This experiment is located in room 63. Keys to this room are available in room 65, next to the blackboard. Please return them to room 65 when you are done.

Please do not move, alter, or adjust any of the equipment in this setup except for the equipment specifically mentioned in this write-up! Finally, write, debug and test your code for this exercise with the available setups in room 65 before you go and use it on the “real” system in room 63!

3.1. Introduction

This experiment will introduce you to some of the fundamentals of theoretical and modern experimental optics. You will use a combination of spatial filter, polarizer and lens to examine diffraction by a straight edge. From a careful study of these diffraction patterns, you will be able to measure the wavelength of the light emitted by the laser. The diffraction pattern is recorded by scanning a light sensitive detector along the pattern. (See figure 3.1.)

Figure 3.1. View of the experimental setup.
The detector is a photomultiplier tube (PMT) which outputs a current proportional to the intensity of the light to which it is exposed. The current is converted to a voltage, which is monitored by a Hewlett Packard 34401A multimeter with a GPIB interface, which will make it possible to read the voltages directly into the computer. The detector is scanned along the pattern by a stepper motor driver controlled by the PC through your prototype card.

### 3.2. Theory

An interference pattern is created when an object partially blocks the path of a monochromatic light source (such as our laser). Constructive interference occurs when there is a path length difference which is an integral multiple of the wavelength of the light source. When the object is a straight edge, the interference pattern will occur on one side only and is caused by a path length difference between light coming from a point source

For the special case of spherical wave fronts incident on a straight edge (see figure 3.1. and figure 3.2.), the $m^{th}$ diffraction maximum can be found at position:

$$x_{\text{max}}(m) = \frac{\lambda L (L - y) (4m + \frac{3}{2})}{2y}$$

where $m = 0, 1, 2, \ldots$ (3.1.)

and the $m^{th}$ diffraction minimum at:

$$x_{\text{min}}(m) = \frac{\lambda L (L - y) (4m + \frac{7}{2})}{2y}$$

where $m = 0, 1, 2, \ldots$ (3.2.)

A plot of intensity vs. position is shown in figure 3.2.

![Figure 3.2. Diffraction pattern produced by a straight edge and an infinitely wide beam. On the left is the dark part of the image. The right side shows the illuminated part of the image. (Due to the finite beam width, the "real" data will show an overall drop in intensity as we move away from the beam center which is usually located near x = 0.)](image)

### 3.3. Apparatus

The setup is located in the optics room. All the equipment is mounted on the optical table. Starting with the laser, the entire experimental setup consists of the following elements:
Laser

The light beam is provided by a polarized He-Ne laser which has a wavelength $\lambda = 632.8$ nm.

Polarizer

In this experiment, a polarizer is placed between the laser and the spatial filter. Since the output beam of the laser is already polarized, rotating this polarizer is simply a way of controlling the intensity of output light. Malus's law states that the intensity of the beam is:

$$I(\phi) = I_0 \cos^2(\phi)$$

(3.5.)

where $\phi$ is the angle between the transmission axes of the laser and polarizer. To prevent light from being reflected off the polarizer back into the laser, the polarizer should be turned slightly away from being perpendicular to the laser beam.

Spatial Filter

The Lens-Pinhole Spatial Filter (LPSF) is designed to provide a very homogeneous Gaussian-profile laser beam. If a lens, in our case a microscope objective, is placed in the path of a light beam, it will focus the beam down to a point, from which it will expand into a wider beam with a Gaussian profile.

With a lens with a very short focal length, the beam will rapidly converge and then spread out into a wide beam. By placing a piece of foil with a very small pinhole (ours has a 50 $\mu$m diameter) at the focal point of the lens, nearly all the light which is diffracted by dust or other means is removed from the beam. The result is a rapidly diverging, very homogeneous output beam. The wave fronts of the beam are circular, thus we are able to study Fresnel diffraction by a straight edge such as the razor blade.

Remove any object which is between the spatial filter and the target. If the spatial filter is already in place, examine its output by placing a large white card against the PMT carriage. On the card the beam should appear as a large disk (about 5 cm across) with little or no diffraction rings or patterns.

If this is not the case you may fine-tune the spatial filter by carefully adjusting the focus screw or the pinhole positioning micrometers but be forewarned, the spatial filter is extremely touchy and a small adjustment can mess up the whole alignment and make your experimental setup useless! Setting up the spatial filter can be a very time consuming process! You may want to ask your TA or Kurt Wick to do it for you.
**Spatial Filter Setup**

**Read First:** Only if the spatial filter is completely out of alignment you may start from scratch and realign everything using the instructions given below. If the beam looks ok, skip this section!

Before proceeding with the instructions below ask your TA for his or her opinion or help! Setting up the spatial filter can be extremely difficult and time consuming, especially if you have never done so!

- With the room lights on, position the Lens-Pinhole Spatial Filter (LPSF) on the stand provided.
- Remove microscope objective and pinhole. Be careful not to let anything touch the foil which has the pinhole in it.
- Hold the positioning card behind the front hole of the LPSF and adjust leg screws until beam hits the center of the crosshairs.
Move the positioning card to the back hole and readjust leg screws until the beam hits the crosshairs.

Now realign the front hole, then the back hole, until both are well-centered. Have patience, this may take a while.

Move the two holes as far apart as possible by adjusting the objective focusing screw.

Screw in the objective and put the pinhole in place.

Place a large white card against the PMT carriage so that you can see the output beam of the spatial filter on it.

Turn off the main room lights (you can leave the door open to let some light in).

Adjust the micrometers holding the pinhole until the most intense pattern can be seen on the card. This will probably be a central disk with diffraction rings around it. You may have a hard time finding anything at first so be patient and take a systematic approach.

Slowly turn the z-axis screw so the lens is closer to the pinhole. You will need to continually adjust the two micrometers to keep the most intense pattern on the card.

As you get closer to the best focus point, the beam will become one central disk with faint diffraction rings around it.

You have found the best focal point when you can see the fewest diffraction rings and the cleanest, brightest central disk.

If you lose the beam, search for it systematically

**Diffracting Objects**

You will diffract the laser beam from the edge of a razor blade.

**Photomultiplier Tube**

The photodetector is a photomultiplier tube, which is capable of measuring extremely low light levels. It is a very linear device which will give an output proportional to the illumination of its sensitive photocathode over many orders of magnitude. Its output voltage may be measured with a HP34401A multimeter using a GPIB connection to a PC. For more specific information on how a photomultiplier tube works read this lab manual, chapter 4.2.

In the interests of safety (both the photomultiplier's and yours), the voltage distribution resistors which set the potentials of the electron multiplier section of the tube have been made very large so that its output current is self-limiting. This means that the output will saturate whenever it is exposed to a very bright light, but the photomultiplier will not be damaged if you operate it in room lights or with the laser beam shining directly on it.

Operate the tube at a high voltage of -1350 V. Do not adjust the phototube's supply voltage. Of course, you should collect your data in darkness.
GPIB

The General Purpose Interface Bus (GPIB) has become an industry standard on how to interconnect electronic devices to transfer data. In our experiment we will use it to read the photomultiplier tube voltages, as measured by the HP34401A voltmeter, directly into the PC. Section 3.7. explains the GPIB programming aspects.

Stepper Motor

A stepper motor is used to drive the phototube across the diffraction pattern. A description on interfacing the stepper motor is given in section 3.6.

For this experiment, the motor shaft rotates a precision screw, which in turn drives the carriage supporting the photodetector. 400 pulses will cause the motor shaft to rotate by one complete revolution. 5 revolutions will advance the table by one inch.

3.4. Data Collection

After you have written and debugged your code to acquire the data (see sections 3.6 to 3.8.) familiarize yourself with the optical setup.

The spatial filter should have been set up properly and should not need any adjustments. Nevertheless, you should check that it has not been tampered with. Temporarily remove the razor blade and place a sheet of white cardboard in front of the PMT and turn off the room lights. You should see a "clean" Gaussian beam (intensity) profile, i.e., a strong central maxima, vertically centered on the slit in front of the PMT and no (or almost no) diffraction patterns. If this is not the case, see your TA.

Next, adjust the intensity of the laser light so that the PMT will not be saturated: with the room lights turned off, adjust the polarizer until you (roughly) observe the maximum brightness; move the carriage so the PMT is in the center of the beam; record the maximum beam intensity value displayed by the voltmeter attached to the PMT; its absolute value should be about 0.4 to 0.5 Volts. (If this is not the case, make sure the power on the PMT has been turned on.). Next, rotate the polarizer until the voltmeter reads about half of the maximum value.

Now move the razor blade between the spatial filter and the target. You should obtain good results for \( y \approx \frac{3}{8} L \). The razor should bisect the beam; make sure the edge of the blade is at the center of the beam. If you place a white card against the PMT carriage, you should be able to see very narrow vertical diffraction lines close to the edge of the razor's shadow. Be sure to observe these before continuing.

Before you take data, carefully measure \( L \) and \( y \). (See figure 3.1.) Note that \( L \) and \( y \) represent the distances from the pinhole to the to the razor blade, or to the aperture directly in front of the PMT respectively. Unfortunately, the pinhole itself is located on a thin, fragile piece of tinfoil inside of the spatial filter and you should not touch it! Instead, measure your distances from the front surface of the spatial filter (directly adjacent to the hole from which the beam emerges) and keep in mind that the foil with the pinhole is located about 11.1 ± 0.2 millimeters inside the filter.

Before you start acquiring the data make one more test run to check that the screw, which controls the carriage position, indeed performs one entire revolution when 400 clock cycles are sent to it.
Finally, run your program and move the carriage with the PMT across the diffraction pattern. You need to record data only over a fairly small range, i.e., you should take readings of only the first 10 interference maxima and minima.

Display and plot the acquired data in an Excel; if it looks similar to Figure 3.2 continue with the data analysis.

### 3.5. Data Analysis

**Part 1: Measurement of the Laser Wavelength**

**Overview of the LSQ Fitting Approach**

From the intensity vs. x-position data collected you can determine the laser's wavelength, $\lambda$, the following way: measure the x-positions of the various minima and maxima and identify their corresponding $m$ value (see Figure 3.2); use this information in a LSQ fit to calculate $\lambda$. (Note: the actual intensity values at the maxima and minima are of no importance; so do not bother to record them for this LSQ fit.)

To determine the LSQ fit parameters, rewrite equations 3.1 and 3.2 as:

$$x_m = \sqrt{\lambda} F(L,y) G(m,k)$$  \hspace{1cm} (3.3.)

where $x_m$ corresponds to the x-position of a maxima or minima. $F(L,y)$ and $G(m,k)$ are defined below (see also Figure 3.1):

$$F(L,y) = \sqrt{\frac{L(L-y)}{2y}}$$ \hspace{1cm} (3.4.)

$$G(m,k) = \sqrt{4m+k}$$ \hspace{1cm} (3.5.)

where $m = 0, 1, 2...$ corresponds to the $m^{th}$ diffraction maxima or minimum and $k = 3/2$ for a maxima or $7/2$ for a minima.

Finally, equation 3.2 needs to be adjusted for an (unknown) offset in the x-axis because as it is written it assumes that the $x = 0$ position is perfectly aligned with the edge of the razor blade (see figure 3.2). In reality, at the start of your data acquisition, your $x = 0$ position was at some arbitrary x-position. Therefore, all your measured $x_m$ positions are offset by an unknown distance, $x_o$. This must be added to equation 3.3, leading to:

$$x_m = \sqrt{\lambda} F(L,y) G(m,k) + x_o$$ \hspace{1cm} (3.6.)
It should now be apparent that fitting the values of \( x_m \) vs. \( F(L,y)G(m,k) \) will result in a linear fit with slope \( \sqrt{\lambda} \) and intercept \( x_o \). While this approach is essentially correct, it involves some subtle but very important points concerning systematic errors vs. random errors that you need to consider. They are explained in the next section.

### Details of the LSQ Fitting Approach

Before setting up the LSQ fit you need to ask yourself an important question: which variable is the dependent variable and which ones the independent ones? The importance of this question is directly related to the error propagation that needs to be carried out for the dependent variable.

Our current LSQ fit algorithm cannot account for errors in both the dependent and independent variables. Instead, it assumes that the (percentage) error in the independent variable, the \( x \)-axis, is much smaller than the one in the independent one, the \( y \)-axis. It then follows that since there is no error in \( m \) or \( k \) that \( G(m,k) \) should be plotted along the \( x \)-axis. Rearranging equation 3.6 to satisfy this condition, with the left hand side of the equation being the dependent “variable,” results in:

\[
\frac{x_m}{F(L,y)} = \sqrt{\lambda} G(m,k) + \frac{x_o}{F(L,y)}
\]

Calculating the error for \( x_m/F(L,y) \) (using the error propagation method with respect to \( \sigma_{x_m}, \sigma_L \) and \( \sigma_y \)) and fitting (as the dependent variable) \( x_m/F(L,y) \) vs. \( G(m,k) \) (as the independent one) will directly result in a slope of \( \sqrt{\lambda} \) and an intercept of \( x_o/F(L,y) \). From this appropriate values for \( \lambda \) and \( x_o \) can easily be calculated. On the other hand, using the values from the uncertainty from the slope and intercept will most likely yield meaningless results for the uncertainties in \( \lambda \), \( x_o \) and the fit’s \( \chi^2 \). So what is went wrong?

What we have neglected so far is to consider the type of errors that entered the analysis. For example, the error in \( x_m \) is a random error. By obtaining many different values for the \( x \)-position of the maxima and minima we improve (hopefully) the accuracy of the fit result because the random errors should be centered around the “correct” result. On the other hand, the errors due to \( L \) and \( y \) are systematic. \( L \) and \( y \) are only measured once or twice at the beginning of the data acquisition and the error in \( F(L,y) \) will not be affected by how many \( x_m \)'s were measured. Therefore, if we perform an error propagation on the left hand side of equation 3.7 with respect to \( \sigma_{x_m}, \sigma_L \) and \( \sigma_y \) then the resulting fit uncertainties will be meaningless.

The correct approach for dealing with the systematic and random errors is to separate them in the LSQ fit. First, we perform an LSQ fit of \( x_m \) vs. \( G(m,k) \). This involves only the random errors and, therefore, should yield the appropriate uncertainties in the slope and intercept. The slope, \( s \), of the fit will be:

\[
s = \frac{\sqrt{\lambda} F(L,y)}{F(L,y)} \tag{3.8.}
\]

and the fit’s intercept is \( x_o \). Solving this equation for \( \lambda \) and performing an error propagation with respect to \( \sigma_s, \sigma_L \) and \( \sigma_y \) will finally yield the appropriate value for \( \sigma_s \). Note that the systematic error, which is due to \( \sigma_L \) and \( \sigma_y \), is added in this last step and will have quite a significant effect on \( \sigma_s \).

### Write-up:

Based on the LSQ fitting method explained in the previous paragraph, calculate the laser wavelength and its uncertainty. In the process you must answer all of the following questions:
a) When performing the LSQ fit of \( x_m \) vs. \( G(m,k) \), what \( \sigma_{xm} \) was required to obtain a \( \chi^2 \) around 1? More specifically, how many steps of the stepper motor does this \( \sigma_{xm} \) correspond to and what do you expect it to be?

b) Derive an equation for \( \lambda \) and \( \sigma_{\lambda} (\sigma_s, \sigma_L, \sigma_y) \) from equation 3.8 and also give a numerical value for \( \lambda \) and \( \sigma_{\lambda} (\sigma_s, \sigma_L, \sigma_y) \) based on your fit results from part a). Specify the values for \( \sigma_L \) and \( \sigma_y \) used in your calculations.

c) The wavelength of a typical He-Ne laser is 632.8 nm. How many sigmas is the result you obtained in part b) off from this accepted value? Calculate (or look up) what the probability of such an event is for a Normal, i.e., Gaussian, distribution.

d) If the result in part c) was more than 3 sigmas off, how much would the values for \( \sigma_L \) and \( \sigma_y \) have to be adjusted to get a reasonable value, i.e., 1 to 2 sigmas?

**Part 2: Intensity at the Razor Blade’s Intercept**

As already mentioned in part 1, the very edge of the razor blade corresponds to the \( x=0 \) position. (See also Figure 3.2.) At this location, half of the incident wave front is obstructed and, therefore, the amplitude is halved resulting in the intensity being reduced to \( \frac{1}{4} \). (Remember: intensity is amplitude squared!)

In your previous LSQ fit, you obtained \( x_0 \), the offset from the \( x=0 \) position. From this and your original data you should be able to determine the intensity of the laser light measured at the “real” \( x = 0 \) position. Record it and calculate what part of the “full” light intensity it represented.

**Write-up:**

e) What was the “full” light intensity and how was it determined. (No need for fancy measurements; use your own judgment of the data already obtained.)

f) What part of the laser light intensity was observed at the \( x = 0 \) position and how does it agree with the predicted value?

### 3.6. Stepper Motor

**Introduction**

You may think of a stepper motor as a “digital motor” which rotates its shaft a preset angle, usually a fraction of a degree, at every digital clock cycle. A detailed description on how stepper motor work and how to interface them to a prototype cards is given in the Fall lab manual, section 11.3.

**Interfacing with the National Instrument PCI-MIO-16E Digital I/O Lines**

You will use the National Instrument PCI-MIO-16E card’s digital I/O lines to control the stepper motor. In addition to the A-to-D and D-to-A converter, the PCI-MIO-16E contains 8 buffered and latched digital I/O lines labeled DIO0 to DIO7. These can be programmed to read or write digital signals. See the table 3.1. for the DIO lines and their corresponding DT21-EZ Terminal Panel pin assignment. The lines can be configured through LabWindows’`WriteToDigitalLine` function which allows you to set any one of the eight DIO lines (DIO0 through DIO8) HI or LO without having to worry about assembling a byte from individual bits.
The \texttt{WriteToDigitalLine} function can be found under: Libraries / Easy I/O for DAQ / Digital Input Output. Though the function has six arguments, the only ones that are of concern to us are the third and the last argument; use the default values for the remaining ones. A typical usage is:

\begin{verbatim}
WriteToDigitalLine (1, "0", 1, 8, 0, bval);
\end{verbatim}

The third argument specifies the DIO line to be controlled and the last argument sets the state of the DIO line; use 1 for HI and 0 LO.

<table>
<thead>
<tr>
<th>DIO Line</th>
<th>Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>DIO0</td>
<td>25</td>
</tr>
<tr>
<td>DIO1</td>
<td>27</td>
</tr>
<tr>
<td>DIO2</td>
<td>29</td>
</tr>
<tr>
<td>DIO3</td>
<td>31</td>
</tr>
<tr>
<td>DIO4</td>
<td>26</td>
</tr>
<tr>
<td>DIO5</td>
<td>28</td>
</tr>
<tr>
<td>DIO6</td>
<td>30</td>
</tr>
<tr>
<td>DIO7</td>
<td>32</td>
</tr>
<tr>
<td>DGnd</td>
<td>33</td>
</tr>
</tbody>
</table>

Table: 3.1. DIO lines and their corresponding DT21-EZ Terminal Panel pin assignment.

To control the stepper motor, you will need two digital signals: The CLOCK signal which rotates the shaft a fraction of a degree and a \textit{CW/CCW} signal to control the direction the shaft rotates. Select any two of the DIO control lines from the above table for these two signals.

\section*{Stepper Motor Programming}

The shaft of the stepper motor rotates when its CLOCK input receives a continuous stream of HI and LO TTL signals. It will move one step at each negative going TTL transition and it requires 400 such HI to LO cycles to turn one complete revolution!

This stream of TTL signals can be generated by calling the \texttt{WriteToDigitalLine} function inside a while-loop which in turn is controlled through a binary switch in your GUI. Consider the sample code shown below which has been placed inside of a callback function associated with such a binary switch called \texttt{PANEL\_MOTORONOFF}.

\begin{verbatim}
switch (event)
{
    case EVENT\_COMMIT:
        GetCtrlVal (panelHandle, PANEL\_MOTORONOFF, &monoff); //read the state of the switch

        //BEGINNING OF WHILE LOOP
        while ( monoff ) //execute as long as 'monoff' is true, i.e., the switch is ON
        {
            //control motor here, i.e., send out TTL signals!
            myclock = ! myclock; //inverts var. 'myclock', i.e., if HI set it LO, if LO set it HI
            WriteToDigitalLine (1, "0", 1, 8, 0, myclock); //sends out signal to DIO1

            //Include additional statements here.
            //For example, control the motor's CW or CCW direction here..
        }
}
\end{verbatim}
Once the binary switch has been turned on, the variable ‘monoff’ will be ‘true.’ The while loop starts and outputs during each (loop) cycle the opposite of the previous TTL signal, i.e., if previously a HI signal was sent it sends a LO signal. Finally, before the loop starts the next cycle it reads the status of the switch again with another GetCtrlVal call. If the switch didn't change its state, the while-loop repeats itself and it will continue to repeat itself for as long as the switch remains on.

Important: the second last statement in the while loop is a `ProcessSystemEvents()` call. This function forces the operating system to process any (system) events that may have occurred while processing the previous statements. Any commit events, such as a right mouse click which could have altered the state of the controls in the GUI are processed before the loop starts a new cycle. This is very important because if these (system) events could not be processed, we would loose control to communicate with the program and the computer would lock up!

Start your program from scratch and create a GUI with the following controls:

- A program termination control.
- An ON/OFF binary switch which activates the stepper motor.
- A CW/CCW binary switch to select the direction of rotation.

Connect your ADC card DIO lines to the red stepper motor control boxes. Specifically, connect:

- the stepper motor CLOCK input to the DIO line that you will select to output the HI / LO clock signals;
- the stepper motor directional input, $CW/CCW$, to the DIO line that you will select to output the directional signal.

You also must connect the ground (DGnd: pin 33) from the ADC card with the GROUND on the stepper motor control box.

**Stepper Motor: Trouble Shooting**

You may find it helpful to add code for your callback functions in the following order:

First, add the program termination control function and test that the program compiles and runs and terminates.

Second, use the code shown above and add it to the callback function of the control (a binary switch) that turns the motor on or off. (You will need to declare some of the variables shown in the code as either global or local variables.) Test your code and, if you haven't already, connect the appropriate DIO line to the stepper motor's *CLOCK* input. Before you proceed, make sure that you can control the motor by turning it on and off with the binary switch in your GUI.
Third, add code to control the motor’s direction: Read the state of the \( \text{CW}/\text{CCW} \) binary switch in your GUI by using a GetCtrlVal function; send its value to a DIO line using another WriteToDigitalLine statement. Both of these statements should be placed inside the while loop discussed above. (Therefore, there’s no need to create a callback function for the \( \text{CW}/\text{CCW} \) control!) Connect the “proper” DIO control line to the motor’s \( \text{CW}/\text{CCW} \) control input and test that everything works.

### 3.7. Software: GPIB and Data Files

Adapt your application from section 3.6 so that with the help of the GPIB interface voltages can be read from the HP34401A multimeter and then stored in an ASCII data file for later analysis in Excel. You will find it useful to add a control to your GUI to start the data acquisition process once the carriage with the PMT has been positioned.

You need to add or modify three sections of your C-code. First, you will need to initialize and configure the GPIB. Second, once the data acquisition process has been started, you will read in the voltages from the multimeter, plot them and store them in an array. Finally write the data to a file and close the GPIB device driver.

**Introduction: Overview of GPIB**

The GPIB project you will be working on makes use of the following hardware components:

- the lab computer with a National Instrument GPIB card installed and,
- the device ‘talking’ to the computer, an HP34401A Multimeter.
- the GPIB cable connecting the computer to the HP34401A Multimeter.

Up to 30 such devices could be connected in series (daisy chained) to the computer using the GPIB cables. (Inspect these cables and note how they can be stacked on top of each other.)

**GPIB Programming: Loading the Instrument Driver**

Though GPIB itself is becoming the standard of interfacing scientific instruments, because hardware standards have long been established and adhered to, when it comes to programming GPIB, there are no such standards. Instead, if you were to start from scratch, you would need to use many low-level C functions, which depend on the specific device that you will be connecting to. Luckily, all of these functions have been combined into a library of functions, or what National Instrument calls an “instrument driver” or “device driver.” This software has already been written by National Instrument and deals with all the tedious and mundane aspects of GPIB programming. Once you have the instrument driver installed, it will provide simple function calls to control almost any property of an instrument. For example, in the case of the HP34401A, only 5 function calls will be needed to initialize, configure, read in voltages and to close the device driver.

These instrument drivers are device specific and are almost guaranteed not to work with any other device or model than the one specified. (For a list of free, available instrument drivers check: http://www.natinst.com/idnet97.nsf/) The required files for the instrument driver for the HP34401A have already been uploaded into the directory: \( U:\text{\pub}\text{\ LW}\text{\InstDrivers} \).

Copy the four files from the \( U:\text{\pub}\text{\ LW}\text{\InstDrivers}\text{\ HP34401a} \) directory into the directory where you are developing your application.
To access the instrument driver's function, you must add the recently copied instrument driver file (hp34401a.fp) to your project. This is done by selecting in the project window: Edit \ Add Files to Project... \ Instrument (*.fp.). Select the hp34401a.fp file and add it to the list of project files. Aside from this file, do not add any of the other instrument driver files to your project window! Once you have added the hp34401a.fp file, all other necessary files will be loaded automatically each time you start your application.

What follows is a detailed description of the four functions required to control the GPIB.

Initialization and Configuring the HP34401A

Before the data acquisition process begins, you need to initialize and configure the HP34401A. This is done by the three functions: hp34401a_init, hp34401a_conf and hp34401a_confTrig. These and all the other HP34401a instrument driver functions can be found in LabWindows under: Instruments \ HP 34401A Multimeter (VISA/IO)....

The arguments for the hp34401a_init function are:

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Value Required</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>GPIB Address</td>
<td>“GPIB::01”</td>
<td>Refers to the physical GPIB address. See comments below.</td>
</tr>
<tr>
<td>ID Query</td>
<td>VI_ON</td>
<td>Default setting: Do ID Query. Do not change.</td>
</tr>
<tr>
<td>Reset Device</td>
<td>VI_ON</td>
<td>Default setting: Reset Device. Do not change.</td>
</tr>
<tr>
<td>Instrument ID</td>
<td>Pointer to a variable</td>
<td>Declare this variable. It identifies the instrument in case there are multiple instruments present.</td>
</tr>
</tbody>
</table>

This function initializes the instrument and checks that the instrument is connected properly and resets it.

The hp34401a_conf function sets the instrument into the appropriate measurement mode. The arguments are:

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Value Required</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instrument ID</td>
<td>Pointer to a variable</td>
<td>Use the variable declared previously in the hp34401a_init function.</td>
</tr>
<tr>
<td>Function</td>
<td>Volt DC</td>
<td>This defines the type of measurement the multimeter will perform.</td>
</tr>
<tr>
<td>Autorange</td>
<td>On</td>
<td>Default setting. Do not change.</td>
</tr>
<tr>
<td>Resolution</td>
<td>5 ½ digits</td>
<td>This defines the resolution of the measurements. See comments below. (5 ½ digits is “good enough” for this experiment.)</td>
</tr>
<tr>
<td>Range</td>
<td>0.0</td>
<td>Ignore, i.e., use default setting.</td>
</tr>
</tbody>
</table>

The "Resolution" argument not only specifies the accuracy at which data is collected, it is also relates to the speed of the data acquisition. As you might remember from previous exercises, increasing the accuracy or resolution requires more bits or more measurements to be performed, both conditions, which increase the time required to acquire one reading. In the case of the HP34401A, if you switch from 5½ digits to 6½ digits resolution you will notice that it will take longer to take one measurement. Keep this in mind, if you should select a higher accuracy.

Finally, the hp34401a_confTrig specifies the trigger mode to be used:

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Value Required</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Instrument ID</td>
<td>Pointer to a variable</td>
<td>Use the variable declared previously in the hp34401a_init function.</td>
</tr>
<tr>
<td>Trigger Source</td>
<td>Internal</td>
<td>(Default) This defines that the multimeter provides its own trigger.</td>
</tr>
</tbody>
</table>
Auto Trigger Delay | On | Default setting. Do not change.
---|---|---
Trigger Delay | 0.000001 | Default setting. Do not change.
Trigger Count | 1 | Default setting. Do not change.
Sample Count | 1 | Number of samples to be acquired for each trigger received.

Sample code to initialize and configure the instrument is shown below.

```c
static ViSession insthandle; //a variable to refer to the instrument
hp34401a_init ("GPIB::01", VI_ON, VI_ON, &insthandle);
hp34401a_conf (insthandle, 1, VI_ON, 0.0, 1);
hp34401a_confTrig (insthandle, 0, VI_ON, 0.000001, 1, 1);
```

The "GPIB Address" refers to the physical address of the instrument attached to the GPIB card. Since up to 30 different GPIB devices can be connected together, each device must have its own unique address or device number. The device number specified in the `hp34401a_init` function argument, arbitrarily chosen to be 1, must match the one specified in the HP34401A Multimeter.

To test if this condition is satisfied, i.e., to check the device number that the HP34401A has been set to, turn your HP34401A on and then press the following keys:

- **SHIFT MENU**
  
  You should see: A: Meas Menu

- Hit the right cursor 4 times till you see:
  
  E: I / O MENU

- Hit the down cursor till you see:
  
  1: HP-IB ADDR

- Again hit the down cursor till you see:
  
  XY ADDR

  In this case XY refers to the address of the device.

- If you want to leave it at the current setting, hit ENTER to exit the menu. Otherwise, use the cursors to set it to a new value. Warning: do not use device address 0 since the GPIB card in the computer uses it! When done, hit ENTER to exit the menu.

Once you have entered the device address, the instrument will "remember" it even after being turned off.

**Acquiring Data:** `hp34401a_singleMeas( Instrument ID, &ValueIn );`

To read one measurement from the HP34401A into the computer, call the `hp34401a_singleMeas` function. The function consists of only two arguments: Instrument ID, which has been described previously, and "ValueIn," the value measured that is being returned by the meter. The "ValueIn" argument is passed by reference, i.e., it is a pointer to a variable of type `ViReal64`, i.e., a 64 bit "double."

For debugging purposes, it might be helpful to display the "ValueIn" variable in a display panel in your GUI. Also checking the return argument of the `hp34401a_singleMeas` function for any errors, or non-zero values, could eventually save you a lot of time later because it will warn you in case the GPIB
is not working properly. Finally, you should save the value returned by the measurement function in an array so that the data can be plotted and saved to a data file later. In addition, you should also keep track of how many data points have been acquired. You will need this number later when you must specify the number of data points to be written to the data file.

Closing the Instrument Driver

Before you terminate the program, you must call the `hp34401a_close(Instrument ID)` function.

Data Files

Your data must be saved to a data file. This is a common practice to allow other "researchers" to use the equipment while you analyze the data.

To do so you may either write your code from scratch using a `fprintf` statement as outlined in section 11.5 of the C & LabWindows Fundamentals Manual or you may use the LabWindows `ArrayToFile` function. (See below.)

Writing your own code using a `fprintf` statement will require about 5 lines of code: declaring the file pointer, opening the file with `fopen`, declaring a for-loop, writing each value to the file using `fprintf` and, finally, closing the file with `fclose`. Again, see section 11.5 of the C & LabWindows Fundamentals Manual.

On the other hand, if you prefer to use the `ArrayToFile` function, it can be found under: Library / Formatting and I/O / File I/O. Its function arguments are:

<table>
<thead>
<tr>
<th>Argument Name</th>
<th>Value Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>Filename</td>
<td>A file name enclosed in either quotation marks or a pointer to a character array storing the file name. See comments below.</td>
</tr>
<tr>
<td>Array</td>
<td>A pointer to the array that contains your data.</td>
</tr>
<tr>
<td>Data Type</td>
<td>&quot;double precision&quot;</td>
</tr>
<tr>
<td>Number of Elements</td>
<td>This corresponds to the variable that keeps track of the total number of data points acquired.</td>
</tr>
<tr>
<td>Number of Groups</td>
<td>Since you have only 1 group, all other arguments can be ignored.</td>
</tr>
</tbody>
</table>

If you store the filename in a character array, and you want to include a path name, you must use double backslashes in the path name because C interprets single backslashes as an "ESC" character. For example, if you want to store a data file called "bob.csv" in the directory "U:\joe\data" you must enter it as: "U:\joe\data\bob.csv"

If you want a professional looking GUI, try the `FileSelectPopup` function. It can be found under Library / User Interface / Pop-up Panels. It provides a nice user interface and allows the user to select a filename. Enter "save" for the "Button Label" argument; for the "Pathname" argument declare and use a pointer to character array with 500 elements that will store the file name and path. You will use the same pointer for the "Filename" argument in the `ArrayToFile` function. Insert the `FileSelectPopup` function call before the `ArrayToFile` function call!
3.8. **Complete Application: Stepper Motor and GPIB**

**GUI**

Add the following elements to the GUI from section 3.6:

- Add a binary switch and a callback function to turn the data acquisition process on or off; turning it off will also save the acquired data.
- Add a panel to display a graph of the data acquired.

**Code**

If you did not get the code to work correctly for exercise 3.6, the stepper motor, talk to your TA before continuing! Make the following changes to the code of section 3.6:

In the callback function of the data acquisition control, read the value of the control switch into a global variable so that it can be accesses from other callback functions.

- Add code to initialize and configure the GPIB and the HP34401 when the switch is turned on.
- Add code to close the GPIB drivers when the switch is turned off.
- Add code to write the acquired data from an array to a file when the switch is turned off.

The rest of the code that you will have to add will be responsible for reading the individual intensity readings from the voltmeter and to store them in array. This code should be placed inside the while-loop discussed in section 3.6, after the lines of code sending out the clock and directional signals to the stepper motor. It should accomplish the following tasks, in the order shown:

1) If the data acquisition control has been turned on AND if the stepper motor has rotated a step then execute tasks 2 and 3 below; otherwise, ignore tasks 2 and 3.

2) Measure the voltage read by the multimeter using the `hp34401a_singleMeas` function; store the value in an array and keep track of the number of points read with a simple counter variable.

3) Plot the value.

**Additional Hints and Comments Regarding Acquiring Data**

Task 1: You want to read the voltmeter only if the motor has actually moved a step. This occurs only every other cycle in the while loop because every other cycle is used to set the control line from LO to HI during which the motor does not advance; it only advances on the HI to LO transitions. A simple way to satisfy this requirement is to check that the clock signal sent out was HI, or “true.”

Task 3: The plotting is simply done for you to see that everything works. You do not have to print the values obtained. You may find that the plotting is probably easier implemented with a strip chart than
with the `PlotXY` function, though the `PlotXY` function allows for auto scaling the y-axis. You may choose either one.

**Testing the GPIB**

Some of the computers in the lab have GPIB cards installed and are connected to HP34401A multimeter. Use these machines for debugging your diffraction experiment program.

To test your diffraction program, you may want to write a program that reads a range of voltages from the Multimeter. Such a voltage range can easily be produced by a function generator sending out a signal at a very low frequency. Send a voltage out through the function generator and read it back using the HP34401A multimeter.

Please note that once the GPIB has been initialized, it sometimes just "hangs." This is particularly true if you happen to terminate your program prematurely or before a "hp34401a_close" has been issued. If this is the case, turn the multimeter temporarily off.

### 3.9. Credits

Professor Ruddick provided the idea and first write-up for this experiment. Marty Stevens and Jon Huber tested the new setup and provided valuable suggestions for improving the experiment and the write-up. In addition, Marty Stevens contributed all the figures used in this write-up.

### 3.10. Additional Information

Listed below are some general optics books with information on diffraction:


Diffraction Experiment Grading

Name: ........................................

Total Pts: ......................

Theory

<table>
<thead>
<tr>
<th>Max Pts.</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overall</td>
<td></td>
</tr>
<tr>
<td>laser wavelength as a function of max / min</td>
<td></td>
</tr>
<tr>
<td>Intensity at intercept</td>
<td></td>
</tr>
</tbody>
</table>

Questions/Analysis/Results

<table>
<thead>
<tr>
<th>Max Pts</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\chi^2$ and $\sigma_{\text{rms}}$ and number of steps, expected &amp; obtained</td>
<td></td>
</tr>
<tr>
<td>$\lambda$, Calculated</td>
<td></td>
</tr>
<tr>
<td>$\lambda$, Obtained by Fit</td>
<td></td>
</tr>
<tr>
<td>$\sigma$, Calculated</td>
<td></td>
</tr>
<tr>
<td>$\sigma$, Obtained by Fit</td>
<td></td>
</tr>
<tr>
<td>$\sigma$ and $\sigma_{\gamma}$</td>
<td></td>
</tr>
<tr>
<td>$\sigma$’s off established value / Probability</td>
<td></td>
</tr>
<tr>
<td>Max. Intensity</td>
<td></td>
</tr>
<tr>
<td>Intensity at $x = 0$</td>
<td></td>
</tr>
</tbody>
</table>

Presentation

<table>
<thead>
<tr>
<th>Max Pts</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sections labeled</td>
<td></td>
</tr>
<tr>
<td>Figures referenced, units</td>
<td></td>
</tr>
<tr>
<td>Overall</td>
<td></td>
</tr>
</tbody>
</table>

Comments: