CHARACTERIZING THE INITIAL STATE OF CANTILEVER SENSORS

MENG XU

Characterizing the Initial State of Cantilever Sensors

by

©Meng Xu

A thesis submitted to the
Department of Physics and Physical Oceanography
in partial fulfillment of the
requirements for the degree of
Master of Science

Department of Physics and Physical Oceanography
Memorial University of Newfoundland

October 2008

ST JOHN'S NEWFOUNDLAND

Abstract

Cantilevers are ultra-sensitive sensors capable of detecting a variety of physical and chemical phenomena. Due to the construction of the sensor, the cantilevers are often pre-bent prior to using them as actual sensors. In order to properly interpret further cantilever deflections due to sensing events, it is important to understand the initial states of the cantilever. Also it is imperative to establish the initial orientation of the cantilever chip with respect to the horizontal. In this work, a new model to measure the initial orientation of the chip has been developed using the standard optical beam deflection system. Using reference chips inclined at 2°, 3.5°, and 5°, the proposed method was shown to be successful. A new method was also developed to measure the initial curvature based entirely on the vertical motion of the incident laser. Results compared to optical images showed our method to be successful. Lastly, based on our ability to measure the angle of inclination of the chip, we have successfully modified the "Rotating Method" developed previously in our group.

Acknowledgement

I would like to thank the following people who support me and my project during the two years in MUN:

- My supervisor, Prof. Luc Beaulieu for his endless time, support and patience.
- My colleague, Mike Coates for sharing his experience working in our lab.
- Our mechanic Gordon Whelan for his excellent works.
- Prof. Mike Morrow for keeping my project focused throughout the entirety study.

Table of Contents

List of Symbols

(By order of Appearance)

- AFM: Atomic Force Microscopy.
- μm: 1 Micron; 10^{-6} meters.
- SOI: Silicon-on-insulator.
- BOX: buried oxide.
- BHF: Buffered hydrofluoric.
- nm: 1 Nano meter; 10^{-9} meters.
- PSD: Position Sensitive Detector.
- SCCM: Standard Cubic Centimeters per Minute.
- OBDS: Optical Beam Deflection System.
- DAQ: Data Acquisition Board.
- θ: Angle of incidence of the incoming laser beam with respect to the XY plane.
- φ: Angle of the inclination of the PSD surface with respect to the XY plane.
- D: Distance from the laser spot on the chip/cantilever to the conjunction point of the chip and the cantilever.
- L_o: Distance from the laser spot on the chip/cantilever to the PSD surface.
- L_o': Distance from the laser spot on the chip/cantilever to the PSD surface of different position of PSD surface.
- Δh: Voltage signal change of PSD caused by different cantilever curvatures.

• $\Delta h'$: Voltage signal change of PSD caused by different cantilever curvatures with a different L_o'.

• Y_1: Output current of a photo-electronic current caused by a .

• Y_2: The other output current of a photo-electronic current.

• L: Length of the effective PSD surface.

• Y: Position of the incident light on the PSD surface.

• RPM: Rotation per Minute.

• CCD: Computer Controlled Display camera.

• VB: Visual Basic Program.

• θ_1: Initial incident laser angle of the rotation method.

• θ_2: Rotated incident laser angle of the rotation method.

• h_1: Initial PSD position of the rotation method.

• h_2: Rotated PSD position of the rotation method.

• β: Angle of the inclination of the chip with respect to the horizontal.

• P_1: Initial position of the laser spot on the chip/cantilever.

• P_2: Moved position of the laser spot on the chip/cantilever.

• ΔP_x: Horizontal displacement of the laser spot on the chip/cantilever.

• ΔP_y: Vertical displacement of the laser spot on the chip/cantilever.

• P_1': Initial position of the laser spot on the PSD surface.

• P_2': Moved position of the laser spot on the PSD surface.

• $\vec{n}$: Direction vector of the reflected laser beam.

• t_1: Scalar of the initial reflected laser beam.

• t_2: Scalar of the moved reflected laser beam.

- $\Delta\vec{P}$: Position change of the laser spot on the chip/cantilever in XY plane.
- $\Delta\vec{P}'$: Position change of the laser spot on the PSD surface in XY plane.
- Δt: t_2-t_1.
- n_x: X component of the direction vector of the reflected laser beam.
- n_y: Y component of the direction vector of the reflected laser beam.
- ΔP_x : X component of the position change of the laser spot on the chip/cantilever.
- ΔP_y : Y component of the position change of the laser spot on the chip/cantilever.
- $\Delta P'_x$: X component of the position change of the laser spot on the PSD surface.
- $\Delta P'_y$: y component of the position change of the laser spot on the PSD surface.
- N_x: X component of the surface normal vector.
- N_y: Y component of the surface normal vector.
- Δh_x: X component of the PSD voltage signal change.
- Δh_y: Y component of the PSD voltage signal change.
- A_x: Parameters of the cubic equation for $\tan\beta$.
- $P(t)$: Position of laser spot read by digital indicator.
- t: Time scale of the movement of the laser spot.
- ε: Angle of the surface normal vector with respect to horizontal.
- α: Slope of the cantilever curvature.

List of Figures

Chapter One

Chapter Two

Chapter Three

Chapter Four

Chapter One: Introduction

In section 1.1, we describe micro-cantilevers, how they are manufactured, and how they are used in our experiments. In section 1.2, we introduce cantilever sensors, how they work and the problems associated with the technology. The motivation and the scope of this study are presented in sections 1.3 and 1.4 respectively.

1.1 Micro-cantilevers

Micro-cantilevers were first introduced by researchers at IBM Research Laboratory and Stanford University in 1985 [1] as detection probes for the Atomic Force Microscope (AFM), for imaging the surface morphology of both conducting and non-conducting samples. There are two basic shapes of cantilevers as shown in figure 1.1: rectangular cantilevers, which are of the order of 200-400 μm long, 30-50 μm wide and 1 μm thick, and V-shaped cantilevers, which are of the order of 90-200 μm long, 40-60 μm wide, and 1 μm thick.

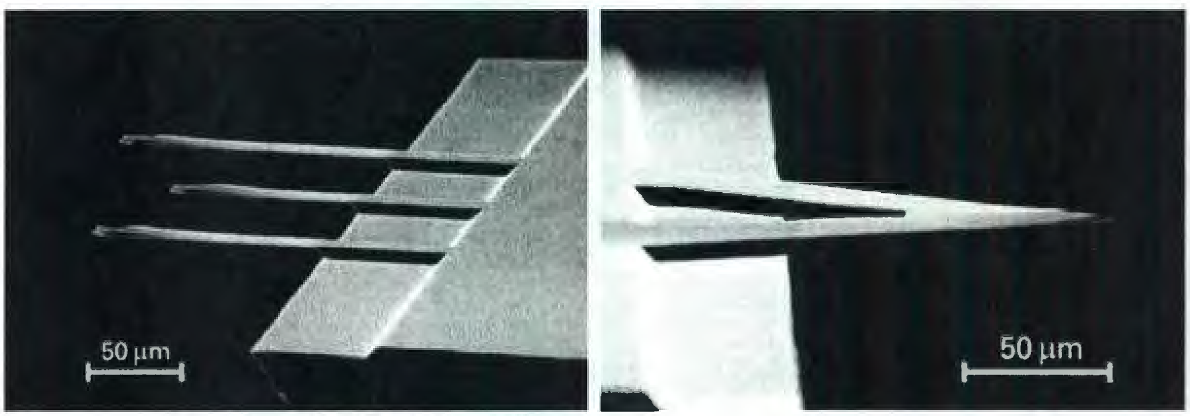

Fig. 1.1 Rectangular and V-shaped micro-cantilevers

Most micro-cantilevers are made of silicon or silicon nitride and are manufactured by chemical etching and conventional photolithographic techniques. The micromachining process starts with a silicon on insulator (SOI) wafer, which has a buried oxide (BOX) layer approximately 1 μm below the top surface of the wafer (figure 1.2a). One method of making cantilevers is to pattern the cantilever shape on the top surface of the SOI wafer using standard photolithography techniques (figure 1.2b). By etching, the exposed Si area is removed to the oxide layer which acts as a natural etch-stop. Following the etch, a new layer of silicon oxide is then deposited on the already formed cantilevers up to a point just beyond the cantilevers as shown in figure 1.2c. The area not covered by silicon oxide is used as an etch window which allows the silicon below the BOX to be removed by etching (figure 1.2d). Removing the silicon oxide followed by intensive rinsing releases the cantilevers as shown in figure 1.2c [2].

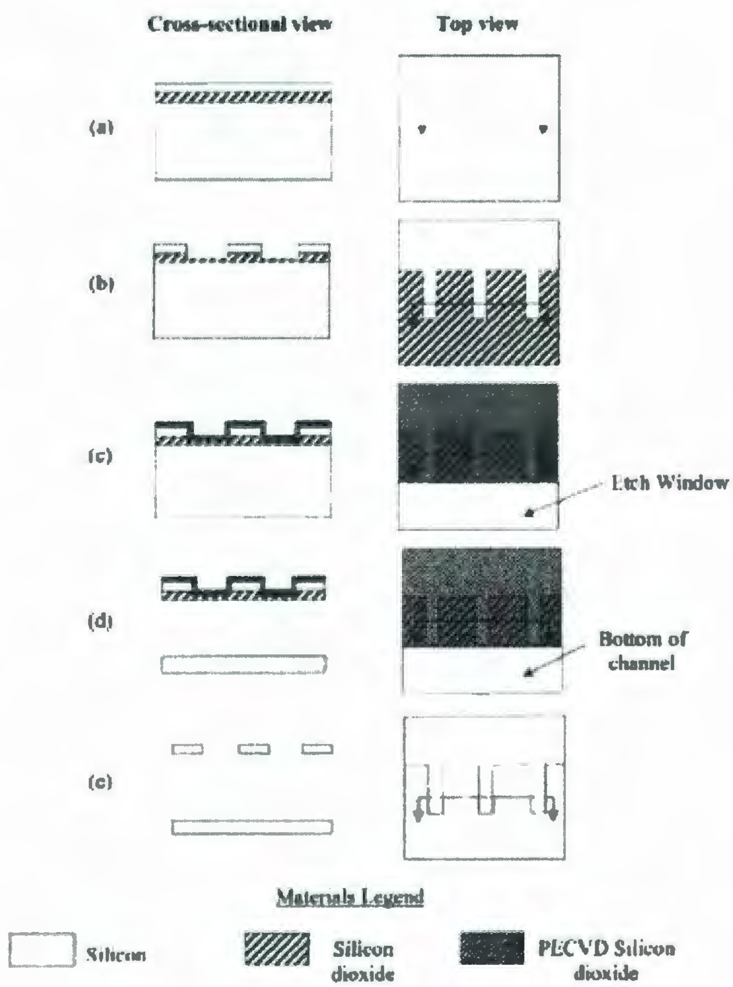

Fig. 1.2 Process of silicon cantilever fabrication

In the cantilever fabrication process, the surface roughness of the cantilever is highly dependent on the solution concentration. Thus, the etched surface becomes rough after etching with highly diluted formic acid [2]. As will become apparent in later chapters, the roughness of the cantilever surface is important because it affects the reflection of the laser spot from the cantilever/chip surface. All of the cantilevers used in this study were purchased from MikroMasch Company (Tallinn, Estonia). In figure 1.3, the top and side views of the cantilever are shown schematically. The shaded rectangular area (3.4mm×1.6mm) is called the chip of the micro-probe, which has six rectangular cantilevers suspended from it. All the cantilevers have the same widths but different lengths. In this work, only the longest cantilever E (width 35 μm, length 350 μm, and thickness 1μm) was used since this is the most sensitive cantilever.

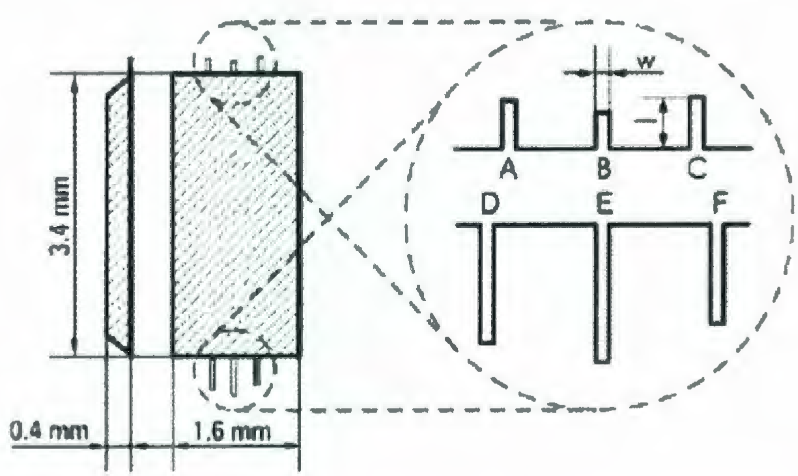

Fig. 1.3 MikroMasch CSC12/Tipless/Non-coated micro-probe

1.2 Cantilever Sensors

Besides being used as AFM imaging probes, micro-cantilevers have been used as ultra-sensitive sensors for a variety of physical and chemical phenomena [3]. Due to their small size, micro-cantilevers have a short response time and ultra-small detection range. In recent studies, these sensors have been used as chemical sensors [4-6], bio-sensors [7-12] and surface stress sensors [13-15]. These ultra-sensitive cantilever sensors can detect quantities in the nanogram (10^{-9}), pictolitre (10^{-12}), femtojoule (10^{-15}) and attomolar (10^{-18}) range, with a short response time on the order of milliseconds [3].

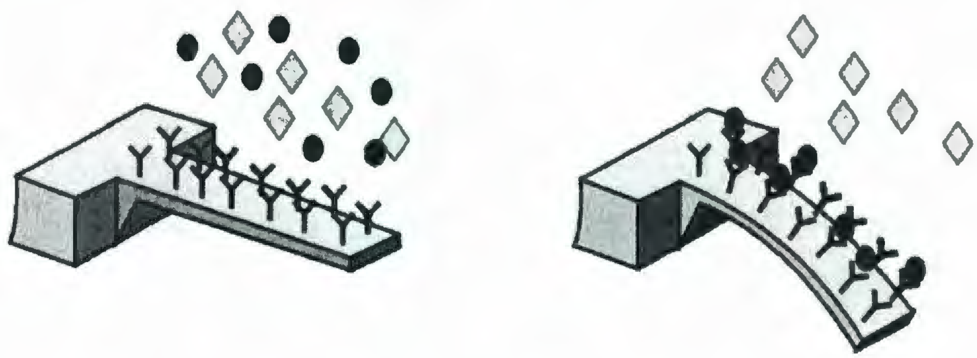

Fig. 1.4: Cantilever sensor detect only the target molecules (circular) though reactions with the functionalized layer.

Cantilever sensors are generally composed of a silicon cantilever on which has been deposited a thin gold film. On the Au film, receptor molecules are attached, which react specifically with the target molecules to be detected. The general process of how a cantilever sensor reacts to target molecules is shown in Fig. 1.4. The Au coated cantilever is functionalized with receptor molecules, which only react with specific molecules in the surrounding environment. Absorption of the target molecules creates a surface stress on the cantilever causing the latter to deflect. The concentration of target molecules in the surrounding media can be estimated by the amount of deflection.

Generally, gold is chosen as the connection layer for two reasons. First, it is usually possible to find receptor molecules that bond strongly to gold. Second, the stable character of gold prevents the functionalized layer from coming off the cantilever due to oxidation effects. Unfortunately, there are several problems with using Au. One problem is that the deposition process sometimes leaves the Au film in a stressed state. The amount and kind of stress (tensile or compressive) suffered by the cantilever is still not completely understood. For example, figure 1.5 below shows four optical pictures of similar cantilevers that were sputter coated with Au *all at the same time*. It is clear from figure 1.5 that two of the cantilevers experienced tensile stress while two of the cantilevers are in compression. Another problem with having Au on one side of the lever is that the lever becomes highly susceptible to changes in temperature due to the bimetallic effect.

Fig. 1.5 Side view of different cantilever curvatures

1.3 Motivation

In order to obtain high precision cantilever sensor measurements, it is important to understand the initial conditions of the cantilever. In a recent paper by Beaulieu *et al.* [16] it was shown how to quantify the cantilever deflection based on the signal measured by an optical beam deflection system. However, in their work the authors assumed that the initial state of the cantilever was un-deflected and perfectly horizontal. As shown in figure 1.5 the deposition of Au on micro-cantilevers can leave them in a highly stressed state. Moreover, attaching the receptor molecules to the Au coated cantilever can further induce a surface stress resulting in increased cantilever deflections. Once the cantilever is deflected it is not possible to infer further deflections of the cantilever from the position sensitive detector signal unless the initial curvature is first obtained. Also it is imperative to establish the initial orientation of the cantilever chip with respect to the horizontal in order to obtain the direction of the surface normal of the cantilever as the lever bends. The surface normal of the cantilever is critical since it dictates the direction of the reflected beam.

1.4 Scope of this Thesis

In this thesis we will derive a method for determining the initial orientation of the nano-probe with respect to the horizontal and develop a method for determining the initial curvature of the cantilever. Chapter Two of this thesis will discuss the sample preparation, the experimental setups, and the software used to process images and collect data. The experimental techniques developed in this study will be described in Chapter Three, combined with the data analysis and results. The conclusion and future work will be given in Chapter Four.

Chapter Two:

Sample Preparation and Experimental Setup

In this chapter we discuss the experimental setup used in this work. In section 2.1, the preparation of the micro-cantilevers is described in detail. In section 2.2, the setup components are presented. Finally in section 2.3, we discuss the method used to acquire optical images and the software written to control the hardware and analyze the data.

2.1 Cantilever Sample Preparation

When gold is deposited on micro-cantilevers, the resulting film often leaves the cantilevers in a state of tensile or compressive stress. For example, the cantilevers shown in Fig. 1.5 were all coated at the same time, yet they all show different degrees of stress. Therefore, there is a need to find the proper deposition parameters to control the stress in the gold film.

In our experiments, micro-cantilevers were prepared as follows. First, the cantilevers "D" and "F" (see Fig. 1.3) were removed to allow the central lever "E" to be viewed from the side. Then the cantilevers were immersed in a Piranha solution (H_2SO_4:H_2O_2=3:1) for 10 minutes to remove any residue on the surface. The levers were then washed twice with de-ionized water to completely remove the Piranha solution. The cleaning process was performed very gently and carefully so as to not break the cantilever. To minimize the chance of damaging the levers, the micro-probes were held by tweezers when immersed in and out of the solutions. After rinsing, the levers were dried with nitrogen gas in a direction along the length of the cantilever.

Thin gold films were deposited on the cleaned cantilever samples by sputtering deposition at 150 W, with a gas flow rate of 20 SCCM (Standard Cubic Centimeters per Minute) for 10 mins.

2.2 Experiment Setup

The schematic diagram in Fig. 2.1 shows an overview of the complete experimental setup used in this work. A laser focuser and a position sensitive detector (PSD), the most essential components of the optical beam deflection system (OBDS), were mounted in a straight line to analyze the change in cantilever curvature. With a precision current source (d) used to power the laser diode (e), the laser beam was excited and focused on the cantilever and then reflected onto the PSD. By rotating the laser incident angle with the laser holder arm, or moving the laser point position on the cantilever with a 12 V DC motor (g), the laser position on the PSD changed correspondingly. The movement of the laser beam, in other words, the movement of the laser focuser holder, was measured with a digital indicator (j). The indicator was connected to the lab computer with an input device (k), and controlled by a pulse generator (l) to read position information at regular intervals. The impinging beam on the PSD surface caused a current to develop in the PSD which was converted into a voltage signal by an amplifier board (c). The voltage data was collected by the Data Acquisition (DAQ) Board (b) and gathered by the lab computer (a).

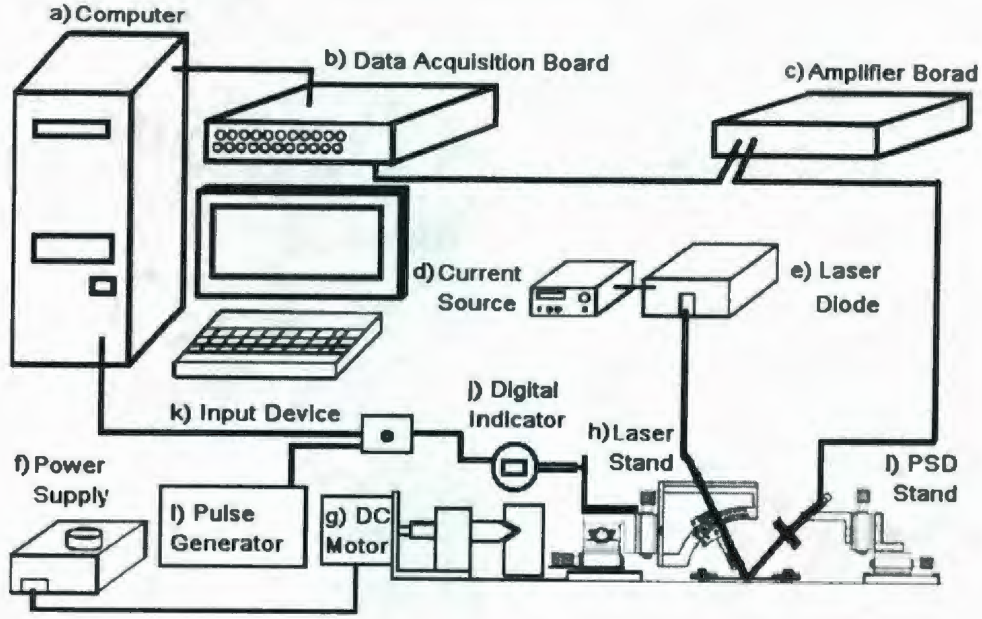

Fig. 2.1 Overall view of experiment setups

2.2.1 Optical Beam Deflection System (OBDS)

This section focuses on the essential parts of the OBDS: the laser focuser, the PSD, and the cantilever. As shown in figure 2.2, an optical beam is focused at an incident angle θ, which reflects into a PSD held at an angle φ. The distance from the laser point on the cantilever to the chip is D. The distance from the laser point on the cantilever to the laser point on the PSD is L_o.

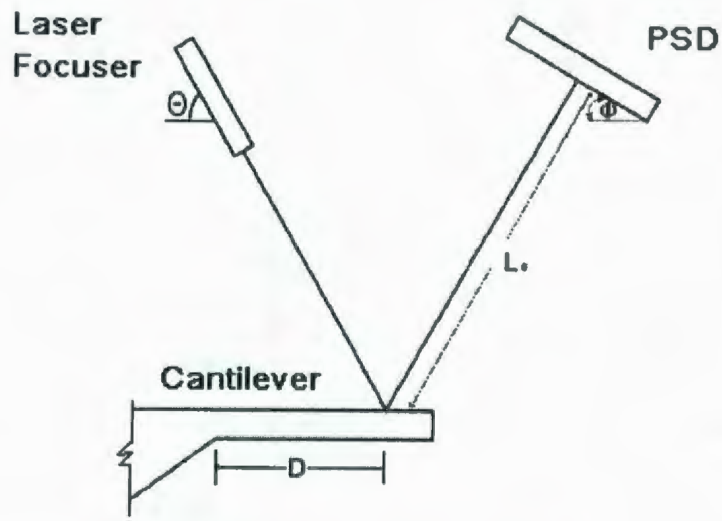

Fig. 2.2 Schematic graph of OBDS

While the laser spot moves along the length of the cantilever, the laser spot on the PSD moves correspondingly. In our system, it is crucial to measure the value of L_o accurately. Consider a deflected cantilever shown in figure 2.3 with different values of L_o. For a given deflection each PSD will give a different signal. It is clear that Δh is smaller than $\Delta h'$, hence causing L_o to act as an amplifier factor in the OBDS.

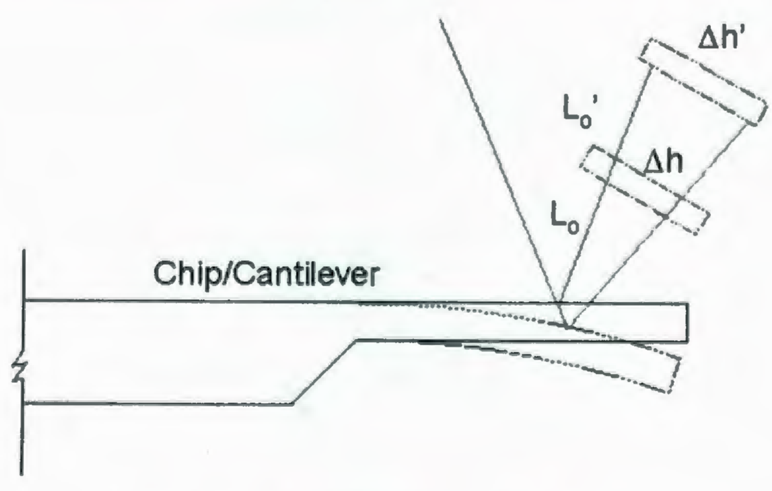

Fig. 2.3 Effect of the value of different L_o

2.2.2 The Laser and the Laser Arm

In this study, a precision current source (LDX-3412, ILX Lightwave Corp.) was used as the current source for the laser diode (FMXL112-00, Claire Lasers). The laser diode was mounted on a special temperature control holder and controlled by a temperature controller (LDT-5412, ILX Lightwave Corp.). For this system, a 10 KΩ setting on the temperature controller corresponds to a temperature of 25 °C, while the 40.6 mA on the precision current source corresponds to a laser power of 1 mW. The laser beam was focused on the cantilever using optical focusers (LPF-01-635-4/125-S-2.4-15-4.7GR-40-3S-1-2, OZ Optics).

To control the incident angle of the laser beam, a laser arm was designed by Ye Tian [17]. Figure 2.4 contains a photograph of the laser arm with the laser focuser fixed on the laser holder (a), the 3D view (b) and the side view (c) of the laser arm, and a 3D side view (d) and the side view (e) of the laser holder. The rotating arc of the holder has the inner and outer radius of 39.0 mm and 46.5 mm respectively. There are 26 positions on the arc that allow the laser focuser to be positioned from 40° to 90°. The laser focuser was secured to the holder with a small screw on the backside.

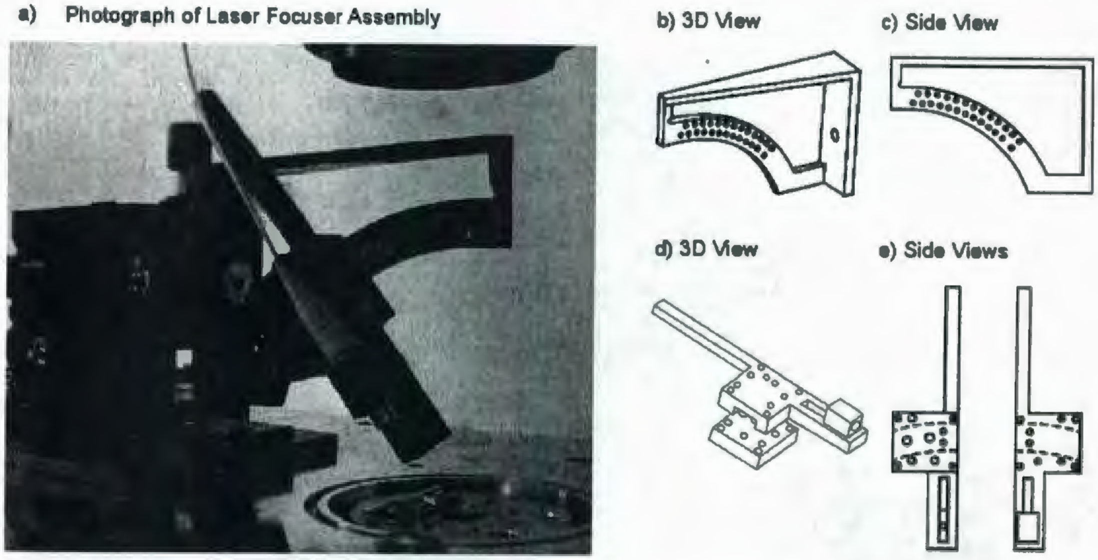

Fig. 2.4 Laser focuser holder assembly: a) photograph of the laser focuser assembly; b) 3D view of the laser arm; c) side view of the laser arm; d) 3D view of the laser holder; e) side views of the laser holder.

2.2.3 Position Sensitive Photo Detector

The PSD is made of a photo-sensitive semiconductor material. When light hits the highly sensitive laminar semiconductor, a photo-electronic current is generated which is divided into two output currents Y_1 and Y_2 (as shown in the figure 2.5).

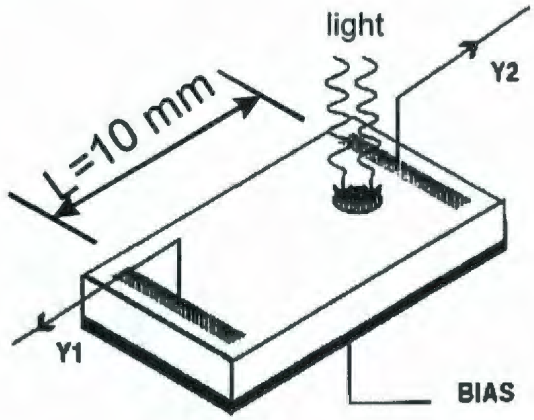

Fig. 2.5 Graph of PSD

Defining the length of the effective PSD surface as L, the relationship between the position Y of the incident light and the currents is given by:

$$Y = \frac{Y_1 - Y_2}{Y_1 + Y_2} \times \frac{L}{2} \tag{2.2}$$

The maximum power density of the PSD surface is 3 W/cm^2, with a maximum power of 1 mW. The laser source used in this work has a power of 1 mW. After reflecting from the cantilever, the diameter of the laser spot on the PSD surface is approximately 2 mm which gives a laser power density of approximately 0.03 W/cm^2 well below the maximum.

In our experiments, the PSD signal was sent to an amplifier board, transferred to a voltage signal and then read by a data acquisition board (PCI6036E, National Instruments). The transfer ratio between the PSD voltage signal and the position of the laser spot on the PSD surface was determined by the maximum and minimum values of PSD output voltage.

2.2.4 DC motor and Translation Stage

In our experiments, a 12V DC motor with a rotation speed of 1 RPM was used to control the movement of the laser beam. To eliminate mechanical vibrations, the DC motor was isolated from the laser mount. The design of the DC motor control mount was improved several times.

First, the DC motor was connected directly to the laser mount through the motor rotation axis. In this configuration, the laser point shifted from left to right continuously, causing the laser point to move in an elongated S-shape track. This was caused by the rotation of the DC motor axis which forced the laser mount to shift around its original position. To avoid this, the connection between the DC motor and the laser mount was redesigned. After several iterations we arrived at our current design.

In the final incarnation (shown in figure 2.6), the DC motor was mounted onto a U-shape aluminum platform. A new connecter was designed to link the DC motor and the laser mount. This connection mount transferred the rotating motion of the DC motor to the linear motion of the laser mount. When the motor rotates, it turns a brass screw and drives a brass dowel forward. A groove is machined in the brass dowel to prevent the latter from rotating. This design ensured a linear movement of the laser point along the cantilever.

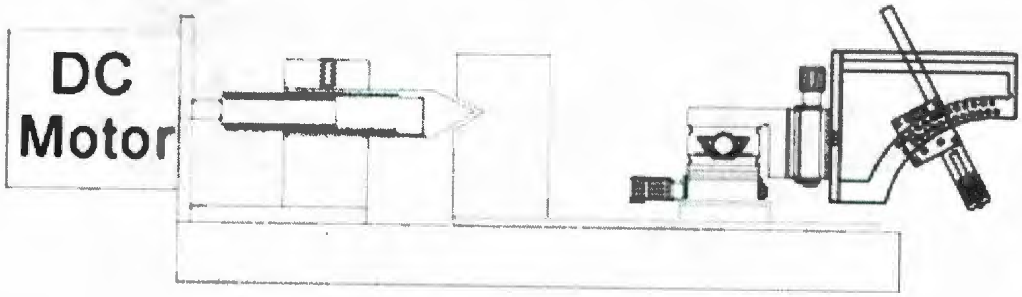

Fig. 2.6 Schematic graph of DC motor mount and laser mount

Lastly, three single axis translation stages were used to control the position of the laser focuser and the PSD.

2.3 Image Collecting System

A Computer Controlled Display (CCD) camera (CV-S3200N, JAI Company) connected to a telescope was mounted above the system for collecting images of the cantilever during the experimental process. Using the CCD camera, monochrome images of the cantilever (800×600 in pixels) were acquired in real time (Fig. 2.7). According to the different shade of each pixel on every image, a Visual Basic program written by us was used to recognize and distinguish the background, the chip, and the cantilever. Using a series of images, the change in position of the laser point on the cantilever was calculated and saved into a data file.

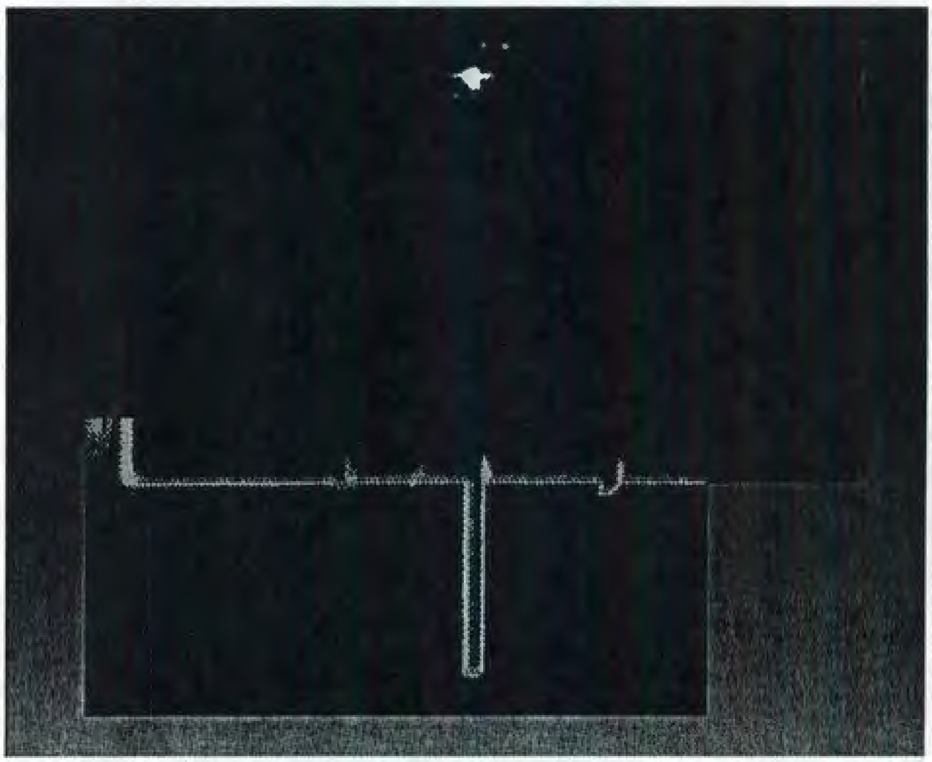

Fig. 2.7 Image analysis by the Visual Basic program

Initially, the VB program was modified to calculate and identify the changing position of the laser point as it moved along the cantilever. However, it was found that some of the positions were improperly identified. The reason for this was because the surface of the chip was not perfectly smooth after it had been coated with an Au film, which distorted the laser spot at some points. In these cases, the program could not recognize the position of the laser point. Also, when the program was set to analyze the scanning area, a few seconds was needed to finish the calculations. Even when the program was changed to analyze a smaller moving area (50×50 pixel^2 for example), it still took too much time compared to the motion of the laser point. Therefore, the program was modified to collect and save optical images while reading and saving the digital indicator text information. When the experiment was stopped, the program reloaded all the images and analyzed them one at that time. This method saved the time of analyzing images, guaranteed the accuracy of laser position movement, and was efficient enough to finish the calculation.

Chapter Three:

Model, Results, and Analysis

In this chapter we discuss the model used to calibrate the OBDS. In section 3.1, a correction to the Rotation Method to include the angle of inclination of the chip is presented. In section 3.2, we show how to measure the angle of inclination of the chip, and the method used to verify our model is discussed in section 3.3. In section 3.4, we discuss how to determine the initial curvature of a cantilever. The analysis of different cantilever curvatures is given in section 3.5.

3.1 Rotation Method

A method to determine L_0, the distance between the laser point on the cantilever and the PSD surface, was developed by Ye Tian [17]. By changing the angle of inclination θ of the incident laser beam with a fixed PSD surface angle φ, the reflected laser spot on the PSD surface moves causing a change in the PSD voltage signal (see figure 3.1). Using simple geometry, the value of L_0 can be related to the change of the incident laser angle and the change of the PSD voltage signal by the sine law.

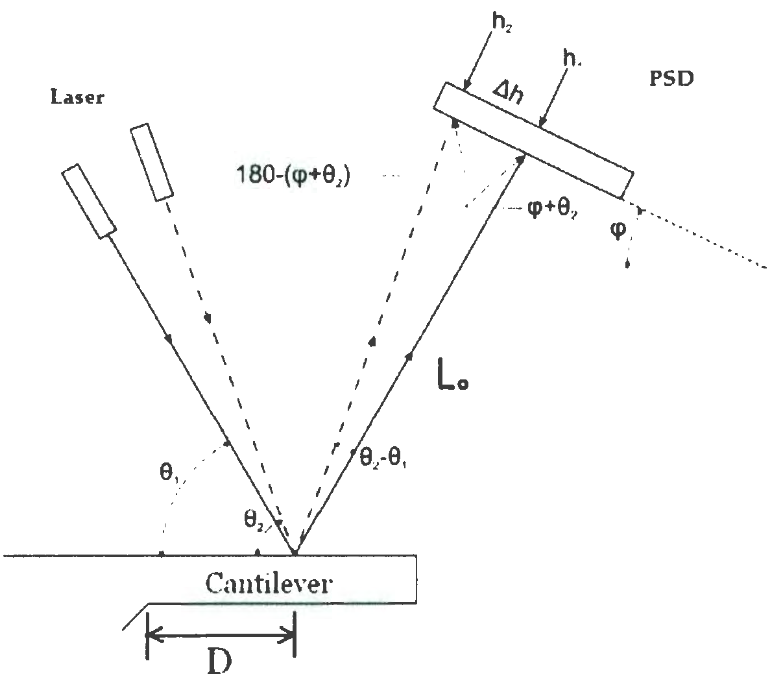

Fig. 3.1 Schematic diagram of Optical Beam Deflection System

For example, by changing the incident angle of the laser beam from θ_1 to θ_2 as shown in figure 3.1, the position of the laser spot on the PSD surface changes from h_1 to h_2. The value of L_0 is related to $\frac{\sin(\theta_2-\theta_1)}{\sin(\theta_2+\phi)}$ and Δh. To measure the average value of L_0, the incident angle θ is changed several times. Plotting the values of Δh vs $\frac{\sin(\theta_2-\theta_1)}{\sin(\theta_2+\phi)}$ gives a straight line with a slope of L_0 as shown by

$$\Delta h = L_o \frac{\sin(\theta_2-\theta_1)}{\sin(\theta_2+\phi)}. \tag{3.1}$$

When using this rotating method, it is very important for the laser spot on the cantilever to be at the same position when varying the angle θ. Before measuring L_0, the position of the laser spot needs to be adjusted to ensure that the center of the rotating laser focuser is on the cantilever surface. Otherwise the laser spot on the cantilever shifts as the laser focuser rotates. Figure 3.2 suggests a relationship between the center of the rotating laser focuser and the cantilever surface. If the center of the rotating laser focuser is higher than the cantilever surface, the laser spot will move backwards when the incident angle increases. In contrast, the laser spot will move forward with increasing incident angle if the center of the rotating laser focuser is lower than the cantilever surface. The height of the laser focuser is adjusted with a transition stage.

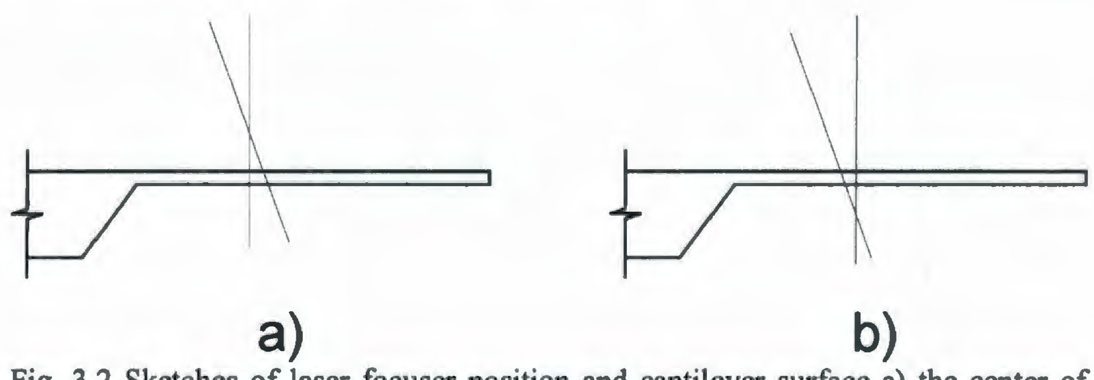

Fig. 3.2 Sketches of laser focuser position and cantilever surface a) the center of rotating laser focuser is higher than the cantilever surface; b) the center of rotating laser focuser is lower than the cantilever surface.

In this rotating method, it was assumed that the chip of the probe was perfectly leveled. Because of the small size of the probe, it was difficult to exam if the chip was leveled. Assuming that the chip is inclined at an angle β, as shown in figure 3.3, the sine law ratio changes from $\frac{\sin(\theta_2 - \theta_1)}{\sin(\theta_2 + \phi)}$ to $\frac{\sin(\theta_2 - \theta_1)}{\sin(\theta_2 - \beta + \phi)}$.

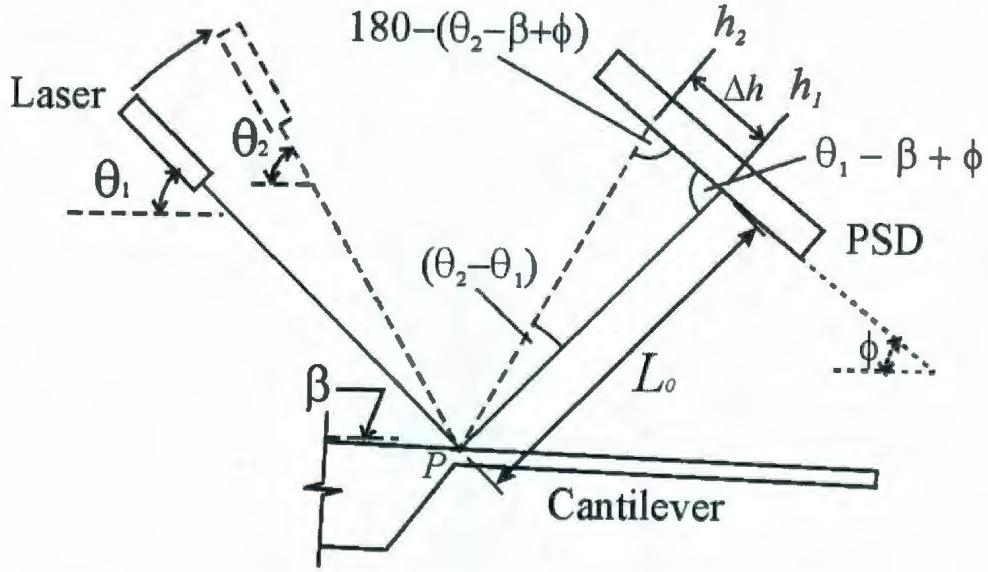

Fig. 3.3 Schematic graph of the β correction

3.2 Angle of Inclination of the Chip

Consider a probe inclined at an angle β with respect to the horizontal as shown in figure 3.4. This section will discuss a method to obtain β.

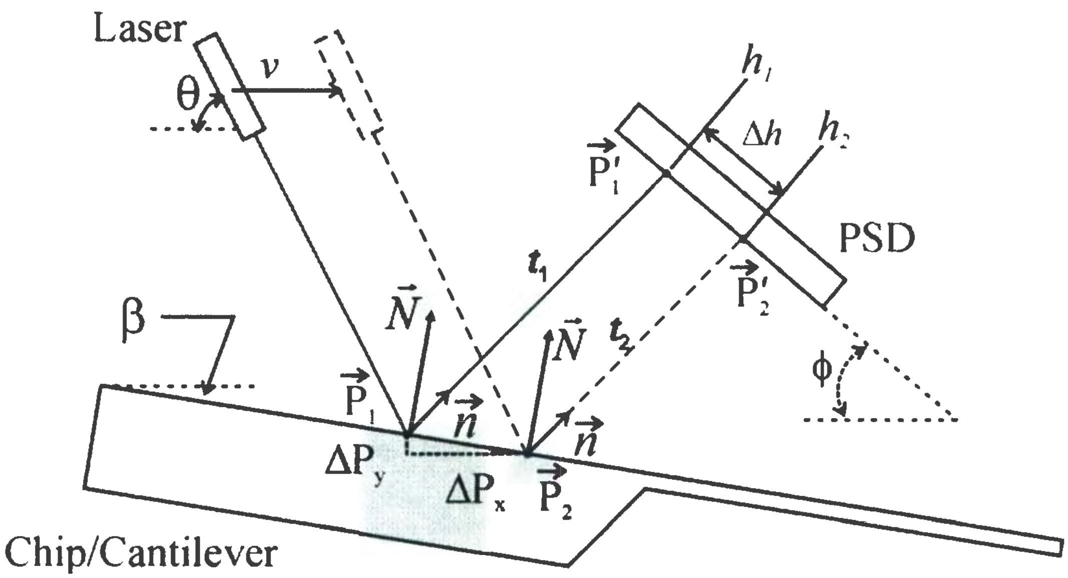

Fig 3.4 Schematic graph of the inclined chip and cantilever

In this setup the incident angle of the laser beam and the angle of inclination of the PSD are fixed and are given by θ and φ respectively. Moving the laser focuser from $\vec{P_1}$ to $\vec{P_2}$ causes a horizontal position change of ΔP_x. The reflected laser point on the PSD surface also moves from $\vec{P_1}'$ to $\vec{P_2}'$, contributing to a change in the PSD signal proportional to Δh. We can represent the equations of the reflected laser beam by the vector lines (1) and (2) as follows:

$$\vec{P_1} + \vec{n} \cdot t_1 = \vec{P_1}', \tag{1}$$

$$\vec{P_2} + \vec{n} \cdot t_2 = \vec{P_2}'. \tag{2}$$

where $\vec{n}$ is the direction vector of the reflected laser beam and t_1 and t_2 are scalars.

Subtracting equation (1) from (2) gives

$$\Delta\vec{P} + \Delta t\vec{n} = \Delta\vec{P'}. \tag{3}$$

Writing this equation in terms of x and y components gives

$$\Delta P_x + \Delta t n_x = \Delta P_x', \tag{4}$$

$$\Delta P_y + \Delta t n_y = \Delta P_y'. \tag{5}$$

Rearranging equations (4) and (5) and dividing gives

$$\frac{n_x}{n_y} = \frac{\Delta P_x' - \Delta P_x}{\Delta P_y' - \Delta P_y}. \tag{6}$$

From figure 3.4 it can be seen that $\tan(\theta - 2\beta) = \frac{n_y}{n_x}$ and $\frac{N_x}{N_y} = -\frac{\Delta P_x}{\Delta P_y} = \tan\beta$.

Using the relationships

$$\tan(\theta - 2\beta) = \frac{\tan\theta - \tan(2\beta)}{1 - \tan\theta\tan(2\beta)} \tag{7}$$

and

$$\tan(2\beta) = \frac{2\tan\beta}{1 - \tan^2\beta} \tag{8}$$

We can obtain a cubic equation for $\tan\beta$:

$$A_3\tan^3\beta + A_2\tan^2\beta + A_1\tan\beta + A_0 = 0 \tag{9}$$

where the four parameters have following forms:

$$\begin{aligned}
A_3 &= \Delta P_x \\
A_2 &= (\Delta h_x - \Delta P_x)\tan\theta - \Delta h_y - 2\Delta P_x\tan\theta \\
A_1 &= 2\Delta h_y\tan\theta - \Delta P_x + (\Delta h_x - \Delta P_x) \\
A_0 &= \Delta h_y - (\Delta h_x - \Delta P_x)\tan\theta.
\end{aligned}$$

These parameters are a combination of the following four variables: ΔP_x, Δh, φ, and θ. ΔP_x is the horizontal displacement of the laser spot which is measured with a CCD camera/VB program (Section 2.3). Δh is the change in laser position on the PSD surface (which is obtained from the PSD voltage change). φ is the PSD surface angle, and θ is the incident laser angle. Both of these two angles are fixed throughout all experiments. The only unknown quantity in equation (9) is the angle of the chip β. Using a VB program written in-house presented in Appendix B, this cubic equation can be solved for $\tan\beta$ using Vieta's Theorem. This method gives three roots however choosing the right root is always clear.

3.3 β Calibration

In order to validate our model for measuring β, three aluminum blocks, as shown in figure 3.5, were constructed with inclined planes of 5°, 3.5°, and 2°. A thin mirror was mounted on the inclined surface to enhance laser reflection. The blocks were mounted in the same position as the cantilevers in our system in a similar manner to the setup shown in figure 3.4. Figure 3.6 is a plot of Δh versus ΔP_x showing the experimental data (points) and the expected value (straight line) based on the value of β. From this plot it appears that the experimental data is very close to the expected value however fitting the data gives a value of β = 4.7° compared to the expected value of 5.0°.

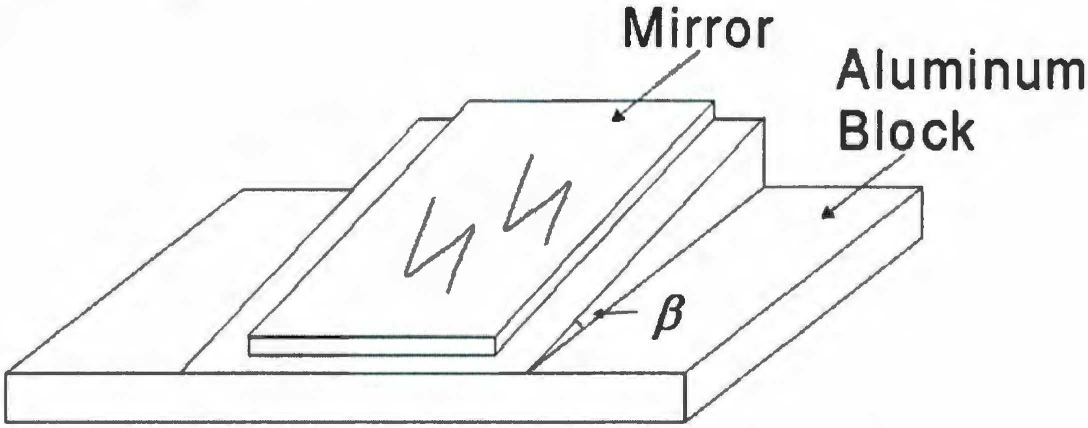

Fig. 3.5 Aluminum block with fixed angle of inclination β

In these experiments the value of β is very sensitive to the slope of the experimental data. For example, the slope of the expected data shown in figure 3.6 is 0.828. However, the slope of the experimental data is 0.8323 which is only slightly different. Other experiments using the 2° and 3.5° blocks have given similar results.

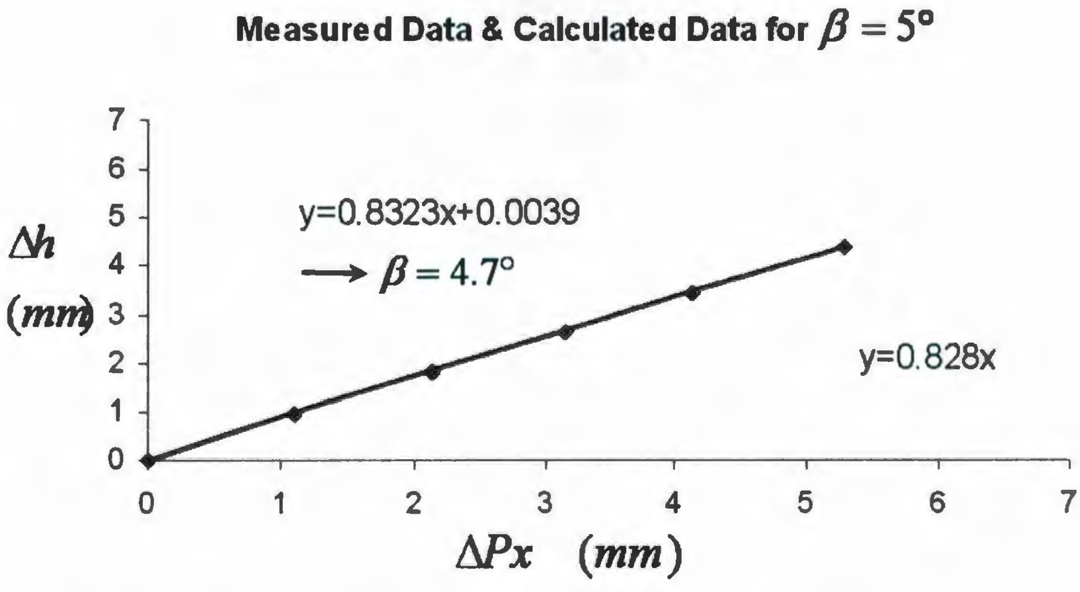

Fig 3.6: Plot of Δh versus ΔP_x showing experimental data (points) and expected value (solid line) for determining the angle of inclination of the chip β.

In order to better understand these results attempts were made to find the sources of possible errors in the system that could influence the value of β. The first thoughts were that the value of β could be affected by either the angle of incidence of the laser θ or the angle of inclination of the PSD ϕ. To bring $\beta = 4.7°$ close to $\beta = 5°$ requires θ to be changed from 60° to 60.35° which is definitely possible. Another possible source of error in obtaining β is the conversion factor relating the number of pixels to length used to obtain the position of the incident laser spot on the cantilever from the collected optical images. The difficulty here lies in obtaining an object of a known an appropriate length in the collected optical images. Also in an optical image as shown in figure 2.7 there are only 648 by 480 pixels which also limits the ability to accurately obtain a conversion factor from pixels to length.

3.4 Cantilever Curvature

Consider a cantilever attached to a chip which is inclined at an angle β. Because of the thin metal film deposited on the probe the cantilever is often initially bent as shown in figure 3.7. At $t = 0$ an optical beam inclined at an angle θ with respect to the horizontal is focused on the intersection point of the cantilever and the chip defined as the origin of the system. At this point the equation of the optical beam is given by $y_L = \tan(\theta)x$. If the optical beam is moved horizontally at a constant velocity v then the beam will intersect the abscissa at the point $x' = P(t) = vt$ which allows the intersection point between the optical beam and the ordinate to be obtained and the line describing the optical beam to be defined as: $y_L = \tan(\theta)(x + P(t))$. Using a telescope positioned above, it is possible to measure the intersection point between the optical beam and the cantilever ΔP_x which can be used to find the vertical displacement of the cantilever $\Delta P_y = \tan(\theta)(\Delta P_x + P(t))$. Therefore the vector normal $\vec{N}$ and the slope (α) of the cantilever at the point (ΔP_x, ΔP_y) can be defined as:

$$\varepsilon = \tan^{-1}\left[\frac{\Delta P_x}{\tan(\theta+\beta)(\Delta P_x + vt)}\right] \qquad (10)$$

$$\vec{N} = (\cos(\varepsilon), \sin(\varepsilon))$$

$$\alpha = \tan^{-1}\left[\frac{\tan(\theta+\beta)(\Delta P_x + vt)}{\Delta P_x}\right] \qquad (11)$$

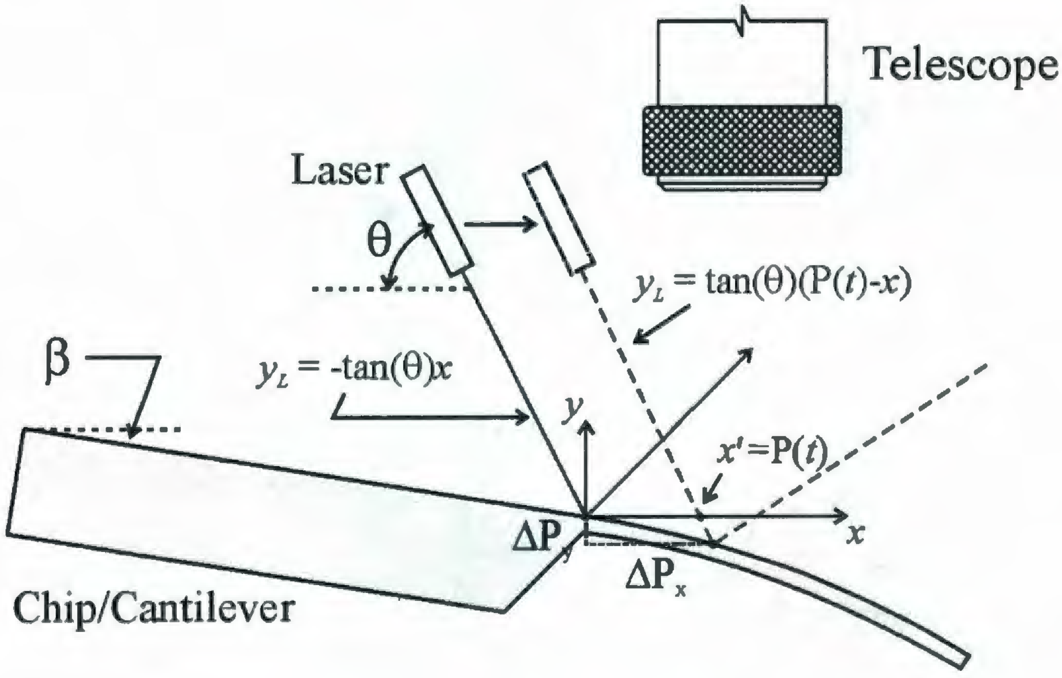

Fig. 3.7 Cantilever curvature with the angle of inclination β

In these experiments the laser beam was moved horizontally by the DC motor (see section 2.2.4), while its position $P(t)$ was measured by a digital indicator (ID-C112E, Mitutoyo). The data points were collected by the digital indicator at discrete intervals (one data point per 0.2 second). Since the data obtained from the digital indicator were not necessarily collected at the same time as the optical images used to measure ΔP_x, it was necessary to fit the data from the digital indicator to get an equation as a function of time $P(t)$. A VB program (given in Appendix C) as shown in figure 3.8 was used to fit the scattered data points to a fourth order polynomial $P(t)$ in a similar way as a Savitzky-Golay filter is used to smooth data. More precisely, a continuous function of time was created by fitting a fourth order polynomial at every point P using 10 data points on both sides of P. Every group of polynomial parameters were then saved and used to represent the real position of the laser beam movement along the x direction.

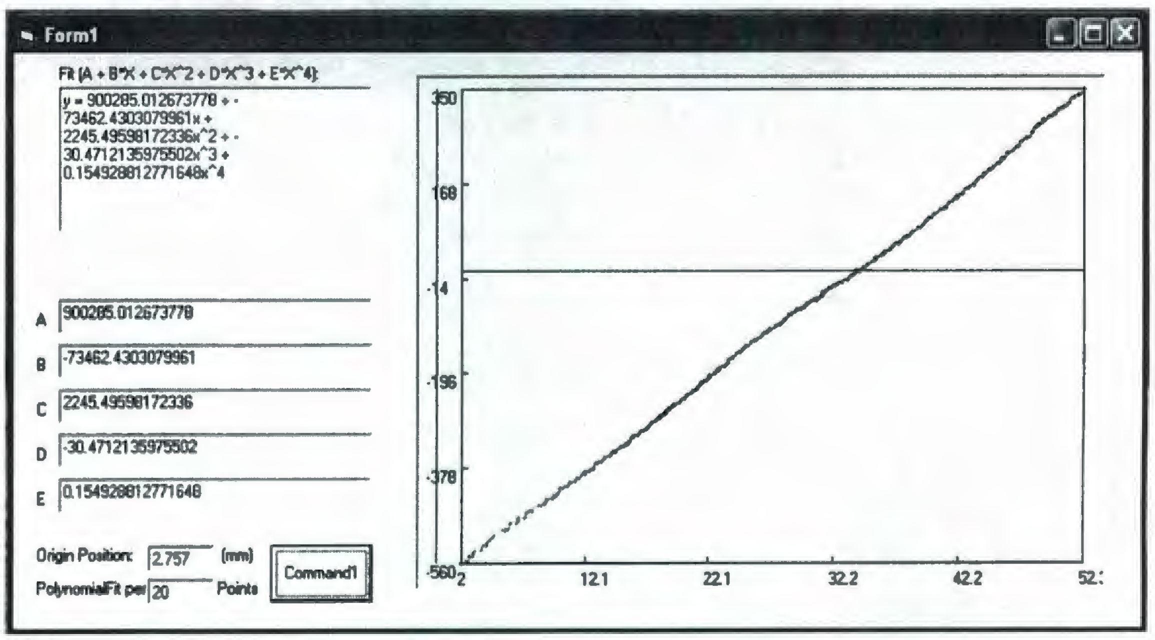

Fig. 3.8 Polynomial fit program panel

3.5 Results and analysis

Figure 3.9 shows the different cantilever curvatures measurements compared to the real curvatures acquired from side view optical images taken with a microscope in our laboratory. During these experiments it was often difficult to obtain an accurate measure of the laser spot at the free end of the cantilever because of the increase in spot size and an increase in scattering. As we can see from these two plots, the measured data (scattered points) are close to the solid lines showing the actual cantilever curvatures.

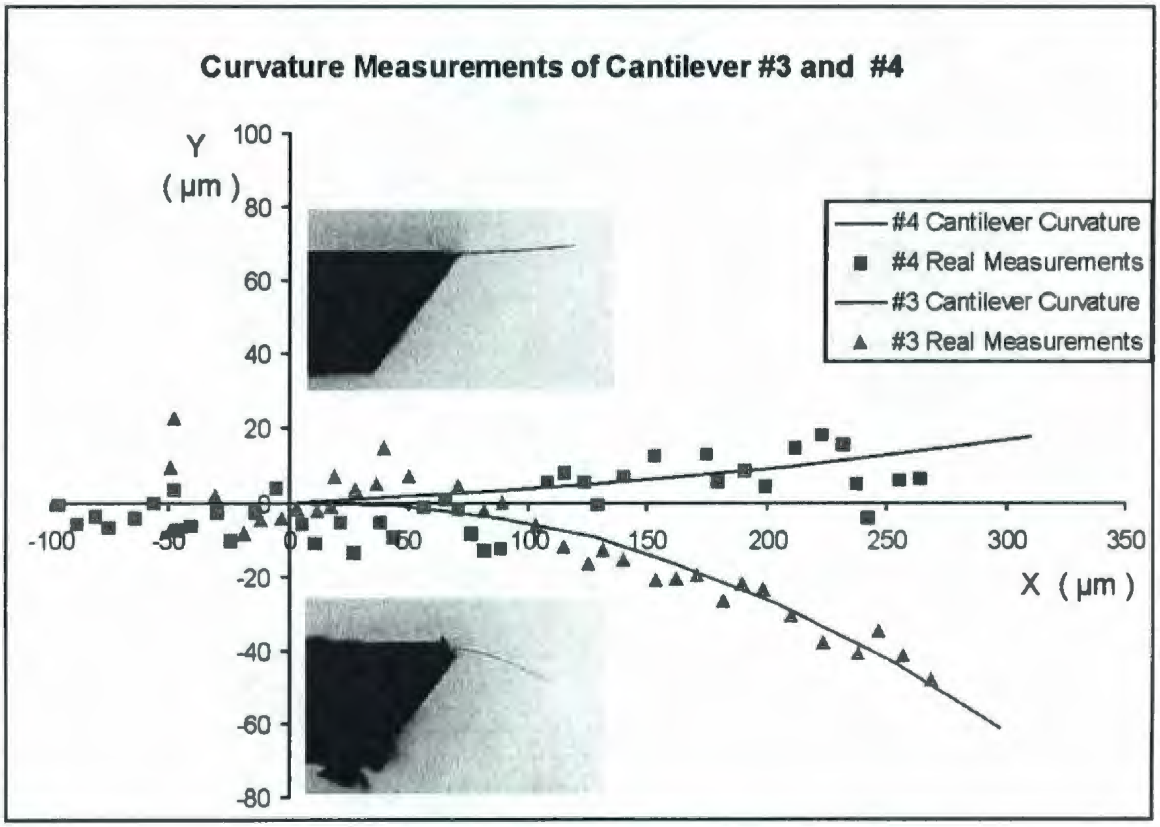

Fig.3.9 Test results of different cantilever curvatures

As discussed in the previous section, the pixel to length conversion factor is a large source of error in these measurements. Increasing the number of pixels would allow us to more accurately identify the position of the laser spot on the cantilever. Also the speed of the motor also plays a large role in the accuracy of the measured data. If it were possible to slow down the motor and take several data points along the length of the lever it would allow us to average the obtained data and reduce the experimental noise shown in figure 3.9.

Chapter Four: Conclusion & Future Work

4.1 Conclusion

We have developed a new method for characterizing the initial state of the cantilever. A new method was developed to define the angle of inclination β of the chip. This method was validated by using three aluminum blocks with known inclination angles. Based on this, the method for finding L_o, initially found by Ye Tian, was modified to include the angle of inclination of the chip β. Lastly, a method was derived to determine the initial curvature of the cantilever. Experiments conducted using deflected cantilever showed the model to be accurate.

4.2 Future work

The work done in this study will allow other researchers to obtain more accurate cantilever sensor measurements. This work will also allow future members of our group to use the deposition system designed by Ye Tian and Mike Coates (fig. 4.1) to study the deposition of Au on Si cantilevers. Using this system will hopefully allow us to develop the means of depositing stress free Au films on Si cantilevers.

Fig.4.1 Deposition system for Au thermal deposition

Reference:

[1] G. Binning, CH. Gerber, and C.F. Quate, *Atomic Force Microscope, Phys. Rev. Lett.* 1986. **56**: p. 930-933.

[2] A. Gupta, D. Akin, and R. Bashir, *Single Virus Particle Mass Detection Using Microresonators with Nanoscale Thickness, Appl. Phys. Lett.* 2004. **84**: p. 1976-1978.

[3] E. A. Wachter and T. Thundat, *Micromechanical Sensors for Chemical and Physical Measurements, Rev. Sci. Instrum.*, 1995. **66**(6): p. 3662-3667.

[4] F. M. Battiston, et al., *A chemical sensor based on a microfabricated cantilever array with simultaneous resonance-frequency and bending readout. Sens. Actuators B*, 2001. **B76** (1-3): p. 393-402.

[5] H. F. Ji and T. Thundat, *In situ detection of calcium ions with chemically modified microcantilevers. Biosens. Bioelectron.*, 2002. **17**(4): p. 337-343.

[6] Yang, Y., H. F. Ji, and T. Thundat, *Nerve agents detection using a Cu2+/L-cysteine bilayer coated microcantilever. J. Am. Chem. Soc.*, 2003. **125**: p. 1124-1125.

[7] Bottomley, L.A., et al., *Microcantilever apparatus and methods for detection of enzymes*. 2002, Protiveris Inc.

[8] C. Grogan, et al., *Characterisation of an antibody coated microcantilever as a potential immuno-based biosensor. Biosens. Bioelectron.*, 2002. **17**(3): p. 201-207.

[9] K. S. Hwang, et al., *Dominant surface stress driven by biomolecular interactions in the dynamical response of nanomechanical microcantilevers, Korea Institute of Science and Technology: Seoul*. p. 15.

[10] M. Su, S. Li, and V. P. Dravid, *Microcantilever resonance-based DNA detection with nanopartical probes. Appl. Phys. Lett.*, 2003. **82**(20): p. 3562-3564.

[11] A. Subramanian, et al., *Glucose biosensing using an enzyme-coated microcantilever. Appl. Phys. Lett.*, 2002. **81**(2): p. 385-387.

[12] T. Thundat, *Microcantilever biosensors*. Scanning, 2001. **23**(2): p. 129.

[13] D. W. Dareinga, and T. Thundat, *Simulation of adsorption-induced stress of a microcantilever sensor. J. Appl. Phys.*, 2005. **97**.

[14] M. Godin, et al., *Quantitative surface stress measurements using a microcantilever. Appl. Phys. Lett.*, 2001. **79**(4): p. 551-553.

[15] Z. Hu, T. Thundat, and R. J. Warmack, *Investigation of adsorption and absorption-induced stresses using microcantilever sensors. J. Appl. Phys.*, 2001. **90**(1): p. 427-431.

[16] L. Y. Beaulieu, M. Godin, O. Laroche, V. Tabard-Cossa, P. Grütter, *A Complete Analysis of the Laser Beam Deflection Systems Used in Cantilever-based Systems, Ultramicroscopy*, 107 (2007) 422-430.

[17] Ye Tian, *A System for Studying Surface Stress Suffered by Cantilevers during Thermal Deposition*, Honours Thesis, Department of Physics and Physical Oceangraphy.2006, Memorial University of Newfoundland: St John's.

Appendix A:
Visual Basic Program Code for Data Acquiring Process

```
'**************************************************************************
'Analyse the position of laser point after data acquisition
'>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
'Two results are related together with SYSTEM TIME
'>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
'Write dial gauge reading to a text box
'**************************************************************************


Private Type POINT_TYPE
   x As Long
   y As Long
End Type

Private Declare Function GetPixel Lib "gdi32" (ByVal hdc As Long, ByVal x As Long,
ByVal y As Long) As Long
Private Declare Function MoveToEx Lib "gdi32" (ByVal hdc As Long, ByVal x As Long,
ByVal y As Long, lpPoint As POINT_TYPE) As Long
Private Declare Function LineTo Lib "gdi32" (ByVal hdc As Long, ByVal x As Long,
ByVal y As Long) As Long


Option Explicit
Private Data() As Double
Private taskHandle As Long
Private taskIsRunning As Boolean
Private StopFlag As Boolean
Private PauseFlag As Boolean
Private StartTime As Double
Dim PrevPressure As String
Dim StartDataPoint As Integer
Dim StartDataFile As Integer
Dim StartDataFileText As String
Dim PointSelect As Integer
Dim AreaX1, AreaY1, AreaX2, AreaY2 As Integer
Dim DoTheAverage As Boolean
Dim PSDaverage, newsystime As Double
Dim PSDcounts As Integer
Dim SystemTimeCount As Integer
```

```
Dim currentx1, currentx2, currenty1, currenty2 As Integer


Private Sub chkMultiFile_Click()
  If Me.chkMultiFile.Value = 1 Then
    Me.txtPointsPerFile.Enabled = True
  Else
    Me.txtPointsPerFile.Enabled = False
  End If

End Sub

Private Sub AutoScaleTimer_Timer()

Call GetNewImage
Call FindLaserPosition
Dim xRegion, yRegion, CantiEnd, FifthYRegion As Integer
Dim pt As POINT_TYPE
Dim retval As Long
xRegion = Mainform.LaserPositionX.Text
yRegion = Mainform.LaserPositionY.Text
CantiEnd = Mainform.CantiEnd.Text

If SaveImageCheck.Value = 1 Then
Call SaveImage
End If

End Sub

Public Sub GetNewImage()

Display.AutoRedraw = False
'Get the interface name and load the parameters set in
'the IMAQ Configuration Utility
CWIMAQ1.Interface = "img0"
CWIMAQ1.LoadInterfaceDefaults

'Acquire asynchronously one buffer
CWIMAQ1.AcquireImage

'Display the most recently acquired picture in a Picture Box
'Note that it could be done more simply with the CWIMAQViewer object
'whose demo version is given, see the "Snap in CWIMAQViewer" sample
 CWIMAQ1.WindowPlot Display.hWnd
```

```
End Sub

Private Sub Clear_Click()
CurrentX = 0
CurrentY = 0
LeftTop.Caption = "LeftTop:"
RightBottom.Caption = "RightBottom:"
PointSelect = 1
AreaX1 = AreaX2 = AreaY1 = AreaY2 = 0
Mainform.MovingLaserPointX.Text = 0
Mainform.MovingLaserPointY.Text = 0
Mainform.LaserPositionX.Text = 0
Mainform.LaserPositionY.Text = 0
Mainform.Distance.Text = 0
Mainform.CantiEnd.Text = 0
Mainform.CantiWidth.Text = 0
Call GetNewImage
End Sub

Private Sub Load_Click()
Call GetNewImage
End Sub


Private Sub PointFound_Click()
PointFound.Default = True
End Sub

Private Sub PositionAnalysis_Click()

Dim ImagePath, NewImagePath, nString, OutputDataPath As String
Dim TotalNumber, Number As Integer
Dim LaserDistance, i As Integer
Dim PI, LASERangle, LaserAnglePrime, PSDangle, Time, Phi, PhiPrime, Delta,
PSDrange As Double
Dim DValue(2), VValue(2), TValue(2)      'DValue is distance, VValue is voltage,
TValue is time
Dim DeltaPx, DeltaPy, DeltaU, DeltaH, VDelta, TDelta, LaserX, LaserY, PSDX, PSDY,
v, LNot, XNotPrime, YNotPrime, Velocity As Double
Dim tmp, CantileverAngle, CantileverNormal, Interface, ChipAngle, AverageChipAngle,
ChipAngleDegree, AverageV As Double
Dim ChipAngleCount, Vcount As Integer
Dim AnalyzedData As String
Dim OriginVoltage, OriginTime, DeltaUSquare As Double
Dim ChipAngleArrayR(3, 100) As Double
Dim ChipAngleArrayI(3, 100) As Double
```

```
Dim XvalueR(3), XvalueI(3) As Double
PI = 4 * Atn(1)

PSDangle = Val(frmPSDAngle.Text) * PI / 180      'These angles are now in radians
LASERangle = Val(frmLaserAngle.Text) * PI / 180      'These angles are now in radians

TotalNumber = Int(Mainform.Interface.Text)
Number = 0
ImagePath = SaveImageText.Text
PSDrange = Val(Mainform.PSDrangeText)

If SaveDataCheck.Value = 1 Then
Open OutputDataFile.Text For Input As #8
i = Len(OutputDataFile.Text)
OutputDataPath = Left(OutputDataFile.Text, i - 8) & "Output.dat"
Open OutputDataPath For Output As #9

i = 0
Do While EOF(8) = False
  If i = 0 Then
    Input #8, tmp, tmp, tmp, tmp, tmp, tmp, tmp, tmp
    Write #9, "Interface", "Time", "DValue", "VValue", "Angle", "Normal"
    ChipAngleCount = 0
  ElseIf i = 1 Then
    Input #8, TValue(2), tmp, tmp, tmp, tmp, tmp, VValue(2), Number

    If VValue(2) <> "#" Then

      If Number < 10 Then
        nString = "00" & CStr(Number)
      ElseIf Number > 9 And Number < 100 Then
        nString = "0" & CStr(Number)
      Else
        nString = CStr(Number)
      End If


      NewImagePath = ImagePath & nString & ".bmp"

      Display.Picture = LoadPicture(NewImagePath)
      Me.Caption = "image " & Number

      Call FindLaserPosition

      DValue(2) = Val(Mainform.Distance.Text)
      Write #9, Number, TValue(2), DValue(2), VValue(2), " ", " "
```

```
        DValue(1) = DValue(2)
        VValue(1) = VValue(2)
        TValue(1) = TValue(2)
      Else
       i = i - 1
      End If

    Else
      Input #8, TValue(), tmp, tmp, tmp, tmp, tmp, VValue(2), Number2

      If VValue(2) <> "#" And VValue(2) <> 0 And Number < TotalNumber Then

        If Number < 10 Then
          nString = "00" & CStr(Number)
        ElseIf Number > 9 And Number < 100 Then
          nString = "0" & CStr(Number)
        Else
          nString = CStr(Number)
        End If


        NewImagePath = ImagePath & nString & ".bmp"

        Display.Picture = LoadPicture(NewImagePath)
        Me.Caption = "Image " & Number

        Call FindLaserPosition

        DValue(2) = Val(Mainform.Distance.Text)


        DeltaPx = DValue(2) - DValue(1)
        VDelta = VValue(2) - VValue(1)
        DeltaH = VDelta * 10 / PSDrange 'DeltaH in the units of mm
        TDelta = TValue(2) - TValue(1)


          If DValue(2) <= 0 And DValue(1) <> 0 And DeltaPx <> Empty And DeltaPx
<> 0 Then
            'laser point on the chip
            'calculate the ChipAngle and Velocity
            'DeltaPy = DeltaPx * Tan(LaserAngle) - 2 * VDelta * Sin(PSDAngle)
            'CantileverAngle = -Atn(DeltaPy / DeltaPx)
            'CantileverNormal = Atn(-DeltaPx / DeltaPy)
            'AverageChipAngle = AverageChipAngle + CantileverAngle
```

```
            Call VietaTheoremModule.SolveCubicEquation

            ChipAngleArrayR(1, ChipAngleCount) = XvalueR(1)
            ChipAngleArrayI(1, ChipAngleCount) = XvalueI(1)
            ChipAngleArrayR(2, ChipAngleCount) = XvalueR(2)
            ChipAngleArrayI(2, ChipAngleCount) = XvalueI(2)
            ChipAngleArrayR(3, ChipAngleCount) = XvalueR(3)
            ChipAngleArrayI(3, ChipAngleCount) = XvalueI(3)

            v = DeltaPx / TDelta
            AverageV = AverageV + v
            Vcount = Vcount + 1
            Write #9, Number, TValue(2), DValue(2), VValue(2), _
            ChipAngleArrayR(1, ChipAngleCount), ChipAngleArrayI(1,
ChipAngleCount); _
            ChipAngleArrayR(2, ChipAngleCount), ChipAngleArrayI(2,
ChipAngleCount); _
            ChipAngleArrayR(3, ChipAngleCount), ChipAngleArrayI(3,
ChipAngleCount)
            DValue(1) = DValue(2)
            VValue(1) = VValue(2)
            ChipAngleCount = ChipAngleCount + 1
        ElseIf DValue(2) > 0 And DValue(2) <= 350 And DeltaPx <> 0 Then
            'laser point on the cantilever
            'when laser point goes on the cantilever,
            'use chip angle value to calculate the cantilever angle.
            If Mainform.frmChipAngle.Text = "" Then
               'find the best fit of chip angle in the array
               Call BestFitModule.BestFit(ChipAngleCount)

               ChipAngle = 0
               'ChipAngle is still in radians, on panel we convert it into degrees for
consistance
               ChipAngleDegree = 180 * ChipAngle / PI
               Mainform.frmChipAngle.Text = ChipAngleDegree
               'Lnot must be defined before experiments
               LNot = Val(Mainform.frmLNot.Text)
               'calculate the XNotPrime and yNotPrime based on LNot and ChipAngle
               XNotPrime = LNot * Cos(LASERangle - 2 * ChipAngle)
               YNotPrime = LNot * Sin(LASERangle - 2 * ChipAngle)
               'calculate the velocity of laser
               Mainform.frmVelocity.Text = AverageV / Vcount
               'get the DValue and VValue when laser point hit the origin
               OriginVoltage = Val(Mainform.OriginVoltage.Text)
               OriginTime = Val(Mainform.OriginTime.Text)
            End If
```

```
            'incident laser is LaserY=-
tan(LaserAngle)*laserX+tan(LaserAngle)*Velocity*TDelta
            Velocity = Val(Mainform.frmVelocity.Text)
            TDelta = TValue(2) - OriginTime
            LaserX = DValue(2)
            LaserY = -Tan(LASERangle) * LaserX + Tan(LASERangle) * Velocity *
TDelta

            'PSD equation is PSDY = -Tan(PSDAngle) * (PSDX - XNotPrime) +
YNotPrime
            DeltaU = VValue(2) - OriginVoltage
            PSDX = -XNotPrime + Abs(DeltaU) * Cos(PSDangle)  'PSDX>0
            PSDY = -Tan(PSDangle) * (PSDX - XNotPrime) + YNotPrime

            Delta = -Atn((LaserY - PSDY) / (LaserX - PSDX))
            CantileverNormal = (PI - LASERangle - Delta) / 2
            CantileverAngle = (LASERangle + Delta) / 2
            Write #9, Number, TValue(2), DValue(2), CantileverAngle,
CantileverNormal

            Else ' DeltaPx=0 means DValue(1)= DValue(2)
            i = i - 1
          End If
        End If
      End If


  i = i + 1

Loop

Close #8
Close #9


End If

Me.Caption = "Done!!!"
End Sub


Private Sub PositionAnalysis_Click()
Dim Interface, i As Integer
Dim iTime As Double
Dim nString, ImagePath, NewImagePath As String
```

```
Dim YesNo As Integer

If SaveDataCheck.Value = 1 Then
Open "C:\Documents and Settings\Josh\Desktop\timeimage.dat" For Input As #8
Open "C:\Documents and Settings\Josh\Desktop\output.csv" For Output As #9

ImagePath = Mainform.SaveImageText.Text

i = 0
Do While EOF(8) = False
   'ReDim Preserve ITime(i)

   If i = 0 Then

         Input #8, iTime, Interface

         If Interface < 10 Then
            nString = "00" & CStr(Interface)
         ElseIf Interface > 9 And Interface < 100 Then
            nString = "0" & CStr(Interface)
         Else
            nString = CStr(Interface)
         End If


         NewImagePath = ImagePath & nString & ".bmp"

         Display.Picture = LoadPicture(NewImagePath)
         Me.Caption = "image " & Interface

         Call FindLaserPosition

         YesNo = MsgBox("Is this the right point?", vbYesNo)
         If YesNo = 6 Then
         ElseIf YesNo = 7 Then
         While (PointFound.Default = False)
         DoEvents
         Wend
         End If

         PointFound.Default = False

         Write #9, Interface, iTime, Val(Mainform.Distance.Text)

   Else
```

```
        Input #8, iTime, Interface

        If Interface < 10 Then
           nString = "00" & CStr(Interface)
        ElseIf Interface > 9 And Interface < 100 Then
           nString = "0" & CStr(Interface)
        Else
           nString = CStr(Interface)
        End If


        NewImagePath = ImagePath & nString & ".bmp"

        Display.Picture = LoadPicture(NewImagePath)
        Me.Caption = "image " & Interface

        Call FindLaserPosition

        YesNo = MsgBox("Is this the right point?", vbYesNo)
        If YesNo = 6 Then
        ElseIf YesNo = 7 Then
        While (PointFound.Default = False)
        DoEvents
        Wend
        End If

        PointFound.Default = False

        Write #9, Interface, iTime, Val(Mainform.Distance.Text)

    End If

   i = i + 1

Loop

Close #8
Close #9


End If

Me.Caption = "Done!!!"
End Sub

Private Sub FindLaserPosition()
```

```
'analyse every image with program first
'find the laser point position according to initial position
Dim points(800, 600) As Long
Dim xPos, yPos, LaserCenterX, LaserCenterY As Integer
Dim pt As POINT_TYPE
Dim retval As Long
Dim s As Integer
Dim SumX, SumY, SumN As Integer
Dim PointSize, PointSizeNew As Integer
Dim yMaxGreen, yMinGreen, xMaxGreen, xMinGreen, xLaser, yLaser, yScanMax,
OriginY As Integer
yMaxGreen = -10000
yMinGreen = 10000
xMaxGreen = -10000
xMinGreen = 10000
Dim Distance, CantiWidth, CantiLength, RealDistance As Integer
Dim GreenValue, BlackValue, CyanValue As Long
GreenValue = GreenText.Text
BlackValue = BlackText.Text
CyanValue = CyanText.Text

LaserCenterX = Mainform.MovingLaserPointX.Text
LaserCenterY = Mainform.MovingLaserPointY.Text

   If LaserCenterX < 20 Then LaserCenterX = 20
   If LaserCenterY < 20 Then LaserCenterY = 20

   'mask scan area according to teh moving position of laser point
   For xPos = LaserCenterX - 20 To LaserCenterX + 20
      For yPos = LaserCenterY - 20 To LaserCenterY + 20

         points(xPos, yPos) = GetPixel(Display.hdc, xPos, yPos)

         If points(xPos, yPos) > GreenValue Then
            points(xPos, yPos) = vbGreen
            PointSize = PointSize + 1
         ElseIf points(xPos, yPos) > BlackValue And points(xPos, yPos) <=
GreenValue Then
            points(xPos, yPos) = vbBlack
         ElseIf points(xPos, yPos) > CyanValue And points(xPos, yPos) <= BlackValue
Then
            points(xPos, yPos) = vbCyan
         ElseIf points(xPos, yPos) <> vbBlack And points(xPos, yPos) <> vbGreen And
points(xPos, yPos) <> vbCyan Then
            points(xPos, yPos) = vbBlue
```

```
        End If

      Next
  Next


'display the mask or not?
If Mask.Value = 1 Then
For xPos = LaserCenterX - 50 To LaserCenterX + 50
   For yPos = LaserCenterY - 50 To LaserCenterY + 50
         Display.PSet (xPos, yPos), points(xPos, yPos)
      Next
   Next
End If

'find the laser point
For xPos = LaserCenterX - 20 To LaserCenterX + 20
   For yPos = LaserCenterY - 20 To LaserCenterY + 20
      If points(xPos, yPos) = vbGreen Then
      SumX = SumX + xPos
      SumY = SumY + yPos
      SumN = SumN + 1
      End If
   Next
Next

If SumN <> 0 Then
'find the laser point center by weighting the x and y coordinates
xLaser = SumX / SumN
yLaser = SumY / SumN

If xLaser > 0 And xLaser < 800 And yLaser > 0 And yLaser < 600 Then
Mainform.MovingLaserPointX.Text = xLaser
Mainform.MovingLaserPointY.Text = yLaser
End If

Display.Circle (xLaser, yLaser), 5, RGB(255, 0, 0)

OriginY = Mainform.LaserPositionY.Text

Distance = LaserCenterY - OriginY 'distance in pixels
CantiWidth = Mainform.CantiWidth.Text 'in pixels
'the real width of cantilever is 35 micros
RealDistance = Distance * 35 / CantiWidth 'in micros
Mainform.Distance.Text = RealDistance
```

```
        End If

End Sub

'Scale within the selected area!!!

Private Sub Scale_Click()

'Call GetNewImage

If PointSelect = 1 Then
MsgBox "Please select the scanning area!!!"
Else

Dim points(800, 600) As Long
Dim xPos, yPos As Integer
Dim pt As POINT_TYPE
Dim retval As Long
Dim So, Sx, Sy, Sxx, Sxy, D, A, B As Double
Dim s As Integer
Dim PointSize, PointSizeNew As Integer

So = 0
Sx = 0
Sy = 0
Sxx = 0
Sxy = 0

Dim yMaxGreen, yMinGreen, xMaxGreen, xMinGreen, xLaser, yLaser, yScanMax As
Integer
yMaxGreen = -10000
yMinGreen = 10000
xMaxGreen = -10000
xMinGreen = 10000

Dim GreenValue, BlackValue, CyanValue As Long
GreenValue = GreenText.Text
BlackValue = BlackText.Text
CyanValue = CyanText.Text

    'masking:  sets the scale regions to known colours so that we can do the math later...
    For xPos = AreaX1 To AreaX2
       For yPos = AreaY1 To AreaY2

          points(xPos, yPos) = GetPixel(Display.hdc, xPos, yPos)
```

```
        If points(xPos, yPos) > GreenValue Then
           points(xPos, yPos) = vbGreen
        ElseIf points(xPos, yPos) > BlackValue And points(xPos, yPos) <= GreenValue
Then
           points(xPos, yPos) = vbBlack
        ElseIf points(xPos, yPos) > CyanValue And points(xPos, yPos) <= BlackValue
Then
           points(xPos, yPos) = vbCyan
        ElseIf points(xPos, yPos) <> vbBlack And points(xPos, yPos) <> vbGreen And
points(xPos, yPos) <> vbCyan Then
           points(xPos, yPos) = vbBlue
        End If

     Next
  Next

  'display the mask?
  If Mask.Value = 1 Then
     For xPos = AreaX1 To AreaX2
        For yPos = AreaY1 To AreaY2
           Display.PSet (xPos, yPos), points(xPos, yPos)
        Next
     Next
  End If


  'lets find the MIN x value where the colour is blue
  'ie: lets describe the line of the edge of the left-most chip
  'but we need to find the horizontal edge first
  Dim xMaxBlue, MaxBlueX, MaxBlueY As Integer
  Dim FindMaxBlue As Boolean
  FindMaxBlue = False
  MaxBlueX = 9999
  MaxBlueY = 9999
  'search the up-right conner for the most left point of chip
  For xPos = AreaX1 To AreaX1 + 50
     For yPos = AreaY1 To AreaY1 + 50
     If FindMaxBlue = False Then
     'points(xPos, yPos - 2) = vbCyan And        And points(xPos + 2, yPos) = vbCyan
     If points(xPos, yPos - 1) = vbCyan And points(xPos, yPos) = vbCyan Then
     If points(xPos, yPos) = vbCyan And points(xPos + 1, yPos) = vbCyan Then
        MaxBlueX = xPos
        MaxBlueY = yPos
        xMaxBlue = MaxBlueX
        FindMaxBlue = True   'To make sure the first suitable point is the most left point
of chip
```

```
        End If
        End If
        End If
        Next
    Next

    Mainform.MaxBlue.Caption = "MaxBlue:" & MaxBlueX & "," & MaxBlueY

    'if we need to find the equation of cantilever edge then do the maths as follows
    'lets find the maximum y values on x where the colour is blue
    'ie: lets describe the line of the cantilever chip

    Dim yMaxBlue() As Integer
    Dim CantiLeftEdge() As Integer
    Dim CantiRightEdge() As Integer

    Dim l As Integer
    Dim Average As Long

    l = 0
    Average = 0

    'find the chip's horizontal edge if we need
    For yPos = AreaY1 To AreaY2
        For xPos = xMaxBlue To AreaX2
            ' find points on the edge
            If points(xPos, yPos) = vbBlue And points(xPos, yPos - 1) = vbBlue And
points(xPos, yPos - 2) = vbBlue Then
                If points(xPos, yPos + 1) <> vbBlue And points(xPos, yPos + 2) <> vbBlue
And points(xPos, yPos + 3) <> vbBlue Then
                    ReDim yMaxBlue(xPos)
                    yMaxBlue(xPos) = yPos
                    Display.PSet (xPos, yPos), vbRed
                        If yMaxBlue(xPos) <> 0 Then
                        So = So + 1
                        Sx = Sx + xPos
                        Sy = Sy + yMaxBlue(xPos)
                        Sxx = Sxx + xPos ^ 2
                        Sxy = Sxy + xPos * yMaxBlue(xPos)
                        End If
                End If
            End If
        Next
    Next
```

```
If Mainform.CheckFindCantiEdge.Value = 1 Then
'calculate the chip edge
D = So * Sxx - Sx ^ 2
A = (Sxx * Sy - Sx * Sxy) / D
B = (So * Sxy - Sx * Sy) / D

'now, the line of the cantilever chip is just y=bx+a
Dim y0, yM As Double
y0 = A
'and at xmax
yM = B * AreaX2 + A

'define scan regions
xLaser = CInt(Val(Mainform.LaserPositionX.Text))
yLaser = CInt(Val(Mainform.LaserPositionY.Text))


Dim xLaserO, yLaserO, aPrime, bPrime As Long
'now, consider that the chip and cantilever are perpendicular
'two lines are said to be perpendicular if the product of their slopes is -1

bPrime = -1 / B
aPrime = yLaser - bPrime * xLaser

xLaserO = (aPrime - A) / (B - bPrime)

yLaserO = bPrime * xLaserO + aPrime
'so the line describing the cantilever is y = bPrime* x+ aPrime

Display.ForeColor = vbRed
retval = MoveToEx(Display.hdc, xLaserO, yLaserO, pt)
retval = LineTo(Display.hdc, xLaser, yLaser)

Mainform.yMaxBlue.Caption = "yMaxBlue: y=" & B & "*x+" & A & "."
Else
Mainform.yMaxBlue.Caption = "yMaxBlue: y=..."
End If



'To find the cantilever width
'let's average the difference between the right edge and the left edge on the middle
canti
'(take the width of canti as 20)
Dim CantLeftX As Integer
Dim CantRightX As Integer
```

```
    Dim xCant, yCant As Integer
    Dim LeftPoint As Integer
    Dim FindLeftEdge, FindRightEdge As Boolean
    Dim CantiEnd As Integer
    Dim FifthYRegion As Integer
    Dim TempString As String
    Dim xRegion, yRegion As Integer

    xRegion = xLaser
    yRegion = yLaser

    For yPos = yRegion - 20 To yRegion + 90
    FindLeftEdge = False
    FindRightEdge = False
       For xPos = xRegion - 25 To xRegion + 25
          'find the cantilevers' left edges
          If points(xPos - 3, yPos) = vbBlack And points(xPos - 2, yPos) = vbBlack And
points(xPos - 1, yPos) = vbBlack Then
             If points(xPos, yPos) = vbCyan And points(xPos + 1, yPos) = vbCyan And
points(xPos + 2, yPos) = vbCyan Then
                ReDim CantiLeftEdge(yPos)
                CantiLeftEdge(yPos) = xPos
                Display.PSet (xPos, yPos), vbWhite
                FindLeftEdge = True

                'if we can find the left edge of canti then we look for the right edge
                For xCant = xPos To xRegion + 25
                If points(xCant + 3, yPos) = vbBlack And points(xCant + 2, yPos) = vbBlack
And points(xCant + 1, yPos) = vbBlack Then
                   If points(xCant, yPos) = vbCyan And points(xCant - 1, yPos) = vbCyan
And points(xCant - 2, yPos) = vbCyan Then
                      ReDim CantiRightEdge(yPos)
                      CantiRightEdge(yPos) = xCant
                      Display.PSet (xCant, yPos), vbMagenta
                      FindRightEdge = True
                   End If
                End If
                Next

             End If
          End If

       Next

    'if we can find both left and right edge points with the same yPos, then calculate the
width
```

```
    'mark the last yPos as the end of cantilver, and show it on the text box
    If FindLeftEdge = True And FindRightEdge = True Then
    Average = Average + CantiRightEdge(yPos) - CantiLeftEdge(yPos)
    l = l + 1
    CantiEnd = yPos
    Mainform.CantiEnd.Text = CantiEnd
    End If
    Next

    End If


End Sub


Private Sub SaveImage()
Dim TempString As String
Dim kString, path, newpath As String
Dim systime As Double
Dim Hour, Minute, Second As String

If Val(Interface.Text) < 10 Then
   kString = "00" & Interface.Text
ElseIf Val(Interface.Text) > 9 And Val(Interface.Text) < 100 Then
   kString = "0" & Interface.Text
Else
   kString = Interface.Text
End If

CWIMAQ1.AcquireImage

path = SaveImageText.Text
newpath = path & kString & ".bmp"

CWIMAQ1.SaveImageToDisk newpath, CWIMAQ1.Images(1)

Interface.Text = Interface.Text + 1
Display.Picture = LoadPicture(newpath)

SystemTime.Text = Format(Now, "hh:nn:ss") & "." & Right(Format(Timer, "#0.00"), 2)
Hour = Val(Left(SystemTime.Text, 2)) * 3600
Minute = Left(SystemTime.Text, 5)
Minute = Val(Right(Minute, 2)) * 60
Second = Val(Right(SystemTime.Text, 5))
systime = Hour + Minute + Second
```

```
If newsystime > systime Then newsystime = systime

systime = systime - newsystime

TempString = systime & "," & Interface.Text
Print #1, TempString

End Sub


Private Sub SaveDataYesNo_Click()
   If SaveDataYesNo.Value = False Then
      Me.optAllData.Enabled = False
      Me.txtScaleFactor.Enabled = False
      Me.optShiftAxis.Value = True
   Else
      Me.optAllData.Enabled = True
      Me.txtScaleFactor.Enabled = True
      Me.optAllData.Value = True
   End If
End Sub

Private Sub startCommandButton_Click()
   Dim sampsPerChanRead As Long
   Dim numChannels As Long
   Dim fillMode As DAQmxFillMode
   Dim bufferSize As Long
   Dim numSampsPerChannel As Long
   Dim arraySizeInSamps As Long
   Dim Channels As String
   Dim TempChannel As String

   Dim XMin, YMin, YMax, XMax As Double
   Dim XOrigin, YOrigin As Double
   Dim ScaleX, ScaleY As Double
   Dim XOriginT, YOriginT, ScaleXT, ScaleYT As Double
   Dim XOrigin2, YOrigin2, ScaleX2, ScaleY2 As Double

   Dim count As Long
   Dim i, k As Long
   Dim temp_i As Long
   Dim j As Long
   Dim item As ListItem
   Dim InputData() As Double
   Dim SumVoltage() As Double
   Dim AveVoltage() As Double
```

```
Dim PreviousTime, pAVGtime, pAVGvoltage() As Double
Dim Time, TotalTime As Double
Dim DeltaV, DeltaT As Double
Dim n As Integer
Dim SumTime, AveTime As Double
Dim Temperature As Double
Dim FirstRun  As Boolean
Dim ThermType As DAQmxThermocoupleType1
Dim PSD1Offset As Double
Dim PSD2Offset As Double
Dim Pressure As String



Mainform.startCommandButton.Enabled = False

FirstRun = True

If Mainform.AutoScaleCheck.Value = 1 Then
Mainform.AutoScaleTimer.Interval = Val(Mainform.AutoScaleText.Text) * 1000
Mainform.AutoScaleTimer.Enabled = True
End If

DoEvents

Me.txtVoltPrecision.Enabled = False
StopFlag = False

'Checks to see that all the fields aren't blank.
If ValidateControlValues Then
   startCommandButton.Enabled = True
   Exit Sub
End If

'Tells the program how to list the data in the array.
'If it is Scan Number, it lists all the first sample points collected from each channel,
'then the second points from each channel, etc.
'If it is Channel, it lists all the sample points from Channel 1, then Channel 2, etc.
If scanOrderOption.Value = True Then
   fillMode = DAQmx_Val_GroupByScanNumber
Else
   fillMode = DAQmx_Val_GroupByChannel
End If

'Tell the program how to collect data samples from the channels.
'If it is Average, take single samples from each channel, which are
```

```
'generated very quickly, and then average samples over the time interval.
'If it is Collect Data every so often, takes a single sample every so many
'samples which are generated on each channel, and can specify the rate
'individual samples are generated in Hz.
bufferSize = 255
If ChangeOption.Value = True Then
   numSampsPerChannel = 1
ElseIf TimeOption.Value = True Then
   numSampsPerChannel = CLng(samplesPerChannelTextBox.Text)
End If


'Create the DAQmx task, and a boolean to say it is running.
DAQmxErrChk DAQmxCreateTask("", taskHandle)
taskIsRunning = True


'This is a string of all the channels, which comes from the Function,
'so we get "Channels = Dev1/ai0,Dev1/ai1,Dev1/ai2" and so on.
Channels = DetermineChannels()
TempChannel = DetermineTempChannel()
ThermType = DetermineThermType()

'Add an analog input channel to the task.
 DAQmxErrChk DAQmxCreateAIVoltageChan(taskHandle, Channels, "", _
            DAQmx_Val_Cfg_Default, minValueTextBox.Text,
maxValueTextBox.Text, _
            DAQmx_Val_VoltageUnits1_Volts, "")
 DAQmxErrChk DAQmxCreateAIThrmcplChan(taskHandle, TempChannel, "",
MinTempYRange.Text, MaxTempYRange.Text, DAQmx_Val_DegC,
DAQmx_Val_ThermocoupleType1_K_Type_TC, DAQmx_Val_CJCSource1_ConstVal,
Int(Me.txtCalibTemp.Text), "")

'Configure task for finite sample acquisition and read in data
DAQmxErrChk DAQmxCfgSampClkTiming(taskHandle, "OnboardClock",
frequencyTextBox.Text, DAQmx_Val_Rising,
DAQmx_Val_AcquisitionType_FiniteSamps, CLng(samplesPerChannelTextBox.Text))
DAQmxErrChk DAQmxGetTaskNumChans(taskHandle, numChannels)
arraySizeInSamps = numSampsPerChannel * numChannels
ReDim Data(arraySizeInSamps)


'acquiringLabel.Visible = True
acquiringLabel.Caption = "Acquiring..."

XMin = 0: XMax = 60
```

```
    Call GraphingModule.InitiateGraph(XMin, XMax, XOrigin, YOrigin, ScaleX, ScaleY,
XOriginT, YOriginT, ScaleXT, ScaleYT, XOrigin2, YOrigin2, ScaleX2, ScaleY2)

    i = 0
    ReDim Preserve InputData(numChannels + 1) 'Time, Channel 0, Channel 1, Channel
3, ...
    ReDim Preserve SumVoltage(numChannels + 1)
    ReDim Preserve AveVoltage(numChannels + 1)
    ReDim Preserve pAVGvoltage(numChannels + 1)

     For j = 2 To numChannels + 1
       InputData(j) = 0
       FirstRun = True     'This indeicates the first time we read data.
    Next j

    pAVGtime = 0

    StartTime = Timer

    If ChangeOption.Value = True Then

       If SaveDataCheck.Value = 1 Then Open OutputDataFile.Text For Output As #2
       Write #2, "Time", "PSD1", "PSD2", "Temp", "Distance", "D", "V", "Interface",
"Position"
       StartDataFileText = OutputDataFile.Text
       PreviousTime = Timer

       'Here we are saving data when we checked the "Save output Data as?"
       Do While StopFlag = False

          If SaveDataCheck.Value = 1 And OutputDataFile.Text <> StartDataFileText
Then
             Close #1
             Open OutputDataFile.Text For Append As #1
          End If

          DoEvents

          InputData(1) = (Timer - StartTime)

          'Read the Data from the DAQ
          DAQmxErrChk DAQmxReadAnalogF64(taskHandle, numSampsPerChannel,
10#, _
                fillMode, Data(0), arraySizeInSamps, sampsPerChanRead, ByVal 0&)

          'Read the Pressure Data
```

```
Pressure = MSComm1.Input
If Pressure <> lblPressure.Caption Then
   If Pressure = "" Then
      lblPressure.Caption = PrevPressure
   Else
      lblPressure.Caption = Pressure
      PrevPressure = Pressure
   End If
Else
   lblPressure.Caption = PrevPressure
End If


'Here we put the data into an array.
For j = 0 To numChannels - 1
k = Data(1)

   InputData(j + 2) = Strings.FormatNumber(Data(j), 6)
Next j

'This is where we use offset controls to compensate for the
'different max and min photocurrents put out by the PSDs.
'It allows us to use a high current for max resolution without going
'over the +/-10V limit of the NI-DAQ.
PSD1Offset = CDbl(txtPSD1Offset.Text)
PSD2Offset = CDbl(txtPSD2Offset.Text)
InputData(2) = InputData(2) + PSD1Offset
InputData(3) = InputData(3) + PSD2Offset



'This is executed only once at the beginning.
If FirstRun = True Then
   PreviousTime = InputData(1)
   SumTime = InputData(1)
   For k = 2 To numChannels + 1
      SumVoltage(k) = InputData(k)
      pAVGtime = 0
      pAVGvoltage(k) = 0 'InputData(k)
      FirstRun = False
   Next k
   n = 1
ElseIf (InputData(1) - PreviousTime) < Val(AvgTimeInt.Text) Then
   'This is where we have to alter the program in order to elliminate spikes from
the data.
   'What we need to do here is to save all the data to an array
```

```
            SumTime = SumTime + InputData(1)
            For k = 2 To numChannels + 1
               SumVoltage(k) = SumVoltage(k) + InputData(k)
            Next k
            n = n + 1
            'PreviousTime = InputData(1)
         ElseIf (InputData(1) - PreviousTime) >= Val(AvgTimeInt.Text) Then
            'When we get here we look at the number in the array and remove the data
points that have a
            'large standard deviation from the rest of the data.
            AveTime = SumTime / n
            For k = 2 To numChannels + 1
               AveVoltage(k) = SumVoltage(k) / n
            Next k

            If SaveDataCheck.Value = 1 Then Call SaveData(AveTime, AveVoltage,
numChannels)
            Call PlotData(AveTime, AveVoltage(), pAVGtime, pAVGvoltage(),
numChannels, XOrigin, YOrigin, ScaleX, ScaleY, XOriginT, YOriginT, ScaleXT,
ScaleYT, XOrigin2, YOrigin2, ScaleX2, ScaleY2)

            pAVGtime = AveTime
            PreviousTime = InputData(1)
            SumTime = InputData(1)
            For k = 2 To numChannels + 1
               pAVGvoltage(k) = AveVoltage(k)
               SumVoltage(k) = InputData(k)
            Next k
            n = 1
         End If


         If (Me.optAllData.Value = True) And (InputData(1) * 1.1 > XMax) Then
               XMax = InputData(1) * CDbl(Me.txtScaleFactor.Text)
               Call GraphingModule.InitiateGraph(XMin, XMax, XOrigin, YOrigin,
ScaleX, ScaleY, XOriginT, YOriginT, ScaleXT, ScaleYT, XOrigin2, YOrigin2, ScaleX2,
ScaleY2)
               Close #1
               Call RePlotData(XOrigin, YOrigin, ScaleX, ScaleY, XOriginT, YOriginT,
ScaleXT, ScaleYT, XOrigin2, YOrigin2, ScaleX2, ScaleY2)
               If SaveDataCheck.Value = 1 Then Open OutputDataFile.Text For Append
As #1
         End If

         If (Me.optShiftAxis.Value = True) And (Int(InputData(1)) = XMax) Then
               XMax = XMax + CDbl(Me.txtDeltat.Text): XMin = Int(InputData(1))
```

```
            Call GraphingModule.InitiateGraph(XMin, XMax, XOrigin, YOrigin,
ScaleX, ScaleY, XOriginT, YOriginT, ScaleXT, ScaleYT, XOrigin2, YOrigin2, ScaleX2,
ScaleY2)
            Close #1
            'If SaveDataCheck.Value = 1 Then Open OutputDataFile.Text For Append
As #1
            If Mainform.startCommandButton.Enabled = True Then Open
OutputDataFile.Text For Append As #1
        End If
    Loop
  ElseIf TimeOption.Value = True Then

  End If


  'Call the StopTask module to stop the DAQmx task.
  StopTask

  If SaveDataCheck.Value = 1 Then
  Close #1
  Close #2
  End If

  startCommandButton.Enabled = True

  ' Display a message indicating the number of samples per channel read.
  acquiringLabel.Caption = "Stopped!"



'Analyze the output data.

Dim LaserDistance As Integer
Dim PI, LASERangle, LaserAnglePrime, PSDangle, Phi, PhiPrime, Delta, PSDrange As
Double
Dim DValue(2), VValue(2), TValue(2)      'DValue is distance, VValue is voltage,
TValue is time
Dim DeltaPx, DeltaPy, DeltaU, DeltaH, VDelta, TDelta, LaserX, LaserY, PSDX, PSDY,
v, LNot, XNotPrime, YNotPrime, Velocity As Double
Dim tmp, CantileverAngle, CantileverNormal, Interface, ChipAngle, AverageChipAngle,
ChipAngleDegree, AverageV As Double
Dim ChipAngleCount, Vcount As Integer
Dim AnalyzedData As String
Dim OriginVoltage, OriginTime, DeltaUSquare As Double
Dim ChipAngleArrayR(3, 100) As Double
Dim ChipAngleArrayI(3, 100) As Double
```

```
Dim XvalueR(3), XvalueI(3) As Double
Dim Solution(6) As Double

PI = 4 * Atn(1)
PSDangle = Val(frmPSDAngle.Text) * PI / 180            'These angles are now in
radians
LASERangle = Val(frmLaserAngle.Text) * PI / 180          'These angles are now in
radians
PSDrange = Val(Mainform.PSDrangeText)

If SaveDataCheck.Value = 1 Then Open OutputDataFile.Text For Input As #6

i = Len(OutputDataFile.Text)
AnalyzedData = Left(OutputDataFile.Text, i - 8) & "Analyzed.dat"
Open AnalyzedData For Output As #7

i = 0
Do While EOF(6) = False
   If i = 0 Then
      Input #6, tmp, tmp, tmp, tmp, tmp, tmp, tmp, tmp
      Write #7, "Interface", "Time", "DValue", "VValue", "X1R", "X1I", "X2R", "X2I",
"X3R", "X3I"
   ElseIf i = 1 Then
      Input #6, TValue(2), tmp, tmp, tmp, tmp, DValue(2), VValue(2), Interface
      If DValue(2) <> "#" And VValue(2) <> "#" Then
         Write #7, Interface, TValue(2), DValue(2), VValue(2), " ", " ", " "
         DValue(1) = DValue(2)
         VValue(1) = VValue(2)
         TValue(1) = TValue(2)
      Else
       i = i - 1
       ChipAngleCount = 1
      End If
   Else
      Input #6, TValue(2), tmp, tmp, tmp, tmp, DValue(2), VValue(2), Interface
      If DValue(2) <> "#" Then
         If VValue(2) <> "#" And VValue(2) <> 0 Then
            DeltaPx = DValue(2) - DValue(1)
            VDelta = VValue(2) - VValue(1)
            TDelta = TValue(2) - TValue(1)
            DeltaH = VDelta * 10 / PSDrange 'DeltaH in the units of mm

            If DValue(2) <= 0 And DValue(1) <> 0 And DeltaPx <> 0 Then
               'laser point on the chip
               'calculate the ChipAngle and Velocity
```

```
            Call VietaTheoremModule.SolveCubicEquation((DeltaPx), (DeltaH),
(PSDangle), (LASERangle), Solution)

            ChipAngleArrayR(1, ChipAngleCount) = Solution(1)
            ChipAngleArrayI(1, ChipAngleCount) = Solution(2)
            ChipAngleArrayR(2, ChipAngleCount) = Solution(3)
            ChipAngleArrayI(2, ChipAngleCount) = Solution(4)
            ChipAngleArrayR(3, ChipAngleCount) = Solution(5)
            ChipAngleArrayI(3, ChipAngleCount) = Solution(6)

            v = DeltaPx / TDelta
            AverageV = AverageV + v
            Vcount = Vcount + 1
            Write #7, Interface, TValue(2), DValue(2), VValue(2), _
            ChipAngleArrayR(1, ChipAngleCount), ChipAngleArrayI(1,
ChipAngleCount); _
            ChipAngleArrayR(2, ChipAngleCount), ChipAngleArrayI(2,
ChipAngleCount); _
            ChipAngleArrayR(3, ChipAngleCount), ChipAngleArrayI(3,
ChipAngleCount)
            DValue(1) = DValue(2)
            VValue(1) = VValue(2)
            TValue(1) = TValue(2)

        ElseIf DValue(2) > 0 And DValue(2) <= 350 And DeltaPx <> 0 Then
            'laser point on the cantilever
            'when laser point goes on the cantilever,
            'use chip angle value to calculate the cantilever angle.
            If Mainform.frmChipAngle.Text = "" Then
                'find the best fit of chip angle in the array
                'Call BestFitModule.BestFit(ChipAngleCount)

                ChipAngle = 0
                'ChipAngle is still in radians, on panel we convert it into degrees for
consistance
                ChipAngleDegree = 180 * ChipAngle / PI
                Mainform.frmChipAngle.Text = ChipAngleDegree
                'Lnot must be defined before experiments
                LNot = Val(Mainform.frmLNot.Text)
                'calculate the XNotPrime and yNotPrime based on LNot and ChipAngle
                XNotPrime = LNot * Cos(LASERangle - 2 * ChipAngle)
                YNotPrime = LNot * Sin(LASERangle - 2 * ChipAngle)

                'get the DValue and VValue when laser point hit the origin
                OriginVoltage = Val(Mainform.OriginVoltage.Text)
                OriginTime = Val(Mainform.OriginTime.Text)
```

```
            DValue(1) = DValue(2)
            VValue(1) = VValue(2)
            TValue(1) = TValue(2)

            'Mark when laser spot gets on the cantilever
            Write #7, "Interface", "Time", "DValue", "VValue", "Angle", "Normal"

            i = i - 1
            End If

            'incident laser is LaserY=-
tan(LaserAngle)*laserX+tan(LaserAngle)*Velocity*TDelta
            Velocity = Val(Mainform.frmVelocity.Text)
            TDelta = TValue(2) - OriginTime
            LaserX = DValue(2)
            LaserY = -Tan(LASERangle) * LaserX + Tan(LASERangle) * Velocity *
TDelta

            'PSD equation is PSDY = -Tan(PSDAngle) * (PSDX - XNotPrime) +
YNotPrime
            DeltaU = VValue(2) - OriginVoltage
            PSDX = -XNotPrime + Abs(DeltaU) * Cos(PSDangle)  'PSDX>0
            PSDY = -Tan(PSDangle) * (PSDX - XNotPrime) + YNotPrime

            Delta = -Atn((LaserY - PSDY) / (LaserX - PSDX))
            CantileverNormal = (PI - LASERangle - Delta) / 2
            CantileverAngle = (LASERangle + Delta) / 2
            Write #7, Interface, TValue(2), DValue(2), CantileverAngle,
CantileverNormal
            DValue(1) = DValue(2)
            VValue(1) = VValue(2)
            TValue(1) = TValue(2)
            Else ' DeltaPx=0 means DValue(1)= DValue(2)
            i = i - 1
          End If
            ChipAngleCount = ChipAngleCount + 1
        End If
      End If
    End If

    i = i + 1

Loop
```

```
Close #6
Close #7



Exit Sub

ErrorHandler:
   If taskIsRunning = True Then
      DAQmxStopTask taskHandle
      DAQmxClearTask taskHandle
      taskIsRunning = False
   End If
   acquiringLabel.Caption = "Stand by..."
   startCommandButton.Enabled = True
   MsgBox "Error: " & Err.Number & " " & Err.Description, , "Error"
End Sub

Private Sub StopTask()
   'Done!
   Me.txtVoltPrecision.Enabled = True
   DAQmxErrChk DAQmxStopTask(taskHandle)
   DAQmxErrChk DAQmxClearTask(taskHandle)
   taskIsRunning = False
End Sub

Private Function ValidateControlValues()
   'This is an error check. if any of the boxes are empty then a message is sent to the user

   ValidateControlValues = 0

   If maxValueTextBox.Text = "" Or minValueTextBox.Text = "" Or
samplesPerChannelTextBox.Text = "" Or frequencyTextBox.Text = "" Then
      MsgBox "Please fill in all empty fields.", , Error
      ValidateControlValues = 1
   End If
End Function

Private Sub Form_Load()
   taskIsRunning = False
   acquiringLabel.Caption = "Stand by..."
   StopFlag = False

   PointSelect = 1
```

```
    'open the commport for pressure
    With MSComm1
       .CommPort = 1
       .Settings = "9600,N,8,1"
       .PortOpen = True
    End With
    PrevPressure = "0.00"

    PointSelect = 1

End Sub



Private Sub startCommandButton_Click()

    newsystime = 1E+26


    CWIMAQ1.Interface = "img0"
    CWIMAQ1.LoadInterfaceDefaults

    'TimeTimer.Enabled = True

    If Mainform.AutoScaleCheck.Value = 1 Then
    Mainform.AutoScaleTimer.Interval = Val(Mainform.AutoScaleText.Text) * 1000
    Mainform.AutoScaleTimer.Enabled = True
    End If
    SystemTime.Text = Format(Now, "hh:nn:ss") & "." & Right(Format(Timer, "#0.00"),
2)

    Open "C:\Documents and Settings\Josh\Desktop\timeimage.dat" For Output As #1
    Open "C:\Documents and Settings\Josh\Desktop\timereading.dat" For Output As #2

End Sub

Private Sub text1_keypress(keyascii As Integer)
Dim newtempstring As String
Dim length As Integer
Dim Hour, Minute, Second As String
Dim systime As Double

'If keyascii = 13 Then Text1.Text = " "

If keyascii >= 48 Or keyascii <= 57 Then
```

```
   length = Len(Text1.Text)
   If length = 5 Then

       SystemTime.Text = Format(Now, "hh:nn:ss") & "." & Right(Format(Timer,
"#0.00"), 2)
       Hour = Val(Left(SystemTime.Text, 2)) * 3600
       Minute = Left(SystemTime.Text, 5)
       Minute = Val(Right(Minute, 2)) * 60
       Second = Val(Right(SystemTime.Text, 5))
       systime = Hour + Minute + Second

       systime = systime - newsystime

       newtempstring = systime & "," & Text1.Text


Print #2, newtempstring
       Text1.Text = ""
   End If
End If

End Sub

' show the mouse_move in picturebox
Public Sub Display_MouseMove(Button As Integer, Shift As Integer, m As Single, n As
Single)
CurrentX = m
CurrentY = n
Coordinates.Caption = "Coordinates: ( " & m & "," & n & ")"
End Sub

Private Sub Display_mouseup(Button As Integer, Shift As Integer, x As Single, y As
Single)
If PointSelect = 2 Then
  CurrentX = x
  CurrentY = y
  AreaX2 = x
  AreaY2 = y
  RightBottom.Caption = "RightBottom:(" & AreaX2 & "," & AreaY2 & ")"
    Display.Line (AreaX2, AreaY2)-(AreaX2, AreaY1), vbGreen
    Display.Line (AreaX2, AreaY2)-(AreaX1, AreaY2), vbGreen
    Display.Line (AreaX1, AreaY1)-(AreaX2, AreaY1), vbGreen
    Display.Line (AreaX1, AreaY1)-(AreaX1, AreaY2), vbGreen
End If
```

```
End Sub

'chose the scaning area
Private Sub Display_MouseDown(Button As Integer, Shift As Integer, x As Single, y As
Single)
Dim yRegion, Distance As Integer
Dim CantiWidth As Integer
Dim CantiLength As Integer
If Button = 1 Then
   If PointSelect = 1 Then
   CurrentX = x
   CurrentY = y
   AreaX1 = x
   AreaY1 = y
   PointSelect = 2
   LeftTop.Caption = "LeftTop:(" & AreaX1 & "," & AreaY1 & ")"
   ElseIf PointSelect = 2 Then
   CurrentX = x
   CurrentY = y
   Mainform.LaserPositionX.Text = CurrentX
   Mainform.LaserPositionY.Text = CurrentY
   Display.Circle (x, y), 1, RGB(0, 255, 0)
   PointSelect = PointSelect + 1
   Else 'manual laser position collection
   CurrentX = x
   CurrentY = y
   Mainform.MovingLaserPointX = x
   Mainform.MovingLaserPointY = y
   yRegion = Val(Mainform.LaserPositionY.Text)
   Distance = y - yRegion 'distance in pixels
   CantiWidth = Val(Mainform.CantiWidth.Text) 'in pixels
   'the real width of cantilever is 35 micros
   CantiLength = Distance * 35 / CantiWidth 'in micros
   Mainform.Distance.Text = CantiLength
   'when distance changes, do the average of voltage
   DoTheAverage = True
   Display.Circle (x, y), 5, RGB(255, 0, 0)
   End If
ElseIf Button = 2 Then 'Right_click means we choose laser position image by image
   If PointSelect = 3 Then 'define the end of the cantilever
   CurrentX = x
   CurrentY = y
   Display.Circle (x, y), 1, RGB(0, 255, 0)
   Mainform.CantiEnd.Text = y
   PointSelect = PointSelect + 1
   Else
```

```
   CurrentX = x
   CurrentY = y
   Mainform.MovingLaserPointX = x
   Mainform.MovingLaserPointY = y
   Display.Circle (x, y), 5, RGB(255, 0, 0)
   End If
End If

End Sub


Private Function DetermineChannels()
'This function determines the physical channels to be used when they are selected
'in the frame.
Dim i As Integer
Dim FirstChannel As String
Dim SecondChannel As String

   DetermineChannels = ""
   FirstChannel = "Dev1/ai" & Me.txtFirstVoltage.Text
   SecondChannel = ",Dev1/ai" & Me.txtSecondVoltage.Text

   DetermineChannels = FirstChannel & SecondChannel

End Function

Private Function DetermineTempChannel()

   DetermineTempChannel = ""
   DetermineTempChannel = "Dev1/ai" & Me.txtTemp.Text

End Function

Private Sub StartContinuousCapture_Click()
   ConstantCaptureTimer.Interval = Val(ConstantCaptureTime.Text)
   ConstantCaptureTimer.Enabled = True
End Sub

Private Sub StopConstantCapture_Click()
ConstantCaptureTimer.Enabled = False
End Sub

Private Sub ConstantCaptureTimer_Timer()
   Call GetNewImage

   If DrawThelineCheck.Value = 1 Then
```

```
    currentx1 = CInt(Val(LineX1Text.Text))
    currenty1 = CInt(Val(LineY1Text.Text))
    currentx2 = CInt(Val(LineX2Text.Text))
    currenty2 = CInt(Val(LineY2Text.Text))
    Display.Line (currentx2, currenty2)-(currentx1, currenty1), vbGreen
    End If


End Sub


Private Sub StopBotton_Click()
    StopFlag = True

    If Mainform.AutoScaleTimer.Enabled = True Then
    Mainform.AutoScaleTimer.Enabled = False
    Mainform.Interface.Text = "0"
    End If

    If ConstantCaptureTimer.Enabled = True Then
    ConstantCaptureTimer.Enabled = False
    End If

    Mainform.startCommandButton.Enabled = True

    Close #1
    Close #2
End Sub
Private Sub SaveData(AveTime, AveVoltage, numChannels)
Dim k As Integer
Dim TempString As String


TempString = CStr(CDbl(FormatNumber(AveTime, 4)))

If StartDataPoint = CInt(txtPointsPerFile.Text) + 1 Then
    StartDataPoint = 1
    OutputDataFile.Text = Replace(OutputDataFile.Text, "-" & CStr(StartDataFile) &
".dat", "-" & CStr(StartDataFile + 1) & ".dat")
    StartDataFile = StartDataFile + 1
End If

If chkPressure.Value = 1 Then
    For k = 2 To numChannels + 1
     TempString = TempString & "," & CStr(CDbl(FormatNumber(AveVoltage(k),
Int(Me.txtVoltPrecision.Text))))
```

```
   Next k
   TempString = TempString & "," & CStr(CDbl(Trim(lblPressure.Caption)))
Else
   If DoTheAverage = False Then

      For k = 2 To numChannels + 1
       TempString = TempString & "," & CStr(CDbl(FormatNumber(AveVoltage(k),
Int(Me.txtVoltPrecision.Text))))
      Next k

      'when the distance is still the same, add the voltage signal
      PSDaverage = PSDaverage + CDbl(AveVoltage(2))
      PSDcounts = PSDcounts + 1
      TempString = TempString & "," & CStr(Mainform.Distance.Text) & ",#" & ",#" &
",#"

   Else

      For k = 2 To numChannels + 1
       TempString = TempString & "," & CStr(CDbl(FormatNumber(AveVoltage(k),
Int(Me.txtVoltPrecision.Text))))
      Next k

      'when we get a new distance, do the math.
      If PSDcounts = 0 Then
      PSDaverage = CDbl(AveVoltage(2))
      TempString = TempString & "," & CStr(Mainform.Distance.Text) & "," &
CStr(Mainform.Distance.Text) & "," & CStr(PSDaverage) & "," &
CStr(Mainform.Interface.Text)
      Else
      PSDaverage = PSDaverage / PSDcounts
      TempString = TempString & "," & CStr(Mainform.Distance.Text) & "," &
CStr(Mainform.Distance.Text) & "," & CStr(PSDaverage) & "," &
CStr(Mainform.Interface.Text)
      PSDcounts = 0
      PSDaverage = 0
      End If

   DoTheAverage = False
   End If

End If

TempString = TempString & "," & Mainform.Text1.Text
Print #2, TempString
```

```
StartDataPoint = StartDataPoint + 1

'if laser is on Origin point, or distance = 0
'then load the system time and PSD1 voltage
'If Val(Mainform.Distance.Text) = 0 Then
'Mainform.OriginVoltage.Text = PSDaverage
'Mainform.OriginTime.Text = AveTime
'End If

End Sub

Private Sub SaveData(AveTime, AveVoltage, numChannels)
Dim k As Integer
Dim TempString As String

TempString = CStr(CDbl(FormatNumber(AveTime, 4)))

If StartDataPoint = CInt(txtPointsPerFile.Text) + 1 Then
   StartDataPoint = 1
   OutputDataFile.Text = Replace(OutputDataFile.Text, "-" & CStr(StartDataFile) &
".dat", "-" & CStr(StartDataFile + 1) & ".dat")
   StartDataFile = StartDataFile + 1
End If

If chkPressure.Value = 1 Then
   For k = 2 To numChannels + 1
    TempString = TempString & "," & CStr(CDbl(FormatNumber(AveVoltage(k),
Int(Me.txtVoltPrecision.Text))))
   Next k
   TempString = TempString & "," & CStr(CDbl(Trim(lblPressure.Caption)))
Else
   For k = 2 To numChannels + 1
    TempString = TempString & "," & CStr(CDbl(FormatNumber(AveVoltage(k),
Int(Me.txtVoltPrecision.Text))))
   Next k
   TempString = TempString & "," & CStr(Mainform.Distance.Text) & "," &
CStr(Mainform.Interface.Text)
End If
Print #2, TempString
StartDataPoint = StartDataPoint + 1

End Sub
```

```
Private Sub PlotData(AveTime, AveVol, pAVGt, pAVGvol, numChannels, XOrigin,
YOrigin, ScaleX, ScaleY, XOriginT, YOriginT, ScaleXT, ScaleYT, XOrigin2, YOrigin2,
ScaleX2, ScaleY2)
Dim PinXPos, PinYpos As Double
Dim inXPos, inYPos As Double


DoEvents


'Here we are drawing the PSD voltage signal
   PinXPos = XOrigin + (pAVGt * ScaleX)
   inXPos = XOrigin + (AveTime * ScaleX)
   PinYpos = YOrigin - (pAVGvol(2) * ScaleY)
   inYPos = YOrigin - (AveVol(2) * ScaleY)
   Mainform.PicChart.ForeColor = vbGreen
   Mainform.PicChart.Line (PinXPos, PinYpos)-(inXPos, inYPos)

'Here we are drawing the PSD2 voltage signal
   PinXPos = XOrigin2 + (pAVGt * ScaleX2)
   inXPos = XOrigin2 + (AveTime * ScaleX2)
   PinYpos = YOrigin2 - (pAVGvol(3) * ScaleY2)
   inYPos = YOrigin2 - (AveVol(3) * ScaleY2)
   Mainform.PicChart2.ForeColor = vbGreen
   Mainform.PicChart2.Line (PinXPos, PinYpos)-(inXPos, inYPos)

'Here we are drawing the temperature signal
  PinXPos = XOriginT + (pAVGt * ScaleXT)
  inXPos = XOriginT + (AveTime * ScaleXT)
  PinYpos = YOriginT - (pAVGvol(4) * ScaleYT)
  inYPos = YOriginT - (AveVol(4) * ScaleYT)
  Mainform.TempChart.ForeColor = vbBlue
  Mainform.TempChart.Line (PinXPos, PinYpos)-(inXPos, inYPos)


End Sub
Private Function ColorCode(k) As String

   If k = 1 Then ColorCode = "vbRed"
   If k = 2 Then ColorCode = "vbGreen"
   If k = 3 Then ColorCode = "vbBlue"
   If k = 4 Then ColorCode = "vbMagenta"

End Function
Private Sub RePlotData(XOrigin, YOrigin, ScaleX, ScaleY, XOriginT, YOriginT,
ScaleXT, ScaleYT, XOrigin2, YOrigin2, ScaleX2, ScaleY2)
```

```
Dim k, j As Long
Dim PinXPos, PinYpos As Double
Dim inXPos, inYPos As Double
Dim iVoltage1, Voltage1, iVoltage2, Voltage2, iTime, Time, iTemp, Temp As Double

DoEvents

j = 1
If SaveDataCheck.Value = 1 Then
   Do While (EOF(1) = False)
      If j = 1 Then
         Input #2, iTime, iVoltage1, iVoltage2, iTemp

         j = j + 1

      Else

         Input #2, Time, Voltage1, Voltage2, Temp


         'Here we are drawing the PSD voltage signal
            PinXPos = XOrigin + (iTime * ScaleX)
            inXPos = XOrigin + (Time * ScaleX)
            PinYpos = YOrigin - (iVoltage1 * ScaleY)
            inYPos = YOrigin - (Voltage1 * ScaleY)
            Mainform.PicChart.ForeColor = vbBlack
            Mainform.PicChart.Line (PinXPos, PinYpos)-(inXPos, inYPos)

         'Here we are drawing the PSD2 voltage signal
            PinXPos = XOrigin2 + (iTime * ScaleX2)
            inXPos = XOrigin2 + (Time * ScaleX2)
            PinYpos = YOrigin2 - (iVoltage2 * ScaleY2)
            inYPos = YOrigin2 - (Voltage2 * ScaleY2)
            Mainform.PicChart2.ForeColor = vbBlack
            Mainform.PicChart2.Line (PinXPos, PinYpos)-(inXPos, inYPos)

         'Here we are drawing the temperature signal
            PinXPos = XOriginT + (iTime * ScaleXT)
            inXPos = XOriginT + (Time * ScaleXT)
            PinYpos = YOriginT - (iTemp * ScaleYT)
            inYPos = YOriginT - (Temp * ScaleYT)
            Mainform.TempChart.ForeColor = vbBlack
            Mainform.TempChart.Line (PinXPos, PinYpos)-(inXPos, inYPos)


            iTime = Time
```

```
            iVoltage1 = Voltage1
            iVoltage2 = Voltage2
            iTemp = Temp

        End If

    Loop
    Close #2
End If

End Sub
Private Function DetermineThermType() As DAQmxThermocoupleType1
        If Me.txtThermType.Text = "K" Then DetermineThermType =
DAQmx_Val_ThermocoupleType1_K_Type_TC
        If Me.txtThermType.Text = "B" Then DetermineThermType =
DAQmx_Val_ThermocoupleType1_B_Type_TC
        If Me.txtThermType.Text = "E" Then DetermineThermType =
DAQmx_Val_ThermocoupleType1_E_Type_TC
        If Me.txtThermType.Text = "J" Then DetermineThermType =
DAQmx_Val_ThermocoupleType1_J_Type_TC
        If Me.txtThermType.Text = "N" Then DetermineThermType =
DAQmx_Val_ThermocoupleType1_N_Type_TC
        If Me.txtThermType.Text = "R" Then DetermineThermType =
DAQmx_Val_ThermocoupleType1_R_Type_TC
        If Me.txtThermType.Text = "S" Then DetermineThermType =
DAQmx_Val_ThermocoupleType1_S_Type_TC
        If Me.txtThermType.Text = "T" Then DetermineThermType =
DAQmx_Val_ThermocoupleType1_T_Type_TC
End Function

Private Sub TimeTimer_Timer()
SystemTimeCount = SystemTimeCount + 1
Mainform.SystemTime.Text = SystemTimeCount / 10
End Sub


Public Sub InitiateGraph(XMin, XMax, XOrigin, YOrigin, ScaleX, ScaleY, XOriginT,
YOriginT, ScaleXT, ScaleYT, XOrigin2, YOrigin2, ScaleX2, ScaleY2)

Dim inCounter
Dim inMaxX As Integer
Dim inMaxY As Integer
Dim inLmarg As Integer
Dim inRmarg As Integer
Dim inBmarg As Integer
Dim inTmarg As Integer
```

```
Dim inXPos As Integer
Dim inYPos As Integer
Dim YLabel, XLabel As Double
Dim XTicks, YTicks As Double

DoEvents

YMin = Val(Mainform.minValueTextBox.Text)
YMax = Val(Mainform.maxValueTextBox.Text)

Mainform.PicChart.ForeColor = vbBlack
Mainform.PicChart.AutoRedraw = True
Mainform.PicChart.ScaleMode = 3
Mainform.PicChart.Cls

'Determine the maximum size of chart
inMaxX = Mainform.PicChart.ScaleWidth
inMaxY = Mainform.PicChart.ScaleHeight

'Determine the chart margins, including
'width for the axis labels
inLmarg = Mainform.PicChart.TextWidth("10000")
inBmarg = 1.35 * Mainform.PicChart.TextHeight("5000")
inRmarg = inMaxX - 0.5 * inLmarg
inTmarg = 0.25 * inLmarg
inBmarg = inMaxY - inBmarg

'Determine scale factors for each axis
ScaleX = (inRmarg - inLmarg) / (XMax - XMin)
ScaleY = (inBmarg - inTmarg) / (YMax - YMin)

'Determine the origin of the graph
If XMin <= 0 Then
   XOrigin = inLmarg + Abs(XMin) * ScaleX
Else
   XOrigin = inLmarg - XMin * ScaleX
End If
YOrigin = inBmarg + YMin * ScaleY

'Draw a blue lines to show the origin
Mainform.PicChart.ForeColor = vbBlue
Mainform.PicChart.Line (inLmarg, YOrigin)-(inRmarg, YOrigin)     'This draws the real
graphical abscissa
Mainform.PicChart.Line (XOrigin, inTmarg)-(XOrigin, inBmarg)     'This draws the real
graphical ordinate
```

```
Mainform.PicChart.ForeColor = vbBlack
'Draw Axes
Mainform.PicChart.Line (inLmarg, inTmarg)-(inLmarg, inBmarg)    'This draws the left
ordinate
Mainform.PicChart.Line -(inRmarg, inBmarg)                      'This draws the bottom
abscissa
Mainform.PicChart.Line (inLmarg, inTmarg)-(inRmarg, inTmarg)    'This draws the top
abscissa
Mainform.PicChart.Line (inRmarg, inTmarg)-(inRmarg, inBmarg)    'This draws the
right ordinate

'Draw labels and tic marks for vertical axis
YTicks = ((YMax - YMin) / 5)
YLabel = Format(YMin, "#0.0")
For inCounter = 1 To 6
   Mainform.PicChart.CurrentX = 5
   inYPos = inBmarg - ((inCounter - 1) * YTicks * ScaleY)
   Mainform.PicChart.CurrentY = inYPos
   Mainform.PicChart.Print Str(FormatNumber(YLabel, 2, vbUseDefault, vbUseDefault,
vbFalse))
   YLabel = YLabel + YTicks
   Mainform.PicChart.Line (inLmarg, inYPos)-(inLmarg + 5, inYPos)   '5 is the length of
the tick mark in pixels
Next inCounter

'Draw labels and tic marks for horizontal axis
XTicks = ((XMax - XMin) / 5)
XLabel = Format(XMin, "#0.0")
For inCounter = 1 To 6
   inXPos = inLmarg + ((inCounter - 1) * XTicks * ScaleX)
   Mainform.PicChart.CurrentX = inXPos - Mainform.PicChart.TextWidth("00") / 2
   Mainform.PicChart.CurrentY = inBmarg + 5
   Mainform.PicChart.Print Str(FormatNumber(XLabel, 1, vbUseDefault, vbUseDefault,
vbFalse))
   XLabel = XLabel + XTicks
   Mainform.PicChart.Line (inXPos, inBmarg)-(inXPos, inBmarg - 5)
Next inCounter

'************************************************************************
**********************************
'Now we initialize the temperature chart
'New variables XOriginT, YOriginT, ScaleXT, ScaleYT

YMin = Val(Mainform.MinTempYRange.Text)
YMax = Val(Mainform.MaxTempYRange.Text)
```

```
Mainform.TempChart.ForeColor = vbBlack
Mainform.TempChart.AutoRedraw = True
Mainform.TempChart.ScaleMode = 3
Mainform.TempChart.Cls

'Determine the maximum size of chart
inMaxX = Mainform.TempChart.ScaleWidth
inMaxY = Mainform.TempChart.ScaleHeight

'Determine the chart margins, including
'width for the axis labels
inLmarg = Mainform.TempChart.TextWidth("10000")
inBmarg = 1.35 * Mainform.TempChart.TextHeight("5000")
inRmarg = inMaxX - 0.5 * inLmarg
inTmarg = 0.25 * inLmarg
inBmarg = inMaxY - inBmarg

'Determine scale factors for each axis
ScaleXT = (inRmarg - inLmarg) / (XMax - XMin)
ScaleYT = (inBmarg - inTmarg) / (YMax - YMin)

'Determine the origin of the graph
If XMin <= 0 Then
   XOriginT = inLmarg + Abs(XMin) * ScaleXT
Else
   XOriginT = inLmarg - XMin * ScaleXT
End If
YOriginT = inBmarg + YMin * ScaleYT

'Draw a blue lines to show the origin
Mainform.TempChart.ForeColor = vbBlue
Mainform.TempChart.Line (inLmarg, YOriginT)-(inRmarg, YOriginT)    'This draws
the real graphical abscissa
Mainform.TempChart.Line (XOriginT, inTmarg)-(XOriginT, inBmarg)    'This draws
the real graphical ordinate

Mainform.TempChart.ForeColor = vbBlack
'Draw Axes
Mainform.TempChart.Line (inLmarg, inTmarg)-(inLmarg, inBmarg)    'This draws the
left ordinate
Mainform.TempChart.Line -(inRmarg, inBmarg)                      'This draws the bottom
abscissa
Mainform.TempChart.Line (inLmarg, inTmarg)-(inRmarg, inTmarg)    'This draws the
top abscissa
Mainform.TempChart.Line (inRmarg, inTmarg)-(inRmarg, inBmarg)    'This draws the
right ordinate
```

```
'Draw labels and tic marks for vertical axis
YTicks = ((YMax - YMin) / 5)
YLabel = Format(YMin, "#0.0")
For inCounter = 1 To 6
   Mainform.TempChart.CurrentX = 5
   inYPos = inBmarg - ((inCounter - 1) * YTicks * ScaleYT)
   Mainform.TempChart.CurrentY = inYPos
   Mainform.TempChart.Print Str(FormatNumber(YLabel, 1, vbUseDefault,
vbUseDefault, vbFalse))
   YLabel = YLabel + YTicks
   Mainform.TempChart.Line (inLmarg, inYPos)-(inLmarg + 5, inYPos)   '5 is the length
of the tick mark in pixels
Next inCounter

'Draw labels and tic marks for horizontal axis
XTicks = ((XMax - XMin) / 5)
XLabel = Format(XMin, "#0.0")
For inCounter = 1 To 6
   inXPos = inLmarg + ((inCounter - 1) * XTicks * ScaleXT)
   Mainform.TempChart.CurrentX = inXPos - Mainform.TempChart.TextWidth("00") / 2
   Mainform.TempChart.CurrentY = inBmarg + 5
   Mainform.TempChart.Print Str(FormatNumber(XLabel, 1, vbUseDefault,
vbUseDefault, vbFalse))
   XLabel = XLabel + XTicks
   Mainform.TempChart.Line (inXPos, inBmarg)-(inXPos, inBmarg - 5)
Next inCounter


'**************************************************************************
**********************************
'Now we initialize the second PSD chart.
'New variables XOrigin2, YOrigin2, ScaleX2, ScaleY2


YMin = Val(Mainform.txtPSD2min.Text)
YMax = Val(Mainform.txtPSD2max.Text)

Mainform.PicChart2.ForeColor = vbBlack
Mainform.PicChart2.AutoRedraw = True
Mainform.PicChart2.ScaleMode = 3
Mainform.PicChart2.Cls

'Determine the maximum size of chart
inMaxX = Mainform.PicChart2.ScaleWidth
inMaxY = Mainform.PicChart2.ScaleHeight
```

```
'Determine the chart margins, including
'width for the axis labels
inLmarg = Mainform.PicChart2.TextWidth("10000")
inBmarg = 1.35 * Mainform.PicChart2.TextHeight("5000")
inRmarg = inMaxX - 0.5 * inLmarg
inTmarg = 0.25 * inLmarg
inBmarg = inMaxY - inBmarg

'Determine scale factors for each axis
ScaleX2 = (inRmarg - inLmarg) / (XMax - XMin)
ScaleY2 = (inBmarg - inTmarg) / (YMax - YMin)

'Determine the origin of the graph
If XMin <= 0 Then
   XOrigin2 = inLmarg + Abs(XMin) * ScaleX2
Else
   XOrigin2 = inLmarg - XMin * ScaleX2
End If
YOrigin2 = inBmarg + YMin * ScaleY2

'Draw a blue lines to show the origin
Mainform.PicChart2.ForeColor = vbBlue
Mainform.PicChart2.Line (inLmarg, YOrigin2)-(inRmarg, YOrigin2)    'This draws the
real graphical abscissa
Mainform.PicChart2.Line (XOrigin2, inTmarg)-(XOrigin2, inBmarg)    'This draws the
real graphical ordinate

Mainform.PicChart2.ForeColor = vbBlack
'Draw Axes
Mainform.PicChart2.Line (inLmarg, inTmarg)-(inLmarg, inBmarg)    'This draws the
left ordinate
Mainform.PicChart2.Line -(inRmarg, inBmarg)                      'This draws the bottom
abscissa
Mainform.PicChart2.Line (inLmarg, inTmarg)-(inRmarg, inTmarg)    'This draws the top
abscissa
Mainform.PicChart2.Line (inRmarg, inTmarg)-(inRmarg, inBmarg)    'This draws the
right ordinate

'Draw labels and tic marks for vertical axis
YTicks = ((YMax - YMin) / 5)
YLabel = Format(YMin, "#0.0")
For inCounter = 1 To 6
   Mainform.PicChart2.CurrentX = 5
   inYPos = inBmarg - ((inCounter - 1) * YTicks * ScaleY2)
   Mainform.PicChart2.CurrentY = inYPos
```

```
   Mainform.PicChart2.Print Str(FormatNumber(YLabel, 2, vbUseDefault, vbUseDefault,
vbFalse))
   YLabel = YLabel + YTicks
   Mainform.PicChart2.Line (inLmarg, inYPos)-(inLmarg + 5, inYPos)  '5 is the length
of the tick mark in pixels
Next inCounter

'Draw labels and tic marks for horizontal axis
XTicks = ((XMax - XMin) / 5)
XLabel = Format(XMin, "#0.0")
For inCounter = 1 To 6
   inXPos = inLmarg + ((inCounter - 1) * XTicks * ScaleX2)
   Mainform.PicChart2.CurrentX = inXPos - Mainform.PicChart2.TextWidth("00") / 2
   Mainform.PicChart2.CurrentY = inBmarg + 5
   Mainform.PicChart2.Print Str(FormatNumber(XLabel, 1, vbUseDefault, vbUseDefault,
vbFalse))
   XLabel = XLabel + XTicks
   Mainform.PicChart2.Line (inXPos, inBmarg)-(inXPos, inBmarg - 5)
Next inCounter


End Sub

Public Sub PlotRedLineGraph(Data, TnP, XOrigin, YOrigin, ScaleX, ScaleY)

Mainform.PicChart.ForeColor = vbRed
For i = 1 To TnP
   inXPos = XOrigin + (Data(1, i) * ScaleX)
   inYPos = YOrigin - (Data(2, i) * ScaleY)
   If i = 1 Then
      Mainform.PicChart.CurrentX = inXPos
      Mainform.PicChart.CurrentY = inYPos
   Else
      Mainform.PicChart.Line -(inXPos, inYPos)
   End If
Next i
End Sub

Public Sub PlotGreenLineGraph(Data, TnP, XOrigin, YOrigin, ScaleX, ScaleY)

Mainform.PicChart.ForeColor = vbGreen
For i = 1 To TnP
   inXPos = XOrigin + (Data(1, i) * ScaleX)
   inYPos = YOrigin - (Data(2, i) * ScaleY)
   If i = 1 Then
      Mainform.PicChart.CurrentX = inXPos
```

```
        Mainform.PicChart.CurrentY = inYPos
    Else
        Mainform.PicChart.Line -(inXPos, inYPos)
    End If
Next i
End Sub



Public Sub DAQmxErrChk(errorCode As Long)

'  Utility function to handle errors by recording the DAQmx error code
'  and message.

    Dim errorString As String
    Dim bufferSize As Long
    Dim status As Long

    If (errorCode < 0) Then
        ' Find out the error message length.
        bufferSize = DAQmxGetErrorString(errorCode, 0, 0)
        ' Allocate enough space in the string.
        errorString = String$(bufferSize, 0)
        ' Get the actual error message.
        status = DAQmxGetErrorString(errorCode, errorString, bufferSize)
        ' Trim it to the actual length, and display the message
        errorString = Left(errorString, InStr(errorString, Chr$(0)))
        Err.Raise errorCode, , errorString
    End If

 End Sub



Public Sub BestFit(Run As Integer) '
Dim i, j, k As Integer
Dim ConsistentXvalueR(), ConsistentXvalueI(), ConsistentXvalueModulus,
XvalueModulus As Double
Dim Rdifference(3), Idifference(3) As Double

MsgBox "Got The BEST FIT part!!!"

For j = 1 To 3 'SAME COMPARATIONS FOR X1, X2, AND X3
```

```
  For i = 1 To Run - 1 'SAME COMPARATION FOR ALL THE OTHER RUNS
  'compare Xvalues of first run with  Xvalues of the next run
    For k = 1 To 3
    Rdifference(k) = ChipAngleArrayR(j, i) - ChipAngleArrayR(k, i + 1)
    Idifference(k) = ChipAngleArrayI(j, i) - ChipAngleArrayI(k, i + 1)
    Next

  'first find out the closest x value in next run by comparing the differences
  'looking for the smallest Rdifference^2+Idifference^2


  'then take the average of two closest x values as consistant value
  'Hope the two closest Xvalues are the exact same Xvalues


  End If
  Next
Next


End Sub


'If we have a cubic equation in form of A3*x^3+A2*x^2+A1*x+A0=0 (A3<>0)
'then we get following relations according to Vieta's Theorem:
'x1+x2+x3=-A2/A3
'x1*x2+x2*x3+x3*x1=A1/A3
'x1*x2*x3=-A0/A3

'$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$
'
'Solution for A3*x^3+A2*x^2+A1*x+A0=0  (A3<>0)
'
'1 Change it in to special form y^3+p*y+q=0
'
'2 Solve y^3+p*y+q=0 by y=AC^(1/3)+BC^(1/3)
'
'3 Take A and B as roots of a*z^2+b*z+c=0
'
'4 Use y=A^(1/3)+B^(1/3) and x=y-A2/(3*A3) to solve x1
'
'5 Vieta's Theorem in three degree to solve x2 and x3
'$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$$
$$$$$$$$$
```

```
Public Sub SolveCubicEquation(DeltaPx As Double, DeltaH As Double, PSDangle As
Double, LASERangle As Double, Solution)
Dim A3, A2, A1, A0, p, q, PI, YModulus, YSita As Double
Dim RealY1, RealY2, RealY3, ImageY1, ImageY2, ImageY3, RealX1, RealX2, RealX3,
ImageX1, ImageX2, ImageX3 As Double
Dim uDelta, u1r, u1i, u2r, u2i, uModulus, u1Sita, u2Sita, zModulus As Double


PI = 4 * Atn(1)

'change microns into mm
DeltaPx = DeltaPx / 1000

'For A3*x^3+A2*x^2+A1*x+A0=0  (A3<>0)
A3 = DeltaPx
A2 = -(DeltaH * Sin(PSDangle) - Tan(LASERangle) * DeltaPx - Tan(LASERangle) *
DeltaH * Cos(PSDangle))
A1 = 2 * Tan(LASERangle) * DeltaH * Sin(PSDangle) + DeltaPx - 2 * DeltaH *
Cos(PSDangle)
A0 = DeltaH * Sin(PSDangle) - Tan(LASERangle) * DeltaPx + Tan(LASERangle) *
DeltaH * Cos(PSDangle)


'Debug.Print "**********************"
If A3 <> Empty And A3 <> 0 Then

'Substitute x=y-A2/(3*A3) to get y^3+p*y+q=0
p = A1 / A3 - A2 ^ 2 / (3 * A3 ^ 2)
q = 2 * A2 ^ 3 / (27 * A3 ^ 3) - A1 * A2 / (3 * A3 ^ 2) + A0 / A3
'Debug.Print "p = " & p
'Debug.Print "q = " & q
If p = 0 And q = 0 Then
   'y^3=0, 3 real roots of same value.
'   Debug.Print "3 same real roots!"
   RealX1 = -A2 / (3 * A3)
   RealX2 = RealX1
   RealX3 = RealX1
   ImageX1 = 0
   ImageX2 = ImageX1
   ImageX3 = ImageX1

ElseIf p = 0 Then
'y^3+q=0, only one real root and 2 image roots
```

```
'   Debug.Print "One real root and two image roots!"
    YModulus = CubicRoot(-q)
    RealX1 = YModulus - A2 / (3 * A3)
    ImageX1 = 0
'   Debug.Print "One real root is x1=" & RealX1

    If YModulus > 0 Then
    YSita = 2 * PI
    Else
    YSita = PI
    End If

    RealX2 = YModulus * Cos(YSita / 3) - A2 / (3 * A3)
    ImageX2 = YModulus * Sin(YSita / 3)
    RealX3 = RealX2
    ImageX3 = -ImageX2
    Debug.Print "Two imaginary roots are: "
    Debug.Print "x2 = " & RealX2 & " + " & ImageX2 & "i"
    Debug.Print "x3 = " & RealX3 & " + " & ImageX3 & "i"

Else

    'To solve y^3+p*y+q=0, use Vieta's substitition y=z-p/z
    'y^3+p*y+q=0 changes to z^6+q*z^3-p^3/27=0
    'with u = z^3, we have u^2+q*u-p^3/27=0
    uDelta = q ^ 2 + 4 * p ^ 3 / 27
    If uDelta >= 0 Then
    u1r = -q / 2 + Sqr(uDelta) / 2
    u2r = -q / 2 - Sqr(uDelta) / 2
    u1i = 0
    u2i = 0
    uModulus = Sqr(u1r ^ 2 + u1i ^ 2)
    Else
       If q = 0 Then
       u1r = 0
       u2r = 0
       Else
       u1r = -q / 2
       u2r = -q / 2
       End If
    u1i = Sqr(-uDelta) / 2
    u2i = Sqr(-uDelta) / 2
    uModulus = Sqr(u1r ^ 2 + u1i ^ 2)
    End If
```

```
'   Debug.Print "u1 =" & u1r & " + " & u1i & "i"
'   Debug.Print "u1 =" & u1r & " + " & u1i & "i"

    'each u gives three z, use polar form to solve them
    If u1r = 0 Then
       If u1i > 0 Then
       u1Sita = PI / 2
       ElseIf u1i < 0 Then
       u1Sita = PI * 3 / 2
       End If
    ElseIf u1r < 0 And u1i > 0 Then '2nd
    u1Sita = PI + Atn(u1i / u1r)
    ElseIf u1r < 0 And u1i < 0 Then '3rd
    u1Sita = Atn(u1i / u1r) + PI
    ElseIf u1r > 0 And u1i < 0 Then '4th
    u1Sita = Atn(u1i / u1r) + PI * 2
    Else ' u1Sita in the 1st phase
    u1Sita = Atn(u1i / u1r)
    End If

    If u2r = 0 Then
       If u2i > 0 Then
       u2Sita = PI / 2
       ElseIf u2i < 0 Then
       u2Sita = PI * 3 / 2
       End If
    ElseIf u2r < 0 And u2i > 0 Then '2nd
    u2Sita = PI + Atn(u2i / u2r)
    ElseIf u2r < 0 And u2i < 0 Then '3rd
    u2Sita = Atn(u2i / u2r) + PI
    ElseIf u2r > 0 And u2i < 0 Then '4th
    u2Sita = Atn(u2i / u2r) + PI * 2
    Else ' u2Sita in the 1st phase
    u2Sita = Atn(u2i / u2r)
    End If

    'we can write u1 and u2 in polar forms
    'u1=uModulus*(cos(u1Sita)+isin(u1sita))
    'u2=uModulus*(cos(u2Sita)+isin(u2sita))
    zModulus = CubicRoot(uModulus)
'   Debug.Print "z1 = " & zModulus * Cos(u1Sita / 3) & " + " & zModulus * Sin(u1Sita /
3) & "i"
'   Debug.Print "z2 = " & zModulus * Cos(u1Sita / 3 + 2 * pi / 3) & " + " & zModulus *
Sin(u1Sita / 3 + 2 * pi / 3) & "i"
'   Debug.Print "z3 = " & zModulus * Cos(u1Sita / 3 - 2 * pi / 3) & " + " & zModulus *
Sin(u1Sita / 3 - 2 * pi / 3) & "i"
```

```
'   Debug.Print "z4 = " & zModulus * Cos(u2Sita / 3) & " + " & zModulus * Sin(u2Sita /
3) & "i"
'   Debug.Print "z5 = " & zModulus * Cos(u2Sita / 3 + 2 * pi / 3) & " + " & zModulus *
Sin(u2Sita / 3 + 2 * pi / 3) & "i"
'   Debug.Print "z6 = " & zModulus * Cos(u2Sita / 3 - 2 * pi / 3) & " + " & zModulus *
Sin(u2Sita / 3 - 2 * pi / 3) & "i"

    'with y=z-p/3/z, we got six y values
    'y=(zModulus-p/3/zmodulus)*cos()+ (zModulus+p/3/zmodulus)*sin()i
'   Debug.Print "y1 = " & (zModulus - p / 3 / zModulus) * Cos(u1Sita / 3) & " + " &
(zModulus + p / 3 / zModulus) * Sin(u1Sita / 3) & "i"
'   Debug.Print "y2 = " & (zModulus - p / 3 / zModulus) * Cos(u1Sita / 3 + 2 * pi / 3) & "
+ " & (zModulus + p / 3 / zModulus) * Sin(u1Sita / 3 + 2 * pi / 3) & "i"
'   Debug.Print "y3 = " & (zModulus - p / 3 / zModulus) * Cos(u1Sita / 3 - 2 * pi / 3) & "
+ " & (zModulus + p / 3 / zModulus) * Sin(u1Sita / 3 - 2 * pi / 3) & "i"
'   Debug.Print "y4 = " & (zModulus - p / 3 / zModulus) * Cos(u2Sita / 3) & " + " &
(zModulus + p / 3 / zModulus) * Sin(u2Sita / 3) & "i"
'   Debug.Print "y5 = " & (zModulus - p / 3 / zModulus) * Cos(u2Sita / 3 + 2 * pi / 3) & "
+ " & (zModulus + p / 3 / zModulus) * Sin(u2Sita / 3 + 2 * pi / 3) & "i"
'   Debug.Print "y6 = " & (zModulus - p / 3 / zModulus) * Cos(u2Sita / 3 - 2 * pi / 3) & "
+ " & (zModulus + p / 3 / zModulus) * Sin(u2Sita / 3 - 2 * pi / 3) & "i"

    'use the three real value of y and the relation x=y-A2/(3*A3) to find x
    'Dim RealY1, RealY2, RealY3, ImageY1, ImageY2, ImageY3, Xvalue1, Xvalue2,
Xvalue3, Xvalue1I, Xvalue2I, Xvalue3I As Double
    RealY1 = (zModulus - p / 3 / zModulus) * Cos(u1Sita / 3)
    ImageY1 = (zModulus + p / 3 / zModulus) * Sin(u1Sita / 3)
    RealY2 = (zModulus - p / 3 / zModulus) * Cos(u1Sita / 3 + 2 * PI / 3)
    ImageY2 = (zModulus + p / 3 / zModulus) * Sin(u1Sita / 3 + 2 * PI / 3)
    RealY3 = (zModulus - p / 3 / zModulus) * Cos(u1Sita / 3 - 2 * PI / 3)
    ImageY3 = (zModulus + p / 3 / zModulus) * Sin(u1Sita / 3 - 2 * PI / 3)

    RealX1 = RealY1 - A2 / (3 * A3)
    ImageX1 = ImageY1
    RealX2 = RealY2 - A2 / (3 * A3)
    ImageX2 = ImageY2
    RealX3 = RealY3 - A2 / (3 * A3)
    ImageX3 = ImageY3

'   Debug.Print "x1 =" & RealX1 & " + " & ImageX1 & "i"
'   Debug.Print "x2 =" & RealX2 & " + " & ImageX2 & "i"
'   Debug.Print "x3 =" & RealX3 & " + " & ImageX3 & "i"

    End If
End If
```

```
Solution(1) = RealX1
Solution(2) = ImageX1
Solution(3) = RealX2
Solution(4) = ImageX2
Solution(5) = RealX3
Solution(6) = ImageX3
'Debug.Print "~~~THE END~~~"
End Sub
Public Function CubicRoot(Value)
'Dim value As Double
If Value < 0 Then
Value = -Value
CubicRoot = Value ^ (1 / 3)
CubicRoot = -CubicRoot
Else
CubicRoot = Value ^ (1 / 3)
End If
End Function
```

Appendix B:
Visual Basic Program Code for Cubic Equation Solution

```
Private Sub Command1_Click()
Dim a, b, c, d As Double    'These are the coeficients of the cubic equations
Dim x(3, 2) As Double       'This is the solution to the cubic equation
Dim InputData(), Data() As Double
Dim i, j, k, iTnp, Tnp As Integer
Dim PSD, Time, iDistance As Double
Dim tmp As Variant

Open "C:\Documents and Settings\Josh\Desktop\Beta Data\AveragedData.csv" For Input
As #1

i = 0
Do While EOF(1) = False

     ReDim Preserve Data(3, i)
     Input #1, Data(1, i), Data(2, i)

  iTnp = i
  i = i + 1
Loop
Close #1


Open "C:\Documents and Settings\Josh\Desktop\Beta Data\outAveragedData.csv" For
Output As #1
For i = 1 To iTnp
  Write #1, Data(1, i), Data(2, i)
Next i
Close #1


Call FitStraightLine(iTnp, Data, m, b)

'Now we step through the value of deltaPx
Dim DPx, Dh, Limit As Double
Dim Phi, Theta, Pi, NewTheta As Double
Pi = 4 * Atn(1)
Phi = 30 * Pi / 180
Theta = 60 * Pi / 180
```

```
Limit = Abs(Data(1, 1))
i = 1

NewTheta = Theta

Open "C:\Documents and Settings\Josh\Desktop\Beta Data\output.csv" For Output As #1
Write #1, "Theta", "Distance", "PSD", "R(x1)", "I(x1)", "R(x2)", "I(x2)", "R(x3)", "I(x3)"

'For Theta = (60 + 0.5) * Pi / 180 To (60 + 1) * Pi / 180 Step 0.0001
  'For DPx = Limit / 100 To Limit Step Limit / 100
  For i = 1 To iTnp
    DPx = Data(1, i)
    Dh = Data(2, i)
    'Dh = m * DPx
    dhy = Dh * Sin(Phi)
    dhx = Abs(Dh * Cos(Phi))
    AlphA = dhx - DPx
    'Tan(Theta + 2 * beta)
    a = DPx
    b = AlphA * Tan(Theta) - dhy + 2 * DPx * Tan(Theta)
    c = -2 * dhy * Tan(Theta) - DPx - 2 * AlphA
    d = dhy - AlphA * Tan(Theta)

    Call Vieta(a, b, c, d, x)
    Write #1, Theta, DPx, Dh, Atn(x(1, 1)) * 180 / Pi, x(1, 2), Atn(x(2, 1)) * 180 / Pi,
x(2, 2), Atn(x(3, 1)) * 180 / Pi, x(3, 2)

    Next i
  'Next DPx
'Next Theta

Close #1

End Sub


Public Sub Vieta(AA, BB, CC, DD, x)

Dim b, c, d, e, f As Double
Dim AAA, BBB, CCC As Double
Dim u1(2), u2(2), u1Norm, u2Norm, u1Theta, u2Theta As Double
Dim z1(2), z2(2), z3(2), z4(2), z5(2), z6(2)  As Double
Dim z1Norm, z2Norm, z3Norm, z4Norm, z5Norm, z6Norm, z1Theta, z2Theta, z3Theta,
z4Theta, z5Theta, z6Theta As Double
Dim y1(2), y2(2), y3(2), y4(2), y5(2), y6(2)  As Double
```

```
Dim y1Norm, y2Norm, y3Norm, y4Norm, y5Norm, y6Norm, y1Theta, y2Theta, y3Theta,
y4Theta, y5Theta, y6Theta As Double
Dim xtmp(6, 2) As Variant
'Dim xtmp(3, 2) As Double
Dim Threshold As Double
Threshold = 0.0000000000001


b = BB / AA
c = CC / AA
d = DD / AA

e = c - (b ^ 2) / 3
f = (2 * b ^ 3) / 27 - (c * b) / 3 + d

'At this point we have a quadratic of the form u^2 + f* u - e^3/27 = 0
'where u = z^3
'Solve this quadratic

AAA = 1
BBB = f
CCC = -e ^ 3 / 27

If BBB ^ 2 - 4 * AAA * CCC < 0 Then
   'If this is the case then we have an imaginary solution
   'Open "C:\Documents and Settings\Luc\My
Documents\DATA\Mun\Programs\Vieta\temp.txt" For Output As #1
   u1(1) = -BBB / (2 * AAA)                          'Real part
   u1(2) = Sqr(Abs(BBB ^ 2 - 4 * AAA * CCC)) / (2 * AAA)         'Imaginary part,
Positive root
   u2(1) = -BBB / (2 * AAA)                          'Real part
   u2(2) = -Sqr(Abs(BBB ^ 2 - 4 * AAA * CCC)) / (2 * AAA)        'Imaginary part,
Negative root
'   Print #1, "u1(1) = " & u1(1)
'   Print #1, "u1(2) = " & u1(2)
'   Print #1, "u2(1) = " & u2(1)
'   Print #1, "u2(2) = " & u2(2)
'   Close #1

   'At this point we have solve for z in z^3 = u
   'Since there are two values of u, there are six values of z
   'four of which are complex
   'The best thing to do is to convert u1 and u2 in to polar form
   u1Norm = Norm(u1(1), u1(2))
   u1Theta = Angle(u1(2), u1(1))
   u2Norm = Norm(u2(1), u2(2))
```

```
  u2Theta = Angle(u2(2), u2(1))

  z1(1) = CubeRoot(u1Norm) * Cos(u1Theta / 3)
  z1(2) = CubeRoot(u1Norm) * Sin(u1Theta / 3)
  z2(1) = Norm(-1 / 2, Sqr(3) / 2) * CubeRoot(u1Norm) * Cos(Angle(Sqr(3) / 2, -1 / 2) +
u1Theta / 3)
  z2(2) = Norm(-1 / 2, Sqr(3) / 2) * CubeRoot(u1Norm) * Sin(Angle(Sqr(3) / 2, -1 / 2) +
u1Theta / 3)
  z3(1) = Norm(-1 / 2, -Sqr(3) / 2) * CubeRoot(u1Norm) * Cos(Angle(-Sqr(3) / 2, -1 / 2)
+ u1Theta / 3)
  z3(2) = Norm(-1 / 2, -Sqr(3) / 2) * CubeRoot(u1Norm) * Sin(Angle(-Sqr(3) / 2, -1 / 2)
+ u1Theta / 3)

  z4(1) = CubeRoot(u2Norm) * Cos(u2Theta / 3)
  z4(2) = CubeRoot(u2Norm) * Sin(u2Theta / 3)
  z5(1) = Norm(-1 / 2, Sqr(3) / 2) * CubeRoot(u2Norm) * Cos(Angle(Sqr(3) / 2, -1 / 2) +
u2Theta / 3)
  z5(2) = Norm(-1 / 2, Sqr(3) / 2) * CubeRoot(u2Norm) * Sin(Angle(Sqr(3) / 2, -1 / 2) +
u2Theta / 3)
  z6(1) = Norm(-1 / 2, -Sqr(3) / 2) * CubeRoot(u2Norm) * Cos(Angle(-Sqr(3) / 2, -1 / 2)
+ u2Theta / 3)
  z6(2) = Norm(-1 / 2, -Sqr(3) / 2) * CubeRoot(u2Norm) * Sin(Angle(-Sqr(3) / 2, -1 / 2)
+ u2Theta / 3)

Else
  u1(1) = (-BBB + Sqr(BBB ^ 2 - 4 * AAA * CCC)) / (2 * AAA)      'Real part, Positive
root
  u1(2) = 0                                   'Imaginary part
  u2(1) = (-BBB - Sqr(BBB ^ 2 - 4 * AAA * CCC)) / (2 * AAA)      'Real part, Negative
root
  u2(2) = 0                                   'Imaginary part
  'At this point we have solve for z in z^3 = u
  'Since there are two values of u, there are six values of z
  'four of which are complex

  z1(1) = CubeRoot(u1(1))
  z1(2) = 0
  z2(1) = -1 / 2 * CubeRoot(u1(1))
  z2(2) = 1 / 2 * 3 ^ (1 / 2) * CubeRoot(u1(1))
  z3(1) = -1 / 2 * CubeRoot(u1(1))
  z3(2) = -1 / 2 * 3 ^ (1 / 2) * CubeRoot(u1(1))

  z4(1) = CubeRoot(u2(1))
  z4(2) = 0
  z5(1) = -1 / 2 * CubeRoot(u2(1))
  z5(2) = 1 / 2 * 3 ^ (1 / 2) * CubeRoot(u2(1))
```

```
  z6(1) = -1 / 2 * CubeRoot(u2(1))
  z6(2) = -1 / 2 * 3 ^ (1 / 2) * CubeRoot(u2(1))
End If


'At this point we have 6 different solutions for z.
'We now need to solve for y = z -e/(3z)
'First rewrite every complex z number in polar form
z1Norm = Sqr(z1(1) ^ 2 + z1(2) ^ 2)
z2Norm = Sqr(z2(1) ^ 2 + z2(2) ^ 2)
z3Norm = Sqr(z3(1) ^ 2 + z3(2) ^ 2)
z4Norm = Sqr(z4(1) ^ 2 + z4(2) ^ 2)
z5Norm = Sqr(z5(1) ^ 2 + z5(2) ^ 2)
z6Norm = Sqr(z6(1) ^ 2 + z6(2) ^ 2)
z1Theta = Angle(z1(2), z1(1))
z2Theta = Angle(z2(2), z2(1))
z3Theta = Angle(z3(2), z3(1))
z4Theta = Angle(z4(2), z4(1))
z5Theta = Angle(z5(2), z5(1))
z6Theta = Angle(z6(2), z6(1))

y1(1) = (z1Norm - e / (3 * z1Norm)) * Cos(z1Theta)
y1(2) = (z1Norm + e / (3 * z1Norm)) * Sin(z1Theta)
y2(1) = (z2Norm - e / (3 * z2Norm)) * Cos(z2Theta)
y2(2) = (z2Norm + e / (3 * z2Norm)) * Sin(z2Theta)
y3(1) = (z3Norm - e / (3 * z3Norm)) * Cos(z3Theta)
y3(2) = (z3Norm + e / (3 * z3Norm)) * Sin(z3Theta)
y4(1) = (z4Norm - e / (3 * z4Norm)) * Cos(z4Theta)
y4(2) = (z4Norm + e / (3 * z4Norm)) * Sin(z4Theta)
y5(1) = (z5Norm - e / (3 * z5Norm)) * Cos(z5Theta)
y5(2) = (z5Norm + e / (3 * z5Norm)) * Sin(z5Theta)
y6(1) = (z6Norm - e / (3 * z6Norm)) * Cos(z6Theta)
y6(2) = (z6Norm + e / (3 * z6Norm)) * Sin(z6Theta)


Call CheckforZeros(y1(1), Threshold)
Call CheckforZeros(y1(2), Threshold)
Call CheckforZeros(y2(1), Threshold)
Call CheckforZeros(y2(2), Threshold)
Call CheckforZeros(y3(1), Threshold)
Call CheckforZeros(y3(2), Threshold)
Call CheckforZeros(y4(1), Threshold)
Call CheckforZeros(y4(2), Threshold)
Call CheckforZeros(y5(1), Threshold)
Call CheckforZeros(y5(2), Threshold)
Call CheckforZeros(y6(1), Threshold)
```

```
Call CheckforZeros(y6(2), Threshold)

'Now we finally get the roots of out cubic equations by solving for
'x = y - b/3
'In principal only three of these roots should be unique.


xtmp(1, 1) = y1(1) - b / 3
If y1(2) <> 0 Then
   xtmp(1, 2) = y1(2) '- b / 3
Else
   xtmp(1, 2) = 0
End If

xtmp(2, 1) = y2(1) - b / 3

If y2(2) <> 0 Then
   xtmp(2, 2) = y2(2) '- b / 3
Else
   xtmp(2, 2) = 0
End If

xtmp(3, 1) = y3(1) - b / 3

If y3(2) <> 0 Then
   xtmp(3, 2) = y3(2) '- b / 3
Else
   xtmp(3, 2) = 0
End If

xtmp(4, 1) = y4(1) - b / 3

If y4(2) <> 0 Then
   xtmp(4, 2) = y4(2) '- b / 3
Else
   xtmp(4, 2) = 0
End If

xtmp(5, 1) = y5(1) - b / 3

If y5(2) <> 0 Then
   xtmp(5, 2) = y5(2) '- b / 3
Else
   xtmp(5, 2) = 0
End If
```

```
xtmp(6, 1) = y6(1) - b / 3

If y6(2) <> 0 Then
   xtmp(6, 2) = y6(2) '- b / 3
Else
   xtmp(6, 2) = 0
End If

k = 1
For i = 1 To 6
   If xtmp(i, 1) <> "" And xtmp(i, 2) <> "" Then
      x(k, 1) = xtmp(i, 1)
      x(k, 2) = xtmp(i, 2)
      For j = 1 To 6
         If i <> j Then
            If x(k, 1) Like xtmp(j, 1) And x(k, 2) Like xtmp(j, 2) Then
               xtmp(j, 1) = ""
               xtmp(j, 2) = ""
            End If
         End If
      Next j
      k = k + 1
      If k = 4 Then Exit For
   End If
Next i

End Sub
Public Function CubeRoot(a) As Double
Dim tmp
If a >= 0 Then
   CubeRoot = a ^ (1 / 3)
Else
   CubeRoot = -Abs(a) ^ (1 / 3)
End If
End Function
Public Function Norm(a, b) As Double
   Norm = Sqr(a ^ 2 + b ^ 2)
End Function
Public Function Angle(deltay, deltax) As Double
Dim Pi As Double
Pi = 4 * Atn(1)

If deltax > 0 And deltay = 0 Then
   Angle = 0
ElseIf deltax < 0 And deltay = 0 Then
   Angle = Pi
```

```
ElseIf deltax >= 0 And deltay >= 0 Then
   Angle = Atn(deltay / deltax)
ElseIf deltax < 0 And deltay >= 0 Then
   Angle = Atn(Abs(deltax) / deltay) + Pi / 2
ElseIf deltax < 0 And deltay < 0 Then
   Angle = Atn(Abs(deltay) / Abs(deltax)) + Pi
ElseIf deltax >= 0 And deltay < 0 Then
   Angle = Atn(Abs(deltax) / Abs(deltay)) + 3 * Pi / 2
End If

End Function
Public Sub CheckforZeros(a, Threshold)
   If Abs(a) < Threshold Then a = 0
End Sub


Public Sub FitStraightLine(iTnp, Data, m, b)
'This routine assumes that he data has two columns x -> Data(1,i) y -> Data(2,i)
'and the tnp is the total number of data points in the array Data()
'This routine fines the best fit line to the data.
'The line is y = m*x + b
Dim S, Sx, Sy, Sxx, Sxy As Double
Dim Delta As Double
'Dim m, b As Double

S = 0
Sx = 0
Sy = 0
Sxx = 0
Sxy = 0

For i = 1 To iTnp
   S = S + 1
   Sx = Sx + Data(1, i)
   Sy = Sy + Data(2, i)
   Sxx = Sxx + Data(1, i) ^ 2
   Sxy = Sxy + Data(2, i) * Data(1, i)
Next i
Delta = S * Sxx - Sx ^ 2
b = (Sxx * Sy - Sx * Sxy) / Delta
m = (S * Sxy - Sx * Sy) / Delta

End Sub
```

Appendix C:
Visual Basic Program Code for Polynomial Fit

```
Option Explicit
Dim fit As New RegressionObject

Private Sub Command1_Click()

Dim i, NPT, j, k, q, NewQ, ImageNPT, AllCoeffNPT, PolyPoint, Interface() As Integer
Dim dataX(), dataY(), ImageDataX(), ImageDataY() As Double
Dim Time(), Gauge(), Coeff0(), Coeff1(), Coeff2(), Coeff3(), Coeff4() As Double
Dim Origin, DeltaPy, PI As Double
Dim X#, Xmin#, Xmax#, Ymin#, Ymax#, Y#
Dim Blue, Green As Integer

PI = Atn(1) * 4
fit.Degree = 4 'we want a 4th order polynomial
Origin = Val(OriginText.Text)

Open "C:\Documents and Settings\Josh\Desktop\timereading.dat" For Input As #1
Open "C:\Documents and Settings\Josh\Desktop\ALLCoeffs-" & OriginText.Text &
".dat" For Output As #2

'Input timereading.dat as Polynomial fit data
i = 0
Do While (EOF(1) = False)
ReDim Preserve dataX(i)
ReDim Preserve dataY(i)

Input #1, dataX(i), dataY(i) ' time and position (mm)
dataY(i) = (dataY(i) - Origin) * 1000  ' microns
i = i + 1
Loop

NPT = i - 1


'**************************************************************************
*********************************
'**************************************************************************
*********************************
Dim inCounter
Dim inMaxX As Integer
```

```
Dim inMaxY As Integer
Dim inLmarg As Integer
Dim inRmarg As Integer
Dim inBmarg As Integer
Dim inTmarg As Integer
Dim inXPos As Integer
Dim inYPos As Integer
Dim YLabel, XLabel As Double
Dim XTicks, YTicks As Double
'Dim Xmin, Xmax, Ymin, Ymax As Double
Dim ScaleX, ScaleY As Double
Dim XOrigin, YOrigin As Double

Xmin = dataX(0)
Xmax = dataX(NPT)
Ymin = dataY(0)
Ymax = dataY(NPT)

MainForm.Pic1.ForeColor = vbBlack
MainForm.Pic1.AutoRedraw = True
MainForm.Pic1.ScaleMode = 3
MainForm.Pic1.Cls

'Determine the maximum size of chart
inMaxX = MainForm.Pic1.ScaleWidth
inMaxY = MainForm.Pic1.ScaleHeight

'Determine the chart margins, including
'width for the axis labels
inLmarg = MainForm.Pic1.TextWidth("10000")
inBmarg = 1.35 * MainForm.Pic1.TextHeight("5000")
inRmarg = inMaxX - 0.5 * inLmarg
inTmarg = 0.25 * inLmarg
inBmarg = inMaxY - inBmarg

'Determine scale factors for each axis
ScaleX = (inRmarg - inLmarg) / (Xmax - Xmin)
ScaleY = (inBmarg - inTmarg) / (Ymax - Ymin)

'Determine the origin of the graph
If Xmin <= 0 Then
   XOrigin = inLmarg + Abs(Xmin) * ScaleX
Else
   XOrigin = inLmarg - Xmin * ScaleX
End If
YOrigin = inBmarg + Ymin * ScaleY
```

```
'Draw a blue lines to show the origin
MainForm.Pic1.ForeColor = vbBlue
MainForm.Pic1.Line (inLmarg, YOrigin)-(inRmarg, YOrigin)    'This draws the real
graphical abscissa
MainForm.Pic1.Line (XOrigin, inTmarg)-(XOrigin, inBmarg)    'This draws the real
graphical ordinate

MainForm.Pic1.ForeColor = vbBlack
'Draw Axes
MainForm.Pic1.Line (inLmarg, inTmarg)-(inLmarg, inBmarg)    'This draws the left
ordinate
MainForm.Pic1.Line -(inRmarg, inBmarg)                       'This draws the bottom abscissa
MainForm.Pic1.Line (inLmarg, inTmarg)-(inRmarg, inTmarg)    'This draws the top
abscissa
MainForm.Pic1.Line (inRmarg, inTmarg)-(inRmarg, inBmarg)    'This draws the right
ordinate

'Draw labels and tic marks for vertical axis
YTicks = ((Ymax - Ymin) / 5)
YLabel = Format(Ymin, "#0.0")

For inCounter = 1 To 6

   MainForm.Pic1.CurrentX = 5
   inYPos = inBmarg - ((inCounter - 1) * YTicks * ScaleY)
   MainForm.Pic1.CurrentY = inYPos
   MainForm.Pic1.Print Str(FormatNumber(YLabel, 2, vbUseDefault, vbUseDefault,
vbFalse))
   YLabel = YLabel + YTicks
   MainForm.Pic1.Line (inLmarg, inYPos)-(inLmarg + 5, inYPos)   '5 is the length of the
tick mark in pixels

Next inCounter

'Draw labels and tic marks for horizontal axis
XTicks = ((Xmax - Xmin) / 5)
XLabel = Format(Xmin, "#0.0")

For inCounter = 1 To 6

   inXPos = inLmarg + ((inCounter - 1) * XTicks * ScaleX)
   MainForm.Pic1.CurrentX = inXPos - MainForm.Pic1.TextWidth("00") / 2
   MainForm.Pic1.CurrentY = inBmarg + 5
   MainForm.Pic1.Print Str(FormatNumber(XLabel, 1, vbUseDefault, vbUseDefault,
vbFalse))
```

```
   XLabel = XLabel + XTicks
   MainForm.Pic1.Line (inXPos, inBmarg)-(inXPos, inBmarg - 5)

Next inCounter


For i = 0 To NPT

      inXPos = XOrigin + (dataX(i) * ScaleX)
      inYPos = YOrigin - (dataY(i) * ScaleY)

      Pic1.Circle (inXPos, inYPos), 1, RGB(0, 0, 0)

Next i

'*************************************************************************
*********************************
'*************************************************************************
*********************************

PolyPoint = CInt(PolyPointText.Text)

Blue = 0
Green = 0

For i = 1 To NPT - PolyPoint

      For j = 0 To PolyPoint
         fit.XYAdd dataX(i + j), dataY(i + j) 'add data to the fit
      Next j

      Text1.Text = "y = " & fit.Coeff(0) & " + " & fit.Coeff(1) & "x + " & fit.Coeff(2) &
"x^2 + " & fit.Coeff(3) & "x^3 + " & fit.Coeff(4) & "x^4"
      Text3.Text = fit.Coeff(0)
      Text4.Text = fit.Coeff(1)
      Text5.Text = fit.Coeff(2)
      Text6.Text = fit.Coeff(3)
      Text7.Text = fit.Coeff(4)

      Print #2, dataX(i) & "," & dataY(i) & "," & fit.Coeff(0) & "," & fit.Coeff(1) & "," &
fit.Coeff(2) & "," & fit.Coeff(3) & "," & fit.Coeff(4)

      'Plot!!!!!!!!
      Xmin = dataX(i)
      Xmax = dataX(i + j - 1)
```

```
    Pic1.CurrentX = XOrigin + (Xmin * ScaleX)
    Pic1.CurrentY = YOrigin - (fit.RegVal(Xmin) * ScaleY)

    For X = Xmin To Xmax
      inXPos = XOrigin + (X * ScaleX)
      inYPos = YOrigin - (fit.RegVal(X) * ScaleY)

      If Blue > 51 Then Blue = 51
      If Green > 51 Then Green = 51

      Pic1.Line -(inXPos, inYPos), RGB(255, 255 - Green * 5, Blue * 5)
    Next X

    'clear for next run
    j = 0
    fit.Init
    Blue = Blue + 1
    Green = Green + 1

Next i


Close #1
Close #2


'Reload All Coeff for calculation
Open "C:\Documents and Settings\Josh\Desktop\ALLCoeffs-" & OriginText.Text &
".dat" For Input As #3
Open "C:\Documents and Settings\Josh\Desktop\output.csv" For Input As #4
Open "C:\Documents and Settings\Josh\Desktop\DeltaPy-" & OriginText.Text & ".dat"
For Output As #5

Print #5, OriginText.Text, "Time", "Image", "Gauge", "Fit", "DeltaPy"

'Input All coeff for Poly fit
q = 0
Do While (EOF(3) = False)
ReDim Preserve Time(q)
ReDim Preserve Gauge(q)
ReDim Preserve Coeff0(q)
ReDim Preserve Coeff1(q)
ReDim Preserve Coeff2(q)
ReDim Preserve Coeff3(q)
ReDim Preserve Coeff4(q)
```

```
Input #3, Time(q), Gauge(q), Coeff0(q), Coeff1(q), Coeff2(q), Coeff3(q), Coeff4(q)
q = q + 1
Loop

AllCoeffNPT = q - 1


'Input Output.csv as image data
k = 0
Do While (EOF(4) = False)
ReDim Preserve ImageDataX(k) 'time
ReDim Preserve ImageDataY(k) 'distance
ReDim Preserve Interface(k)

Input #4, Interface(k), ImageDataX(k), ImageDataY(k) ' time and position (micron)
k = k + 1
Loop

ImageNPT = k - 1


'Do the math when Imagetime falls in the readingtime scale
For k = 0 To ImageNPT

   For q = NewQ To AllCoeffNPT - 1

      If ImageDataX(k) <= Time(NewQ) Then
      NewQ = q
      Exit For
      ElseIf ImageDataX(k) <= Time(q + 1) Then
      NewQ = q + 1
      Exit For
      End If

   Next q


X = ImageDataX(k)
Y = Coeff0(NewQ) + Coeff1(NewQ) * X + Coeff2(NewQ) * X * X + Coeff3(NewQ) *
X * X * X + Coeff4(NewQ) * X * X * X * X

DeltaPy = Tan(60 * PI / 180) * (ImageDataY(k) - Y)

Print #5, ImageDataX(k), ImageDataY(k), Gauge(NewQ), Y, DeltaPy

Next k
```

```
Close #3
Close #4
Close #5
End Sub




Option Explicit

Private Const MaxO& = 25
Private GlobalO& '"Ordnung" = degree of the polynom expected
Private Finished As Boolean

Private SumX#(0 To 2 * MaxO)
Private SumYX#(0 To MaxO)
Private M#(0 To MaxO, 0 To MaxO + 1)
Private C#(0 To MaxO) 'coefficients in: Y = C(0)*X^0 + C(1)*X^1 + C(2)*X^2 + ...

Private Sub GaussSolve(O&)
'gauss algorithm implementation,
'following R.Sedgewick's "Algorithms in C", Addison-Wesley, with minor modifications
Dim i&, j&, k&, iMax&, T#, O1#
 O1 = O + 1
 'first triangulize the matrix
 For i = 0 To O
  iMax = i: T = Abs(M(iMax, i))
  For j = i + 1 To O 'find the line with the largest absvalue in this row
   If T < Abs(M(j, i)) Then iMax = j: T = Abs(M(iMax, i))
  Next j
  If i < iMax Then 'exchange the two lines
   For k = i To O1
    T = M(i, k)
      M(i, k) = M(iMax, k)
            M(iMax, k) = T
   Next k
  End If
  For j = i + 1 To O 'scale all following lines to have a leading zero
   T = M(j, i) / M(i, i)
   M(j, i) = 0#
   For k = i + 1 To O1
    M(j, k) = M(j, k) - M(i, k) * T
   Next k
  Next j
 Next i
```

```
  'then substitute the coefficients
  For j = O To 0 Step -1
    T = M(j, O1)
    For k = j + 1 To O
      T = T - M(j, k) * C(k)
    Next k
    C(j) = T / M(j, j)
  Next j
  Finished = True
End Sub

Private Sub BuildMatrix(O&)
Dim i&, k&, O1&
  O1 = O + 1
  For i = 0 To O
    For k = 0 To O
      M(i, k) = SumX(i + k)
    Next k
    M(i, O1) = SumYX(i)
  Next i
End Sub

Private Sub FinalizeMatrix(O&)
Dim i&, O1&
  O1 = O + 1
  For i = 0 To O
    M(i, O1) = SumYX(i)
  Next i
End Sub

Private Sub Solve()
Dim O&
  O = GlobalO
  If XYCount <= O Then O = XYCount - 1
  If O < 0 Then Exit Sub
  BuildMatrix O
  On Error Resume Next
    GaussSolve (O)
    While (Err.Number <> 0) And (1 < O)
      Err.Clear
      C(0) = 0#
      O = O - 1
      FinalizeMatrix (O)
    Wend
  On Error GoTo 0
End Sub
```

```
Private Sub Class_Initialize()
  Init
  GlobalO = 2
End Sub

Public Sub Init()
Dim i&
  Finished = False
  For i = 0 To MaxO
    SumX(i) = 0#
    SumX(i + MaxO) = 0#
    SumYX(i) = 0#
    C(i) = 0#
  Next i
End Sub

Public Property Get Coeff#(Exponent&)
Dim Ex&, O&
  If Not Finished Then Solve
  Ex = Abs(Exponent)
  O = GlobalO
  If XYCount <= O Then O = XYCount - 1
  If O < Ex Then Coeff = 0# Else Coeff = C(Ex)
End Property

Public Property Get Degree&()
  Degree = GlobalO
End Property
Public Property Let Degree(NewVal&)
  If NewVal < 0 Or MaxO < NewVal Then
    Err.Raise 6000, "RegressionObject", NewVal & " is an invalid property value! Use
0<= Degree <= " & MaxO
    Exit Property
  End If
  Init
  GlobalO = NewVal
End Property

Public Property Get XYCount&()
  XYCount = CLng(SumX(0))
End Property

Public Function XYAdd(ByVal NewX#, ByVal NewY#)
Dim i&, j&, TX#, Max2O&
  Finished = False
```

```
  Max2O = 2 * GlobalO
  TX = 1#
  SumX(0) = SumX(0) + 1
  SumYX(0) = SumYX(0) + NewY
  For i = 1 To GlobalO
    TX = TX * NewX
    SumX(i) = SumX(i) + TX
    SumYX(i) = SumYX(i) + NewY * TX
  Next i
  For i = GlobalO + 1 To Max2O
    TX = TX * NewX
    SumX(i) = SumX(i) + TX
  Next i
End Function

Public Function RegVal#(X#)
Dim i&, O&
  If Not Finished Then Solve
  RegVal = 0#
  O = GlobalO
  If XYCount <= O Then O = XYCount - 1
  For i = 0 To O
    RegVal = RegVal + C(i) * X ^ i
  Next i
End Function
```