
#define ITU_TSNC  (* (volatile UBYTE  *)(0x05ffff01))
#define ITU_TMDR  (* (volatile UBYTE  *)(0x05ffff02))
#define ITU_TFCR  (* (volatile UBYTE  *)(0x05ffff03))
#define ITU_TOCR  (* (volatile UWORD  *)(0x05ffff31))

/*ITU CHANNEL 0*/
#define ITU_TCR0  (* (volatile UBYTE  *)(0x05ffff04))
#define ITU_TIOR0 (* (volatile UBYTE  *)(0x05ffff05))
#define ITU_TIER0 (* (volatile UBYTE  *)(0x05ffff06))
#define ITU_TSR0  (* (volatile UBYTE  *)(0x05ffff07))
#define ITU_TCNT0 (* (volatile UWORD  *)(0x05ffff08))
#define ITU_GRA0  (* (volatile UWORD  *)(0x05ffff0A))
#define ITU_GRB0  (* (volatile UWORD  *)(0x05ffff0C))


/*ITU CHANNEL 4*/
#define ITU_TCR4  (* (volatile UBYTE  *)(0x05ffff32))
#define ITU_TIOR4 (* (volatile UBYTE  *)(0x05ffff33))
#define ITU_TIER4 (* (volatile UBYTE  *)(0x05ffff34))
#define ITU_TSR4  (* (volatile UBYTE  *)(0x05ffff35))
#define ITU_TCNT4 (* (volatile UWORD  *)(0x05ffff36))
#define ITU_GRA4  (* (volatile UWORD  *)(0x05ffff38))
```

```
#define ITU_GRB4  (* (volatile UWORD  *)(0x05ffff3A))
#define ITU_BRA4  (* (volatile UWORD  *)(0x05ffff3C))
#define ITU_BRB4  (* (volatile UWORD  *)(0x05ffff3E))

/*DMAC CHANNELS 0-3 SHARED*/
#define DMAOR     (* (volatile UWORD *)(0x05ffff48))

#define DMAOR_PR1           0x0200
#define DMAOR_PR0           0x0100
#define DMAOR_AE            0x0004
#define DMAOR_NMIF          0x0002
#define DMAOR_DME           0x0001

/*DMAC CHANNEL 0*/
#define DMA_SAR0  (* (volatile ULONG  *)(0x05ffff40))
#define DMA_DAR0  (* (volatile ULONG  *)(0x05ffff44))
#define DMA_TCR0  (* (volatile UWORD *)(0x05ffff4a))
#define DMA_CHCR0 (* (volatile UWORD *)(0x05ffff4e))

#define DMA_CHCR_DM1        0x8000
#define DMA_CHCR_DM0        0x4000
#define DMA_CHCR_SM1        0x2000
#define DMA_CHCR_SM0        0x1000
#define DMA_CHCR_RS3        0x0800
#define DMA_CHCR_RS2        0x0400
#define DMA_CHCR_RS1        0x0200
#define DMA_CHCR_RS0        0x0100
#define DMA_CHCR_AM         0x0080
#define DMA_CHCR_AL         0x0040
#define DMA_CHCR_DS         0x0020
#define DMA_CHCR_TM         0x0010
#define DMA_CHCR_TS         0x0008
#define DMA_CHCR_IE         0x0004
#define DMA_CHCR_TE         0x0002
#define DMA_CHCR_DE         0x0001

/*DMAC CHANNEL 1*/
#define DMA_SAR1  (* (volatile ULONG  *)(0x05ffff50))
#define DMA_DAR1  (* (volatile ULONG  *)(0x05ffff54))
#define DMA_TCR1  (* (volatile UWORD *)(0x05fffF5a))
#define DMA_CHCR1 (* (volatile UWORD *)(0x05ffff5e))

/*DMAC CHANNEL 3*/
#define DMA_SAR3  (* (volatile ULONG  *)(0x05ffff60))
#define DMA_DAR3  (* (volatile ULONG  *)(0x05ffff64))
#define DMA_TCR3  (* (volatile UWORD *)(0x05fffF6a))
#define DMA_CHCR3 (* (volatile UWORD *)(0x05ffff6e))

/*DMAC CHANNEL 4*/
#define DMA_SAR4  (* (volatile ULONG  *)(0x05ffff70))
#define DMA_DAR4  (* (volatile ULONG  *)(0x05ffff74))
#define DMA_TCR4  (* (volatile UWORD *)(0x05fffF7a))
#define DMA_CHCR4 (* (volatile UWORD *)(0x05ffff7e))
```

```
/*INTC*/
#define INTC_IPRA (* (volatile UWORD *)(0x05ffff84))
#define INTC_IPRB (* (volatile UWORD *)(0x05ffff86))
#define INTC_IPRC (* (volatile UWORD *)(0x05ffff88))
#define INTC_IPRD (* (volatile UWORD *)(0x05ffff8A))
#define INTC_IPRE (* (volatile UWORD *)(0x05ffff8C))
#define INTC_ICR  (* (volatile UWORD *)(0x05ffff8E))

/*UBC*/
#define UBC_BARH  (* (volatile UWORD *)(0x05ffff90))
#define UBC_BARL  (* (volatile UWORD *)(0x05ffff92))
#define UBC_BAMRH (* (volatile UWORD *)(0x05ffff94))
#define UBC_BAMRL (* (volatile UWORD *)(0x05ffff96))
#define UBC_BBR   (* (volatile UWORD *)(0x05ffff98))

/*BSC*/
#define BSC_BCR   (* (volatile UWORD *)(0x05ffffA0))
#define BSC_WCR1  (* (volatile UWORD *)(0x05ffffA2))
#define BSC_WCR2  (* (volatile UWORD *)(0x05ffffA4))
#define BSC_WCR3  (* (volatile UWORD *)(0x05ffffA6))
#define BSC_DCR   (* (volatile UWORD *)(0x05ffffA8))
#define BSC_PCR   (* (volatile UWORD *)(0x05ffffAA))
#define BSC_RCR   (* (volatile UWORD *)(0x05ffffAC))
#define BSC_RTCSR (* (volatile UWORD *)(0x05ffffAE))
#define BSC_RTCNT (* (volatile UWORD *)(0x05ffffB0))
#define BSC_RTCOR (* (volatile UWORD *)(0x05ffffB2))

/*WDT*/
#define WDT_TCSR  (* (volatile UBYTE  *)(0x05ffffB8))
#define WDT_TCNT  (* (volatile UBYTE  *)(0x05ffffB9))
#define WDT_RSTCSR (* (volatile UBYTE *)(0x05ffffBB))

/*POWER DOWN STATE*/
#define PDT_SBYCR (* (volatile UBYTE  *)(0x05ffffBC))
#define SBYCR_HIZ       0x40
#define SBYCR_SBY       0x80

/*PORT A*/
#define PADR      (* (volatile UWORD *)(0x05ffffC0))

/*PORT B*/
#define PBDR      (* (volatile UWORD *)(0x05ffffC2))

/*PORT C*/
#define PCDR      (* (volatile UWORD *)(0x05ffffD0))

/*PFC*/
#define PFC_PAIOR (* (volatile UWORD *)(0x05ffffC4))
#define PFC_PBIOR (* (volatile UWORD *)(0x05ffffC6))
#define PFC_PACR1 (* (volatile UWORD *)(0x05ffffC8))
#define PFC_PACR2 (* (volatile UWORD *)(0x05ffffCA))
#define PFC_PBCR1 (* (volatile UWORD *)(0x05ffffCC))
#define PFC_PBCR2 (* (volatile UWORD *)(0x05ffffCE))
#define PFC_CASCR (* (volatile UWORD *)(0x05ffffEE))
```

```
/*TPC*/
#define TPC_TPMR  (* (volatile UWORD *)(0x05ffffF0))
#define TPC_TPCR  (* (volatile UWORD *)(0x05ffffF1))
#define TPC_NDERH (* (volatile UWORD *)(0x05ffffF2))
#define TPC_NDERL (* (volatile UWORD *)(0x05ffffF3))
#define TPC_NDRB  (* (volatile UBYTE  *)(0x05ffffF4))
#define TPC_NDRA  (* (volatile UBYTE  *)(0x05ffffF5))
#define TPC_NDRB1 (* (volatile UBYTE  *)(0x05ffffF6))
#define TPC_NDRA1 (* (volatile UBYTE  *)(0x05ffffF7))


#define SCI0_BASE       0x05FFFEC0  /* sci channel 0 base address */
#define SCI1_BASE       0x05FFFEC8  /* sci channel 1 base address */
#define SCI_SMR         0       /* serial mode register offset */
#define SCI_BRR         1       /* bit rate register offset */
#define SCI_SCR         2       /* serial control register offset */
#define SCI_TDR         3       /* transmit data register offset */
#define SCI_SSR         4       /* serial status register offset */
#define SCI_RDR         5       /* receive data register offset */

#define SCI0_SMR_REG        (*(volatile UBYTE *)(SCI0_BASE+SCI_SMR))
#define SCI0_BRR_REG        (*(volatile UBYTE *)(SCI0_BASE+SCI_BRR))
#define SCI0_SCR_REG        (*(volatile UBYTE *)(SCI0_BASE+SCI_SCR))
#define SCI0_TDR_REG        (*(volatile UBYTE *)(SCI0_BASE+SCI_TDR))
#define SCI0_SSR_REG        (*(volatile UBYTE *)(SCI0_BASE+SCI_SSR))
#define SCI0_RDR_REG        (*(volatile UBYTE *)(SCI0_BASE+SCI_RDR))

#define SCI1_SMR_REG        (*(volatile UBYTE *)(SCI1_BASE+SCI_SMR))
#define SCI1_BRR_REG        (*(volatile UBYTE *)(SCI1_BASE+SCI_BRR))
#define SCI1_SCR_REG        (*(volatile UBYTE *)(SCI1_BASE+SCI_SCR))
#define SCI1_TDR_REG        (*(volatile UBYTE *)(SCI1_BASE+SCI_TDR))
#define SCI1_SSR_REG        (*(volatile UBYTE *)(SCI1_BASE+SCI_SSR))
#define SCI1_RDR_REG        (*(volatile UBYTE *)(SCI1_BASE+SCI_RDR))

/*
 * Serial mode register bits
 */

#define SCI_SYNC_MODE       0x80
#define SCI_SEVEN_BIT_DATA  0x40
#define SCI_PARITY_ON           0x20
#define SCI_ODD_PARITY          0x10
#define SCI_STOP_BITS_2         0x08
#define SCI_ENABLE_MULTIP       0x04
#define SCI_CKS1        0x02
#define SCI_CKS0        0x01

/*
 * Serial control register bits
 */
#define SCI_TIE             0x80    /* Transmit interrupt enable */
#define SCI_RIE             0x40    /* Receive interrupt enable */
#define SCI_TE              0x20    /* Transmit enable */
```

```
#define SCI_RE              0x10    /* Receive enable */
#define SCI_MPIE            0x08    /* Multiprocessor interrupt enable */
#define SCI_TEIE            0x04    /* Transmit end interrupt enable */
#define SCI_CKE1            0x02    /* Clock enable 1 */
#define SCI_CKE0            0x01    /* Clock enable 0 */

/*
 * Serial status register bits
 */
#define SCI_TDRE            0x80    /* Transmit data register empty */
#define SCI_RDRF            0x40    /* Receive data register full */
#define SCI_ORER            0x20    /* Overrun error */
#define SCI_FER             0x10    /* Framing error */
#define SCI_PER             0x08    /* Parity error */
#define SCI_TEND            0x04    /* Transmit end */
#define SCI_MPB             0x02    /* Multiprocessor bit */
#define SCI_MPBT            0x01    /* Multiprocessor bit transfer */

/*
 * BRR value calculation
 */
#define SCI_BRR_VALUE(cks,rate)
 ((CPU_CLOCK_SPEED / (32 * (1 << (cks*2)) * rate)) - 1)

/*------------------------------------------------------
**   Pin Function Controller register definitions
**------------------------------------------------------*/
#define PBIOR           (*(volatile UWORD *)(0x5ffffc6))
#define PBCR1           (*(volatile UWORD *)(0x5ffffcc))
#define PBCR2           (*(volatile UWORD *)(0x5ffffce))
//#define PBDR          (*(volatile UWORD *)(0x5ffffc2))

/*
 * Port B control register 1 bits
 */
#define PB15MD1             0x8000      /* bit 15*/
#define PB15MD0             0x4000      /* bit 14*/
#define PB14MD1             0x2000      /* bit 13*/
#define PB14MD0             0x1000      /* bit 12*/
#define PB13MD1             0x0800      /* bit 11*/
#define PB13MD0             0x0400      /* bit 10*/
#define PB12MD1             0x0200      /* bit 9 */
#define PB12MD0             0x0100      /* bit 8 */
#define PB11MD1             0x0080      /* bit 7 */
#define PB11MD0             0x0040      /* bit 6 */
#define PB10MD1             0x0020      /* bit 5 */
#define PB10MD0             0x0010      /* bit 4 */
#define PB9MD1              0x0008      /* bit 3 */
#define PB9MD0              0x0004      /* bit 2 */
#define PB8MD1              0x0002      /* bit 1 */
#define PB8MD0              0x0001      /* bit 0 */


#define PAIOR               (*(volatile UWORD *)(0x5ffffc4)
```

```
#define PACR1               (*(volatile UWORD *)(0x5ffffc8))
#define PACR2               (*(volatile UWORD *)(0x5ffffca))
//#define PADR              (*(volatile UWORD *)(0x5ffffc0))

/*
 * Port A control register 1 bits
 */
#define PA15MD1             0x8000      /* bit 15*/
#define PA15MD0             0x4000      /* bit 14*/
#define PA14MD1             0x2000      /* bit 13*/
#define PA14MD0             0x1000      /* bit 12*/
#define PA13MD1             0x0800      /* bit 11*/
#define PA13MD0             0x0400      /* bit 10*/
#define PA12MD1             0x0200      /* bit 9 */
#define PA12MD0             0x0100      /* bit 8 */
#define PA11MD1             0x0080      /* bit 7 */
#define PA11MD0             0x0040      /* bit 6 */
#define PA10MD1             0x0020      /* bit 5 */
#define PA10MD0             0x0010      /* bit 4 */
#define PA9MD1              0x0008      /* bit 3 */
#define PA9MD0              0x0004      /* bit 2 */
#define PA8MD1              0x0002      /* bit 1 */
#define PA8MD0              0x0001      /* bit 0 */

/*
 *Analog to digital converter stuff
 *
 */
#define ADC_ADDRA   (* (volatile UWORD  *)(0x05fffee0)) /* a/d data registers */
#define ADC_ADDRB   (* (volatile UWORD  *)(0x05fffee2))
#define ADC_ADDRC   (* (volatile UWORD  *)(0x05fffee4))
#define ADC_ADDRD   (* (volatile UWORD  *)(0x05fffee6))

/* a/d contro/status register */
#define ADC_ADCSR   (* (volatile UBYTE  *)(0x05fffee8))
/* a/d control register */
#define ADC_ADCR    (* (volatile UBYTE  *)(0x05fffee9))
```

laser.c

```
/*
 * main.c
 *
 */
#include "IOSH7030.h"
#include "laser.h"
#include <stdlib.h>

char tmp[200];
int  adc_chan1;
int adc_chan2;
int timer_0;
int debug=0;

void sample(void){  //sample I and Q and Timer 0
    read_adc_ch12(&I_Sample, &Q_Sample);
    get_timer_0(&Time_Temp);
}

int Start_Scan(){ //start photo transistor
    return(PBDR & 0x0001);
}

int Stop_Scan(){ //stop photo transistor
    return (PBDR & 0x0002);
}

void store_baseline(void){  //find Baseline for i and q
   int i=0;
   int I_Base_Temp=0;
   int Q_Base_Temp=0;
   I_Base=0;
   Q_Base=0;
   while(Start_Scan()){}
   while(!Start_Scan()){}
    while(Stop_Scan()){
       ++i;
       sample();
       I_Base_Temp=I_Base_Temp+I_Sample;
       Q_Base_Temp=Q_Base_Temp+Q_Sample;
     }
    I_Base=I_Base_Temp/i;
    Q_Base=Q_Base_Temp/i;
}


void set_threshold(void){
   int t;
   t=console_getc();
   t=t<<8;
   Threshold=t+console_getc();
   sprintf(tmp,"Threshold set to %4X\r\n",Threshold);
```

```
   console_puts(tmp);
   }


void set_wait(void){
   int t;
   t=console_getc();
   t=t<<8;
   Wait_Time=t+console_getc();
   sprintf(tmp,"Wait Time set to %4X\r\n", Wait_Time);
   console_puts(tmp);
   }

void set_debug(void){
    if(debug)
        {debug=0;
        sprintf(tmp,"Debug Mode Off\r\n");
        console_puts(tmp);
        }
        else{debug=1;
        sprintf(tmp,"Debug Mode On\r\n");
        console_puts(tmp);
        }
}

void output_data(void){
   int i=0;
   console_putc(11);
   console_putc(27);
   console_putc(48);
   console_putc(187);
   /*
     Num_Peaks=2;
   for(i=0; i <4 ; i++){
     Ins[i]=-238;
     Quads[i]=999;
     Time[i]=32324;
     Time_Start[i]=9800;
     Time_End[i]=56000;
     Ins_Sum[i]=-1000;
     Quads_Sum[i]=80077;
       }
   */
     console_putc(Num_Peaks);
      for (i=0; i<4; i++){
      console_putshort(Ins[i]);
      Ins[i]=0;
      console_putshort(Quads[i]);
      Quads[i]=0;
      console_putshort(Time[i]);
      Time[i]=0;
      console_putshort(Time_Start[i]);
      Time_Start[i]=0;
      console_putshort(Time_End[i]);
```

```
      Time_End[i]=0;
      console_putlong(Ins_Sum[i]);
      Ins_Sum[i]=0;
      console_putlong(Quads_Sum[i]);
      Quads_Sum[i]=0;

      }
   Num_Samples=0;
   Num_Peaks=0;
}

void output_data_debug(void){
   int i=0;
   sprintf(tmp,"Num_Peaks = %d\r\n",Num_Peaks);
   console_puts(tmp);
   sprintf(tmp,"Number of Samples = %d\r\n",Num_Samples);
   console_puts(tmp);
   for (i=0; i<4; i++){
      sprintf(tmp,"I=%d Q=%d\r\n",Ins[i],Quads[i]);
      console_puts(tmp);
      sprintf(tmp,"Time=%d. Time_Start=%d.\r\n",Time[i] ,Time_Start[i]);
      console_puts(tmp);
      sprintf(tmp, "Time_End=%d. Time_Sample=%d.\r\n",
      Time_End[i], Time_Sample[i]);
      console_puts(tmp);
      sprintf(tmp,"I_Sum=%d Q_Sum=%d\n\r",Ins_Sum[i],Quads_Sum[i]);
      console_puts(tmp);
      Ins[i]=0;
      Quads[i]=0;
      Time[i]=0;
      Time_Start[i]=0;
      Time_End[i]=0;
      Ins_Sum[i]=0;
      Quads_Sum[i]=0;
           }
   Num_Samples=0;
   Num_Peaks=0;
}

void read_data(void){
while(1){
   while(Start_Scan()){}
   while(!Start_Scan()){}
   clear_timer_0();
   Time_Temp=0;
   while(Time_Temp < Wait_Time){get_timer_0(&Time_Temp);}
   clear_timer_0();
   while(Stop_Scan())
   {
   sample();
   ++Num_Samples;
   I_Temp=I_Sample-I_Base;
   Q_Temp=Q_Sample-Q_Base;
   Sum_Temp=abs(I_Temp)+abs(Q_Temp);
```

```
//sprintf(tmp,"\rS=%8X T=%4X\n",Sum_Temp,Threshold);
//console_puts(tmp);

if (Sum_Temp > Threshold)
   {
   I_Sum_Temp=I_Sum_Temp+I_Temp;
   Q_Sum_Temp=Q_Sum_Temp+Q_Temp;

                                // Check if Above Threshold
      if(In_Peak)
     {
         // Check if Currently in a Peak
         if(Sum_Temp>Sum_Max)
         {         // Check to see if Value is Greater than Max
            Sum_Max=Sum_Temp;
            I_Max=I_Temp;                 // If it is store it
            Q_Max=Q_Temp;
            Time_Max=Time_Temp;

         }
       }
      else
      {In_Peak=1;                     // Now in peak set flag
       Time_Start_Temp=Time_Temp;
       Sum_Max=Sum_Temp;
       I_Max=I_Temp;
       Q_Max=Q_Temp;
       Time_Max=Time_Temp;
   tmp_time_sample = Num_Samples;
      }
   }
   else                            // sum is less than threshold
      { if(In_Peak)
         {         // out of peak for first time
           In_Peak=0;
           Sum_Max=0;
          // sprintf(tmp,"num=%d\n",Num_Peaks);
          // console_puts(tmp);
           if(Num_Peaks < 4)
           {
           Ins[Num_Peaks]=I_Max;
           Quads[Num_Peaks]=Q_Max;
           Time[Num_Peaks]=Time_Max;
           Time_Start[Num_Peaks]=Time_Start_Temp;
           Time_End[Num_Peaks]=Time_Temp;
           Ins_Sum[Num_Peaks]=I_Sum_Temp;
           Quads_Sum[Num_Peaks]=Q_Sum_Temp;
       Time_Sample[Num_Peaks] = tmp_time_sample;
           ++Num_Peaks;
           I_Sum_Temp=0;
           Q_Sum_Temp=0;
           }
         }
       }
```

```
      }

   In_Peak=0;
         if(debug){
         output_data_debug();
         }
         else {output_data();}
         Sum_Max=0;
         I_Sum_Temp=0;
         Q_Sum_Temp=0;
}
}

void wait_char(void){
   unsigned int command;
   command=console_getc();
   if (command=='T'){
      set_threshold();}
   if (command=='W'){
      set_wait();}
   if (command=='D'){
      set_debug();}
   if (command=='R'){
      read_data();
      }
}

int
enter()
{
    /* Turn on the LED */
    PBIOR |= 0x8000;
    PBCR1 &= 0x7fff;
    PBDR  |= 0x8000;
    PFC_PAIOR |= 0x1000;
    PFC_PAIOR &= ~0x2000;
    PFC_PACR1  = 0x0F00;
    /*
     * This is necessary to run on the SH1 LCEVB:
     * use /HBS,/LBS,/WR  instead of  /WRH,/WRL,A0
     */
    BSC_BCR |= 0x0800;

    SCI_init ();

    console_puts ("Initialized SCI!\r\n");

    copy_data ();

    console_puts ("Copied Data!\r\n");

    PBCR2=0x0000;
    sprintf(tmp,"Laser Online\r\n");
    console_puts(tmp);
```

```
    start_adc_ch12();
    setup_timer_0();
    store_baseline();
    start_timers();
    sprintf(tmp,"I_Base= %4X  Q_Base= %4X\r\n",I_Base,Q_Base);
    console_puts(tmp);
    while(1){
      wait_char();
    }

    return 0;
}
```

analog.c

```
/* functions for reading adc */
/* Josh Strickon 7/1/98 */

#include "IOSH7030.h"
void
setup_adc(){
}
void
read_adc_ch12(int *chan1, int *chan2){
*chan1=ADC_ADDRA >> 6;
*chan2=ADC_ADDRB >> 6;
}
void
start_adc_ch12(){
ADC_ADCSR=0x39;
}
void
stop_adc(){
    ADC_ADCSR=0x00;
}
```

timer.c

```
/* timer funcitons for sh */
/* Josh Strickon 7/1/98 */


#include "IOSH7030.h"

void
start_timers(void){
ITU_TSTR=0xff;
}

void
stop_timers(void){
ITU_TSTR=0x00;
}

void
clear_timer_0(void){
ITU_TCNT0=0x00;
}

void
setup_timer_0(void){
ITU_TCR0=0x03; /*count clock div 8*/
}

void
get_timer_0(int *timer){
*timer=ITU_TCNT0;
}
```

util.c

```
/* espcam.c Everest Summit Probe camera controller program
 *
 * -*- Mode:C -*-
 *
 * History:
 *
 * 09-Apr-1998 rehmi@media.mit.edu
 * Created this file to control a VLSI Vision
 * VV5300 CMOS digital camera on the VV300 sensor card.
 *
 * 19-Apr-1998 rehmi
 * Massive cleanup to make this code maintainable/hackable;
 * Minor changes to make the code use the actual ESPCAM
 * board and to make it compile standalone.
 *
 */

#define DEVELOPMENT 1

unsigned long MARKER = 0x12345678;

#include "IOSH7030.h"

#define CPU_CLOCK_SPEED 20000000


delay_ms (int n)
{
  int i;

  /*
   * This is calibrated empirically using do_timing_blips()
   */

  while (n > 0) {
    for (i = 0; i < 390; i++);
    n--;
  }
}

delay_100_us ()
{
  int i;

  /*
   * This is calibrated guesstimatically from delay_ms()
   */

  for (i = 0; i < 34; i++)
    i = i;
}
```

```
SCI0_38400 ()
{
SCI0_SMR_REG = (SCI0_SMR_REG & ~(SCI_CKS1|SCI_CKS0));
SCI0_BRR_REG = SCI_BRR_VALUE(0, 38400);
}

SCI_init ()
{
int i;

/*
 * Set up SCI 0 to run at async 8-N-1 1200 baud
 * (to talk to VV5300 camera)
 */

/* disable Tx and Rx */
SCI0_SCR_REG &= ~(SCI_TE | SCI_RE);

/* select communication format, async 8-N-1 */
SCI0_SMR_REG &= ~(SCI_SYNC_MODE   | SCI_SEVEN_BIT_DATA |
     SCI_PARITY_ON   | SCI_ODD_PARITY     |
     SCI_STOP_BITS_2 | SCI_ENABLE_MULTIP);

/* select baud rate */
SCI0_38400();

/* select internal clock, output on SCK */
SCI0_SCR_REG |=  SCI_CKE0;
SCI0_SCR_REG &= ~SCI_CKE1;

/* wait at least one bit time */
delay_ms (1);

/* enable Tx and Rx */
SCI0_SCR_REG |= SCI_TE | SCI_RE;

/*
 * Set up SCI 1 to run at async 8-N-1 38400 baud
 * (to talk to ESP main controller board)
 */
#if 0

/* disable Tx and Rx */
SCI1_SCR_REG &= ~(SCI_TE | SCI_RE);

/* select communication format, async 8-N-1 */
SCI1_SMR_REG &= ~(SCI_SYNC_MODE   | SCI_SEVEN_BIT_DATA |
     SCI_PARITY_ON   | SCI_ODD_PARITY     |
     SCI_STOP_BITS_2 | SCI_ENABLE_MULTIP);

/* select baud rate */
#if 1
SCI1_38400 ();
```

```
#else
SCI1_1200 ();
#endif

/* select internal clock, output on SCK */
SCI1_SCR_REG |=  SCI_CKE0;
SCI1_SCR_REG &= ~SCI_CKE1;

/* wait at least one bit time */
delay_ms(10);

/* enable Tx and Rx */
SCI1_SCR_REG |= SCI_TE | SCI_RE;
#endif

/* select TxD and RxD pin functions with the PFC
 * enable TxD0, TxD1, SCK1 as outputs
 * enable RxD0, RxD1, SCK0 as inputs
 */

PBIOR &= ~(0x5500); // input pins (RXD0, RXD1, SCK0, IRQ6)
PBIOR |=   0xAA00; // output pins (TXD0, TXD1, SCK1, IRQ7)
PBCR1 |=    PB8MD1 // RXD0 input
  | PB9MD1 // TXD0 output
     | PB10MD1 // RXD1 input
     | PB11MD1 // TXD1 output
     | PB13MD1; // SCK1 i/o
PBCR1 &= ~( PB8MD0  // RXD0 input
  | PB9MD0 // TXD0 output
  | PB10MD0 // RXD1 input
  | PB11MD0 // TXD1 output
  | PB12MD1 // PB12 normal i/o
  | PB12MD0 // instead of SCK0
  | PB13MD0 // SCK1 i/o
  | PB14MD0 // IRQ6 i/o
  | PB14MD1 // IRQ6 i/o
  | PB15MD0 // IRQ7 i/o
  | PB15MD1 // IRQ7 i/o
  );
}

SCI0_transmit (UBYTE c)
{
/* SCI status check and transmit data write:
  read the SSR, check that TDRE is 1, then write
  transmit data into TDR and clear TDRE to 0 */

while (!(SCI0_SSR_REG & SCI_TDRE)) {
/* spin */
}

SCI0_TDR_REG = c;
SCI0_SSR_REG &= ~SCI_TDRE;
}
```

```
SCI0_receive_ready ()
{
if (SCI0_SSR_REG & (SCI_ORER | SCI_PER | SCI_FER)) {
   SCI0_SSR_REG &= ~ (SCI_ORER | SCI_PER | SCI_FER);
SCI0_transmit ((UBYTE)7); // output BEL on error
}

return (SCI0_SSR_REG & SCI_RDRF) != 0;
}

UBYTE SCI0_receive ()
{
UBYTE r;

while (!SCI0_receive_ready()) {
/* spin */
}
r = SCI0_RDR_REG;
SCI0_SSR_REG &= ~SCI_RDRF;
return r;
}

LED_init ()
{
PFC_PBIOR |= 0x8000;
PFC_PBCR1 &= 0x3fff;
}

LED_toggle ()
{
PBDR  ^= 0x8000;
}

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


console_puts (char *s)
{
while (s && *s) {
SCI0_transmit ((UBYTE)*s++);
}
}

console_putc (int c)
{
   SCI0_transmit ((UBYTE)c);
}

console_putshort(int c)
{
  short tmp=c;

console_putc(tmp>>8);
```

```
console_putc(tmp& 0xff);
}



console_putlong(int c)
{
console_putc(c>>24);
console_putc((c>>16)&0xff);
console_putc((c>>8)&0xff);
console_putc(c&0xff);

}

console_peek ()
{
return SCI0_receive_ready();
}

console_getc ()
{
return SCI0_receive ();
}

console_puthexn (unsigned long n)
{
console_putc ("0123456789ABCDEF"[n&15]);
}

console_puthexb (unsigned long b)
{
console_puthexn (b >> 4);
console_puthexn (b);
}

console_puthexw (unsigned long w)
{
   console_puthexb (w >> 8);
   console_puthexb (w);
}

console_puthexl (unsigned long l)
{
   console_puthexw (l >> 16);
   console_puthexw (l);
}

console_putdec (int n)
{
  if (n < 0) {
    console_putc ('-');
    n = -n;
  }
```

```
  if (n > 9)
    console_putdec (n / 10);

  console_putc ("0123456789"[n%10]);
}

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

void
copy_data ()
{
ULONG *src;
ULONG *dst;
ULONG *end;
extern ULONG _data, _edata, _text, _etext, _bss, _ebss;

src = &_etext;
dst = &_data;
end = &_edata;
while (dst < &_edata) {
  *dst++ = *src++;
}
}

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

int
exit ()
{
again:
  asm ("sleep");
  goto again;
}



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

#if 0
uprintf (char *fmt, long a, long b, long c, long d, long e, long f)
{
char buf[4096];
sprintf (buf, fmt, a, b, c, d, e, f);
console_puts (buf);
}
#endif

unsigned long REKRAM = 0x9ABCDEF0;
```

vect.s

```
!---------------------------------------------------------------------------
!
! vect.s
!
! This file was automatically generated by the HiVIEW project generator.
!
!---------------------------------------------------------------------------

            .global     _vector_table
            .section    .vect

_vector_table:
            .long       start
            .long       __stack
            .long       start
            .long       __stack

! TODO: complete interrupt vector table
```

crt0.s

```
    .section .text
    .global start
start:
    mov.l   stack_k,r15

!!!
!!!  Don't zero out the BSS!  We depend upon static storage.
!!!
    ! zero out bss
    mov.l   edata_k,r0
    mov.l   end_k,r1
    mov #0,r2

start_l:
     mov.l  r2,@r0
    add #4,r0
    cmp/ge  r0,r1
    bt  start_l

    ! call the mainline
    mov.l   main_k,r0
    jsr @r0
    or  r0,r0

    ! call exit
    mov r0,r4
    mov.l   exit_k,r0
    jsr @r0
    or  r0,r0



    .global __text
    .global __etext
    .global __data
    .global __edata
    .global __bss
    .global __ebss


    .align 2
stack_k:
    .long   __stack
edata_k:
    .long   __edata
end_k:
    .long   _end
main_k:
    .long   _enter
exit_k:
    .long   _exit
```

```
    .section .stack
__stack:    .long   0xdeaddead
```

lcevb.ld

```
OUTPUT_FORMAT("symbolsrec")
OUTPUT_ARCH(sh)
INPUT(libm.a libc.a libgcc.a libc.a libgcc.a)

MEMORY
{
  rom   : ORIGIN = 0x00000000, LENGTH = 64k
  ram   : ORIGIN = 0x0A000000, LENGTH = 256k
}

SECTIONS
{
  .vect 0x00000000:
  {
    *(.vect);
  } > rom = 0xffff


  .text 0x00000400:
  {
    CREATE_OBJECT_SYMBOLS;
    *(.text);
    *(.strings);
  } > rom = 0xffff

  __text  = ADDR(.text);
  __etext = ((ADDR(.text) + SIZEOF(.text) + 1k - 1) & ~ (1k - 1));

  .data 0x0A000000: AT(__etext)
  {
    *(.data);
  } > ram

  __data = ADDR (.data);
  __edata = ADDR(.data) + SIZEOF(.data) ;


  .bss BLOCK(0x100):
  {
    *(.bss);
    *(COMMON);
    _end = . ;
  } > ram

  __bss  = ADDR(.bss);
  __ebss = ADDR(.bss) + SIZEOF(.bss);

  .stack 0x0A03FFFC:
  {
    *(.stack);
  } > ram
  __stack = ADDR(.stack);
```

```
  .tors :
  {
    ___ctors = . ;
    *(.ctors);
    ___ctors_end = . ;
    ___dtors = . ;
    *(.dtors);
    ___dtors_end = . ;
  } > ram

  .stab 0 (NOLOAD):
  {
    *(.stab);
  }

  .stabstr 0 (NOLOAD):
  {
    *(.stabstr);
  }
}
```

MAKEFILE

```
# MAKEFILE  Makefile for ESPCAM project
#
# History:
#
# 18-Apr-98 Created by Rehmi Post <rehmi@media.mit.edu>
#

CC=gcc
RUN=run
GDB=gdb
ASYNC=asynctsr
COM=1
BAUD=9600

CCFLAGS=-nostartfiles # -nostdlib -nodefaultlibs
ASFLAGS=
LDFLAGS=#-e _entry

###
### rules to make object files from C and assembly source
###

.SUFFIXES : .hex .x .o .c .s

%.o : %.c
    $(CC) $(CCFLAGS) -Wa,-ahls,-a=$*.lst -c -m1 -O4 -o $@ -g $*.c

%.o : %.s
    $(AS) $(ASFLAGS) -ahls -a=$*.lst -o $@ $^

###
###  end of preamble.
###

all:    laser.hex

laser.hex:  laser.o crt0.o analog.o timer.o util.o vect.o lcevb.ld
    $(LD) $(LDFLAGS) -g -o $@ $^ -Tlcevb.ld -Map $*.map

laser.o:        IOSH7030.h laser.h
analog.o:       IOSH7030.h
timer.o:        IOSH7030.h
util.o:         IOSH7030.h

#all:    espcam.hex

espcam.o: IOSH7030.h

espcam.hex: espcam.o crt0.o vect.o lcevb.ld
    $(LD) $(LDFLAGS) -g -o $@ $^ -Tlcevb.ld -Map $*.map

clean:
```

```
    -del *.o
    -del *.lst

cleanall: clean
    -del *.hex
    -del *.map

# demo of gdb running and talking to eval board
# on serial port

gdbdemo : flash.x
    - mode com$(COM):9600,n,8,1,p
    - $(ASYNC) $(COM)
    $(GDB) -command flash.gsc
```

B.2 Matlab Scripts

B.2.1 Lasercal.m

```
clear all;
load lasernew.dat
laser = lasernew;
% Note: This program assumes that you've already
% loaded all data into the array "laser"!!
%
%clear all;
%load laser2.dat;
%laser = laser2;
%
%   Coordinate Testing

%y = [480,480,480,240,240,240,0,0,0];
%x = [620,310,0,620,310,0,620,310,0];
%y = [0,240,480,0,240,480,0,240,480];
%x = [0,0,0,310,310,310,620,620,620];
%r = sqrt(x.^2 + y.^2);
%t = atan2(y,x);
%
%yr = r.*sin(t);
%xr = r.*cos(t);
%
%xo = .2;
%yo = .16;
%yp = y + yo;
%xp = x + xo;
%rp = sqrt(xp.^2 + yp.^2);
%tp = atan2(yp,xp);
%
%rp = .1*rp + 2.;
%tp = .3*tp + 0.5;
%
%  Invert the coordinates
%
x = laser(:,1)';
y = laser(:,2)';
x = 640 - x;
y = 480 - y;
x0 = x;
y0 = y;
%
%rp = laser(:,3);
%tp = laser(:,4);
%
rp = laser(:,3);
%tp = (laser(:,4)/2)*pi/180.;
tp = (laser(:,4)/457)*pi/180.;  %  Josh's SH Scanner
rp = rp';
tp = tp';
rp0 = rp;
```

```
tp0 = tp;
%
%  Cut out bad points in these arrays
%
sz = size(x);
nw = 0;
for np = 1:sz(2);
   %if (x(np) > 100)|(y(np) > 150)
      nw = nw + 1;
         xq(nw) = x(np);
         yq(nw) = y(np);
         rq(nw) = rp(np);
         tq(nw) = tp(np);
   %end;
   end;
   x = xq; y = yq; rp = rq; tp = tq;
%
r = sqrt(x.^2 + y.^2);
t = atan2(y,x);
%
% Linear cal into r and theta
%
mp(:,1) = rp';
mp(:,2) = tp';
mp(:,3) = 1;
sp= size(mp);
cv = eye(sp(1));
rc = lscov(mp,r',cv)
tc = lscov(mp,t',cv)
rcc = rc(1)*rp0 + rc(2)*tp0 + rc(3);
tcc = tc(1)*rp0 + tc(2)*tp0 + tc(3);
ycr = rcc.*sin(tcc);
xcr = rcc.*cos(tcc);
%
% Quadratic cal into r and theta, with quadratic r term
%
rqq = rp.^2;
rqq0 = rp0.^2;
mp(:,1) = rqq';
mp(:,2) = rp';
mp(:,3) = tp';
mp(:,4) = 1;
sp= size(mp);
cv = eye(sp(1));
rcq = lscov(mp,r',cv)
tcq = lscov(mp,t',cv)
rcc = rcq(1)*rqq0 + rcq(2)*rp0 + rcq(3)*tp0 + rcq(4);
tcc = tcq(1)*rqq0 + tcq(2)*rp0 + tcq(3)*tp0 + tcq(4);
ycrq = rcc.*sin(tcc);
xcrq = rcc.*cos(tcc);
%
% Fit into simply assumed x and y space
%
rccp = rc(1)*rp + rc(2)*tp + rc(3);
```

```
tccp = tc(1)*rp + tc(2)*tp + tc(3);
ycrp = rccp.*sin(tccp);
xcrp = rccp.*cos(tccp);
cp(:,1) = xcrp';
cp(:,2) = ycrp';
cp(:,3) = 1;
xcp = lscov(cp,x',cv)
ycp = lscov(cp,y',cv)
xcc = xcp(1)*xcr + xcp(2)*ycr + xcp(3);
ycc = ycp(1)*xcr + ycp(2)*ycr + ycp(3);
%
% Fit in r and theta space, not using crossterm
%
mr(:,1) = rp';
mt(:,1) = tp';
mr(:,2) = 1;
mt(:,2) = 1;
sp= size(mr);
cv = eye(sp(1));
rcg = lscov(mr,r',cv)
tcg = lscov(mt,t',cv)
rccg = rcg(1)*rp0 + rcg(2);
tccg = tcg(1)*tp0 + tcg(2);
ycrg = rccg.*sin(tccg);
xcrg = rccg.*cos(tccg);
%
% Nonlinear fit
%
%opt(18) = 100;
opt(14) = 70000;
%z0 = [80,-80,rcg(2),tcg(2),rcg(1),tcg(1),0];
%z0 = [rc(3),rc(3),rc(3),tc(3),rc(1),tc(2),0];
%z0 = [80,80,rcq(4),tcq(4),rcq(2),tcq(3),0];
z0 = [120,35,2.5,-10.,-80,1,0]
'crap'
%z0 = [80100,-500,0.5,6,50,.8,50];
load zstart.m
%z0 = zstart;
e0 = dltap(z0,x,y,rp,tp)
z = fmins('dltap',z0,opt,[],x,y,rp,tp);
zp = z';
save zfit zp -ascii
ef = dltap(z,x,y,rp,tp)
z0
z
rcr = (rp0 + z(3));
rcr2 = rcr*z(5) + z(7)*rcr.^2;
xnon = rcr2.*cos((tp0 + z(4))*z(6)) - z(1);
ynon = rcr2.*sin((tp0 + z(4))*z(6)) - z(2);
%


%
zv = [.2,.16,-2.,-0.5,10,3.3333]
```

```
%
plot(x0,y0,'o',xcr,ycr,'-',xcrq,ycrq,'-',
xcc,ycc,'--',xcrg,ycrg,':',xnon,ynon,'-.')
%
%fopen(5,'w')
%fprintf(5,'Linear fit (
rc
tc
rcq
tcq
```

B.2.2 Dltap.m

```
function e = dltap(z,x,y,rp,tp);
%
%z = [x0,y0,r0,t0,sr,st,srq]
%
rcr = (rp + z(3));
rcr2 = rcr*z(5) + z(7)*rcr.^2;
xx = rcr2.*cos((tp + z(4))*z(6)) - z(1);
yy = rcr2.*sin((tp + z(4))*z(6)) - z(2);
xd = xx - x;
yd = yy - y;
q = sum(xd.^2 + yd.^2);
e = q;
%e = sqrt(q);
```

B.2.3 Postplot.m

```
plot(x,y,'o')
hold on
d = 5;
k = 5;
for j = 1:k
   plot(xnon((j-1)*d+1:j*d),ynon((j-1)*d+1:j*d),'g')
   plot(xnon((j-1)*d+1:j*d),ynon((j-1)*d+1:j*d),'+g')
end;
for j = 1:k
   plot(xcrq((j-1)*d+1:j*d),ycrq((j-1)*d+1:j*d),'r')
   plot(xcrq((j-1)*d+1:j*d),ycrq((j-1)*d+1:j*d),'xr')
end;
hold off
```

B.2.4 Bernd.m

```
clear all;
load lasernew.dat
laser = lasernew;
% Note: This program assumes that you've already
% loaded all data into the array "laser"!!
%

x_raw = laser(:,1);
y_raw = laser(:,2);
r_raw = -laser(:,3);
theta_raw = laser(:,4);

%%%%%%%%%%%%%%%%%%%%%%%%
% inititalize
x_pixels = 640;
y_pixels = 480;

A = x_pixels;
B = y_pixels;

r0 = - min(r_raw);
r1 = sqrt(x_pixels^2+y_pixels^2) / (max(r_raw) - min(r_raw));
theta0 = - min(theta_raw);
theta1 = (pi/2) / (max(theta_raw) - min(theta_raw));

x_given = A-x_raw;
y_given = B-y_raw;

r = (r0 + r_raw) * r1;
theta = (theta0 +  theta_raw) * theta1;

x = r.* cos(theta);
y = r.* sin(theta);


%%%%%%%%%%%%%%%%%%%%%%%%%%%
% assign input/output vector
x_w_poly=[x'; y']';
x_poly=x_w_poly;
y_poly=[x_given'; y_given']';
x_d_poly=[];
y_d_poly=[];
dim_yd=0;
nclusters=1;
niterations=1;
% polyorder=5;
polyorder=input('Enter the polynomila order (>=1)!')
polyorder
crossvalidationfactor=0;
covarianceinput=0;
fixedclasseratio_x=0;
fixedclasseratio_y=0;
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% build a grid of points to check if prediciton is smooth
l=0;
step=40;
delta=30
for (k1=-delta:step:x_pixels+delta)
   for (k2=-delta:step:y_pixels+delta)
       l=l+1;
       x_w_mesh_poly(l,1)=A-k1;
       x_w_mesh_poly(l,2)=B-k2;
   end
end
L=l;
x_mesh_poly=x_w_mesh_poly;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% build polynomial model
[datastatistics, pcluster, mucluster, varxcluster, ...
          varycluster, betacluster, proby_dcluster] = ...
         cwm_all(x_w_poly, x_poly, x_d_poly, y_poly, y_d_poly, ...
         nclusters, niterations, polyorder, ...
         crossvalidationfactor, covarianceinput, ...
         fixedclasseratio_x, fixedclasseratio_y);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% predict data and grid
[ypred, y_dpred, y_dp] = predict_all(x_w_poly, x_poly, x_d_poly,..
         dim_yd, polyorder,...
         covarianceinput, datastatistics, ...
         pcluster, mucluster, varxcluster, varycluster,...
         betacluster,...
         proby_dcluster);

[ypred_mesh, y_dpred, y_dp] = predict_all(x_w_mesh_poly,...
          x_mesh_poly,x_d_poly, dim_yd, polyorder,...
         covarianceinput, datastatistics, ...
         pcluster, mucluster, varxcluster, varycluster,...
         betacluster, proby_dcluster);


figure(1)
   clf
   plot(-x_given, y_given,'bo')
   hold on
   plot(-x, y,'g+')
   plot(-ypred(:,1), ypred(:,2),'r+')
   title('b: pointing location, g:original laser data, ...
   r:corrected laser data')
   hold off

figure(2)
   clf
   for (l=1:L)
```

```
   plot([-x_mesh_poly(l,1) -ypred_mesh(l,1)],...
   [x_mesh_poly(l,2) ypred_mesh(l,2)],'r-')
   hold on
   plot(-ypred_mesh(l,1),ypred_mesh(l,2),'ro')
end
title('correction vector field, o: corrected value')
hold off
```

B.3 PC Software

B.3.1 Laser Interface

laser_interface.cpp

```
// laser_interface.cpp

// this file defines the methods an application will use to talk to the
// laser range finder hardware through the laser_interface object

// C++ methods


#include "laser_interface.h"
#include "serial2.h"
#include "test_main.h"


#define PI 3.14159265359

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

laser_interface *li;

laser_interface::laser_interface(double in_x_range, double in_y_range)
{
    polling = true;
    user_x_range = in_x_range;
    user_y_range = in_y_range;


    //com_port_num = 1;
    //baudrate = 19200;

    //InitializeSerial(com_port_num, baudrate);
    comSetup();

    //reset();

    for(int i=0; i<MAX_SIGNALS; i++) {
        signals[i].I = 0;
        signals[i].Q = 0;
        signals[i].timep = 0;
        signals[i].timei = 0;
        signals[i].timef = 0;
        signals[i].i_sum = 0;
        signals[i].q_sum = 0;
        signals[i].theta = 0;
        signals[i].r = 0;
        signals[i].x = 0;
```

```
        signals[i].y = 0;
        signals[i].x_old = 0;
        signals[i].y_old = 0;
    }

    for(i=0; i<NUMXYCOEFFS; i++) {
        r_coeff[i] = 0;
        theta_coeff[i] = 0;
    }

    for(i=0; i<4; i++) {
        q_r_coeff[i] = 0;
        q_theta_coeff[i] = 0;
    }

    //load config file here
    FILE *fp;
    threshold = 0;
    wait_time = 0;
    fp = fopen("laser.cfg", "r");
    fscanf(fp, "%d %d", &threshold, &wait_time);
    set_threshold(threshold);
    //set_wait(wait_time);
    fclose(fp);

    // setup averaging variables
    sig_set_index = 0;
    sig_set_num = 0;
    collecting_data = false;

//  load_calibration();
//  load_quad_coeffs();
    load_nl_coeffs();

}

laser_interface::~laser_interface()
{
}

void laser_interface::msg(char *s)
{
    // insert your relevant printing message here
    rogu->tw.pr("%s", s);
}

void laser_interface::error_msg(char *s)
{
    // insert your relevant printing function here
    rogu->tw.pr("ERROR: %s\n", s);
}

void laser_interface::reset()
{
```

```
    polling = false;
    //char out_str[5];
    char in_str[200];
    char output[200];
    int in_len = strlen("Initialized SCI!\r\nCopied
    Data!\r\nLaser Online\r\nI_Base= xxxx  Q_Base= xxxx\r\n");

    MessageBox(NULL, "Please reset the board now.", "LaserGL!", MB_ICONSTOP);

    read_bytes(in_len, (unsigned char *)in_str);
//  Communicate(NULL, 0, in_str, in_len, 0);

    // this scanf doesn't work right at all, but it doesn't matter
    // because we don't use these values anyway!
    int q, i;
    sscanf(in_str, "Laser Online\r\nI_Base= %4x Q_base= %4x\r\n", &i, &q);

    Ibase = i;
    Qbase = q;


    //sprintf(output, "RESET!   Qbase: %f Ibase: %f\n", Qbase, Ibase);
    sprintf(output, "%s\n\n", in_str);
    msg(output);

//  polling  = true;
}

void laser_interface::set_threshold(short int t)
{
    polling = false;
    char str[5];
    char reply[40];
    char s[50];

    int len = strlen("Threshold set to xxxx\r\n");

//  char tmp = t;

    sprintf(str, "T%c%c", t/256, t%256);
    rogu->tw.pr("Threshold command:%d %d\n", t/256, t%256);

    write_data((unsigned char *)str);

    read_bytes(len, (unsigned char *)reply);

    sprintf(s, "thresh reply: %s\n", reply);
    msg(s);

//  polling  = true;
}

void laser_interface::set_wait(short int w)
{
```

```
    polling = false;
    char str[5];
    char reply[40];
    char s[50];

    int len = strlen("Wait Time set to xxxx\r\n");

    sprintf(str, "W%c%c", w/256, w%256);
    rogu->tw.pr("wait command:%d %d\n", w/256, w%256);

    write_data((unsigned char *)str);

    read_bytes(len, (unsigned char *)reply);

    sprintf(s, "wait reply: %s\n", reply);
    msg(s);

//  polling  = true;
}


void laser_interface::readdata() {

    // first zero out the all of the signal structures
    for(int q=0; q<4; q++) {
        signals[q].I=0;
        signals[q].Q=0;
        signals[q].timep=0;
        signals[q].timei=0;
        signals[q].timef=0;
        signals[q].i_sum=0;
        signals[q].q_sum=0;
    }

    unsigned char str[100];

    read_data(str);

    num_signals = str[0];

    int tmp=0;
    int tmp1=0;
    int tmp2=0;

    for(int i=0; i<num_signals; i++) {

                // read in I as signed 16 bit and extend to 32
        tmp = str[i*BLOCK+1];
        if(tmp&0x80) {signals[i].I = 0xffff0000| tmp<<8 |str[i*BLOCK+1+1];}
        else{signals[i].I=tmp<<8 |str[i*BLOCK+1+1];}

                // read in Q as signed 16 bit and extend to 32
        tmp = str[i*BLOCK+2+1];
        if(tmp&0x80){signals[i].Q = 0xffff0000| tmp<<8 |str[i*BLOCK+3+1];}
```

```
        else{signals[i].Q=tmp<<8 |str[i*BLOCK+3+1];}



        tmp=(str[i*BLOCK+4+1] <<8 );
        signals[i].timep =  tmp | (str[i*BLOCK+5+1]& 0x000000ff);

        tmp=(str[i*BLOCK+6+1] <<8);
        signals[i].timei = tmp  | (str[i*BLOCK+7+1]&0x000000ff);

        tmp=(str[i*BLOCK+8+1] << 8);
        signals[i].timef =tmp | (str[i*BLOCK+9+1]&0x000000ff);


        tmp=(str[i*BLOCK+10+1] <<24);
        tmp1=(str[i*BLOCK+11+1] <<16);
        tmp2=(str[i*BLOCK+12+1] <<8);
        signals[i].i_sum = tmp | tmp1| tmp2  | (str[i*BLOCK+13+1]& 0x000000ff);


        tmp=(str[i*BLOCK+14+1] <<24);
        tmp1=(str[i*BLOCK+15+1] <<16);
        tmp2=(str[i*BLOCK+16+1] <<8);
        signals[i].q_sum = tmp | tmp1| tmp2  |(str[i*BLOCK+17+1]& 0x000000ff) ;

        signals[i].theta = .5*(float)(signals[i].timei + signals[i].timef);
        signals[i].r = atan2(signals[i].i_sum, signals[i].q_sum);

        if(signals[i].r>0){
            signals[i].r=signals[i].r-2*3.14159;
        }

/*      if(signals[i].i_sum > 0 && signals[i].q_sum > 0) {
            signals[i].good = false;
        }
        else
        {
            */
              signals[i].good = true;
        //}

    }
//  for(i=0;i<1;i++){
//  char s[200];
//  sprintf(s, "Peaks=%d Numsignal=%d I:%d Q:%d  timep:%d
// timei:%d timef:%d \ni_sum:%d q_sum:%d\n",
//          num_signals,i,signals[i].I, signals[i].Q,
//signals[i].timep, signals[i].timei,
//          signals[i].timef, signals[i].i_sum, signals[i].q_sum);
//  msg(s);
//  }

    // here we record the data for averaging and calibration
    if(collecting_data) {
```

```
    // first determine which is the biggest signal
        int current_biggest = 0;
        int biggest_index =0;

        //sprintf(s, "\nnum_signals: %d\n", num_signals);
        //msg(s);

        for(int i=0; i<num_signals; i++) {
            if(abs(signals[i].i_sum) + abs(signals[i].q_sum) > current_biggest) {
                current_biggest = abs(signals[i].i_sum) + abs(signals[i].q_sum);
                biggest_index = i;
            }
    //  sprintf(s, "I:%d Q:%d  timep:%d  timei:%d timef:%d \ni_sum:%d q_sum:%d\n",
    //      signals[i].I, signals[i].Q, signals[i].timep, signals[i].timei,
    //      signals[i].timef, signals[i].i_sum, signals[i].q_sum);

    //  msg(s);
        }

        if(num_signals>0 && sig_set_index < MAX_DATA_POINTS &&
            signals[biggest_index].good) {

            sig_set[sig_set_index].I = signals[biggest_index].I;
            sig_set[sig_set_index].Q = signals[biggest_index].Q;
            sig_set[sig_set_index].timep = signals[biggest_index].timep;
            sig_set[sig_set_index].timei = signals[biggest_index].timei;
            sig_set[sig_set_index].timef = signals[biggest_index].timef;
            sig_set[sig_set_index].i_sum = signals[biggest_index].i_sum;
            sig_set[sig_set_index].q_sum = signals[biggest_index].q_sum;
            sig_set_index++;


        }
    }



}


void laser_interface::first_poll(){
    char s = 'R';

    write_bytes(1, (unsigned char*)&s);

}

// least squares stuff

void laser_interface::record_data_point
(int n, float expected_r, float expected_t)
{
    char s[100];
```

```
    cal_data[n][0] = atan2(good_avg_i_sum, good_avg_q_sum);
    if(cal_data[n][0]>0){
                cal_data[n][0]=cal_data[n][0]-2*3.14159;
                }
    cal_data[n][1] = good_avg_t;

    sprintf(s, "avg_i_sum: %f avg_q_sum: %f avg_t: %f\n",
    good_avg_i_sum, good_avg_q_sum, good_avg_t);
    msg(s);
    sprintf(s, "translates to... r: %f theta: %f\n",
    cal_data[n][0], good_avg_t);
    msg(s);
    sprintf(s, "from %d samples\n", sig_set_index);
    msg(s);


    target_data[n][0] = expected_r;
    target_data[n][1] = expected_t;
}

void XYDivideMatrices(double A[NUMPOINTS][NUMSENSORS + 1],
                      double B[NUMPOINTS],
                      double X[NUMSENSORS + 1]);
void XYGaussJ(double a[NUMSENSORS +1][NUMSENSORS + 1]);

void laser_interface::calculate_xys()
{
    for(int i=0; i<num_signals; i++) {
        double adj_r = signals[i].r*r_coeff[0] +
        signals[i].theta*r_coeff[1] + r_coeff[2];
        double adj_theta = signals[i].r*theta_coeff[0] +
        signals[i].theta*theta_coeff[1] + theta_coeff[2];
        signals[i].x = adj_r*cos(adj_theta);
        signals[i].y = adj_r*sin(adj_theta);
    }
}

void laser_interface::quad_calculate_xys()
{
    for(int i=0; i<num_signals; i++) {
        double tr = signals[i].r*-1;
        double tt = signals[i].theta/2*(PI/180);
        double adj_r = q_r_coeff[0]*tr*tr + q_r_coeff[1]*tr + q_r_coeff[2]*tt +
            q_r_coeff[3];
        double adj_theta = q_theta_coeff[0]*tr*tr + q_theta_coeff[1]*tr +
            q_theta_coeff[2]*tt + q_theta_coeff[3];

        signals[i].x = adj_r*cos(adj_theta);
        signals[i].y = adj_r*sin(adj_theta);
    }
}

void laser_interface::nl_calculate_xys()
{
```

```
    for(int i=0; i<num_signals; i++) {
        double tt = signals[i].theta/457*(PI/180);
        double rcr = signals[i].r + nl_coeff[2];
        double rcr2 = rcr*nl_coeff[4] +
        nl_coeff[6]*rcr*rcr;
        signals[i].x = rcr2*cos((tt + nl_coeff[3])*
        nl_coeff[5]) - nl_coeff[0];
        signals[i].y = rcr2*sin((tt + nl_coeff[3])
        *nl_coeff[5]) - nl_coeff[1];
    }

    //rogu->tw.pr("r: %f theta: %f\n",
    //signals[0].r, signals[0].theta);
    //rogu->tw.pr("num_sigs: %d x: %f y: %f\n",
    //num_signals, signals[0].x, signals[0].y);
}

void laser_interface::calibrate_laser()
{
    double A[NUMPOINTS][NUMSENSORS + 1];
    int    i, j;
    double DesiredX[NUMPOINTS], DesiredY[NUMPOINTS];

    /* Set up the Matrix A: */
    for(i = 0; i < NUMPOINTS; i++) /* copy in the sensor data */
        for(j = 0; j < NUMSENSORS; j++)
            A[i][j] = cal_data[i][j];
    for(i = 0; i < NUMPOINTS; i++) /* offset values for each point */
        A[i][NUMSENSORS] = 1.0;

    /* Set the desired values of X and Y
       The cal_data rows must match these rows
       in order for the points to match */
    SetDesiredValues(DesiredX, DesiredY);

    /* Calculate the x coefficients */
    XYDivideMatrices(A, DesiredX, r_coeff);

    /* Calculate the y coefficients*/
    XYDivideMatrices(A, DesiredY, theta_coeff);
}

void laser_interface::save_calibration()
{
    int i;
    char filename[25];
    char tmp[40];
    FILE* myfile;

    sprintf(filename, "laser.cal");

    myfile = fopen(filename,"w");
    if (myfile == NULL)
```

```
    {

        sprintf(tmp, "Unable to open %s. Calibration not saved!\n",filename);
        error_msg(tmp);
        return;
    }

    for(i=0;i<NUMXYCOEFFS;i++) // initialize sensor values
        fprintf(myfile, "%f\n", r_coeff[i]);
    for(i=0;i<NUMXYCOEFFS;i++)
        fprintf(myfile, "%f\n", theta_coeff[i]);

    fclose(myfile);
    sprintf(tmp, "Calibration saved to file %s\n",filename);
    msg(tmp);
}

void laser_interface::load_calibration()
{
    int i;
    char filename[25];
    char tmp[40];
    FILE* myfile;
    double d;

    sprintf(filename, "laser.cal");

    myfile = fopen(filename,"r");
    if (myfile == NULL)
    {
        sprintf(tmp, "Unable to open %s. Calibration not read!\n",filename);
        error_msg(tmp);
        return;
    }

    for(i=0;i<NUMXYCOEFFS;i++) // initialize sensor values
    {
        fscanf(myfile, "%lf", &d);
        r_coeff[i] = d;
    //  tw.pr("%f\n", d);
    }
    for(i=0;i<NUMXYCOEFFS;i++)
    {
        fscanf(myfile, "%lf", &d);
        theta_coeff[i] = d;
    //  tw.pr("%f\n", d);
    }

    fclose(myfile);

    sprintf(tmp, "Calibration loaded from file %s\n",filename);
    msg(tmp);
}
```

```
void laser_interface::load_quad_coeffs()
{
    int i;
    char filename[25];
    char tmp[40];
    FILE* myfile;
    double d;

    sprintf(filename, "laserq.cal");

    myfile = fopen(filename,"r");
    if (myfile == NULL)
    {
        sprintf(tmp, "Unable to open %s. Calibration not read!\n",filename);
        error_msg(tmp);
        return;
    }

    for(i=0;i<4;i++) // initialize sensor values
    {
        fscanf(myfile, "%lf", &d);
        q_r_coeff[i] = d;
    //  tw.pr("%f\n", d);
    }
    for(i=0;i<4;i++)
    {
        fscanf(myfile, "%lf", &d);
        q_theta_coeff[i] = d;
    //  tw.pr("%f\n", d);
    }

    fclose(myfile);

    sprintf(tmp, "Calibration loaded from file %s\n",filename);
    msg(tmp);
}

void laser_interface::load_nl_coeffs()
{
    int i;
    char filename[25];
    char tmp[40];
    FILE* myfile;
    double d;

    sprintf(filename, "las_nl.cal");

    myfile = fopen(filename,"r");
    if (myfile == NULL)
    {
        sprintf(tmp, "Unable to open %s. Calibration not read!\n",filename);
        error_msg(tmp);
        return;
    }
```

```
    for(i=0;i<7;i++) // initialize sensor values
    {
        fscanf(myfile, "%lf", &d);
        nl_coeff[i] = d;
        rogu->tw.pr("%f\n", d);
    }

    fclose(myfile);

    sprintf(tmp, "Calibration loaded from file %s\n",filename);
    msg(tmp);
}

void DoError(char *Error)
{
    char s[40];
    sprintf(s, "%s\n", Error);
    li->error_msg(s);
}

/* This sets the desired values Matrices for CalibrateGestureWall */
void laser_interface::SetDesiredValues(double
DesiredR[NUMPOINTS], double DesiredT[NUMPOINTS])
{
    for(int i=0; i<NUMPOINTS; i++) {
        DesiredR[i] = target_data[i][0];
        DesiredT[i] = target_data[i][1];
    }


    //double tx, ty;

//  tx=0; ty = user_y_range;
//  DesiredR[0] = ty;  /* Upper left corner */
//  DesiredT[0] = tx;

//  tx=user_x_range/2; ty=user_y_range;
//  DesiredR[1] = sqrt(tx*tx+ty*ty); /* Upper middle */
//  DesiredT[1] = atan(tx/ty);

//  tx=user_x_range; ty=user_y_range;
//  DesiredR[2] = sqrt(tx*tx+ty*ty); /* Upper right corner */
//  DesiredT[2] = atan(tx/ty);
    /* ---------------------------------------- */
//  tx=0; ty=user_y_range/2;
//  DesiredR[3] = sqrt(tx*tx+ty*ty); /* Middle left side */
//  DesiredT[3] = atan(tx/ty);

//  tx=user_x_range/2; ty=user_y_range/2;
//  DesiredR[4] = sqrt(tx*tx+ty*ty); /* Middle */
//  DesiredT[4] = atan(tx/ty);

//  tx=user_x_range; ty=user_y_range/2;
```

```
//  DesiredR[5] = sqrt(tx*tx+ty*ty); /* Middle right side */
//  DesiredT[5] = atan(tx/ty);
    /* ---------------------------------------- */
//  tx=0; ty=0;
//  DesiredR[6] = sqrt(tx*tx+ty*ty); /* Lower left corner */
//  DesiredT[6] = PI/2;

//  tx=user_x_range/2; ty=0;
//  DesiredR[7] = sqrt(tx*tx+ty*ty); /* Lower middle */
//  DesiredT[7] = PI/2;

//  tx=user_x_range; ty=0;
//  DesiredR[8] = sqrt(tx*tx+ty*ty); /* Lower right */
//  DesiredT[8] = PI/2;
}

/* This divides matrices A and B and puts the result in X.
   This is used exclusively by CalibrateGestureWall
   Note that the Matrix dimensions are static. */
void XYDivideMatrices(double A[NUMPOINTS][NUMSENSORS + 1],
                      double B[NUMPOINTS],
                      double X[NUMSENSORS + 1])
{
    double ATranspose[NUMSENSORS + 1][NUMPOINTS];
    double ProductOfATransposeAndA[NUMSENSORS + 1][NUMSENSORS + 1];
    double ProductOfATransposeAndB[NUMSENSORS + 1];
    int    i, j, k;
    double Accumulator;

    /* Transpose the A matrix: */
    for(i = 0; i < NUMSENSORS + 1; i++)
        for(j = 0; j < NUMPOINTS; j++)
            ATranspose[i][j] = A[j][i];

    /* Multiply ATranspose and A so we have a square matrix we can invert: */
    for(i = 0; i < NUMSENSORS + 1; i++)
    {
        for(j = 0; j < NUMSENSORS + 1; j++)
        {
            Accumulator = 0; /* Reset the accumulator */
            for(k = 0; k < NUMPOINTS; k++)
             /* take the dot product of the row and column */
                Accumulator += (ATranspose[j][k] * A[k][i]);
            ProductOfATransposeAndA[j][i] = Accumulator;
        }
    }

    /* Invert the ProductOfATransposeAndA matrix: */
    XYGaussJ(ProductOfATransposeAndA);

    /* Multiply ATranspose with the B matrix of desired solutions */
    for(j = 0; j < NUMSENSORS + 1; j++)
    {
        Accumulator = 0; /* Reset the accumulator */
```

```
        for(k = 0; k < NUMPOINTS; k++)
         /* take the dot product of the row and column */
            Accumulator += (ATranspose[j][k] * B[k]);
        ProductOfATransposeAndB[j] = Accumulator;
    }

    /* Now we multiply ProductOfATransposeAndA with ProductOfATransposeAndB
       This is the final answer so we throw it in X */
    for(j = 0; j < NUMSENSORS + 1; j++)
    {
        Accumulator = 0; /* Reset the accumulator */
        for(k = 0; k < NUMSENSORS + 1; k++)
        /* take the dot product of the row and column */
            Accumulator += (ProductOfATransposeAndA[j][k] *
                            ProductOfATransposeAndB[k]);
        X[j] = Accumulator;
    }
}

#define SWAP(a,b) {temp=(a);(a)=(b);(b)=temp;} /* Used by GaussJ */
/* Gets the Inverse of the square matrix A with NUMSENSORS + 1 rows
   Taken mostly from Numerical Recipies in C */
void XYGaussJ(double a[NUMSENSORS +1][NUMSENSORS + 1])
{
    int    *indxc, *indxr, *ipiv;
    int    i, icol = 0, irow = 0, j, k, m, mm;
    int    n = NUMSENSORS + 1;
    double big = 0, dum = 0, pivinv, temp;

    /* Allocate space for these guys */
    indxr = (int *)malloc(sizeof(int)*n);
    indxc = (int *)malloc(sizeof(int)*n);
    ipiv = (int *)malloc(sizeof(int)*n);

    for(j = 0; j < n; j++)
        ipiv[j] = 0;

    for(i = 0; i < n; i++)
    {
        big = 0;
        for(j = 0; j < n; j++)
            if(ipiv[j] != 1)
                for(k = 0; k < n; k++)
                    if(ipiv[k] == 0)
                    {
                        if(fabs(a[j][k]) >= big)
                        {
                            big = fabs(a[j][k]);
                            irow = j;
                            icol = k;
                        }
                    }
                    else
                    {
```

```
                    if(ipiv[k] > 1)
                        DoError("Gaussj: Singular Matrix-1");
                }
        ++(ipiv[icol]);

        if(irow != icol)
        {
            for(m = 0; m < n; m++)
                SWAP(a[irow][m], a[icol][m]);
        }

        indxr[i] = irow;
        indxc[i] = icol;

        if(a[icol][icol] == 0)
            DoError("GaussJ: Singual Matrix-2");

        pivinv = 1 / a[icol][icol];

        a[icol][icol] = 1;

        for(m = 0; m < n; m++)
            a[icol][m] *= pivinv;

        for(mm = 0; mm < n; mm++)
            if(mm != icol)
            {
                dum = a[mm][icol];
                a[mm][icol] = 0;

                for(m = 0; m < n; m++)
                    a[mm][m] -= a[icol][m] * dum;
            }
    }

    for(m = n-1; m >= 0; m--)
    {
        if(indxr[m] != indxc[m])
            for(k = 0; k < n; k++)
                SWAP(a[k][indxr[m]], a[k][indxc[m]]);
    }

    /* remember to free them!*/
    free(ipiv);
    free(indxr);
    free(indxc);
}

// signal averaging stuff

void laser_interface::begin_collecting_data()
{
    sig_set_index = 0;
    sig_set_num = 0;
```

```
    collecting_data = true;
}

void laser_interface::stop_collecting_data()
{
    collecting_data = false;
    dump_data_to_file();
    find_data_avg();
}


void laser_interface::dump_data_to_file()
{
    FILE *fp;
    fp = fopen("laser.dat", "w");

    for(int i=0; i<sig_set_index; i++) {
        fprintf(fp, "%f %il %il\n", .5*(sig_set[i].timef+sig_set[i].timei),
            sig_set[i].i_sum, sig_set[i].q_sum);
    }

    fclose(fp);
}


void laser_interface::write_cal_data_to_file()
{
    FILE *fp;
    fp = fopen("lasercal.dat", "w");
    for(int i=0; i<25; i++){
        fprintf(fp, "%f %f %f %f\n", target_data[i][0],
        target_data[i][1], cal_data[i][0], cal_data[i][1]);
    }
    fclose(fp);

}

void laser_interface::find_data_avg()
{
    char s[40];
    sprintf(s, "Computing Avg...\n");
    msg(s);
    // calculate the first average
    float avg_q_sum = 0;
    float avg_i_sum = 0;
    float avg_time = 0;

    for(int i=0; i<sig_set_index; i++) {
        avg_q_sum += sig_set[i].q_sum;
        avg_i_sum += sig_set[i].i_sum;
        avg_time += .5*(sig_set[i].timef + sig_set[i].timei);
    }

    avg_q_sum = avg_q_sum/sig_set_index;
```

```
    avg_i_sum = avg_i_sum/sig_set_index;
    avg_time = avg_time/sig_set_index;

    // calcualte the variance
    float q_var = 0;
    float i_var = 0;
    float t_var = 0;

    for(i=0; i<sig_set_index; i++) {
        q_var += (sig_set[i].q_sum-avg_q_sum)*(sig_set[i].q_sum- avg_q_sum);
        i_var += (sig_set[i].i_sum-avg_i_sum)*(sig_set[i].i_sum-avg_i_sum);
        t_var += (.5*(sig_set[i].timef + sig_set[i].timei)-avg_time)*
            (.5*(sig_set[i].timef + sig_set[i].timei)-avg_time);
    }

    q_var = q_var/sig_set_index;
    i_var = i_var/sig_set_index;
    t_var = t_var/sig_set_index;

    // select good points and calculate a new average
    float new_q_avg=0;
    float new_i_avg=0;
    float new_t_avg=0;
    int num_new_data=0;
    for(i=0; i<sig_set_index; i++) {
        if((sig_set[i].q_sum-avg_q_sum)*(sig_set[i].q_sum- avg_q_sum) <= q_var &&
            (sig_set[i].i_sum-avg_i_sum)*(sig_set[i].i_sum-avg_i_sum) <= i_var &&
            (.5*(sig_set[i].timef + sig_set[i].timei)-avg_time)*
            (.5*(sig_set[i].timef + sig_set[i].timei)-avg_time) <= t_var) {

            new_q_avg += sig_set[i].q_sum;
            new_i_avg += sig_set[i].i_sum;
            new_t_avg += .5*(sig_set[i].timef + sig_set[i].timei);
            num_new_data++;
        }
    }

    new_q_avg = new_q_avg/num_new_data;
    new_i_avg = new_i_avg/num_new_data;
    new_t_avg = new_t_avg/num_new_data;

    sprintf(s, "num_new_data: %i\n", num_new_data);
    msg(s);

    good_avg_q_sum = new_q_avg;
    good_avg_i_sum = new_i_avg;
    good_avg_t = new_t_avg;

}

void laser_interface::process_data(void){

    int i;
```

```
    float alpha = ALPHA;

    //The filter operating system generates values of xPos and yPos based on
    //what it thinks it was linked to the previous time.



    //peaks are same, just average
    if(num_signals == num_signals_old) {
        for(i=0;i<num_signals;i++) {

            signals[i].x = alpha * signals[i].x_old + (1-alpha)*signals[i].x;

            signals[i].y = alpha * signals[i].y_old + (1-alpha)*signals[i].y;
        }
    }
     //copy data over
    num_signals_old=num_signals;
    for(i=0; i<num_signals;i++){
        signals[i].x_old=signals[i].x;
        signals[i].y_old=signals[i].y;
    }


}

// interface functions


void initialize_rangefinder(double in_x_range, double in_y_range)
{
    li = new laser_interface(in_x_range, in_y_range);
    //li->reset();
}

void rangefinder_first_poll()
{
    li->first_poll();
}

void get_rangefinder_points(double *x, double *y, int *num_points)
{
//  char s[100];
    //li->calculate_xys();
    //li->quad_calculate_xys();

    li->nl_calculate_xys();
    (*num_points) = li->num_signals;

    //li->process_data();
    for(int i=0; i<*num_points; i++) {
    //  sprintf(s,"r= %f theta=%f ", li->signals[i].r, li->signals[i].theta);
    //  li->msg(s);
```

```
        x[i] = 640 - li->signals[i].x;
        y[i] = 480 - li->signals[i].y;
        //x[i] = li->signals[i].x;
        //y[i] = li->signals[i].y;
    }
    //if((*num_points)<0) exit(0);
    /*
    char ww[30];
    sprintf(ww, "np: %d", *num_points);
    g_moe->debug_output(ww);
    */

//  rogu->tw.pr("num_peaks: %d x: %d y: %d\n", *num_points, x[0], y[0]);

}

void reset_rangefinder()
{
    li->reset();
}


void set_rangefinder_threshold(short int t)
{
    li->set_threshold(t);
}

void set_rangefinder_wait(short int w)
{
    li->set_wait(w);
}

void cleanup_rangefinder()
{
    delete li;
}

void write_data_file()
{
    li->write_cal_data_to_file();
}

void record_rangefinder_datapoint(int n, float x, float y)
{
//  li->record_data_point(n, sqrt(x*x+y*y), atan2(y, x));
    li->record_data_point(n, y, x);
}

void load_rangefinder_calibration()
{
    li->load_calibration();
}

void save_rangefinder_calibration()
```

```
{
    li->save_calibration();
}

void calibrate_rangefinder()
{
    li->calibrate_laser();
}

void begin_collecting_data()
{
    if(!(li->collecting_data)) {
    char s[40];
    sprintf(s, "beginning to collect data....\n");
    li->msg(s);
    li->begin_collecting_data();
//  li->polling=true;
    }
}

void stop_collecting_data()
{
    li->stop_collecting_data();
    char s[40];
    sprintf(s, "done collecting data!\n");
    li->msg(s);
//  li->polling=false;
}

void run()
{
    if(!(li->collecting_data)) {
    char s[40];
    sprintf(s, "running\n");
    li->msg(s);
    li->begin_collecting_data();
    li->polling=true;
    }
}

void rangefinder_readdata()
{
    li->readdata();
}
```

laser_interface.h

```
//d:\sdk\rogus\c\1.9a\include\rogus\1.9a
//d:\sdk\rogus\c\1.9a\lib\rogus\1.9a\Rog_Debug.lib
//c:\Rogus1.9\lib\Rog_Debug.lib
//c:\Rogus1.9\include\

// laser_interface.h

#define MAX_SIGNALS 30
#define BLOCK 18
#define MAX_BYTES 270
#define NUMPOINTS 25
#define NUMSENSORS 2
#define NUMXYCOEFFS NUMSENSORS + 1
#define MAX_DATA_POINTS 1000
#define ALPHA .7


class laser_interface
{
public:
laser_interface(double in_x_range, double in_y_range);
~laser_interface();

bool polling;
int com_port_num;
int baudrate;

int threshold;
int wait_time;
int Ibase;
int Qbase;

void msg(char *s);
void error_msg(char *s);
void reset();
void set_threshold(short int t);
void set_wait(short int w);
void poll();
void first_poll();
void process_data();

// least squares stuff
double user_x_range; // the x, y ranges of values that
double user_y_range; // the program expects to get back

double cal_data[NUMPOINTS][NUMSENSORS];
double r_coeff[3];
double theta_coeff[3];
double q_r_coeff[4];
double q_theta_coeff[4];
double nl_coeff[7];
```

```
void calculate_xys();
void quad_calculate_xys();
void nl_calculate_xys();
void calibrate_laser();
void save_calibration();
void load_calibration();
void load_quad_coeffs();
void load_nl_coeffs();
void SetDesiredValues(double DesiredR[NUMPOINTS], double DesiredT[NUMPOINTS]);
void record_data_point(int n, float expected_r, float expected_t);
float target_data[NUMPOINTS][NUMSENSORS];


typedef struct _signal { //a single signal readout unit
int I;
int Q;
int timep;
int timei;
int timef;
int i_sum;
int q_sum;
float r; //r=arctan(q/i)
float theta; //theta=width/r
float x;
float y;
float x_old;
float y_old;
bool good; //1 means we plot it
} Signal;

int num_signals;
int num_signals_old;
Signal signals[MAX_SIGNALS];

// signal averaging stuff
int sig_set_index;
int sig_set_num;
Signal sig_set[MAX_DATA_POINTS];
bool collecting_data;
void begin_collecting_data();
void stop_collecting_data();
void dump_data_to_file();
void write_cal_data_to_file();
void find_data_avg();
float good_avg_q_sum;
float good_avg_i_sum;
float good_avg_t;

void readdata();

};

extern laser_interface *li;
```

```
void initialize_rangefinder(double in_x_range, double in_y_range);
void get_rangefinder_points(double *x, double *y, int *num_points);
void reset_rangefinder();
void set_rangefinder_threshold(short int t);
void set_rangefinder_wait(short int w);
void cleanup_rangefinder();
void record_rangefinder_datapoint(int n, float x, float y);
void load_rangefinder_calibration();
void save_rangefinder_calibration();
void calibrate_rangefinder();
void begin_collecting_data();
void stop_collecting_data();
void rangefinder_first_poll();
void run();
void write_data_file();

void rangefinder_readdata();
```

B.3.2 Serial

serial2.cpp

```
#include "test_main.h"
#include "serial2.h"

static HANDLE hCom;


void
comSetup()
{
DCB dcb;
DWORD dwError;
BOOL fSuccess;
const char* serialPort = "COM1";

hCom = CreateFile(serialPort,
GENERIC_READ | GENERIC_WRITE,
0,    /* comm devices must be opened w/exclusive-access */
NULL, /* no security attrs */
OPEN_EXISTING, /* comm devices must use OPEN_EXISTING */
0,    /* not overlapped I/O */
NULL  /* hTemplate must be NULL for comm devices */
);

if (hCom == INVALID_HANDLE_VALUE) {
dwError = GetLastError();

/* handle error */
rogu->ew.w("Error opening %s",serialPort);
return;
}

/*
* Omit the call to SetupComm to use the default queue sizes.
* Get the current configuration.
*/

fSuccess = GetCommState(hCom, &dcb);

if (!fSuccess) {
/* Handle the error. */
rogu->ew.w("Error getting the comm state");
return;
}

/* Fill in the DCB: baud=19200, 8 data bits, no parity, 1 stop bit. */

dcb.BaudRate = 38400;
dcb.ByteSize = 8;
dcb.Parity = NOPARITY;
dcb.StopBits = ONESTOPBIT;
```

```
/*
dcb.fOutxCtsFlow=FALSE;
    dcb.fOutxDsrFlow=FALSE;
    dcb.fDtrControl=DTR_CONTROL_DISABLE;
    dcb.fDsrSensitivity=FALSE;
dcb.fNull=FALSE;
*/
fSuccess = SetCommState(hCom, &dcb);

if (!fSuccess) {
    /* Handle the error. */
rogu->ew.w("Error setting the comm state");
return;
}

SetupComm(hCom, 300, 300);

COMMTIMEOUTS cto;
cto.ReadIntervalTimeout = MAXWORD;
    cto.ReadTotalTimeoutMultiplier = 0;
    cto.ReadTotalTimeoutConstant = 0;
    cto.WriteTotalTimeoutMultiplier = 0;
    cto.WriteTotalTimeoutConstant = 100;
if (!SetCommTimeouts(hCom, &cto))
rogu->ew.w("Unable to set proper timeouts");
}


void read_data(unsigned char *data)
{

bool noheader=1;
unsigned char foo[100];
unsigned char temp_foo[100];
DWORD bytesread;

int i;
bool reading=1;
COMSTAT ComStat;
unsigned long dummy;


foo[0] = 7;

/*
// buffer empty
ReadFile(hCom, foo, 1, &bytesread, NULL);
while(bytesread>0) {
ReadFile(hCom, foo, 1, &bytesread, NULL);
//rogu->tw.pr("%i\n", foo[0]);
}
exit(0);
*/
```

```
        ClearCommError(hCom, &dummy, &ComStat ) ;



while(reading){
while(noheader){



ReadFile(hCom, foo, 1, &bytesread, NULL);
//rogu->tw.pr("1: %d bytes read %d\n", bytesread, foo[0]);
if (bytesread == 0)
{
exit(0);
continue;
}
if(bytesread==1 && foo[0]==11){
ReadFile(hCom, foo, 1, &bytesread, NULL);
//rogu->tw.pr("2: %d bytes read %d\n", bytesread, foo[0]);
if(bytesread==1 && foo[0]==27){
ReadFile(hCom, foo, 1, &bytesread, NULL);
//rogu->tw.pr("3: %d bytes read %d\n", bytesread, foo[0]);
if(bytesread==1 && foo[0]==48){
ReadFile(hCom, foo, 1, &bytesread, NULL);
//rogu->tw.pr("4: %d bytes read %d\n", bytesread, foo[0]);
if(bytesread==1 && foo[0]==187){
noheader=0;
} else continue;
} else continue;
} else continue;
} else continue;

//if(foo[0]==11&&foo[1]==283&&foo[2]==48&&foo[3]==187){
// noheader=0;
//}
//} else
//{
//}

}

ReadFile(hCom, foo,  73, &bytesread, NULL);


if (bytesread == 0)
continue;



if(bytesread==73) {

for (i = 0; i < 73; i++)
```

```
{
// rogu->tw.pr("%d ", (int)foo[i]);
if (foo[i] != temp_foo[i])
{
temp_foo[i] = foo[i];
}
}
ClearCommError(hCom, &dummy, &ComStat ) ;

if(ComStat.cbInQue < 77) reading = 0;
else {

ReadFile(hCom, foo, 77, &bytesread, NULL);
while(ComStat.cbInQue>=77) {
for (i = 0; i < 73; i++)
{
// rogu->tw.pr("%d ", (int)foo[i]);
if (foo[i+4] != temp_foo[i])
{
temp_foo[i] = foo[i+4];
}
}
ReadFile(hCom, foo, 77, &bytesread, NULL);
//rogu->tw.pr("bytes: %d\n", bytesread);
ClearCommError(hCom, &dummy, &ComStat ) ;
}
reading=0;
}

}
else {
rogu->ew.w("Expected 73 data bytes; read %lu", bytesread);
noheader=1;
continue;
// reading=0;
if(bytesread == 72) Sleep(3);
}


/*

if (bytesread != 73)
{
rogu->ew.w("Expected 73 data bytes; read %lu", bytesread);
noheader=1;
continue;
// reading=0;
if(bytesread == 72) Sleep(3);
}
else{reading=0;}

*/
}
```

```
// rogu->tw.pr("Num_Peaks=%d\n",(int)foo[0]);

// rogu->tw.pr("\n");

for(i=0; i<73; i++) {
if (foo[i] != data[i])
{
data[i] = foo[i];
}
}
}


void write_data(unsigned char *data) {

bool writing=true;
DWORD byteswritten=0;

while(writing){

WriteFile(hCom, data, 3, &byteswritten, NULL);
if (byteswritten == 0)
continue;
// always writes only 3 bytes at a time
if(byteswritten != 3) {
rogu->ew.w("Error! wrong number of data bytes written: %d\n", byteswritten);
} else
writing = false;
}


}


void write_bytes(unsigned int num_bytes, unsigned char *data){

bool writing=true;
DWORD byteswritten=0;

while(writing){

WriteFile(hCom, data, num_bytes, &byteswritten, NULL);
if (byteswritten == 0)
continue;

if(byteswritten != num_bytes) {
rogu->ew.w("Error! wrong number of data bytes written: %d\n", byteswritten);
} else
writing = false;
}


}
```

```
void read_bytes(unsigned int num_bytes, unsigned char *data){

bool reading=true;
DWORD bytesread=0;

while(reading){

ReadFile(hCom, data, num_bytes, &bytesread, NULL);
if (bytesread == 0)
continue;

if(bytesread != num_bytes) {
rogu->ew.w("Error! wrong number of data bytes read: %d\n", bytesread);
} else
reading = false;
}


}
```

serial2.h

```
void comSetup();
void read_data(unsigned char *data);
void write_data(unsigned char *data);
void read_bytes(unsigned int num_bytes, unsigned char *data);
void write_bytes(unsigned int num_bytes, unsigned char *data);
```

Bibliography

[AD695] Ad603 low noise, 90mhz variable-gain amplifier. Analog Devices Data Sheet, 1995.

[AP96] Ali Azarbayejani and Alex Pentland. Real-time self-calibrating stereo person tracking using 3-d shape estimation from blob features. In *Proceedings of ICPR '96*, Vienna, Austria, August 1996. IEEE.

[Aya91] Nicholas Ayache. *Artificial Vision for Mobile Robots Stereo Vision and Multisensory Perception*. The MIT Press, Cambridge, 1991.

[BFB+98] Nunzio Alberto Borghese, Giancarlo Ferrigno, Guido Baroni, Antonio Pedotti, Stefano Ferrari, and Riccardo Savar. Autoscan: A flexible and portable 3d scanner. *IEEE Computer Graphics and Applications*, 18(3):38–41, May/June 1998.

[Bov88] V. M. Bove, Jr. Pictorial applications for range sensing cameras. In *Image Processing, Analysis, Measurement, and Quality*, volume 901, pages 10–17, Bellingham, 1988. Society of Photo-Optical Instrumentation Engineers.

[BS91] George J. Blanar and Richard Sumner. New time digitizer applications in particle physics experiments. In *Proceedings on the First Annual Conference on Electronics for Future Colliders*, pages 87–97. Research Systems Division, LeCroy Corporation, 1991.

[CG95] Walt Chapelle and Ed Gillis. Sensing automobile occupant position with optical triangulation. *Sensors*, pages 18–21, December 1995.

[Cha97] Joel Chadabe. *Electric Sound The Past and Promise of Electronic Music*. Prentice Hall, Upper Saddle River, 1997.

[Col96] Timothy D. Cole. Laser altimeter designed for deep-space operation. *Laser Focus World*, pages 77–86, September 1996.

[Egl94] H. Eglowstein. Almost as good as being there. *Byte Magazine*, pages 173–175, April 1994.

[Eve95] H. R. Everett. *Sensors for Mobile Robots Theory and Application*. A K Peters, Ltd., Wellesley, 1995.

[FW95] C. R. Fisher and S. Wilkinson. Diy: Build the em optical theremin. *Electronic Musician*, 11(5):58–64, 1995.

[Gen91] Jean-Franois Genat. High resolution digital tdc's. In *Proceedings on the First Annual Conference on Electronics for Future Colliders*, pages 229–238. Research Systems Division, LeCroy Corporation, 1991.

[Ger99] Neil Gershenfeld. *The Nature of Mathematical Modeling*. Cambridge University Press, Cambridge, 1999.

[Gra95] Jerald Graeme. *Photodiode Amplifiers Op Amp Solutions*. McGraw-Hill, New York, 1995.

[GSM] N. Gershenfeld, B. Schoner, and E. Metois. Cluster-weighted modeling for time series analysis. *Nature*. (to appear).

[Hec90] Eugene Hecht. *Optics*. Addison-Wesley Publishing Company, Reading, 2 edition, 1990.

[HHO94] K. S. Hashemi, P. T. Hurst, and J. N. Oliver. Sources of error in a laser rangefinder. *Rev. Sci. Instrum.*, 65(10):3165–3171, October 1994.

[KF97] Renate Kempf and Chris Frazier. *OpenGL Reference Manual*. Addison-Wesley Publishing Company, Reading, 2 edition, 1997.

[Mah98] Diana Phillips Mahoney. Laser technology gives video walls a hand. *Computer Graphics World*, 21(10):17–18, October 1998.

[McC88] Gordon McComb. *The Laser Cookbook*. McGraw-Hill, New York, 1988.

[ML98] Richard S. Muller and Kam Y. Lau. Surface-micromachined microoptical elements and systems. In *Proceedings of the IEEE*, volume 86, pages 1705–1720. IEEE, August 1998.

[Par97] Joseph A. Paradiso. Electronic music: new ways to play. *IEEE Spectrum*, 34(12):18–30, December 1997.

[Par98] Joseph A. Paradiso. Getting the picture. *IEEE Computer Graphics and Applications*, 18(3):26–27, May/June 1998.

[PS97] Joseph A. Paradiso and Flavia Sparacino. Optical tracking for music and dance performance. In *Optical 3-D Measurement Techniques IV*, pages 11–18. A. Gruen, H. Kahmen eds., 1997.

[PTR+98] Michael Petrov, Andrey Talapov, Timothy Robertson, Alexei Lebedev, Alexander Zhilyaev, and Leonid Polonshiy. Optical 3d digitizers: Bringing life to the virtual world. *IEEE Computer Graphics and Applications*, 18(3):28–37, May/June 1998.

[Re90] J. M. Reger. *Electronic Distance Measurement.* Springer-Verlag, New York, 3 edition, 1990.

[Rea97] John F. Ready. *Industrial Applications of Lasers.* Academic Press, New York, 1997.

[Ric98] Pete W. Rice. Stretchable music: A graphically rich, interactive composition system. Master's thesis, MIT Media Lab, September 1998.

[RK94] J. Rehg and T. Kanade. Digiteyes: Vision-based hand tracking for human-computer interaction. In *Proc. of hte Worksop on Motion of Non-Rigid and Articulated Objects*, pages 116–22, Austin, November 1994. IEEE.

[RM87] J. Rekimoto and N. Matsushita. Perceptual surfaces: Towards a human and object sensitive interactive display. In *Workshop on Perceptual user Interfaces*, pages 30–32, Banff, October 1987.

[ROMK98] Jun Rekimoto, Masaaki Oka, Nobuyuki Matsushita, and Hideki Koike. Holowall: Interactive digital surface. In *Conference Abstracts and Applications Siggraph 98*, page 108. ACM, 1998.

[SP98] Joshua Strickon and Joseph Paradiso. Tracking hands above large interactive surfaces with a low-cost scanning laser rangefinder. In *CHI98 Extended Abstracts, April 98*, pages 231–232. ACM, 1998.

[SRP98] Joshua Strickon, Pete Rice, and Joseph Paradiso. Stretchable music with laser range finder. In *Conference Abstracts and Applications Siggraph 98*, page 123. ACM, 1998.

[SWD+98] Joshua Smith, Tom White, Christopher Dodge, Joseph Paradiso, Neil Gershenfeld, and David Allport. Electric field sensing for graphical interfaces. *IEEE Computer Graphics and Applications*, 18(3):54–60, May/June 1998.

[Urb95] Ellison C. Urban. The information warrior. *IEEE Spectrum*, 32(11):66–70, November 1995.

[VKY+97] G. Varner, H. Kichimi, H. Yamaguchi, R. Sumner, and G. Blanar. A time expanding multihit tdc for the belle tof detector at the kek b-factory. In *Proceedings of the International Conference on Electronics for Particle Physics*, pages 17–24. Research Systems Division, LeCroy Corporation, 1997.

[Weh97] Aloysisus Wehr. *3D-Laserscanning.* Institute of Navigation, University Stuttgart, Zurich, 1997.

[Wil97] Scott Wilkinson. Phantom of the brain opera. *Electronic Musician*, 13(1), 1997.

[WND96] Mason Woo, Jackie Neider, and Tom Davis. *OpenGL Programming Guide*. Addison-Wesley Publishing Company, Reading, 2 edition, 1996.

[Wre97] C. et. al. Wren. Pfinder: Real-time tracking of the human body. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 19(7):780–785, 1997.