

School of Physics and Astronomy

Physics Course 2 Project
Second Semester Report
2003

**The development of the Computer Controlled
Spectrograph to study the Fraunhofer lines of
the Solar Spectrum**

By S.J. George

Supervisor: Dr K Elliott
Tutor: Dr L Jones

Abstract

The aim of this project was to design a fully working graphical user interface for use with the Triax 180 Spectrograph and then observe the Fraunhofer lines in the solar spectrum. This was achieved by writing a program in Visual C++. The spectrograph was calibrated using a Sodium Lamp and then the Fraunhofer lines were observed.

Introduction

The University had recently purchased a Triax 180 spectrograph and had intentions of using this on the University telescope, however the spectrograph needed to have a user interface written for it; this had to be compatible with the Microsoft® Windows Operating System. This naturally led to the program being written in Microsoft® Visual C++. This will then allow the spectrograph to be used on future projects on the University telescope.

A spectrograph is a device which is used in many optics experiments to determine the composition of the source of light. A spectrograph is an instrument which presents a range of wavelengths at an aperture for detection by an array detector, in this case a Charged Coupled Device (CCD). This then allows the spectral lines of a source of light to be viewed. [1]

Spectral lines are brightly coloured bands separated by regions of darkness when a source of light is shone into the spectrograph. The spectral lines are associated with different wavelengths and depend on what the source of the light is. [2] This is commonly seen with Sodium (Na) discharge tubes where there are clear spectral lines (see figure 1) with dominate lines known as the Sodium D - Lines at 588.9950 and 589.5924 nanometres [3]. The Sodium D-lines are produced due to the emission of photons due to accelerated electrons in various orbits around the Sodium nucleus. Spectral lines are a 'chemical fingerprint' for each different element. [4] [6]



Figure 1

Spectrographs work by focussing diffracted light onto the exit aperture (connected to a CCD). In general the Czerny-Turner configuration is used (see figure 2), in this case the source light is collimated by lenses (producing a parallel light) onto the diffraction grating (in this case a reflection grating), it is then focussed on the exit aperture.

The diffraction grating is the main constituent of this arrangement, as it spatially separates the spectrum of the incident light producing the characteristic spectral lines. Each wavelength of the light is incident upon the exit aperture at different angles, rotating the diffraction grating produces different wavelengths of light being able to be viewed at the exit aperture. [2] [5]

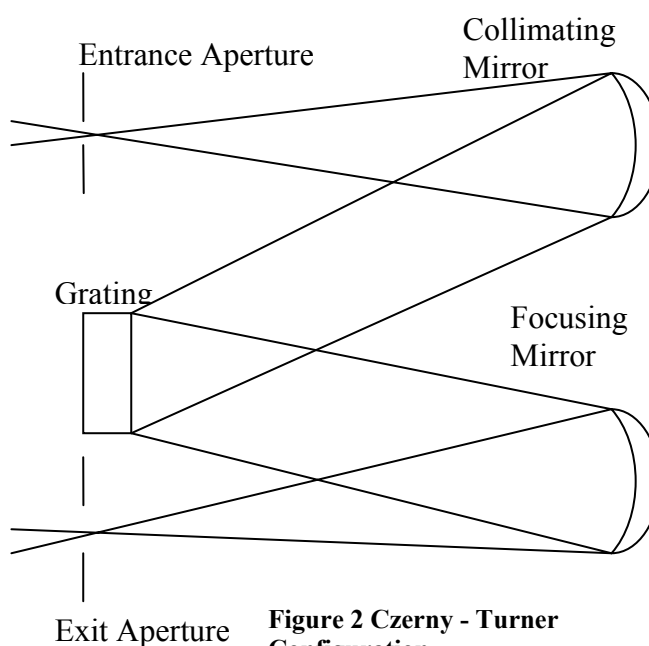


Figure 2 Czerny - Turner Configuration

The Triax 180 Spectrograph has a turret which has 3 diffraction gratings on; these grating are all of different groove densities (1200 g/mm, 300 g/mm and 150 g/mm respectively). This is a good feature as it allows the user complete control over what type of image that is taken, either a high resolution image or a strong signal. This is due to the resolution increasing with an increase in the groove density of the grating, but at the expense of spectral range and signal strength. [5]

As the spectral lines of the source of light can be found then this means that the spectrograph is very useful in astrophysical research. For example the spectrograph can be used to observe the Fraunhofer lines of the solar spectrum; this is the final aim of the project. Fraunhofer lines (figure 2) are dark lines in the solar spectrum, these where first observed by William Wollaston and he incorrectly interpreted as gaps separating the colours of the sun. [7]

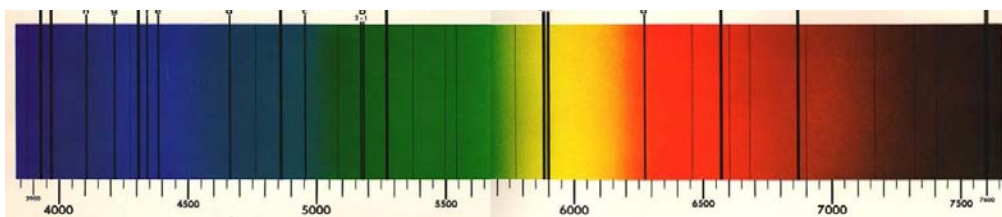


Figure 2 - the Solar Spectrum.

In 1817 Joseph Fraunhofer (whom the lines are named after) discounted Wollaston's interpretation as his own observations did not produce these gaps, he actually observed a continuous change of colour across the solar spectrum (figure 3). He then used similar techniques to observe the spectra produced by other stars and again found these dark lines in there spectra. While he was doing this he also noted that some of these lines did not appear in all stars so he concluded that these lines where not of terrestrial origin. There are many prominent lines in the solar spectrum, for example the Sodium D-lines (at 589.0nm and 589.6nm [3]) are quite strong (see appendix 1 for typical values of the Fraunhofer lines).

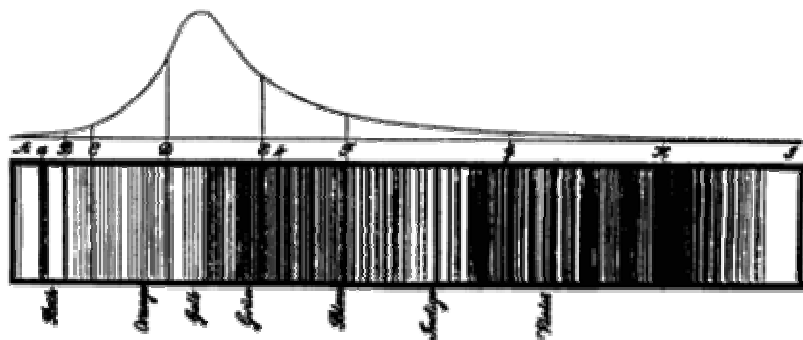


Figure 3 - Fraunhofer's solar spectrum

Theory

Diffraction Gratings

A diffraction grating is a set of parallel slits used to disperse light using Fraunhofer diffraction. Fraunhofer diffraction is when the Fresnel number is much less than 1. The Fresnel number is defined by:

$$F \equiv \frac{a^2}{\lambda R} \quad (1)$$

Where a is the radius of the aperture, λ is the wavelength and R is the distance from the aperture.

The scalar diffraction theorem can then be used to find where minima and maxima points due to diffraction are. If there are N slits of width $2a$ (centred at the origin) and separated by a distance d , then the scalar diffraction theorem is given by:

$$\psi = C \int e^{-ikxx'/R} dx' = C \sum_{n=0}^{N-1} \int_{2nd-a}^{2nd+a} e^{-ikxx'/R} dx' \quad (2)$$

Where k is the wave number and x is the distance from the origin (perpendicular to the slits).

For Fraunhofer diffraction, the slit width is D , thus giving:

$$\psi = C \int_{-D/2}^{D/2} e^{-ikxx'/R} dx' \quad (3)$$

If this is integrated then this gives:

$$\psi = CD \frac{\sin \beta}{\beta} \quad (4)$$

Where C is a constant of integration and $\beta = \frac{kD}{2R} x$.

By squaring the magnitude of the wave function we get the intensity, I , which is:

$$I = D^2 C^2 \left(\frac{\sin \beta}{\beta} \right)^2 = \left(\frac{\sin \beta}{\beta} \right)^2 I_o \quad (5)$$

The maxima and minima are then found by differentiating equation 5 and then setting the derivative to zero. So:

$$\sin \beta (\sin \beta - \beta \cos \beta) = 0 \quad (6)$$

So minima occur at: $\sin \beta = 0$, or $\beta = \pi$, or $\beta = \pm 2\pi$, or $\beta = \pm 3\pi$

So:
$$\sin \psi = \frac{\lambda}{D}, \frac{2\lambda}{D}, \frac{3\lambda}{D}, \dots$$

Maxima occur at:

$$\sin \beta - \beta \sin \beta = 0 \quad (7)$$

This allows the maxima and minima points of diffraction to be calculated and the diffraction grating can be set up to these points. [8] [9]

This will allow images to be taken, but diffraction gratings also cause the source image to be angularly magnified, so have to account for this angular dispersion which is defined as, the ratio of the angular separation between two wavelengths (in radians) to the differential separation between the two wavelengths in nanometres. The ratio is given by:

$$\frac{d\beta}{d\lambda} = \frac{(kn \cdot 10^{-6})}{\cos \beta} \quad (8)$$

This leads onto the general equation for diffraction gratings:

$$\sin \alpha + \sin \beta = 10^{-6} kn\lambda \quad (9)$$

Serial Communication

To communicate between the spectrograph and the computer there are certain criteria that have to be met. The major criterion which has to be met before anything else is done is the protocol (language) which is going to be used. For the Triax 180 Spectrograph it has two modes either RS-232 (which shall be used) or IEEE488. This is so important because both the port on the computer and the spectrograph need to know when each piece of data has finished being transmitted or received.

For RS-232 this is handled by the device sending an extra bit (a binary value of 0 or 1) of information once the read buffer (RX) or the write buffer (TX) is empty, the extra data bit is read. Once this is done other items such as parity checks can be implemented, parity checks are used to check the consistent of the data (if needed).

The next part of establishing a protocol is to set what the speed of the communication is to be, this can vary from 2400 bits per second up to 115000 bits per second. This is used to ensure that the RX buffer is not filled to capacity; this is due to the hardware not being able to read from this buffer as fast as the computer is sending the data. If this is not setup then it can cause the system to crash or data to be lost.

Once this is done it means the protocol is established and now the timeouts need to be set. These need to be set otherwise the computer could wait for a signal for a long time when the spectrograph could have possibly crashed, when the timeout is reached it means the computer will no longer be waiting for the signal from the spectrograph. Once this is set then communications to the spectrograph can begin.

Apparatus

The apparatus for the experiment can be split into two parts; the development of the computer program and the taking of the measurements. For the computer program development, the apparatus used was: a computer with Microsoft ® Visual C++ installed, the Triax 180 spectrograph, serial port communication cables, a light source and a Vernier travelling microscope.

For the measurements part of the project the apparatus used was; a computer which has a serial port so that the spectrograph can be connected, a computer with the appropriate CCD software installed (Maxim DL¹), a CCD (in this case a Finger Lakes Instrumentation CCD was used, model number: CM9-OE), a Sodium Lamp, a telescope, fibre optic cables, a stand for the CCD and a box to protect the whole arrangement so that no background light gets into arrangement.

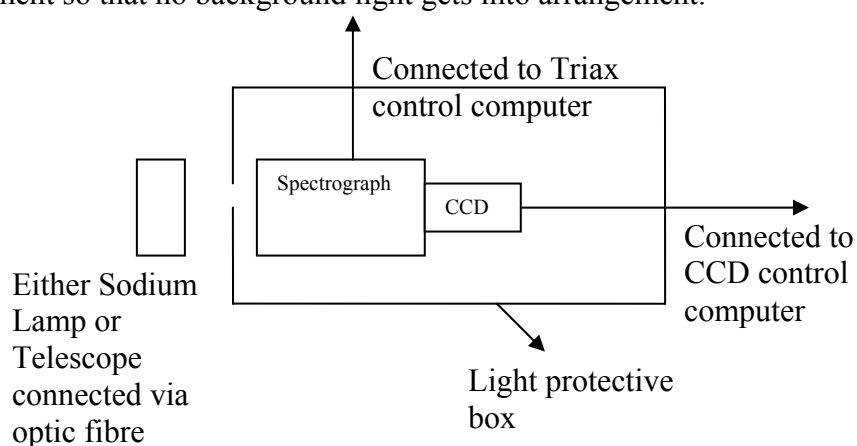


Figure 4 The setup of the spectrograph for measurements of the Sodium and Fraunhofer Lines
Triax 180 specifications²: (for 1200 groves/ mm grating) [10]

Focal Length	0.19 m
Aperture	f/3.9
Resolution	0.3 nm
Dispersion	3.53 nm/mm
Accuracy	+/- 0.3 nm
Step Size	0.06 nm (nominal)
Stray Light	$1 \times 10^{-5} < 20$ nm from 632.8 nm laser line
Optical Layout	Corrected Cross Czerny Turner
Magnification	1.00
Wavelength Range	185 nm to far Infrared (with appropriate gratings)
Mechanical Range	0-1400 nm
Flat Image Field	30 mm x 12 mm
Slits	Motorised: 0-2 mm in 2 μ m steps or Motorised: 0-7 mm in 6.35 μ m steps or Interchangeable fixed slits
Dimensions	230 (W) x 260 (L) x 320 (H) mm



Figure 5 Triax 180

¹ Maxim DL is an image processing software (see cyanogen.com)

² Specifications from the Manual for the Triax 180 (see <http://www.jyhoriba.co.uk/mono/t180190.htm>)

Experimental Procedure

Program Development

The first problem that needed to be addressed in the development of the program would be what mode of communication to use. The RS-232 mode was chosen over the IEE488 mode (see theory section), this will allow easy communication via the com port of the computer. The boot sequence then needed to be wrote, this will allow communication of the functions of the spectrograph (see appendix 2 for C++ code).

Boot Sequence (see figure 7):

- 1) Setup communication to the spectrograph via RS-232 mode.
- 2) Send "<space>": the response from the spectrograph will be "*" if this is the first time power up or after a re-boot.
OR
WHERE AM I command "<space>", true if the spectrometer controller has already been started; in this case the response will be either "B" (for Boot) or "F" (for Main) depending on what the previous state of the spectrometer controller was.
- 3) Then have to wait 0.5 seconds before sending any more information.
- 4) Flush the input buffer
- 5) Send the decimal value "<247>"
- 6) Send "<=>"
- 7) Send "O2000<Null>", this will then transfer the control from the BOOT to the MAIN program. "<MAIN>" is sent and then have to wait 0.5 seconds.
- 8) Initialize the mono and get initial settings from the user.
- 9) Main Program Code

The full boot sequence is shown in figure 7 which is the flow chart of operations that need to be followed to get the spectrograph to boot-up.

Once the boot sequence was executed then this would allow the main program to be initiated. The boot sequence does not need any user input so all that is required is for the user to click an activate button (figure 6).

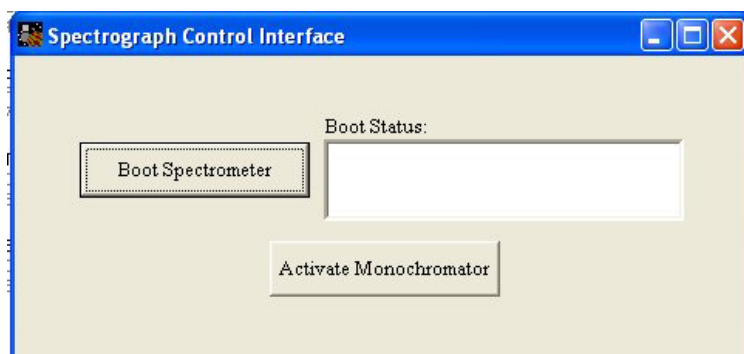


Figure 6 - Boot Screen

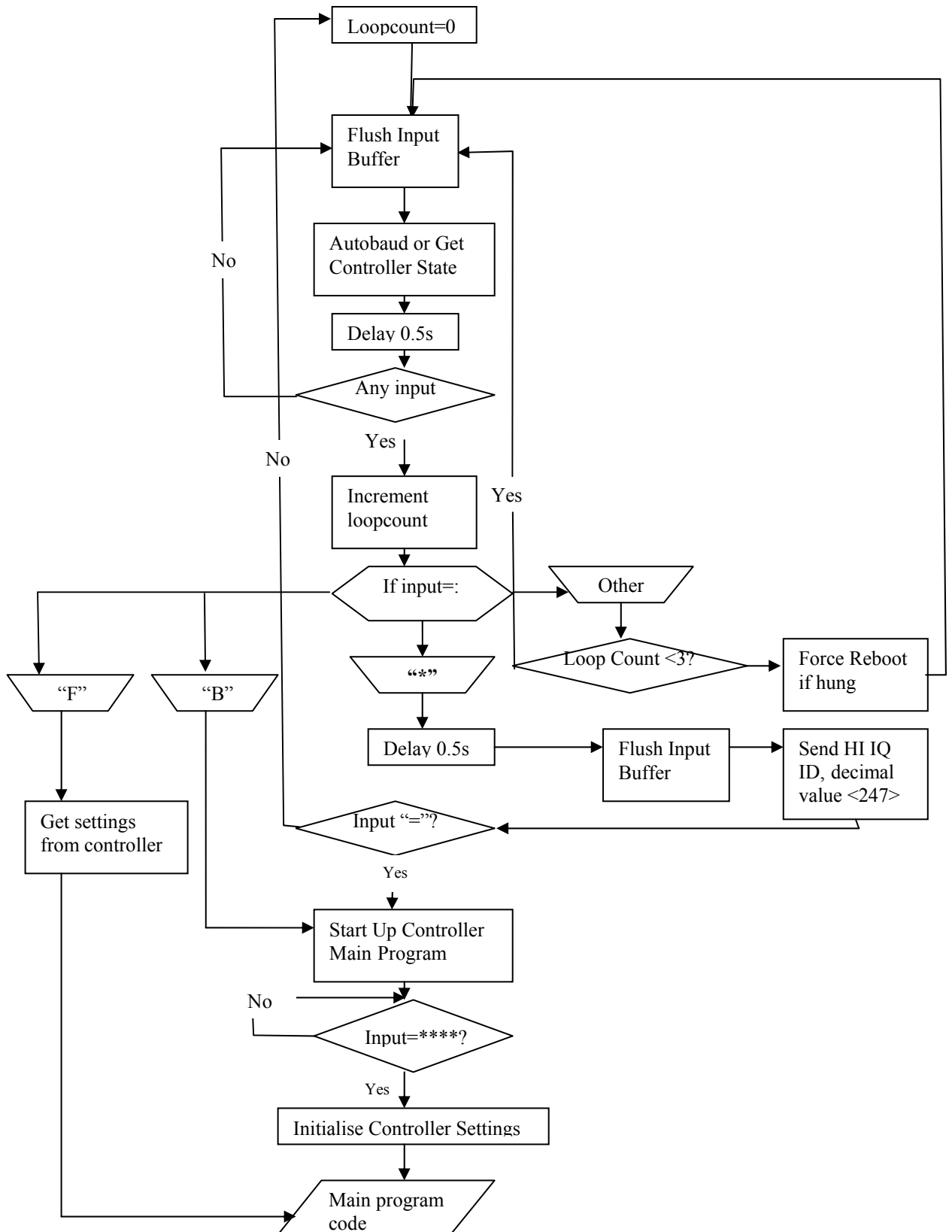


Figure 7 - The spectrograph boot-up sequence

Main Program Code

Since Visual C++ was being used then this allowed a work space to be setup; this allows the design of the program to be easily designed and will allow the functions that are wrote in the main part of the program (see appendix 2) to be easily activated by clicking of buttons.

The functions that the Triax used had to be then placed into the program so when a command was sent from the interface then the program would send the correct information to the spectrograph (see appendix 3 for listing); these functions would later be called into by the interface when they were used.

The interface itself then had to be created for the main program; this had to allow text to be input so that it could be used to easily set the spectrograph to the required wavelength. Firstly the layout of the graphical user interface was decided (see figure 8), including what type of input box would be used for each element of the program. For the choosing of the grating to use it was thought best to use radio boxes which are all grouped together, which will only allow one to be chosen at a time. The input of distances, i.e. wavelength and slit width was done by a text box which can

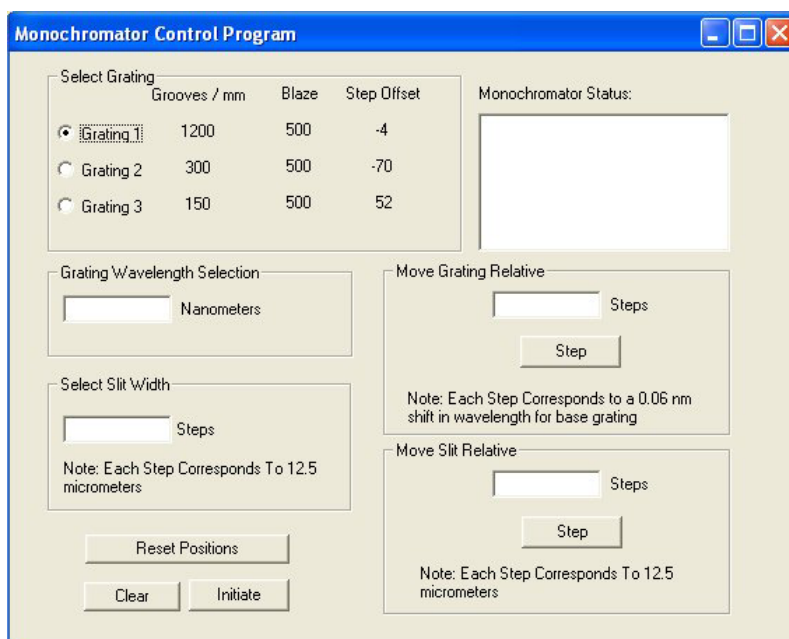


Figure 8 - The Monochromator Control Interface

have its value stored into the memory of the computer and sent to the spectrograph (the spectrograph is programmed to understand wavelengths in nanometres and slit width in steps, 1 step = 12.5 micrometres). Once all the buttons had been designed the clear function had to be setup so that all the information the user has typed in the program can be reset to a default value (see appendix 4). The only part of the interface left to setup is the status box which displays if the spectrograph has been set and what to, this is a useful feature as it is possible to see what the setting of the spectrograph are without having to remember the values.

Measurements

Once the program was fully developed and all the debugging of the program was completed, the spectrograph was then able to be used for measurements. The spectrograph was setup (see apparatus) inside the light protective box, this is important as the background light would interfere with the experiment as there would be spectral lines observed from the lighting.

Once inside the box, the CCD camera had to be placed in position so that it was perfectly level with the Spectrograph and that the CCD was at the focal point of the

spectrograph; this was very hard to accomplish and would be a cause for errors in the results obtained. The CCD was connected to the PC with Maxim DL on and then was cooled down to an approximate temperature of -20°C . The protective box was sealed up using tape so that no light could enter through the corners of the box and in points where the cables to the spectrograph and CCD camera go into the box. There was only one place in which light could enter the spectrograph and that was via the entrance aperture of the spectrograph.

The spectrograph had to be first setup with the Sodium lamp so that the calibration of the images could take place; this would allow the number of pixels per nanometre to be calculated and then allow us to calculate the position of certain dark lines in the solar spectrum. The Sodium lamp was chosen as the distance between the two prominent spectral lines of Sodium is well known; they are at: 588.9950nm and 589.5924nm [3], which is a difference of only 6 \AA .

Once the images of the Sodium D-lines were taken they could then be analysed in Spectra PC; this is a program which allows a cross section of the image to be taken and allows the intensity of the light (the number of photons) to be plotted against distance across the image (in pixels). This allowed accurate results to be taken as the peak intensities of the lines could be found and so the calibration could be achieved.

Now that the calibration of the CCD had been done it was then possible to move on to the main aim of the experiment which was to take an image of the solar spectrum and find the Fraunhofer lines. The equipment was in the same configuration as for the images of the Sodium lamp except for instead of having a Sodium lamp as the light source the Sun was the source. The image of the Sun onto the slits of the spectrograph was achieved by using a refracting telescope which was aimed at the Sun with optic fibres placed at the focal point of the telescope. The optic fibres were then placed so that they were shining on the slits of the spectrograph. Once protected from the background light allowed the image of the solar spectrum to be taken, this required a longer exposure of the CCD since the light through the optic fibres was not as intense as the light produced from the Sodium lamp.

Again the images were analyzed using Spectra PC which gave a plot of the Spectrum and so the Fraunhofer lines could be easily detected. Again the Sodium lines in the spectra were found and seen if they were in the expected position, if they were (within errors) then this would mean that the Spectrograph was operating as expected.

Results

From the design aspect of the computer program the result was that there was a fully functional program that seemed to operate the spectrograph perfectly, without any problems. However this would have been useless if the spectrograph did not produce any accurate images. The results found from the measurements part of the project are below.

Firstly the sodium lamp was used to check that the spectrograph was setup correctly before the images of the solar spectrum was taken.

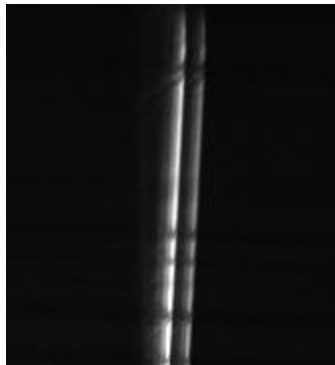


Figure 9 Na D-Lines at 590nm

Figure 9 shows the two Sodium lines, this was with the spectrograph set at a wavelength of 590nm. The slit width was approximately $25\mu\text{m}$ and the exposure time for the camera was approximately 0.1 seconds. The grating used was 1200 grooves/mm.

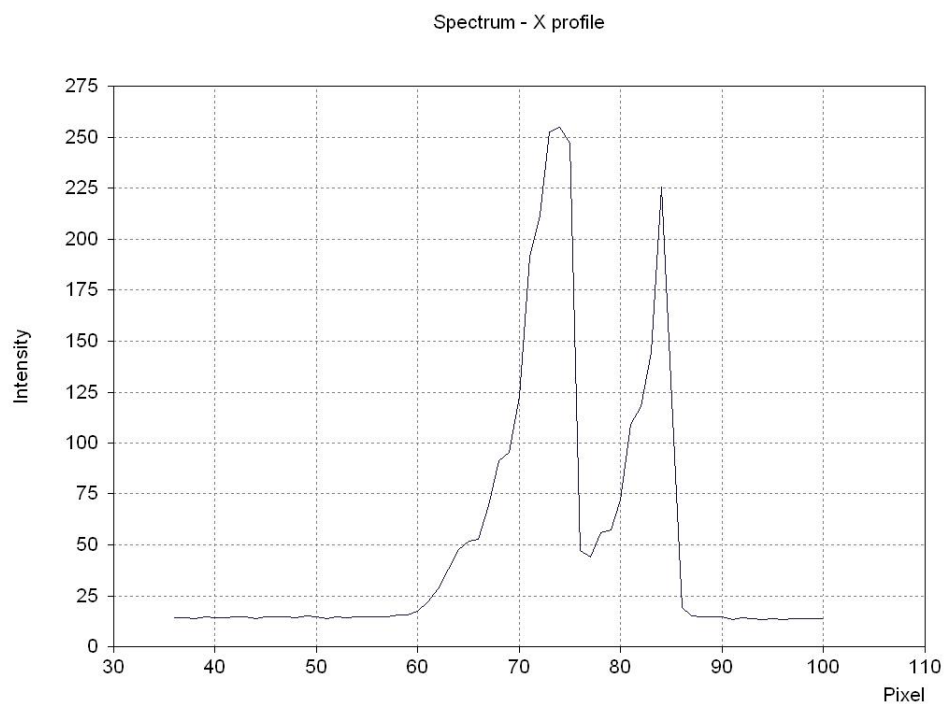


Figure 10 - Intensity against Pixel (distance) plot of Sodium D-lines

Figure 10 shows a plot of the intensity of the photons at each point across the spectrum; the two peaks are at values of 73 pixels and 82 pixels (produced by using Spectra PC). So the two peaks are 9 pixels apart.

The spectrograph was then setup so that the solar spectrum could be taken (see experimental procedure):

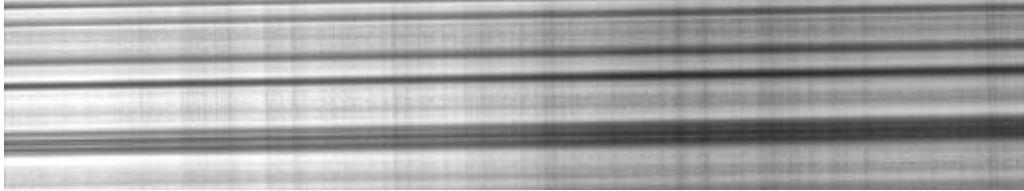


Figure 11 - The Solar Spectrum taken at 400nm

Figure 11 shows the solar spectrum, this was with the spectrograph set at a wavelength of 400nm. The slit width was approximately 25 μ m and the exposure time for the camera was approximately 60 seconds. In this case the grating that was used to produce this image was the 300 grooves /mm.

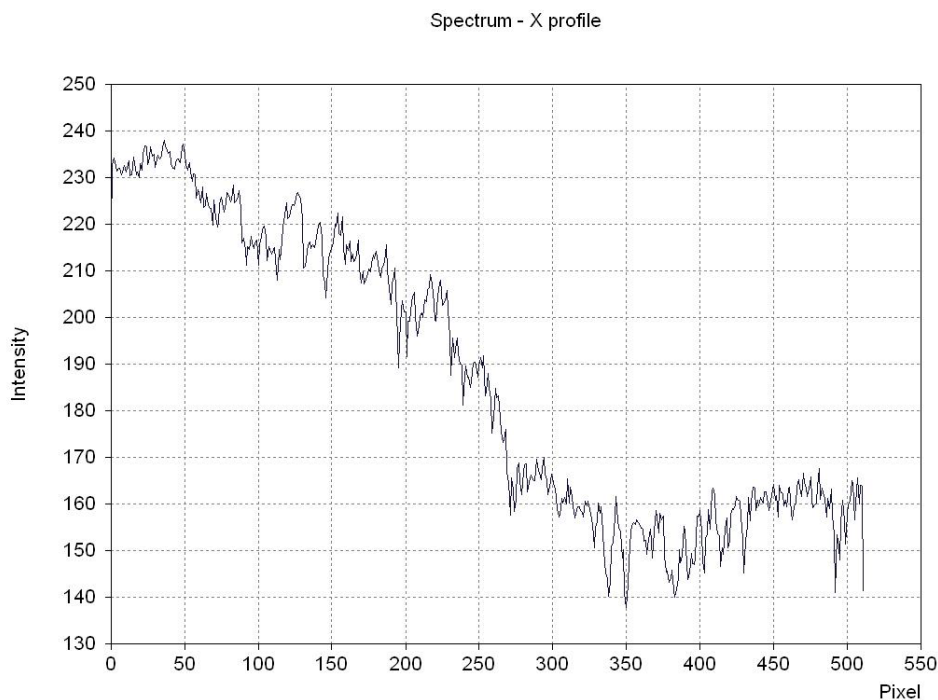


Figure 12 - Intensity against Pixel (distance) plot of the Solar Spectrum

Figure 12 shows a plot of the intensity of the photons at each point across the spectrum; there are many troughs where the Fraunhofer lines are, the two most obvious are what are believed to be the Sodium lines, these occur at 338 and 350 pixels. (produced by using Spectra PC). So the two Sodium peaks are 12 pixels apart.

Errors

In this experiment it is quite hard to get a numerically value for the errors in the measurements this is due to the equipment being used. The major problem was that the spectrograph did not have a mount for the CCD camera being used, so not only was it difficult to get the camera to the focal point of the Spectrograph, which caused the image not to be of the greatest resolution, but also it meant that it was difficult to get the camera into a position where it was in the central point (horizontally and vertically) of the exit aperture. This definitely caused there to be errors in the measurements. With the images being slightly rotated and since the resolution was slightly lost then the peaks will not be as clear as possible, so it was believed that the peaks associated with the spectral lines could easily be at a position of ± 1 pixel.

Discussion and Interpretation

It was found that in the case of the Sodium Lamp that the two peaks were $9 \text{ pixels} \pm 2 \text{ pixels}$ apart, so this meant that the calibration of how many pixels per nanometre could be calculated:

Since the two Sodium lines in question are 0.6 nm apart and the pixel distance apart is known to be $9 \text{ pixels} \pm 2 \text{ pixels}$ then this leads to there being $0.067 \text{ nm per pixel} \pm 0.02 \text{ nm per pixel}$.

This can then be compared with the values obtained for the solar spectrum, in this case the two Sodium lines were $12 \text{ pixels} \pm 2 \text{ pixels}$ and within the errors this can be found to be similar to what was obtained with the calibration of the Sodium lines, in this case there is $0.050 \text{ nm per pixel} \pm 0.02 \text{ nm per pixel}$. So within errors the two values agree, this shows that the Spectrograph was calibrated properly and that the image obtained of the solar spectrum is accurate and that the Fraunhofer lines on the image are quite accurate.

It was expected that the two images may not be exactly the same as there was considerable problems in getting a good image of the Sun as the only method to get a good image was via the optic fibres and this led to light leakage into the experiment, however within the errors of the measurements the two values agree.

There were horizontal lines produced on the images (see figure 10), this seems to be due to the edge of the slits of the spectroscope not being flat, possibly some dust on the edge of the slits causing this pattern in the data; this does not interfere with the results taken.

The solar spectrum with the Fraunhofer lines and possibly some atmospheric absorption lines is quite clear, this may lead to these atmospheric lines being identified and then being able to discover what atomic transition is responsible for these spectral lines. Since the solar spectrum was taken quite accurately this may lead the spectrograph being fitted on to the University telescope and being used to take spectra of other stars in the sky, again to find the chemical composition of the star via these dark absorption lines in the spectra of the stars.

Conclusion

From the results obtained it would be possible to say that the computer interface that has been written for the spectrograph seemed to work well. The results we obtained were consistent with what we believed would occur, the two Sodium lines were observed at the correct wavelength, when the spectrograph was set to there wavelength then they appeared in the images which indicates that the spectrograph was moving to the correct position. Also the solar spectrum image produced was what was expected and since the sodium lines where in the spectrum then this allowed the two to be compared. The errors associated with the measurements taken can be reduced if this was repeated, this could firstly and most importantly done by building a proper mount for the CCD camera based on the exit aperture of the Triax 180; this would allow the images to be resolved properly as the CCD would be fixed at the focal point, it would also reduce the problems with the images being rotated. Another way in which errors could be reduced would be by having a better light protective case for the arrangement; obviously if this was being used on the University telescope at night then this would not be strictly necessary.

In conclusion the experiment was a successful one; the aim of writing a working computer controlled interface was accomplished. This was verified by observing the solar spectrum; this lead to the experimental evidence needed to show that the computer controlled interface had in fact worked as it was designed to.

Acknowledgements

As always there are a few people whom I would like to thank for the help during this project. The below are the people I would like to thank:

Mr Jordan Stewart (Laboratory Partner)

Mr Colin Bull (Laboratory Partner)

Dr Ken Elliott (Supervisor)

Mr Barry Weston (Laboratory Technician)

Mr Wayne Twigger (Software Engineer for supplier of Triax 180) and the rest of the team at Triax who helped us with our program development.

References

1. Eugene Hecht, "Optics", 4th Edition (San Francisco, USA, Addison Wesley, 2002)
2. Paul A. Tipler, "Physics" 4th Edition (USA, W.H. Freeman, 1999)
3. MIT Wavelength Tables (MIT Publishing, USA, 1969)
4. Robert Eisberg and Robert Resnick, "Quantum Physics of Atoms, Molecules, Solids, Nuclei, and Particles", 2nd Edition (USA, Hamilton Printing Company, 1985)
5. Manuals for the Triax 180
6. R.J. Tayler, "The Stars: their structure and evolution", 2nd Edition (Cambridge UK, University Press (Cambridge), 1994)
7. Christopher Bishop, "Astrophysics" (London UK, John Murray Publishers, 2000)
8. <http://scienceworld.wolfram.com>
9. Mary L. Boas "Mathematical Methods in the Physical Sciences" 2nd Edition (USA, John Wiley and Sons, 1983)
10. <http://www.jyhoriba.co.uk/mono/t180190.htm>

- Figure 1: http://archive.ncsa.uiuc.edu/Cyberia/Bima/Images/NaLines_lg.jpg
Figure 2: <http://www.harmsy.freeuk.com/images/spectrum.jpeg>
Figure 3: <http://home.achilles.net/~jtalbot/spectra/Fraunhofer.html>
Figure 5: <http://www.jyhoriba.co.uk/mono/t180190.htm>

Appendices

Appendix 1

Typical values for the Fraunhofer lines in the solar spectrum are:

Lines	Due To ...	Wavelengths (nm)
A - (band)	O ₂	759.4 - 762.1
B - (band)	O ₂	686.7 - 688.4
C	H	656.3
a - (band)	O ₂	627.6 - 628.7
D - 1, 2	Na	589.6 & 589.0
E	Fe	527.0
B - 1, 2	Mg	518.4 & 517.3
C	Fe	495.8
F	H	486.1
D	Fe	466.8
E	Fe	438.4
F	H	434.0
G	Fe & Ca	430.8
G	Ca	422.7
h	H	410.2
H	Ca	396.8

Appendix 2

Boot up Sequence - Serial Interface.

```
// SpecControl.cpp: implementation of the CSpecControl class.
//////////////////////////////////////////////////////////////////
#include "stdafx.h"
#include "SpecControl.h"
#include "TriaxAPI.h"
#include "SerialInterface.h"
//////////////////////////////////////////////////////////////////
// Construction/Destruction
CSpecControl::CSpecControl()
{
}
CSpecControl::~CSpecControl()
{
}
CSerialInterface * Port = new CSerialInterface; //Pointer to Serial Interface
BOOL b_TriedReBoot = false; //Boolean if Reboot been tried
int x = 0;
//Reboots Spectrometer back to begging of start up seq..
void CSpecControl::Reboot()
{
//Sends incomplete command then calls 247 & 222 to reboot spectrometer
if(!Port->SendChar("G")) {Port->ClosePort(); AfxMessageBox("ReBoot Failed"}
char Buf[1];
Buf[0] = static_cast<char>((BYTE)247); //Converts BYTE into Char, i.e 247 into it's //character form e.g. A == 65, B == 66
if(!Port->Write(&Buf[0],1)) {Port->ClosePort(); AfxMessageBox("ReBoot Failed");}
Buf[0] = static_cast<char>((BYTE)222);
if(!Port->Write(&Buf[0],1)) {Port->ClosePort(); AfxMessageBox("ReBoot Failed");}
Port->ResetPort();
Boot2(); } //Boot Sequence of Spectrometer
BOOL CSpecControl::Boot2()
{
char rBuffer[50] = ""; //IO buffers
char cmpBuf[30];
int i = 0; //Counters
```

```

int x = 0;
if(!b_TriedReBoot) //Trivial, if already attempted since port already open...
{
    if(!Port->OpenPort()) {Cleanup(); return false;}
    if(!Port->ConfigPort()) {Cleanup(); return false;}
    if(!Port->SetTimeOuts()) {Cleanup(); return false;}
    Port->ClearRTS();
    Port->SetDTR();
    Port->ResetPort();
}

if(!Port->SendChar(UTIL_WHERE_AM_I)) {Cleanup(); return false;}
Sleep(500);
Port->Read(rBuffer,sizeof(rBuffer)); //Sends *
switch(rBuffer[0]) //Switch statement test results, from UTIL_WHERE_AM_I
{
    case '*':
        {
            Port->ResetPort();
            Sleep(500);
            Port->RXClear(); //Clears other stuff like Press[enter] sent by the spectrometer
            char Buf;
            Buf = static_cast<char>((BYTE)247); //buf becomes hexadecimal F7 / decimal 247
            if(!Port->SendChar(&Buf)) {Cleanup(); return false;} //Hexadecimal as F7 on port
            Sleep(500);
            Port->Read(cmpBuf,sizeof(cmpBuf)); //Reads response of spectrometer
            if(StrCompare(cmpBuf[0],"=")) //Compares cmpBuf, with "="
            if(!StartCtrl()){Cleanup(); return false;} //Starts Control program
            else {return true;}
        }
    else
    {
        Cleanup(); //Deletes port etc....
        return false;
    }
} // "*"
case 'B':
{
    Sleep(500);
    if(!StartCtrl()){Cleanup(); return false;}
    else {Port->ResetPort(); return true;}
} // "B"
case 'F':
{
    Sleep(500);
    Port->ResetPort();
    return true;
} // "F"
default:
{x = MessageBox(GetDesktopWindow(),"Is Spectrometer Turned On","Error:", MB_YESNO + MB_ICONQUESTION);
if(x == 6)
{
    if(i != 3) {b_TriedReBoot = true; i++; Reboot();}
    else {return false;}
}
else
{
    AfxMessageBox("Please turn Spectrometer on and try again");
    Cleanup();
    return false;
} //default
} //switch
Cleanup();
return false; //should never get here
}
void CSpecControl::Cleanup()
{
    Port->ClosePort();
}
//Converts strcmp!!! Into a boolean output since not interested which string has greater value
BOOL CSpecControl::StrCompare(CString comp, const char* rBuf)
{
    switch(strcmp(comp,rBuf))
    {
    case 1:

```

```

        return false;
    case 0:
        return true;
    case -1:
        return false;
    }
    return false;
}

//Hands control from Boot Program to main Program In Spectrometer "O2000<NULL>" command
BOOL CSpecControl::StartCtrl()
{
    char cmpBuf[30];
    Port->ResetPort();
    if(!Port->Write(UTIL_START_MAIN_PROGRAM,6)) {Cleanup(); return false;}
    else
    {
        Sleep(1000);
        if(Port->Read(cmpBuf,1) == 0) {Cleanup(); return false;}
        else
        {
            if(StrCompare(cmpBuf[0],"*") {return true;}
        }
    }
    return false;
}

//Initialises, Motor a starts slef calibration
BOOL CSpecControl::MotorInit()
{
    char rBuf;
    if(!Port->SendChar(MOTOR_INIT)){return false;}
    while(Port->Read(&rBuf,1) == 0)
    {
        Sleep(500);
    }
    if(StrCompare(rBuf,"o")){return true;}
    else {return false;}
}

//Steps slit x amount of slits...
BOOL CSpecControl::MoveSlitRel(CString Mono, CString Slit, CString Steps)
{
    CString sBuf = SLIT_MOVE_RELATIVE;
    char send[30];
    char rBuf;
    sBuf += Mono + ",";
    sBuf += Slit + ",";
    sBuf += Steps;
    sBuf +=static_cast<char>(13);
    strcpy(send,sBuf);
    if(!Port->Write(&send,10)){return false;}
    while(Port->Read(&rBuf,1) == 0)
    {
        Sleep(500);
    }
    if(StrCompare(rBuf,"o"))
    {
        while(!MotorStatus(0)){} //Wait for slit to stop moving
    }
    else {return false;}
    return true;
}

//Moves to particular Wavelength
BOOL CSpecControl::SetWavelength(CString Mono, CString Wavelength)
{
    CString sBuf = MW_X_MOVE_WORKING_ABS_POSITION;
    char send[30];
    char rBuf;
    int Bytes = 0;
    sBuf += "," + Mono + ",";
    sBuf += Wavelength;
    sBuf +=static_cast<char>(13);
    Bytes = sBuf.GetLength();
    strcpy(send,sBuf);
    if(!Port->Write(&send,Bytes)){return false;}
    Port->RXClear();
    while(Port->Read(&rBuf,1) == 0)

```

```

    {
        Sleep(100);
    }
    if(StrCompare(rBuf,"o"))
    {
while(!MotorStatus(0)){ //Test to make sure grating stopped moving
    }
    else {return false;}
return true;
}
}
//Sets Grating, 1 2 or 3 BOOL CSpecControl::SetGrating(CString Mono, CString DeviceType, CString DeviceNumber, CString
NewPostion)
{
    CString sBuf= MW_X_SET_INDEX_DEVICE_POS;
    char send[30];
    char rBuf;
    int Bytes = 0;
    sBuf += " " + Mono + " ";
    sBuf += DeviceType + " ";
    sBuf += DeviceNumber + " ";
    sBuf += NewPostion;
    sBuf +=static_cast<char>(13);
    Bytes = sBuf.GetLength();
    strcpy(send,sBuf);
    if(!Port->Write(&send,Bytes)){return false;}
    Port->RXClear();
    while(Port->Read(&rBuf,1) == 0)
    {
        {
            Sleep(2000);
        }
        if(StrCompare(rBuf,"o"))
        {
            while(!MotorStatus(0)){ //tests grating set finished
            }
        }
        else {return false;}
    }
    return true;
}
//Since only one motor can be moved at a time we need to test to see if the previous call has finished, before the next command
executes
BOOL CSpecControl::MotorStatus(int Test)
{
    CString sBuf= ACC_BUSY_CHECK;
    char send[30];
    char rBuf[3];
    int Bytes = 0;
    int Read = 0;
    Bytes = sBuf.GetLength();
    strcpy(send,sBuf);
    Port->RXClear();
    if(!Port->Write(&send,Bytes)){return false;}
    else
    {
        {
            Sleep(300);
            Read = Port->Read(&rBuf,3);
            if(Test == 1 && Read == 0) {return false;}
        }
        if(StrCompare(rBuf[1],"z"){return true;}
        else{return false;}
    }
}
//Returns position of slit....
CString CSpecControl::ReadSlitPos(CString Mono, CString Slit)
{
    CString sBuf= SLIT_READ_POSITION;
    CString pBuf= static_cast<char>(13);
    CString Pos;
    char send[30];
    char rBuf[20];
    int i=0;
    sBuf += Mono + " ";
    sBuf += Slit;
    sBuf +=static_cast<char>(13);
    strcpy(send,sBuf);
    if(!Port->Write(&send,10)){return "-1";}
    while(Port->Read(&rBuf,5) == 0)
    {

```

```

        Sleep(500);
    }
    if(StrCompare(rBuf[0],"o"))
    {
        while( i != 5)
        {
            if(!StrCompare(rBuf[i],pBuf))
            {
                if(!StrCompare(rBuf[i],"b")){Pos += rBuf[i];}
            }
        }
        return Pos;
    }
    else {return "-1";}
}
//Closes slit!! from current size
BOOL CSpecControl::CloseSlit(CString Mono, CString Slit, CString Steps)
{
    CString sBuf = SLIT_MOVE_RELATIVE;
    char send[30];
    char rBuf;
    sBuf += Mono + ",";
    sBuf += Slit + "-";
    sBuf += Steps;
    sBuf +=static_cast<char>(13);
    strcpy(send,sBuf);
    if(!Port->Write(&send,10)){return false;}
    while(Port->Read(&rBuf,1) == 0)
    {
        Sleep(500);
    }
    if(StrCompare(rBuf,"o"))
    {
        while(!MotorStatus(0)){
        }
    }
    else {return false;}
    return true;
}
//Steps grating Steps number of Steps....
BOOL CSpecControl::MoveMotorRel(CString Mono, CString Steps)
{
    CString sBuf = MOTOR_MOVE_RELATIVE;
    char send[30];
    char rBuf;
    int Bytes = 0;
    Bytes = sBuf.GetLength();
    sBuf += Mono + ",";
    sBuf += Steps;
    sBuf +=static_cast<char>(13);
    Bytes = sBuf.GetLength();
    strcpy(send,sBuf);
    if(!Port->Write(&send,Bytes)){return false;}
    while(Port->Read(&rBuf,1) == 0)
    {
        Sleep(500);
    }
    if(StrCompare(rBuf,"o"))
    {
        return true;
    }
    else {return false;}
}
}

```

```

// SerialInterface.cpp: implementation of the CSerialInterface class.
///////////////////////////////////////////////////////////////////
#include "stdafx.h"
#include "SerialInterface.h"
///////////////////////////////////////////////////////////////////
// Construction/Destruction
CSerialInterface::CSerialInterface()
{
}
CSerialInterface::~CSerialInterface()
{
}
//Opens COM1
BOOL CSerialInterface::OpenPort()
{
    portname = "\\.\";
    portname = portname + "com1";
    h_Port = CreateFile(portname, //In this case COM1
        GENERIC_READ | GENERIC_WRITE, //Allow access to port
        0,
        0,
        OPEN_EXISTING, //We don't want to create a new port
        0, //Not Overlapped
        0);
    if(h_Port == INVALID_HANDLE_VALUE) {return false;}
    else {return true;}
}
BOOL CSerialInterface::ConfigPort()
{
    if(!_PReady = GetCommState(h_Port, &_config) == 0 )
    {
        MessageBox(NULL,"Comm Port State Error","Error",MB_OK+MB_ICONERROR);
        CloseHandle(h_Port);
        return false;
    }
    _config.BaudRate = CBR_19200; //Baud 19200 com speed
    _config.ByteSize = 8; //Byte Packet size
    _config.Parity = NOPARITY ; //No Parity bit
    _config.StopBits = ONESTOPBIT;//Stop bit with communication
    _config.fBinary=TRUE;
    _config.fDsrSensitivity=false;
    _config.fParity= false;
    _config.fOutX=false;
    _config.fInX=false;
    _config.fNull=false;
    _config.fAbortOnError=TRUE;
    _config.fOutxCtsFlow=FALSE;
    _config.fOutxDsrFlow=false;
    _config.fDtrControl=DTR_CONTROL_DISABLE;
    _config.fDsrSensitivity=false;
    _config.fRtsControl=RTS_CONTROL_DISABLE;
    _config.fOutxCtsFlow=false;
    _config.fOutxDsrFlow=false;
    _PReady = SetCommState(h_Port, &_config);
    if(!_PReady == 0)
    {
        MessageBox(NULL,"Can't Configure Port","Error",MB_OK+MB_ICONERROR);
        CloseHandle(h_Port);
        return false;
    }
    return true;
}
//Sets comm timeouts to prevent HANGS if no response,
BOOL CSerialInterface::SetTimeOuts()
{
    if(!_PReady = GetCommTimeouts (h_Port, &_Timeout))==0) {return false;}
    _Timeout.ReadIntervalTimeout = -1; //infinte, since don't no how long //spectrometer takes to init... Handled by
    //CSpecControl class
    _Timeout.ReadTotalTimeoutConstant = 0;
    _Timeout.ReadTotalTimeoutMultiplier = 0;
    _Timeout.WriteTotalTimeoutMultiplier = 0;
    _Timeout.WriteTotalTimeoutConstant = 500; //Enough time to write anything
    _PReady = SetCommTimeouts (h_Port, &_Timeout);
    if(!_PReady == 0)
    {

```

```

        MessageBox(NULL,"Could Not Set Timeouts", "Error", MB_OK+MB_ICONERROR);
        CloseHandle(h_Port);
        return false;
    }
    return true;
} //Write from pBuffer to the Com port, (Any Number Of Bytes)
BOOL CSerialInterface::Write(const void* buf, DWORD dwBytes)
{
    DWORD dwBytesWritten = 0;
    if (!WriteFile(h_Port, buf, dwBytes, &dwBytesWritten, NULL))
    {
        MessageBox(NULL, "Can't Write to Port", "Error", MB_OK+MB_ICONERROR);
        return false;
    }
    return true;
} //Reads to pBuffer, from the com port, (Any Number of bytes)
DWORD CSerialInterface::Read(void *pbuf, DWORD dwBytes)
{
    DWORD dwBytesWritten = 0;
    if (!ReadFile(h_Port, pbuf, dwBytes, &dwBytesWritten, NULL))
    {
        MessageBox(NULL, "Can't Read Port", "Error", MB_OK+MB_ICONERROR);
        return false;
    }
    return dwBytesWritten;
}
BOOL CSerialInterface::Flush()
{
    if (!FlushFileBuffers(h_Port)) //Flushes IO Buffers
    {
        MessageBox(NULL, "Unable to Flush Buffer", "Error", MB_OK + MB_ICONERROR);
        return false;
    }
    return true;
}
void CSerialInterface::ClosePort()
{
    CloseHandle(h_Port); //Deletes Port Handle
    return;
}
void CSerialInterface::SetDTR()
{
    EscapeCommFunction(h_Port,SETDTR); //Sets DTR
}
void CSerialInterface::ClearRTS()
{
    EscapeCommFunction(h_Port,CLRRTS); //Clears Return To Send Buffer
}
//Sends a single character to the com port, stated by h_Port
BOOL CSerialInterface::SendChar(char *c_Char)
{
    if (!TransmitCommChar(h_Port,*c_Char))
    {
        MessageBox(NULL, "Unable to Transmit CHAR", "Error", MB_OK + MB_ICONERROR);
        return false;
    }
    return true;
}
//Resets port, Clears any IO errors and flushes buffers
void CSerialInterface::ResetPort()
{
    DWORD dwError;
    ClearCommError(h_Port,&dwError,&_Stat);
    Flush();
}
void CSerialInterface::RXClear()
{
    PurgeComm(h_Port,PURGE_RXCLEAR); //Clears Recieve Buffer
}

// SerialInterface.h: interface for the CSerialInterface class.
//
#ifdef AFX_SERIALINTERFACE_H_CBDF6013_288E_11D7_987E_00C095EE619C_INCLUDED_
#define AFX_SERIALINTERFACE_H_CBDF6013_288E_11D7_987E_00C095EE619C_INCLUDED_
#endif
#endif

```

```

#pragma once
#endif // _MSC_VER > 1000
class CSerialInterface
{
public:
    void RXClear();
    void ResetPort();
    BOOL Flush();
//Implementation
    CString portname;
    HANDLE h_Port;
    DCB _config;
    COMMTIMEOUTS _Timeout;
    COMSTAT _Stat;
    BOOL _PReady;
//Functions
    BOOL OpenPort();
    BOOL ConfigPort();
    BOOL SetTimeOuts();
    BOOL Write(const void* buf, DWORD dwBytes);
    BOOL SendChar(char *c_Char);
    void ClearRTS();
    void SetDTR();
    DWORD Read(void* pbuf, DWORD dwBytes);
    void ClosePort();
//Constructors and Destructors
    CSerialInterface();
    virtual ~CSerialInterface();
};
#endif // !defined(AFX_SERIALINTERFACE_H__CBDF6013_288E_11D7_987E_00C095EE619C__INCLUDED_)

```

Appendix 3

The defining of the functions of the main program

```

#ifndef _TRIAXAPI_H_
#define _TRIAXAPI_H_

/* Written By Colin Bull 14/01/2003 */ /* Defines the command descriptions, to the spectrometer to enhance ease of use */
// NOTE'S AFTER EACH DEFINE ARE PARAMETERS TO BE PASSED AS CHARACTERS

//Following commands are UTIL based

#define UTIL_WHERE_AM_I " "
#define UTIL_STARTUP_INTELLIGENT_MODE 247
#define UTIL_SET_INTELLIGENT_MODE 248 //No RETURN
#define UTIL_SET_TERMINAL_MODE "Y"
#define UTIL_START_MAIN_PROGRAM "O2000\0"
#define UTIL_REBOOT_IF_HUNG 222
#define UTIL_READ_BOOT_VERSION "y"
#define UTIL_READ_MAIN_VERSION "z"
#define UTIL_CHANGE_IEEE488_ADD "E"

// MOTOR control

#define MOTOR_INIT "A"
#define MOTOR_SET_SPEED "B" //MONO SYSTEM, MIN FRQ, MAX FRQ, RAMP TIME (ms)
#define MOTOR_READ_SPEED "C" //MONO SYSTEM
#define MOTOR_BUSY_CHECK "E"
#define MOTOR_MOVE_RELATIVE "F" //MONO SYSTEM, STEPS TO MOVE
#define MOTOR_SET_POSITION "G" //MONO SYSTEM, STEP POSTION
#define MOTOR_READ_POSITION "H" //MONO SYSTEM,
#define MOTOR_LIMIT_STATUS "K" //returns MOTOR LIMIT status followed by <cr>
#define MOTOR_STOP "L"

// SLIT control (NOTE, THE FOLLOWING STRINGS NEED TO HAVE PARAMETERS TAGGED TO THEM)

#define SLIT_SET_SPEED "g" //MONO SYSTEM, SLIT#, FRQ (STEPS/SEC)
#define SLIT_READ_SPEED "h" //MONO SYSTEM, SLIT# // returns FRQ
#define SLIT_SET_POSITION "i" //MONO SYSTEM, SLIT#, STEP POSITION
#define SLIT_READ_POSITION "j" //MONO SYSTEM, SLIT#, // returns SLIT STEP POSTION
#define SLIT_MOVE_RELATIVE "k" //MONO SYSTEM, SLIT#, STEPS TO MOVE

//MONOCHROMATOR ACCESSORY control

#define ACC_SHUTTER_OPEN "W" //MONO SYSTEM,
#define ACC_SHUTTER_CLOSE "X" //MONO SYSTEM,

```

```

#define ACC_TURRET_POS1      "a" //MONO SYSTEM,
#define ACC_TURRET_POS0      "b" //MONO SYSTEM,
#define ACC_ENTR_MIRROR_SIDE "c" //MONO SYSTEM,
#define ACC_ENTR_MIRROR_FRONT "d" //MONO SYSTEM,
#define ACC_EXIT_MIRROR_SIDE "e" //MONO SYSTEM,
#define ACC_EXIT_MIRROR_FRONT "f" //MONO SYSTEM,
#define ACC_BUSY_CHECK      "I" //returns BUSY STATUS "q" = BUSY "z" = NOT BUSY

//DATA ACQUISITION control

#define ACQ_MEASURE_OFFSETS  "w" //CHANNEL, //returns OFFSET GAIN 1,10,100,1000
#define ACQ_SET_OFFSETS     "x" //CHANNEL, OFFSET GAIN 1,10,100,1000
#define ACQ_CHANNEL_GAIN_SET "R" //CHANNEL, GAIN LEVEL [0..4]
#define ACQ_GAIN_READ       "S" //CHANNEL, //returns GAIN LEVEL [0..4]
#define ACQ_INTEGRATION_TIME_SET "O" //CHANNEL, TIME(ms)
#define ACQ_INTEGRATION_TIME_READ "P" //CHANNEL //returns TIME(ms)
#define ACQ_START           "M" //CHANNEL
#define ACQ_STOP            "N"
#define ACQ_BUSY            "Q" //returns BUSY STATUS "q" = BUSY "z" = NOT BUSY
#define ACQ_READ_DATA       "T" //CHANNEL //returns ACQDATA, OVERRANGE[0..1],
GAIN[0..4]

//HIGH VOLTAGE control

#define HIGH_VOLTAGE_SET      "U" //HV MODULE, HV LEVEL
#define HIGH_VOLTAGE_READ    "V" //HV MODULE, //returns HV LEVEL

//TTL I/O Port controls

#define TTL_WRITE_OUTPUT     "m" //PORT VALUE
#define TTL_READ_OUTPUT      "n" //returns PORT VALUE
#define TTL_READ_INPUT       "o" //returns PORT VALUE
#define TTL_AUTOMATIC_OUTPUTS "Z11" //BINARY IN LINES [0..7] TO PORT

// Triax Specific Commands

#define MW_X_SET_WORKING_ABS_POSITION "Z60"
#define MW_X_MOVE_WORKING_ABS_POSITION "Z61"
#define MW_X_READ_WORKING_ABS_POSITION "Z62"
#define MW_X_SET_INDEX_DEVICE_POS "Z451"
#define MW_X_INDEX_DEVICE_STATUS "Z453"

//Exception

#define ERROR_BOOT 0
#define ERROR_SLIT 1
#define ERROR_GRATING 2
#define ERROR_WAVELENGTH 3
#endif

```

Appendix 4

The Monochromator dialog box:

```

// monDlg.cpp : implementation file

#include "stdafx.h"
#include "SpecUI.h"
#include "monDlg.h"
#include "SpecControl.h"

#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif

//////////////////////////////////////
// CmonDlg dialog

CmonDlg::CmonDlg(CWnd* pParent /*=NULL*/)
: CDialog(CmonDlg::IDD, pParent)
{
   //{{AFX_DATA_INIT(CmonDlg)
    m_Wavelength = _T("");
    //}}AFX_DATA_INIT
}

void CmonDlg::DoDataExchange(CDataExchange* pDX)

```

```

{
    CDialog::DoDataExchange(pDX);
    //{{AFX_DATA_MAP(CmonDlg)
    DDX_Control(pDX, IDC_MONOSTATUS, m_MonoStatus);
    DDX_Text(pDX, IDC_GRATANGLEM1, m_Wavelength);
    DDV_MaxChars(pDX, m_Wavelength, 6);
    //}}AFX_DATA_MAP
}

BEGIN_MESSAGE_MAP(CmonDlg, CDialog)
    //{{AFX_MSG_MAP(CmonDlg)
    ON_BN_CLICKED(IDC_CLEAR2, OnClear2)
    ON_BN_CLICKED(IDC_GOM, OnGom)
    ON_BN_CLICKED(IDC_STEP, OnStep)
    ON_BN_CLICKED(IDC_STEPSLIT, OnStepslit)
    ON_BN_CLICKED(IDC_RESET, OnReset)
    //}}AFX_MSG_MAP
END_MESSAGE_MAP()

////////////////////////////////////
// CmonDlg message handlers

BOOL CmonDlg::OnInitDialog()
{
    CDialog::OnInitDialog();
    InitEdit(); //Is used to initiate all the boxes
    return TRUE; // return TRUE unless you set the focus to a control
                // EXCEPTION: OCX Property Pages should return FALSE
}

void CmonDlg::OnClear2()
{
    InitEdit(); // Is used to initiate the clear function (button)
}

void CmonDlg::InitEdit()
{
    CButton* pGrating1M = (CButton*)GetDlgItem(IDC_GRATING1M);
    CButton* pGrating2M = (CButton*)GetDlgItem(IDC_GRATING2M);
    CButton* pGrating3M = (CButton*)GetDlgItem(IDC_GRATING3M);
    // Make a pointer for each of the grating boxes and initiate gratings
    pGrating1M->SetCheck(1); // Sets default grating as the grating 1 and set it on
    if(pGrating2M->GetCheck() == 1) {pGrating2M->SetCheck(0);} //Grating 2 selected remove
    if(pGrating3M->GetCheck() == 1) {pGrating3M->SetCheck(0);} // Grating 3 selected remove
    // Returns the grating to the default, which is grating 1
    CEdit* pEdit1 = (CEdit*)GetDlgItem(IDC_GRATANGLEM1);
    CEdit* pEdit2 = (CEdit*)GetDlgItem(IDC_SLITWIDTHM);
    CEdit* pEdit3 = (CEdit*)GetDlgItem(IDC_GRATANGLEREL);
    CEdit* pEdit4 = (CEdit*)GetDlgItem(IDC_SLITREL);
    pEdit1->SetReadOnly(FALSE);
    pEdit2->SetReadOnly(FALSE);
    pEdit1->SetWindowText("");
    pEdit2->SetWindowText("");
    pEdit3->SetWindowText("");
    pEdit4->SetWindowText(""); //Clears the text box by making null
}

void CmonDlg::OnGom()
{
    CSpecControl * Ctrl = new CSpecControl; //Create pointer to Spectrometer control class
    //Pointers to Controls
    CEdit* pEdit1 = (CEdit*)GetDlgItem(IDC_GRATANGLEM1); //Grating Wavelength
    CEdit* pEdit2 = (CEdit*)GetDlgItem(IDC_SLITWIDTHM); //Slit Width
    CButton* pGrating1M = (CButton*)GetDlgItem(IDC_GRATING1M); //Grating 1 Radio Button
    CButton* pGrating2M = (CButton*)GetDlgItem(IDC_GRATING2M); //Grating 2 Radio Button
    CButton* pGrating3M = (CButton*)GetDlgItem(IDC_GRATING3M); //Grating 3 Radio Button
    m_MonoStatus.ResetContent();
    CString wavelength, Wave;
    CString slitwidth, slit;
    //Extracts input text from Edit Boxes
    pEdit1->GetWindowText(wavelength);
    pEdit2->GetWindowText(slitwidth);
    //Finds Which grating is selected
    if(pGrating1M->GetCheck() == 1)
    {
        if(Ctrl->SetGrating("0","0","0","0")){m_MonoStatus.AddString("Grating 1 Set"); m_MonoStatus.UpdateWindow();}
        else {m_MonoStatus.AddString("Error: Cannot Set Grating"); m_MonoStatus.UpdateWindow();}
    }
    if(pGrating2M->GetCheck() == 1)
    {

```

```

        if(!Ctrl->SetGrating("0","0","0","1")){m_MonoStatus.AddString("Error: Cannot Set Grating");
m_MonoStatus.UpdateWindow();}
        else{m_MonoStatus.AddString("Grating 2 Set");}
    }
    if(pGrating3M->GetCheck() == 1)
    {
        if(!Ctrl->SetGrating("0","0","0","2")){m_MonoStatus.AddString("Error: Cannot Set Grating");
m_MonoStatus.UpdateWindow();}
        else{m_MonoStatus.AddString("Grating 3 Set"); m_MonoStatus.UpdateWindow();}
    }

    //Outputs Wavelength positon and Status
    Wave += "Wavelength set to: " + wavelength;
    Wave += " nm";
    if(Ctrl->SetWavelength("0",wavelength)){m_MonoStatus.AddString(Wave);
m_MonoStatus.UpdateWindow();}
    else{m_MonoStatus.AddString("Error: Cannot Set Wavelength"); m_MonoStatus.UpdateWindow();}
    pEdit1->SetReadOnly(TRUE);
    Sleep(2000);
    //Outputs Slit positon and Status
    slit += "Slit Width set to: " + slitwidth;
    slit += " steps";
    if(Ctrl-
>MoveSlitRel("0","0",slitwidth)){m_MonoStatus.AddString(slit);m_MonoStatus.UpdateWindow();}
    else{m_MonoStatus.AddString("Error: Cannot Set Slit Width");m_MonoStatus.UpdateWindow();}
    pEdit2->SetReadOnly(TRUE);
    delete Ctrl; //Deletes pointer to control class
}
// Controls move relative on the grating allows you to tweak the wavelength
void CmonDlg::OnStep()
{
    CSpecControl * Ctrl = new CSpecControl;
    CString Steps, Rel;
    CEdit* pEdit1 = (CEdit*)GetDlgItem(IDC_GRATANGLEREL);
    pEdit1->GetWindowText(Steps)
    Rel += "Grating Stepped: " + Steps;
    Rel += " steps";
    if(Ctrl->MoveMotorRel("0",Steps)){m_MonoStatus.AddString(Rel); m_MonoStatus.UpdateWindow();}
    else{m_MonoStatus.AddString("Error: Cannot Step Grating"); m_MonoStatus.UpdateWindow();}
delete Ctrl;
}
// Controls move relative on the slit allows you to tweak the width
void CmonDlg::OnStepslit()
{
    CSpecControl * Ctrl = new CSpecControl;
    CString Steps, Rel;
    CEdit* pEdit1 = (CEdit*)GetDlgItem(IDC_SLITREL);
    pEdit1->GetWindowText(Steps);
    Rel += "Slit Stepped: " + Steps;
    Rel += " steps";
    if(Ctrl->MoveSlitRel("0","0",Steps)){m_MonoStatus.AddString(Rel); m_MonoStatus.UpdateWindow();}
    else{m_MonoStatus.AddString("Error: Cannot Step Grating"); m_MonoStatus.UpdateWindow();}
delete Ctrl;
}
//Resets Message Boxes
void CmonDlg::OnReset()
{
    CSpecControl * Ctrl = new CSpecControl;
    if(!Ctrl->MotorInit()){MessageBox("Unable To Reset Postions Please
Reboot","Error:",MB_OK+MB_ICONERROR);}
    delete Ctrl;
    OnGom();
}
}

```