REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	31 March 1998	Final Report: 951130-980301

4. TITLE AND SUBTITLE
Visualization of Circuit Card EM Fields

5. FUNDING NUMBERS

6. AUTHOR(S)
Daniel Zwillinger

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Aztec Corporation
371 Moody Street
Waltham, MA 02154

8. PERFORMING ORGANIZATION REPORT NUMBER
980331

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Department of the Air Force
SA-ALC Test Systems Branch
308 Avionics Circle, Suite 2
Kelly AFB, Texas 78241

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; SBIR report;
distribution unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Printed circuit boards (PCBs) are used in nearly every electrical appliance. Most PCB failures cause differing currents in the traces. This causes PCBs to radiate differing electromagnetic fields. Imaging such radiated fields (equivalently, measuring the field) could be used for error detection. Previous work by Aztec indicated that the electrooptic effect appeared to be sufficiently sensitive for use in a PCB imaging system. Aztec identified an appropriate sensor recently patented by SRICO.
This project was to protoype a system to image the electric field above a PCB and determine the traces from that image. SRICO was to produce a 2×4 array of sensors. Aztec sucessfully constructed (1) an automated test system to measure electric fields above circuit boards, and (2) software to perform image analysis operations on the measured fields. However, after receiving almost one-third of the total funds, SRICO was incapable of prototyping a single (1×1) sensor within 30 dB of their initial specifications. Additionally, SRICO could not control for thermal fluctuations.
Using a short antenna sensor we demonstrated the system's data gathering and image analysis capabilties. As predicted previously, an inferior sensor leads to poor resolution of the traces on a PCB.

14. SUBJECT TERMS

15. NUMBER OF PAGES
131

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
unclassified	unclassified	unclassified	unlimited

FINAL REPORT

VISUALIZATION OF CIRCUIT CARD EM FIELDS

CONTRACT NUMBER F41608–96–C–0116

Aztec Corporation
Suite 104
371 Moody Street
Waltham, MA 02154

781-647-1534
http://www.az-tec.com

31 MARCH 1998

Contents

List of Tables

List of Figures

Executive Summary

This is the final report for Air Force contract F41608–96–C–0116: "Visualization of Circuit Card EM Fields".

Circuit cards (or circuit boards) of one type or another are used in nearly every electrical appliance used by the military or by commerce. This naturally includes computers, which are themselves used in many electrical appliances. To assure quality, all circuit board manufacturers use quality control testing to verify that the circuit board perform appropriately. There exist automated, readily available, and inexpensive testing mechanisms for unpopulated boards. However, for the testing of populated boards, the usual testing techniques are expensive due to the cost of construction of specialized test fixtures.

Most board failures are due to either bad solder joints or faulty components. Either of these errors results in differing currents in the circuit board traces and in the components on the board. These differing currents result in a differing electromagnetic field radiated by the circuit board. This differing electromagnetic field can be used as the basis for a detection mechanism of errors.

In Phase I of this project Aztec evaluated the following physical mechanisms to determine if they could be used to image circuit board electromagnetic fields: the electrooptical effect, the magnetooptical effect, the piezoelectric effect, the electrodynamic effects, and thermal techniques. Aztec determined that sensors using the electrooptical effect (Pockels effect) appear to be sufficiently sensitive for use in a circuit board imaging system. Aztec also performed an extensive review of existing electromagnetic field sensors. Most of the sensors identified could not be applied to measuring circuit board electromagnetic fields, due to frequency limitations or other factors. However, one electrooptical sensor we located, which had recently been patented, seemed ideal for the application.

Phase II of this project was to protoype a system that would automatically sense the field above an active circuit board and perform image analysis to identify the traces on the circuit board. We intended to use the electrooptical sensor identified in Phase I. The inventor of the sensor (SRICO, of Powell, OH) was to produce a 2×4 array of sensors in single package.

Aztec sucessfully constructed an automated test system to gather data from simple circuit boards. Aztec sucessfully constructed software to perform image analysis operations on the measured fields.

However, after receiving almost one-third of the total Phase II funds, the sub-contractor for the electrooptical sensor (SRICO) was incapable of prototyping a single (1×1) sensor within 30 dB of their initial specifications. Additionally, SRICO could not control for the thermal fluctuations of the measured signal.

Without the SRICO high precision sensor, we were forced to use an inferior sensor (a short antenna) to gather data for the image analysis demonstration. As predicted in Phase I, the use of an inferior sensor leads to very poor resolution of the traces on a printed circuit board.

Our conclusions are as follows:

- We have demonstrated that useful information about PCB errors can be obtained from measurement of the electric fields radiated by a circuit board.
- Electrooptic sensors necessary for this application are still in an experimential stage. They are not sufficiently mature for production of a PCB imaging and evaluation system.

1 Introduction

This is the final report for Air Force contract F41608–96–C–0116: "Visualization of Circuit Card EM Fields". The contents of this report are:

- An executive summary (see page 9)
- An overall introduction (this section)
- A discussion of the initial system design (see section 2)
- A discussion of the problems with SRICO trying to obtain a sensor (see section 3)
- A discussion of the revised system design (see section 4)
- A discussion of the data acquisition system (see section 5)
- A discussion of the data acquired (see section 6)
- A discussion of our image processing efforts (see section 7)
- A discussion of state-of-the-art for PCB manufacturing and testing (see section 8)
- A discussion of how to power up an unknown PCB (see section 9)
- A collection of conclusions (see section 10)
- A collection of reference material (see appendix A)
- An explanation of why the system had to be re-designed (see appendix C)
- Copies of all the C++ and Matlab computer programs (see appendix D)
- An index

1.1 Introduction to project concept

Circuit cards (or circuit boards) of one type or another are used in nearly every electrical appliance used by the military or by commerce. This naturally includes computers, which are themselves used in many electrical appliances. To assure quality, all circuit board manufacturers use quality control testing to verify that the circuit board perform appropriately. There exist automated, readily available, and inexpensive testing mechanisms for unpopulated boards. However, for the testing of populated boards, the usual testing techniques are expensive due to the cost of construction of specialized test fixtures.

Most board failures are due to either bad solder joints or faulty components. Either of these errors results in differing currents in the circuit board traces and in the components on the board. These differing currents result in a differing electromagnetic field radiated by the circuit board. This differing electromagnetic field can be used as the basis for a detection mechanism of errors.

The technical need to use EM field detection as a non-destructuve diagnostic tool in Air Force automatic test systems was identified by Jeffrey S. Dean (of SA-ALC/LDAE at Kelly AFB).

The Southwest Research Institute (SWI) performed work on using radiated EM fields to diagnose faulty circuit boards. They performed electric field, magnetic field, and thermal field measurements of circuit boards. They investigated the use of electrooptic materials. To analyze their data and identify faults on circuit cards, they used artificial neural networks, edge finding, FFTs, and image processing. They did not collect data at multiple frequencies, as we did (although their final report recommended the use of data gathering at multiple frequencies—see also [36]). Figure 1 (from [38]) represents the magnetic field above two circuit boards (their magnetometer had a physical diameter of 0.1 inch), one of the two circuit boards had faults. These images, as they stand, would not easily allow a human operator to identify the fault.

Hewitt Associates (see [19]) performed SBIR research on visualizing electric fields from circuit boards using Emscan. They demonstrated the use of Emscan to detecting faulty circuit boards. Due to the intrinsic limitation of the Emscan device (an array of loop sensors 0.3 inches apart), it was not possible to obtain fine detail.

1.2 Previous work by Aztec Corporation

In Phase I of this SBIR project Aztec Corporation[1] evaluated the following physical mechanisms to determine if they could be used to image circuit board electromagnetic fields: the electrooptical effect, the

[1]Then known as "Zwillinger & Associates".

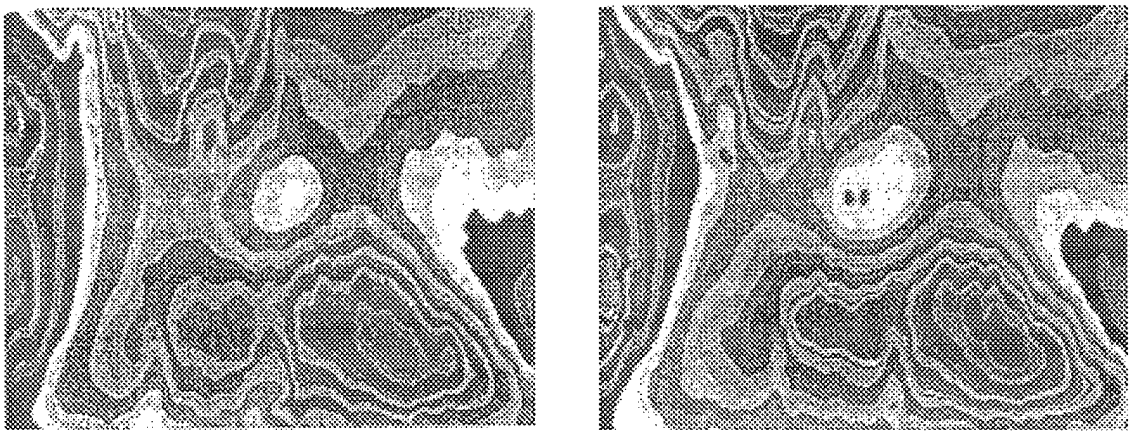


Figure 1: Spence's images (see [38]) of magnetic fields above a printed circuit board: good board (left), faulted board (right)

magnetooptical effect, the piezoelectric effect, the electrodynamic effects, and thermal techniques (see [49]). Aztec determined that sensors using the electrooptical effect (Pockels effect) appeared to be sensitive enough to create images understandable by humans of circuit boards. Aztec also performed an extensive review of existing electromagnetic field sensors. Most of the sensors identified could not be applied to measuring circuit board electromagnetic fields, due to frequency limitations or other factors. However, one electrooptical sensor we located, which had recently been patented, seemed ideal for the application.

The conclusions from the Phase I report are listed below. They are summarized in Table 1, listed in order of feasibility for near term applications.

1. Advantages of linear sensing technique

   Different sensor technologies may be described as either linear (that is, the effect scales with the electromagnetic fields on the circuit board) or quadratic (that is, the effect scales with the square of the electromagnetic fields on the circuit board). Linear methods include electrooptical and piezoelectric techniques. Quadratic methods include the thermal techniques. For low power fields, the linear methods generate a much larger sensed signal than the quadratic methods, and therefore are more useful for circuit board applications.

2. Electrooptical measurement

   The most well documented technique for sensing high frequency electromagnetic fields employs electrooptic materials to generate an optical phase shift through the Pockels effect. This effect is one in which an applied electric field causes a change in the index of refraction. Imaging a circuit board would be done with an array of electrooptic waveguides located close to the circuit board. The normal electric field on the board would produce an optical phase shift in the light waves traversing the electrooptical waveguides. To detect the optical phase shift a Mach–Zehnder interferometer would be used with phase–coherent detection techniques. These detection techniques have been demonstrated in laboratory facilities. The effect will produce an amount of phase shift which we judge to be sufficient for imaging the fields on a circuit board.

3. Magnetooptic Faraday effect

   When used with materials having very large Verdet constants, the magnetooptic Faraday effect can provide amounts of polarization rotation of a linearly polarized optical beam which are comparable to the phase shifts produced by the electrooptic effect. Dilute magnetic semiconductors, a recently created kind of material, have extremely large Verdet constants, on the order of 100,000 when the material is held at very low temperature, i.e., 5 degrees Kelvin. Certain ferromagnetic semiconductors, such as europium oxide, have extremely large equivalent Verdet constants at room temperature and may

Electrooptical technique	This technique appears to be very promising for imaging of low power digital circuit board electromagnetic fields. A sensor array concept that achieves a linear spatial separation of 0.03 inches has been described (recall that Emscan achieves only 0.3 inches). This is a high–frequency technique.
Magnetooptical techniques	The polarization rotation angles obtainable with conventional diamagnetic and paramagnetic materials are near the limit of presently detectable polarization rotation angles. If dilute magnetic semiconductors at low temperature are used the polarization angle rotation will be about 100 times larger and comparable to the phase shifts produced by the electrooptical effect. Several ferromagnetic semiconductor materials may also provide large polarization rotation at room temperature. This is a high–frequency technique.
Electrodynamic force technique	For piezoelectric detection this technique could work if the frequency of detection is between 3 Hz and 100 KHz, or higher (up to hundreds of MHz) if used in a heterodyne mode. For optical detection of the deformations, the measurement time for commercial off the shelf devices is too long (the material experiences significant thermal expansion during the measurement).
Direct thermal imaging	This technique can be used directly, and applications for circuit boards have already been developed by others
Piezoelectric techniques	For the piezoelectric sheet technique the sheet deformations are too small to be detected. For the piezoelectric slurry technique the temperature rise is too small to be useful.

Table 1: Suitability of measurement techniques for circuit board imaging (conclusions from Phase I report)

provide suitable performance. An important feature of magnetooptic imaging is that the magnetooptic material is a simple sheet close to the circuit board.

4. High frequency measurements with heterodyne detection

   Several measurement techniques, which employ low frequency mechanical motion, have been shown to be effective when used in a heterodyne mode. In this mode, a high frequency field mixes with a high frequency field to be measured in a nonlinear device. The force at the difference frequency excites the device at it's resonance. Many of these heterodyne measurements have been demonstrated. The demonstrations have all been with large sensor devices, which would not provide the spatial resolution required for imaging the small scales on circuit boards. A heterodyne technique employing a matrix of small piezoelectric rods does have the potential for the required spatial resolution.

Aztec further recommended that research be pursued in the following areas (listed in prioritized order):

1. Determine if and how the Emscan device will be modified to increase it's resolution. With a sufficient increase in resolution, it may be practical to use this product on low power digital circuit boards. This course of action might have the most immediate pay-back from the Air Force point of view.

2. Explore Schlumberger's plans to package and sell their patented measurement system which has been designed for the visualization of circuit board electromagnetic fields. (Their patent was less than one year old.)

3. Specify and select, in consultation with experts in the Air Force and in industry, a class of circuit boards and defining operating parameters. Relevant parameters needed for detailed sensor design will be measured from a representative of this class. This process will determine some of the parameters that were only estimated in the Phase I final report, for example

   At which frequencies is the spectral energy concentrated?
   What is the trace separation on the circuit board?
   How close can the sensing sheet come to the circuit board?

4. Design and construct in conjunction with SRICO a prototype electromagnetic field sensor based on the electrooptical effect in the form of a linear array. The SRICO sensor appears to be adequate for the measurement of electric fields produced by a low power digital circuit board. While a two-dimensional array of sensors is the eventual product, it would be prudent to first construct a single linear array of sensors.

5. Determine the magnetooptical pperformance available from dilute magnetic semiconductor materials and from ferromagnetic semiconductors. While these materials show great promise, because of their large Verdet constants, many important practical parameters need to be explored.

This report is for Air Force contract F41608–96–C–0116, which was awarded to pursue option number 4.

1.3 Goals of phase II

Phase II of this project was to prototype a system that would automatically sense the field above an active circuit board and perform image analysis to identify the traces on the circuit board. We intended to use the electrooptical sensor identified in Phase I. The inventor of the sensor (SRICO, of Powell, OH) was to produce a 2×4 array of sensors in single package.

Aztec successfully constructed an automated test system to gather data from simple circuit boards. Aztec successfully constructed software to perform image analysis of measured fields.

However, after receiving almost one-third of the total Phase II funds, the sub-contractor for the electrooptical sensor (SRICO) was incapable of prototyping a single (1×1) sensor within 30 dB of their initial specifications. Additionally, SRICO could not control for the thermal fluctuations of the measured signal.

Without the SRICO high precision sensor, we were forced to use an inferior sensor (a short antenna) to gather data for the image analysis demonstration. As predicted in Phase I, the use of an inferior sensor leads to very poor resolution of the traces on a printed circuit board.

Our conclusions in brief are as follows (these are expanded in section 10):

- Useful information about PCB errors can be obtained from measurement of the electric fields radiated by a circuit board.
- Electrooptical sensors necessary for this application are still in an experimental stage. They are not a sufficiently mature technology to use in this application.

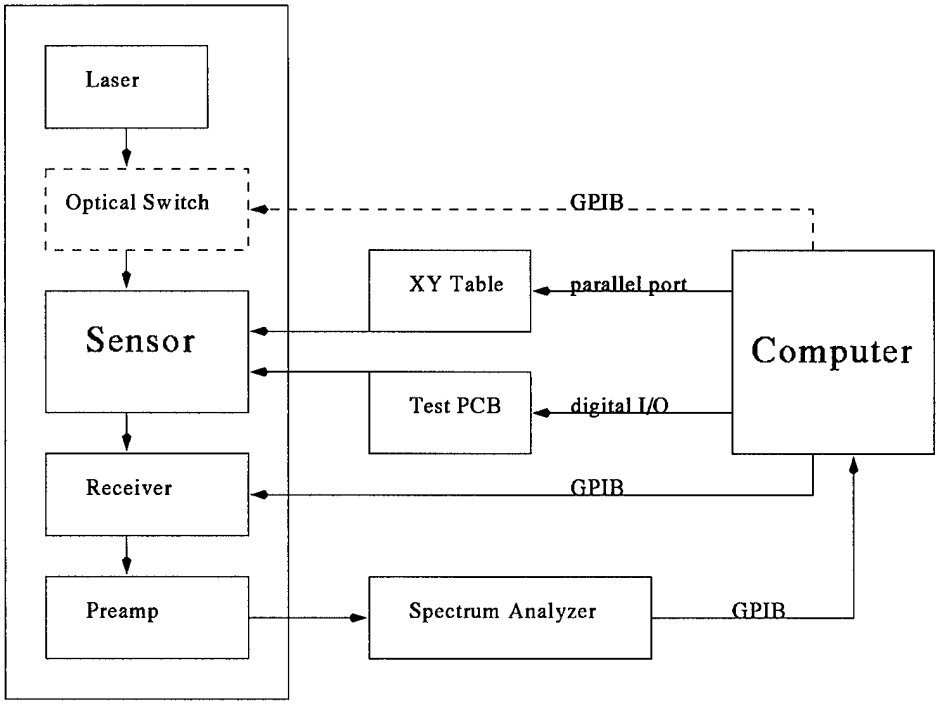


Figure 2: Initial overall hardware system design

2 Initial design of sensor system

The overall design of the initially designed hardware system is shown in Figure 2. This Figure shows the main features of the system, and the type of data lines connecting the components.

1. The computer drives the XY table. This moves the sensor over different portions of the PCB.
2. The computer drives the PCB. The PCB radiates electromagnetic energy.
3. The computer drives the receiver, telling it when to receive data.
4. The electrooptical sensor uses the laser to detect the radiated electromagnetic field. This received signal is passed through a preamp and then a spectrum analyzer.
5. The computer saves the data acquired.
6. The optical switch is shown dotted since the system could be fully run without using an optical switch. However, to efficiently use the sensor array that was contracted for would require multiplexing the laser to the array of sensing elements.

Photographs of the system actually constructed are in figures 3 and 4.

2.1 Electrical system

The electrical system is shown in more detail in Figure 5. The specific components of the detection system are described below.

LightWave 125 laser

The LightWave 125 laser is a 150 mW laser operating at 1300 nm. It is a fiber-coupled diode-pumped, single-frequency, non-planar ring laser. It has a polarization maintaining fiber pigtail, a 30 dB isolator, and an industry standard FC/PC connector. The laser has low noise, narrow linewidth, and frequency tuning ability. The laser system includes control electronics, a noise reduction system, and power supply. The amplitude noise is 0.02% rms over the range 10 Hz

Figure 3: Photograph of workbench with positioning table

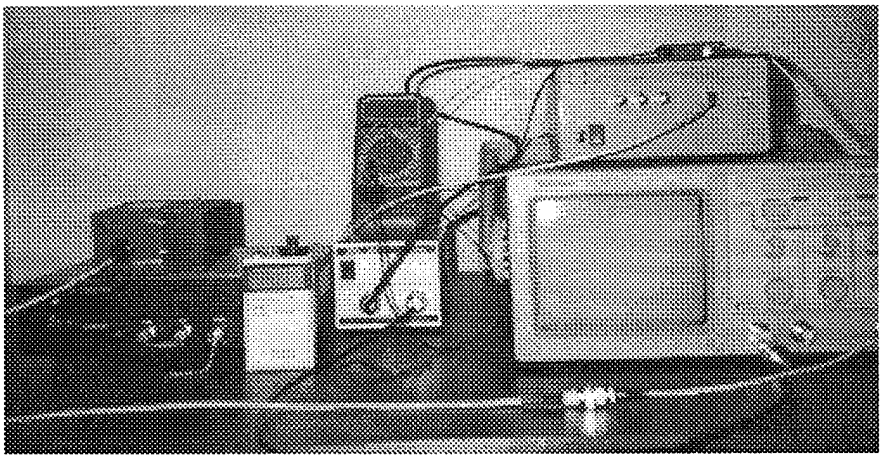


Figure 4: Photograph of test equipment

to 2Mz and less than -165 dB/Hz above 10 MHz (in the shot noise limited regime). The laser includes a noise reduction feedback circuit. There is a noticeable noise peak in the laser output near 300 kHz. The laser linewidth is less than 5 kHz, measured over 1 msec. The laser also has thermal and piezoelectric tuning options.

Optical fiber

The optical fiber is a special new grade of polarization maintaining fiber, with very low loss. It is designed to be compatible with standard communication grade fibers and lithium niobate optical waveguides. It is FS-CG-6121 fiber from 3M Corporation. The minimum bending radius of these fibers is 12 mm.

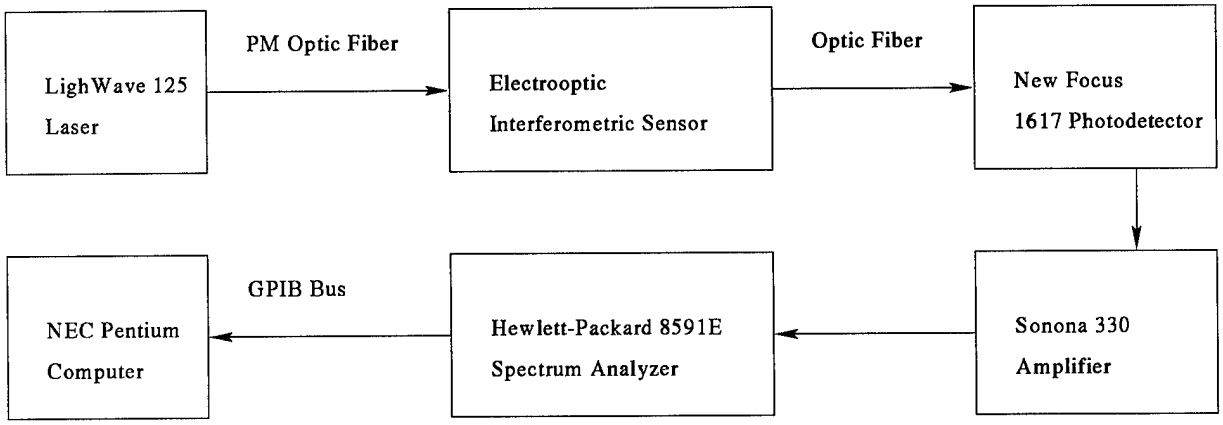


Figure 5: Detection system

Sonoma 330 amplifier

The Sonoma amplifier is a broadband amplifier with flat gain. Its specifications are bandwidth 10 kHz – 2.5 GHz, 140 ps rise time, 20 dB gain, gain flatness error ± 0.75 dB max., and noise Figure 8 dB.

New Focus 1617 photoreceiver

The New Focus photoreceiver is a wideband DC-coupled InGaAs/PIN detector with a built-in amplifier. The input to the photoreceiver is through a FC connector bulkhead. The receiver specifications are wavelength 800-1800 nm, 3 dB bandwidth of 800 MHz, rise time 500 ps, detector diameter 0.1 mm, maximum conversion gain 200 V/W, maximum responsivity 0.8 A/W, transimpedance gain 250 V/A, output impedance 50 Ω, minimum NEP 40 pW/$\sqrt{\text{Hz}}$, and continuous wave saturation power 20 mW at 950 nm.

Hewlett Packard 8519E spectrum analyzer

The 8591E spectrum analyzer has the following specifications: frequency range 9 kHz – 1.8 GHz, frequency accuracy 210 Hz, resolution bandwidth range 30 Hz – 3 MHz, average noise level (in the narrowest resolution bandwidth) −130 dBm, optimum dynamic range 77 dB, and amplitude accuracy 1.7 dB. It has a GPIB interface for communication with the computer.

2.2 Positioning system

The specific components of the positioning system are described below.

Arrick XY-9 positioning table

The Arrick XY-9 is a belt-driven table. The table has an aluminum top plate which rides on polished steel shafts and bronze bearings. It can position payloads up to 15 lb. The table has home switches for position calibration on each axis. The specifications for the table are repeatability 0.005", accuracy 0.010" per foot of travel, resolution 0.005", and speed of 2" per second with a 2 lb payload. The top plate, for mounting the sensor, is 6" by 4".

Arrick MD2a dual stepper-motor system

The system contains drive electronics, two stepper-motors, two home switches, and software. Communication to the driver is either through a centronics parallel port or through a digital input/output port. The motors are size #23, 2.25" diameter. They have a resolution of 0.9 degree half-steps (400 per revolution), maximum speed of 10,000 half-steps per second, and holding torque of 50 in/oz.

Arrick software

Arrick provides a turnkey software package for operating the stepper-motor system. In addition, a subroutine library of the motor software drivers is provided in several languages for inclusion in third-party programs. The C subroutine library was converted to a form suitable for use in a LabWindows/CVI program. This mostly involved modularizing the code.

2.3 Software system

The specific components of the software system are described below.

LabWindows/CVI

LabWindows/CVI is an integrated visual development environment for data acquisition and instrument control. It is produced by National Instruments. It simulates a PC based virtual instrument. The programming is done in C. LabWindows/CVI is built around hardware control libraries for GPIB, VXI, serial, and plug-in data acquisition instrumentation and includes extensive analysis and user interface programming tools.

System controller software

This program is used for taking data from the electrooptic sensor. This program controls the Arrick stepper-motor driver and therefore the positioning table and the location of the sensor. It also controls the Hewlett Packard spectrum analyzer, initializing the analyzer and capturing data, and transferring the data to the computer over a GPIB bus.

The user interface displays the regions that are accessible to the positioning table while under software control. There is a programmable "allowed region" which sets a software limit on the movement of the table, and a user controllable "target region" which defines the region for automated data taking. The user interface displays the position in absolute, allowed region coordinates.

The program lets the user move to any point in the allowable region, take data at any point, display the data, move through a two dimensional grid of arbitrary size and density.

A detailed description of this software (including images of all input screens) is in section 5.

2.4 Sensor system

As reported in [49], Srico[2] has expertise in electrooptic effect and Mach–Zehnder interferometers (see [39]). US patent number 5,267,336 was awarded for their "Electro-optical sensors for detecting electric fields" on 30 November 1993.

Srico has developed a unique method of creating an electrode-less Mach–Zehnder interferometer in lithium niobate by combining a titanium diffusion step followed by a proton exchange process. The titanium diffusion in selected sections of the crystal in a +Z oriented substrate reverse poles that section. Subsequently the Mach–Zehnder waveguide structure is created at a low temperature by using proton-exchange waveguide techniques. A graphical illustration of the active components in Srico's sensor is shown in Figure 6 along with typical dimensions. The device is made from an electrooptical material, is about 50 mm long by 500μ wide, and is fabricated on a plate 1 mm thick. The lines shown are about 5μ deep. The diamond-shaped area has upper and lower "legs" (shown darker). The upper leg is doped so that it is "oppositely poled" from the lower leg.

A light ray is injected from the left into the device. This ray is split and each ray travels down one side of the diamond-shaped area. The rays are combined to determine a phase difference. If there is no ambient electric field, then there is a zero phase difference between the two light rays. If there is an ambient electric field, then there will be a detectable phase difference since the optical properties in each leg will be different. Because of the orientation of the opposing electrooptical components, this device can detect electric fields that are in to (or out of) the page.

[2]Srico; 664 Petworth Court; Powell, OH, 43065

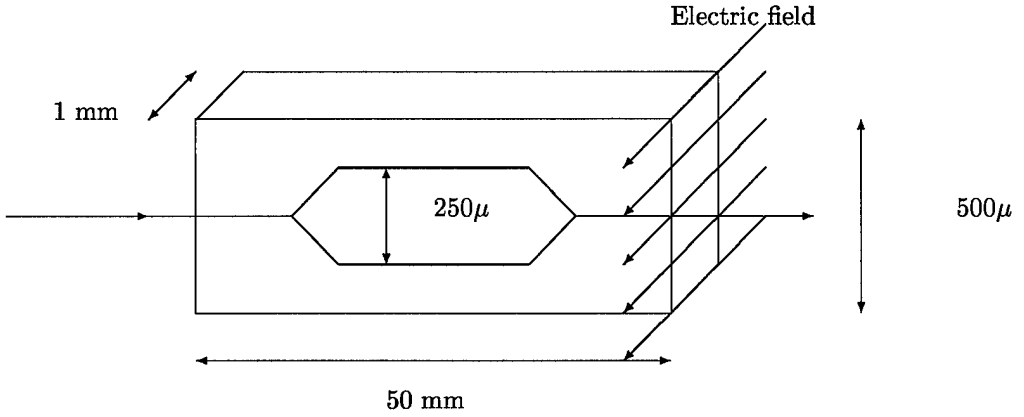


Figure 6: Srico's electromagnetic field sensor (not to scale)

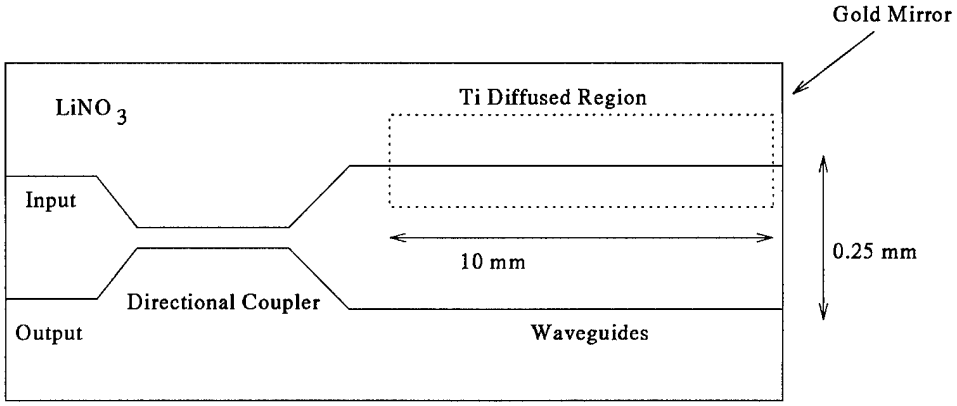


Figure 7: New design for sensor layout for PCB applications

Note that the device has no metallic electrodes, hence there is no perturbation of the electromagnetic field being measured. Connection of fiber optic cables requires that a 5 mm long connector be attached where the light ray enters the device.

Naghski *et al* [29] describes the results of tests on fabricated Srico devices. They indicate a measured minimum detectable field of 0.22 V/m/$\sqrt{\text{Hz}}$. This is interpreted as follows: if the bandwidth of the measurement system (the bandwidth around some clock frequency) is 1 Hz, then the device can detect fields as small as 0.2 Volts per meter. If the measurement bandwidth around a center frequency were 100 Hz, then the device can detect fields as small as 2 Volts per meter. The product literature from Srico indicates that maximum detectable electric field is 150 kV/m. Furthermore, Srico had estimated that:

- the measurement time is on the order of 10–30 milliseconds.
- the repeatability of measurements using the same probe is about 5%
- two ostensively identical sensors may disagree by as much as 5%

The sensor that Srico had been using is difficult to use for PCB applications, since optical fibers are attached at opposite ends of the device. What was needed was a device that had optical fibers only on one end, so that the other end could be placed close to a PCB. In conjunction with Srico, we designed such a one-ended detector, it is shown in Figure 7.

Note that the sensor shown in Figure 7 (and abstracted to Figure 8) could also, in principle, be constructed as a linear array, as shown in figure 9. Fabrication requirements (the size of a 3 inch silicon wafer)

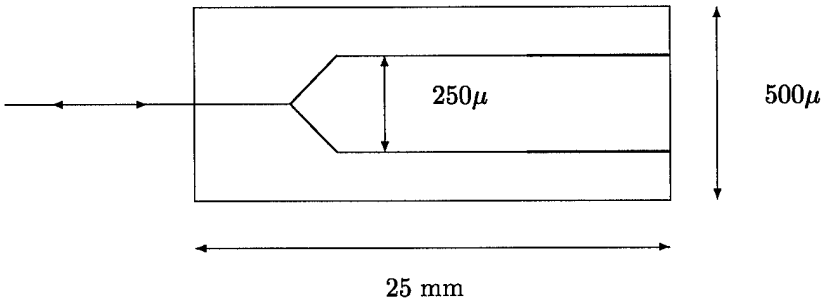


Figure 8: A modified Srico electromagnetic field sensor

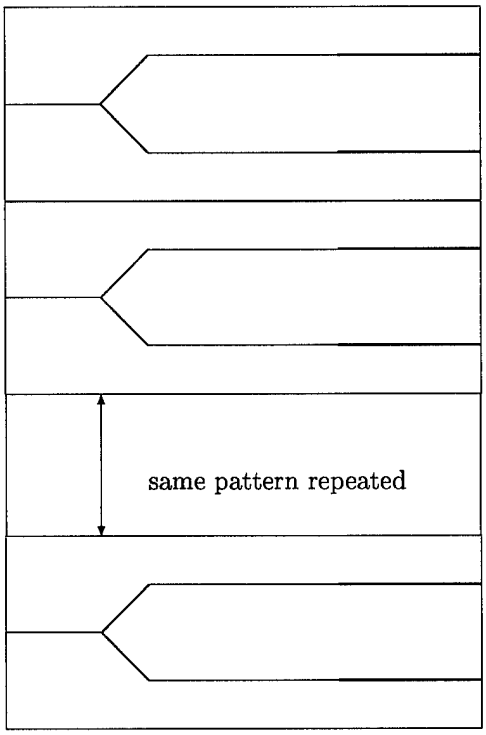


Figure 9: An array of modified Srico electromagnetic field sensors

could restrict the length of this array to about 2.5 inches (which could contain, possibly, 100 individual sensors). Since the array is only 1 mm thick, it is possible to laminate several arrays together to obtain a two-dimensional array of sensors. If 100 arrays were combined the resulting two-dimensional array could, conceivably, have 10,000 sensors in an approximate area of 2.5 inches by 4 inches. This many sensors in 10 square inches results in an average of 30 sensors to the linear inch.

2.4.1 Type of data acquired

In [49] we mentioned that we would be content with comparing two-dimensional images.[3] However, the signal to noise ratio is far worse in the time domain that it is in the frequency domain, and there is always the

[3]Since the fields are temporal the input data–in the time domain–consists of a sequence of two-dimensional images. Since two sequences of images can be compared one-by-one, we now restrict attention to a single two-dimensional image.

problem of synchronization of signals. Hence, we believe that obtaining frequency information is superior, and allows better "image" comparison and analysis. Note, however, that there are several ways in which our three-dimensional data set can be used to create two-dimensional images. For example, the maximum value over all frequency bins could be taken, or the average response over a specific range of frequency bins could be taken.

However, the full benefit of our system will only be obtained when we utilize the complete three-dimensional data sets that we collect. The only difficulty with this approach is that we are then working with rather large data sets (several megabytes for even modest experiments). One way to reduce the amount of experimental data that needs to be stored is via the use of principal components analysis (see section 7.3.1).

2.5 Sensor background

The sensor is an integrated optic device using a Mach–Zehnder interferometer in a lithium niobate electrooptic crystalline material. The electric field sensor uses the electrooptic properties of lithium niobate to modulate the phase of the light propagating through each arm of the Mach–Zehnder interferometer. This phase modulated optical signal is converted to intensity modulation at the output of the interferometer. Conventional, integrated optic electric field sensors use electrodes to generate phase modulation in the two arms of the interferometer. This sensor uses a selective reverse poling of one arm by high temperature diffusion of titanium into the lithium niobate. This causes simultaneous opposing changes in phase in each arm of the interferometer in the presence of an electric field. The optical waveguides of the integrated interferometer are produced by low temperature proton-exchange process.

2.5.1 Pockels effect

The sensor is a crystal of lithium niobate ($LiNbO_3$). $LiNbO_3$ is a ferroelectric material, in which variations in the local electric potential or electric field cause variations in the crystal structure. The crystal structure variations cause changes in the index of refraction. This effect is commonly called the Pockels effect or the linear electrooptic effect. It occurs in crystals with noncentered symmetries. A general equation for the relation between the electric field and the change in index of refraction is

$$\Delta\left(\frac{1}{n^2}\right)_{ij} = \sum_k r_{ijk} E_k,$$

where $\Delta\left(\frac{1}{n^2}\right)_{ij}$ is a second rank tensor showing the change in the relative permittivity, E_k is the k^{th} component of the local electric field, and r_{ijk} is the linear electrooptic coefficient tensor. Materials with a large value for some element of the r_{ijk} tensor have a large Pockels Effects. The electrooptic coefficient tensor for $LiNbO_3$ has only four independent elements. The largest of these is called r_{33}. This coefficient couples between an electric field pointing along the c axis of the crystal (also called the z axis) and the component of the index of refraction (in zero electric field) of light polarized along the same axis. For historical reasons, the index of refraction for light polarized along this axis is called the extraordinary index n_e. The coupling between the c axis component of the electric field and the variation in the extraordinary index is described by the simpler equation

$$\Delta n_e = -\frac{1}{2} n_e^3 r_{33} E_{zi},$$

where E_{zi} is the electric field inside the crystal. $LiNbO_3$ has a large relative dielectric coefficient ϵ_r which reduces the internal electric field with respect to the external field E_z. The extraordinary index in terms of the external field is

$$\Delta n_e = -\frac{1}{2} n_e^3 r_{33} \frac{E_z}{\epsilon_r}.$$

The variation in the extraordinary index is proportional to the component of the electric field along the crystalline c axis.

2.5.2 Interferometer sensitivity to electric fields

An interferometer measures changes in the phase of the light propagating through it. The phase changes by 2π every wavelength. Recalling that the wavelength for light polarized along the extraordinary axis is λ/n_e (where λ is the free space wavelength), the phase change for light traveling a distance L through one leg of the interferometer is

$$\Delta\phi = -\frac{\pi}{\lambda\epsilon_r} n_e^3 r_{33} L E_z.$$

The device operates in a push-pull fashion. The phase change in the second leg is opposite that of the first leg. Therefore the total phase change is

$$|\Delta\phi_t| = |\Delta\phi_1 - \Delta\phi_2| = \frac{2\pi}{\lambda\epsilon_r} n_e^3 r_{33} L E_z$$

For this system, the wavelength is $\lambda = 1300$ nm, the electrooptic coefficient is $r_{33} = 3 \times 10^{-11}$ m/V, the extraordinary index of refraction is $n_e = 2.15$, and the relative dielectric constant is $\epsilon_r = 28$. The sensor is designed to have a reverse poled region length L = 1 mm. One measure of the sensor performance is the external electric field required to produce a π radian phase shift. This is given by

$$E_\pi = \frac{\epsilon_r \lambda}{2 n_e^3 r_{33} L}$$

which in this system is 61×10^6 V/m. The device sensitivity $\frac{\Delta\phi_t}{\Delta E}$, measured as phase shift per applied electric field, is 51 nanoradians per (V/m).

2.5.3 Power output by interferometer

The power output by an interferometer depends on the total phase change $\Delta\phi_t$, which is the difference in phase change along two distinct paths. The total power output

$$P = P_s \cos^2\left(\frac{\Delta\phi_t}{2}\right),$$

where P_s is the carrier power. The carrier is the unmodulated (i.e., in zero electric field), DC output from the laser reaching the photoreceiver. In the presence of an electric field, the interferometer modulates the power by a phase $\Delta\phi$. Note that this is a nonlinear dependence. For certain values of $\Delta\phi_t$ the power is insensitive to variations in the total phase change, whereas at other values of $\Delta\phi_t$ the power is almost a linear function. The zero electric field value of $\Delta\phi_t$ is called the operating point. Ideally the operating point is close to an odd multiple of $\pi/2$ (i.e., $\pi/2$, $3\pi/2$, ...), the linear regime of the output power function. In this regime, the finite frequency components of the power are approximately

$$P = P_s \frac{(\Delta\phi)^2}{2}. \quad (1)$$

The cosine squared behavior of the output power function limits the range over which the sensor can be used as a linear device. In practice, electric fields large enough to move the sensor from the linear regime to the nonlinear regime are unlikely to be encountered in the study of circuit boards. A more pressing issue is that due to material property issues and temperature dependence, the operating point may be in the low sensitivity nonlinear regime. This will be discussed in greater detail elsewhere.

Equation 1 shows that the signal to noise ratio (SNR) is the carrier to noise ratio (CNR) times the modulation amplitude, i.e., SNR = CNR $(\Delta\phi^2)/2$ where $\Delta\phi$ is the peak phase deviation.

The minimum detectable peak phase deviation $\Delta\phi_{\min}$ occurs when SNR = 1, i.e., $\Delta\phi_{\min} = \sqrt{\frac{2}{\text{CNR}}}$. Here $\Delta\phi_{\min} = 1.6$ nanoradians per root Hertz. The CNR is 179 dB (this is calculated below). The minimum detectable electric field $E_{\min} = 0.031$ V/m/$\sqrt{\text{Hz}}$. The minimum detectable electric field for 1 GHz bandwidth is about 1000 V/m. Note that the electric field 1 mm away from a 5V trace is within an order of magnitude of 1000 V/m, and is therefore measureable over the full bandwidth.

The maximum SNR occurs when $\Delta\phi$ is approximately one radian (≈ 0.95 radian). Then SNR = 175.5 dB.

2.5.4 Photoreceiver CNR

The signal-to-noise of the sensor system depends on several factors including the stability of the laser, the operating point of the sensor, the phase modulation index of the interferometer, and the carrier-to-noise ratio of the photoreceiver. The signal-to-noise ratio is SNR $= \frac{\Delta\phi^2}{2}$ CNR where $\Delta\phi^2$ is the peak phase deviation, and CNR is the carrier-to-noise ratio of the photoreceiver.

The carrier-to-noise ratio of the photoreceiver is given by

$$\mathrm{CNR} = \frac{2\left(\frac{\eta e P_s M}{h f_c}\right)^2}{\left[e\left(\frac{\eta e P_s M}{h f_c} + I_d\right) M^{2+x} + \frac{2kTF}{R_L}\right] B_n}$$

where F is the noise Figure of amplifier, I_d is the dark current, M is the avalanche gain (equal to 1 for a PIN diode), P_s is the received optical power in each beam (75 mW in our system), R_L is the effective load resistance, η is the quantum efficiency (typically 0.67), e is the charge (1.6×10^{-19} C), f_c is the optical carrier frequency (2.3×10^{14} Hz for 1300 nm light), h is Planck's constant (6.6×10^{-34} J s), k is Boltzmann's constant (1.4×10^{-23} J/K), x is the excess noise factor (equal to 0 for a PIN diode), and B_n is the noise equivalent bandwidth ($\pi/2$ times the -3dB system bandwidth).

If the photoreceiver is shot-noise limited, then only the first term in the denominator is relevant and the above expression simplifies to

$$\mathrm{CNR} = \frac{2\eta P_s M}{h f_c B_n}.$$

For this system, shot-noise limiting occurs if the power reaching the photoreceiver is above 1mW. Substituting the above values into this equation yields CNR = 179 dB/Hz.

As discussed above, the optimum phase modulation index of the interferometer occurs when the operating point is in the linear regime of power output.

2.5.5 Sensor geometry

The coordinate system we adopted for the sensor system is the same as the coordinate system adopted by Srico for the crystal itself. The sensor that Srico produces is based on what is called a Z-cut crystal, i.e., the crystal is cut along a plane perpendicular to the z crystalline axis. The interferometric waveguides diffused into the crystal support only TM mode light, which has an electric field vector aligned in the z direction. Only the local z component of the test electric field couples to the sensor.

In Figures 10, 11 and 12 we have shown a trace aligned with the sensor device. Note that x is "up", y is along the axis of the trace, and z measures the "horizontal" distance from the axis of the trace. For this trace, d is the "horizontal" distance from the trace to the sensor, and h is the "vertical" distance from the trace to the sensor.

In practice, of course, the trace could be rotated anywhere in the x-y plane.

2.5.6 Prediction of sensor performance

Knowing the sensor response to a "point source" is needed when deconvolution filters are used in the imaging process (see section 7.4), as it is needed to extract the traces from the images. This section approximates this response.

Maxwell's equations for an anisotropic media can be written (assuming that the electric charge density, ρ, is zero) as:

$$\begin{aligned} \nabla \cdot \mathbf{D} &= 0 \\ \nabla \times \mathbf{E} &= 0 \end{aligned} \tag{2}$$

where

- **D** is the electric displacement (units of Coulombs/m^2)
- **E** is the electric field strength (units of Volts/m)

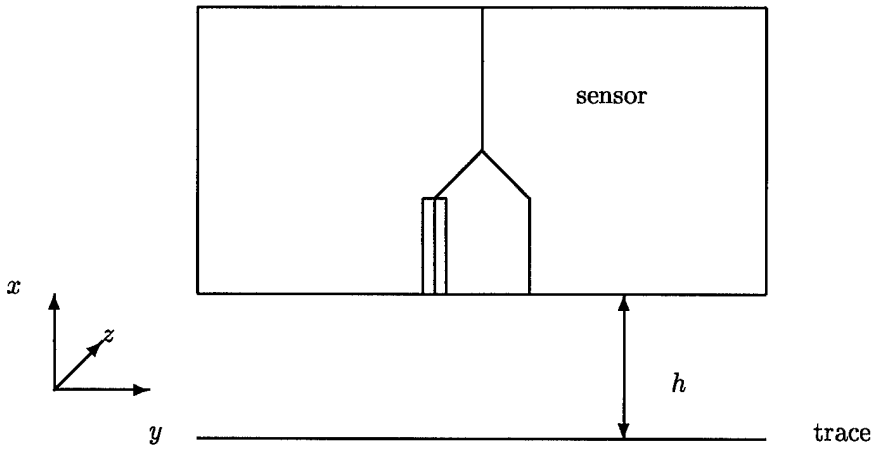


Figure 10: Geometry for sensor: side view

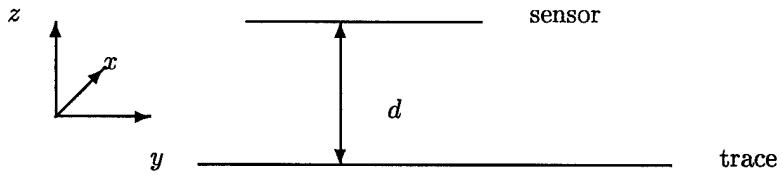


Figure 11: Geometry for sensor: top view

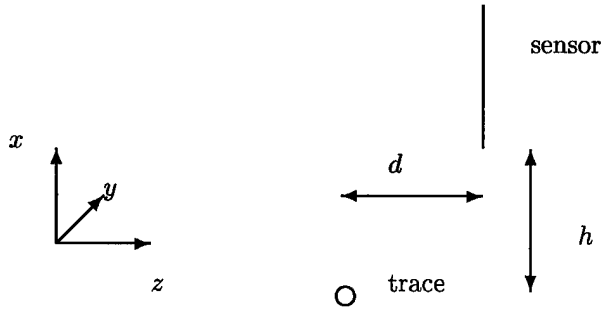


Figure 12: Geometry for sensor: edge view

These quantities are related by $\mathbf{D} = \epsilon\mathbf{E}$, where ϵ is the permittivity tensor. In free space $\epsilon_{xx} = \epsilon_{yy} = \epsilon_{zz} = \epsilon_0$ where $\epsilon_0 = 8.85 \times 10^{-12}$ farads/m is the permittivity of free space. Srico has supplied the values for the permittivity tensor in the crystal from which the sensor is made:

$$\epsilon_{xx} = \epsilon_{yy} = 84.6\,\epsilon_0 \qquad \text{and} \qquad \epsilon_{zz} = 29.1\,\epsilon_0 \tag{3}$$

Equation (2) is equivalent to $\mathbf{E} = -\nabla\Phi$, where Φ is the potential (in units of Volts). The above equations can be combined to produce

$$\epsilon_0 \frac{\partial^2\Phi}{\partial x^2} + \epsilon_0 \frac{\partial^2\Phi}{\partial y^2} + \epsilon_0 \frac{\partial^2\Phi}{\partial z^2} = 0 \qquad \text{in free space}$$

$$\epsilon_{xx} \frac{\partial^2\Phi}{\partial x^2} + \epsilon_{yy} \frac{\partial^2\Phi}{\partial y^2} + \epsilon_{zz} \frac{\partial^2\Phi}{\partial z^2} = 0 \qquad \text{in the crystal}$$

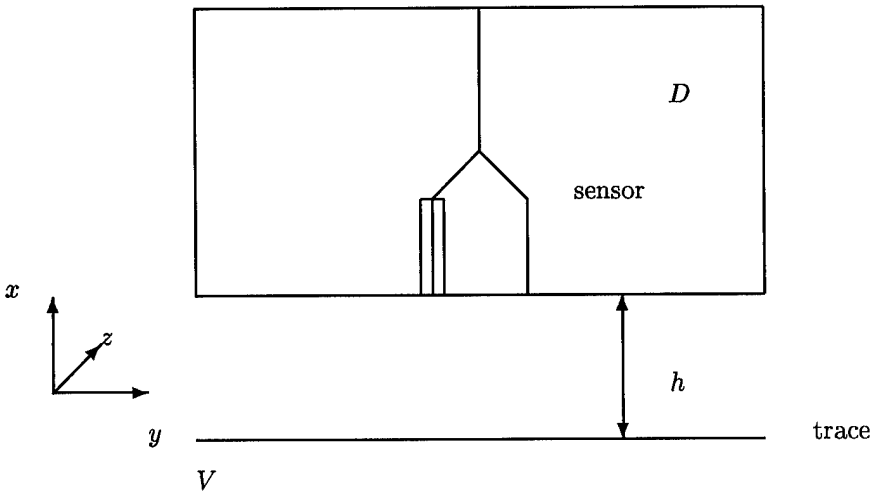


Figure 13: Idealized geometry for sensor/trace combination

Note that Φ, the parallel component of the gradient $(\nabla_{||}\Phi)$, and the perpendicular component of the gradient scaled by the permittivity $\left(\frac{\nabla_{\perp}\Phi}{\epsilon}\right)$ are continuous at the interface where the crystal meets the air. Defining the variable of interest to be $W = \epsilon_0\Phi$, the equations become:

$$\begin{aligned} W_{xx} + W_{yy} + W_{zz} &= 0 \quad \text{in free space} \\ 84.6W_{xx} + 84.6W_{yy} + 29.1W_{zz} &= 0 \quad \text{in the crystal} \end{aligned} \tag{4}$$

Approximations

We presume a trace and a coordinate system as described in section 2.5.5. We assume that the geometry is infinite and uniform in the y direction, so that all derivatives with respect to y may be discarded.

If the dielectric constant of the sensor were equal to unity (which is not the case), then the sensor response could be simply and analytically determined as shown below.

Simple analysis

Here we analyze the sensor response assuming the dielectric constant of the sensor is equal to one (which is *not* a correct assumption). In this case, the geometry is as portrayed in Figures 10–12 and 13. We assume

- ϵ is the dielectric constant of the sensor (assumed to be unity)
- V is the voltage on a trace
- h is the height from the trace to the sensor
- r is the distance from the trace to the sensor
- r_0 is the diameter of the trace

The solution to Laplace's equation (that is, equation (4)) in this case (i.e., $\nabla^2 W = 0$ everywhere) is $W = \epsilon_0 V \log\left(\frac{r}{r_0}\right)$, where $r = \sqrt{x^2 + z^2}$. (This result is used in section 4.3.) Since the sensor is only responsive to $E^{(z)}$, we only need to compute that term: $E^{(z)} = -\partial_z\Phi = -\frac{1}{\epsilon_0}\partial_z W = -V\partial_z \log\left(\frac{r}{r_0}\right) = -V\frac{\partial_z r}{r}$. Hence $E^{(z)} = -\frac{Vz}{r^2}$.

For the interferometric measurement, the response is the absolute value of the difference in the phase changes in the doped and undoped legs of the sensor (see Sriram [39, equation (13)]). In our sensor, the phase changes in the legs differ only in sign, so the measured response is given by $\left|2E^{(z)}\right| = \left|\frac{2Vz}{z^2 + h^2}\right|$.

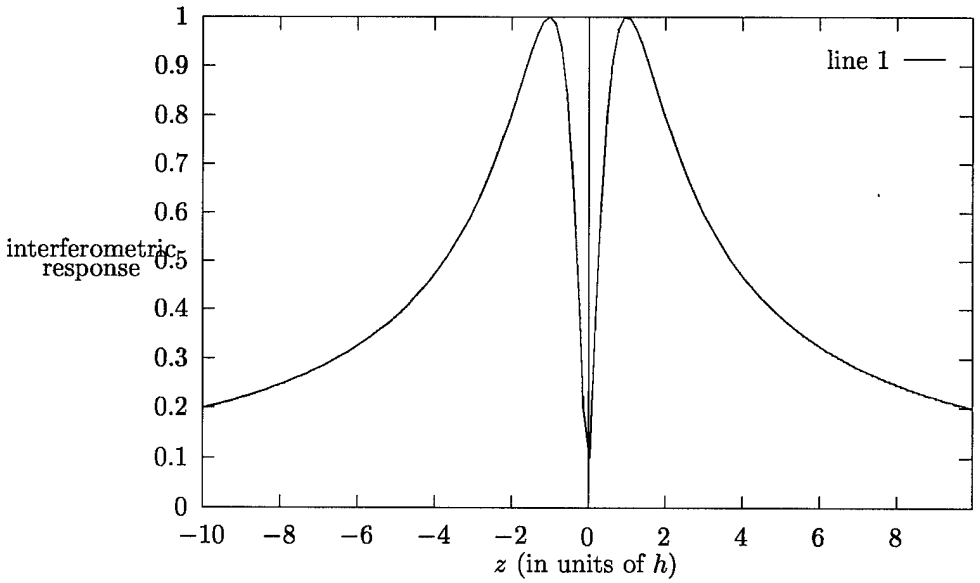


Figure 14: Response of idealized sensor using $\epsilon = 1$, $V = 1$ and varying z.

A plot of this response is shown in Figure 14; it clearly shows that the response is zero when $z = 0$ and $= \pm\infty$. However, our experimental results have not shown a zero response when z is near 0. We believe this is due to two factors. Firstly, the sensing crystal is not perfectly aligned rectilinearly with the trace. Secondly, the width of the idealized response (for unity dielectric constant) is about $2h$ (about 4 mm or 160 mil in the experiments). Numerical simulations (using Macsyma) have shown that use of a larger value of the dielectric constant within the crystal causes the electric field lines within the crystal to become much more spread out. This causes a sharp narrowing of the response response near $z = 0$. Noting that sensor measurements were taken every 18 mil, it is entirely reasonable that the null would not be seen.

2.6 Practice using data acquisition system

As practice for our data acquisition system, we created a simple dipole loop sensor and acquired data with it. The PCB on page 17 of [49, page 17] was used. The PCB was rotated relative to the X-Y motion of the sensor, so we expect to see a "V" of electrical activity (corresponding to the corner of the board). The data was measured with a Tektronics oscilloscope and an FFT was performed in the oscilloscope. This was downloaded by GPIB bus to the computer.

The sampling region was gridded by a 30×30 array. At each physical location the signal was received at 512 frequency values. Since the driving signal was at a single frequency, the major electrical activity was at a single frequency (in bin number 401).

Using the data collected by the loop sensor, we created images showing the magnetic field. In all images, larger numbers are shown lighter and smaller numbers are shown darker. Figure 15 shows the maximum value at each spatial location (autoscaled to the range $[0, 255)$). The Figure on the left is for all 512 frequency bins, the Figure on the right is for frequency bins 390–409.

Figure 17 shows the value in the 401^{st} frequency bin. The Figure on the left is for the data as collected, processed according to the sequence inferred by Figure 16. Note that there appears to be a mis-alignment in the lower right corner of this image; we speculated that this was due to an offset. In the Figure on the right, the first 1024 bytes were removed from the source data set, and then the data was re-processed. In this image the powered edge is more clearly visible. We are uncertain exactly why this improves the image, we believe it to be due to the tension on the cable retarding the positioning table.

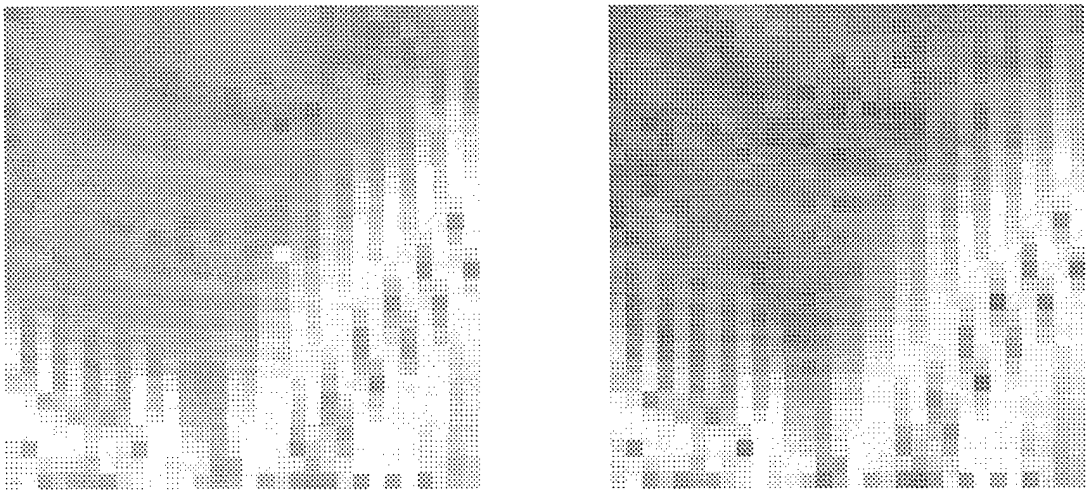


Figure 15: The maximum value in any frequency bin for each of the 30 × 30 physical locations. Left images is for bins 1–512, right image is for bins 390–409.

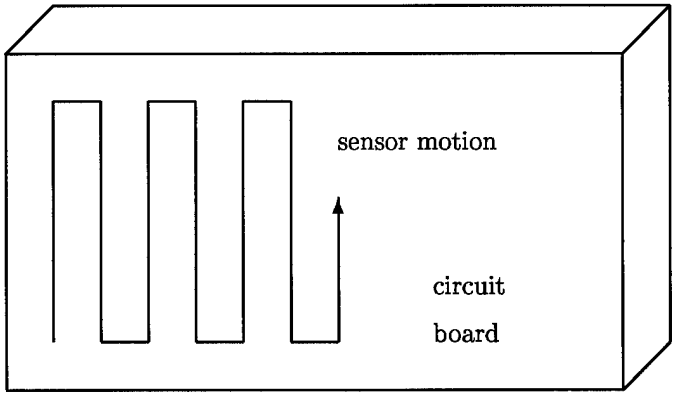


Figure 16: Path of positioning system for data acquisition

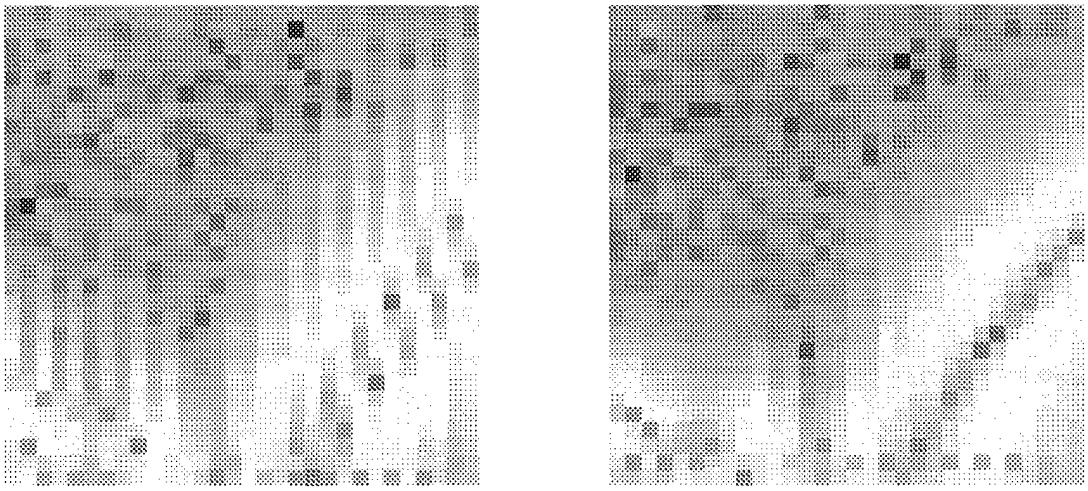


Figure 17: Data in the 401st frequency bin. Left image is original data, right image has had initial 1024 bytes removed.

3 Electrooptic sensors constructed by Srico

Our primary sub-contractor, Srico of Columbus, Ohio, was contracted to build four prototype sensors and sensor packages: two single sensors (which we refer to as 1×1 sensors) demonstrating the success of the design in Figure 7; a 4×1 sensor array demonstrating the success of the design in Figure 9; and a 4×2 sensor array demonstrating that two 4×1 sensor arrays could be laminated together. Unfortunately, the contract Aztec had with Srico called upon Srico to provide their "best effort". This was not nearly good enough.

This section describes the various sensors that Srico delivered, and how they were all inadequate. More specifics may be found in Appendix C.

3.1 Prototype sensors

The section describes the actual sensors obtained from Srico and tested in our system. Three sensors were considered: the flat end sensor (tested), the Ramar sensor (never delivered), and the HDL sensor (tested). Delivery of the sensors was very delayed.

3.1.1 Flat end electrooptical sensor

This sensor belongs to Srico. It has a different geometry than the sensors contracted for by Aztec. This sensor has leads at either end and is held parallel to the circuit board under test; otherwise it is similar to the contracted for sensor. Data was taken with this sensor on a comparable system in June 1996. Some data from this sensor is shown in Figures 18–24.

The sensing region is etched into the top surface of the $LiNbO_3$ waveguide structure. The electrooptic sensor's sensing region is approximately 7 mm long by 150 μm wide, i.e., the normal interaction length is 7 mm and the distance between the two arms of the interferometer is 150 μm. The orientation of the sensor's substrate was parallel to the surface of the printed circuit board, and orthogonal to the printed circuit board conductor (trace) to simulate the smaller section planned for the preliminary "first cut" sensor design. This "first cut" sensor is developed around an "end" type of etching to reduce the surface area and allow assembly of an array of sensors. The effective interaction length for these tests is thus smaller than the 7 mm on the electrooptic sensor, but larger than the 1 mm and 0.3 mm wide traces employed for testing. The effective interaction lengths will be slightly larger than 1 mm and 0.3 mm, respectively, due to fringing field effects over the 3.5 mm separation between the printed circuit board and the sensor.

The sensor was placed in a package for protection. The distance between the active elements of the sensor and the outside of the package is approximately 3 mm. From the bottom (i.e., part adjacent to the circuit board) to the top (i.e., the sensor region), the components are: a glass protective plate approximately 1 mm thick, a $LiNbO_3$ base plate approximately 1 mm thick, and a $LiNbO_3$ sensing waveguide structure (substrate) approximately 1 mm thick.

This sensor was pigtailed with polarization-maintaining fibers at both ends and either fiber could be used for optical input or output.

3.1.2 Flat end electrooptical sensor: Data

The following data on the flat end electrooptical sensor was gathered at Srico.

First series of tests

The first series of tests looked at the lowest measurable voltage that the sensor could measure in its present circuit configuration and setup. Several tests were conducted using both square and sine wave inputs. For purposes of accuracy in calculations, the sine wave signal inputs were used for documentation and reporting requirements.

Figure 18 shows a 21 V peak to peak signal at a frequency of approximately 2 MHz for $d = 3.5$ mm (d represents the separation between the PCB and the front surface of electrooptic sensor). The shot noise limited noise floor at 4 MHz is a few dB above the high frequency value. The low frequency excess intensity

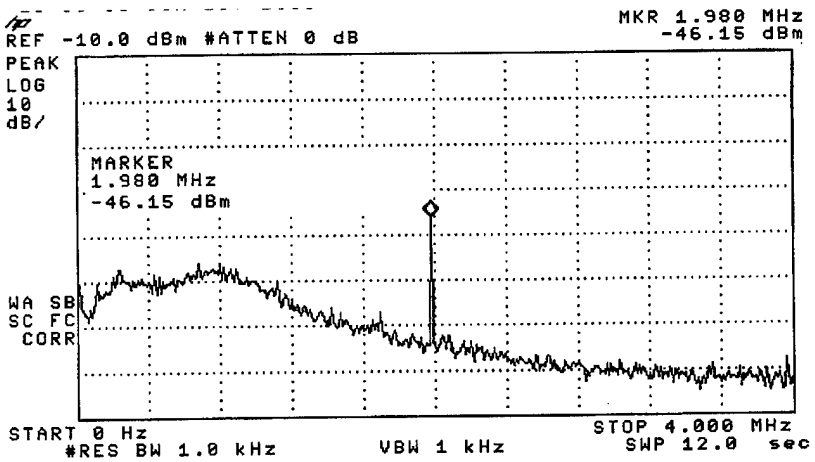


Figure 18: Scan of from DC to 4 MHz of 21 V_{p-p} signal at 2 MHz

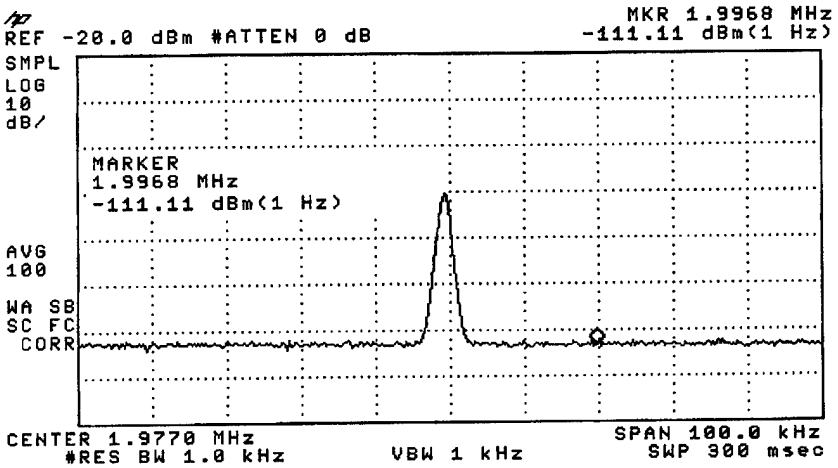


Figure 19: 100 kHz scan of 21 V_{p-p} signal at 2 MHz

noise peak below 1 MHz is clearly seen and is about 25 dB above the noise floor at 4 MHz. This demonstrates the reduction in sensitivity at frequencies below 5 MHz. Since the results reported here were taken at 2 MHz, they will underestimate the electrooptic sensor sensitivity obtainable at frequencies in the noise floor plateau region, i.e., at frequencies in excess of 10 MHz.

Figure 19 shows the signal-to-noise ratio (SNR) obtained for this 21 V peak to peak modulation condition. From the plot we see that the signal amplitude is -50 dBm, while the normalized noise floor measured close to the signal is -111 dBm re 1 Hz bandwidth. Thus, the SNR is $-50 - (-111)$ or 61 dB re 1 Hz. From this result we can determine the SNR for any applied voltage and detection bandwidth. For instance, if the detection bandwidth remain at 1 Hz, we can just detect (SNR = 0 dB) a peak voltage of $10.5 \times 10(-61/20)$ V, i.e., 10 mV. Alternatively, if the detection bandwidth is increased to 1 kHz, we can just detect (SNR = 0 dB) a voltage of $10 \times 10(-1000/20)$ V, i.e., 300 mV.

Figure 20 is a similar plot but for the applied signal reduced to 1 V peak. Again the separation between the PCB and the sensor $d = 3.5$ mm. Here we see that the signal = -77 dBm and the Noise = -114 dBm/Hz. Thus, SNR = 37 dB re 1 Hz. For a just detectable signal (SNR = 0 dB), we can just detect a peak voltage of $1 \times 10(-37/20)$ V, i.e., 10 mV. This is the same result as we obtained for the 21 V peak to peak excitation of the board. If we assume an SNR margin of 10 dB, then the minimum detectable signal

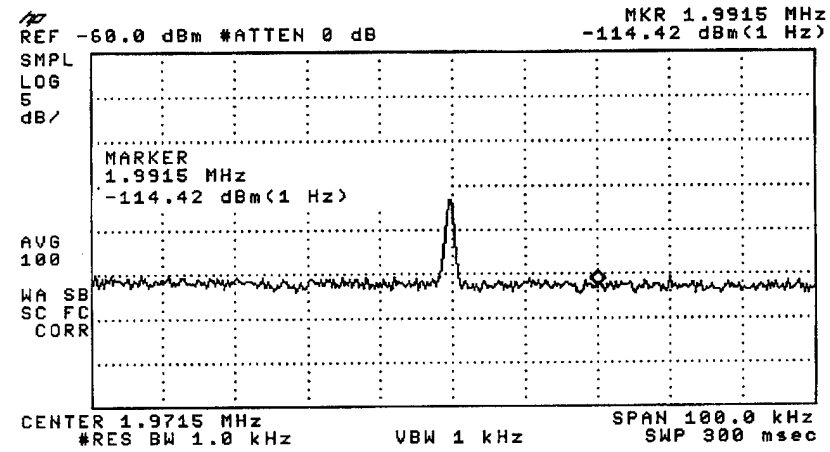


Figure 20: 100 kHz scan of 1 V_{p-p} signal at 2 MHz

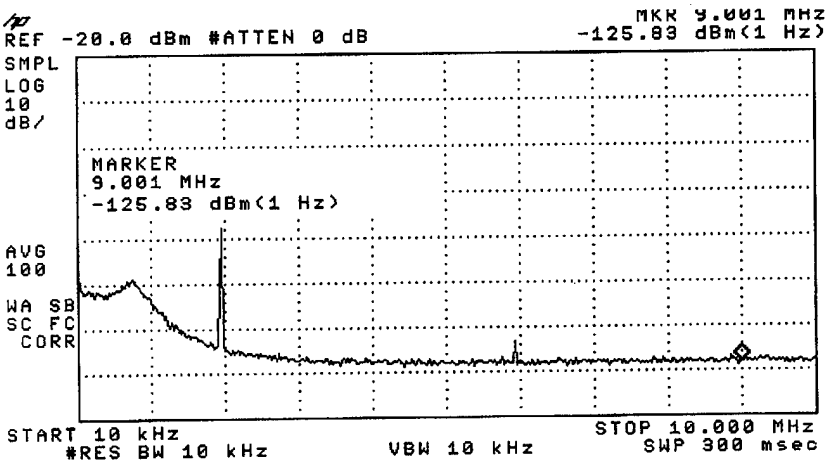


Figure 21: Scan of from DC to 10 MHz of 21 V_{p-p} signal at 2 MHz

in a 1 Hz bandwidth is 30 mV, and 1 V in a 1 kHz bandwidth.

Figure 21 is a plot for a 21 V peak to peak excitation, is similar to Figure 18, except that the sweep width of the spectrum analyzer has been increased to 10 MHz. Here the noise floor at 10 MHz is at -126 dBm/Hz. This is 15 dB below the noise floor at 2 MHz, as was seen in Figure 19. Thus, the sensitivities we have just determined can be increased by 15 dB, if the frequency is 10 MHz and above.

Figure 22 demonstrates the effect of switching the laser on, with a commensurate increase in the noise floor indicating that the optical receiver is shot noise limited. Note that these plots have been averaged 100 times to clean up the traces. The noise floor increases at 9 MHz from -127 dBm to -123 dBm, an increase of 4 dB. An optical receiver is said to be "shot noise limited" when the quantum noise is equal to the thermal kT noise. In this situation, the noise floor increases by 3 dB when the laser is turned on. Our receiver is clearly "shot noise limited," since the noise floor increases by 4 dB.

Second series of tests

The second series of tests looked at sensor drift during periods of changes in sensor throughput. This covered a range of throughputs from approximately 4 dBm to detector saturation, approximately 10 dBm.

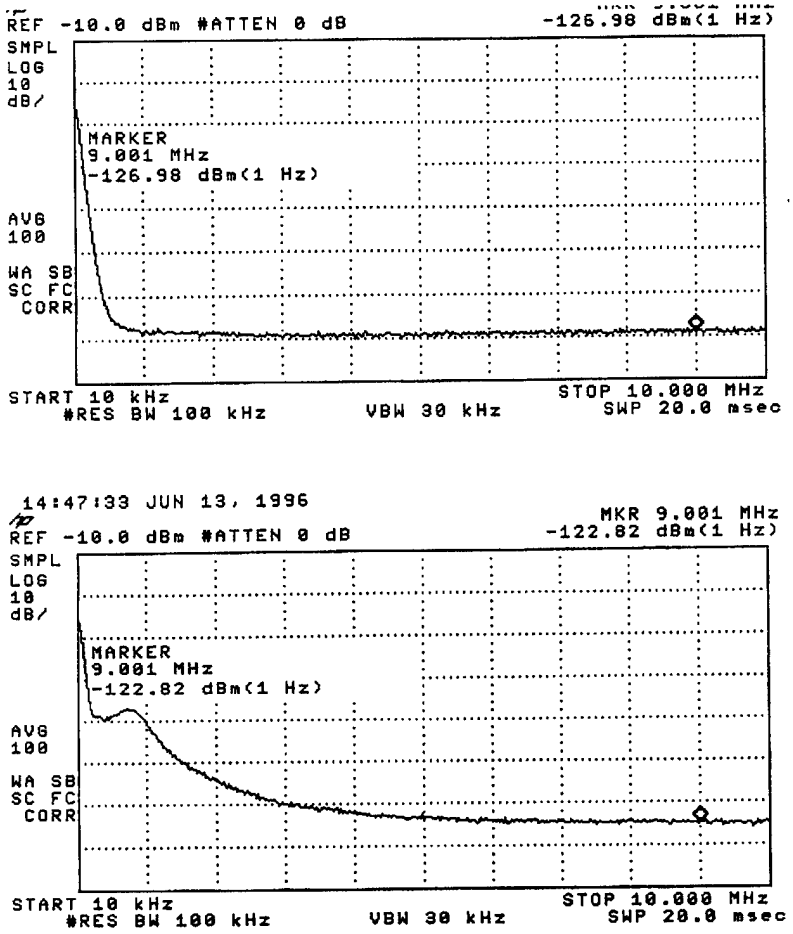


Figure 22: Noise due to laser

The input AC signal was a 5 volt peak-to-peak at approximately 1.97 MHz. These parameters were measured on the signal generator, oscilloscope, and spectrum analyzer. All instrumentation agreed within less than 5%.

As expected, a wider signal trace, 1 mm, provides a larger output signal due to the increased electrooptic area. Figure 23 shows an increased signal-to-noise ratio of approximately 7 dB over that of the thin, 0.3 mm, trace. The received optical power here is somewhat higher than for the previous measurements, and was approximately +8 dBm. The upper response indicates an SNR = 64 dB re 1 Hz, corresponding to a minimum detectable signal of 3 mV/ Hz. The lower trace response indicates an SNR = 57 dB re 1 Hz, corresponding to a minimum detectable signal of 7 mV/ Hz. Note that 2.5 dB is added to the noise floor to take account of the square-law detector and log circuitry in the spectrum analyzer. This signal improvement is less than we would expect from the ratio of 1 mm to 0.3 mm (10.5 dB), but not unexpected considering the lack of spatial resolution and related fringing field effects.

Figure 24 shows plots for the thin, 0.3 mm trace show an increased signal-to-noise ratio of approximately 2 dB when the received power was increased from about +4.3 dBm to +5.6 dBm. The upper response indicates a SNR = 59 dB re 1 Hz, corresponding to a minimum detectable signal of 3 mV/$\sqrt{\text{Hz}}$. The lower trace response indicates a SNR = 57 dB re 1 Hz, corresponding to a minimum detectable signal of 7 mV/$\sqrt{\text{Hz}}$.

Results of the two series of tests

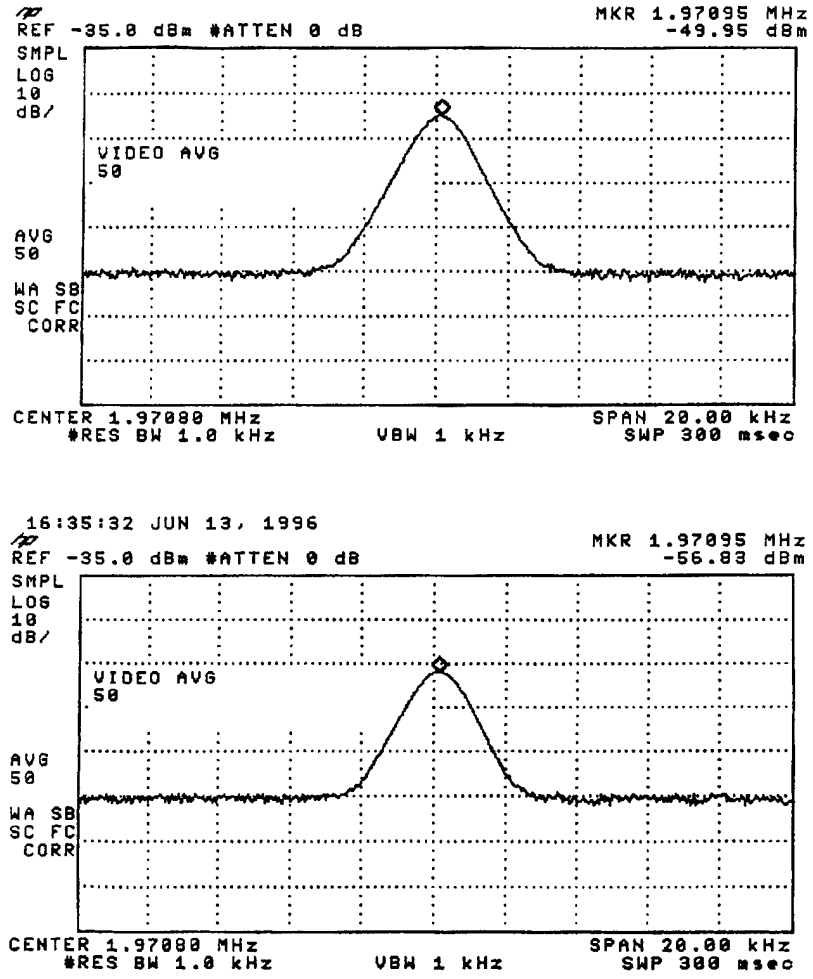


Figure 23: Top graph: wide trace (1 mm). Bottom graph: narrow trace (0.3mm)

At 2 MHz, the measured sensitivity (SNR = 0 dB) was 10 mV/ Hz at a received optical power of 4 mW, for a conductor 1 mm wide, and for a sensor separation of 3.5 mm. At 10 MHz and above, the sensitivity should increase by about 15 dB to 2 mV/$\sqrt{\text{Hz}}$. With an increase in received optical power to 7 mW, the sensitivity should increase to better than 1 mV/$\sqrt{\text{Hz}}$. By reducing the stand-off distance below 3.5 mm, significant further increases in sensitivity should be obtainable. This would probably need to be done in order to increase the spatial resolution.

3.1.3 Ramar sensor

Srico notified us that producing a new sensor would take several months because it was necessary to prepare a new process. They recommended fabricating a substitute sensor from some old unpackaged, integrated chips in their stock. These chips had 10 mm reverse poled regions and were therefore very temperature dependent. Srico had tested this sensor and found it had the expected large temperature dependence. The mutually accepted design for the substitute sensor included electrodes to remove the temperature drift.

Srico evaluated its inventory to identify possible "good" sensor dies for demonstrating a prototype end-on electric field sensor. They completed tests and identified a sensor crystal that could yield several useful

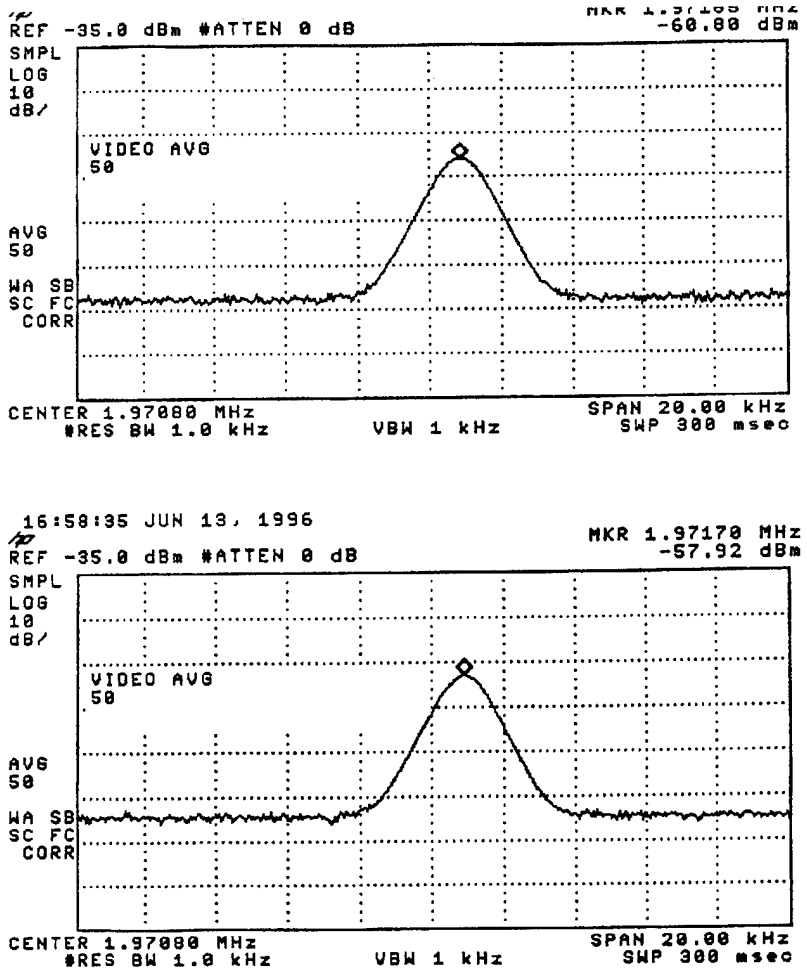


Figure 24: Top graph: low power (received power 4.3 dBm). Bottom graph: high power (received power 5.6 dBm).

devices. A photomask was produced to allow temperature compensation for the sensors made from the old dies.

Then without informing Aztec, Srico changed the design of the sensor. Instead of adding electrodes, Srico added a coating of tin oxide in an attempt to reduce the temperature dependence. The addition of the coating caused a large optical loss for light traversing the sensor. The tin oxide coating was then removed by an etching process. At this point, when the sensor was tested, it no longer reflected light back from the end face of the sensor. It was thought that the end face was damaged by the etch process.

This sensor was never delivered to Aztec.

3.1.4 HDL Sensor

The electrooptical sensors produced by Srico have always exhibited a significant temperature dependence. After Srico fabricated our sensor they attempted to limit this temperature dependence; which resulted in the breakage of our sensor. Since we were insistent on getting a sensor, they modified a pre-existing sensor and delivered it to us. This sensor was not the one fabricated specifically for our project, but the modification of an existing sensor produced for Harry Diamond Laboratories (HDL) some years ago. This sensor allowed us

to test our system, but did not allow accurate measurements to be made. We received this first electrooptical sensor (the HDL sensor) from Srico in November 1996.

The HDL sensor was made by repackaging an old sensor in the Srico inventory, it was markedly inferior to the originally proposed sensor. It has a signal-to-noise ratio approximately 100 times lower than the proposed sensor. This sensor has a reverse poled region of 7 mm which causes a large temperature dependence. The sensor element begins 3.5 mm from the end of the package which decrease the positional accuracy. The fiber leads are keyed for polarization along the fast axis, but the rest of our system is keyed for polarization along the slow axis. Transferring from one axis to the other causes signal loss. The absorption loss in the sensor is 21.8 dB. (It was supposed to be a 6 dB loss.) The laser outputs 22 dBm ($\approx$ 150 mW). The light available at the photoreceiver is only -2.0 dBm. The photoreceiver needs to be receiving at least 0 dBm to be in the desired low noise regime.

The HDL sensor has a large temperature sensitivity. In addition, there is a random drift in the gain. It is supposed that this is due to aging in the epoxy holding the optic fiber leads in the V-grooves attaching the fibers to the chip. The aging results in softening of the epoxy, allowing the fibers to shift position and become misaligned with the waveguides on the chip. This results in the replacement sensor being almost unusable for extended data taking. However, the replacement sensor is sufficiently stable to test some of the functionality of our system.

The HDL sensor had a severe drift in the operating point. During a one hour period, the average output of the photoreceiver varied several times over a range from 60 mV to 600 mV. The average photoreceiver output should have remained at a fixed value if the operating was constant.

3.1.5 HDL sensor: Data

We used the HDL sensor in our system to capture data from the strip line. Measurements were taken of a 10 MHz sine wave of amplitude 5 V_{p-p} going through a 50Ω microstrip. The microstrip was 2 mm wide. The microstrip was 1.5 mm from the end of the sensor package, so the nearest part of the sensor element was 5 mm from the microstrip. A rough estimate for the field at the sensor element is 1000 V/m. The spectrum analyzer was set to a resolution bandwidth of 1 kHz, a sweep of 20 kHz, and the sweep was averaged 100 times. This results in an effective bandwidth of 100 Hz. A clear peak appears with peak height 23 dB above the noise floor. The noise floor was 100 dBm. The minimum detectable electric field for this sensor is then 7 V/m. This is 24 dB less sensitive than specified.

The sensor was moved back and forth across the strip line (a distance of about 50 mm, in steps of 1.1 mm), while not advancing down the strip line. The number of sweeps was 10. Hence, if the sensor system were perfectly repeatable, then there would be 10 repetitions of the same response pattern. At each location, the frequency was swept 20 kHz, centered at 10 MHz (data was collected at 401 different frequencies).

Figure 25 shows the sensor frequency response when the sensor was held stationary, centered on the 2 mm wide microstrip. (This data is taken from the first sweep.) Figure 26 shows the sensor response at a single frequency (10 MHz) when the sensor was moved across the strip line. The plots are logarithmic along the amplitude axis. This causes the peaks to appear broader. The full-width at half-maximum (3 dB) of the peak in is only 5 mm. Note that the sensitive portion of the sensor was approximately 4 mm above the microstrip. Therefore, even at this distance, the positional resolution of the sensor is good.

3.2 Srico's temperature drift problem

Srico's ferroelectric materials are pyroelectric, i.e., their behavior depends on the temperature, and sometimes on the time derivative of the temperature in a complex way. To first order the output power of the sensor has a sinusoidal dependence on the temperature. For the replacement sensor this dependence has a periodicity of about 2π radians per 20 °C. This sensitivity causes the sensor to drift away from the useful operating point which is where the output power is linear in the electric field. The temperature dependence is complicated by an unknown but large additional dependence on the time derivative of the temperature. The size of the drift is proportional to the length of the reverse poled region. Therefore reducing the length of the region will reduce the magnitude of the drift. The drift is caused by the accumulation of charges on the surface of the chip. These charges accumulate as the temperature shifts. The charges have a long relaxation time constant. The charges can be removed (or at least redistributed) so that the operating point returns to the

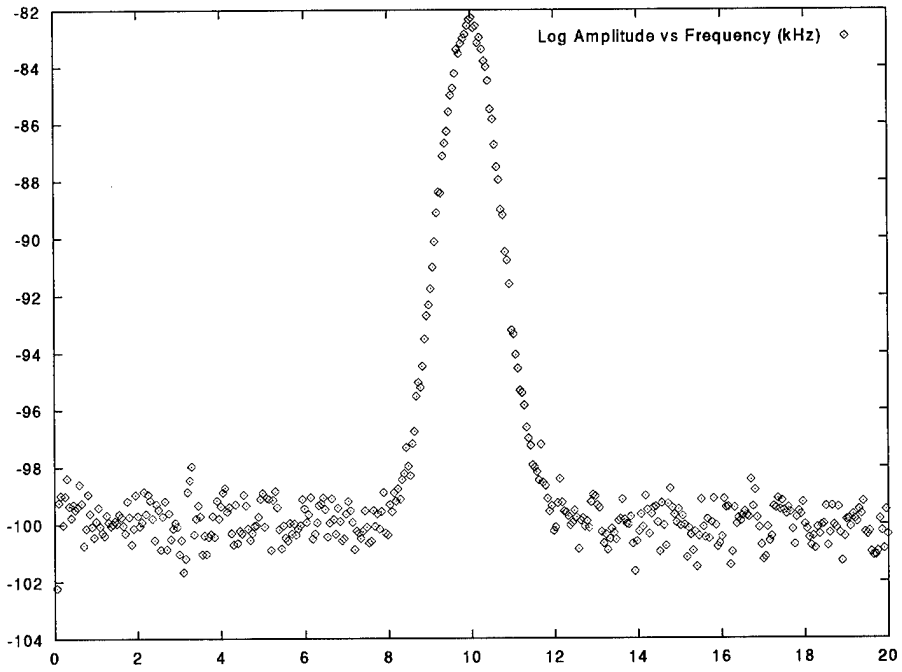


Figure 25: Response of HDL sensor: fixed position, varying frequency

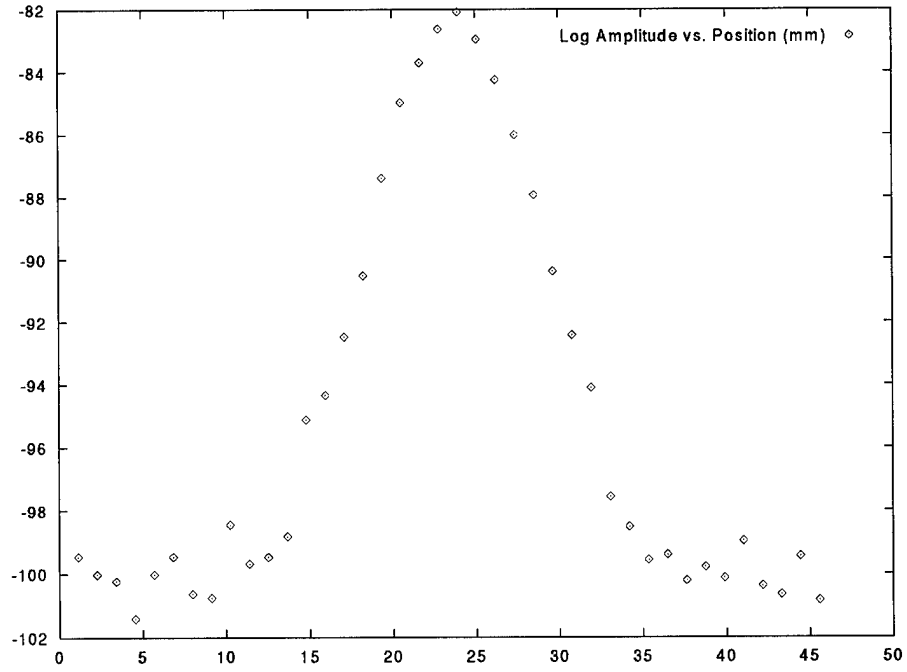


Figure 26: Response of HDL sensor: fixed frequency, varying position

linear regime by the sudden application of heat for several minutes. This shock treatment causes the charges to shift sufficiently so that after twenty minutes it is often possible to be in the linear regime for periods of up to half-hour (in the case of the replacement sensor).

Three methods to reduce the temperature drift were considered. The first method is to use a short reverse poled region (1 mm as opposed to 7 mm). The only portion of the waveguide sensitive to temperature effects is the reverse poled region, where the arms of the interferometer are in a differential mode. Elsewhere phase changes in the waveguides are balanced. The second method is an experimental technique of coating the chip with tin oxide. The idea is that the tin oxide will allow the charges causing the temperature dependence to relax away rapidly. The theory was never properly tested, because the addition of the tin oxide caused the large absorption losses of light in the waveguides of the chip, making the test chip unusable. A third method to eliminate the temperature dependence is to add electrodes around one of the arms of the interferometer and manually bias the interferometer to the useful operating point by putting a voltage across the electrodes. With this active biasing, shifts in the operating point could be eliminated with a feedback system.

3.3 How to make an improved sensor

In an informal collaboration with an electrooptics lab at NIST, we enumerated the following difficulties with the current Srico design:

- minimal path length (therefore sensitivity) near circuit board
- bad crystal orientation to circuit board
- large bending radius of waveguides
- large temperature dependence of output
- photorefractive effect increases light absorption at high power input. (This fact was unknown to Srico; their recommendation to Aztec was to obtain a laser with a very high power output. Yet this can result in a *smaller* signal!)

It appears that the only obvious advantage to the Srico sensor is a factor of two in signal-to-noise due to the reverse-poled region of the interferometer. However, there are several ways to regain this, some of which are mentioned below.

We began to develop a new design that attempted to have a long optical path near the circuit under test to increase the signal-to-noise. Ideally, the electrooptic crystal is oriented parallel to the circuit-under-test and the waveguide follows a zig-zagging path along the surface of the crystal. Additionally, in this orientation, the maximum coupling between the light and the electric field due to the circuit must occur. To fabricate this requires finding a manufacturer who can fabricate waveguides with small bending radius in an electrooptic material of high electrooptic-coefficient to dielectric constant ratio, cut at at the correct crystal orientation. This is complicated by the fact that each electrooptic material has its own set of crystal orientations for maximal electrooptic coupling. Because of time constraints, we were not able to finish the design nor have a prototype fabricated.

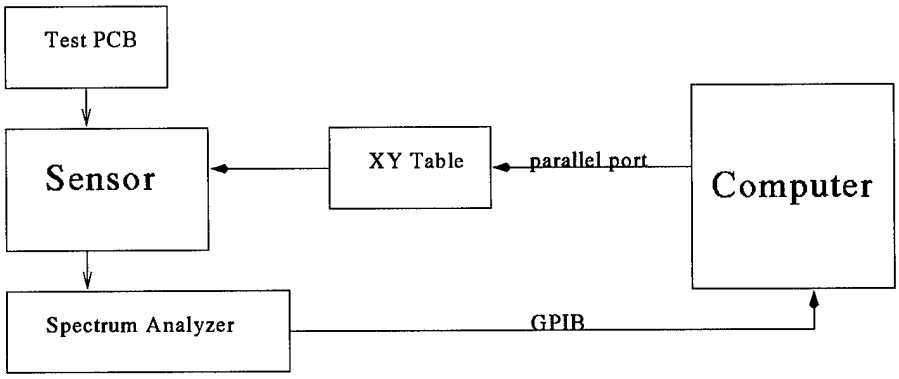


Figure 27: Final overall hardware system design

4 Revised design of sensor system

4.1 Replacement for Srico sensor

In order to demonstrate the overall system, Aztec purchased an electric field/magnetic field sensor kit from Credence Technology for use as an alternative to integrated electrooptic sensors. The sensor kit, called ScanEM, contains a dipole antenna probe for detecting electric fields and a dipole loop probe for detecting magnetic fields; a short review is in Rowe [34, page 46]. Both probes contain internal pre-amplification. Their electric (magnetic) field probe has a frequency response of 2 MHz – 2 GHz (1 MHz – 1 GHz) and a typical sensitivity of -10 dBm/(V/m) (-10 dBm/mT). The probes were attached to Aztec's robot test station. Initial tests show the electric field probe is effective at measuring fields near printed circuit boards in the frequency range from 100 kHz to 1 GHz. The signal intensity is proportional to frequency. Signals near DC are extremely weak.

With this new probe we modified the system design to be as shown in Figure 27.

4.2 Caveat about subsequent analysis

Analysis of data obtained by electric field sensors was delayed until after we obtained a usable sensor. This was because each sensor has different data acquisition capabilities, resolution, and types of errors. It makes little sense to perfect image analysis techniques for a sensor model different from the actual sensor used (especially when many of the relevant parameters need to be determined experimentally).

The conclusions we obtain from use of the Credence Technology sensor that we used are (probably) different from the conclusions we would have obtained if Srico had been able to supply the sensor they contracted to build. As such, all subsequent image analysis in this report is included to indicate the type of analysis that can be done on electric field sensor data from PCBs. However, from our Phase I report [49] we anticipate, prior to performing any analysis, that this sensor will not allow fine resolution of PCB traces.

The following section indicates one of the restrictions of using the Credence Technology sensor.

4.3 Sensor resolution determines sampling rate

For the following physical description:

- infinitely thin trace in a PCB
- with no ground plane
- carrying a charge per unit length of λ
- with the sensor at a constant height of a from the PCB plane
- with the sensor moving transverse to the line on the PCB

the magnitude of the electric field is given by (see section 7.4)

$$E(x) = \frac{\lambda}{\sqrt{a^2 + x^2}}$$

Our plan has been to perform fine spatial sampling above the PCB, and then use the signal values obtained to "recover" the lines on the PCB. A question that naturally arises is: "Are fine spatial samples better than coarse spatial samples?" The answer is yes, if there is information in the fine spatial scale (i.e., if the field has high spatial-frequency components), and no if it does not. The Fourier transform of E is found to be:

$$\begin{aligned} \widehat{E}(\omega) = \mathcal{F}[E(x)] &= \int_{-\infty}^{\infty} E(x) e^{i\omega x}\, dx \\ &= 2\int_{0}^{\infty} E(x)\cos(\omega x)\, dx \qquad \text{(by symmetry, since } E \text{ is even)} \\ &= 2\lambda \int_{0}^{\infty} \frac{\cos(\omega x)}{\sqrt{a^2 + x^2}}\, dx \\ &= 2\lambda K_0(a\omega) \qquad \text{(from [16, formula 3.754.2])} \end{aligned} \tag{5}$$

where ω is the number of samples taken per unit length.

The question now becomes: How rapidly does the Fourier transform $\widehat{E}(\omega)$ decay to zero as $\omega \to \infty$ (i.e., as the number of sample points per unit length increases without bound)? From [4, formulae 9.7.2] we have $K_0(z) \sim \sqrt{\frac{\pi}{2z}} e^{-z}$ as $z \to \infty$. From [4, pages 417, 419] we have the sample numbers

x	1	2	4	10	20
$e^x K_0(x)$	1.1	0.84	0.61	0.39	0.28

If we assume the sample value of $a = 2$ mm then we can construct the following table:

ω (sampling rate)	1/mm	10/mm	20/mm
$\widehat{E}(\omega)/2\lambda = K_0(a\omega)$	1.1×10^{-1}	1.8×10^{-5}	5.8×10^{-10}

Noting that the data acquisition system which uses the Credence Technology sensor has a maximum resolution of 1 part in 8000 (and noting that $1/8000 = 1.25 \times 10^{-4}$), we recognize that we cannot resolve $\widehat{E}(\omega)/2\lambda$ at even 10 samples per mm; i.e., we cannot measure the high spatial frequency of the electric field.

We conclude: for the given distance above the PCB, and the given resolution of the data acquisition system, there is no reason to sample any finer than 10 samples per mm.

5 Data acquisition system

5.1 Introduction

The **Electric Field Imaging System** (EFIS) measures the electric field above a circuit board and produces an image of the electric field.

The electric field imaging system consists of an electric field sensor, a positioning table, a stepper-motor driver, a spectrum analyzer, a computer, and a software package. The software package directly controls the spectrum analyzer, the stepper-motor driver, and, through the stepper-motor driver, the positioning table. The spectrum analyzer is connected to the computer via a GPIB (General Purpose Interface Bus) and the stepper-motor driver is connected to the computer via a parallel printer port.

The software package, written in LabView[4], user interface consists of a main panel and several subpanels. The main panel has controls for motion and data acquisition. The subpanels set the parameters of the positioning table, spectrum analyzer, and data files. One subpanel displays the data from the spectrum analyzer.

The program lets the user move the positioning table to any point, acquire data at that point, display the data, and automatically acquire data from all points in a two dimensional grid.

5.2 Re-use with other sensor technologies

EFIS was designed to be a general tool for acquiring radiated electromagnetic field data from a PCB. As described in this section, all the experimental parameters

- Turning equipment comonents on and off
- Defining the number of measurements
- Defining the header associated with each data file
- Defining the physical region of the PCB to obtain data from
- Defining the frequency range to sample from
- Defining the name of the data files

can be controlled from the softeware interface.

Any other type of electromagnetic field sensor can be inserted into the system and used for data acquistion.

5.3 Theory of operation

5.3.1 Electric field measurements

Electronic devices, for example, printed circuit boards, radiate electric and magnetic fields. The electric fields are produced by the regions of high and low charge and are intimately connected to the voltages on the board. Close to a circuit board, the pattern of the electric field mimics the pattern of the voltages on the board. Unlike voltages, which are usually require contact measurements, electric fields are easily measured at a distance, for example, using antennas. There is a large electric field in regions of rapidly varying voltage and the electric field is larger near an active trace on a circuit board than far from an active trace.

The amplitude of the electric field due to a line trace decreases as the inverse distance from the trace. Because of this, the minimum resolving distance of an electric field based sensor for two neighboring traces is proportional to the height of the sensor above the traces. A rough rule of thumb is that the resolving distance is the distance of the sensor above the board. The resolving distance is improved through the use image reconstruction techniques.

Because electric field field measurements are non-contact, they are more sensitive to noise and crosstalk than voltage measurements.

The electric field is a vector quantity, which means that its value at a given location is directionally dependent. In practical terms, a sensor measuring a single component of the electric field near a trace produces

[4]LabView is distributed by National Instruments Corporation; 6504 Bridge Point Parkway; Austin, TX 78730; 512-794-0100; `http://www.natinst.com/products/`.

different readings depending on the orientation of the sensor to the trace. To overcome the complications due to this directional dependency, this system uses an isotropic sensor, which is sensitive only to the magnitude of the electric field. This means the sensor reading is independent of orientation of the sensor to the trace.

5.3.2 Images

Electric field images are produced by moving the sensor, at a constant height above the board, through a series of points and acquiring electric field data at each point. The data at each point consists of a spectrum of 401 frequency bins (the number of bins is determined by the spectrum analyzer). The data from all the points and a given frequency bin form an image slice or, more simply, an image. Each point in this image is a pixel. After acquisition, the data is stored for later image processing.

5.3.3 Stepper-motor driver and positioning table

The sensor is moved through the grid by a positioning table. The positioning table moves the sensor in the most efficient path over the board, which is a zig-zag. The positioning table is moved in precise steps by two stepper-motors. The positioning table has separate calibration points for each of its two directions of motion. The positioning table travel used in this system is 9" by 9". The minimal step of this system is 4.5 mil. However, the hysteresis is considerably reduced if a minimum of 9 mil steps are used.

The stepper-motors are activated by a stepper-motor driver, which supplies the current to drive the stepper-motors. The stepper-motor driver is connected to the controlling computer by a cable to the parallel printer port. The computer sends digital signals down the cable to activate the stepper-motor driver. The stepper-motor driver sends signals up to the computer only when the positioning table reaches a calibration point.

5.3.4 Spectrum analyzer

The output of the sensor is fed into a spectrum analyzer. The spectrum analyzer measures the signal by sweeping through a range of frequencies and recording the strength of the signal at that frequency. Alternatively, it can measure repeatedly at a single frequency. The sweep is digitized and transferred to the computer for storage over a GPIB bus. Via the GPIB bus, the computer can set the parameters of the spectrum analyzer, trigger data sweeps and data transfers.

The frequency range of the spectrum analyzer used in this system is 9 kHz to 1.8 GHz. The sweep span is either 0 Hz or in the range of 100 Hz to 1.8 GHz. The resolution bandwidth is in the range 30 Hz to 3 MHz. The range of the reference amplitude is -140 dBm to 30 dBm.

5.4 Software operation

5.4.1 Controls

The main panel appears when the program is started. This panel has four controls buttons. The **Initialize** button opens communication to and configures the stepper-motor driver and the spectrum analyzer. The **Move** button moves the positioning table to a desired located. This location is shown by the **Target** indicator located next to the **Move** button. The target destination can be set by the user. The **Target** indicator displays the target location either in coordinates relative to the current location or in coordinates from a fixed point on the positioning table. Above the **Target** indicator is the **Position** indicator. This indicator displays the current location of the positioning table, in coordinates either from the home, i.e., calibration position of the table, or from the origin of the allowed region of motion. The last control button on the main panel is the **Run Grid** button. This is the control for automated acquisition of electric field images. This button moves the positioning table through a two-dimensional grid, acquiring data at each location. The main panel also has indicators displaying the name of the data file for the acquired data, and comments to be placed in the header of the data file.

5.4.2 Data files

The data files consist of two parts, the header and the body. The header contains information about the data acquisition conditions. The body contains the data.

The header is variable length. It is in ASCII format. It ends with the first blank line. The body begins after the first blank line. The format of the standard header is:

<header style> <body style> <CR>
<data file name> <CR>
<date> <time> <CR>
<num. freq. bins> <num. columns> <num. rows> [pixel length] [pixel width] <CR>
<center freq.> <span freq.> <resolution bandwidth> <reference amplitude> <CR>
[comments] <CR>
<CR>

where **<text>** is required and **[text]** is optional.

The body of the data file can be in one of several formats. These include the choice of binary or ASCII. Also, there is the choice of saving data from all frequency bins or just a single value from either one frequency bin or a statistical measure of the bins. The spectrum analyzer provides 401 bins of data, leading to large files (several megabytes) when scanning 100x100 pixel or larger images. The options for a single value are data from a single bin, data from the maximum valued bin, and data from the average of the bins. If data from the maximum valued bin is selected, both the bin number and the value are saved.

If the data is stored in binary format, the data is stored continuously, without separators, cycling most quickly through frequency bins, then columns, then rows. The data is saved as four byte reals. If the data is stored in ASCII format the data is ordered the same method as for binary, but the data is stored with spaces between the frequency bins, and carriage returns between the columns and between the rows.

The name of the data file is set using the menu item **File>>Select Name Of Next File**. The default name upon program initialization is `ftYYMMDD01.dat`, where `YY` is the year, `MM` is the month, `DD` is the day. If the **File>>Automatically Chooses Names For New Data File** option is used, the last two digits of the file name prefix is incremented, from `01` to a maximum of `99`.

5.4.3 Settings files

The settings file contains all the parameters needed by the system. A custom setting file is created by using the menu command **File>>Save Settings File** to save the the current settings. The file is reloaded with the menu command **File>>Load Settings File**.

The settings file is an ASCII file. The format of the settings file is repeated lines of:

<string>: <value> <CR>

where **<string>** is an explanation of the parameter and **<value>** is the value of the parameter, which may be integer, floating or character string. The first line lists the settings file type. Settings file names have the suffix `set`.

5.4.4 Positioning table regions

The positioning table is divided into several regions. Two coordinate systems (**absolute position coordinates** and **allowed motion position coordinates**)are used to describe locations on the table.

- The **total table region** consists of the entire table and includes the entire possible range of travel of the table. The calibration point for the table is in the corner of the table with the two switches; when the table is in this position, both green lights on the stepper motor driver are lit. The **absolute position coordinates** are defined with their origin at the the calibration point.

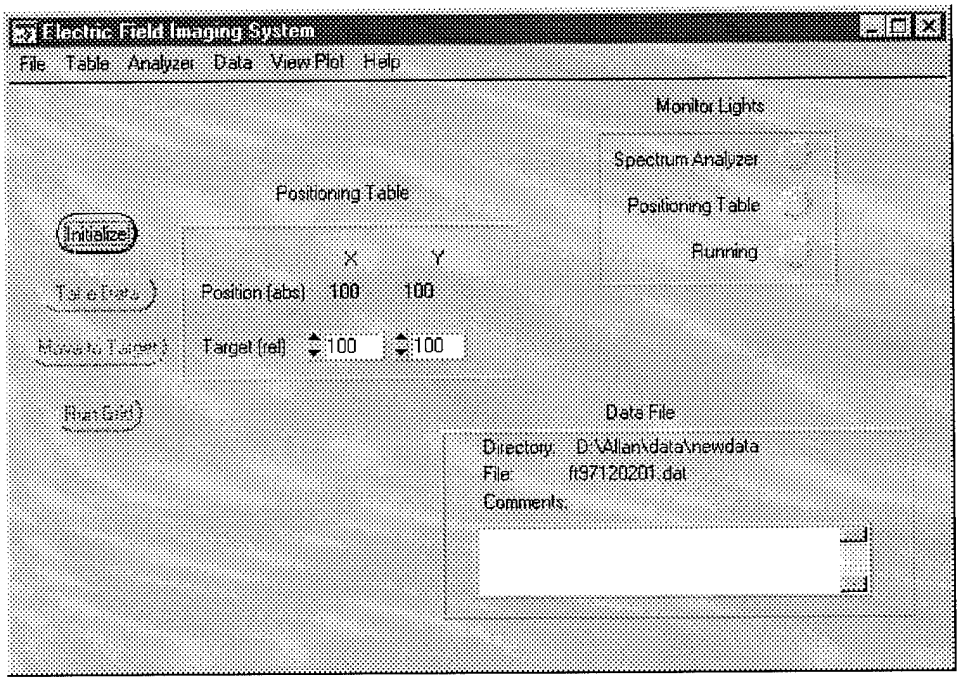


Figure 28: Data acquisition: the main panel.

- The **region of allowed motion** is a rectangular subset of the total table region. Motion during **Move** and **Run Grid** is limited by software to this region. However, during **Initialization** and **Calibrate Table**, the positioning table goes outside of the region of allowed motion in order to reach the calibration point. The motion outside of the allowed region is first to the first row, then to the first column (the calibration point) and then back out to the first column of the allowed region, then the first row of the allowed region. This provides protection for delicate electronics on the table and prevents the table from colliding with packaging. The **allowed motion position coordinates** are defined with their origin at the corner of the region of allowed motion closest to the calibration point.
- The last defined region is the **grid region**. This region is a rectangular subset of the region of allowed motion. It is the region traversed by the table during **Run Grid**.

5.5 User interface

The major controls appear on the main panel and the menu to the main panel. The following controls and indicators appear on the main panel:

5.5.1 Main panel

The main panel (see Figure 28) has the **Initialize**, **Data**, **Move**, and **Run Grid** buttons. The **Move** button moves the positioning table, the **Data** button acquires data from the spectrum analyzer, and the **Run Grid** button acquires data from a two-dimensional grid of locations of the positioning table. Indicators show the positioning table position and the target motion of the table. In the lower right hand corner is a display for the data file and comments for the header of the data file.

Initialize This control button initializes the positioning table and the spectrum analyzer. These can be individually initialized from the menus **Table>>Calibrate Table** and **Analyzer>>Turn Analyzer On**.

Position This numeric indicator shows the current location of the positioning table. The units are units of stepper-motor motion, approximately 4.5 mil. The coordinates are either from the origin of the positioning table or the origin of the allowed region of motion. The coordinate choice is set by the menu control **Table>>Set Allowed Region Position Coordinate**.

Target These numeric controls set the target distance for moving the positioning table. The units are units of stepper-motor motion. The coordinates are either for relative motion coordinates from the current location of the positioning table or fixed coordinates which in turn are either from the origin of the positioning table or the origin of the allowed region of motion. The coordinate choice is set by the menu control **Table>>Set Allowed Region Position Coordinate** and **Table>>Set Fixed Move Target Coordinates**.

Directory This text indicator shows the current drive and directory of data files.

File This text indicator shows the name of the current data file.

Comments The contents of this text window is copied into the header of the data file.

Move to Target This control button moves the positioning table by the amount set in the target field. The button is de-activated until the stepper-motor is turned on. Pressing this button first moves the positioning table along a row to a new column and then moves it along a column to a new row. The positioning table only moves if the table has been calibrated. Calibration occurs after pressing the Initialization button or the menu control **Table>>Calibrate Table**.

Take Data This control button collects data from a single location. If the menu item **Table>>Run Repeatedly** has been selected, this button acquires data repeatedly at that location. Pressing the button writes a header to the data file, puts the spectrum analyzer in single sweep mode, collects data, returns spectrum analyzer to continuous sweep mode, transfers the data to the computer, and writes the data to the data file. The spectrum analyzer is off, it returns an error.

Run Grid This control button moves the positioning table through a grid and acquires data at each node in the grid. This button moves the positioning table to the start of the target region, then moves the table through all the points in the grid, acquiring data at each point. The table moves forward along each odd row and back along each even row. It moves steadily from the low rows to the high rows. After moving through the grid of points, the positioning table moves to the origin of the allowed region of motion. If the menu item **Table>>Run Repeatedly** has been selected, the positioning table runs through the grid repeatedly, and re-calibrates before each run through. Before starting the grid, the function tests that the positioning table is calibrated, the stepper-motors driver is on, and the spectrum analyzer is open for communication with the computer. Then the function opens a data file and writes a header. The spectrum analyzer is put in single sweep mode. The positioning table is moved through a grid and data is collected at each point in the grid, transferred to the computer, and written to the data file. At the end of the grid, the spectrum analyzer is returned to continuous sweep mode.

5.5.2 Main panel menu

File

The file menu (see Figure 29) has several controls relating to file management.

Select Name of Next Data File This menu item selects the name of the next data file and updates the display with this information.

Automatically Choose Names for New Data Files / Manually Choose Names for New Data Files This menu item sets the flag for automatic versus manual choice for names of new data files. If true, this flag automatically selects a different name for the new data if the current name already exists. If false, the user is asked about overwriting existing files.

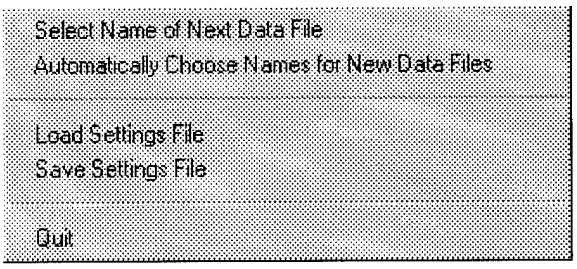


Figure 29: Data acquisition: the file menu.

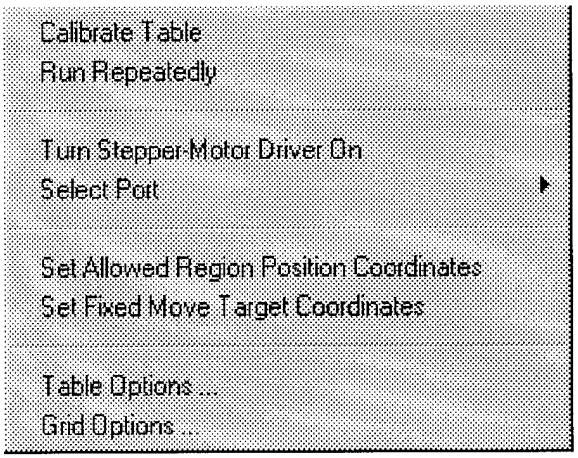


Figure 30: Data acquisition: the table menu.

Load Settings File This menu item reads in a settings file and adds these settings to the program.

Save Settings File This menu item saves the current settings to a settings file.

Quit This menu item ends the program.

Table

The table menu (see Figure 30) has several controls relating to the operation of the positioning table.

Calibrate Table This menu item calibrates the positioning table. The stepper-motor driver must be activated before reseting the positioning table. The stepper-motor is activated either by the **Initialize** control button or the menu item **Table>>Turn Stepper-Motor Driver On.**

Run Repeatedly / Run Once This menu item sets the control buttons **Data** and **Run Grid** to either a one-shot or repetitive mode.

Turn Stepper-Motor Driver On / Turn Stepper-Motor Driver Off This menu item controls the status of the MD2 stepper-motor driver. If the driver flag is off, this routine initializes the parallel port, turns the MD2 stepper-motor driver on, activates the **Move** control button, and if the spectrum analyzer flag is on activates the **Run Grid** control button. If the driver flag is on, routine resets the parallel port, turns the MD2 stepper-motor driver off, de-activates the **Move** control button, and de-activates the **Run Grid** control button.

Select Port This menu item selects the parallel printer port used to communicate with the MD2 stepper-motor driver. The parallel printer port addresses are `0x3BC`, `0x378`, and `0x278`. The stepper-motor drivers must be off to change the port setting.

Set Allowed Region Position Coordinates / Set Absolute Position Coordinates This menu item sets the flag for the mode of the position coordinates. The choices are absolute position (determined from the home of the positioning table) or allowed region position (determined from the origin of the region of allowed motion). This affects the **Position** indicator on the main panel and the **Grid Origin** indicator on the grid parameters panel. If the move target flag is set to fixed position, it also affects the **Target** indicator on the main panel.

Set Fixed Move Target Coordinates/ Set Relative Move Target Coordinates This menu item sets the flag for the mode of the coordinates in the **Target** indicator on the main panel. The choices are relative coordinates or fixed coordinates. If the flag is set for absolute coordinates, the display will be either relative to the home of the positioning table or the the origin of the allowed region of motion, depending on the setting of the position coordinate flag.

Table Options This menu item opens the table subpanel.

Grid Options This menu item opens the grid subpanel.

Analyzer

Turn Analyzer On / Turn Analyzer Off This menu item opens and closes the spectrum analyzer for communication with the computer and resets the spectrum analyzer state.

Analyzer Options This menu item opens the spectrum analyzer subpanel.

Data

Save Data in ASCII Format If checked, this menu item sets the flag to save data files in ASCII format.

Save Data in Binary Format If checked, this menu item sets the flag to save data files in binary format.

Other Format Options This menu item opens the file format subpanel.

View plot

This menu item sets a flag which, when true, updates the plot display each time data is scanned. Updating the plot slows data taking during **Run Grid**.

Help

This menu item displays a help panel.

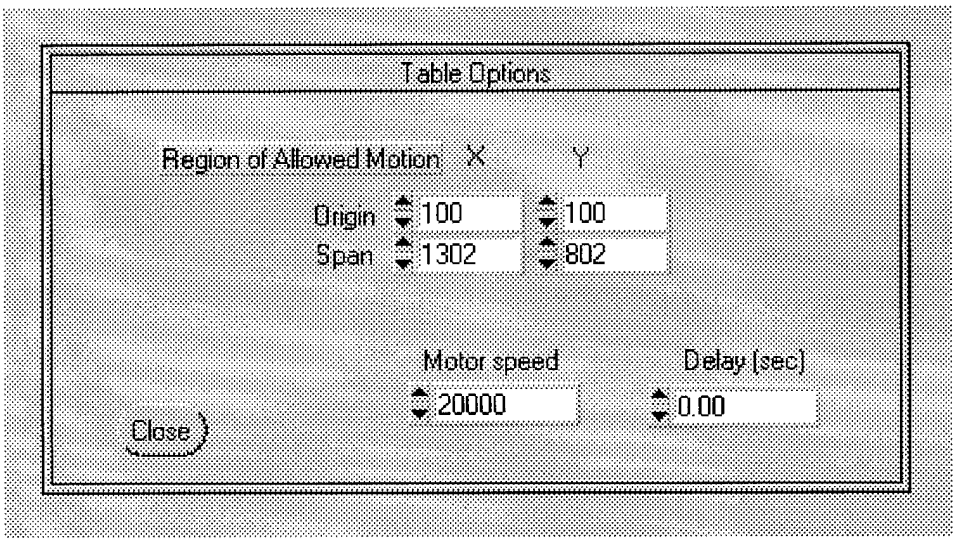


Figure 31: Data acquisition: the table panel.

5.5.3 Table panel

This panel (see Figure 31) has indicators which show the region of allowed motion of the positioning table and indicators controlling the speed of motion of the positioning table.

Region of Allowed Motion These numeric controls set the values for the region of allowed motion of the positioning table. The maximum size of this region is the size of the positioning table.

Motor Speed This numeric control sets the time delay between steps of the stepper-motor, which is proportional to the speed of the stepper-motor. 0 is the fastest speed and 32766 is the slowest speed. The actual speed is computer dependent.

Delay This numeric control sets the length of the time delay before and after moving the positioning table. This is used to reduce noise during data acquisition.

Close This control button closes the Table panel.

5.5.4 Grid options panel

This panel (see Figure 32) controls the size and resolution of the grid used for data acquisition with the control button **Run Grid**.

Grid Parameters These numeric controls read new grid parameters. They also check if the new parameter is within allowed limits. If the parameter is not, the the display returns to its previous setting.

Number of Pixels The number of pixels in the grid along each direction.

Pixel Size The length of the pixels in stepper-motor units.

Grid Span The length of the grid in stepper-motor units.

Grid Origin The location of the origin of the grid in stepper-motor units. These coordinates are either relative to the home of the positioning table or the the origin of the allowed region of motion, depending on the setting of the position coordinate flag.

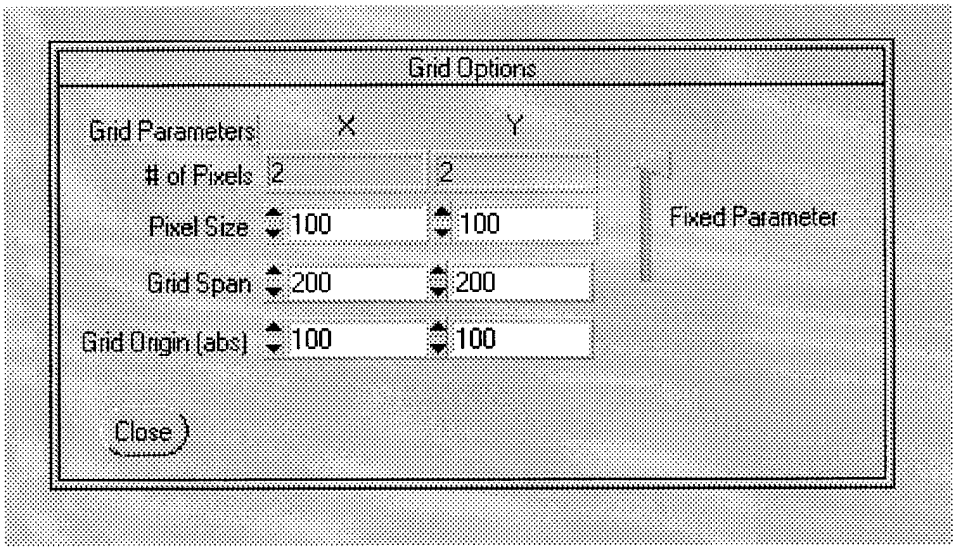


Figure 32: Data acquisition: the grid panel.

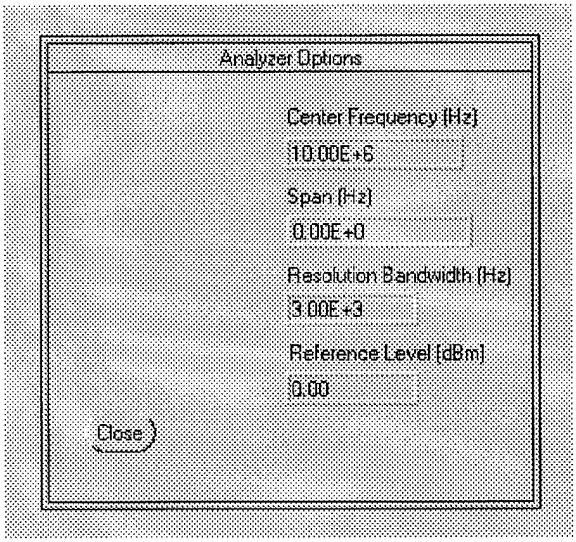


Figure 33: Data acquisition: the spectrum analyzer panel.

Fixed Parameter In order to calculate a self-consistent set of grid parameters, one of the variables (number of pixels, pixel size, and grid span) must be held constant. This control slide selects the grid parameter to hold constant. First this function turns the display indicator for the previous constant grid parameter to the active setting, then the function reads in the new constant parameter, and sets the display indicator for that parameter to in-active.

Close This control button closes the Grid panel.

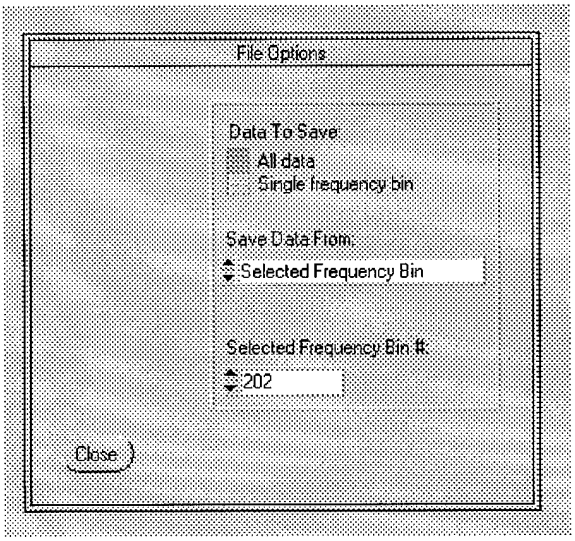


Figure 34: Data acquisition: the data panel.

5.5.5 Analyzer panel

The numeric controls on this panel (see Figure 33) set the parameters of the spectrum analyzer from the display. It is activated when the spectrum analyzer is turned on by the **Initialize** control button or the menu item **Analyzer>>Turn Analyzer On.**

Center Frequency This numeric control sets the center frequency of the spectrum analyzer. The range 9 kHz to 1.8 GHz.

Span Frequency This numeric control sets the sweep span frequency of the spectrum analyzer. The range is either 0 Hz or 100 Hz to 1.8 GHz.

Resolution Bandwidth This numeric control sets the resolution bandwidth of the spectrum analyzer. The range is 30 Hz to 3 MHz.

Amplitude Reference Level This numeric control sets the amplitude reference level of the spectrum analyzer. The range is -140 dBm to 30 dBm.

Close This control button closes the Analyzer panel.

5.5.6 Data panel

This panel (see Figure 34) controls the data which is saved in the data file.

Data To Save This binary switch selects whether to save data from all frequency bins or to save data from just one frequency bin.

Save Data From This ring selects whether to save data from a selected frequency bin, from the frequency bin with maximum amplitude, or from the average of the frequency bins.

Selected Frequency Bin This numeric control selects the frequency bin to save data from when the **Save Data From** control is set to selected frequency bin.

Close This control button closes the data panel.

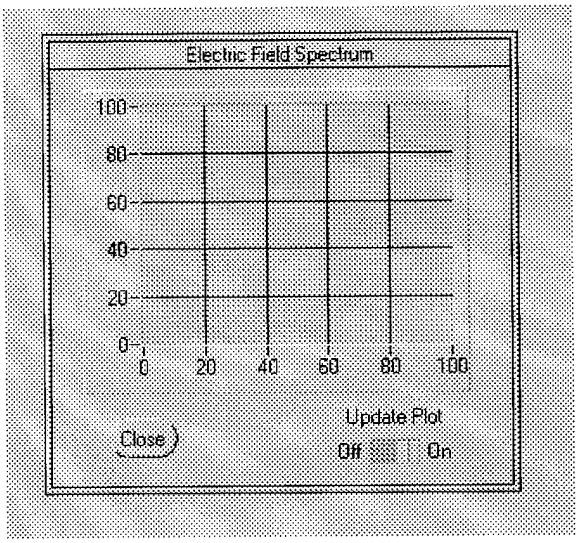


Figure 35: Data acquisition: the plot panel.

5.5.7 Plot panel

This panel (see Figure 35) displays a plot of the measured spectrum.

Update Plot This binary switch sets a flag which, when true, updates the plot display each time data is scanned. Updating the plot slows data taking during **Run Grid**.

Close This control button closes the plot panel.

5.5.8 Help panel

This panel lists on-line help for the user interface.

Close This control button closes the help panel.

5.6 List of functions

A list of functions follows.

5.6.1 Initialization functions

main This function loads the user interface panels and menu, sets the default values of the parameters, displays the user interface, and begins an interactive session.

defaultParameters This function initializes the program parameters.

uirInitialize This function initializes the positioning table and the spectrum analyzer. It is called from the main panel control button **Initialize**.

setUI This function copies the values of parameters to the user interface.

panelResponseInt This function tests changes in the value of integer parameters displayed on the user interface. If the value is within range, it copies the new value to an internal variable. Otherwise it returns the display to its previous setting.

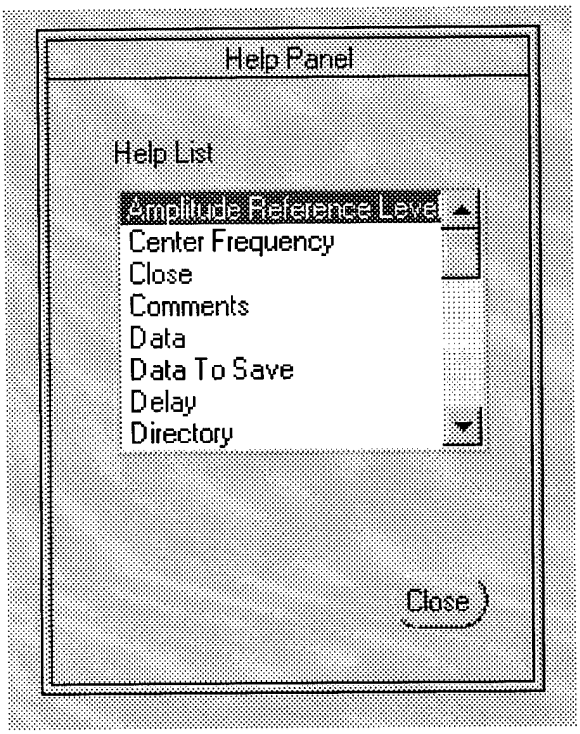


Figure 36: Data acquisition: the help panel.

panelResponseDouble This function tests changes in the value of double parameters displayed on the user interface. If the value is within range, it copies the new value to an internal variable. Otherwise it returns the display to its previous setting.

uimMenuOptions This user interface function opens subpanels.

uirUpdatePlot This function, called by the binary switch **Update Plot** on the plot panel, sets the flag `updatePlot` which, when true, updates the plot display each time data is scanned. Updating the plot slows data taking during **Run Grid**.

uirClosePanel This function closes a user interface panel.

5.6.2 File related functions

openNewFile This function opens a data file and writes the header. As a preliminary, this function tests if the spectrum analyzer is connected, reads settings information from the spectrum analyzer, moves the file pointer to the data file directory, and tests if the file already exits. The function then checks if file already exists. It it does, the function responds by notifying the user or incrementing file number, depending on the choose file name mode. If in choose file name mode, it checks that the file name matches the file name pattern, which is ``*##.*''. Next the function opens the file and writes the header. Then the function closes the file. The data file format is described in Section 5.4.2.

checkStandardFilePattern This function checks whether file name matches standard pattern of ``*##.*'' where * is any number of any characters and # is a digit The function returns 0 if the pattern matches, -1 otherwise. If true the function increments the ## in the filename (if 0<=##<99).

saveDataToFile This function saves data to a file. The format of the data in the data file is determined by the setting of the flags `saveAll`, `selectedData`, and `asciiData`.

dataPanelUpdate This function sets the visibility of the file parameters controls on the data panel of the user interface.

uirFiles This function checks the user interface for changes in the flags `saveAll`, `selectedData`, and `binNum` used for data file formats

uimSaveDataFile This function select the name of the next data file and update display with this information.

uimIncrFiles This function selects the Automatic/manual choice for names of new data files flag `chooseDataFileName` If true, this flag automatically selects a different name for the new data if the current name already exists. If false, the user is asked about overwriting existing files.

uimLoadDefaultFile This function reads in a settings file.

uimSaveDefaultFile This function writes the current settings to a settings file.

uimAsciiData This function sets the flag `asciiData` used to store data in data file as either ASCII or binary.

5.6.3 Positioning table functions

cviHome This function moves the positioning table to the home position and updates the display.

cviMove This function moves the positioning table and updates the display. It optionally inserts a delay before and after moving the positioning table. This delay is set by the **Delay** control on the table panel.

resetTable This function moves the positioning table to the home position for calibration and then moves the table to the origin of the allowed region of motion. If the stepper-motor driver is off, it activates the driver and the **Move** control button. If the spectrum analyzer flag is on, it activates the **Run Grid** control button. The motion of the table is first to the first row, then the first column (home position) and then back out to the first column of the allowed region, then the first row of the allowed region. The motion of the table is halted if an error is returned from the **md2Home** or **md2Move** subroutines.

uirAllowedRegion This function reads in the values for the region of allowed motion of the positioning table. The maximum size of this region is the size of the positioning table.

uirSpeed This function sets the time delay between steps of the stepper-motor, which is proportional to the speed of the stepper-motor. 0 is the fastest speed and 32766 is the slowest speed. The actual speed is computer dependent.

uirDelay This function sets the length of the time delay before and after moving the positioning table. This is used to reduce noise during data acquisition.

uimTablePosition This routine sets the flag `absPosition` used for the mode of the position coordinates. The choices are absolute position (determined from the home of the positioning table) or allowed region position (determined from the origin of the region of allowed motion). This affects the `Position` indicator on the main panel, the `Grid Origin` indicator on the grid panel, and if the move target flag `relTarget` is set to fixed position, the `Target` indicator on the main panel.

uimTableTarget This function sets the flag `relTarget` for the mode of the **Target** indicator. The choices are relative coordinates or fixed coordinates. This affects the `Target` display on the main panel. If the flag is set for absolute coordinates, the display will be either relative to the home of the positioning table or the the origin of the allowed region of motion, depending on the setting of the position coordinate flag `absPosition`.

uimResetTable This function is called by the menu item **Table>>Calibrate Table**. It is used to calibrate the positioning table. The stepper-motor driver must be activated before resetting the positioning table.

uimMotorsOn This routine controls the status of the MD2 stepper-motor driver. If the driver flag `motorsOn` is off, this routine initializes the parallel port, turns the MD2 stepper-motor driver on, activates the `Move` control button, and if the spectrum analyzer flag `analyzerOn` is on activates the **Run Grid** control button. If the driver flag is on, this function resets the parallel port, turns the MD2 stepper-motor driver off, de-activates the **Move** control button, and de-activates the **Run Grid** control button.

uimPort This function selects the parallel printer port used to communicate with the MD2 stepper-motor driver. The parallel printer port addresses are `0x3BC`, `0x378`, and `0x278`. These are declared in the array **port**. The variable **portInt** is used as an index to the array **port**. The stepper-motor drivers must be off (flag `motorsOn`) to change the port setting.

inAbsRegion This function tests if the new position is within the limits of the positioning table.

inArmRegion This function tests if the new position is within the limits of the allowed region of motion.

md2Setup This function sets the internal stepper-motor driver parameters.

md2On This function initializes the parallel printer port and turns on the stepper-motor driver. It is used after the **md2Setup** function.

md2Off This function returns the parallel printer port to its previous state, ready for use with a printer, and disables the stepper-motor driver.

step This function moves the positioning table one step. It returns 0 on normal termination; 'K' on key-press termination. The function quits moving if a key is pressed.

md2Home This function calibrates the positioning table by moving the table to a known home position. All other moves are relative to this home position. The positioning table is moved in reverse until a calibration switch is activated, then forward until the switch is de-activated. The current position is then set to zero — this is the home position. A key-press will stop the motion.

md2Move This function is used to move the positioning table. The calibration switch is ignored. A key-press will stop the motion.

5.6.4 Spectrum analyzer functions

uimAnalyzerOn This function resets the spectrum analyzer state.

resetAnalyzer This function opens or closes the spectrum analyzer for communication over the GPIB bus. If the function opens the spectrum analyzer, it queries the analyzer for its ID, and initializes the analyzer. In particular it sets the center frequency, span frequency, resolution bandwidth, amplitude reference level, sweep mode, and transfer data format.

writeSpect This function writes text string commands to the spectrum analyzer.

hp859x'invalidViInt16Rangelocal This function tests for the range of the spectrum analyzer source.

readAnalyzerSettings This function reads several settings of the spectrum analyzer. These settings are the center frequency, the span frequency, the resolution bandwidth, and the amplitude reference level. It reads the log/linear state. It is called by the function **openNewFile**.

getData This function reads data from the HP 8591E spectrum analyzer. It plots the data if the plot panel is open and the `updatePlot` flag is true. This function then saves the data to a file.

readDataFromAnalyzer This function gets data the from spectrum analyzer. It tells the spectrum analyzer to sweep data into an array and transfer that array to the computer. The data is converted to double format and scaled.

uirAnalyzerSettings This function is called by the user controls on the analyzer panel. This function sets the parameters of the spectrum analyzer. These parameters are the center frequency (9 kHz to 1.8 GHz), the span frequency (0 Hz or 100 Hz to 1.8 GHz), the resolution bandwidth (30 Hz to 3 MHz), and the amplitude reference level (-140 dBm to 30 dBm).

5.6.5 Control functions

uirData This function, called by the control button **Data**, collects data from a single location. If the `runRepeatedly` flag is true, this function collects data `MAXITER` times. This function open a data file and writes a header to data file with the function `openNewFile`. It then puts the spectrum analyzer in single sweep mode, collects data with the function `getData`, returns the spectrum analyzer to continuous sweep mode, and saves the data to a file.

uirMove This function, called by the control button **Move**, moves the positioning table by the amount set in the **Target** indicator. This function first moves the positioning table along a row to a new column and then moves it along a column to a new row. Before moving, this function tests that the positioning table is calibrated and that the stepper-motor driver is on.

uirMoveParameters This function copies the values of the **Target** from the user interface into an internal record.

uirRungrid This function, called control button **Run Grid**, moves the positioning table through a grid and takes data at each node in the grid. This function moves the positioning table to the start of the target region, then moves the table through all the points in the grid, taking data at each point. The table moves forward along each odd row and back along each even row. It moves steadily from the low rows to the high rows. After moving through the grid of points, the positioning table moves to the origin of the allowed region of motion. If the `runRepeatedly` flag is true, the positioning table runs through the grid repeatedly, and re-calibrates before each run through. Before starting the grid, the function tests that the positioning table is calibrated, the stepper-motors driver is on, and the spectrum analyzer is open for communication with the computer. Then the function opens a data file and writes a header. The spectrum analyzer is put in single sweep mode. The positioning table is moved through a grid and data is collected at each point in the grid. The spectrum analyzer is returned to continuous sweep mode. Finally the data is written to a file.

setGridSize This function calculates a set of consistent grid parameters **Number of Pixels**, **Pixel Size**, and **Grid Span**. It also updates the user interface.

uirGridParameters This function reads a new grid parameter from the user interface. It also checks if the new parameter is within allowed limits. If the parameter is not, the function returns the display to its previous setting.

uirFixedGridParameter This function is called by the **Fixed Parameter** slide on the grid panel. This slide chooses the grid parameter to hold constant while calculating a new set of consistent grid parameters. First this function turns the display indicator for the previous constant grid parameter to the active setting, then the function reads in the new constant parameter, and sets the display indicator for that parameter to in-active.

uimRunRepeatedly This function sets the flag to repeatedly run through grid during **uirRungrid**, or repeatedly take data during **uirData**.

uimHelp This is the help menu.

quit This function exits the program from the main panel. It turns off the stepper-motor and closes the spectrum analyzer.

uimQuit This function exits the program from the menu. It turns off the stepper-motor and closes the spectrum analyzer.

5.7 Notes

5.7.1 Hardware

The hardware consists of:

XY-9 Positioning Table Arrick Robotics, P.O.Box 1574, Hurst, TX 76953, 817-571–4528.

MD-2 Dual Stepper-Motor Driver Arrick Robotics, P.O.Box 1574, Hurst, TX 76953, 817-571–4528.

8591E Spectrum Analyzer with option 010 (Tracking Generator), option 041 (HP-IB interface), and option 130 (Narrow Resolution Bandwidths), Hewlett-Packard, Test and Measurement Customer Business Center, P.O.Box 4026, Englewood, CO 80155, 800-829–4444.

ScanEM-EC Electric Field Sensor Credence Technology, P.O.Box 5146, Santa Cruz, CA 95063, 408-459–7488.

Ready 9532 Personal Computer NEC Technologies, 1255 Michael Drive, Wood Dale, Illinois 60191, 800-366–9500.

GPIB Controller Card National Instruments, 6504 Bridge Point Parkway, Austin, TX 78730, 800-328–2203.

5.7.2 Software

The software consists of:

Windows96 Operating System Microsoft.

LabWindows/CVI Programming Environment National Instruments, 6504 Bridge Point Parkway, Austin, TX 78730, 800-328–2203.

Custom Electric Field Imaging System Software Aztec Corporation, 371 Moody St., Suite 104, Waltham, MA 02154, 781-647–1534.

5.7.3 Other notes

Note: If the spectrum analyzer does not work, check at the operating system level for IRQ conflicts with `C:/GPIB95/diagnostic.exe` (or **Start>>Programs>>NI-488.2M Software for Windows95>>Diagnostic**). Also check the Device Manager in **Start>>Settings>> Control Panel:System.**

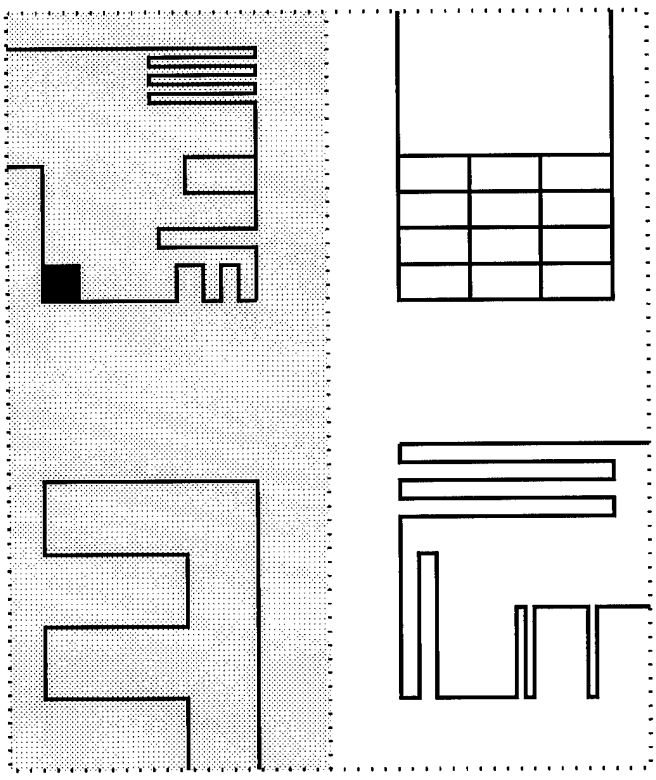


Figure 37: Schematic of manufactured circuit board used for testing; the individual circuits are labeled according to

IV	III
II	I

.

6 Data acquired from system

This section describes the different data sets acquired by our various sensors. In our data collection efforts we used:

1. A 50 Ω stripline (see section 6.3)
2. A simple loop constructed from 20 gauge wire on a plastic backing (see Figure 44 on page 59)
3. A series of loops constructed from 20 gauge wire on a plastic backing (see images on page 77)
4. A 6 × 6 inch circuit board constructed for us with 10 mil traces (shown schematically in Figure 37). Note that the left half of the board has a ground plane on the side of the board opposite the traces. The four separate circuits on this board are labeled I–IV as follows:

IV	III
II	I

.

After the system was demonstrated to be working, we had planned on performing in situ tests on a system consisting of a chassis, a CPU card, and a test board. The chassis is an open passive backplane chassis computer box with five slots for full or half length AT-style cards. This chassis has the advantage that it is easy to maneuver the sensor attached to the positioning table to locations directly above the board. The chassis is also compact. In one slot is the CPU card which drives both the ISA/EISA bus and the test card. The printed circuit board under test is in another slot. The software CHECK IT (manufactured by Touchstone Software corporation) which is stored on the CPU card, would have been used to drive the test card. This commercial software knows how to drive every chip on many standard AT-style cards. Unfortunately, with a properly functioning Srico sensor not being available, we did not continue with data from this source.

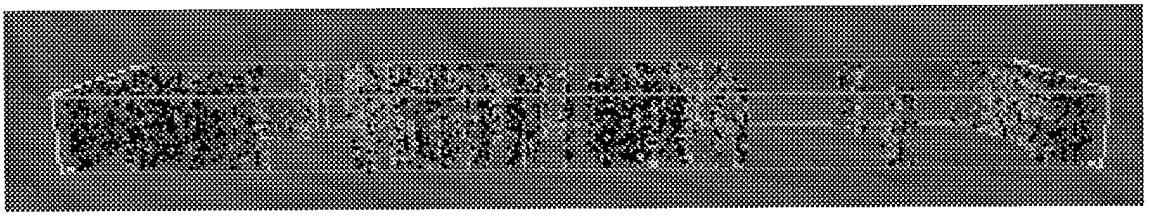


Figure 38: Visualization of the explicit three dimensional data set

In each experiment (except one) the traces were driven with a 5V peak-to-peak sinusoidal signal at 10 MHz. Note that all data was collected in dBm. Data was acquired at a grid spacing of 18 mil (from 4×4.5 mil, see page 40).

We experimented with the data acquistion system (see section 6.1), and then made several systematic studies (the data files are described in section 6.2).

6.1 First look at the data

The first sets of data were taken to determine how to proceed. The following data sets were obtained at 401 uniformly spaced frequency values in the range 1–51 MHz (each data value was for a band of width 30 KHz).

- Data set 1 contains data taken from the stripline at 81×31 spatial locations. The data values ranged from -78 dBm to -7 dBm, with a mean of -65 dBm.
- Data set 2 contains data taken from the stripline on the following day, with the sensor rotated 90 degrees; the data was taken at 36×81 spatial locations.
- Data set 3 contains data taken from the simple loop with a 1 V peak-to-peak sinusoidal signal at 10 MHz (all other experiments had a 5 V peal-to-peak signal). The sensor took data from 1–21 MHz.

Having acquired these data sets, we looked at them (see section 6.1.1) and then applied simple image processing techniques to understand them better (see sections 6.1.2–6.1.5).

6.1.1 Initial data visualization

When given implicit three-dimensional data (i.e., data specified by $z = z(x, y)$) it is common to construct "contour" maps. These are usually represented as curves in a two-dimensional image, with each contour represented by a single curve. With explicit three-dimensional data (i.e., a data value assigned to each point in space, or $v = v(x, y, z)$) it is also possible to construct "contour" maps. In this case, each "contour" is given by a three-dimensional surface.

Figure 38 contains a three-dimensional contour image of data set 1 (all of our image analysis and visualization in this sectin was performed with the Khoros software package, see [1] and section B) with higher activity shown darker. The bounding box, a rectangular parallelepiped with a 5-to-1 aspect ratio, is also shown. There are several disjoint "blobs" which roughly correspond to enhanced activity in several disjoint frequency bands.

It is easier to see regions of electromagnetic activity by looking at slices through this three dimensional data set. Figure 39 contains two slices down the center of the stripline, one in the x direction (of size 401×31) and one in the y direction (of size 401×81). In these images, regions of high electrical activity are shown lighter. Figure 40 contains perspective views of the same data. From these images, the regions of higher electrical activity clearly stand out. The light band towards the left corresponds to the 10 MHz input signal while the other, less intense, bands represent higher harmonics (30 MHz and 40 MHz) and two strong interference frequencies (near 22 MHz and 44 MHz).

A slice through the data set at 10 MHz is shown in Figures 41 (planar view with color coding of heights) and 42 (a perspective view showing the range of the data). As predicted, while the input data was very narrow (the stripline had a real width of 3 pixel values) the output response is much wider.

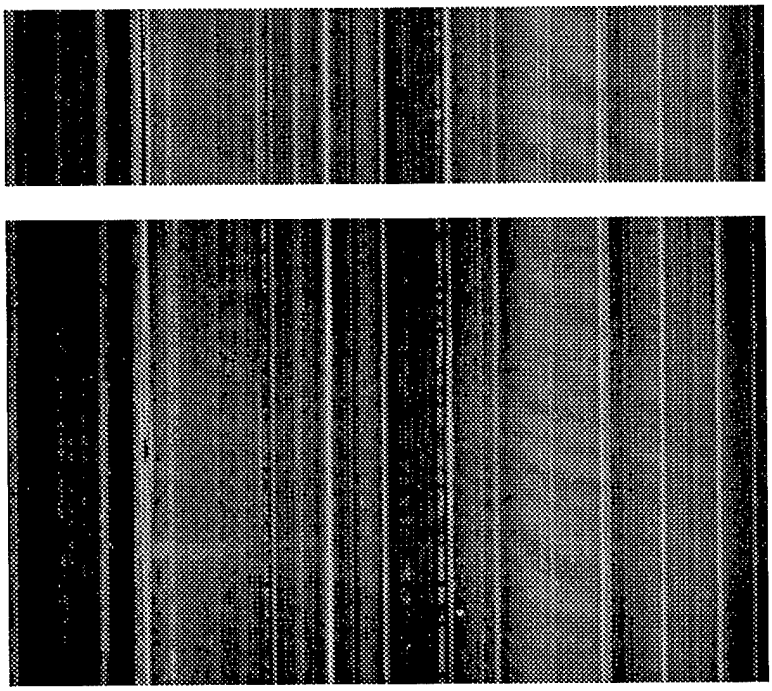


Figure 39: Two slices through dataset 1 (top image has dimensions 401×31, bottom image has dimensions 401×81)

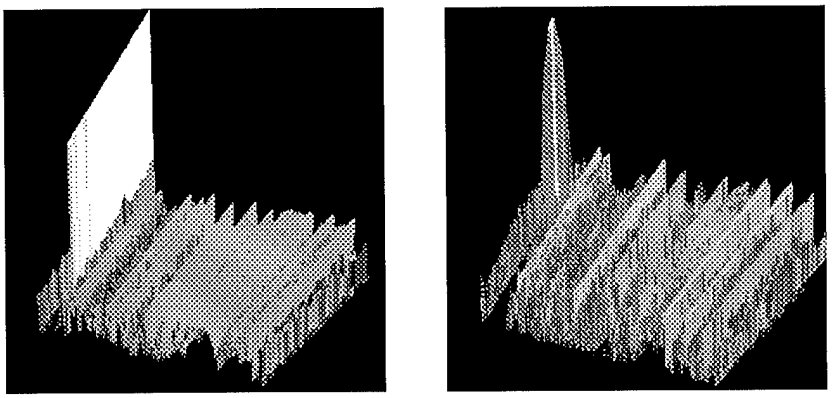


Figure 40: Perspective images of the data in Figure 39.

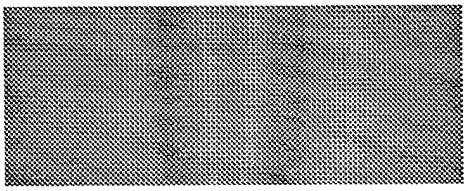


Figure 41: A two-dimensional spatial image of the stripline at 10 MHZ (dataset 1)

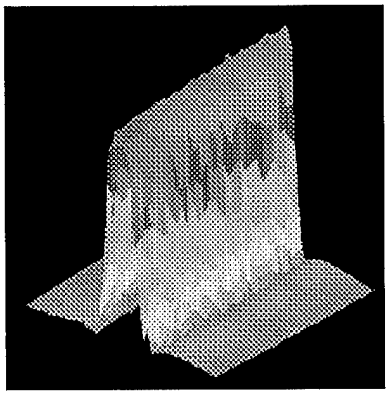


Figure 42: A perspective view of the stripline at 10 MHZ (dataset 1)

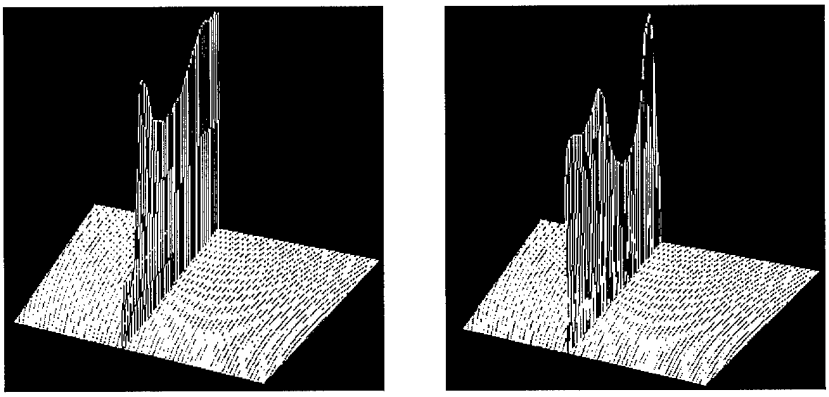


Figure 43: Spatial slices showing data at 10 MHz

Spatial slices of dataset 3 are shown in Figure 43. A large signal is apparent at 10 MHz and there are no other strong signals. The data at 10 MHz is shown in Figure 45; in these images the outline of the wire is clearly visible. Figure 45 shows how raising the "noise" floor (below which data is rejected) has the effect of sharpening the image of the wire.

6.1.2 Finding the convolution filter: dataset 1

Section 7.4 describes how to use deconvolution techniques to sharpen images. Sections 6.1.2–6.1.5 show the results of such an analysis.

We took the data shown in Figures 41 and 42 (recall the 10 MHz data is from dataset 1) and averaged it (in the x (i.e., 31 point) direction) to obtain a composite curve representing the spreading of the input. This is **d**, as defined in section 7.4 (shown in Figure 46). We know the original input **s** was the stripline, which looks like a boxcar function of width 3 pixels. Using Fourier transforms (after subtracting the mean from the image values to make them have mean zero and padding the width to be 256 pixels instead of 81 so that fast Fourier transforms can be used) we find the functions **f** and **g**, also shown in Figure 46.

6.1.3 Applying the convolution filter: dataset 1

We took the entire 10 MHz data set shown in Figures 41 and 42, subtracted off the mean and zero padded it to obtain 47 (top). We applied the convolution filter we obtained, **g** to this data and obtained Figures 47 (bottom) and 48, which show the de-convoluted data. The stripline is clearly visible in these images. Applying a simple thresholding (recall, the data is in dBm) would result in a clear delineation of the stripline.

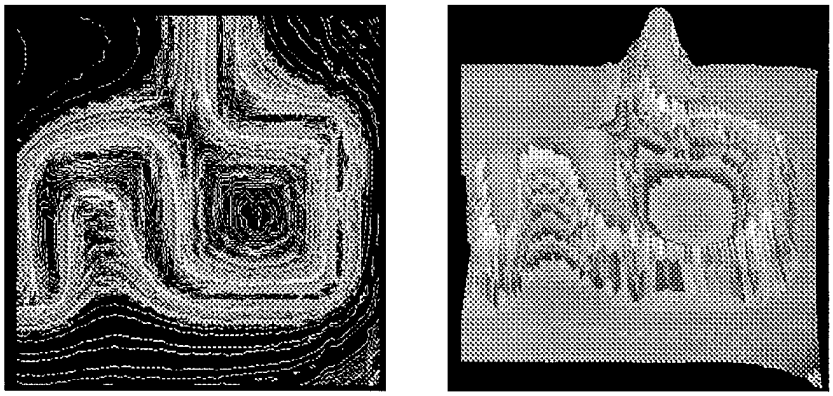


Figure 44: Data at 10 MHz: contour plot and perspective view

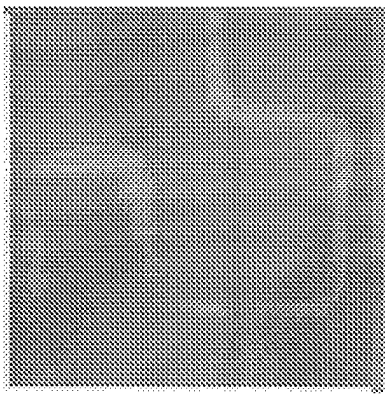


Figure 45: Data at 10 MHz with a high "noise" threshold

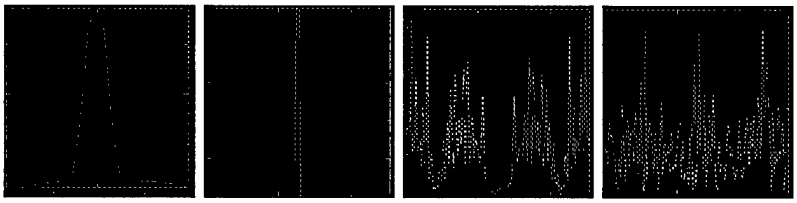


Figure 46: From left to right: the averaged spreading function (**d**), the known input function (**s**), the magnitude of the point spreading function (**f**), the convolution filter (**g**).

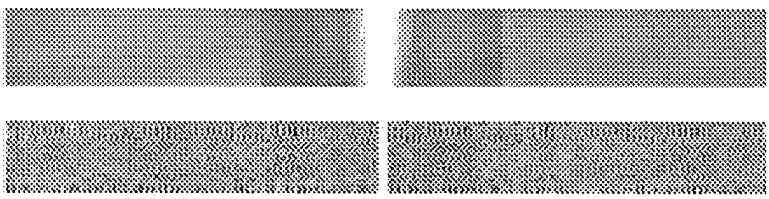


Figure 47: Top: The input to the de-convolution process (note zero padding at the edges of the image) Bottom: The output of the de-convolution process

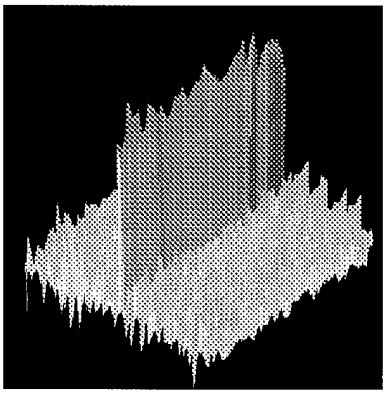


Figure 48: A perspective view of the filtered stripline data at 10 MHZ

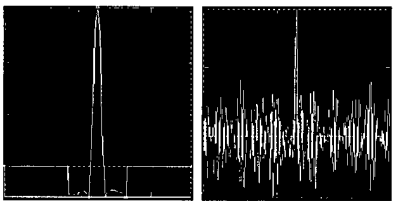


Figure 49: Left: A slice through the original data. Right: A slice through the de-convoluted data.

As a simple proof of principle, Figure 49 shows a line through the data both before and after the filtering operation. The de-convolution process has clearly sharpened up the signal, at the expense of adding noise away from the main signal.

6.1.4 Finding the convolution filter: dataset 2

Dataset 2 showed the same behavior in three dimensions that dataset 1 did (essentially, a large signal at 10 MHz and other activity at 22, 30, 40, and 44 MHz), see Figure 50. The data at 10 MHz (analogous to Figure 41) is shown in Figure 51. The data shows what we believe to be a physical disruption to the experimental setup (i.e., sometime during data collection the setup was shaken in some way). This perturbation of the data was not compensated for in any way during the subsequent data processing.

6.1.5 Applying the convolution filter: dataset 2

A convolution filter was obtained in the usual way and was used to de-convolute the measured data, see Figure 52.

The convolution filters found for both datasets 1 and 2 would be the same (recall the difference was that the sensor was rotated 90 degrees) if the sensor was isotropic. The convolution filter obtained from dataset 1 was applied to dataset 2 to determine if the response is isotropic. The results are shown in Figure 54. These images clearly show that the original stripline is recovered (but with an increase in the noise floor, as expected). After checking that the magnitude of the response, we conclude that the sensor is nearly isotropic.

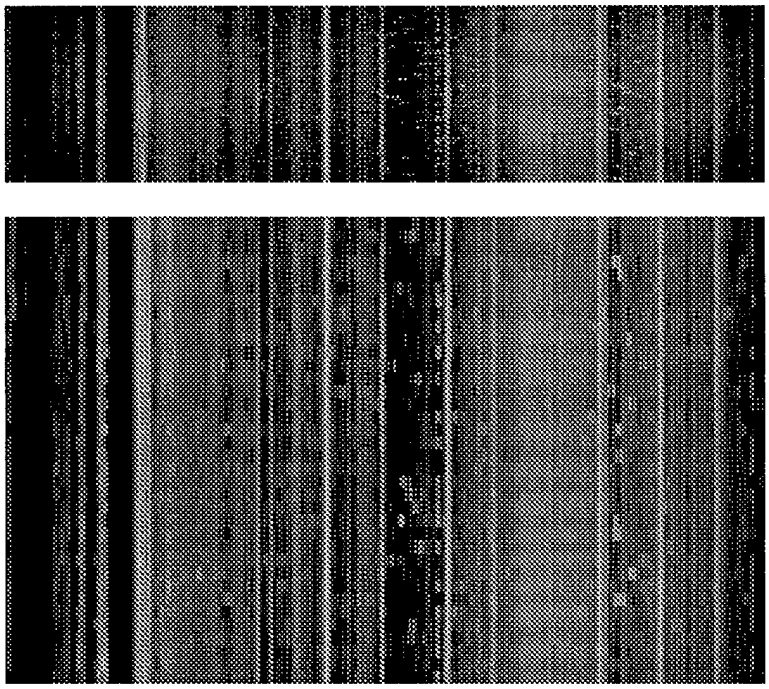


Figure 50: Two slices through dataset 2 (top image has dimensions 401 × 36, bottom image has dimensions 401 × 81)

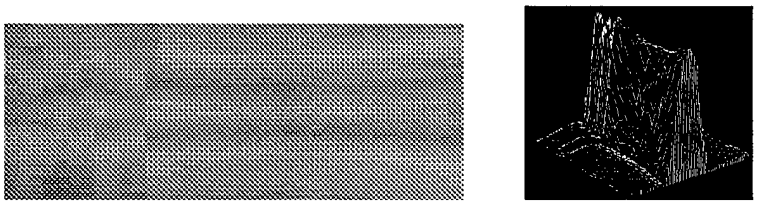


Figure 51: A two-dimensional spatial image of the stripline at 10 MHZ and a perspective view (dataset 2)

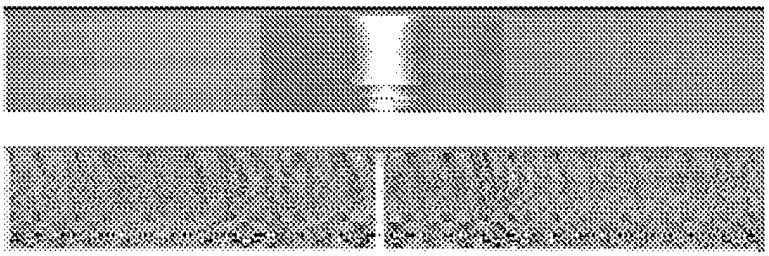


Figure 52: Top: The input to the de-convolution process Bottom: The output of the de-convolution process

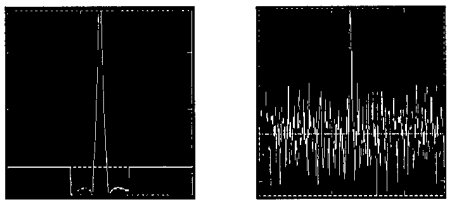


Figure 53: A slice through the original data and a slice through the de-convoluted data

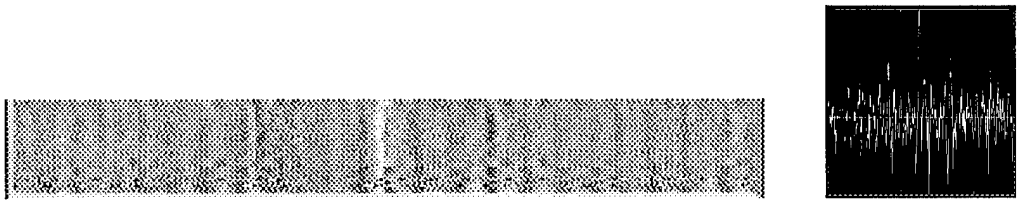


Figure 54: The output of the de-convolution process and a slice through the de-convoluted data

6.2 Systematic data collection

We used the Credence Technology sensors[5] (see section 4.1) to collect data from the circuit boards of type 2–4 described on page 55. Each datafile was translated to Matlab and given a name that has the structure `ft(year)(month)(day)(file number).m`.

- The following files contain data obtained from board 2 (see Figure 44), using the electric field sensor
  - `ft97072302.m` — `ft97072304.m`
  - `ft97072401.m` — `ft97072403.m`
  - `ft97072501.m`
  - `ft97081601.m` — `ft97081602.m`
  - `ft97081905.m`
- The following files contain data obtained from board 2, using the magnetic field sensor
  - `ft97081903.m` — `ft97081904.m`
- The following files contain data obtained from board 3 (see images on page 77), using the electric field sensor
  - `ft97082001.m`
  - `ft97103101.m` — `ft97103129.m`
  - `ft97110501.m` — `ft97110507.m`
- The following files contain data obtained from PCB 4 (see Figure 37), using the electric field sensor (see section 7.5.2)
  - `ft98012901.m` — `ft98012902.m` (for circuit I)
  - `ft98013001.m` — `ft98013005.m` (for circuit I)
  - `ft98013101.m` — `ft98013107.m` (for circuit II)
  - `ft98020101.m` — `ft99020103.m` (for circuit III)

[5]Most of the data was taken with the electric field sensor, two data files were taken with the magnetic field sensor.

6.3 Stripline

The estimated performance of the system built depends critically on the size of the electric fields to be measured. To verify the performance estimates of the sensor system (performed in the phase I final report, see [49]), we needed to accurately determine the electric field for some of test configurations.

We choose to measure the field above a stripline (which we also constructed). We choose to use `Kaskade 2.1` (a publically available finite element package) in order to numerically determine the field.[6]

6.3.1 Geometry of stripline and numerical domain

The stripline sits on a substrate as shown in figure 55. The stripline is a few microns in thickness, and has a width of 1/8 inches. The stripline sits 1–2 mm about the (grounding) substrate. The sensor will be a few mm above the stripline.

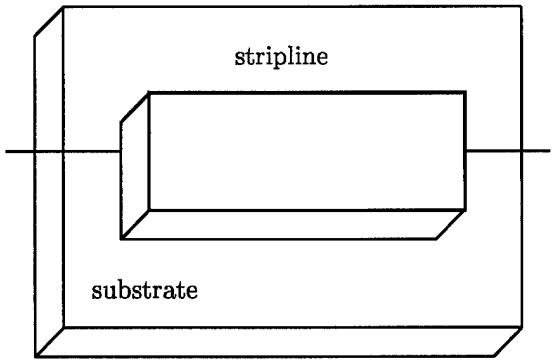


Figure 55: Sketch of stripline

Assuming that the stripline is infinitely long, the view from the end of the stripline appears as in Figure 56. Due to the line of symmetry going through the center of the stripline, the geometry of the numerical problem to be solved appears as in figure 57. In this Figure we have introduced artificial boundaries on the top ($y = B$) and right-hand side ($x = A$); these are required to solve the problem in a finite domain. In practice, we will choose A and B to be very large.

The equation to be solved in the numerical domain is Laplace's equation $\nabla^2 V = 0$. In Figure 57 the $V = 1$ refers to the constant voltage on the stripline, the $V = 0$ on the bottom is the constant voltage on the grounding plane, and the $V = 0$ on the top and right edges of the domain ($y = B$ and $x = A$) corresponds

[6]See `http://www.zib.de/SciSoft/kaskade/`. for more information, or acquire the code by ftp'ing to `ftp elib.zib-berlin.de` and obtain the files from `/pub/kaskade/3.x` and the manuals from `/pub/kaskade/Manuals/3.x`.

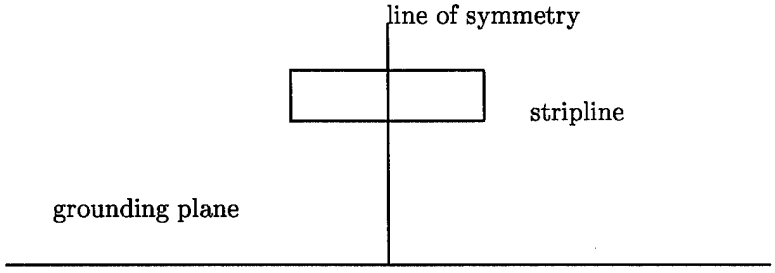


Figure 56: End-on view of stripline

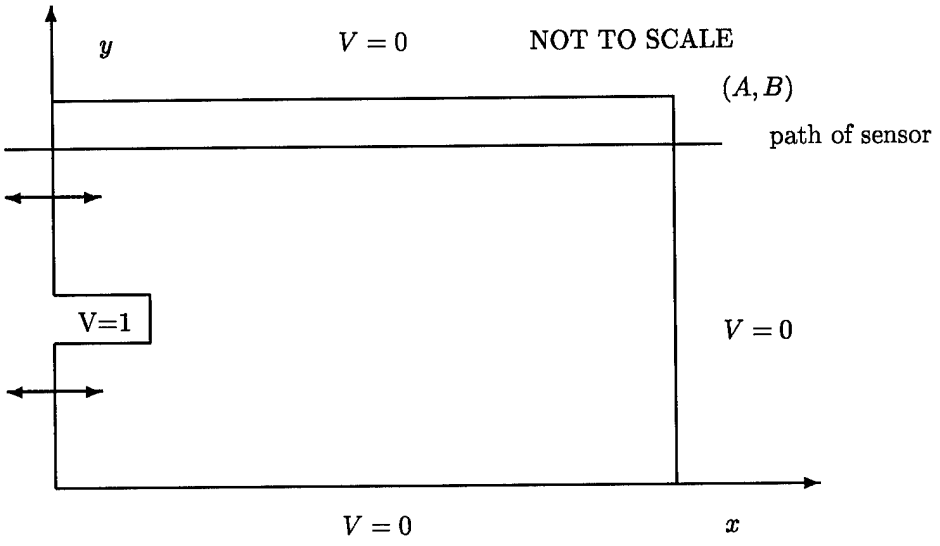


Figure 57: Numerical domain for solving Laplace's equation.

to $y = x = \infty$. The double headed arrows on the left of the figure are meant to indicate Neumann boundary conditions ($\partial V/\partial n = 0$).

While this problem can be solved analytically (in principle) using the Schwartz–Christoffel transformation (see [50]) we have chosen to solve this problem numerically using **Kaskade.**

6.3.2 Numerical results

We have selected the following values (all units are in microns):

- the stripline has a height of 2 units
- the bottom of the stripline is 1000 units above the grounding plane (corresponding to 1 mm)
- the stripline extends 1600 units to the right (corresponding to 1/16 inch)
- $A = 12000$ units (i.e., a large value)
- $B = 7000$ units (i.e., a large value)
- the sensor path is 3002 units above the grounding plane (corresponding to 2 mm above the stripline)

We performed an initial triangulation of the domain (see the sketch in Figure 58, which is not to scale); this triangulation has 8 points, 13 lines, and 6 triangles. This was input to **kaskade**, and the program automatically refined the triangulation to get more accurate results. Figure 59 contains the final triangulation (it has 40132 points, 119967 lines, and 79836 triangles) and figure 60 shows 25 field contour lines corresponding to the final solution.

Now that we have the solution (i.e., V) numerically determined over the domain, we can compute the electric field (which is the gradient of V). We compute the gradients in the x and y directions ($E_x = \partial V/\partial x$ and $E_y = \partial V/\partial y$), and the resultant electric field ($E = \sqrt{E_x^2 + E_y^2}$). Note that the assumption of infinite extent of the strip line implies that $E_z = 0$.

For a sensor that depends on an oriented electric field (such as the Srico sensor, or a loop sensor), the response will depend on either E_x, E_y, or a linear combination of these. Having numerically computed E_x and E_y, we can then compute the total electric field as $E = \sqrt{E_x^2 + E_y^2}$. Figures 61, 62, and 63 show the numerically computed E, E_x, and E_y.

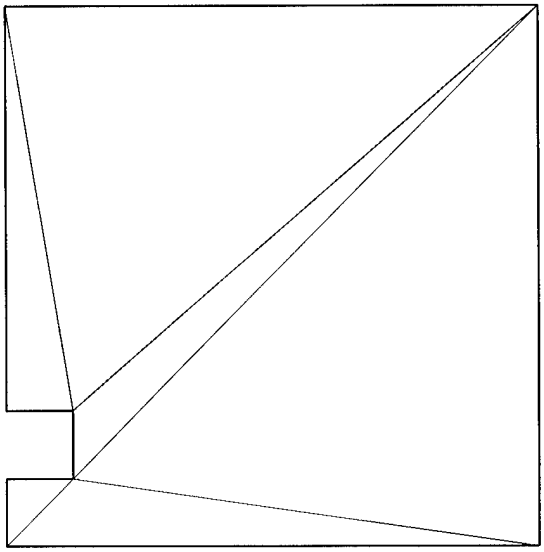


Figure 58: Initial triangulation of the numerical domain (not to scale)

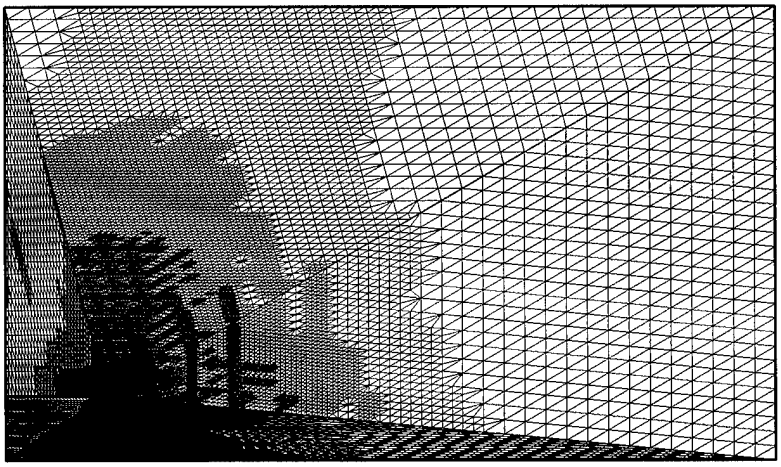


Figure 59: Final triangulation of the numerical domain

6.3.3 Conclusions

The electric field values in figures 61–63 are very close to the values modelled in the Phase I final report (see [49, Figure 11 on page 25]). We conclude that the system characteristics are as specified in the Phase I final report.

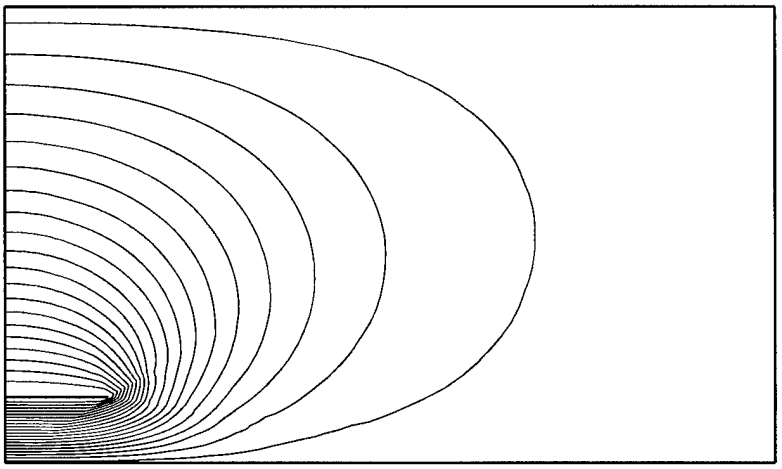


Figure 60: Contours of constant voltage

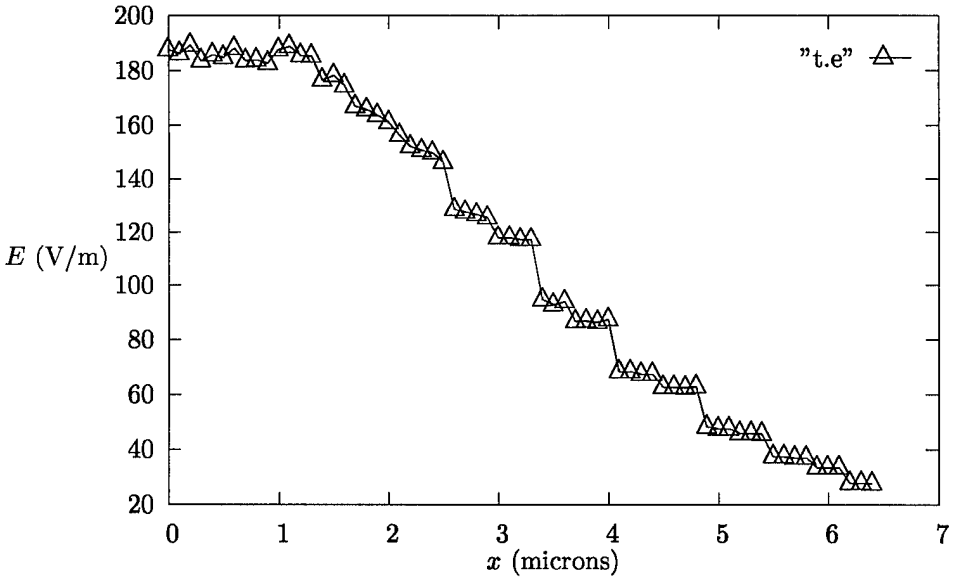


Figure 61: E for varying x and fixed y.

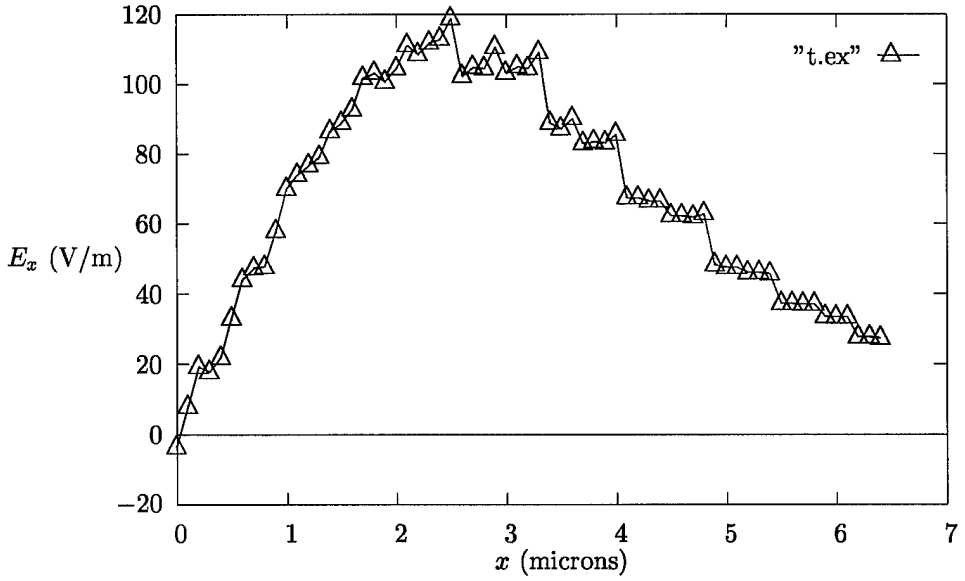


Figure 62: E_x for varying x and fixed y.

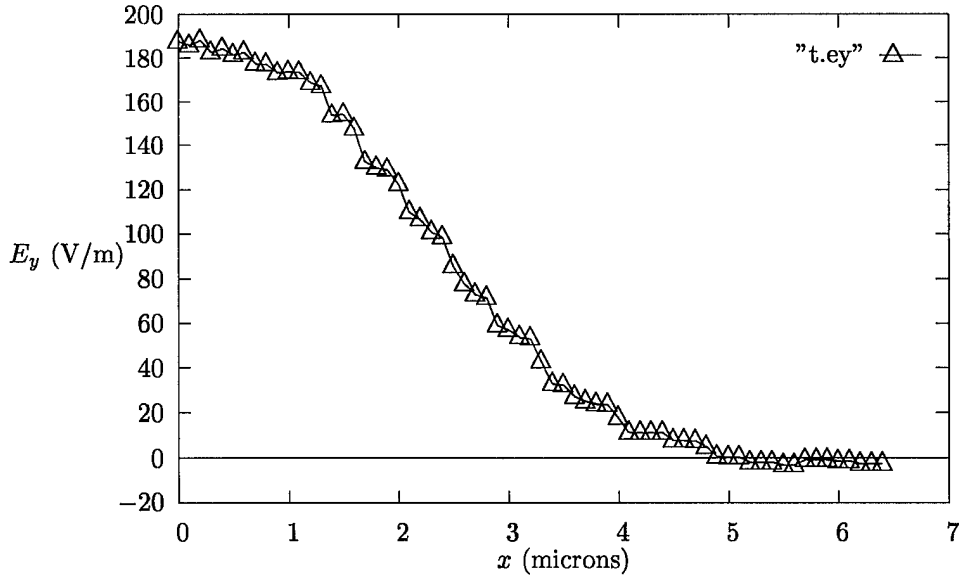


Figure 63: E_y for varying x and fixed y.

7 Image Processing

This section describes several different imaging investigations that we performed:

1. The use of Radon transforms for image alignment (see section 7.1)
2. A study of the sensor noise (see section 7.2)
3. An investigation using principal components analysis (see section 7.3.1)
4. The use of deconvolutions for image restoration (see section 7.4)
5. The use of the "log" filter to identify image components (see section 7.5)

7.1 Alignment of images

For testing circuit boards, we will want to compare images of the radiated field from two different boards. A natural concern is correcting for the alignment of the boards (i.e., one board may be rotated 30° relative to another board, making comparisons difficult).

Because the traces on a PCB are mostly rectilinear (running in the x and y directions) it is possible to correct for modest rotations using the Radon transform. The Radon transform computes projections of a function defined in a two-dimensional region (represented by a matrix of values) along specified directions. The Radon transform of $f(x, y)$ is the line integral (see [48])

$$R_\theta(x) = \int_{-\infty}^{\infty} f(x\cos\theta - y\sin\theta, x\sin\theta + y\sin\theta)\, dy$$

The Radon transform values $R_\theta(x)$ can be shown as the intensity in a two-dimensional plot of values (θ versus x).

Our technique for alignment is as follows:

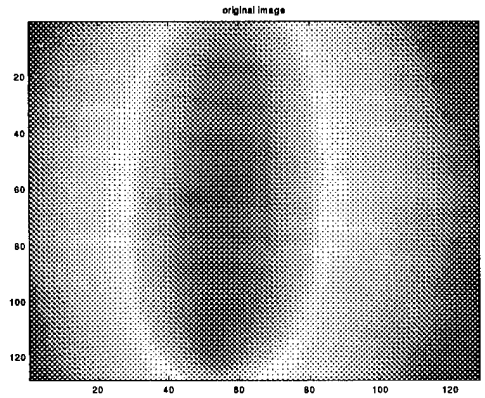
1. From an input image, compute a specified number of contours (these will be mostly rectilinear, since the underlying electric fields are).
2. Compute the Radon transform of the contour image for a wide range of angles (say for θ from 0° to 180° in steps of 1°).
3. When θ is correctly aligned with the PCB (i.e., the unknown image rotation has been discovered), then the Radon transform will appear as horizontal lines (see the figure accompanying item number 6 in section 7.1.1). The variance (in the x direction) will be a strong indicator of whether or not the lines are horizontal; this value will be large for horizontal lines, and smaller for smeared out data and line segments that appear at differing angles.
   Compute the variance (computationally, we compute the standard deviation) and determine the angle θ where it is a maximum. This corresponds to the correct rotation angle.
   (Note that this is *not* the way that is described in the Matlab manual[44, pages 6-23 to 6-25]. They recommend just using the peak values of the Radon transform to determine the correct angle θ. That technique does not work for our stripline data.)
4. Given an approximation to the rotation angle θ, reduce the range of angles and iterate the last two steps.

Section 7.1.1 contains a worked example. Section D.9 contains the source code for the alignment subroutine (`align.m`). Section D.10 contains a driver that demonstrates the use of the `align.m` routine.

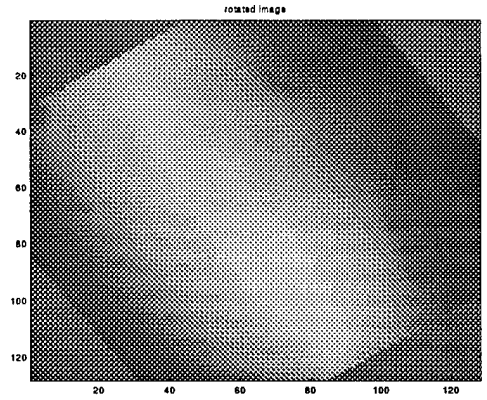
7.1.1 Alignment example

This section considers a complete example. The results are from using the Matlab program in section D.10.

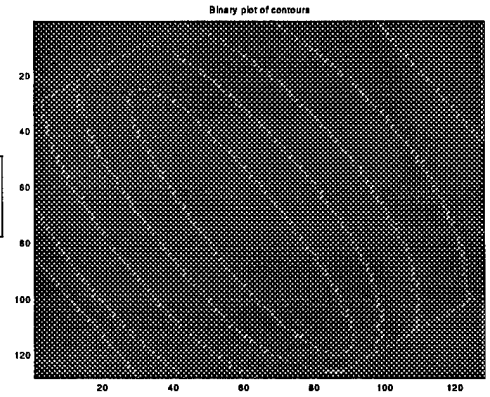
1. The input Figure is as shown



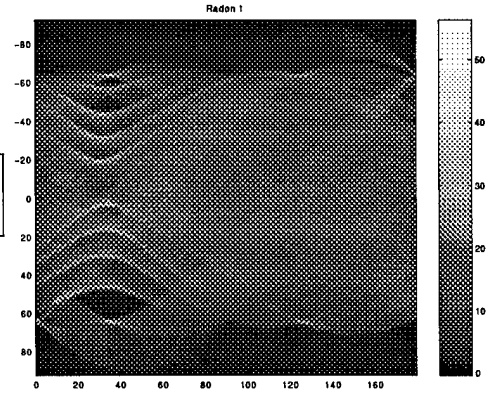
2. An angle of 34.567° is chosen and the Figure is rotated by this amount. This is the "unknown" rotation that must be corrected for.



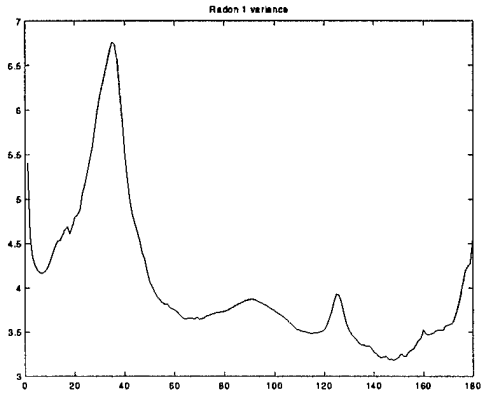
3. We call the alignment routine with a request for 10 levels. The contours it finds are as shown



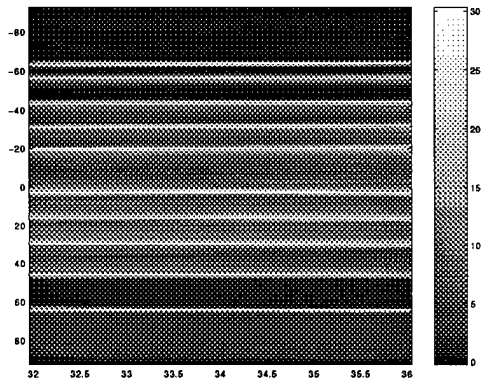
4. The Radon transform of the contour image is as shown (here, θ varies from 0° to 180° in steps of 1°)



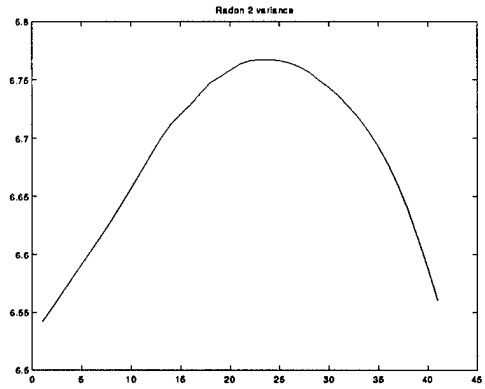
5. The column standard deviations are as shown to the right. The peak in the standard deviation can clearly be seen. This peak is at $\theta = 34$



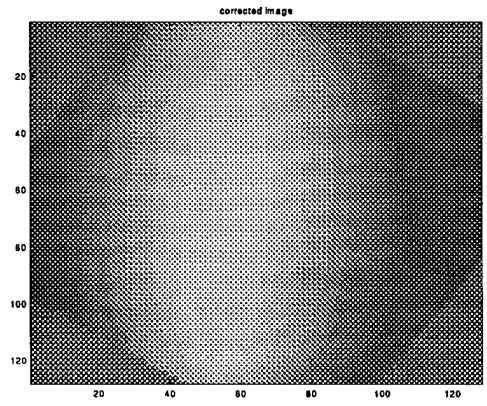
6. We localize the search to a region around $\theta = 34°$. The Radon transform of the contour image is as shown (here, θ varies from 14° to 54° in steps of 0.1°). The fact that it looks like horizontal lines indicates we have the approximate angle correct. There are as many lines as there were contours.



7. The column standard deviations are as shown to the right (there is one column for each angle value). The peak in the standard deviation can clearly be seen. This peak is at $\theta = 34.3$, which is close to the input value.



8. After finding the approximate alignment angle (i.e., $\theta \approx 34°$) we can rotate the image by that angle to determine our aligned image as shown.



7.2 Testing for normality of data

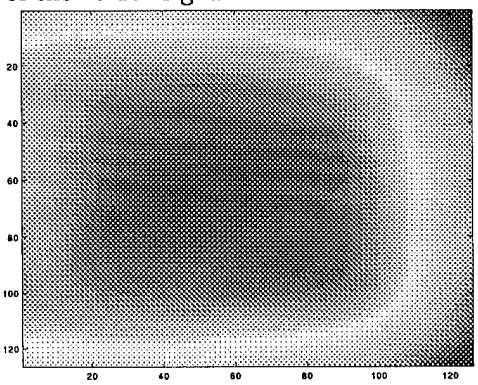
The theoretical background for many signal and image processing algorithms requires that some component of the signal (such as the noise) be normally distributed. Hence, one of the first tests we performed was to determine if the errors in the data acquired are normally distributed. We had originally presumed that the errors would be normally distributed on a linear scale, but this was not the case. Analysis of some of the data, described in section 7.2.1, indicates the the data is not normally distributed on either the db or the linear scale.

7.2.1 Testing for normality of sensor data

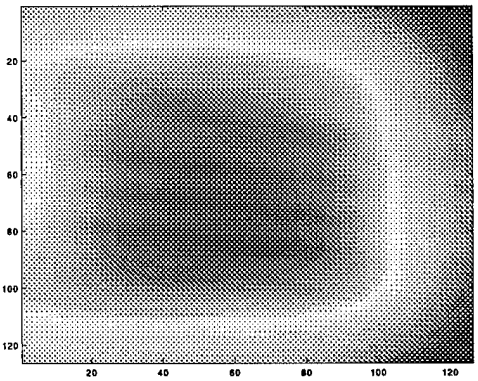
To determine whether the errors in our data acquisition process were normally distributed, we did the following (see the Matlab program `do_test_normality` listed in section D.14):

- Note: all of the following steps were performed for the data both on a db scale (as collected from the sensor) and on a linear scale.

1. We read in all six of the 971029 datasets (each had dimension 127×127, see section 6.2). On the db scale, these values have a mean (μ) of -16 and a standard deviation (σ) of 1.9. On the linear scale, these values have $\mu = 0.16$ and $\sigma = 0.03$.
2. These 6 datasets were averaged together, to get an estimate of the "true" signal.

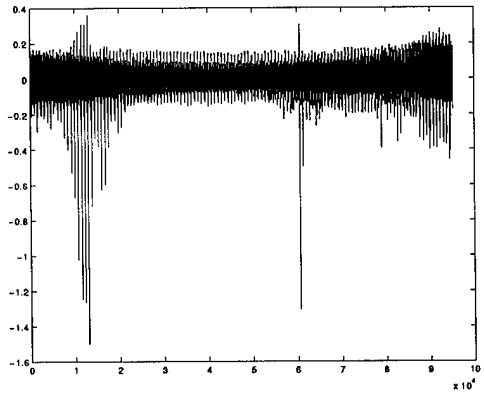


3. The "true" signal on a db scale.

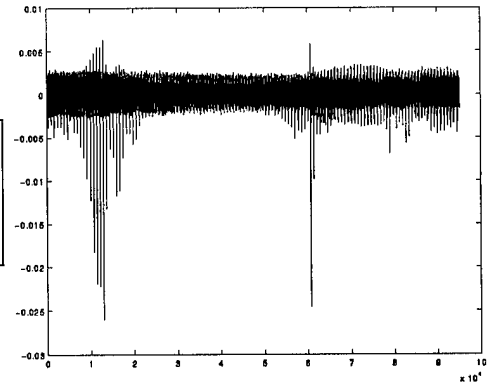


4. The "true" signal on a linear scale.

5. The "true" signal was subtracted from each of the six images, to obtain six arrays of size 127×127 that should contain nothing but noise values. These values were combined into a single "noise vector" of 95,256 elements.

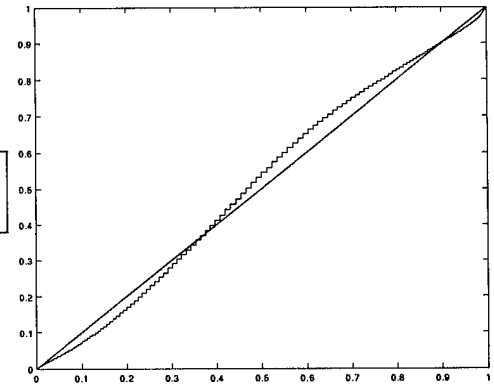


6. The "noise vector" on a db scale. Note that some values appear to be spurious. This vector has $\mu = -2 \times 10^{-17}$ (which is essentially zero, in finite precision arithmetic) and $\sigma = 0.06$.



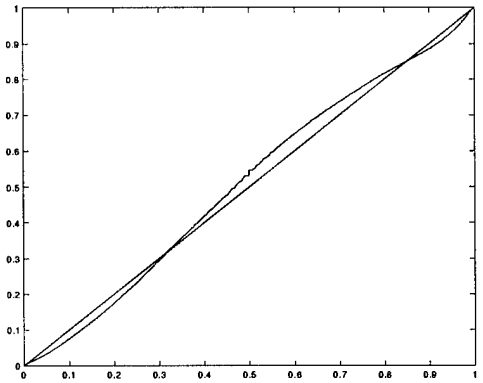
7. The "noise vector" on a linear scale. The spurious values also appear here. This vector has $\mu = -2 \times 10^{-19}$ (which is essentially zero, in finite precision arithmetic) and $\sigma = 0.001$.

8. We used the non-parametric Kolmogorov–Smirnoff test to determine normality (see [51, page 623]). The subroutine we used was from an early version of the MathWorks Statistics Toolbox. The Kolmogorov–Smirnoff test computes the maximum of the absolute value of the difference between the data's cumulative distribution function and the hypothesized cumulative distribution function (in this case, corresponding to a normal distribution). If this value is too large, then it is concluded that the two distribution functions are not the same. The results of the test clearly stated that neither dataset is normally distributed.



9. The two cumulative distribution functions for data on the db scale.

10. The two cumulative distribution functions for data on the linear scale.



11. Reasoning that the spurious looking values were keeping the data from looking like it came from a normal distribution, we attempted three fixes
    (a) We passed each "noise vector" though a 3-pt, 5-pt, and 7-pt median filter.
    (b) We replaced all values more than 3 standard deviations from the mean with the mean value. (Of this 95,256 values, this changed 580 values in the db case and 331 values in the linear case.)
    (c) We consider subsets of data away for the spurious looking values.

    None of these techniques resulted in datasets that were normally distributed.

7.3 Large data sets

One way to work effectively on large data samples is to apply statistical methods. Multivariate data analysis methods are not intended to replace physical analysis—these should be seen as complementary. Statistical methods can effectively be used to run a rough preliminary investigation, to sort out ideas, to put a new ("objective" or "independent") light on a problem, or to point out aspects which would not come out in a classical approach. Physical analysis is necessary subsequently to refine and interpret the results. Widely-used multivariate methods include:

- Correspondence analysis
- Discriminant analysis
- Hierarchical cluster analysis
- Minimal spanning tree
- Non–hierarchical clustering, or partitioning
- Principal components analysis

The last method, the use of principal components analysis, is readily applicable to our imaging needs (see next section).

7.3.1 Principal components analysis: Description

Among the objectives of principal components analysis are the following.

- dimensionality reduction;
- the determining of linear combinations of variables;
- feature selection: the choosing of the most useful variables;
- visualization of multidimensional data;
- identification of underlying variables; and
- identification of groups of objects or of outliers.

Consider the n_f "images" that the sensor system obtains (each image indicating the intensity at a specific frequency, at $n_x \times n_y$ spatial locations). Call this image $I^{(f)}$. These multiple images might have arisen from, for example, different frequency bins or datasets acquired at different times.

To motivate the following analysis, consider turning each image $I^{(f)}$ (which is a $n_x \times n_y$ array of pixels) into a single array of $n_x n_y$ pixels (by, for example, stacking the rows of pixel data end-to-end). The images, in this interpretation, define an n_f-element basis in a $n_x n_y$ dimensional space.

Finding a set of k principal axes (with $k < n_f$) allows the objects to be adequately characterized on a smaller number of (artificial) variables. This is advantageous as a prelude to further analysis as the $n_f - k$ dimensions may often be ignored as constituting noise; and, secondly, for storage economy. Typically, reduction of dimensionality is practicable if the new axes account for approximately 75% or more of the variance (there is no set threshold—the analyst must judge). The cumulative percentage of variance explained by the principal axes is consulted in order to make this choice.

The technique for finding new principal axes is to determine the eigenstructure of the appropriate covariance matrix. Define σ_{ij} to the covariance between images i and j, that is $\sigma_{ij} = \mathbf{E}\left[I^{(i)} I^{(j)}\right]$, Then the $n_f \times n_f$ matrix $\Sigma = (\sigma_{ij})$ will have a zero eigenvalue whenever there is a linear dependency within the data. The eigenvector associated with a zero (or numerically very small) eigenvalue indicates the linear dependence. (Note that Σ is a real symmetric matrix, and hence all of its eigenvalues are real, see [51, page 131]).

In feature selection we want to simplify the task of characterizing each object by a set of attributes. Linear combinations among attributes must be found; highly correlated attributes (i.e., closely located attributes in the new space) allow some attributes to be removed from consideration; and the proximity of attributes to the new axes indicate the more relevant and important attributes.

Often it is relatively easy to identify the highest or second highest components, but it becomes increasingly difficult as less relevant axes are examined. The objects with the highest loadings or projections on the axes (i.e., those which are placed towards the extremities of the axes) are usually worth examining: the axis may be characterizable as a spectrum running from a small number of objects with high positive loadings to those with high negative loadings.

7.3.2 Principal components analysis: Illustrative example

As a illustrative example of principal components analysis, consider having 4 images of 3×3 data. For computational purposes we choose $\{I^{(1)}, I^{(2)}, I^{(3)}\}$ to be random 3×3 matrices (i.e., each element uniformly distributed on the interval $[0, 1)$). (The computations in this section were performed with the programs in sections D.16 and D.15.)

$$I^{(1)} = \begin{bmatrix} 0.9501 & 0.4860 & 0.4565 \\ 0.2311 & 0.8913 & 0.0185 \\ 0.6068 & 0.7621 & 0.8214 \end{bmatrix} \quad I^{(2)} = \begin{bmatrix} 0.4447 & 0.9218 & 0.4057 \\ 0.6154 & 0.7382 & 0.9355 \\ 0.7919 & 0.1763 & 0.9169 \end{bmatrix}$$

$$I^{(3)} = \begin{bmatrix} 0.4103 & 0.3529 & 0.1389 \\ 0.8936 & 0.8132 & 0.2028 \\ 0.0579 & 0.0099 & 0.1987 \end{bmatrix}$$

We then chose several different matrices to combine with the above three.

- We choose $I^{(4)}$ to also be a random 3×3 matrix:

$$I^{(4)} = \begin{bmatrix} 0.104491 & 0.391965 & 0.642526 \\ 0.095317 & 0.296533 & 0.761220 \\ 0.971972 & 0.073174 & 0.923405 \end{bmatrix}$$

The covariance matrix for $\{I^{(1)}, I^{(2)}, I^{(3)}, I^{(4)}\}$ is

$$\Sigma = \begin{bmatrix} 3.8192 & 3.2413 & 1.7657 & 2.5473 \\ 3.2413 & 4.5099 & 2.1338 & 2.4351 \\ 1.7657 & 2.1338 & 1.8560 & 1.4266 \\ 2.5473 & 2.4351 & 1.4266 & 2.4949 \end{bmatrix}$$

The eigenvalues of this Σ are $\{10.4, 1.10, 0.70, 0.50\}$. None of these eigenvalues are particularly small, and we expect that there is no linear dependence within the data.

- We choose $I^{(5)} = I^{(1)} + 2I^{(2)}$, so that

$$I^{(5)} = \begin{bmatrix} 1.8395 & 2.3296 & 1.2679 \\ 1.4620 & 2.3677 & 1.8894 \\ 2.1907 & 1.1146 & 2.6552 \end{bmatrix}$$

The covariance matrix for $\{I^{(1)}, I^{(2)}, I^{(3)}, I^{(5)}\}$ is

$$\Sigma = \begin{bmatrix} 3.8192 & 3.2413 & 1.7657 & 10.3018 \\ 3.2413 & 4.5099 & 2.1338 & 12.2610 \\ 1.7657 & 2.1338 & 1.8560 & 6.0333 \\ 10.3018 & 12.2610 & 6.0333 & 34.8238 \end{bmatrix}$$

The eigenvalues of this Σ are $\{43.3, 0.96, 0.78, 2.6 \times 10^{-15}\}$. The last eigenvalue is clearly "small" (it is zero to numerical precision). The eigenvector associated with this eigenvalue is $\begin{bmatrix} 1 & 2 & 0 & -1 \end{bmatrix}^{\mathrm{T}}$. Note that it clearly shows the linear dependence that we introduced.

- Now consider the addition of "noise". We choose $I^{(6)} = I^{(1)} + 2I^{(2)}) + 0.1I^{(4)}$. The covariance matrix is then given for $\{I^{(1)}, I^{(2)}, I^{(3)}, I^{(6)}\}$:

$$I^{(6)} = \begin{bmatrix} 1.8999 & 2.3311 & 1.3611 \\ 1.4892 & 2.4424 & 1.9360 \\ 2.2106 & 1.1591 & 2.6971 \end{bmatrix} \qquad \Sigma = \begin{bmatrix} 3.8192 & 3.2413 & 1.7657 & 10.5565 \\ 3.2413 & 4.5099 & 2.1338 & 12.5045 \\ 1.7657 & 2.1338 & 1.8560 & 6.1760 \\ 10.5565 & 12.5045 & 6.1760 & 36.3323 \end{bmatrix}$$

The eigenvalues of this Σ are $\{44.8, 0.96, 0.78, 0.001\}$. This last eigenvalue is still "small", and the eigenvector associated with this eigenvalue is $\begin{bmatrix} 1 & 1.9134 & 0.0164 & -0.952 \end{bmatrix}^{\mathrm{T}}$, Hence, even with a fairly "large" perturbation to a linear relationship, we can identify the linear relationship.

7.3.3 Principal components analysis: Application to sensor data

In section 7.3.1 we tried to ascertain whether or not some of the image data was normally distributed. We found that, using the Kolmogorov–Smirnoff test we could not conclude that six datasets (i.e., images) of the same portion of a PCB had normally distributed errors. A possible reason for this could be that each dataset was normally distributed, but that the parameters of the normal distribution varied on the different datasets. (Perhaps the settings controlling the image acquisition were unintentionally changed between images).

Principal components analysis can determine the similarity of the images acquired. We read in the six of the 971029 datasets (each had dimension 127×127) and used them as in the above example (the Matlab code is in sections D.16 and D.15.) When analyzing the data on the db scale (as collected from the sensor) the eigenvalues were $\{2 \times 10^{7}, -2 \times 10^{-9}, -7 \times 10^{-9}, 6 \times 10^{-10}, -10^{-25}, -10^{-26}\}$. When analyzing the data on a linear scale the eigenvalues were $\{2 \times 10^{3}, 10^{-13}, -7 \times 10^{-14}, 0, 0, 0\}$. In both cases, the one dominant eigenvalue indicates that there is only one fundamental image present in the data. Hence, we still do not know why the data in the 971029 datasets is not normally distributed.

7.4 Deconvolution of images

Traces on a modern PCB trace may be as small as 4 mils in width. Any conceivable electric field sensor used on PCBs will be larger than this width. Hence the electric field sensed at a single spatial location will represent an aggregate electric field extending over a region larger than the trace. However, fine pitch traces on a PCB can be imaged by using processing techniques to reduce the finite extent over which a single electric field value is returned (see, for example, Duda and Hart [14] and Russ [35, pages 213–217]).

The image processing computation is straightforward. Suppose that the actual electric field is known at a collection of spatial locations (in, say, one dimension). Create a vector from these actual electric field values and call it $\mathbf{s}$. Suppose that the electric field measured at these spatial points is given by the data vector $\mathbf{d}$. Then the finite width of the sensing element can be represented by the statement $\mathbf{d} = \mathbf{s} \star \mathbf{f}$, where the vector $\mathbf{f}$ represents the spreading of a point input (see section 2.5.6), and a star ($\star$) denotes convolution.

Knowing $\mathbf{s}$ and $\mathbf{d}$ we can determine $\mathbf{f}$ by the usual computation involving Fourier transforms $\mathcal{F}\mathbf{d} = (\mathcal{F}\mathbf{s})(\mathcal{F}\mathbf{f})$ (where $\mathcal{F}$ indicates the the Fourier transform operator, see [51]) and then $\mathbf{f} = \mathcal{F}^{-1}(\mathcal{F}\mathbf{f}) = \mathcal{F}^{-1}[(\mathcal{F}\mathbf{d})(\mathcal{F}\mathbf{s})^{-1}]$. If we define the vector $\mathbf{g}$ by $\mathbf{g} = \mathcal{F}^{-1}[(\mathcal{F}\mathbf{f})^{-1}]$ then we can also write $\mathbf{s} = \mathbf{d} \star \mathbf{g}$.

Hence, the prescription for calibration of the sensor and then using it for data collection is as follows:

- Specify a given experimental configuration (i.e., type of sensor, distance from sensor to board, etc).
- For a known vector of values of the electric field $\mathbf{d}$, find the measured signal strength $\mathbf{s}$. Using $\mathbf{d}$ and $\mathbf{s}$, compute $\mathbf{f}$ and then $\mathbf{g}$.
- Thereafter, for any new measurement $\mathbf{d}_*$ infer the underlying electric field strength from $\mathbf{s}_* = \mathbf{d}_* \star \mathbf{g}$.

Note that the technique described in this section was used on actual data in sections 6.1.2–6.1.5.

7.5 Using "log" filters to identify traces

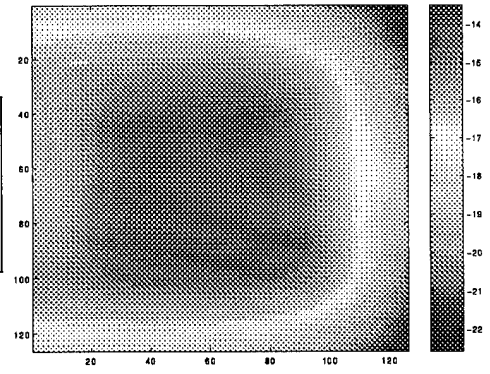
When the distance of an electric field sensor above a PCB is small, then the received magnitude of the electric field will appear multi-modal, with a small peak above each trace. Having an electric field "dip" between two traces makes recovery of the traces very simple since the concavity of the magnitude of the electric field changes (i.e., a local minimum is observed). This concavity change can be readily observed by looking at the second derivative of the received field. (Since we have no preferred orientation, and the derivative must be radially symmetric, we really compute the Laplacian).

However, since the signals are noisy, we usually prefer to smooth the data with a Gaussian filter before computing the Laplacian. (The Gaussian filter is a low-pass filter that removes the high-frequency noise.) This suggest that we use the "Laplacian of the Gaussian" filter that is common in image processing.[7]

7.5.1 Example usage of "log" filter

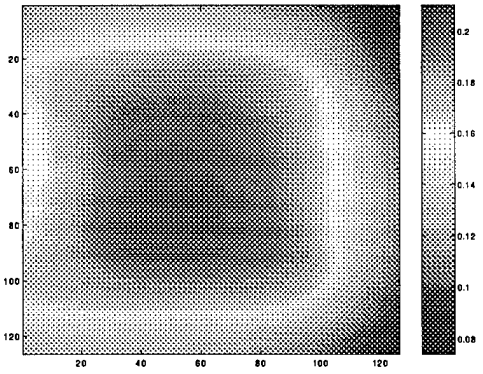
This section graphically shows the results of using a "log" filter. The demonstration uses a dataset taken from the board containing a series of loops (see section 6) using the Credence Technology sensor. To perform the computations, we used the Matlab programs `do_log.m` listed in section D.12 and `create_log_image.m` listed in section D.11. These routines used Matlab's `fspecial` command with the `log` argument. This function also allows the input of two other parameters, the number (`n`) of rows and columns used in the Gaussian filter and the standard deviation (`sigma`) of the Gaussian filter (measured in pixels). Proper selection of the parameters `n` and `alpha` requires a knowledge of the noice structure of the images.

1. The 97102908 dataset (see section 6.2), is shown here. This data is on the dB scale (i.e., the way it was originally collected by the sensor). This data was used in all processing described in this section. While there are dips between the traces, they are difficult to see.

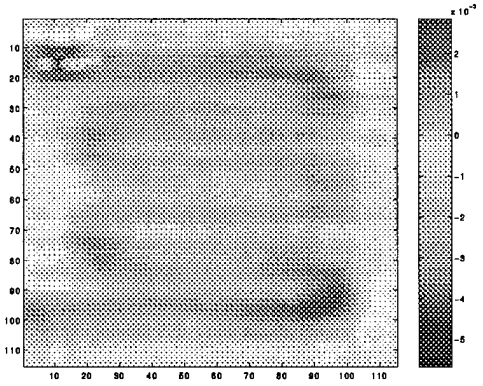


[7]See, for example, the predefined image processing filter available in Matlab using the `fspecial` command with the `log` argument.

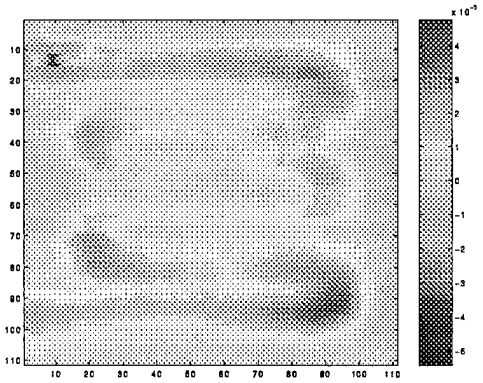
2. The same dataset as the above, but shown on a linear scale (and not a db scale).



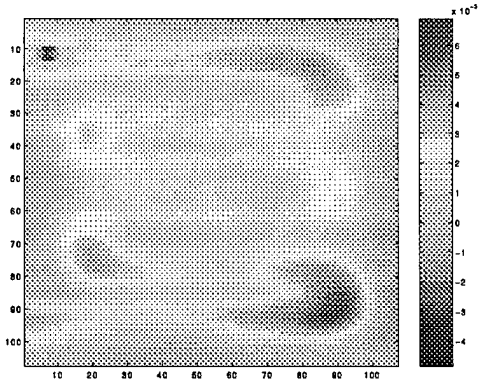
3. Using a "log" filter with `n=12` and `alpha=0.5` on the data with a linear scale. The traces are clearly visible.



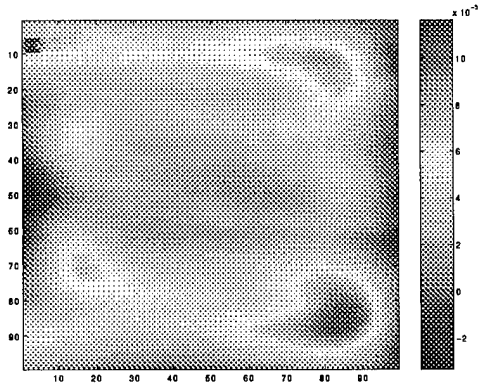
4. Using a "log" filter with `n=16` and `alpha=0.5` on the data with a linear scale. Changing `n` changes the appearance of the traces.



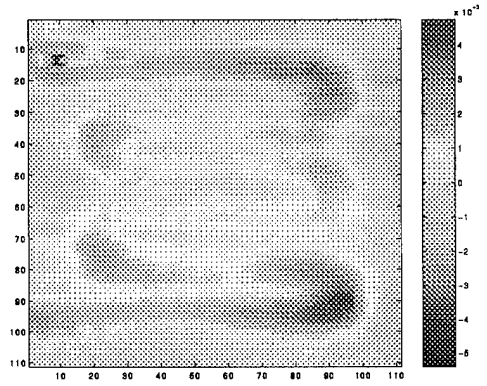
5. Using a "log" filter with `n=20` and `alpha=0.5` on the data with a linear scale.



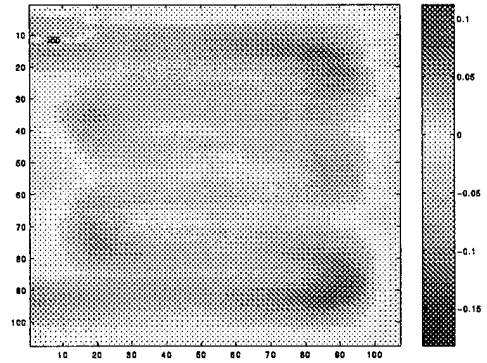
6. Using a "log" filter with **n=28** and **alpha=0.5** on the data with a linear scale.



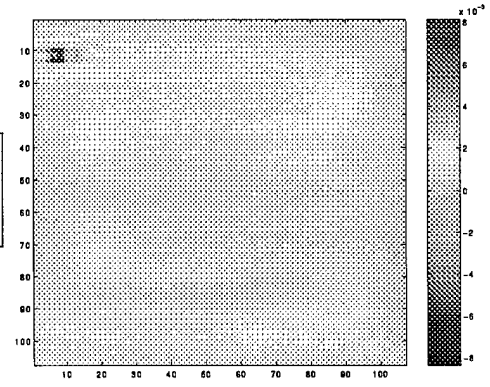
7. Using a "log" filter with **n=16** and **alpha=0.5** on the data with a linear scale.



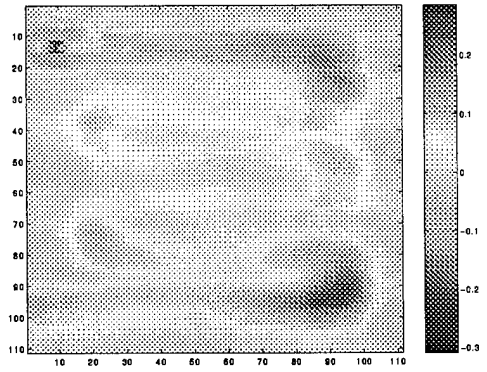
8. Using a "log" filter with **n=20** and **alpha=0.3** on the data with a linear scale. A reduction in **alpha** does not make much of a change.



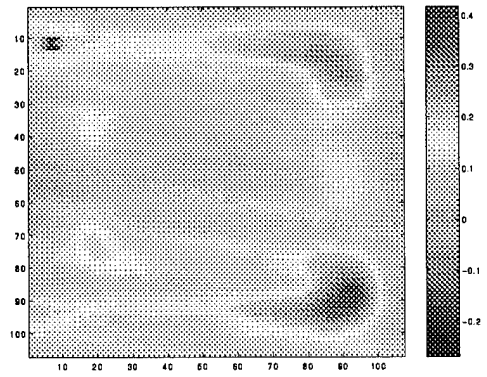
9. Using a "log" filter with **n=20** and **alpha=0.55** on the data with a linear scale. An increase in **alpha** does change the output greatly.



10. Using a "log" filter with `n=16` and `alpha=0.5` on the data with a db scale.



11. Using a "log" filter with `n=20` and `alpha=0.5` on the data with a db scale.

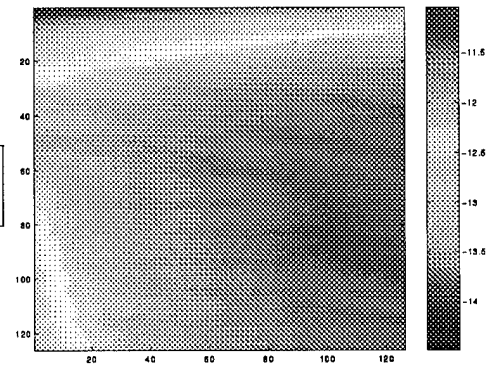


We observe that using data on a linear scale appears to give better results than using data on the db scale. In neither case, however, do the wires appear as narrow as they are in practice.

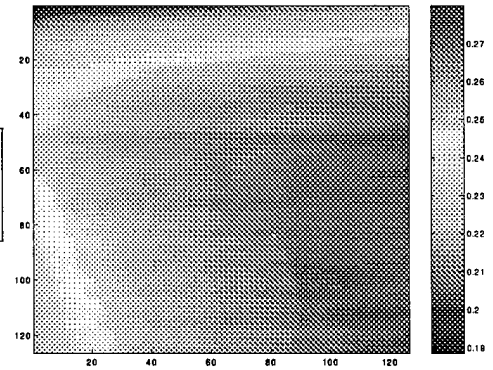
7.5.2 Log filter applied to other data

This section shows the results of applying the log filters to the data obtained from the circuit board shown in Figure 37 (note that the following images appear reversed and/or rotated). The source code is in section D.17.

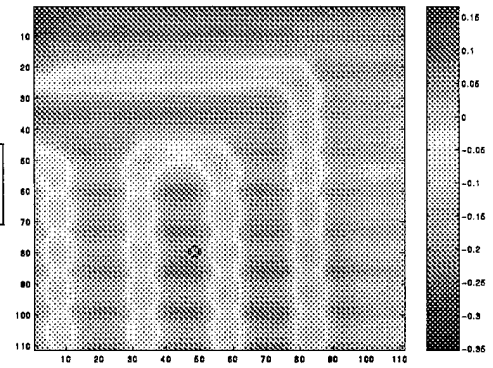
1. Circuit I: This is the raw data in file 98013002 (on a log scale).



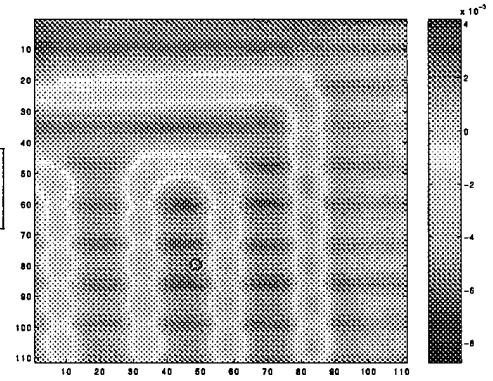
2. The same dataset as the above, but shown on a linear scale (and not a db scale). Neither image has any easily understood features.



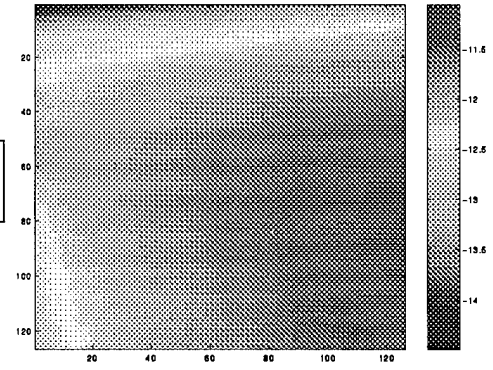
3. Using a "log" filter with `n=16` and `alpha=0.5` on the 98013002 data with a db scale.



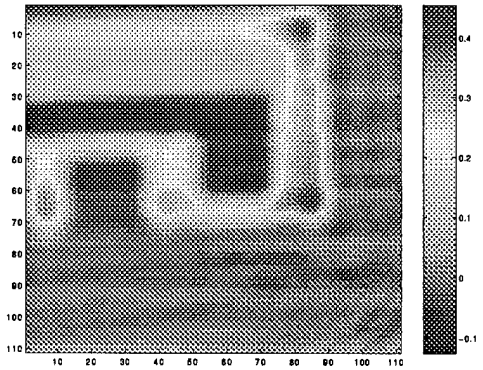
4. Using a "log" filter with `n=16` and `alpha=0.5` on the 98013002 data with a linear scale.



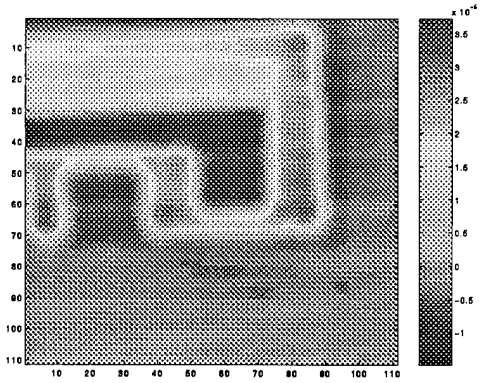
5. Circuit II: The raw data in file 98013102 (on a log scale) does not reveal any features.



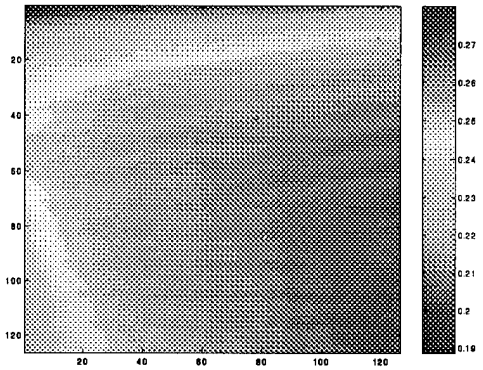
6. Using a "log" filter with `n=16` and `alpha=0.5` on the 98013102 data with a log scale brings out some of the circuit features. Note, however, that many of the individual traces are not resolved.



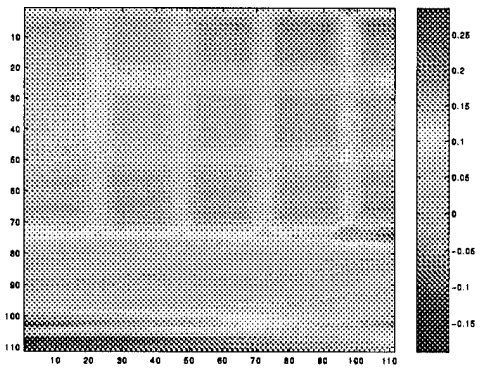
7. Using a "log" filter with `n=16` and `alpha=0.5` on the 98013102 data with a linear scale also brings out the circuit features.



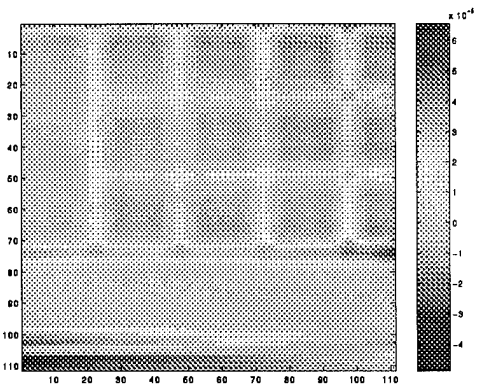
8. Circuit III: The raw data in file 98020103 (on a linear scale) does not reveal any features.



9. Using a "log" filter with `n=16` and `alpha=0.5` on the 98020103 data with a log scale brings out some of the circuit features.



10. Using a "log" filter with `n=16` and `alpha=0.5` on the 98020103 data with a linear scale also brings out the circuit features.



7.5.3 Resolution of a "log" filter

This section quantifies how far above the PCB the sensor can be and still notice the "dip" in the field between two parallel traces, .

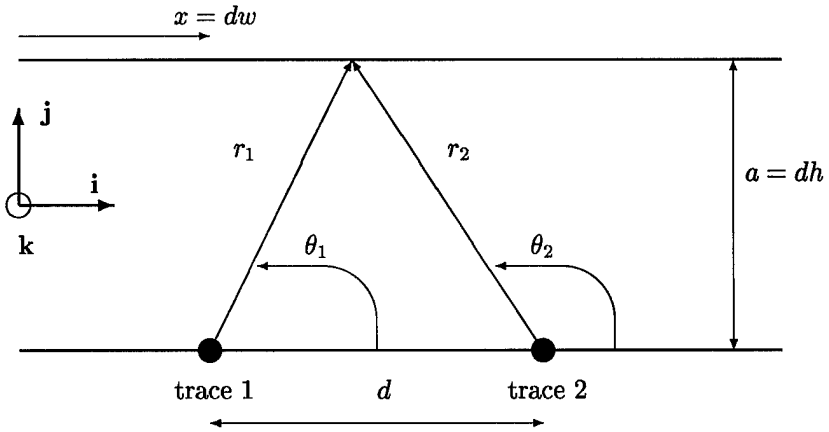
If we assume the following physical specifications:

- two parallel infinitely thin traces on a digital PCB
- the traces have one of two states, e.g., 0 Volts or 5 Volts
- the traces are separated by a distance d
- each trace carries a charge[8] per unit length of $\pm\lambda$
- with no ground plane
- with the sensor at a constant height of a from the PCB plane
- with the sensor moving transverse to the traces on the PCB.

Our conclusions indicate that this technique will not work for populated PCBs of interest to us (see page 85). It will, however, indicate useful results for the boards we have used, since they ony have traces on them (and no components).

Analysis

We use the following geometry with the origin of the coordinate system midway between the two traces:



[8]For measuring electric fields, there are only two cases: the charge per unit length is the same or it is different. For the same case, we define each trace to be carrying a charge per unit length of λ. For the different case, we can (without loss of generality) consider the charges to be $\pm\lambda$. We refer to these two cases as "common mode" and "differential mode", even though that term usually denotes different cases of magnetic fields (when currents can be in the same or opposite directions).

In order to scale all parameters by the physical distance d (i.e. the separation of the traces), we take $x = dw$ and $a = dh$.

A unit vector in the direction of a vector field that is radial from the location (x_0, y_0) at an angle of θ_0 is given by

$$(\text{direction})(x,y) = (\cos\theta_0)\mathbf{i} + (\sin\theta_0)\mathbf{j} = \frac{(x-x_0)\mathbf{i} + (y-y_0)\mathbf{j}}{\sqrt{(x-x_0)^2 + (y-y_0)^2}} = \frac{(x-x_0)\mathbf{i} + (y-y_0)\mathbf{j}}{r}$$

Hence, the vector electric field due to trace 1 (with charge per unit length of λ) is given by

$$\mathbf{E}_1(\mathbf{x}_1) = \mathbf{E}_1(w,h) = \frac{\lambda}{r_1^2}(\text{electric field direction}) = \frac{\lambda}{r_1^2}\left(\frac{x_1\mathbf{i} + y_1\mathbf{j}}{r_1}\right)$$

Using $x_1 = d\left(w + \frac{1}{2}\right)$, $y_1 = a = dh$, and $r_1 = \sqrt{x_1^2 + y_1^2}$ we can readily compute the electric field $\mathbf{E}_1$.

For trace number two, the electric field is given by

$$\mathbf{E}_2(\mathbf{x}_2) = \frac{\pm\lambda}{r_2^2}\left(\frac{x_2\mathbf{i} + y_2\mathbf{j}}{r_2}\right)$$

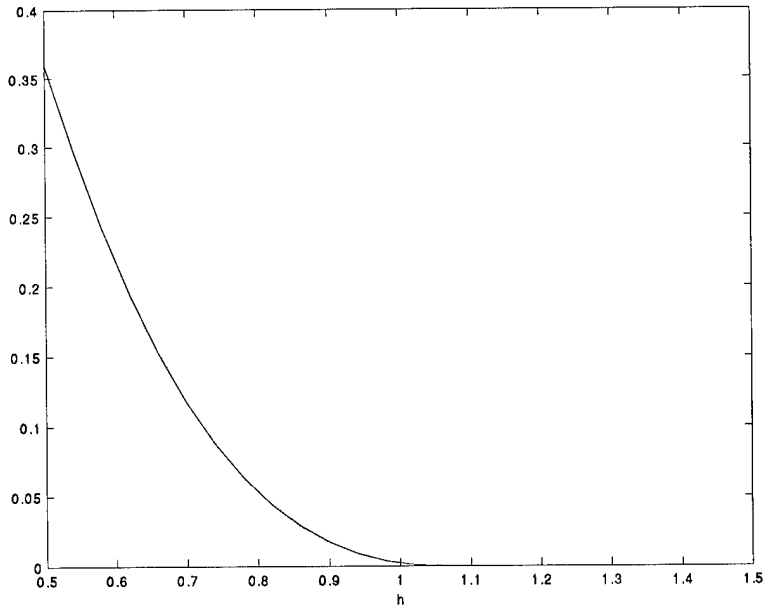
depending on whether the traces are in "differential mode" $(-\lambda)$ or "common mode" $(+\lambda)$, with $x_2 = d\left(w - \frac{1}{2}\right)$, $y_2 = a = dh$, and $r_2 = \sqrt{x_2^2 + y_2^2}$. The scalar electric field, which is measured in our present antenna-based system, is given by $|\mathbf{E}| = \sqrt{|\mathbf{E}_1 + \mathbf{E}_2|^2}$.

The Matlab code in section D.13 was used to create the plots in this section.

"Common Mode"

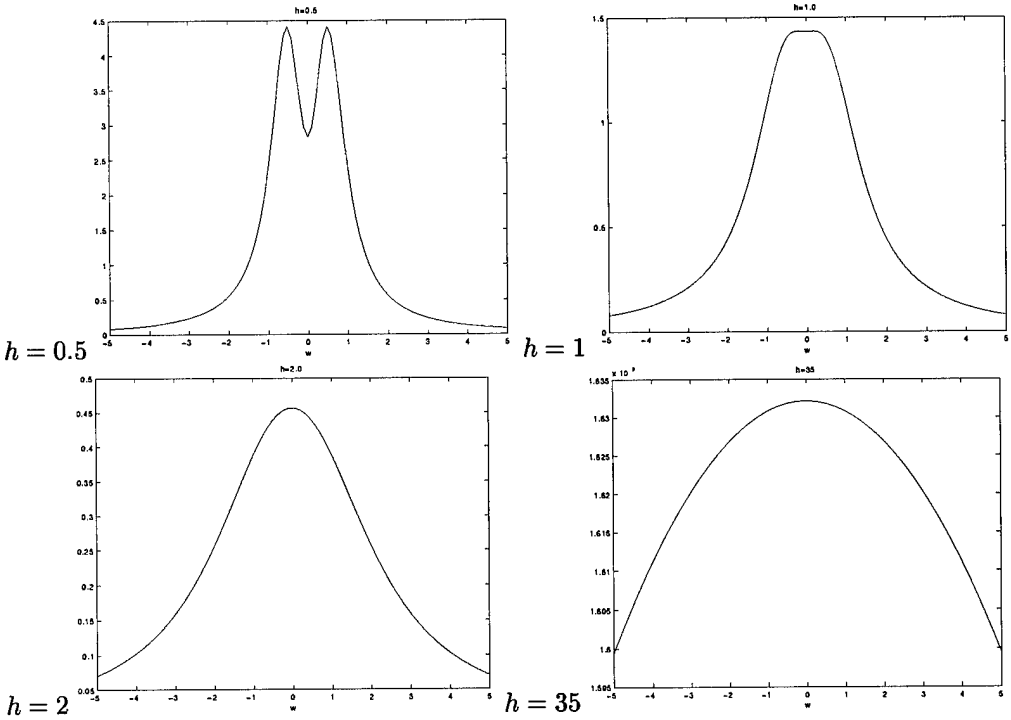
In "common mode," each trace carries a charge per unit length of λ.

The relative size of the dip, as a function of h, is given in the following figure:



In this case, there is a cutoff when h is about 1.06 times d. Above this value we are "far above" the PCB and the two wires appear as a single wire and there is no "dip".

For different values of h, we obtain the following plots of $|\mathbf{E}|$ versus w:

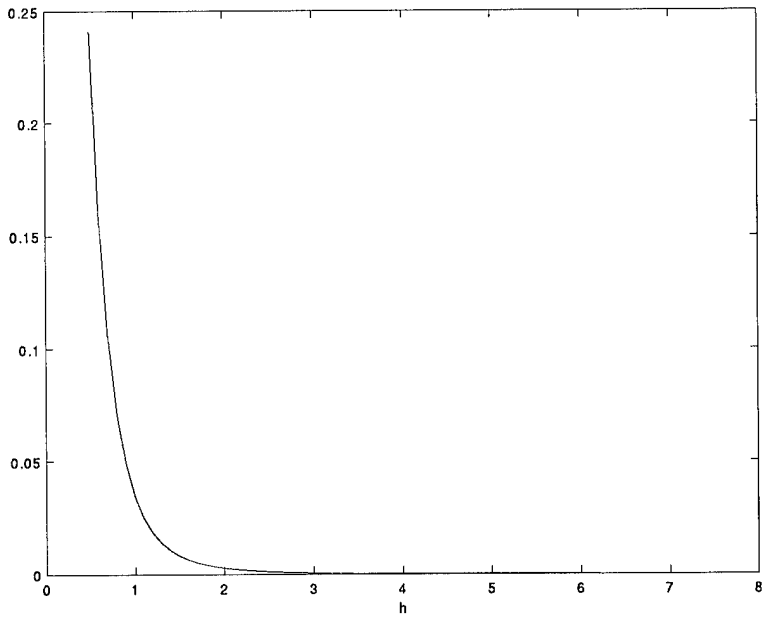


$h = 0.5$ $h = 1$

$h = 2$ $h = 35$

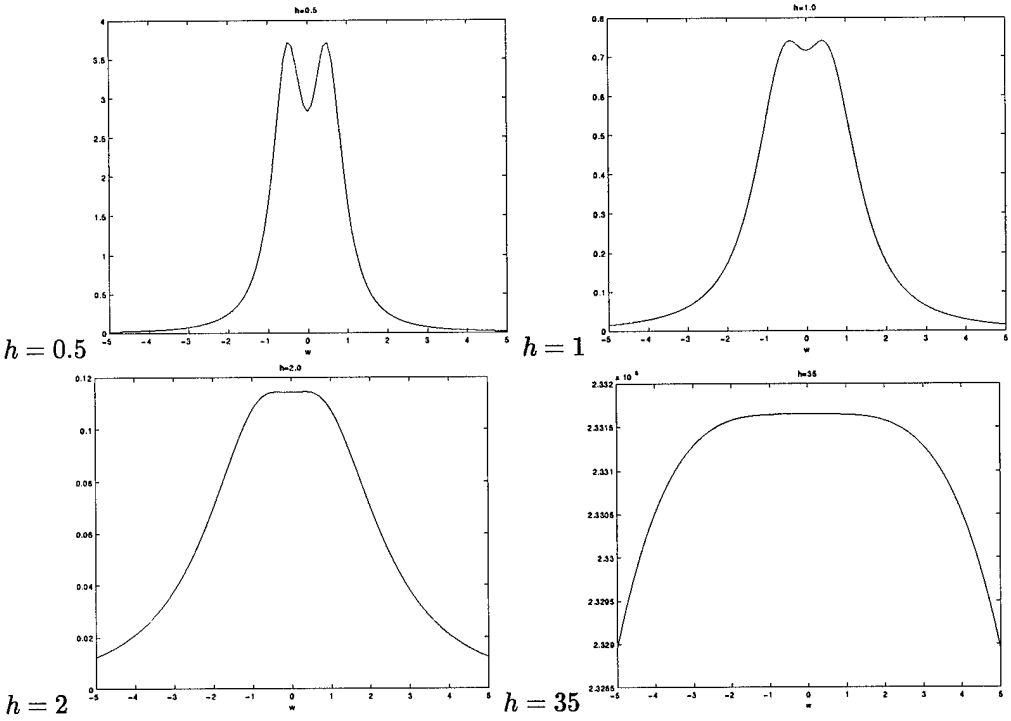
"Differential Mode"

In "differential mode," the two traces carry charges per unit length of λ and $-\lambda$. The relative size of the dip, as a function of h, is given by



In this case, there is a practical cutoff when h is about 3 times d (it is zero to 4 decimal places when h is about 5.6 times d). Above this value we are "far above" the PCB and the two wires appear as a single wire and there is no "dip".

For different values of h, we obtain the following plots of $|\mathbf{E}|$ versus w:



$h = 0.5$ $h = 1$

$h = 2$ $h = 35$

Conclusions

The common technique of using a "log" filter on an image to recover image features[9] will not work for PCBs unless the sensor is *very close* to the board. For common and ordinary digital PCBs (say boards with a trace separation of 10 mil) this results in a required sensor height of 30–50 mils, which is unrealistically close to a populated PCB.[10] The antenna we are presently using has a height above the board of 2 mm (about 80 mils).

We conclude that looking at image concavity changes is not going to be practically useful for feature extraction. It appears that feature information may be extracted, however, from the "width" of the effective measured electric field.

[9]Note that the features we are interested in are

- the state of the wire (i.e., $+\lambda$ or $-\lambda$); or
- the position of the wire; or
- both of the above

[10]Note that for an unpopulated board the distance from the sensor to the board can be arbitrarily close; the actual distance is based on board warpage, steadiness of the robot moving the sensor, etc.

8 Circuit board study

This section summarizes the state of the art for circuit board characteristics, from design, manufacturing, and testing points of view. If the sensor system that was developed in this project is to be useful it must be at least as useful as present testing techniques, and it must continue to be useful as manufacturing methods evolve.

There are many different types of circuit boards and there are many parameters that characterize the technical sophistication of circuit boards. Some of these quantities are:

1. Whether the board is digital, analog, or mixed
2. Maximum voltage on the board
3. Operating frequency
4. Size of pads
5. Size of traces
6. Whether the PCB has a single layer or is multilayer
7. Whether a ground plane is present
8. Vias: whether or not present, blind and buried, size
9. Types of components: surface mount, ball grid arrays, flip chips, etc
10. Use of differential mode (instead of common mode)
11. Density of components
12. Physical size of PCB
13. Material thickness
14. Communications protocol
15. Connections to components: wire wrap, solder
16. The types of connectors used on the PCB
17. Whether built-in-testing is present

The parameters that most directly affect the technology described in this report are numbers 1–11 above.

The conclusions of this section are that:

- Feature scale will continue to be reduced
- Small feature size presently prevents many types of existing boards from being tested by the most common testing techniques (bed-of-nails and flying probes).
- Designers are continually making the radiated electromagnetic fields smaller.

8.1 Designing PCBs: State of the art and beyond

Current PCB design capability can be ascertained by determining the parameters in PCB design software. From the information in section 8.2 we see that it is not difficult to design PCBs with the following characteristics:

- Extremely small resolution (down to 1/1000 mil or .000001 inch)
- Large physical size (up to 144″ × 144″)
- Many layers (up to 1,024)
- Many types of vias (up to 250)
- Many components per board (up to 32,000)
- Many pins per components (up to 3,200)
- Any number of pad and drill diameters

Comparing these numbers with the present and future manufacturing capabilities in Table 3 (see page 90) one sees that the present generation of design software well exceeds manufacturing capability any time in the near future.

8.2 Designing PCBs: A sampling of vendors

The Electronic Design Automation Consoritum[11] (EDAC) serves the EDA community as an information clearinghouse and a central repository for EDA news and information. More than 70 companies participate

[11]Located in San Jose, CA; 408-287–3322; `www.edac.org`.

	Layers	Vias	Space/Trace Width	Copper Wire Length
Dansk Data Elektronik	2	3,806	0.0065"/0.0060"	3,376.9"
EDIF	4	2,158	0.0060"/0.0060"	(not given)
OrCAD	2	2,410	0.0065"/0.0060"	3,458.2"
US CAD Software	2	1,900	0.0065"/0.0060"	2,700"
VeriBest	2	2,227	0.0065"/0.0060"	2,975"
Auken–Redac	2	3,065	0.0065"/0.0060"	3,316"

Table 2: Comparison of PCB design software (from Hamburg [18])

in EDAC. A tabular comparison of 51 different design tools (from 34 companies) is in Clarkson [11]. His table characterizes 17 properties of each package, such as whether or not it has interactive routing and editing.

An interesting comparison of design tools is in [18]. In this comparison the electronic layout of a PCB was specified and different vendors were invited to use their tools to design the actual board. The results are in Table 2.

The following is a sampling of manufacturers of PCB design software.

8.2.1 CadSoft Computer

CadSoft Computer[12] produces a layout program called "Eagle". It has the following capabilities:

- Maximum drawing area 1.6 × 1.6m (64″ × 64″)
- Resolution 1/10,000 mm (0.1 micron)
- Width can be adjusted separately for each wire
- Any number of pad and drill diameters
- Up to 255 layers (multilayer)
- Conventional and SMT parts
- Round, square, oblong and octagon shaped pads
- Parts can be placed on both sides of the board
- Up to 99 sheets in one schematic

8.2.2 Douglas Electronics

Douglas Electronics[13] produces a layout program and they also manufactures the boards that are designed using it.

> Professional Layout is a full-featured printed circuit board CAD/CAM program capable of designs up to 32" x 32".... Completely user-definable pads and parts, 50 different view sizes, and 0.001" resolution make both initial layout and subsequent editing quick and easy.

Their AutoRouter "is capable of accommodating large, complex designs up to 16 layers. All layers are routed simultaneously, ... " They state:

> The standard PCB order is solder plated over copper on 0.062" FR-4 material, but SMOBC and/or exotic board materials and thicknesses are also available. For the best price on on board orders, a minimum hole size of 0.020" and a minimum line width/spacing of 0.008" is required, however a minimum hole size of 0.012" and a minimum line width/spacing of 0.006" is available at additional cost.

[12]Located at 801 South Federal Highway; Delray Beach, FL 33483; 1-800-858-8355; `http://www.cadsoftusa.com`.

[13]Located at 2777 Alvarado St.; San Leandro, CA 94577; 510-483–8770; `http://www.douglas.com/DEhome.html`.

8.2.3 Mentor Graphics

Mentor Graphics[14] produces a layout program called *Board Station.* It has the following capabilities:

- 100" × 100" maximum board size
- Database resolution of 0.01 micrometers
- 1,024 maximum layers including signals, power, and silkscreen
- 32,000 maximum component count per board
- No pin count limit per board
- 3,200 pin count limit per component
- 32,000 maximum nets per board
- No limit on number of pins per net
- No limit to number of vias per board

8.2.4 OrCAD

OrCAD[15] sells several different PCB layout tools. However, all of these tools have the following capabilities:

- 144″ × 144″ maximum board size
- 30 total layers
- 16 simultaneous routing layers
- 7,500 components per board
- 10,000 nets per board
- 32,000 connections per board
- 16,000 connections per net
- 3,200 pins per component
- 1,000 different pad stacks
- 250 characters per net name
- 1/60 mil or 1 micrometer base resolution
- 1/60 (1 minute) degree component rotation

8.2.5 Protel

Protel[16] produces a layout program called *Advanced PCB 98.* It has the following capabilities:

- 16 signal layers, 4 power planes, and 4 mechanical layers.
- 32 bit design database, providing a resolution of 0.001 mil in a 100″ × 100″ workspace.
- Dimensions (for all entities) and grids to .000001 inch accuracy
- Angular resolution is to .001 degree for either arcs or circular arrays.

8.2.6 Wise Software Solutions, Inc

Wise[17] sells a layout tool called *GerbTool.* It has the following capabilities:

- Accurate to 1/100 mil (.00001 in)
- Can view up to 999 layers simultaneously.
- Can handle over 4000 apertures in up to 999 aperture lists.

[14]Located at 8005 SW Boeckman Road; Wilsonville, Oregon 97070-7777; 503-685–7000; `http://www.mentorg.com`.

[15]Located at 9300 S.W. Nimbus Avenue; Beaverton, Oregon 97008; 503-671–9500; `http://www.orcad.com`.

[16]Located at Level 3, 12a Rodborough Rd; Frenchs Forest NSW 2086; Australia; 800-544–4186; `http://www.protel.com/DEhome.html`.

[17]Located at 8285 SW Nimbus Ave., Suite 191; Beaverton, Oregon 97008; 503-626–7800; `http://www.gerbtool.com`.

8.3 Manufacturing PCBs: State of the art and beyond

The density of conventional boards today ranges from 30–40 pads/sq. in. for commodity PCBs to about 140 pads/sq. in. for leadng edge PCBs with lind and buried vias. The Sony camcorder board has a double-sided density of over 600 pads/sq. in., with small blind vias in numbers proportional to the pads.

There are three types of PCMCIA cards: Type I (3.3 mm thick) — commonly used for memory expansion; Type II (5.0 mm thick) — commonly used for fax/modem cards; and Type III (10.5 mm thick) — commonly used for storage applications. To effectively fit PCMCIA slots, PCBs are designed in the following thickness ranges (this is the total thickness including all layers and coatings):

- 4-layer: 0.012–0.018"
- 6-layer: 0.018–0.025"
- 8-layer: 0.025–0.033"

Dixon [13] provides the following current design recommendations on land/line sizes:

Lines per 50 mil grid	Land size (mils)	Hole size (mils)	Line/space
1	32	16	0.006"/0.006"
2	25	12	0.005"/0.005"
3	25	12	0.0035"/0.0035"

Nakahara [30] has

> "Japanese fabricators' yield for 0.0004" PCBs is typically in the low to mid 90% range. One Japanese fabricator is getting 95% first pass yield for 4-layer MLBs with 0.004" lines on outer layers ... "

Nakahara [30] also gave the following table of difficulty associated with small pitch sizes

Degree of difficulty related to line width				
Line width, inches	0.008"	0.006"	0.005"	0.004"
Degree of difficulty	1	3	6	12

In O'Shea [33] it is stated

> "Common via hole sizes in production are 0.010" dia while a 0.006"-dia hole now is used for prototype runs."

In Kuzawinski and Blackwell [22] a plastic ball grid array (PBGA) with 0.7 mil lines and 1.1 mil spaces (for a 1.8 mil pitch) is described.

Sterling [40] quotes a 1996 IPC study which states that, of the ESM market, 74% of the components placed were surface mount parts, 4.6% were fine pitch parts, and chip-mounted packages comprised less than 0.5%. This same study had smaller companies reporting 31% through-hole components placement, while larger companies only placed 13% of the component in this manner.

Buetow [8] contains the information in Table 3. In this table, "leading edge" means a technology practiced by 10% of the industry and "state of the art" means a technology practiced by 1% of the industry.

The information in Table 4 is from a 1994 workshop that was quoted by Vaucher [45].

8.4 Manufacturing PCBs: A sampling of manufacturers

Electronic Buyers' News (EBN) maintains a list of electronic manufacturing services providers in North America (see `http://techweb.cmp.com/ebn/942/`). Their third annual directory lists of 175 manufacturers, ranging in size from $5 million service providers with single facilities, to billion-dollar companies with plants all over the world. They made every effort to include all the major providers—nearly every contractor with annual revenue of $100 million or more, for instance, is in the directory.

An idea of the market size of PCB bare board production is given by Nakahara [31]. He indicates that the largest bare board manufacturer, CMK, had revenues of $1,270 million in 1997. There are 67 PCB manufacturing companies with revenues of more than $100 million each. They account for $15,927 million

Attribute	Current (1997–1998)		Near Term (1999–2001)		Long Term (2002–2007)	
	Leading edge	State of the art	Leading edge	State of the art	Leading edge	State of the art
Line width	0.004"	0.003"	0.003"	0.002"	0.003"	0.002"
Space (minimum)	0.005"	0.004"	0.004"	0.003"	0.004"	0.003"
Plated hole diameter (minimum)	0.008"	0.006"	0.008"	0.006"	0.006"	0.006"
Microvias	none	0.004"	0.005"	0.003"	0.004"	0.002"
Range of layer count	4–28		6–24		2–18	

Table 3: Key PCB manufacturing parameters (from Buetow [8])

Chip Level	1992	1995	1998	2001	2004	2007	Change/year
Feature size	0.5	0.35	0.25	0.18	0.12	0.1	90%
Defect density (defects/cm^2)	0.1	0.05	0.03	0.01	0.004	0.002	77%
Number of IC levels for logic	3	4.5	5	5.5	6	6.5	190%
Performance (Mhz) – off chip	60	100	175	250	350	500	120%
Performance (Mhz) – on chip	120	200	350	500	700	1000	120%
Number of I/Os	500	750	1500	2000	3500	5000	120%

Table 4: PCB technology trends (from Vaucher [45])

of the total of $16,300 million spent annually on PCB production. The printed wiring board market is a larger market. The US share is about $7,200 million (see [12]).

The electronics manufacturing services (EMS) industry is, of course, a much larger market. The $14,000 million US EMS industry is highly fragmented, with over 1,000 firms participating (see Sterling [40]). In 1996 the top 25–30 companies controlled nearly 70% of this market.

The following is a sampling of PCB manufacturers.

8.4.1 Capital Electro-Circuits Inc.

Capital Electro-Circuits Inc.[18] is a contract PCB manufacturer with the following manufacturing capabilities (all dimensions are in inches):

- Material Thicknesses: 0.005 – 0.200
- Largest Panel Size:
  - Single / Double-Sided: 18 × 24
  - Multilayer: 18 × 24
- Maximum Number of Layers: 16
- Copper Thickness: 0.5 oz. minimum to 4 oz. maximum
- Minimum Conductor Width and Spacing: 0.005
- Smallest Hole Size: 0.010
- Plated-Thru Hole Tolerance: ±0.003
- Dimensional Tolerance: ±0.005
- Conductor Spacing to Edge of Board (Min. A/W): 00.010
- Solder mask Registration:
  - Screen Liquid Mask (Min.): 0.010
  - Photoimageable (Min.): 0.005

[18]Located at 7845-J Airpark Road; Gaithersburg, Maryland 20879. 301-977-0303, `http://www.capitalelectro.com`

Additionally, they give the following tips

- Hole Size and Pad Size:

  For power and ground layers, make the pad size 0.045 in larger than the hole size. For component, solder and inner layers make the pad size 0.017 in larger than the hole size. It allows more room for drilling and decreases the chances for shorts. Solder mask pads should be 0.005 in. larger than the soldering pad

- Silkscreen rules:

  Keep silkscreen at least 0.006 in away from pads (except vias). The minimum line thickness is 0.008 in and the minimum letter height is 0.100 in. Ensure that no silkscreen falls on any pads this could effect the soldering process. Avoid printing component outlines, especially if you are using automatic assembly.

- Routing:

  Hole sizes smaller than 0.020, especially in boards thicker than 0.062 boards, limits the number of panels which can be drilled at the same time, therefore, adding cost to the drilling operation and high aspect ratios which makes plating difficult. Also, limit the number of different hole sizes. Review your design to see what flexibility you have with the hole sizes.

- Line width and spacing:

  Manufacturing cost increases significantly when line width and spacing drop below 10 mil, so you should avoid them if your application does not require such fine line and space.

- Electrical Testing:

  The complexity involved in making the test fixture depends on factors such as; type of board (thru hole or SMT one side or SMT both side), number of test points, spacing between pads (pitch), therefore, the cost of test fixture can vary from $200 to over $1000. Also, testing would add 1-2 days to manufacturing cycle. So consider the cost and time factors when deciding if you want Electrical Testing or not.

8.4.2 Fineline Circuits Limited

Fineline Circuits Limited[19] is a contract PCB manufacturer with the following capabilities

- For flexible PWBs, number of layers can be 1–12
- For rigid PWBs, number of layers can be 2–22

8.4.3 RD Circuits

RD Circuits[20] is a contract PCB manufacturer with the following manufacturing capabilities

- 1 mil trace and space
- 4 mil holes
- 2 mil dielectric
- 20:1 aspect ratio
- Blind and buried vias to ± 1 mil
- Overall thickness to ± 2 mil

[19]Located at 6530 Lawrence Avenue East; Scarborough, Ontario, Canada M1C 4A7; 416-283–9560; `http://www.finelinepwb.com/`.

[20]Located at 9 Olsen Avenue; Edison, NJ 08820; 732-549–4554; `http://www.rdcircuits.com`.

8.5 Testing PCBs

8.5.1 Types of errors

There are many types of errors that can occur in PCBs. PCB errors can be categorized into two major classes; failures intrinsic to a board (process faults), and failures that develop after a board has been manufactured (functional faults). Process faults are composed of two categories: component insertion errors and solder faults.

- Component insertion errors can include wrong, missing, and mis-inserted components. Normally, these faults constitute 25–50% of all process problems on a board. Automatic insertion equipment can reduce this percentage to a range of 10–25%.
- Solder faults typically include board opens and shorts. The fault distribution can range from 35–65% of all process faults.

Manufacturing-induced faults typically account for over 80% of all board failures. The term "First Pass Yield" (FPY) is the percentage of boards that pass subassembly test, with no rework required. High FPY will exceed 90%, medium FPY will be in the range 0–75%, and low FPY is less than 40%.

Performance, or functional, faults, include race conditions caused by crowded board designs, component incompatibility, and workmanship problems. Traditionally, this category has totaled less than 20% of all board faults.

Failures in PCB construction include:

1. shorts and opens
2. wrong polarity
3. wrong part
4. missing components
5. SMD open pads
6. functional errors
7. bare board errors
8. discrete component errors

Failures that occur after construction include:

1. delamination
2. abraded traces
3. timing problems

Functional faults usually manifest themselves through improper operation (either continuous of intermittent).

Teska [42] contains a table of the distribution of process faults:

Fault Spectrum						
Shorts	Opens	Missing Components	Insufficient Solder	Misaligned Components	Defective Components	Other
40%	13%	11%	9%	8%	7%	12%

MacLean [26] gives a table with somewhat different values:

Fault Spectrum				
Shorts	Opens	Missing Components	Bad/Wrong Part	Damaged Part
22%	48%	19%	7%	4%

8.5.2 Testing strategies

There are many different test techniques used for PCBs. Some of the techniques for populated PCBs are:

1. **Boundary scan testing**

   Boundary scan incorporates the use of special components which are linked together to allow access to the internal nodes of a PCB without the need to make physical contact through a bed of nails fixture.

2. **Combinatorial testers**

   Combinatorial testers (CT) are combinations of in-circuit testers and functional board testers.

3. **Functional board testers**

   Functional board testers (FBT) represent the first board level ATE approach. Functioning as a programmable hot-bed of the actual system in which the board will reside, functional testers test the board as a whole. Typically, functional test fault coverage can reach 98% of the faults.

   There are two major approaches to functional testing. The older approach, which uses edge-connector testers, drive and sense signals to the board through its edge connector. The newer approach is to use microprocessors or boot ROM emulation to drive signals into the board.

4. **In-circuit testers**

   In-circuit testers (ICT) have all the capabilities of MDA systems, plus the ability to power and test active components. Because the test is performed with power to the board, specific test vectors must be applied. ICT can diagnosis most faults, missing only performance, or functional, failures.

   Note that newer high-speed VLSI devices, such as ASICs, and hybrid circuits can have problems with the "backdriving" techniques used to isolate components for in-circuit testing.

5. **Manufacturing defects analyzers**

   Manufacturing Defects Analyzers (MDA) are the most basic approach to board ATE. They interface to the device under test (DUT) through a bed of nails fixture. MDA systems perform an un-powered test to verify these parameters within a board

   - solder shorts
   - board opens
   - wrong components
   - missing components
   - mis-inserted components

   These process faults are traditionally the greatest area of board problems. Because the test runs without power being applied to the board, MDA systems cannot normally detect faulty active components.

MacLean [26] has a comparative analysis of test processes, see Table 5 (note that his "Vectorless" is the same as capacitive probing, see page 95).

For testing substrates (i.e., bare boards) there are three standard testing methods, listed below. A comparison of these technologies with a non-contact testing technique (capacitive probing) is in Table 6 (taken from Hague and Kaida [17]). A testing technique still under development is the "RF resonator method", see [41].

1. **Moving probe**

   Flying probe testers are available with very fine accuracies, optical positions systems and high probing rates. These have a relatively high unit test cost since throughput is limited by the need to mechanically probe each of the lands and bond pads. A concern with moving probe testers is that the probe should not make any indentations on the soft gold that would impair subsequent bonding.

   From [2]

Process	Cost	Program Development	Test Speed	Faults Detected	Faults Missed
ICT	low	low	very fast	opens, shorts, missing bypass, tolerance, function	hi-speed parametric
MDA	low	low	very fast	opens, shorts, tolerance	missing bypass, function
Vectorless	low	low	fast	opens	shorts, toelrance, function
Vision	high	minimal	slow	opens, shorts on visible parts	tolerance, function, unviewable connections
X-ray	high	low	slow	solder opens, shorts, alilghment, outline	tolerance, function
Boundary scan	low	minimal	medium/fast	opens, shorts on boundary cells, internal	tolerance, function, parts without boundary scan
Functional	medium	low to very expensive	fast; except for diagnostics	speecified test parameters	component parametrics not evident in the circuit

Table 5: Comparison of test methods for populated PCBs (from MacLean [26])

	Moving probe	Direct Probe	Shorting Rubber	Non-contact
Speed	slow	medium	fast	fast
Fixture cost	low	high	medium	medium
Positional tolerance	high	low	high	high
Test accuracy	high	low	medium	high
Contamination	no	no	possible	no

Table 6: Comparison of test methods for semiconductor substrates (from Hague and Kaida [17])

> "The flying probe testers are very accurate in their positioning accuracy. They can hit pads of 1 mil square or less. This means that for very fine pitches like multichip module substrates, the flying probe is the only way to achieve short and open testing on 4, 6 and 8 mil vias and pitches at 10 mils or less. They have the ability to provide 100 percent nodal visibility on high-density PCBs containing ball grid arrays, chip scale packages, TAB, and known-good die."

2. **Shorting rubber**

   In this technique, shorting rubber is used to short out the bond pads, and spring probes are used to access the lands. In this way, continuity from the land to the pad can be measured. The shorting is then removed and the circuits are tested for isolation conventionally. Drawbacks include: The shorting rubber wears out, the substrate must be extremely clean, and there is the potential of silicon oil leaking onto the bond pads.

3. **Direct probing**

   Typically, a spring probe fixture can be manufactured to contact both the lands (typically 0.020–0.030" diameter) and the bond pads (down to 0.0006" pitch). Such fixtures require a CCD alignment system

	In-Circuit Testers	Manufacturing Defect Analyzers	Flying Probe Testers	Solder-Paste Inspection	Automatic Optical Inspection	X-ray
Test Development Time	2–8 weeks	3–4 days	2–4 days	1 week	1–2 weeks	< 1 week
Fixturing required	$5,000–$15,000	$3,000–$10,000	none	none	none	none
Relative throughput	1	1	6–12 times slower	3 times slower	3 times slower	3 times slower
Purchase price	$150,000–$500,000	$50,000	$180,000–$300,000	$100,000–$300,000	$200,000–$300,000	$300,000–$450,000

Table 7: Time and cost comparisons for several PCB testing techniques

to maintain precise alignment. A pressure control system is also required to prevent gouging of the bond pads.

A comparatively new approach for testing boards is the use of non-contact testing. The following are a sampling of non-contact technologies (note that several non-contact testing patents were described in [49]):

1. **Capacitive probing**

   Capacitive opens detection techniques operate by typically applying an AC stimulus to a pin on the device under test while simultaneously sensing the response signal on a plate located on top of the IC. These methods can have difficulty in testing certain packaging types such as chip-on-board and multichip modules. Commercial products that use capacitive probing include GenRad's *Opens Xpress* and Hewlett–Packard's *Testjet.*

2. **Ultrasound inspection**

   Ultrasound techniques can detect cracks, disbonds, delaminations and voids, see [5]. Sonoscan[21] makes an a device for inspection of IC package reliability

3. **X-ray inspection**

   Automated X-ray inspection (AXI) is a non-destructive, in-line solution for inspecting limited-test-access board assemblies. Cross-sectional AXI does a very good job of finding many defects related to solder joints or component placement. AXI can handle dense, fine-pitch boards regardless of node access (even flip chips), detecting not only defective joints but also marginal solder joints, see [23] and [28].

   Bonham and Singbeil [7] argue that AXI and in-circuit testers (ICT) are complementary test strategies. Their article includes a table of faults covered only by X-ray, faults covered only by ICT, and faults covered by both AXI and ICT.

   Hewlett–Packard's automated X-ray inspection of joints can analyze 30 joints/sec.

Oresjo [32] has a rough comparison of the time and cost required to implement several of the above test techniques. His information is summarized in Table 7.

8.5.3 Testing PCBs: State of the art

Concerning a capacitive non-contact tester, Hague and Kaida [17] state:

[21]Located at 530 E. Green St.; Bensenville,IL 60106 ; 630-766–7088
`http://www.astm.org/labs/PAGES/098250.htm`.

> "Non-contact testing technology is capable of detecting faults on circuits as small as 0.001" line width with 0.001" spaces. a test capability which for the moment exceeds the fabrication capability of virtually all PCB fabricators."

Jacob [21] quotes the president of Advanced Probing Systems (APS) as saying that the diameters of probes is decreasing steadily. While five years ago 80% of APS's sales were for probes 10 mil or larger in diameter, this year 50% of sales will be for probes with a diameter of 5 to 8 mils.

Interconnect Devices[22] sells probes for use in bed-of-nails fixtures. Their standard probes (sizes SS30 to Size 5) can be used on 39 to 187 mil centers. Their Microseries probes (model Penta 0) can be used on .010" centers.

An article by Teska [43] states

> "Flip chip pitches often range from 10 to 15 mils, presenting quite a challenge to get a trace out of a pad. Compare that to a board with a typical BGA, where the pitch is usually 5 mils; its trace routing is not so challenging. At a 20-mil pitch, routing traces can be difficult. Traces as close as 10–15 mils require a high technology PCB, including special plating of the board to ensure planarity of the substrate, extremely precise registration, and a "density patch" (a means to handle routing from the very dense chip to the less-dense board)."
>
> ⋮
>
> Automated X-ray inspection is capable of inspecting pitches associated with flip chips down to an 8 mil pitch and 4 mil feature size. It can detect misalignment, shorts, and opens that are the result of insufficient solder without requiring electrical access to the the board."

As silicon technology progresses, increasing demands are being placed on the interconnection technology. A component is typically placed in a chip carrier (this is the first level packaging) and then these are placed on printed wiring boards (this is the second level packaging). Ball grid arrays (BGS) are increasingly becoming the solution of choice to meet the requirements at the PCB interface.

8.5.4 Reducing EMI

The technology described in this report requires a sensor to detect the near-field electromagnetic radiation of PCBs. Very little data concerning the near-field electromagnetic radiation of PCBs exists, primarily because there has historically not been interest in this information. Recent desire to base prediction models of the health of electronic equipment on radiated electric fields may alter this situation.

However, there has been a significant study of the behavior of high frequency electromagnetic fields far from a PCB for the purpose of obeying government regulations of EMC (electromagnetic compatibility) and EMI (electromagnetic interference), see [6]. The goal of these regulations is to minimize the electromagnetic radiation produced by electronic devices. With the increasing complexity of PCBs and increases in computer clock rates, the importance of EMI analysis to detect EMC problems has risen dramatically (see Lipman [25]). There are several products on the market that predict EMI performance from PCB schematics (allowing for design changes and reduced emissions). These products include (for a review of products, see Vollmer [46]):

- Applied Simulation Technologies[23] has *ApsimRADIA*. This product can predict both differential and common mode radiated emissions, see [37].
- Incases[24] has *EMC–Workbench.* which consists of a suite integrated tools for the analysis of EMC problems on printed circuit boards.

[22]Located at 5101 Richland Avenue; Kansas City, Kansas 66106; 913-342–5544; `http://www.idinet.com/`.

[23]Located at 2188 Bering Drive; San Jose, CA 95131; 408-434–0967; `http://www.edac.org/Apsim/`.

[24]Located at 5956 Sherry Lane; Dallas, TX 75225; 214-373 7344; `http://www.incases.com/`.

- Pacific Numerix[25] has *PCB Radiation*, which predicts radiated emission noise from printed circuit boards and multi-chip modules. It accurately calculates current distributions on the traces, vias, and planes.
- Quad Design Technology[26] has *Quiet*, which is an integrated EMI tool that finds layout EMI problems without prototyping. *Quiet* predicts radiated noise from printed circuit boards, see [24].
- Quantic[27] has *Compliance*, which "is acknowledged to yield simulation results with 1% of emperical laboratory results".
- Viewlogic Systems[28] has *Quiet Expert*, an electronic design automation tool billed as an EMI tool that employs "expert" technology to locate potential problems and to suggest ways in which to fix them.

We note that reducing the far-field radiated electromagnetic field (such as replacing common mode traces with differential mode traces) will also reduce the near-field radiated electromagnetic field. This, in turn, makes it more difficult for a sensor of the type used in this study to accurately measure the radiated fields.

Note also that other advancing technologies can limit the amount of electromagnetic energy radiated by a PCB:

- Reductions in radiated electromagnetic energy will occur as use of the "buried capacitance" technique increases, see Wang [47]. (Buried capacitance uses a dielectric layer of non-exotic material laminated between the PCB power and ground planes which generates sufficient bypass capacitance to be shared through the board.)
- Reductions in radiated electromagnetic energy will occur as voltages and powers on PCBs are reduced, see Goodenough [15].

8.6 Testing PCBs: A sampling of vendors

There are many manufacturers of automatic test equipment (ATE) for testing PCB's. A comparison of 26 different products, from 15 companies, is in [20].

The following is a sampling of manufacturers of PCB test equipment.

8.6.1 Emscan: Electromagnetic field sensor

Emscan[29] is an ElectroMagnetic SCANner which measures the near-field magnetic field under an active circuit board. *Emscan* was described in detail in [49], and is reviewed in Rowe [34]. Since [49] was written, the basic product has been improved to become the *Emscan/Q* model. For this model, frequency ranges have been extended from the original 10 MHz–750 MHz to 50 KHz–1.5 GHz, while spatial scan times have decreased from about 2 minutes per scan for *Emscan 2.1* to 0.3 seconds per scan using *Emscan/Q*.

The main drawback for *Emscan/Q* (for the present application) is that the resolution has not improved, still is still a spacing of 0.3 inches between sensors. This is very large distance when compared to modern digital circuit boards traces.

8.6.2 IBM: Flying probe testing

IBM manufactures a flying probe tester called the *Hummingbird*. It operated within a 12 mm work envelope and can make 50 open and short measurements per second per head. This is about five times faster than other flying probe testers (see Bunker and Lo [9]).

[25]Located at 7333 East Doubletree Ranch Road; Suite 280; Scottsdale, AZ 85258; 602-483–6800; `http://www.pnc.com/index.htm`.

[26]Located at 1385 Del Norte Rd; Camarillo, CA 93010; 805-988–8250; `http://www.quaddesign.com`.

[27]Located at 191 Lombard Ave; Winnipeg, Manitoba; Canada R3B 0X1; 204-942–4000

[28]Located at 293 Boston Post Road, West; Marlboro, MA 01752; 508-480–0881; `http://www.viewlogic.com/eagle/index.html`.

[29]Located at `http://www.emscan.com`.

8.6.3 Intelligent Automation Systems: Vision system

Intelligent Automation Systems[30] manufactures *WebMaster*, an on-line PCB inspection device. The system captures 50,000 coordinates in 1/10,000 seconds, provides measurement accuracy better than 1/1000 of the field of view, and inspects a range of PCBs up to $18'' \times 18''$.

8.6.4 Probot: Flying probe testing

Probot[31] manufactures flying probe test equipment. Their Series Six can have either 2 or 4 moving probes. Their specifications include the following information about probe placement:

- accuracy: 0.0025"
- repeatability: 0.0006"
- resolution: $x = 0.0002"$ $y = 0.0002"z = 0.0072"$

8.6.5 Probotech: Flying probe testing

Probotech[32] manufactures flying probe test equipment. Their flying probes are magnetically propoelled (so no lead screws, cables, or gears are required). Their specifications include the following information about probe placement:

- repeatability: 0.0002"
- probes 3 mil pads

8.6.6 Testron: Bed-of-nails testing

Testron[33] sells equipment for bed-of-nails testing. In their product literature they recommend that

> "Probe pads must be .035/.040 inches (.89/1.0 mm) diameter"

This requirement is due to worst case tolerance analysis. However,

> "Generally test pads can be smaller (.025" for boards under 12" square)"

Also

> "Component height on the probed side of DUT's shall not exceed .350 inches (6.4 mm)."

Components exceeding this height will require special consideration. And, to avoid having a probe touch other components on the DUT

> "There must be a 0.18 inches (.46 mm) annulus around each probe pad that is free of any component or other intrusion."

[30]Located at 149 Sidney Street; Cambridge, MA 02139; 617-354–3830; `http://www.ias.com/`.
[31]Located at 36 East Industrial Road; Branford, CT 06405; 203-481–4764, `http://www.probot.com`.
[32]Located at 6848 Hawthorn Park Drive; Indianapolis, IN 46220; 317-849–6197.
[33]Located at 41 Century Drive; Woonsocket, RI 02895; 800-262–4894;
`http://www.ttitestron.com/index.html`.

9 Powering up unknown PCBs

By "powering up" a circuit board we mean determining a sequence of input data values that are sent to the circuit board to ensure that all parts of the board are electrically active.

One of the goals of this project is to analyze any circuit board, even those for which we have no formal specifications. Given an undocumented board it is clear that none of the standard test techniques (see, for example, Abramovici *et al.* [3]) for circuit boards are appropriate. Instead, we have developed some heuristics for powering up an undocumented circuit board.

9.1 Overview

Our approach to powering up an undocumented circuit board is as follows:

1. Initialize the board:

   Create a pattern that places a given board into a repeatable state.

2. Determine an input sequence for the board:

   Create a pattern of inputs to feed to the board (see section 9.2 for how this is done).

3. Determine how effective the input sequence is:

   Use the electrooptic sensor system under development to determine those areas of the circuit board that are undergoing electrical activity.

4. Modify the input sequence:

   Adaptively modify the inputs that are fed to the board. The goal is to have all parts of the board electrically active.

This section describes a method we have developed for which:

1. The input patterns are pseudo-random (i.e., seemingly random but predictable and repeatable) and created by linear feedback shift registers (LFSR) (for a background on LFSRs, see section 9.1.1).

2. The adaptation is achieved by genetic algorithms (see section 9.2).

9.1.1 Linear feedback shift registers

The most widely used source for pseudo-random patterns is the linear feedback shift register (LFSR). An n-bit LFSR consists of n memory elements and at most n 2-input XOR gates. A model of an LFSR is shown in figure 64.

Note that McCluskey [27] (among others) used LFSRs to try to exhaustively examine all states in a circuit.

Associated with every LFSR is a feedback polynomial that determines the sequence of patterns that will be generated. For example, the LFSR in figure 64 has the feedback polynomial $1+c_1x+c_2x^2+\cdots+c_{n-1}x^{n-1}+x^n$, where each of the $\{c_i\}$ determines whether there is a feedback connection from the output of the ith latch as shown in figure 64. An n-bit LFSR will generate every n bit pattern except the all-zero pattern if its feedback polynomial is primitive.[34]

LFSRs have long been used to generate pseudo-random patterns because the hardware requirements are low.

[34] A monic irreducible binary polynomial $f(x)$ of degree k is *primitive* if the order of x in $Z_2[x]/(f(x))$ is $2^k - 1$, see [51, page 143].

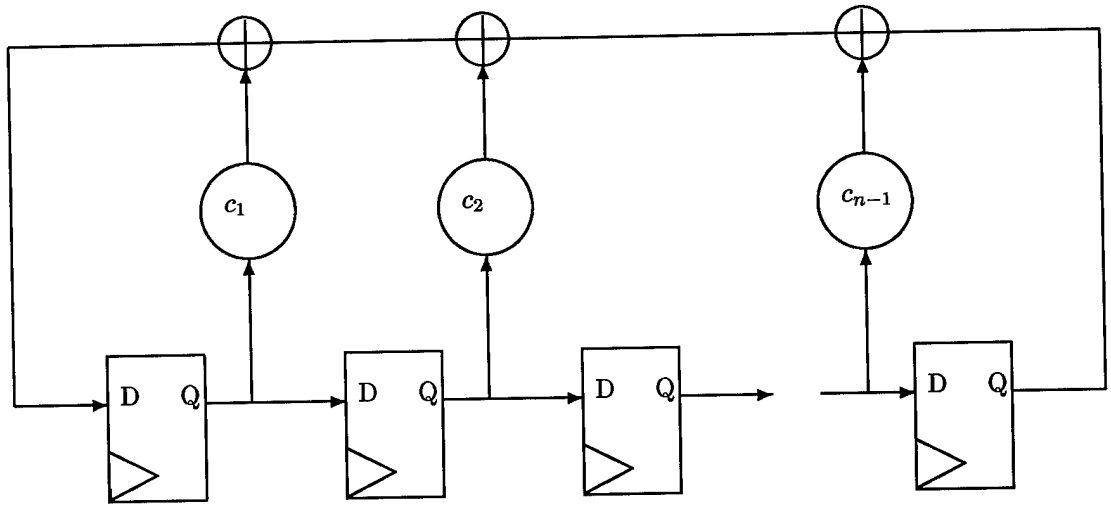


Figure 64: Generic linear feedback shift register (LFSR)

9.1.2 Parameters describing the powering up process

The fixed parameters include:

- K: the number of inputs to the circuit board itself
- M: the number of regions on the circuit board checked for electrical activity (Initially, a two dimensional grid of regions will be used, say a 3×4 region. Later might check that every trace is experiencing electrical activity).

For each linear feedback shift register (LFSR), there are the parameters:

- L: the number of registers in the LFSR
- I: the initialization of the LFSR (a number less than $2^L - 1$ representing a binary sequence)
- T: the taps of the LFSR (a number less than $2^L - 1$ representing a binary sequence)

Finally, the parameters connecting the LFSRs to the circuit board must be specified:

1. How many LFSR's should be used?
2. Should each circuit board input have it's own LFSR? Or should a single LFSR be iterated K times and those values used as inputs to the board? Or some intermediate strategy?
3. How often should a LFSR be iterated before it is replaced by a different LFSR?

9.1.3 Initialization of circuit boards

Note that initialization sequences are usually known for a given board. From Abramovici *et al.* [3] (page 104)

> This type of testing is based on the fundamental assumption that such an initialization sequence $T+l$ exists. Almost any circuit used in practice has an initialization sequence, which is employed to starts its operation from a known start. Circuits are usually designed so that they can be easily initialized. An often used technique is to provide a common reset (or preset) line to every F/F. Then a single vector is needed to initialize the circuit. However, an initiation sequence for the fault-free circuit N may fail to initialize some faulty circuit N_f. Such a fault f is said to *prevent initialization.*

We will presume, however, that we do not know the initialization sequence. Our algorithm presently sends in a sequence of zeros to each of the K inputs to the circuit board. The sequence length needs to be large enough to allow the circuit to settle down to its quiescent state, but short enough that it does not take too long. We presently plan to use a length of 10^4 bits for each input to the circuit board.

9.2 Optimization

Genetic algorithms are a fairly recent tool used for approximating the solution to optimization problems[35]. The concept is quite simple:

- Assume that there is an initial pool of "techniques".
- Assume that each "technique" can be "scored".
- The highest scoring techniques are separated into random components and randomly "spliced" together to create new techniques.
- These new techniques, and their scores, are added to the original pool of techniques.

This process is iterated until some criteria is met about how good the "best" score is.

For our application this general algorithm becomes as follows: Let us presume that the number of LFSRs powering a circuit board has been fixed, and that the connection of these LFSRs to the circuit board has been specified (say, for example, that each input to the circuit board corresponds to a distinct LFSR). We can then choose the parameters for each LFSR (i.e., values of the parameters $\{L_i, I_i, T_i\}$ for $i = 1, \ldots, K$). For any given set of the parameters (i.e., each "technique"), we can iterate the circuit board using the input LFSRS. We can evaluate how "good" a set of parameters is by counting the number of regions on the circuit board that were electrically excited (this is the "score"). The highest scoring techniques can be used to create a new techniques (in our case, the LFSRs from two different techniques could be combined, in any of several ways). These new parameter sets are added to the original pool of parameter sets. This process is iterated until some criteria (such as having all regions electrically active) is met.

For example, if at some stage of the adaptation we have (assume now that a single LFSR is used to drive all the circuit board inputs) the following scored techniques (i.e., parameter sets):

L_i	I_i	T_i	score
10	789	14	96
12	213	456	86
10	123	345	60
12	234	678	20

Then at the next stage we may try new LFSR's which are variations of a selected number of the high scoring LFSR's:

L_i	I_i	T_i	score
10	789	⟨rand⟩	to be determined
10	⟨rand⟩	14	to be determined
12	213	⟨rand⟩	to be determined
12	⟨rand⟩	456	to be determined
10	213	456	to be determined
12	789	14	to be determined

where ⟨rand⟩ represents a randomly generated number, or a number dependent on the previous state/tap vectors. The scores from these new LFSR's are then combined with the old scores, the scores are sorted, and the process repeated.

A detailed example showing all details is in section 9.4.

9.3 Software

There are many tools for circuit construction, and circuit simulation. Many of these are graphically based, and make the layout process quite simple. We decided to not use these tools, for reasons which we will describe.

[35] See, for example, Chambers [10]. For genetic algorithm FAQs, look to `http://www.cs.cmu.edu/Groups/AI/html/faqs/ai/genetic/top.html` for the "Genetic Algorithm Archives" see `http://www.aic.nrl.navy.mil/galist/`

The FAQ associated with the Usenet newsgroup `lsi.cad` contains descriptions of many such tools. We surveyed these tools, and learned how to use the Chipmunk tools, developed at Caltech. The Chipmunk tools allow an analog or digital circuit to be easily entered, and then simulated.

Armed with this powerful tool, we went looking for the design of a circuit board that we could simulate, and practice powering up. Unfortunately, we could find only isolated components that had been modeled using these tools (single chips, for example). This appears to be what the academic and research communities tend to publish. Since there was no great advantage obtained from using the circuit simulation tools (and a great learning curve in learning how to use the tool effectively), we wrote some very simple circuit simulation tools of our own in C++.

9.3.1 C++ Software

Aztec constructed a small C++ program that allows simple circuits to be input and to be simulated. This program also generates LFSRs and powers up the circuit with them. The program also performs the necessary optimization to power up all (or most) parts of a circuit board. The source code to all the routines is given in section D. A detailed example of the use of this code is in section 9.4.

LFSR code

Once a LFSR is established (by saying, for example, "`LFSR ab(6)`", which creates a LFSR with the name `ab` of length `6`), there are several operations that can be applied to it. For understanding the program, the important commands are:

- `ab.set_length(⟨argument⟩)` re-defines the length of the LFSR
- `ab.set_state(⟨argument⟩)` specifies the initial state
- `ab.set_taps(⟨argument⟩)` specifies the tap sequence
- `ab.iterate()` iterates the LFSR
- `ab.output()` returns the output of the iterated LFSR

Circuit generation code

For testing purposes, Aztec generated circuits for testing with the following 2-input logical devices: `AND` gates, `NAND` gates, `OR` gates, `NOR` gates, and JK flip-flops.[36] For ease in constructing our test circuits, we have restricted ourselves to circuits that have the appearance of figure 66. This circuit has n inputs, shown on the left. There are also m levels of logical devices (of which only the levels 1 and 2 are shown in the figure). The levels are illustrated as columns. We presume that all logical devices in a given level are identical (for example, at Device Level 1 there may be a column of `AND` gates). Since each logical device requires two inputs, the inputs to logical device number i at Device level I are taken to be the logical values on lines i and $(i+I)$ (appropriately reduced modulo the number of devices, n)[37] of the previous level.

For example, if ℓ_a^b represents the logical value of line a at Device Level b, and L_b represents the binary function of the logical devices in Device Level b, then we have (for a 3 input system):

$$\begin{array}{lccc}
\text{compute outputs after device level 1} \rightarrow & \ell_0^1 = L_1(\ell_0^0, \ell_1^0) & \ell_1^1 = L_1(\ell_1^0, \ell_2^0) & \ell_2^1 = L_1(\ell_2^0, \ell_0^0) \\
\text{compute outputs after device level 2} \rightarrow & \ell_0^2 = L_2(\ell_0^1, \ell_2^1) & \ell_1^2 = L_2(\ell_1^1, \ell_0^1) & \ell_2^2 = L_2(\ell_2^1, \ell_1^1) \\
\text{compute outputs after device level 3} \rightarrow & \ell_0^3 = L_3(\ell_0^2, \ell_1^2) & \ell_1^3 = L_3(\ell_1^2, \ell_2^2) & \ell_2^3 = L_3(\ell_2^2, \ell_0^2) \\
\text{compute outputs after device level 4} \rightarrow & \ell_0^4 = L_4(\ell_0^3, \ell_1^3) & \ell_1^4 = L_4(\ell_1^3, \ell_2^3) & \ell_2^4 = L_4(\ell_2^3, \ell_0^3) \\
& \vdots & &
\end{array}$$

The C++ program constructed consists of the following files (these file are listed in appendices D.1–D.8):

[36] Note that JK flip-flops retain a state while the other logical devices do not (see their logic diagram in figure 65).

[37] Note that if $i \equiv i + I \pmod{n}$ (equivalently, I is a multiple of n), then the two inputs are taken to be i and $i+1$.

Present State	Desired Next State	Required Excitations	
		J	K
0	0	0	-
0	1	1	-
1	0	-	1
1	1	-	0

Figure 65: Logic for a JK flip-flop

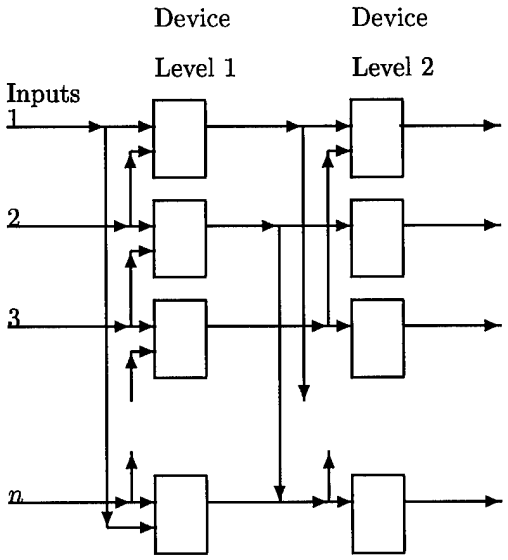


Figure 66: Generic circuit used for simulations

- `lfsr.h` header file used by all routines (see section D.1)
- `main.h` header file used by the main routine (see section D.2)
- `extern.h` header file used by all routines except the main routine (see section D.3)
- `main.c` contains the main driving routine (see section D.4)
- `lfsr.c` contains the code that defines and iterates a LFSR (see section D.5)
- `system.c` the routines that define the circuit, optimization scoring metric, etc (see section D.6)
- `util.c` a collect of needed utilities (see section D.7)
- `Makefile` is the makefile for the entire process. (see section D.8)

Aztec used the following programming conventions:

- The global variable names begin with `G_`
- The `defined` variables have names composed of all uppercase letters

An example usage of these programs is in section 9.4.

9.4 Example usage of the C++ programs

9.4.1 Optimization algorithm used for testing

When using an optimization algorithm, we must have a single metric (or score) that we try to maximize. Ideally, we would like there to be a great deal of electrical activity in all regions of the circuit board. Alternately, we would like there to be a specified minimum amount of activity in *every* region of the circuit board. In the computer tests we performed, we defined the score of a technique to be the average number of

transitions for each region (in our case, the number of transitions of each output of each device) multiplied by the minimum number of transitions in each region (i.e., the minimum number of transitions among all wires). If this was zero (since some lines experienced no transitions and the minimum value was equal to zero) then the score was instead defined to be the negative of the number of lines with no transitions. This produced a single Figure of merit that we attempted to optimize; when negative it meant that some lines had never changed. The more negative the number, the fewer lines that experienced transitions.

9.4.2 Testing

We used the code discussed in sections 9.3.1 and 9.3.1 to show how well a LFSR can power up a circuit. We took a circuit with 11 inputs and 20 levels of logical devices. The devices used in the circuit were NAND gates for all levels except that NOR gates were used in device levels 14 and 15, and JK flip-flops were used in device levels $\{5, 7, 8, 10, 11, 12, 17\}$.

Two single LFSRs of length 13 were used to start the optimization process. At each clock cycle, the LFSR was iterated 11 times and the 11 resulting bits (one output is obtained each time the LFSR is iterated) were used as inputs to the circuit. The circuit was stepped 250 times. Each time the entire circuit was allowed to come to rest. Note that sometimes the state of the JK flip-flops did not change even when the inputs to the JK flip-flop did change.

While the circuit was being iterated, we counted the number of times that each line (or "wire") output from a device changed from 0 to 1 or from 1 to 0. When this number was displayed, and the number was larger than 9, a "*" was displayed.

When showing all values of inputs and outputs (here are four examples of this below), down the page is the number of input lines. Across the page are (1) the present state of the logic for all lines, (2) the present state of any JK flip-flops that might be present, and (3) the number of times the state of a line has changed.

Some of the output is displayed below. In the lines that show a score, the last three numbers in parentheses indicate the LFSR parameters (the length of the LFSR, the initial state of the LFSR, and the tap sequence for the LFSR) and the final two numbers—separated by a slash—indicate which recombination technique was used to find this set of parameters, and what optimization round it was found in.

```
******************
Date is Jun 20 1996
Time is 14:47:34

D Entering create_circuit with:
  G_number_device_levels=20
  G_number_inputs=11

The devices used in this simulated circuit are:
...device that produced level  1 is a(n) NAND gate
...device that produced level  2 is a(n) NAND gate
...device that produced level  3 is a(n) NAND gate
...device that produced level  4 is a(n) NAND gate
...device that produced level  5 is a(n) JK   flip-flop
...device that produced level  6 is a(n) NAND gate
...device that produced level  7 is a(n) JK   flip-flop
...device that produced level  8 is a(n) JK   flip-flop
...device that produced level  9 is a(n) NAND gate
...device that produced level 10 is a(n) JK   flip-flop
...device that produced level 11 is a(n) NAND gate
...device that produced level 12 is a(n) JK   flip-flop
...device that produced level 13 is a(n) NAND gate
...device that produced level 14 is a(n) OR   gate
...device that produced level 15 is a(n) OR   gate
...device that produced level 16 is a(n) NAND gate
...device that produced level 17 is a(n) JK   flip-flop
...device that produced level 18 is a(n) NAND gate
...device that produced level 19 is a(n) NAND gate

D initial data (SHOULD RANDOMIZE INITIAL STATE INFORMATION!)
           01234567890123456789         123456789012345678          123456789012345678
...line  0:                      states: ----1-11-1-1----1-- changed: 0000000000000000000
```

```
...line  1:                      states: ----1-11-1-1----1-- changed: 0000000000000000000
...line  2:                      states: ----1-11-1-1----1-- changed: 0000000000000000000
...line  3:                      states: ----1-11-1-1----1-- changed: 0000000000000000000
...line  4:                      states: ----1-11-1-1----1-- changed: 0000000000000000000
...line  5:                      states: ----1-11-1-1----1-- changed: 0000000000000000000
...line  6:                      states: ----1-11-1-1----1-- changed: 0000000000000000000
...line  7:                      states: ----1-11-1-1----1-- changed: 0000000000000000000
...line  8:                      states: ----1-11-1-1----1-- changed: 0000000000000000000
...line  9:                      states: ----1-11-1-1----1-- changed: 0000000000000000000
...line 10:                      states: ----1-11-1-1----1-- changed: 0000000000000000000
After 249 time steps
           01234567890123456789          1234567890123456789          1234567890123456789
...line  0: 01010010011011110010 states: ----0-00-1-1----0-- changed: **************33122
...line  1: 11010110011001110010 states: ----1-00-1-0----0-- changed: **************33122
...line  2: 01110111011101110010 states: ----1-10-1-0----0-- changed: **************33122
...line  3: 01010110010111110110 states: ----1-00-0-1----1-- changed: **************55222
...line  4: 10110111110101110010 states: ----1-11-0-0----0-- changed: **************33122
...line  5: 11010110011101110010 states: ----1-00-1-0----0-- changed: **************33122
...line  6: 01010010100110010010 states: ----0-01-0-1----0-- changed: **************33122
...line  7: 01010010011101110010 states: ----0-00-1-0----0-- changed: **************33122
...line  8: 01010011100110110010 states: ----0-11-0-1----0-- changed: **************33122
...line  9: 01010011111110110110 states: ----0-11-1-1----1-- changed: **************35222
...line 10: 11010011100111110010 states: ----0-11-0-1----0-- changed: **************33122

D initial data (SHOULD RANDOMIZE INITIAL STATE INFORMATION!)
           01234567890123456789          1234567890123456789          1234567890123456789
...line  0:                      states: ----1-11-1-1----1-- changed: 0000000000000000000
...line  1:                      states: ----1-11-1-1----1-- changed: 0000000000000000000
...line  2:                      states: ----1-11-1-1----1-- changed: 0000000000000000000
...line  3:                      states: ----1-11-1-1----1-- changed: 0000000000000000000
...line  4:                      states: ----1-11-1-1----1-- changed: 0000000000000000000
...line  5:                      states: ----1-11-1-1----1-- changed: 0000000000000000000
...line  6:                      states: ----1-11-1-1----1-- changed: 0000000000000000000
...line  7:                      states: ----1-11-1-1----1-- changed: 0000000000000000000
...line  8:                      states: ----1-11-1-1----1-- changed: 0000000000000000000
...line  9:                      states: ----1-11-1-1----1-- changed: 0000000000000000000
...line 10:                      states: ----1-11-1-1----1-- changed: 0000000000000000000
After 249 time steps
           01234567890123456789          1234567890123456789          1234567890123456789
...line  0: 01011010010110110101 states: ----0-00-0-1----1-- changed: *****2*******311011
...line  1: 01011011110111110011 states: ----0-11-0-1----0-- changed: ********3*****33121
...line  2: 01011110010110110010 states: ----1-00-0-1----0-- changed: **************311122
...line  3: 11010110100101110111 states: ----1-01-0-0----1-- changed: **************13021
...line  4: 01101110011111110010 states: ----1-00-1-1----0-- changed: *****2********11122
...line  5: 01101110010101110110 states: ----1-00-0-0----1-- changed: *****2******3313022
...line  6: 10101111111101110010 states: ----1-11-1-0----0-- changed: *****2********11122
...line  7: 10110011110111110101 states: ----0-11-0-1----1-- changed: *****2*******113011
...line  8: 10110010100111110010 states: ----0-01-0-1----0-- changed: *****2********33122
...line  9: 10110010011001110101 states: ----0-00-1-0----1-- changed: *****2********11011
...line 10: 11010010011101110011 states: ----0-00-1-0----0-- changed: *****2********33121

The ordered list of LFSR'd is:
i= 0 order_lfsr[i]= 0 array_lfsr[order_lfsr[i]].score=  81.67(13,  100,  101)-1/0
i= 1 order_lfsr[i]= 1 array_lfsr[order_lfsr[i]].score=  -5.00(13,  200,  201)-1/0
```

and later, after several optimization steps:

```
The ordered list of LFSR'd is:
i= 0 order_lfsr[i]=62 array_lfsr[order_lfsr[i]].score= 426.80(13,  300,  931)1/18
i= 1 order_lfsr[i]=61 array_lfsr[order_lfsr[i]].score= 403.55(13,  300, 6883)1/47
i= 2 order_lfsr[i]=75 array_lfsr[order_lfsr[i]].score= 303.44(13,  300, 2019)1/67
i= 3 order_lfsr[i]=21 array_lfsr[order_lfsr[i]].score= 257.71(13,  300, 5731)1/77
i= 4 order_lfsr[i]=14 array_lfsr[order_lfsr[i]].score= 251.28(13,  300, 6051)3/21
i= 5 order_lfsr[i]=74 array_lfsr[order_lfsr[i]].score= 249.95(13,  611, 6051)7/22
i= 6 order_lfsr[i]=39 array_lfsr[order_lfsr[i]].score= 249.23(13,  300, 5219)1/5
i= 7 order_lfsr[i]=30 array_lfsr[order_lfsr[i]].score= 247.53(13,  300, 5155)1/4
i= 8 order_lfsr[i]= 6 array_lfsr[order_lfsr[i]].score= 246.48(13,  300,  101)4/1
i= 9 order_lfsr[i]=45 array_lfsr[order_lfsr[i]].score= 246.45(13,  300, 7779)3/32
```

```
i=10 order_lfsr[i]=89 array_lfsr[order_lfsr[i]].score= 236.52(13,  300, 1626)2/19
i=11 order_lfsr[i]=49 array_lfsr[order_lfsr[i]].score= 171.51(13,  300, 4701)2/6
i=12 order_lfsr[i]= 7 array_lfsr[order_lfsr[i]].score= 171.51(13,  300, 7459)1/72
i=13 order_lfsr[i]= 9 array_lfsr[order_lfsr[i]].score= 169.86(13,  300, 1251)1/23
i=14 order_lfsr[i]=67 array_lfsr[order_lfsr[i]].score= 169.23(13,  300, 6499)3/44
i=15 order_lfsr[i]=65 array_lfsr[order_lfsr[i]].score= 168.61(13,  300,  995)1/51
i=16 order_lfsr[i]=44 array_lfsr[order_lfsr[i]].score= 168.03(13,  300, 3491)3/45
i=17 order_lfsr[i]=25 array_lfsr[order_lfsr[i]].score= 167.97(13,  300,  163)1/38
i=18 order_lfsr[i]=31 array_lfsr[order_lfsr[i]].score= 167.95(13, 1091, 3275)8/89
i=19 order_lfsr[i]=76 array_lfsr[order_lfsr[i]].score= 167.94(13, 4131, 5155)7/13
```

10 Conclusions

In the statement of work for this project, there are several specific areas that were to be addressed:

1. *An evaluation of the effectiveness of the electromagnetic field imaging technologies tested, with recommendations on how the system may be used.* (See section 10.1.)
2. *Developmental requirements that must be met to mature the prototype into a production tool.* (See section 10.2.)
3. *Documentation of the algorithms and programs developed.* (See section 10.3.)
4. *Documentation of the test results.* (See section 10.4.)

Summarizing the results addressed above, we make several recommendations:

- To use this system effectively, a new sensor will need to be identified.
- As each sensor has it's own characteristic errors (due to such things as thermal noise), it is difficult to generalize about how to process the data acquired from an unknown sensor. Aztec demonstrated several different techniques for processing the data in section 7.
- To avoid the problems encountered in the research described here, it is recommended that the Air Force not depend on one vendor for key technology, and that the key technology developer have manufacturing capabilities (not just research capabilities—and prototypes—that may not be duplicatable).

10.1 Effectiveness

The sensor (which was to be manufactured by SRICO) that we wished to use to demonstrate the capabilities of the sensor and imaging system was never obtained. The reasons are stated in Appendix C. The preliminary sensors constructed by SRICO, described in section 3, were unusable.

The overall system depended critically on obtaining this sensor. In [49] Aztec determined that sensors using the electrooptical effect (Pockels effect) appeared to be sensitive enough to create images understandable by humans of circuit boards. It was also stated that the SRICO sensor appeared to be adequate for the measurement of electric fields produced by a low power digital circuit board. No other sensor appeared as promising. Without the SRICO sensor, we used a commercially available antenna-based sensor (see section 4.1), in an effort to validate the overall system concept. Use of this inferior sensor did indicate that data could be obtained and visualized (see section 7.5). However, as predicted in [49], this sensor was not accurate enough to obtain human understandable images of low power digital I/O boards.

As stated in section 5.2 the software system (EFIS) controlling the experimental setup can be easily adapted to other sensors.

10.2 Maturity

To mature this technology a new electromagnetic sensor must be developed. Characteristics of the sensor will dictate characteristics of the system. (For example, sensor resolution—see section 4.3—will dictate system sampling frequency). These resulting system characteristics will dictate how human understandable the resulting images will be. It is impossible to state exactly how to perform the image processing, or even what the final limitation of the images will be without a complete specification of a new electromagnetic sensor. Note that it is likely that unforeseen effects, such as the thermal drift with the prototype SRICO sensors (see section 3.2), will compromise the image resolution until the inaccuracies due to these unforeseen effects are corrected for.

10.3 Documentation of algorithms

All of the algorithms studied have been given a textual motivation and description (see sections 5 and 7). Additionally, the computer code is written in high level languages (LabView, C++, and Matlab) and each routine has been internally documented; see section D.

10.4 Documentation of test results

All results and datasets have been carefully documented, see sections 3, 5, and 7. In particular, we identified a severe temperature drift problem (see section 3.2).

10.5 Deliverables

We are delivering the following information created during the course of this contract:

1. Two printed copies of this final report
2. Two copies of a zip disk with the following directories and files (only some names shown)
   - Directory: `distribution`
     (contains all LabView programs for running the sensing system described in section 5)
   - Directory: `matlab_info`
     (Contains all matlab related materials)
     - Subdirectory: `matlab_data`
       * The files in the various sub-directories have the extensions
         `.dat` (output of EFIS program, snaked information– see Figure 16 on page 26)
         `.tad` (desnaked form of data—this data is in a single column)
         `.rct` (desnaked data in array form—still has header)
         `.m` (data suitable for matlab "load" command—header removed)
     - Subdirectory: `matlab_programs`
       * Contains files listed in Appendix D
   - Directory: `FinalReport`
     (All files in this directory, except for the `final.dvi` file, are in ascii.)
     - File: `final.tex` (source TEX file for final report)
     - File: `final.dvi` (dvi version of final report)
     - File: `final.bib` (bibliography)
     - File: `final.bst` (bibliography style)
     - File: `final.ist` (index style)
     - File: `final.macro` (macros for use by TEX file)
     - File: `doit` (executing this file on a UNIX system will create the final report. It runs LATEX, BibTEX, and then LATEX twice again)
     - LATEX class file: `article.cls`
     - LATEX file: `Umsa.fd`
     - LATEX file: `Umsb.fd`
     - LATEX file: `psfig.tex`
     - LATEX file: `psfig.tex`
     - LATEX file: `size10.tex`
     - LATEX style file: `afterpage.sty`
     - LATEX style file: `amstex.sty`
     - LATEX style file: `dvips.sty`
     - LATEX style file: `fancyheadings.sty`
     - LATEX style file: `fullpage.sty`
     - LATEX style file: `graphics.sty`
     - LATEX style file: `lastpage.sty`
     - LATEX style file: `multicol.sty`
     - LATEX style file: `trig.sty`
     - LATEX style file: `url.sty`
     - LATEX style file: `verbatimfiles.sty`
     - Subdirectory: `FIGURE` (contains figures)
     - Subdirectory: `LFSR` (contains C++ code)
     - Subdirectory: `MATLAB` (contains Matlab code figures)

A Reference material

A.1 Acknowledgment

Aztec gratefully acknowledges the guidance and support provided by Jeffrey S. Dean of SA-ALC/LDAE at Kelly AFB. Aaron Dalton of SA-ALC/LDAE at Kelly AFB made numerous suggestions which improved this final report.

A.2 Paper presented

The paper [52] by Zwillinger, Heff, and Dean was presented at the 3rd annual Joint Avionics and Weapon System Support, Software, and Simulation Conference (JAWSS^3), 8–13 June 1997, Riviera Hotel, Las Vegas, NV.

A.3 Abbreviations and acronyms

Some of the common abbreviations and acronyms in the fields of printed circuit board design, analysis, and test are the following:

ASCII	American Standard Code for Information Interchange
ATE	automated test equipment
ATS	automatic test system
BGA	ball grid array
BUT	board under test
C++	an object oriented version of C
CAD	computer aided design
CNR	carrier to noise ratio
CNR	carrier to noise ratio
COTS	commercial off-the-shell
CT	combinatorial testers
cw	continuous wave
dBm	decibels relative to 1 milliwatt
dB	decibels
DUT	device under test
D	electric displacement
EMC	electromagnetic compatibility
EMI	electromagnetic interference
E	electric field strength
FAQ	frequently asked questions
FBT	functional board testers
FEA	finite-element analysis
FFT	fast Fourier transform
FPY	first pass yield
GPIB	general purpose interface bus
HDL	Harry Diamond Laboratories
Hz	Hertz
IC	integrate circuit
ICT	in-circuit testers
IPC	The Institute for Interconnecting and Packaging Electronic Circuits
LFSR	linear feedback shift register
$LiNbO_3$	lithium niobate
MDA	manufacturing defects analyzers
mil	a thousandth of an inch
MLB	multiple layer board

mm	millimeter: 1 millimeter is equal to 40 mil or 0.040 inches
nm	nanometer (equal to 10^{-9} meter)
PBGA	plastic ball grid array
PCB	printed circuit board; these are either "bare" or "assembled"
PWB	printed wiring board
RF	radio frequency
s-a-0	stuck at 0
s-a-1	stuck at 1
SNR	signal to noise ratio
TTL	transistor–transistor logic
XOR	exclusive or logic operation

A.4 Terminology

- **Accuracy** The closeness of the agreement between the result of a measurement and the value of the specific quantity subject to measurement, i.e. the measurand. Although most equipment manufacturers still use the term as a tolerance in their specifications, NIST and other international standards bodies have classified it as a qualitative concept not to be used quantitatively. The current uniform approach is to report a measurement result accompanied by a quantitative statement of uncertainty.
- **Error** The result of a measurement minus the value of the measurand.
- **Precision** The closeness of agreement between independent test results obtained under stipulated conditions. Precision is a qualitative terms used in the context of repeatability or reproducibility and should never be used interchangeably with accuracy.
- **Repeatability** The closeness of agreement between the results of measurements of the same measurand carried out under changed conditions of measurement.
- **Reproducibility** The closeness of the agreement between the results of measurements of the same measurand carried out under changed conditions of measurement.
- **Resolution** A measure of the smallest portion of the signal that can be observed. For example, a thermometer with a display that reads to three decimal digits would have a resolution of 0.0001°. In general, the resolution if an instrument has a better rating than its accuracy.
- **Sensitivity** The smallest detectable change in a measurement. The ultimate sensitivity of a measuring instrument depends on both its resolution nd the lowest measurement range.
- **Uncertainty** The estimated possible deviation of the result of measurement from its actual value. The uncertainty of the result of a measurement generally consists of several components that may be grouped into two categories according to the method used to estimate their numerical values: (1) those evaluated by statistical methods, and (2) those evaluated by other means. Uncertainty and error are not to be used interchangeably.

A.5 Glossary

- **comparative tester** Evaluation of a component by comparing its performance to that of a properly functioning component.
- **digital signal** A discrete signal that assumes one of two states: high or low.
- **fields**
  - **electrostatic fields** These are fields produced by static electric charges.
  - **magnetostatic fields** These are fields produced by the motion of electric charges with uniform velocity (direct current).
  - **time-varying fields** These are fields produced by accelerated charges or time-varying currents.
- **free space** A space containing no particles and no force fields (note that a vacuum may contain fields).
- **ground plane** A region on a circuit card maintained at 0 Volts.
- **in-circuit testing** Evaluating the functioning of a component without removing the component from the circuit in which it is being used.

- **printed circuit assembly** A printed circuit board on which separately manufactured components have been added.
- **printed contact** That part of a printed circuit used to connect the conductive pattern to a plug-in receptacle, or the equivalent of a pin on a male plug. It was previously known as "tab" or "finger".
- **terminal area** That part of a printed circuit that makes connection to the conductive pattern, such as the enlarged area of conductor material around a component-mounted hole. This is preferred to such terms as "boss, land, pad, or terminal pad".
- **trace** This is a conductor on a printed circuit board. Standard sizes for copper are 1 oz (corresponding to 0.0014" in thickness) and 2 oz (corresponding to 0.0028").
- **TTL** Transistor transistor logic defines a type of digital circuit. It is characterized by a high digital signal state above 2 VDC, and low digital state below 0.8 VDC. Operating voltage is typically 5 VDC.
- **via** A via is a connection between two layers of a circuit board. A buried via is one in which neither end of the via terminates on an external layer of the circuit board. A via that extends between the top and bottom of a circuit board is called a "through hole".
- **comparative tester** Evaluation of a component by comparing its performance to that of a properly functioning component.
- **digital signal** A discrete signal that assumes one of two states: high or low.
- **ground plane** A region on a circuit card maintained at 0 Volts.
- **in-circuit testing** Evaluating the functioning of a component without removing the component from the circuit in which it is being used.
- **pitch** The pitch is the width of one trace and the adjacent space. It measures how fine the wires are.

- **printed circuit assembly** A printed circuit board on which separately manufactured components have been added.
- **printed contact** That part of a printed circuit used to connect the conductive pattern to a plug-in receptacle, or the equivalent of a pin on a male plug. It was previously known as "tab" or "finger".
- **terminal area** That part of a printed circuit that makes connection to the conductive pattern, such as the enlarged area of conductor material around a component-mounted hole. This is preferred to such terms as "boss, land, pad, or terminal pad".
- **trace** This is a conductor on a printed circuit board. Standard sizes for copper are 1 oz (corresponding to 0.0014" in thickness) and 2 oz (corresponding to 0.0028").
- **TTL** Transistor transistor logic defines a type of digital circuit. It is characterized by a high digital signal state above 2 VDC, and low digital state below 0.8 VDC. Operating voltage is typically 5 VDC.
- **via** A via is a connection between two layers of a circuit board. A buried via is one in which neither end of the via terminates on an external layer of the circuit board. A via that extends between the top and bottom of a circuit board is called a "through hole".

A.6 Bibliography

[1] *Khoros.* Khoral Research, Inc, Albuquerque, NM. `http://www.khoros.unm.edu`.

[2] Technology Transfer. *Electronic Packaging & Production*, page 41, June 1997.

[3] M. Abramovici, M. A. Breuer, and A. D. Friedman. *Digital Systems Testing and Testable Design.* IEEE Press, New York, 1990.

[4] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions.* National Bureau of Standards, Washington, D.C., 1964.

[5] T. E. Adams. Nondestructive internal inspection of mounted components. *Surface Mount Technology*, pages 63–65, November 1993.

[6] H. Benitez. Routes to EMC compliance: You have choices. *Evaluation Engineering*, pages 123–125, March 1997.

[7] S. Bonheim and W. Singbeil. No single test strategy. *Circuits Assembly*, pages 68–71, November 1997.

[8] M. Buetow. Just what constitutes a "True revolution"? *Printed Circuit Fabrication*, 20(7):40–41, July 1997.

[9] C. J. Bunker and J.-C. Lo. The hummingbird tester. *Printed Circuit Design*, 20(8):40–41, August 1997.

[10] L. D. Chambers. *Practical Handbook of Genetic Algorithms.* CRC, Boca Raton, FL, 1995.

[11] M. Clarkson. PCB and IC design software. *Desktop Engineering*, pages 46–54, June 1997.

[12] Design News. U.S. PWB production tops $7 billion. *Printed Circuit Design*, 20(8):10, August 1997.

[13] J. Dixon. The savvy designer saves money .... *Printed Circuit Fabrication*, 18(8):27–29, August 1995.

[14] R. O. Duda and P. E. Hart. *Pattern Classification and Scene Analysis.* John Wiley & Sons, New York, 1973.

[15] F. Goodenough. Sophisticated techniques are being applied to lower voltage and power level for analog ICs. *Electronic Design*, pages 31–38, 2 September 1997.

[16] I. S. Gradshteyn and M. Ryzhik. *Tables of Integrals, Series, and Products.* Academic Press, New York, 1994.

[17] D. Hague and M. Kaida. The future of substrate testing. *Printed Circuit Design*, 20(8):10, August 1997.

[18] L. Hamburg. The PCB benchmark that never was. *Printed Circuit Design*, pages 32–39, June 1996.

[19] Hewitt and Associates. EM visualization of printed circuit board assemblies. *US Government report AD A293-355*, June 1994.

[20] G. Jacob. Low-cost ATE $\neq$ low performance. *Evaluation Engineering*, pages 50–57, June 1995.

[21] G. Jacob. Probe cards and stations cope with submicron IC features. *Evaluation Engineering*, pages 90–93, March 1998.

[22] M. J. Kuzawinski and K. J. Blackwell. Ultra fine pitch wire bond PBGAs. *Electronic Packaging & Production*, pages 54–62, September 1997.

[23] J. Laatikainen and P. Kess. X-ray inspectionhelps improve process yield. *Circuits Assembly*, pages 38–45, March 1997.

[24] C.-W. Lam and J. Powell. Use simulation to spot and fix EMI problems. *Electronic Design*, pages 95–106, 22 July 1996.

[25] J. Lipman. Tools and techniques stifle EM emissions. *EDN*, pages 83–92, 6 November 1997.

[26] K. MacLean. Design for test. *SMT*, pages 130–134, February 1997.

[27] E. J. McCluskey. Condensed linear feedback shift register (LFSR): A pseudoexhaustive test technique. *IEEE Transactions on Computing*, C35, April 1986.

[28] D. Miller. X-ray systems fill modern PCB inspection voids. *Evaluation Engineering*, pages 48–50, November 1997.

[29] D. H. Naghski, J. T. Boyd, H. E. Jackson, S. Sriram, S. A. Kingsley, and J. Latess. An integrated photonic mach–zender interferometer with no electrodes for sensing electric fields. *Journal of Lightwave Technology*, 12(6):1092–1098, June 1994.

[30] H. Nakahara. Success with finer lines. *Printed Circuit Fabrication 1997–1998*, 20(9):36, September 1997.

[31] H. Nakahara. 1997 Top-of-the world PCB makers. *Printed Circuit Fabrication*, 21(3):46–49, March 1998.

[32] S. Oresjo. Test and inspection. *SMT's Solutions for Today*, pages 35–37.

[33] P. O'Shea. PCB inspection paves the way to process improvements. *EE Evaluation Engineering*, pages 38–39, July 1997.

[34] M. Rowe. Find the source of EMI emissions. *Test & Measurement World*, pages 44–52, November 1997.

[35] J. C. Russ. *The Image Processing Handbook.* CRC, Boca Raton, FL, 1992.

[36] M. Slamani and B. Kaminska. Multifrequency analysis of faults in analog circuits. *IEEE Design & Test of Computers*, pages 70–80, Summer 1995.

[37] C. H. Small. New paradigms ensure high-performane PCBs. *Computer Design*, pages 60–69, October 1997.

[38] H. F. Spence. Printed circuit board diagnosis using artificial neural networks and circuit magnetic fields. *IEEE AES Systems Magazine*, pages 20–24, February 1994.

[39] S. Sriram. Optical interferometers for sensing electromagnetic fields. *US Government report AD A273 712*, 1991.

[40] K. Sterling. What's happening in the EMS industry? *Circuits Assembly/IPC Market Supplement*, pages S4–S6, April 1997.

[41] M. Swaminathan, A. Chatterjee, and B. Kim. Testing high density substrates. *Advanced Packaging*, pages 33–46, January/February 1997.

[42] M. Teska. X-RAY inspection for every line. *SMT*, pages 48–49, August 1997.

[43] M. Teska. Fine line PCB assembly leads to new test issues. *Electronic Packaging and Production*, pages 54–56, January 1998.

[44] The Mathworks. *Image Processing Toolbox: User's Guide.* The Mathworks, Natick, MA, 1997.

[45] C. Vaucher. Bare board test. *Printed Circuit Fabrication*, 20(8):20–25, August 1997.

[46] A. Vollmer. New simulation tools help PC boards meet EMC requirements. *Electronic Design*, pages 93–98, 13 October 1997.

[47] T. Wang. Characteristics of buried capacitance. *EMC Test & Design*, February 1993.

[48] A. I. Zayed. *Handbook of Function and Generalized Function Transformations.* CRC, Boca Raton, FL, 1996.

[49] Zwillinger & Associates. Visualization of circuit card electromagnetic fields. *US Government report AD A291 491*, January 1995. Final report for Air Force contract F41608-94-C-1126.

[50] D. Zwillinger. *Handbook of Differential Equations.* Academic Press, Orlando, Fl, second edition, 1992.

[51] D. Zwillinger. *Standard Mathematical Tables and Formulae.* CRC, Boca Raton, FL, 30th edition, 1995.

[52] D. Zwillinger, A. Heff, and J. S. Dean. New sensor system for electronic testing. In *THE 3RD JOINT AVIONICS AND WEAPON SYSTEM SUPPORT, SOFTWARE, AND SIMULATION CONFERENCE (JAWSS^3).* Riviera Hotel, Las Vegas, NV, 8–13 June 1997.

B Use of Khoros

We originally used Khoros [1] to perform image analysis operations. While we had wanted to use Matlab (since it has a familiar syntax and an image processing toolbox) it was constrained by the fact that the basic Matlab data structure is two-dimensional. Admittedly, this structure is ideal for representing matrices and is very useful for two-dimensional images (a color image is represented by 3 matrices; one for each primary color).

In contrast, the fundamental data structure in Khoros is a five-dimensional data structure. This is a more useful fundamental structure as it can contain data (3 dimensions) that is changing in time (1 more dimension). The last data dimension is for the the three primary colors (or other parameters that are chosen to be associated with a volumetric point changing in time).

Khoros is distributed by Khoral Research. Their mission is to create the preferred scientific software environment. Khoral Research plans to achieve this by widely distributing then continually improving Khoros through collaboration within the user community. For this Advanced Khoros through the Internet, and low-cost access to Khoros Pro (which Aztec has obtained). Note, however, that no version of Khoros is shareware software or in the public domain.

When Matlab 5 was released, with it's full support for multi-dimensional data structures, we stopped using Khoros and began using Matlab.

C Srico's poor performance in constructing a sensor

Aztec is extremely disappointed with the performance of Srico. Their poor performance on their subcontract was disgraceful and compromised our position on this SBIR contract.

C.1 Recitation of Srico's performance

A short recitation of their performance is in order:

- Srico had a continuing stream of excuses for the lateness of their prototypes. Such as "one vendor broke two devices during handling" and "[another] vendor mis-read the drawing and made the active length [of the sensor] 2.5 mm and not 1.5 mm."
- Srico lent Aztec two sensors to use in testing and debugging of our system. Their performance was so poor that we could achieve little with them.

- The first sensor prototype was delivered over one year past the due date. It did not meet its specifications. It also had a severe temperature drift problem (see section 3.2).
- The second prototype was delivered in July 1997. According to their tests the sensitivity of the sensor is about 20 Volts/meter/$\sqrt{\text{Hz}}$. Our tests confirmed this result. At the beginning of the contract, Srico told us that a comparable sensor they had fabricated and produced had a sensitivity of 0.1 Volts/meter/$\sqrt{\text{Hz}}$, i.e, 45 dB more sensitive than the sensor which we received.

  Additionally, the gain of this sensor fluctuated by more than 15 dB, sometimes in as little as four minutes.

C.2 Srico in material default of their contract

Srico's poor performance required us to find them in material default of their contract. Our July 1997 letter to Srico informing them of their breach contained the following direct quote:

- **"Failure to make deliveries**

  The first prototype was required to be delivered in April, 1996. The first prototype was delivered this week, over one year past the due date. It does not meet its specifications.

  The second prototype, a linear array of four sensors on a single lithium niobate substrate, was to be delivered in October, 1996. That is now nine months past the due date and is unacceptable. As far as delivery of the third and fourth prototype, we have been informed by you that we can no longer expect delivery. This is in material breach of the contract.
- **Failure to Fabricate an Active Bias Sensor**

  In June 1996, Srico and Aztec had agreed that Srico would fabricate an active bias sensor. In November 1996, however, Srico informed Aztec that it had in fact been fabricating a passive bias sensor, a fact to which Srico had failed to make Aztec aware. The passive bias sensor lacks critical features of an active bias which are necessary to the development of inspection and testing system for the circuit cards, as set forth in the Contract. In addition, at the same time, you have indicated to us that Srico's design in fact does not work as it had specified under the Contract.
- **Inherent flaws in Srico Sensor Design**

  The temperature dependence of the Srico sensor material adversely affects the behavior of the sensor, making it unusable for ordinary usage. We understand that you are not able to reduce temperature dependence of your sensor. As a result, Srico is unable to perform under the Contract with respect to producing a sensor that would be useful within the terms of the Project. In addition, the acoustic resonances of the Srico sensor also adversely affect the behavior of the sensor making it unusable in the project. The sensors provided to Aztec by Srico in December 1996 and January 1997 for testing performed well below specifications as required under the agreement. Once again, the Srico sensor is inadequate and Aztec has to look for more reliable sensors."

D Programs

This appendix lists many of the programs created in this project. First are listed the C++ programs, then the Matlab programs.

D.1 Program `lfsr.h`

This is the main header file (See section 9.)

```
// this is lfsr.h:  it is the main header file for the similations

//-------------------------------------------------------------------------
// #defines (always use all capitals)
//-------------------------------------------------------------------------
// I always like to have TRUE and FALSE defined!
#define FALSE   0
#define TRUE    1

// magic numbers for deciding operation type
#define MAGIC_NUMBER_AND          2
#define MAGIC_NUMBER_NAND         3
#define MAGIC_NUMBER_NOR          4
#define MAGIC_NUMBER_OR           5
#define MAGIC_NUMBER_JK           6

#define MAGIC_NUMBER_NO_STATE     8

// this is for setting up the arrays
#define MAX_NUMBER_GATES        50
#define MAX_NUMBER_OUTPUTS      50
#define MAX_NUMBER_LFSR         100

#define BIG_NUMBER           9999999
#define BIG_NEG_NUMBER      -9999999
#define MAX_UPDATE_NUMBER       100

//-------------------------------------------------------------------------
// classes (defined elsewhere)
//-------------------------------------------------------------------------
class LFSR_CLASS {
// here are the public parts of the package
public:
// Member Functions
LFSR_CLASS (int);
int         length()                { return lfsrLength; }
int         output()                { return Next_Output_Value; }
void        describe();
void        iterate();
void        print_state();
void        print_taps();
void        set_length(int  );
void        set_state( int *);
void        set_state_zero( );
void        set_taps(  int *);
~LFSR_CLASS() { };
// here are the private parts of the package
protected:
// Data Members
int         lfsrTapVector[100];
int         lfsrStateVector[100];
int         lfsrLength;
int         Next_Input_Value;
int         Next_Output_Value;
// Member Functions
void        find_next_input_value();
};

//-------------------------------------------------------------------------
// structures
//-------------------------------------------------------------------------
typedef struct {
  float score;
  int   length;
  int   number_state;
  int   number_taps;
  int   vec_state[100];
  int   vec_taps[100];
  int   have_score;
  int   generation_number;
  int   iteration_number;
} lfsr_specific;

//-------------------------------------------------------------------------
// subroutines (defined elsewhere)
//-------------------------------------------------------------------------
float  rand_unif_01();
int    i_min(       int, int);
int    Function_ALL( int, int, int);
int    Function_AND( int, int);
int    Function_JK(  int, int, int);
int    Function_NAND(int, int);
int    Function_NOR( int, int);
int    Function_OR(  int, int);
int    binary_vector_to_int(int, int *);
int    integer_power(int, int);
void   compute_optimization_score();
void   copy_present_state_to_last_state();
void   create_circuit();
void   describe_specific_lfsr(lfsr_specific *);
void   display_state_values();
void   initialize_circuit();
void   int_to_binary_vector(int, int *, int *);
void   iterate_optimization_of_system(lfsr_specific *);
void   iterate_system(LFSR_CLASS);
void   print_out_device_vector();
void   run_system_with_single_lfsr(lfsr_specific *);
void   step_system_from_new_input (int *);
void   test_routines();
```

D.2 Program main.h

This header file is only used by the main routine. It define global variables. (See section 9.)

```
//-------------------------------------------------------------------------
// global variables
//-------------------------------------------------------------------------

int     G_change_state[MAX_NUMBER_GATES][MAX_NUMBER_OUTPUTS];
int     G_line_values[MAX_NUMBER_GATES][MAX_NUMBER_OUTPUTS];
int     G_state[MAX_NUMBER_GATES][MAX_NUMBER_OUTPUTS];
int     G_line_values_last[MAX_NUMBER_GATES][MAX_NUMBER_OUTPUTS];
              // state-[gate level number]-[output line]

float   G_score;
int     G_current_device_level;
int     G_current_device_number;
int     G_device_vector[MAX_NUMBER_GATES];
int     G_max_time_steps;
int     G_number_device_levels;
int     G_number_inputs;
int     G_number_lfsr;
int     G_show_detail;
int     G_iteration_number;
```

D.3 Program extern.h

This header file is used by all routines except the main routine. It define global variables. (See section 9.)

```
//-------------------------------------------------------------------------
// global variables
//-------------------------------------------------------------------------

extern int     G_change_state[MAX_NUMBER_GATES][MAX_NUMBER_OUTPUTS];
extern int     G_line_values[MAX_NUMBER_GATES][MAX_NUMBER_OUTPUTS];
extern int     G_state[MAX_NUMBER_GATES][MAX_NUMBER_OUTPUTS];
extern int     G_line_values_last[MAX_NUMBER_GATES][MAX_NUMBER_OUTPUTS];
                 // state-[gate level number]-[output line]

extern float   G_score;
extern int     G_current_device_level;
extern int     G_current_device_number;
extern int     G_device_vector[MAX_NUMBER_GATES];
extern int     G_max_time_steps;
extern int     G_number_device_levels;
extern int     G_number_inputs;
extern int     G_number_lfsr;
extern int     G_show_detail;
extern int     G_iteration_number;
```

D.4 Program main.c

This file contains the main driving routine. (See section 9.)

```
// This is main.c: the main driving routine for the circuit simulation
//                 and LFSR optimization

//------------------------------------------------------------------------
// required includes
//------------------------------------------------------------------------
#include <stream.h>
#include "lfsr.h"
#include "main.h"

//------------------------------------------------------------------------
//------------------------------------------------------------------------
// this is the main routine
//------------------------------------------------------------------------
//------------------------------------------------------------------------

main()
{

// declare variables
LFSR_CLASS     master_lfsr(1);
lfsr_specific array_lfsr[MAX_NUMBER_LFSR];
int           i, number_optimization_steps;

// Maybe test some of the routines
//test_routines();

// Date stamp and time stamp the output
cout << "\n\n*******************\n";
cout << "Date is "  << __DATE__ << endl;
cout << "Time is "  << __TIME__ << endl;
cout << endl;

// Initialize some things
G_iteration_number=0;

// Define the circuit parameters
G_number_inputs= 11;
G_number_device_levels= 20;
// Create the circuit
create_circuit();

// Define the simulation parameters
G_max_time_steps= 250;
number_optimization_steps=1000;

// debugging values
//G_number_inputs=  3;
//G_number_device_levels= 13;
//G_max_time_steps= 20;
//number_optimization_steps=1;

// Define the seed LFSR_CLASS's that will be the first
// patterns used in the optimization calculation.
G_number_lfsr= 2;
for ( i=0; i<G_number_lfsr; ++i)
  {
   array_lfsr[i].length=       13;
   array_lfsr[i].number_state= 100*(i+1);
   array_lfsr[i].number_taps=  100*(i+1)+1;
   array_lfsr[i].have_score=   FALSE;
   array_lfsr[i].generation_number= -1;
   array_lfsr[i].iteration_number=   0;
  }

//decide on the amount of detail
G_show_detail= TRUE;

// run a bunch of optimization steps
for ( i=0; i<number_optimization_steps; ++i)
{
  // increment the iteration counter
  G_iteration_number += 1;

  // decide on whether or not the final states should be shown
  if ( i > 0 ) G_show_detail=FALSE;

  // run the optimization routine one iteration
  iterate_optimization_of_system(array_lfsr);
}

cout << "\n\nNormal termination of program\n";
cout << "*******************\n";
}
```

D.5 Program lfsr.c

This file defines the LFSR class. (See section 9.)

```
// This is lfsr.c

//------------------------------------------------------------------------
// required includes
//------------------------------------------------------------------------
#include <stream.h>
#include "lfsr.h"
#include "extern.h"

//------------------------------------------------------------------------
//------------------------------------------------------------------------
// class LFSR_CLASS defined here
//------------------------------------------------------------------------
//------------------------------------------------------------------------

//------------------------------------------------------------------------
// Creator
//------------------------------------------------------------------------
LFSR_CLASS::LFSR_CLASS (int input_size=1)
{
    int DIAG=FALSE;
    int i;
    if ( DIAG ) cout << "D Entering LFSR_CLASS::LFSR_CLASS(" << input_size << ")\n";

    // check if the input value is reasonable
    if ( input_size < 1 )
    {
      cout << "Non-fatal error: " << input_size << " is not a reasonable length for a LFSR_CLASS\n";
      return;
    }

    // initialize the state vector and the tap vector
    lfsrLength= input_size;
    for ( i=0; i<lfsrLength; ++i)
        {
            lfsrTapVector[i]= 1;
            lfsrStateVector[i]= 1;
        }

    // determine the next output value
    Next_Output_Value= lfsrStateVector[lfsrLength-1];

    // compute the next value to be input
    find_next_input_value();

    if ( DIAG ) cout << "D Leaving LFSR_CLASS::LFSR_CLASS\n";
}

//------------------------------------------------------------------------
// Set the length of the LFSR_CLASS and reset the parameters
//------------------------------------------------------------------------
void LFSR_CLASS::set_length (int input_size=1)
{
    int DIAG=FALSE;
    int i;
    if ( DIAG ) cout << "D Entering LFSR_CLASS::set_length(" << input_size << ")\n";

    // check if the input value is reasonable
    if ( input_size < 1 )
    {
      cout << "Non-fatal error: " << input_size << " is not a reasonable length for a LFSR_CLASS\n";
      input_size=1;
    }

    // initialize the state vector and the tap vector
    lfsrLength= input_size;
    for ( i=0; i<lfsrLength; ++i)
        {
            lfsrTapVector[i]= 1;
            lfsrStateVector[i]= i;
        }

    // determine the next output value
    Next_Output_Value= lfsrStateVector[lfsrLength-1];

    // compute the next value to be input
    find_next_input_value();

    if ( DIAG ) cout << "D Leaving LFSR_CLASS::set_length\n";
}

//------------------------------------------------------------------------
// This determines the LFSR_CLASS
//------------------------------------------------------------------------
void LFSR_CLASS::describe()
{
    int sum;

    cout << "Describing LFSR_CLASS";
    cout << " length=" << length();

    cout << " taps=";
    print_taps();
    sum= binary_vector_to_int(lfsrLength, lfsrTapVector);
    cout << " (" << sum << ")";

    cout << " next_input=" << Next_Input_Value;

    cout << " state=";
    print_state();
    sum= binary_vector_to_int(lfsrLength, lfsrStateVector);
    cout << " (" << sum << ")";

    cout << " output=" << output();
```

```
    cout << endl;
}

//-------------------------------------------------------------------------
// This iterates the LFSR_CLASS one step
//-------------------------------------------------------------------------
void LFSR_CLASS::iterate()
{
    int DIAG=FALSE;
    int i;
    if ( DIAG ) cout << "D Entering LFSR_CLASS::iterate()\n";
    if ( DIAG ) cout << "D lfsrLength=" << lfsrLength << endl;

    // shift everything one location
    for ( i=lfsrLength-1; i>0; --i) lfsrStateVector[i]= lfsrStateVector[i-1];

    // the first element is brought in from the
    // (already computed) inner product
    lfsrStateVector[0]= Next_Input_Value;

    // determine the next output value
    Next_Output_Value= lfsrStateVector[lfsrLength-1];

    // compute the next input value by performing an inner product
    find_next_input_value();
}

//-------------------------------------------------------------------------
// This determines the next value to be input
//-------------------------------------------------------------------------
void LFSR_CLASS::find_next_input_value()
{
    int i;
    Next_Input_Value=0;
    // take an inner product
    for ( i=0; i<lfsrLength; ++i)
        {
            Next_Input_Value += lfsrTapVector[i]*lfsrStateVector[i];
        }
    // now reduce mod 2
    // cout << "  Next_Input_Value=" << Next_Input_Value << endl;
    Next_Input_Value = (Next_Input_Value % 2);
}

//-------------------------------------------------------------------------
// This prints out the state vector
//-------------------------------------------------------------------------
void LFSR_CLASS::print_state()
{
    int DIAG=FALSE;
    int i;
    if ( DIAG ) cout << "D Entering LFSR_CLASS::print_state()\n";
    if ( DIAG ) cout << "D lfsrLength=" << lfsrLength << endl;

    for ( i=0; i<lfsrLength; ++i) cout << lfsrStateVector[i];

    if ( DIAG ) cout << "D Leaving LFSR_CLASS::print_state\n";
}

//-------------------------------------------------------------------------
// This prints out the tap vector
//-------------------------------------------------------------------------
void LFSR_CLASS::print_taps () {
    for ( int i=0; i<lfsrLength; ++i) cout << lfsrTapVector[i]; }

//-------------------------------------------------------------------------
// This sets the tap vector
//-------------------------------------------------------------------------
void LFSR_CLASS::set_taps (int *vector) {
    for ( int i=0; i<lfsrLength; ++i) lfsrTapVector[i]=vector[i]; }

//-------------------------------------------------------------------------
// This sets the state vetor
//-------------------------------------------------------------------------
void LFSR_CLASS::set_state (int *vector) {
    for ( int i=0; i<lfsrLength; ++i) lfsrStateVector[i]=vector[i]; }

//-------------------------------------------------------------------------
// This sets the state vector to all zeros
//-------------------------------------------------------------------------
void LFSR_CLASS::set_state_zero () {
    for ( int i=0; i<lfsrLength; ++i) lfsrStateVector[i]=0; }
```

D.6 Program system.c

This file has all the routines that define the circuit to be simulated, how the LFSR's are to be connected to the circuit, how the circuit is to be scored, and how new LFSR are created from old LFSR's. (See section 9.)

```
// This is system.c
// This contains routines that specify the system and its evolution

//-------------------------------------------------------------------------
// required includes
//-------------------------------------------------------------------------
#include <stream.h>
#include "lfsr.h"
#include "extern.h"

//-------------------------------------------------------------------------
//-------------------------------------------------------------------------
// subroutines
//-------------------------------------------------------------------------
//-------------------------------------------------------------------------

//-----------------------------------------------------------
// This routine is input 2 specific LFSR's (*in0 and *in1), and an index
// Based on the index, it trys to create a new LFSR (*out)
// from those old LFSR's

void create_new_lfsr( lfsr_specific *in0, lfsr_specific *in1,
  lfsr_specific *out, int index)
{
// declare variables
int  modulus, rand1, rand2;
int  DIAG=FALSE;
int  length0, num_state0, num_taps0;
int  length1, num_state1, num_taps1;

// (maybe) show some of the variables
if ( DIAG )
  {
cout << "D Entering create_new_lfsr";
cout << "  with index=" << index << " and " << endl;
cout << "  LFSR-in0" << endl;
describe_specific_lfsr( in0 );
cout << "  LFSR-in1" << endl;
describe_specific_lfsr( in1 );
cout << endl;
  }

// pull off some parameters that may be useful

length0=    in0->length;
num_state0= in0->number_state;
num_taps0=  in0->number_taps;

length1=    in1->length;
num_state1= in1->number_state;
num_taps1=  in1->number_taps;

modulus= integer_power(2,length0)-1;

//grab some integer-valued random numbers (between 0 and modulus)
rand1=((int) modulus*rand_unif_01());
rand2=((int) modulus*rand_unif_01());

//now combine these two input LFSR's, based on the value of index
switch (index) {

case 0:
    out->length=       length0;
    out->number_state= num_state0;
    out->number_taps=  (num_taps0+num_taps1) % modulus;
    out->have_score=   FALSE;
    break;

case 1:
    out->length=       length0;
    out->number_state= num_state0;
    out->number_taps=  rand1;
    out->have_score=   FALSE;
    break;

case 2:
    out->length=       length0;
    out->number_state= num_state1;
    out->number_taps=  (num_taps0+num_taps1) % modulus;
    out->have_score=   FALSE;
    break;

case 3:
    out->length=       length0;
    out->number_state= num_state1;
    out->number_taps=  rand1;
    out->have_score=   FALSE;
    break;

case 4:
    out->length=       length0;
    out->number_state= (num_state0+num_state1) % modulus;
    out->number_taps=  num_taps0;
    out->have_score=   FALSE;
    break;

case 5:
    out->length=       length0;
    out->number_state= rand1;
    out->number_taps=  num_taps0;
```

```
    out->have_score=   FALSE;
    break;

case 6:
    out->length=       length0;
    out->number_state= (num_state0*num_state1) % modulus;
    out->number_taps=  num_taps1;
    out->have_score=   FALSE;
    break;

case 7:
    out->length=       length0;
    out->number_state= rand1;
    out->number_taps=  num_taps1;
    out->have_score=   FALSE;
    break;

default:
    // The defaul option is to create random entries
    // of the same length

// cout << "WARNING" << endl;
// cout << "WARNING in create_new_lfsr" << cout;
    // cout << " do not know index=" << index << endl;
// cout << "WARNING" << endl;
// cout << "Will just create random state and tap values" << endl;

    out->length=       length0;
    out->number_state= rand1;
    out->number_taps=  rand2;
    out->have_score=   FALSE;
}

// Set the flag indicated how and when the new LFSR was generated.
// This is useful for determining the best combination techniques.
out->generation_number= index;
out->iteration_number=  G_iteration_number;

//perhaps describe the newly created LFSR
if ( DIAG )
  {
cout << "D Leaving create_new_lfsr with";
cout << "  LFSR-out" << endl;
describe_specific_lfsr( out );
cout << endl;
  }
}

//-------------------------------------------------------------------------
// this create the circuit to be tested

void create_circuit() {
// declare variables
int  i;
int  DIAG=TRUE;

// (maybe) show some of the variables
if ( DIAG )
  {
cout << "D Entering create_circuit with:" << endl;
cout << "  G_number_device_levels=" << G_number_device_levels << endl;
cout << "  G_number_inputs=" << G_number_inputs << endl;
cout << endl;
  }

//define the system of devices
for ( i=i; i<G_number_device_levels; ++i)
  {
    G_device_vector[i]= MAGIC_NUMBER_NAND;
  }
G_device_vector[ 5]= MAGIC_NUMBER_JK;
G_device_vector[ 7]= MAGIC_NUMBER_JK;
G_device_vector[ 8]= MAGIC_NUMBER_JK;
G_device_vector[10]= MAGIC_NUMBER_JK;
G_device_vector[12]= MAGIC_NUMBER_JK;
G_device_vector[14]= MAGIC_NUMBER_OR;
G_device_vector[15]= MAGIC_NUMBER_OR;
G_device_vector[17]= MAGIC_NUMBER_JK;
print_out_device_vector();

// i=42 order_lfsr[i]=93 array_lfsr[order_lfsr[i]].score= 426.32(13,  300, 1411)3
// i=43 order_lfsr[i]=95 array_lfsr[order_lfsr[i]].score= 426.32(13,  300, 1411)3
// i=44 order_lfsr[i]= 8 array_lfsr[order_lfsr[i]].score= 331.20(13,  300, 7235)1
// i=45 order_lfsr[i]= 1 array_lfsr[order_lfsr[i]].score= 331.20(13,  300, 7235)1
// i=46 order_lfsr[i]=46 array_lfsr[order_lfsr[i]].score= 331.20(13,  300, 7235)1
// i=47 order_lfsr[i]=48 array_lfsr[order_lfsr[i]].score= 331.20(13,  300, 7235)1
// i=48 order_lfsr[i]=49 array_lfsr[order_lfsr[i]].score= 331.20(13,  300, 7235)1
}

//-------------------------------------------------------------------------
// this initializes the circuit to be tested

void initialize_circuit() {
// declare variables
int  i, j;
int  DIAG=TRUE;

//initialize the entire array
for ( i=0; i<G_number_inputs; ++i)
  {
    for ( j=0; j<G_number_device_levels; ++j)
      {
        // put random (useless) values on the lines
        G_line_values[j][i]= 9;
        // put random values in for states
        G_state[j][i]= 1;
        // put zeros in for change of states
        G_change_state[j][i]= 0;
      }
  }

// for those devices that do not maintain a state
// fill in the state array appropriately

for ( j=1; j<G_number_device_levels; ++j)
  {
if ( G_device_vector[j] != MAGIC_NUMBER_JK )
  {
for ( i=0; i<G_number_inputs; ++i)
  G_state[j][i]=MAGIC_NUMBER_NO_STATE;
  }
  }

// (maybe) show the initial data
if ( DIAG && G_show_detail)
  {
cout << "\nD initial data (SHOULD RANDOMIZE INITIAL STATE INFORMATION!)\n";
display_state_values();
  }
}

//-------------------------------------------------------------------------
// This takes the array of changed values and
//      computes the score (this is what we attempt to optimize).
// This routines should be changed if a new metric becomes a better
//      measure of entire board electrical activity.

void compute_optimization_score() {
int   DIAG=FALSE;
int   i, j, sum, min, count_zero;

// initialization
sum= 0;
min= BIG_NUMBER;
count_zero= 0;

// loop through all the inputs
for ( i=0; i<G_number_inputs; ++i)
  {
    // loop through all of the device
    for ( j=1; j<G_number_device_levels; ++j)
      {
        sum += G_change_state[j][i];
min = i_min(min, G_change_state[j][i] );
if ( G_change_state[j][i]==0 ) count_zero += 1;
      }
  }

// compute the score to be the average number of changes times the
// minimun number of changes
if ( count_zero==0 )
  G_score= ((float) min) * ((float) sum)
         / ((float) (G_number_inputs*G_number_device_levels));
else
  G_score = -count_zero;

if ( DIAG )
  {
cout << "D sum="        << sum        << endl;
cout << "D min="        << min        << endl;
cout << "D count_zero=" << count_zero << endl;
cout<< "Leaving compute_optimization_score with score=" << G_score << endl;
  }
}

//-------------------------------------------------------------------------
// This iterates a single optimization step.
// All the LFSR's which have not been simulated are simulated here.
// All known scores are ordered.
// The best two LFSR's are used to create new LFSR's.

void iterate_optimization_of_system(lfsr_specific *array_lfsr)
{
// declare variables
int         i, j, temp, flag_switch, order_lfsr[MAX_NUMBER_LFSR];
int         length0, num_state0, num_taps0;
int         length1, num_state1, num_taps1;
int         best0, best1, place;
int         index0, index1, top_number;
int         new_location[MAX_UPDATE_NUMBER], update_number, flag_unique;
int         DIAG=TRUE, DDIAG=FALSE;

// diagnostics
if ( DDIAG )
  {
cout << "DD entering iterate_optimization_of_system with G_number_lfsr=" << G_number_lfsr << endl;
  }

// run the system for all present LFSR's, and compute the score of each
for ( i=0; i<G_number_lfsr; ++i)
  {
if ( array_lfsr[i].have_score==FALSE )
      {
        // verify that this LFSR is a new one
        flag_unique= TRUE;
        for ( j=0; j<G_number_lfsr; ++j)
          {
            if (    ( i!=j )
                 && ( array_lfsr[j].have_score==TRUE                          )
                 && ( array_lfsr[i].length==      array_lfsr[j].length       )
                 && ( array_lfsr[i].number_state==array_lfsr[j].number_state )
                 && ( array_lfsr[i].number_taps== array_lfsr[j].number_taps  ) )
```

```
            {
              // Have re-discovered a previous LFSR,
              //      no need to simulate it
              // cout << "HAVE A DUPLICATED LFSR  i="<<i<<" j="<<j<< endl;
              array_lfsr[i].score=      BIG_NEG_NUMBER;
              array_lfsr[i].have_score= TRUE;
              flag_unique=              FALSE;
            }
        }
      if ( flag_unique==TRUE ) run_system_with_single_lfsr( &array_lfsr[i] );
    }
  }

// diagnostics
if ( DDIAG )
{
 cout << "\nD Here are all the LFSR's, and their scores:" << endl;
 for ( i=0; i<G_number_lfsr; ++i)
   {
 cout << "MAIN Using length="     << array_lfsr[i].length;
 cout << ", number_state="        << array_lfsr[i].number_state;
 cout << ", number_taps="         << array_lfsr[i].number_taps;
 cout << "  we obtain the score " << array_lfsr[i].score;
 cout << endl;
   }
}

// order all the LFSR's based on their score
//initialize the ordered list
for ( i=0; i<G_number_lfsr; ++i) order_lfsr[i]=i;
// use a slow bubble sort
for ( i=1; i<G_number_lfsr; ++i)
  {
    flag_switch= FALSE;
    for ( j=0; j<i; ++j)
      {
        if ( array_lfsr[order_lfsr[i]].score > array_lfsr[order_lfsr[j]].score )
          {
            // cout << "Switching " << array_lfsr[order_lfsr[i]].score;
            // cout << " and "      << array_lfsr[order_lfsr[j]].score << endl;
            temp=          order_lfsr[i];
            order_lfsr[i]= order_lfsr[j];
            order_lfsr[j]= temp;
          }
      }
  }

cout << endl;
cout << "The ordered list of LFSR'd is:" << endl;
// for ( i=0; i<G_number_lfsr; ++i)
for ( i=0; i<i_min(20,G_number_lfsr); ++i)
  {
printf("i=%2d",i);
printf(" order_lfsr[i]=%2d", order_lfsr[i]);
printf(" array_lfsr[order_lfsr[i]].score=%7.2f",array_lfsr[order_lfsr[i]].score);
printf("(%2d,%5d,%5d)", array_lfsr[order_lfsr[i]].length,
                        array_lfsr[order_lfsr[i]].number_state,
                        array_lfsr[order_lfsr[i]].number_taps);
cout << array_lfsr[order_lfsr[i]].generation_number;
cout << "/";
cout << array_lfsr[order_lfsr[i]].iteration_number;
cout << endl;
  }

// now that we have the LFSR's ordered, let's use a genetic algorithm to
// compute some more possabilities
// Take the top two best ones... and make 'update_number' new ones
//
// No....take 2 from among the top 6
//
// WAS
//  length0=   array_lfsr[order_lfsr[0]].length;
//  num_state0=array_lfsr[order_lfsr[0]].number_state;
//  num_taps0= array_lfsr[order_lfsr[0]].number_taps;
//
//  lengthi=   array_lfsr[order_lfsr[i]].length;
//  num_statei=array_lfsr[order_lfsr[i]].number_state;
//  num_tapsi= array_lfsr[order_lfsr[i]].number_taps;

top_number6;
index0=((int) top_number*rand_unif_01());
index1=((int) top_number*rand_unif_01());
// if ( index0==index1 ) (index1 = (index1+1) % top_number)
length0=   array_lfsr[order_lfsr[index0]].length;
num_state0=array_lfsr[order_lfsr[index0]].number_state;
num_taps0= array_lfsr[order_lfsr[index0]].number_taps;

lengthi=   array_lfsr[order_lfsr[index1]].length;
num_statei=array_lfsr[order_lfsr[index1]].number_state;
num_tapsi= array_lfsr[order_lfsr[index1]].number_taps;

if ( DDIAG )
{
cout << "DD Combining the LFSR's with:" << endl;
cout << " Length=" << length0 << "  state=" << num_state0 << " taps=" << num_taps0 << endl;
cout << " Length=" << lengthi << "  state=" << num_statei << " taps=" << num_tapsi << endl;
}

// this is the number of new LFSR's to be introduced at each step
update_number= 9;

// make sure that there is enough array space for new LFSR's
if ( G_number_lfsr+update_number+1 < MAX_NUMBER_LFSR )
{
  // fill up the locations where the new LFSR should go
  for ( i=0; i<update_number; ++i)
{
  new_location[i]= G_number_lfsr;
  G_number_lfsr += 1;
}
}
else
{
  // Have run out of space for new LFSR's,
  //      will re-use some of the old locations

  // cout << "****************************************" << endl;
  // cout << "HAVE RUN OUT OF LIST SPACE FOR NEW LFSR's" << endl;
  // cout << "     G_number_lfsr=" << G_number_lfsr      << endl;
  // cout << "WILL RE-USE SOME OF THE OLD LOCATIONS"     << endl;
  // cout << "****************************************" << endl;
  for ( i=0; i<update_number; ++i)
{
  new_location[i]= order_lfsr[G_number_lfsr-i-1];
}
}

// Create new LFSR's based on the two best LFSR's
// Combine them according to a specific index
best0= order_lfsr[0];
besti= order_lfsr[1];

for ( i=0; i<update_number; ++i)
{
  place= new_location[i];
  create_new_lfsr( &array_lfsr[best0], &array_lfsr[besti],
   &array_lfsr[place], i);
}

// now return, without simulating these new LFSR_CLASS's
if ( DDIAG )
  {
cout << "DD leaving iterate_optimization_of_system with G_number_lfsr=" << G_number_lfsr << endl;
  }
}

//-----------------------------------------------------------------------
// This iterates the digital system for a special forcing strategy.
// Here, all inputs are connected to a single LFSR.
// This routine can be replaced by other routines that
//      connect the outputs of LFSR(s) to the circuit inputs.

void run_system_with_single_lfsr(lfsr_specific *specific_lfsr)
{
// declare variables
LFSR_CLASS input_lfsr(1);
int        i, dummy;
int        DIAG=FALSE, DDIAG=FALSE;

// define the vectors used for the input LFSR_CLASS
for ( i=0; i<specific_lfsr->length; ++i) specific_lfsr->vec_state[i]=0;
for ( i=0; i<specific_lfsr->length; ++i) specific_lfsr->vec_taps [i]=0;
int_to_binary_vector(specific_lfsr->number_state, &dummy,
 specific_lfsr->vec_state);
int_to_binary_vector(specific_lfsr->number_taps,  &dummy,
 specific_lfsr->vec_taps);

// set the parameters for the input LFSR_CLASS
input_lfsr.set_length(specific_lfsr->length);
input_lfsr.set_state( specific_lfsr->vec_state);
input_lfsr.set_taps(  specific_lfsr->vec_taps);

if ( DIAG )
  {
input_lfsr.describe();
cout << endl;
  }

// iterate the system
iterate_system(input_lfsr);

// store the score away with this LFSR_CLASS
specific_lfsr->have_score= TRUE;
specific_lfsr->score= G_score;

if ( DDIAG )
  {
cout << "DD   Using length="     << specific_lfsr->length;
cout << ", number_state="        << specific_lfsr->number_state;
cout << ", number_taps="         << specific_lfsr->number_taps;
cout << "  we obtain the score " << specific_lfsr->score;
cout << endl;
  }
}

//-----------------------------------------------------------------------
// loop through all of the devices in the system
//      and determine the outputs on all of the lines

void step_system_from_new_input (int *vector ){
int  DIAG=FALSE;
int  i, k, input1, input2, low, high, offset;

if ( DIAG )
  {
    cout << "D Entering step_system_from_new_input with ";
    for ( i=0; i<G_number_inputs; ++i) cout <<  vector[i];
    cout << endl;
  }

// load the inputs into the system
```

```
for ( i=0; i<G_number_inputs; ++i) G_line_values[0][i]= vector[i];

// iterate the system
for ( k=1; k<G_number_device_levels; ++k)
  {
    // set a useful global variable
    G_current_device_level= k;

    // loop through all of the devices
    for ( i=0; i<G_number_inputs; ++i)
      {
        G_current_device_number= i;

        // Here is where the two inputs are specified
//      for each device
// For device number j we use input lines j and j+offset
        //      (modulo G_number_inputs)
        //      where j+offset is not allowed to equal j
        // If it is, then j+offset is incremented by 1.
        low= i;
        offset= k;
        high= (i+offset+100*G_number_inputs) % G_number_inputs;
        if ( high < 0 ) printf("ERROR: negative number for input\n");
        if ( high==low ) high=(low+1) % G_number_inputs;

        input1= G_line_values[k-1][low];
        input2= G_line_values[k-1][high];
        // select the correct function
        G_line_values[k][i]=
          Function_ALL( input1, input2, G_device_vector[k]);
      }
  }
}
```

D.7 Program util.c

This file has many utility routines. (See section 9.)

```
// This is util.c


//-------------------------------------------------------------------------
// required includes
//-------------------------------------------------------------------------
#include <strean.h>
#include "lfsr.h"
#include "extern.h"

// values required for the random number generator
int G_rand_state=3;
int G_rand_a = 3;
int G_rand_modulus=2048;

//-------------------------------------------------------------------------
//-------------------------------------------------------------------------
// subroutines
//-------------------------------------------------------------------------
//-------------------------------------------------------------------------

//-----------------------------------------------------------------
int Function_AND(int a, int b)
{
int result, DIAG=FALSE;
result= (a && b);
if ( DIAG ) cout << "D Function_AND: " << a << " AND " << b << " ==>" << result << endl;
return result;
}

//-----------------------------------------------------------------
int Function_OR(int a, int b)
{
int result;
result= (a || b);
return result;
}

//-----------------------------------------------------------------
int Function_NOR(int a, int b)
{
int result;
result= !(a || b);
return result;
}

//-----------------------------------------------------------------
int Function_NAND(int a, int b)
{
int result;
result= !(a && b);
return result;
}

//-----------------------------------------------------------------
int Function_JK(int J, int K, int old_state)
{
int result, DIAG=FALSE;
if ( old_state==0 )
  {
    if ( J==0 ) { result= 0; }
    else        { result= 1; }
  }
else
  {
    if ( K==0 ) { result= 1; }
    else        { result= 0; }
  }
if ( DIAG ) {
  cout << "D Function_JK: J=" << J << " K=" << K;
  cout << " old_state=" << old_state << " result=" << result << endl;
}
return result;
}

//-----------------------------------------------------------------
void display_state_values() {
//display the entire array of values
int i, j;

//put a header line in
cout << "           ";
for ( j=0; j<G_number_device_levels; ++j) { cout << (j % 10); }
cout << "           ";
for ( j=1; j<G_number_device_levels; ++j) { cout << (j % 10); }
cout << "           ";
for ( j=1; j<G_number_device_levels; ++j) { cout << (j % 10); }
cout << "\n";

for ( i=0; i<G_number_inputs; ++i)
  {

    // show the line values
    printf("...line %2d: ", i);
    for ( j=0; j<G_number_device_levels; ++j)
      {
if ( (G_line_values[j][i]==TRUE) || (G_line_values[j][i]==FALSE) )
  {
cout << G_line_values[j][i];
  }
else
  {
cout << " ";
  }
      }

    // show the state values
    cout << " states: ";
    for ( j=1; j<G_number_device_levels; ++j)
      {
        if ( G_state[j][i]==MAGIC_NUMBER_NO_STATE ) { cout << "-"; }
        else                                        { cout << G_state[j][i]; }
      }

    // show the changed state values
    cout << " changed: ";
    for ( j=1; j<G_number_device_levels; ++j)
      {
        if ( G_change_state[j][i] >9 )
          {
            cout << "*";
          }
        else
          {
            cout << G_change_state[j][i];
          }
      }
    cout << "\n";
  }
}

//-----------------------------------------------------------------------
void print_out_device_vector(){
// print out the system of devices
int i;

cout << "The devices used in this simulated circuit are:\n";
// note that the index starts at 1
for ( i=1; i<G_number_device_levels; ++i)
  {
    cout << "...device that produced level ";
    printf("%2d", i);
    cout << " is a(n) ";
    switch( G_device_vector[i] ) {
    case MAGIC_NUMBER_AND:
      cout << "AND  gate";
      break;
    case MAGIC_NUMBER_NAND:
      cout << "NAND gate";
      break;
    case MAGIC_NUMBER_OR:
      cout << "OR   gate";
      break;
    case MAGIC_NUMBER_NOR:
      cout << "NOR  gate";
      break;
```

```
    case MAGIC_NUMBER_JK:
      cout << "JK   flip-flop";
      break;
    default:
      cout << "SERIOUS ERROR!";
    }
    cout << endl;
  }
}

//---------------------------------------------------------------------
int  Function_ALL( int input1, int input2, int function_type)
{
  int result, old_state;

  switch(function_type) {
  case MAGIC_NUMBER_AND:
    result = Function_AND(input1, input2);
    G_state[G_current_device_level][G_current_device_number]= MAGIC_NUMBER_NO_STATE;
    break;

  case MAGIC_NUMBER_NAND:
    result = Function_NAND(input1, input2);
    G_state[G_current_device_level][G_current_device_number]= MAGIC_NUMBER_NO_STATE;
    break;

  case MAGIC_NUMBER_OR:
    result = Function_OR(input1, input2);
    G_state[G_current_device_level][G_current_device_number]= MAGIC_NUMBER_NO_STATE;
    break;

  case MAGIC_NUMBER_NOR:
    result = Function_NOR(input1, input2);
    G_state[G_current_device_level][G_current_device_number]= MAGIC_NUMBER_NO_STATE;
    break;

  case MAGIC_NUMBER_JK:
    old_state= G_state[G_current_device_level][G_current_device_number];
    result = Function_JK(input1, input2, old_state);
    G_state[G_current_device_level][G_current_device_number]=result;
    break;

  default:
    cout << "ERRDR!!! Unknown circuit operation";
  }
  return result;
}


//---------------------------------------------------------------------
int binary_vector_to_int(int length, int *vector) {

int DIAG=FALSE;
int power2, result, i;

if ( DIAG ) {
    cout << "D entering binary_vector_to_int" << endl;
}

    // turn the state into a decimal number and display
    result=  vector[0];
    power2= 1;
    for ( i=1; i<length; ++i)
        {
            power2 *= 2;
            result += vector[i]*power2;
        }
if ( DIAG ) {
    cout << "D leaving binary_vector_to_int with result=" << result << endl;
}

return result;
}


//---------------------------------------------------------------------
void copy_present_state_to_last_state() {
int i, j;
// loop through all entries
for ( i=0; i<G_number_inputs; ++i)
  {
    //cout << "  i=" << i << endl;
    for ( j=0; j<G_number_device_levels; ++j)
      {
        //cout << "     j=" << j;
        //cout << " G_state[j][i]="      << G_state[j][i];
        //cout << " G_line_values_last[j][i]=" <<  G_line_values_last[j][i];
        //cout << endl;

        // modify the change state array if a change has occurred
        if ( G_line_values[j][i] != G_line_values_last[j][i] )
          {
            G_change_state[j][i] += 1;
          }

        G_line_values_last[j][i] = G_line_values[j][i];
      }
  }
}

//---------------------------------------------------------------------
// turn the integer input value into a binary vector

void int_to_binary_vector(int input, int *length, int *vector) {
int DIAG=FALSE;
int i, remainder, temp;

if ( DIAG ) {
    cout << "D entering int_to_binary_vector with input=" << input << endl;
}

*length= 0;
temp= input;

while ( temp > 0 )
  {
remainder= (temp % 2);
temp= (temp-remainder)/2;
vector[*length]= remainder;
*length += 1;
  }

if ( DIAG )
  {
    cout << "D leaving int_to_binary_vector with *length=" << *length << endl;
    cout << "D                  and vector= ";
for ( i=0; i<*length; ++i) cout << vector[i];
cout << endl;
  }
}


//---------------------------------------------------------------------
// This iterates the circuit using the output from a single LFSR.
// It iterates the system      G_max_time_steps     steps.
// It then computes the resultant score.


void iterate_system(LFSR_CLASS master_lfsr) {
// declare variables
int   DIAG=FALSE;
int   i, j, time_step, vec[100];

// (maybe) show some of the variables
if ( DIAG )
  {
cout << "D Entering iterate_system with:" << endl;
cout << "  G_max_time_steps=" << G_max_time_steps << endl;
cout << "  G_number_device_levels=" << G_number_device_levels << endl;
cout << "  G_number_inputs=" << G_number_inputs << endl;
cout << endl;
  }

// initialize the circuit values (states, lines, number of changes)
initialize_circuit();

// step the system for lots of time values
for ( time_step=0; time_step<G_max_time_steps; ++time_step)
  {

    // define the inputs (should be the outputs of one or more LFSR's)
    for ( i=0; i<G_number_inputs; ++i)
      {
        master_lfsr.iterate();
        vec[i]= master_lfsr.output();
      }

    // iterate the system
    step_system_from_new_input ( vec );

    // for the first time step, initialize the G_line_values_last array
    if (time_step==0 )
      {
        for ( i=0; i<G_number_inputs; ++i)
          {
            for ( j=0; j<G_number_device_levels; ++j)
              {
                // put present values in for last values
               G_line_values_last[j][i]= G_state[j][i];
              }
          }
      }

    // copy the present state to the old state, and update the change state
    copy_present_state_to_last_state();

    // maybe show all the variables
    if ( G_show_detail )
      {
        // Myabe qualify the printing of all details on the number of the time step
        // if ( (time_step % 10)==0 )
        // if ( i==1 )
        if ( (time_step==G_max_time_steps-1) )
          {
            cout << "After " << time_step << " time steps\n";
            display_state_values();

           // show the score used for optimization
           compute_optimization_score();
           // cout << "score (so far)=" << G_score << endl;
          }
      }
  }

compute_optimization_score();
}

//---------------------------------------------------------------------
int i_min(int A, int B)
{
if ( A<B ) return A;
else       return B;
```

```
}

//---------------------------------------------------------------------
int integer_power(int base, int power)
{
int i, result;

switch ( power )
  {
  case 0:
result=1; break;

  case 1:
result=base; break;

  default:
result=base;
for ( i=0; i<power-1; ++i) result *= base;
  }

return result;
}


//---------------------------------------------------------------------
void print_logical_variable(int input) {
// this prints out "TRUE " or "FALSE" or "?    " depending on the input
  if ( input==TRUE )
cout << "TRUE ";
  else
{
  if ( input==FALSE )
cout << "FALSE";
  else
cout << "?    ";
}
}

//---------------------------------------------------------------------
void test_routines() {
// Test some of the routines

int vec[100], i, length, input, output;
int base, result;
int j, k, input1[2], input2[2];

cout << endl << endl;

cout << "\n\n****************** BEGIN TESTING OF ROUTINES\n";

cout << "Testing out the conversion of integers to and from vectors." << endl;
input=20;
length=-1;
int_to_binary_vector(input, &length, vec);
cout << "Using an input value of " << input;
cout << " the routine int_to_binary_vector yields" << endl;
cout << " length=" << length;
cout << " and vector=";
for ( i=0; i<length; ++i) cout << vec[i];
cout << endl;

output= -1;
output= binary_vector_to_int(length, vec);
cout << "Using this as input to the routine binary_vector_to_int";
cout << " results in" << endl;
cout << " the value " << output << endl;

cout << "\n\nTesting out the integer power routine." << endl;
base=2;
for ( i=0; i<25; ++i)
  {
result= integer_power(base,i);
cout << "   integer_power(" << base << "," << i << ")=" << result << endl;
  }

cout << "\n\nTesting out the logical functions." << endl;
cout << "  Input1 Input2 AND    NAND   OR     NOR" << endl;
input1[0]=TRUE;
input1[1]=FALSE;
input2[0]=TRUE;
input2[1]=FALSE;
for ( j=0; j<2; ++j)
  {
for ( k=0; k<2; ++k)
  {
cout << "  ";
print_logical_variable(input1[j]);
cout << "  ";
print_logical_variable(input2[k]);
cout << "  ";
print_logical_variable( Function_AND(input1[j], input2[k]) );
cout << "  ";
print_logical_variable( Function_NAND(input1[j], input2[k]) );
cout << "  ";
print_logical_variable( Function_OR(input1[j], input2[k]) );
cout << "  ";
print_logical_variable( Function_NOR(input1[j], input2[k]) );
cout << endl;
  }
  }

cout << "\n****************** END TESTING OF ROUTINES\n\n";
}

//---------------------------------------------------------------------
void describe_specific_lfsr( lfsr_specific *in )
{
// this describes a specific LFSR

int i;

cout << "Entering describe_specific_lfsr" << endl;
cout << "           length= " << in->length            << endl;
cout << "generation_number= " << in->generation_number << endl;
if ( in->have_score==TRUE ) cout << "          score= " << in->score << endl;
cout << "      number_state= " << in->number_state << endl;
cout << "       number_taps= " << in->number_taps  << endl;

cout << "          vec_taps= ";
for ( i=0; i<in->length; ++i) cout << in->vec_taps[i];
cout << endl;

cout << "         vec_state= ";
for ( i=0; i<in->length; ++i) cout << in->vec_state[i];
cout << endl;

cout << endl;
}

//---------------------------------------------------------------------
float rand_unif_01()
{
// This returns a random number in the range [0,1)
// Based on the technique in CRC's "Standard Math Tables", page 593
float result;
int   DIAG=FALSE;

G_rand_state *= G_rand_a;
G_rand_state = (G_rand_state % G_rand_modulus);

result= ((float) G_rand_state) / ((float) G_rand_modulus);

if ( DIAG )
{
  cout << " G_rand_a="       << G_rand_a;
  cout << " G_rand_state="   << G_rand_state;
  cout << " G_rand_modulus=" << G_rand_modulus;
  cout << " result="         << result;
  cout << endl;
}

return result;
}
```

D.8 Program `Makefile`

This re-compiles all the routines when needed and links everything together. (See section 9.)

```
SOURCES.C = lfsr.c main.c util.c system.c
OBJECTS.C = lfsr.o main.o util.o system.o
CC=       g++
CFLAGS=   -g
LDFLAGS=   -I /usr/lib/g++-include -I /usr/lib/g++-include/gen
OBJECTS=  $(OBJECTS.C)
PROGRAM=  go
LIBS=     -lm

$(PROGRAM): $(OBJECTS.C)
$(CC) -o $(PROGRAM) $(OBJECTS.C) $(CFLAGS) $(LDFLAGS) $(LIBS)

# this is a comment
# -----------------

lfsr.o:   lfsr.c    lfsr.h extern.h
g++ -c -o lfsr.o  lfsr.c  -g -I /usr/lib/g++-include -Wall

main.o:   main.c    lfsr.h main.h
g++ -c -o main.o  main.c  -g -I /usr/lib/g++-include -Wall

util.o:   util.c    lfsr.h extern.h
g++ -c -o util.o  util.c  -g -I /usr/lib/g++-include -Wall

system.o:   system.c    lfsr.h extern.h
g++ -c -o system.o  system.c  -g -I /usr/lib/g++-include -Wall
```

D.9 Program align.m

This Matlab subroutine uses the Radon transform to rotate a PCB image so that it is rectilinearally aligned. (See section 7.1.)

```
function result=align(I,num_levels)

% This functions takes an image and a number of contours
% It then aligns the image correctly

% (maybe) show input things
imagesc(I); title('Input image to align.m');
print -deps align_input.eps; figure;
% DIAG_num_levels=num_levels

% find min and max
val_min=min(min(I));
val_max=max(max(I));
delta=(val_max-val_min)/(num_levels+1);

% initialize a new image
J=zeros(size(I));
% loop for the number of levels
for loop=1:num_levels
    num= val_min + loop*delta;
    temp1= ( I > num );
    temp2= filter2( [1 -1], temp1);

    % (maybe) show each contour
    % show_it(abs(temp2),2); pause;

    % add in the contours to our running total
    J = J + abs(temp2);
end

% make sure that all entries are only 0 or 1
% (with lots of levels, or steep contours, there may be overlap)
image_contour=J - J.*(J>1) + (J>1);
% (maybe) show it
imagesc(image_contour);  title('Binary plot of contours');
print -deps align_contour.eps; figure;

% now use the radon transform to align image
% (see page 6-23 in the image toolbox manual)

% define the range of angles
delta_theta=1;
delta_low=0;
delta_high=179;
theta=delta_low:delta_theta:delta_high;
% take the radon transform
[R,xp]=radon(image_contour,theta);
% (maybe) show the results
imagesc(theta,xp,R); colormap(hot); colorbar; title('Radon 1');
print -deps align_radon1.eps; figure;
% use the standard deviation to indicate rotation angle
vec_std=std(R);
% (maybe) show the results
plot(vec_std); title('Radon 1 variance');
print -deps align_radon1_variance.eps; figure;
[val_max,val_std]=max(vec_std);
angle=theta(val_std)

% do it again with a refined scale

% define the range of angles
delta_theta=0.1;
delta_low= angle-20*delta_theta;
delta_high=angle+20*delta_theta;
theta=delta_low:delta_theta:delta_high;
% take the radon transform
[R,xp]=radon(image_contour,theta);
% (maybe) show the results
imagesc(theta,xp,R); colormap(hot); colorbar; title('Radon 2');
print -deps align_radon2.eps; figure;
% use the standard deviation to indicate rotation angle
vec_std=std(R);
% (maybe) show the results
plot(vec_std); title('Radon 2 variance');
print -deps align_radon2_variance.eps; figure;
[val_max,val_std]=max(vec_std);
angle=theta(val_std)

% rotate the image
result=imrotate(I,-angle,'nearest','crop');
% (maybe) show the final result
imagesc(result); title('final rotated image');
print -deps align_final.eps; figure;
```

D.10 Program do_align.m

This Matlab routine demostrates how to align images by use of the Radon transform (See section 7.1.)

```
% This demonstrates the use of the alignment algorithm (align.m)

% Load in one of the stripline images
load f07;
image_original=f07;
imagesc(image_original); title('original image');

% specify the number of contours to use
num_levels=10;

% rotate the initial image
rotation_angle=34.567;
image_rotated=imrotate(image_original,rotation_angle,'nearest','crop');
imagesc(image_rotated); title('rotated image');

% fix the image by use of the Radon transform
image_corrected=align(image_rotated,num_levels);
imagesc(image_correctd); title('corrected image');

% crop the image (cut off the edges)
[size_row,size_col]=size(image_corrected);
offset=10;
image_crop=image_corrected(offset:size_row-offset,offset:size_col-offset);
imagesc(image_cropped); title('cropped image');
```

D.11 Program create_log_image.m

This Matlab subroutine is needed for do_log.m. (See section 7.5.)

```
% This subroutine creates the Laplacian of Gaussian image

function create_log_image(input_image,log_size_n,log_size_alpha,range)

% define the filter
%------------------
log_filter=fspecial('log',log_size_n,log_size_alpha);

% apply the filter
%-----------------
new_image=conv2(log_filter,input_image,'valid');

% display the results
%--------------------
if ( nargin==3 )
    imagesc(new_image),colorbar;
else
    imagesc(new_image,range),colorbar;
end

return;
```

D.12 Program do_log.m

This Matlab routine demonstrates the use of the "log" filter. (See section 7.5.)

```
% load in the 3D data
%--------------------
for i=1:6,
        c=loadfiles(97102902+i);
        d_db(i,:,:)=        c(2:127,2:127);
        d_lin(i,:,:)=10.^(c(2:127,2:127)/20);
end

% show samples of the input data
%-------------------------------
imagesc(squeeze(d_lin(6,:,:))),colorbar;
print -deps out_log_fig_data_raw_lin.eps; figure;

imagesc(squeeze(d_db(6,:,:))),colorbar;
print -deps out_log_fig_data_raw_db.eps; figure;

% do the Laplacian of Gaussian filter
%------------------------------------

% LINEAR data
%------------------------------------
data_lin=squeeze(d_lin(6,:,:));

create_log_image(data_lin,12,0.5);
print -deps out_log_fig_lin_12.eps; figure;

create_log_image(data_lin,16,0.5);
print -deps out_log_fig_lin_16.eps; figure;

create_log_image(data_lin,20,0.5);
print -deps out_log_fig_lin_20.eps; figure;

create_log_image(data_lin,28,0.5);
print -deps out_log_fig_lin_28.eps; figure;

% show variation in alpha value
%------------------------------
create_log_image(data_lin,20,0.3);
print -deps out_log_fig_lin_20_3.eps; figure;

create_log_image(data_lin,20,0.55);
print -deps out_log_fig_lin_20_55.eps; figure;


% DB data
%------------------------------------
data_db=squeeze(d_db(6,:,:));

create_log_image(data_db,16,0.5);
print -deps out_log_fig_db_16.eps; figure;

create_log_image(data_db,32,0.5);
print -deps out_log_fig_db_32.eps; figure;
```

D.13 Program e_field.m

This computes the electric field for a simple geometry (See section 7.5.3.)

```
function e_field()

% This performs the E_field computation

% this value is irrelevant (since it scales out)
d=1;

% set the ranges
w=-20:0.005:20'; w=w(:);

% common mode
pm=1;
rel_diff_vec=0;
h_vec=0.5:0.03:1.3;
len_h=length(h_vec);
for i=1:len_h
        h=h_vec(i);
        [rel_diff,mag_e]=find_field(d,w,h,pm);
        rel_diff_vec(i)=rel_diff;
end
plot(h_vec,rel_diff_vec); xlabel('h');
print -deps figcomm.eps; figure;
display_h_vec_and_rel_diff_vec=[h_vec' rel_diff_vec']

% illustrate the result with some sample values
h=0.5;
[rel_diff,mag_e]=find_field(d,w,h,pm);
plot(w,mag_e); xlabel('w'); title('h=0.5');
print -deps fig_comm_h05.eps; figure;

h=35;
[rel_diff,mag_e]=find_field(d,w,h,pm);
plot(w,mag_e); xlabel('w'); title('h=35');
print -deps fig_comm_h35.eps; figure;

% differential mode
pm=-1;
rel_diff_vec=0;
h_vec=0.5:0.2:6.0;
len_h=length(h_vec);
for i=1:len_h
        h=h_vec(i);
        [rel_diff,mag_e]=find_field(d,w,h,pm);
        rel_diff_vec(i)=rel_diff;
end
plot(h_vec,rel_diff_vec); xlabel('h');
print -deps figdiff.eps; figure;
display_h_vec_and_rel_diff_vec=[h_vec' rel_diff_vec']

% illustrate the result with some sample values
h=0.5;
[rel_diff,mag_e]=find_field(d,w,h,pm);
plot(w,mag_e); xlabel('w'); title('h=0.5');
print -deps fig_diff_h05.eps; figure;

h=35;
[rel_diff,mag_e]=find_field(d,w,h,pm);
plot(w,mag_e); xlabel('w'); title('h=35');
print -deps fig_diff_h35.eps; figure;

% ********************************************
function [rel_diff,mag_e]=find_field(d,w,h,pm)

% This subroutine performs the actual computation of the field

x1= d*(w+1/2);
x2= d*(w-1/2);
y1=d*h;
y2=d*h;
r1=sqrt( x1.^2 + y1^2 );
r2=sqrt( x2.^2 + y2^2 );

i_comp= x1./(r1.^3) + pm*x2./(x2.^3);
j_comp= y1./(r1.^3) + pm*y2./(x2.^3);

mag_e=sqrt(i_comp.^2+j_comp.^2);

min_loc=floor(length(mag_e)/2)+1;
min_val=mag_e(min_loc);
max_val=max(mag_e);
rel_diff=(max_val-min_val)/max_val;

return;
```

D.14 Program do_test_normality.m

This uses the Kolmogorov–Smirnoff test to determine if the data is normally distributed. (See section 7.2.1.)

```
% load in the 3D data
%--------------------
for i=1:6,
    c=loadfiles(97102902+i);
    data_db(i,:,:)=      c(2:127,2:127);
    data_lin(i,:,:)=10.^(c(2:127,2:127)/20);
end

% compute some baseline statistics
%---------------------------------
mean_std_db= [mean(data_db(:)), std(data_db(:)) ]
mean_std_lin=[mean(data_lin(:)),std(data_lin(:))]

% show some of the data
%----------------------

imagesc(squeeze(data_lin(3,:,:)));
print -deps out_ks_fig_lin.eps; figure;
imagesc(squeeze(data_db(3,:,:)));
print -deps out_ks_fig_db.eps; figure;

% find the mean for each slice in db space
%-----------------------------------------
mean_array_db=squeeze(mean(data_db));
imagesc(mean_array_db);
print -deps out_ks_fig_db_mean.eps; figure;
% subtract off this mean from each slice
%---------------------------------------
for i=1:6,
    data_db_mod(i,:,:)=squeeze(data_db(i,:,:))-mean_array_db;,
end
```

```
% make a vector out of the data
%------------------------------
vec_db_mod=data_db_mod(:);
plot(vec_db_mod);
print -deps out_ks_fig_vector_db.eps; figure;
% find the mean and variance
%---------------------------
mean_db_mod=mean(vec_db_mod)
std_db_mod=  std(vec_db_mod)
% find the mean for each slice in linear space
%---------------------------------------------
mean_array_lin=squeeze(mean(data_lin));
imagesc(mean_array_lin);
print -deps out_ks_fig_lin_mean.eps; figure;
% subtract off this mean from each slice
%---------------------------------------
for i=1:6,
    data_lin_mod(i,:,:)=squeeze(data_lin(i,:,:))-mean_array_lin;,
end
% make a vector out of the data
%------------------------------
vec_lin_mod=data_lin_mod(:);
plot(vec_lin_mod);
print -deps out_ks_fig_vector_lin.eps; figure;
% find the mean and variance
%---------------------------
mean_lin_mod=mean(vec_lin_mod)
std_lin_mod=  std(vec_lin_mod)

% use the Kolmogorov-Smirnoff test to test for normalacy
%-------------------------------------------------------

% actual data: lin
%-----------------
[h,sig,xvec,predict]=kshyp(vec_lin_mod,0.10,2,'normal',mean_lin_mod,std_lin_mod);
h_is=h
sig_is=sig
[xb1,yb1] = stairs(predict,xvec);
[xb2,yb2] = stairs(xvec,   xvec);
plot(xb1,yb1,xb2,yb2)
print -deps out_ks_fig_lin_test_test.eps; figure;

% run a median filter over the data
%----------------------------------
vec_lin_mod_median=medfilt1(vec_lin_mod,3);
plot(vec_lin_mod_median);
print -deps out_ks_fig_vector_lin_median.eps; figure;
mean_lin_mod_median=mean(vec_lin_mod_median);
std_lin_mod_median=  std(vec_lin_mod_median);
% re-run the KS test
%-------------------
[h,sig,xvec,predict]=kshyp(vec_lin_mod_median,0.10,2,'normal',mean_lin_mod_median,std_lin_mod_median);
h_is=h
sig_is=sig
[xb1,yb1] = stairs(predict,xvec);
[xb2,yb2] = stairs(xvec,   xvec);
plot(xb1,yb1,xb2,yb2)
print -deps out_ks_fig_lin_test_median.eps; figure;

% remove the extreme (noisy) values
%----------------------------------
plot(vec_lin_mod);
mean_vec=mean(vec_lin_mod)
std_vec= std(vec_lin_mod)
low_vec= mean_vec-3*std_vec
high_vec=mean_vec+3*std_vec
indicator_vec_01= (vec_lin_mod<low_vec) | (vec_lin_mod>high_vec);
sum_indicator_vec_01=sum(indicator_vec_01)
indicator_vec=indicator_vec_01.*vec_lin_mod-mean_vec;
mod_vec=vec_lin_mod-indicator_vec;
% show the new data
plot(mod_vec);
print -deps out_ks_fig_vector_lin_extreme3.eps; figure;
% find the mean and variance of the modified data
%------------------------------------------------
mean_mod_vec=mean(mod_vec)
std_mod_vec= std(mod_vec)
% re-run the KS test
%-------------------
[h,sig,xvec,predict]=kshyp(mod_vec,0.10,2,'normal',mean_mod_vec,std_mod_vec);
h_is=h
sig_is=sig
[xb1,yb1] = stairs(predict,xvec);
[xb2,yb2] = stairs(xvec,   xvec);
plot(xb1,yb1,xb2,yb2)
print -deps out_ks_fig_lin_test_extreme3.eps; figure;

% actual data: db
%----------------
[h,sig,xvec,predict] = kshyp(vec_db_mod,0.10,2,'normal',mean_db_mod,std_db_mod);
h_is=h
sig_is=sig
[xb1,yb1] = stairs(predict,xvec);
[xb2,yb2] = stairs(xvec,   xvec);
plot(xb1,yb1,xb2,yb2)
print -deps out_ks_fig_db_test.eps; figure;

% run a median filter over the data
%----------------------------------
vec_db_mod_median=medfilt1(vec_db_mod,3);
plot(vec_db_mod_median);
print -deps out_ks_fig_vector_db_median.eps; figure;
mean_db_mod_median=mean(vec_db_mod_median);
std_db_mod_median=  std(vec_db_mod_median);
% re-run the KS test
%-------------------
[h,sig,xvec,predict]=kshyp(vec_db_mod_median,0.10,2,'normal',mean_db_mod_median,std_db_mod_median);
h_is=h
sig_is=sig
[xb1,yb1] = stairs(predict,xvec);
[xb2,yb2] = stairs(xvec,   xvec);
plot(xb1,yb1,xb2,yb2)
print -deps out_ks_fig_db_test_median.eps; figure;

% remove the extreme (noisy) values
%----------------------------------
plot(vec_db_mod);
mean_vec=mean(vec_db_mod)
std_vec= std(vec_db_mod)
low_vec= mean_vec-3*std_vec
high_vec=mean_vec+3*std_vec
indicator_vec_01= (vec_db_mod<low_vec) | (vec_db_mod>high_vec);
sum_indicator_vec_01=sum(indicator_vec_01)
indicator_vec=indicator_vec_01.*vec_db_mod-mean_vec;
mod_vec=vec_db_mod-indicator_vec;
% show the new data
plot(mod_vec);
print -deps out_ks_fig_vector_db_extreme3.eps; figure;
% find the mean and variance of the modified data
%------------------------------------------------
mean_mod_vec=mean(mod_vec)
std_mod_vec= std(mod_vec)
% re-run the KS test
%-------------------
[h,sig,xvec,predict]=kshyp(mod_vec,0.10,2,'normal',mean_mod_vec,std_mod_vec);
h_is=h
sig_is=sig
[xb1,yb1] = stairs(predict,xvec);
[xb2,yb2] = stairs(xvec,   xvec);
plot(xb1,yb1,xb2,yb2)
print -deps out_ks_fig_db_test_extreme3.eps; figure;
```

D.15 Program principal_components.m

This Matlab subroutine performs a principal components analysis of the input array. (See section 7.3.)

```
function result=process(array)

% This function takes a collection of vectors
% and performs a principle components analysis

[n_row,n_col]=size(array)

cov_matrix=zeros(n_col,n_col);

for i=1:n_col,
    for j=1:n_col,
        cov_matrix(i,j)= array(:,i)'*array(:,j);
    end
end

% find the eigenvalues/vectors
[d,e]=eig(cov_matrix);

d
ev=sum(e)
```

D.16 Program do_principal_components.m

This is the driver for all principal component analysis computations. (See section 7.3.)

```
% set the seed for the random number generator
%-----------------------------------------------
rand('state',0);

% size of data
%-------------
n=3;

% create the random data sets
%----------------------------
I1= rand(n,n)
I2= rand(n,n)
I3= rand(n,n)

% we demonstrate principle components using each of these
%---------------------------------------------------------
I4= rand(n,n)
I5= I1 + 2*I2
I6= I1 + 2*I2 + 0.1*I4

% set up the array, and compute the quantities of interest
%---------------------------------------------------------
mat(:,1)=I1(:);
mat(:,2)=I2(:);
mat(:,3)=I3(:);

mat(:,4)=I4(:);
principal_components(mat);

mat(:,4)=I5(:);
principal_components(mat);

mat(:,4)=I6(:);
principal_components(mat);

% now do the real 3D data
%------------------------
for i=1:6,
    c=loadfiles(97102902+i);
    data_db(i,:,:)=      c(2:127,2:127);
    data_lin(i,:,:)=10.^(c(2:127,2:127)/20);
end

% set up the array, and compute the quantities of interest
%---------------------------------------------------------
clear mat;
for i=1:6,
    temp=squeeze(data_db(1,:,:));
    mat(:,i)=temp(:);
end
principal_components(mat);

% set up the array, and compute the quantities of interest
%---------------------------------------------------------
clear mat;
for i=1:6,
    temp=squeeze(data_lin(1,:,:));
    mat(:,i)=temp(:);
end
principal_components(mat);
```

D.17 Program do_image_board.m

This is the driver for the log filter applied to the data acquired from the PCB shown in Figure 37. (See section 7.5.2.)

```
% load in the data (circuit I)
%-----------------------------
c=loadfiles(98013002);
data_db=      c(2:127,2:127);
data_lin=10.^(c(2:127,2:127)/20);

% show input data
%----------------
imagesc(d_lin),colorbar;
print -deps out_image_board_98013002_lin.eps; figure;

imagesc(d_db),colorbar;
print -deps out_image_board_98013002_db.eps; figure;

% do the Laplacian of Gaussian filter
%------------------------------------
create_log_image(data_lin,16,0.5);
print -deps out_image_board_98013002_lin_16.eps; figure;

create_log_image(data_db,16,0.5);
print -deps out_image_board_98013002_db_16.eps; figure;


% circuit II
%-----------
c=loadfiles(98013102);
data_db=      c(2:127,2:127);
data_lin=10.^(c(2:127,2:127)/20);

imagesc(d_lin),colorbar;
print -deps out_image_board_98013102_lin.eps; figure;
imagesc(d_db),colorbar;
print -deps out_image_board_98013102_db.eps; figure;
create_log_image(data_lin,16,0.5);
print -deps out_image_board_98013102_lin_16.eps; figure;
create_log_image(data_db,16,0.5);
print -deps out_image_board_98013102_db_16.eps; figure;


% circuit III
%------------
c=loadfiles(98020103);
data_db=      c(2:127,2:127);
data_lin=10.^(c(2:127,2:127)/20);
imagesc(d_lin),colorbar;
print -deps out_image_board_98020103_lin.eps; figure;
imagesc(d_db),colorbar;
print -deps out_image_board_98020103_db.eps; figure;
create_log_image(data_lin,16,0.5);
print -deps out_image_board_98020103_lin_16.eps; figure;
create_log_image(data_db,16,0.5);
print -deps out_image_board_98020103_db_16.eps; figure;
```

Index

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X

Z