Experiments on de Vries Liquid Crystals
A Software Approach

A Senior Project
presented to
the Faculty of the Physics Department
California Polytechnic State University, San Luis Obispo

In Partial Fulfillment
of the Requirements for the Degree
Bachelor of Science

by
Austin Havens
June, 2010

© 2010 Austin Havens
<table>
<thead>
<tr>
<th>Module</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Avi analyze and datamatching</td>
<td>21</td>
</tr>
<tr>
<td>Dataset class</td>
<td>22</td>
</tr>
<tr>
<td>oopdatamatching1.m</td>
<td>26</td>
</tr>
<tr>
<td>avianalyze.m</td>
<td>27</td>
</tr>
<tr>
<td>Dataimport</td>
<td>28</td>
</tr>
<tr>
<td>LCGUI</td>
<td>29</td>
</tr>
<tr>
<td>Videoprocessingmodified.m</td>
<td>29</td>
</tr>
<tr>
<td>vidprocessing.m</td>
<td>32</td>
</tr>
<tr>
<td>mypreview_fcnmodified.m</td>
<td>33</td>
</tr>
<tr>
<td>update2.m</td>
<td>34</td>
</tr>
<tr>
<td>popup.m</td>
<td>35</td>
</tr>
</tbody>
</table>
Table of Figures

Figure 1. computer generated model of an isotropic liquid crystal sample. ................................. 6
Figure 2. computer generated model of a nematic liquid crystal sample...................................... 7
Figure 3, computer generated model of a smectic A liquid crystal sample ................................. 8
Figure 4, computer generated model of a smectic C liquid crystal sample ................................. 8
Figure 5. Screenshot of the liquid crystal graphical user interface (LCGUI). ............................. 10
Figure 6. Birefringence data taken without LCGUI................................................................. 15
Figure 7. Birefringence data taken with LCGUI................................................................. 15
Figure 8. plot of intensity vs. applied voltage ................................................................. 16
Figure 9. plot of the standard deviation vs. applied voltage ............................................... 16
Figure 10. contour plot of the histograms of each frame at the voltage they were taken........ 17
Abstract

This paper describes two programs I developed to facilitate the study of liquid crystals. The first program is a graphical user interface to increase the accuracy of the birefringence measurements, which relates to their orientational order, by using a camera. The second program was designed to help study the effects of a time varying fields on liquid crystals by matching data from a recorded video to oscilloscope data in order to attach data from image analysis to the voltage applied to the cell.

Introduction

This paper describes software developed to aid in the testing of the birefringent property of liquid crystals. The complete experiment performed by our group was to measure the effects of voltage on several properties of liquid crystals as a means to identify useful.

The term liquid crystals encompasses a wide variety of materials that have some behavior similar to solid crystals and some behavior similar to liquids. Additionally, liquid crystal molecules tend to be long and thin. The shape of liquid crystal molecules contributes to much of the way they behave. In most liquids, the behavior of a molecule is determined by the position and movement of the surrounding molecules. In a liquid crystals the orientation of the surrounding molecules also makes a contribution. Because of this, liquid crystal molecules tend to align themselves with their surroundings.

Because of their elongated nature, liquid crystal molecules have a different indexes of refraction along their different axes. This difference in index of refraction is known as birefringence. What this means is that light waves react differently with the molecules
depending on the direction of the oscillations of the light. This direction of oscillation of light is known as polarization. Most light has a broad distribution of polarizations and is considered un-polarized. Light that is polarized in the same direction as the long axis of the liquid crystal molecules will travel at a different speed than light polarized in the direction of the short axis. The net birefringence of a bulk liquid crystal sample depends on the birefringence of its molecules as well as the molecules orientations. For example, a sample with no orientational order has no birefringence. Because birefringence of a bulk sample depends on the orientation of the molecules, it can be used to study that orientation and determine how much order the material possesses. Liquid crystals live in the gray area between the ordered crystals and the disordered isotropic liquid phase. In this area there are a multitude of ways that matter can arrange itself to be ordered in some ways and disordered in others. The phase of a material is determined by what kind of order it has. In a crystal, the positions of the particles have a high amount of order. Particles in a traditional liquid are disordered and their positions and orientations have a broad distribution, this is known as the isotropic phase.

Figure 1. computer generated model of an isotropic liquid crystal sample.
Liquid crystals are usually long molecules that add the possibility of having an orientation unlike spherical or point like particles that make up most common liquids. The additional dimensions allow for more ways for a material to be ordered. For example, a material could have its molecules positions distributed randomly like an isotropic liquid but have them pointing in the same direction. This is known as a nematic liquid crystal phase, relating to the mesomorphic state in which the molecules are oriented in loose parallel lines.

Other phases will have varying degrees of positional and orientational order. Our group's experiments are on a type of liquid crystals known as de Vries materials. De Vries liquid crystals have smectic phases where the molecules form layers and have some positional order or alignment.
Some liquid crystals can be controlled by electric fields. This property is what makes liquid crystals useful for devices such as liquid crystal displays (LCD) such as are found in modern cell phones, televisions and computer monitors. The de Vries liquid crystals that we are studying, will rotate the layers of molecules in an electric field. The amount the layers rotate depends on the strength of the electric field.
Birefringence Experiment

We want to explore how the orientational order of de Vries liquid crystals varies with temperature, especially near the transition between the smectic A and smectic C liquid crystal phases. We use the birefringence to tell us about the orientational order of the sample. To measure the birefringence of a bulk sample, we use a crossed polarizers, which will not allow light to pass unless there is a net birefringence between them. Between the crossed polarizers we put our sample and a compensating crystal of known birefringence. When the birefringence of the compensating crystal matches that of the liquid crystal sample but in the opposite direction, there is no net birefringence between the crystals and no light passes through the test setup. My contribution to this experiment was to develop a computer program to assist in the measurement of the birefringence by using a camera attached to a computer. The program that I developed for this experiment includes a graphical user interface (GUI). The program's name is LCGUI. The program takes a live video feed and allows the user to select a rectangular region of interest (ROI) to examine by giving the user statistics about the region such as average pixel intensity or a histogram of the pixel intensities. The average pixel intensity is the most important feature of the GUI program because it allows the user to get a numerical value for the brightness of the region instead of the user relying on their eyes and judgment.

Liquid Crystal Graphical User Interface

Here is a description of the general purpose GUI that we use for the birefringence experiments. This program takes the raw video feed from the camera and allows the user to interact with the data contained in the streaming video frames. The largest section of the GUI is used to display the live video feed from the camera. Below the image, there are buttons that
allow the user to control the GUI and perform actions such as selecting a region of interest. Above the image are menus that allow the user to change the modes of operation of the GUI. To the right of the image, there are graphs that plot the most recently analyzed frame data. At the bottom of the GUI, there are readouts of the current average intensity of the region of interest. Next to the average intensity, are the transition temperatures of the currently selected material.

![Figure 5. Screenshot of the liquid crystal graphical user interface (LCGUI).](image)

**Operating**

The GUI is controlled by several buttons and menus. The menus are found at the top of the window and allow the user to select various options. One menu allows the user to choose between a color image and a grayscale image. The material menu allows the user to select a liquid crystal material and displays the transition temperatures of the selected materials in the lower right corner of the GUI for easy reference. The buttons at the bottom of the GUI perform one time operations. The start and stop preview buttons begin and end the video feed. The close button closes the program. The start and stop record buttons are not currently
functioning. The Rectangle button allows the user to select a rectangular region of interest (ROI) on the image. Only the pixels inside the ROI are analyzed, this allows the user to select a well aligned section of the liquid crystal cell and avoid large defects. In current versions of LCGUI the ROI is stored in a file so that when the program is closed and restarted the region is in the same place.

**Time Varying Electric Field Experiment**

The first experiment dealt with temperature. The second experiment has to do with the electrical properties of de Vries liquid crystals. In this experiment we are primarily examining how the intensity of light transmitted through a sample changes when an electric field is applied. The intensity of transmitted light is related to the tilt angle of the molecules in relation to the crossed polarizers by Equation 1.

\[
I(\theta) = I_{\text{min}} + I_{\text{max}}\sin^2 2\theta
\]

Equation 1

Where \(\theta\) is the angle between the polarizer and the molecule and \(I\) is the intensity of transmitted light. It is not possible for human eyes to quantify the intensity, so we need to use another method. We have two pieces of equipment in our lab capable of measuring the intensity of light. The first piece is a photo diode, the second piece is our microscopy camera. Each piece of equipment have some advantages over the other. Therefore, it makes sense to use both of them. The photo diode is useful because it can directly measure the intensity of transmitted light and because it is attached to the same oscilloscope as the function generator. The setup shares the same time data and can record the applied voltage. On the other hand, the camera can provide information about what different parts of the cell are doing as well as
the average behavior. The down side of the camera in our lab is that the video it takes does not have a reliable start time that can be compared to the oscilloscope time. To get around this, I made a program that matches the camera data to the photodiode data using the common average intensity data. Once the data is matched so that they are linked to a common time, data from one set can be interpolated at times when data exists in the other set so that they can be compared. The program is called AVI Analyze.

Data matching

This program connects datasets that share a measurement. Currently this program is specific to one experiment. In our experiment, we link data collected from an oscilloscope to data collected from a camera. The oscilloscope collects the voltage applied to the cell and the intensity of light transmitted as a voltage from a photo diode. The applied voltage and light intensity are both connected to the oscilloscope time. The camera collects the average pixel intensity and other data from analyzing video frames such as standard deviation of intensity. From the camera frame rate, we can create a camera time and connect the average intensity data and other data from the camera to the camera time. The oscilloscope data and the camera data both have the time dependence of the intensity of transmitted light but because of the different methods of collecting the data, the two sets of data have different sample rates, amplitudes, start times, and other characteristics and cannot be compared directly. In order to compare the two data sets, we all of the data to be linked to a common feature, in our case this feature is the time. To link the data to a common time, we rely on another common measurement that depends on the time, for us this is the intensity. The idea behind the data matching program is to find minimize the difference between the two intensity data series. The
two data series cannot be matched directly for a few reasons, all of which come down to the data series not being on the same scale in some way. The easiest scaling issues to correct are amplitude scaling issues. To solve the amplitude scaling issues, all of the data is normalized and the offsets removed. The harder scaling issues come from the different sample rates of the oscilloscope and the camera. As it turns out, the time scaling has to be incorporated into the data matching in a more involved way than the intensity scaling. The objective of the data matching program is to find out how much to offset the start times of the two data series. To do this, we create a new time series for one of the data sets that has the same time spacing as the other data set and we interpolate the intensity data at those times. The two data series will not completely match up at first because the start times are not right. We use the root mean square error to tell us how well the two data sets match. We minimize the root mean square error by iteratively changing the start times and interpolating the intensity at the new time series. This data can be compared to link all the data to a common time line. Once a common time line is created, we can use interpolation to get values from both data sets at the same time.

**Procedures**

To start this experiment, it is best to have a well aligned sample that is switching polarization and orientation. Set the function generator to produce the desired waveform. Then, using the photodiode and the oscilloscope, set the oscilloscope to X-Y mode to plot the photodiode voltage against the applied voltage on the cell. Rotate the cell such that the maximum intensity is the same for negative and positive voltages of the same magnitude. Switch the oscilloscope to the normal mode and save the data to a floppy as a comma
separated variable file(*.CSV). Transfer the data to the computer and remove the oscilloscope information at the end of the file and adjust the time so that it starts with 0 so that it works with the interpolation program. Next, change out the photodiode for the camera and take an video of the cell switching at as high of a frame rate as you can get for several cycles using the Lumenera infinity capture software. Run the AVI through the AVI Analyze program to get the time, intensity and standard deviation data. Now, put all of the data into an excel spread sheet in a worksheet labeled “formattedData”. In the “formattedData” sheet, the data needs to be in columns in the following order: oscilloscope time, photodiode voltage, applied voltage, camera time, camera intensity, camera standard deviation. Then run the oopdatamatching1 script, which is a purpose build script for this experiment, and select the excel file and the AVI. This is the method that utilizes the oopdatamatching script as it is currently written to load data into the data sets. It uses the dataset class in a limited capacity.

Alternate ways of loading data into dataset classes may be better. For example you could load data into matlab variables and then assign those variables to the appropriate fields of the dataset class.

Results

Figure 6 shows retardation data taken before I created the LCGUI program. Figure 7 shows data taken with the LCGUI program. The retardation is what we read from the compensating crystal and is the proportional to the birefringence and the thickness. The data was taken on cells of different thicknesses so the data from the two figures is in different ranges. Figure 6 shows a broader spread in the data than that of Figure 7 which demonstrates
the increased precision obtained by using the LCGUI program. Using the LCGUI program, we are able to reach the precision of the compensating crystal which we were unable to do before.

Figure 6. Birefringence data taken without LCGUI

Figure 7. Birefringence data taken with LCGUI
Figure 7, Figure 8, and Figure 9 are produced by the data matching program and are from data taken on a TSiKN65 sample at 19º C. These plots combine the data taken with our photo diode with the data from our camera.

**Figure 8.** Plot of intensity vs. applied voltage.

Figure 8 is a plot of average pixel intensity versus voltage applied to the cell. This shows the capability of the data matching program. The striping of the data is probably due to some inconsistencies between the photodiode data and the camera data frame rates.

**Figure 9.** Plot of the standard deviation vs. applied voltage.
Figure 9 shows the standard deviation vs. voltage. While the standard deviation may give us some information about the liquid crystals, such as an indication of the degree to which molecules move in unison or spread out or how well aligned our sample is (a well aligned sample having a low standard deviation), it also demonstrates what the data matching program does. Our laboratory does not have an instrument that can measure standard deviation against voltage, or even against a time that we can relate to the time from other instruments.

Figure 10. contour plot of the histograms of each frame at the voltage they were taken.

Figure 10 shows the distribution of pixel intensities as it changes with applied voltage which is a good example of the capabilities of the data matching program and camera because it demonstrates how multidimensional data may be linked together. Figure 10 contains all of the information in Figure 8 and Figure 9 and more. If all of the molecules moved in unison, the red region should extend through the entire voltage range. Instead, the distribution spreads out
and even seems to split, this could be due to different domains tilting differently and would explain the large jump in the standard deviation at around 20 volts.

**Future Development**

Future development of these programs could expand the *LCGUI* program to further interface with our equipment and streamline the data collection and analysis in our liquid crystal experiments. Future development can be separated into three areas. The first area would consist of incorporating existing programs into the GUI so that they are easier to use. The second area of development would involve improving communications between the computer and the equipment. The third area would consist of getting new data from current equipment and adding equipment to get new data.

In the first area of development, *LCGUI* may be modified to create different modes for the different experiments. This can be done by creating the multiple update functions (Mypreview_fcnmodified.m) and having a menu to chose which to use. For example the default mode would be the current GUI used for birefringence measurements. Another mode might involve a GUI that walks the user through all of the steps needed to use the *AVI Analyze* program and data matching programs. Another feature that could be included in the *LCGUI* would be an user interface for data collection. Having one interface for data collection would establish a standard data format and file naming convention that could be used to easily find previously recorded data no matter who recorded it. The best data collection interface should involve getting as much data from the equipment as possible without operator having to do anything. This leads to the second area of future development.
In the second area of future development, the LC GUI could be modified to interact with other pieces of equipment instead of just the camera. Some examples of increased communications with the equipment would be getting LC GUI to record AVI files, getting LC GUI to read the temperature from the PID controller, and getting LC GUI to interface with the oscilloscope. Getting LC GUI to record AVI files will probably require writing some functions in C/C++ using the camera API from Lumenera because many functions in the API are not available in MatLab. Reading the temperature from the PID controller would involve using the protocol from Omega. Interfacing with the oscilloscope would require new equipment.

The third section of future development could involve new equipment or new uses of old equipment. One example of the later would be calculating the birefringence from the color of the transmitted light seen by the camera and the thickness of the cell. In order to achieve this, you would have to figure out the wavelength of light from the RGB values reported by the camera. Lumenera provides a chart of the pixel sensitivity based on color that could be used to find the wavelength from the relative intensities of red green and blue. The second step would be to use the cell thickness and wavelength to find birefringence. The Michel-Levi interference color chart shows how the interference color and thickness are related to birefringence. Using this method for finding birefringence and by getting data from the PID controller, it should be possible to completely automate the collection of birefringence vs. temperature data.

Apendex A: Where to find things

Everything I have done is in the Havens folder on the desktop of the lab computer. In this folder, the programs are generally found in the programs under development sub folder. In
the programs under development folder are subfolders for the different programs, some of which have multiple builds. At the time of writing this, the most current working version of the LCGUI is version 3.1. The most current version of the AVI analyzing program and data matching are in the oopdatamatching folder.

Appendix B: Tips and Tricks

Tips for Getting Good Video

Getting good video for data matching can be tricky, the goal is to get high frame rate video with a large range of intensities but not to saturate the pixels. The frame rate is the hardest to get fast. The camera frame rate is determined by one of a few possible limiting factors. The two main limiting factors are the exposure and the data transfer. The exposure can be controlled with sliders in the infinity capture program, a lower exposure will result in a higher frame rate unless the camera is limited by something else. Lower exposures also lead to a smaller range of intensities. The data transfer is essentially constant unless limited by the exposure so the frame rate can be increased by lowering the resolution of the frames. Generally lowering the resolution and exposure will help increase the frame rate. Other settings also help increase frame rate such as grayscale and binning. The highest frame rate we have achieved with our equipment is around 50 frames per second but the camera does not always get the predicted frame rate so it is best to check the frame rate of the file by right clicking it and looking at the details in its properties.
Data Handling

Getting data to work well with the data matching programs has been streamlined in the `oopdatamatchign1` script but there still are some potential pitfalls. The data normalization and time zeroing is now built into the AVI analyze function for the video data and the `datainput` function for the photodiode data so they are not as important as in previous iterations of the data input procedure. There still are potential problems getting the photodiode data to MATLAB because of the different excel file types and subtleties of how operating systems can try to archive excel files. Also the data import will frequently produce error messages about excel failing even when it works. The formatting of the excel file is important as well. First, the name of the worksheet must be consistent with the name that the MATLAB `datainput` function tries to read ("formattedData" as written). Second, there cannot be anything under the data, which is important because the CSV file from the oscilloscope puts the oscilloscope parameters at the bottom of the file so they must be removed. The top two rows of text are ignored and the order of the data is important (time, photodiode voltage, then drive voltage). Additionally, any strange formatting can potentially cause problems.

Appendix C: Code

Avi analyze and datamatching

These programs are used for the time varying field experiment. I am putting this code before the Liquid Crystal Graphical User Interface (`LCGUI`) code because there is not a good user interface for these programs so they require a little understanding to use. The reason for using this object oriented approach is to reduce the number of variable names. The way this program
is designed is for there to be an instance of the dataset class for each time scale. For example, in our experiment, the oscilloscope time is different from the camera time so they would require different instances of the dataset class. The ooopdatamatching.m script provides an example of how to load data into the dataset class. In the dataset class the x and y data are used to match the two data sets, the otherydata holds the other data linked to the x and y data. In our experiment, the x data is time, y data is intensity, and the otherydata are variables like input voltage in the oscilloscope dataset or standard deviation or histograms in the camera dataset.

Once the data is loaded into instances of dataset classes, the matching can be performed using methods in the dataset class. The ooopdatamatching1.m script has examples of how this is done.

**Dataset class**

```matlab
%ooopdatamatching1
%
% dataset class for oop data matching
% an object oriented approach to the data matching program
% created 5/6/2010 by Austin Havens
% modified 5/12/2010 by Austin Havens
% added some surface plotting capabilities
% modified 6/9/2010
% added a method to return data instead of plotting it
%
classdef dataset
    properties
        points;%number of data points ex. frames
        samplerate;% ex. frame rate
        xdata;% ex. time
        ydata;% ex. intensity
        otherydata;% other data ex stdev in a map
        adjustment;
    end
    methods
        function obj = dataset(n,r,x,y,o)
            obj.points = n;
            obj.samplerate = r;
            obj.xdata = x;
        end
    end
end
```
obj.ydata = y;
obj.otherydata = o;
end

function DEV = ferrorah(obj, obj2, a)
q = obj;
x1a = (q.xdata) - a;
x2 = obj2.xdata;
if max(obj2.xdata) > max(x1a)
x2 = obj2.xdata(1:find(obj2.xdata <= max(x1a), 1, 'last'));
  L = length(x2);
else
  L = length(x2);
end
oneprime = interp1(x1a, obj.ydata, obj2.xdata, 'pchip');
DEV = ((sum((oneprime(1:L - double(int16(a)))) - obj2.ydata((double(int16(a)) + 1:L)).^2))/L);
end

function obj = normalize(obj)
  miny = min(obj.ydata);
ydat1 = obj.ydata - miny;
  maxy = max(ydat1);
  obj.ydata = ydat1./maxy;
end

function A = match(obj, obj2)
c = 0;
dataset1 = obj.normalize;
dataset2 = obj2.normalize;
dataset1.ferrorah(dataset2, 0)
f = @(c) dataset1.ferrorah(dataset2, c);
G = fminunc(@(c) f(c), 0); % an alternative minimization route
H = fminsearch(@(c) f(c), 0);
% obj.adjustment = H;
A = H;
end

function plots(obj, obj2, xaxis, yaxis)
  % plots the obj2 otherydata on the x and the obj1 otherydata on the y
  data1 = obj.otherydata;
  % data1('xdata') = obj.xdata;
  data1('ydata') = obj.ydata;
  data2 = obj2.otherydata;
  data2('ydata') = obj2.ydata;
  data2('xdata') = obj2.xdata;
  X = obj2.xdata;
  data3 = containers.Map();
  XI = obj.xdata;
  for k = keys(data2)
    s = cell2mat(k);
    l = num2str(s);
    dim = size(data2(l), 2);
    if dim == 1
      try
        Y = data2(l);
        YI = interp1(X, Y, XI);
        data3(l) = YI;
        catch ME
          disp(ME)
      end
    end
  end
end
disp('interpolation failure')
end
else
    try
        [X2,Y]=meshgrid(1:size(data2(l),2),X);
        Z=data2(l);
        size(X)
        size(X2)
        size(Y)
        size(Z)
        [XI2, YI]=meshgrid(1:size(data2(l),1));
        size(XI2)
        ZI=interp2(X2,Y,Z,XI,YI);
        data3(l)=ZI;
    catch ME
        disp(ME);
        disp('3d failure')
    end
end
end
data3('X')=X;
dim2=size(data1(yaxis),2);
disp('dimension 2')
disp(dim2)
if dim2 ==1
    XP=data3(xaxis);
    YP=data1(yaxis);
    plot(XP,YP,'k.');
    xlabel(xaxis);
    ylabel(yaxis);
else
    try
        ZP=transpose(data1(yaxis));
        size(data3(xaxis))
        %mesh(ZP)
        [XP YP]=meshgrid(data3(xaxis),1:size(data1(yaxis),2));
        %surf(XP,YP,ZP);
        contour(XP,YP,ZP,50);
        xlabel(xaxis)
        ylabel(yaxis)
    catch ME
        disp('3d plots not working')
        size(ZP)
        size(XP)
        size(YP)
        disp(ME)
    end
end
end
function data=get_matched(obj,obj2,xaxis, yaxis)_
%plots the obj2 otherydata on the x and the obj1 otherydata on
%the y
data1=obj.otherydata;
%datal('xdata')=obj.xdata;
data1('ydata')=obj.ydata;
data2=obj2.otherydata;
data2('ydata')=obj2.ydata;
data2('xdata')=obj2.xdata;
X=obj2.xdata;
data3=containers.Map();
XI=obj.xdata;
for k=keys(data2)
s=cell2mat(k);
l=num2str(s);
dim = size(data2(l),2);
if dim ==1
    try
        Y=data2(l);
        YI=interp1(X,Y,XI);
        data3(l)=YI;
    catch ME
        disp(ME);
        disp('interpolation failure')
    end
else
    try
        [X2,Y]=meshgrid(1:size(data2(l),2),X);
        Z=data2(l);
        size(X)
        size(X2)
        size(Y)
        size(Z)
        [XI2, YI]=meshgrid(1:size(data2(l),1));
        size(XI2)
        ZI=interp2(X2,Y,Z,XI,YI);
        data3(l)=ZI;
    catch ME
        disp(ME);
        disp('3d failure')
    end
end
end
data3('X')=X;
dim2=size(data1(yaxis),2);
disp('dimension 2')
disp(dim2)
if dim2 ==1
    XP=data3(xaxis);
    YP=data1(yaxis);
    data=[XP YP];
    %plot(XP,YP,'k.');
    %xlabel(xaxis);
    %ylabel(yaxis);
else
    try
        ZP=transpose(data1(yaxis));
        size(data3(xaxis))
        %mesh(ZP)
        [XP YP]=meshgrid(data3(xaxis),1:size(data1(yaxis),2));
        %surf(XP,YP,ZP);
        %contour(XP,YP,ZP,50);
        data=[XP YP ZP];
    catch ME
        disp(ME)
disp('3d data retrieval not working')
size(ZP)
size(XP)
size(YP)
disp(ME)
end
end
end
end

ooopdatamatching1.m

{%
 oopdatamatching.m
 object oriented approach to the data matching program
 Description: This program gets an avi file and calls avi analyze and then
 retrieves data from an excel file that has the other data. this program
 relays on a dataset class that allows the user to load a sets of data into
 instances of the dataset class. In order to be compared, the data sets need
 to have the same (periodic?)y dependance on the x data and be normalized
 (a normalize method is part of teh dataset class). other data can be
 attached to the otherydata attribute and be carried along with the matching
 data.
 this script is particular to the liquid crystal experiments, to use the
 data matching programing for other purposes, make instances of the dataset
 class with your data as appropriate

 created on 5/6/2010 by Austin Havens
 modified on 5/8/2010 by Austin Havens
 added data matching part of program to the datasetclass
 changed other y data to be a map container (dictionary) to allow for
 multiple carried data for each data set.
 modified on 5/11/2010 by Austin Havens
 added capability for data of larger dimensions
 incorporated avi analyze
 modified on 6/9/2010 by Austin Havens
 changed the way data is imported so that the AVI file does not have to
 be pre analyzed, added teh normalization of the photodiode data.
 %}
close all;
clear all;

[avifile, avipath] = uigetfile(*.avi','pick the right avi file');
[Data stdevavi fps frames histo timel dataone] = AviAnalyze3([avipath,
avifile]);
[file, path] = uigetfile(*.xlsx','pick the right excel file');

[rawdiodedata time2 voltage datatwo]=dataimport([path,file]);

k1={'Standard Deviation','Intensity', 'Histogram'};
v1 = {stdevavi, Data, histo};
k2 = {'Input voltage','Raw Diode Data'};
v2 = {voltage,rawdiodedata};
otherdata1 = containers.Map(k1,v1);
otherdata2 = containers.Map(k2,v2);
dataset1 = dataset(length(dataone),fps,time1, dataone, otherdata1);
dataset2 = dataset(length(datatwo), 'na', time2, datatwo, otherdata2);
%
c=0;
dataset1 = dataset1.normalize;
dataset2 = dataset2.normalize;
dataset1.ferrorah(dataset2,0)

f = @(c) dataset1.ferrorah(dataset2,c);
%G=fminunc(@(c) f(c),0); an alternative minimization rout
H=fminsearch(@(c) f(c),0);
%G,H
dataset1.ferrorah(dataset2,H)
KK=[k1 k2];
matchedthings=containers.Map([k1, k2],[v1, v2]);
%

H=dataset1.match(dataset2);

matcheddata1 = dataset(length(dataone),...
   'na',dataset1.xdata-H,dataset1.ydata,dataset1.otherydata);
figure(1)
matcheddata1.plots(dataset2,'Input voltage','Standard Deviation')
figure(2)
matcheddata1.plots(dataset2,'Input voltage','Intensity')
figure(3)
matcheddata1.plots(dataset2,'Input voltage', 'Histogram')

avianalyze.m

This program takes an audio visual interleave (AVI) file and analyzes it frame by frame
and returns the results. This program returns the average pixel intensity and standard
deviation. This program is used with the data matching program to link the data obtained from
the video frames with other data such as the voltage applied to the cell.

function [Data stdev fps frames histo time datanorm] = AviAnalyze3(file)
%file='C:\Documents and
Settings\Fernsler\Desktop\Havens\testdata_1_28_2010.avi';
%(modified on 5/11/2010 by Austin Havens
   added histogram data
modified on 6/9/2010 by Austin Havens
   added time and normalized data
%}
N=100;
a=aviread(file);
b=aviinfo(file);
fps=b.FramesPerSecond;
frames=b.NumFrames;

timea=0:frames-1;
time=transpose(timea./fps);
for i=1:b.NumFrames
    A=sum(sum(sum(a(i).cdata)))./(b.Width.*b.Height.*3);
    C(i)=A;
    Average = mean(mean(a(1).cdata));
    stdev(i)=sqrt(sum(sum((a(i).cdata-A).^2)))./(b.Width.*b.Height.*3));
    dat=sum(a(i).cdata,3);
    %vec=reshape(dat,1,size(dat,1).*size(dat,2));
    %vec=sort(vec);
    H=hist(sum(a(i).cdata,3),N);
    I(i,:)=sum(H,2)./size(H,2);
end
histo=I;
Data=transpose(C);
datanorm=(Data-min(Data))./(max(Data)-min(Data));
stdev=transpose(stdev);

Dataimport

This function gets data from an excel file and formats it into MatLab arrays so for the main program. The excel file that this program calls has to meet several requirements in order for the program to work. First the excel file must have a sheet named “formattedData”. this is the sheet that will contain the data to be used. Other sheets are ignored and can be used to hold the raw data and/or the final data. In the “formated data” sheet, the first two rows are for headings. The columns have to contain data in the correct order. The first column is the oscilloscope time and it needs to start at zero. The second column is the voltage from the photo diode. The third column is the voltage applied to the cell.

function [rawdata time2 voltage datanorm] = dataimport(filepath)
A=importdata(filepath);
data=A.data.formattedData;
time=data(:,1);
rawdata=data(:,2);
voltage=data(:,3);
time2=time-time(1);
datanorm=(rawdata-min(rawdata))./(max(rawdata)-min(rawdata));
This is the main program which creates the GUI window and the data structures. This script creates the `HandleToTimestampLabel` names are relics from the evolution of the program and correspond to different values being passed to the main function from the window updating function (mypreview_fcnmodified.m).

```matlab
clear all;
close all;

% Create a video input object.
vid = videoinput('winvideo');
set(vid, 'ReturnedColorSpace', 'rgb');
vidRes = get(vid, 'VideoResolution');
imWidth = vidRes(1);
imHeight = vidRes(2);
nBands = get(vid, 'NumberOfBands');

% Create a figure window. This example turns off the default toolbar and menubar in the figure.
fig = figure('ToolBar', 'auto', ...
    'NumberTitle', 'Off', ...
    'Name', 'My Custom Preview GUI', ...
    'Position', [80 80 .8*imWidth .8*imHeight]); % 'Menubar', 'none', ...

% Create the image object in which you want to display the video preview data.
image = image( zeros(imHeight, imWidth, nBands, 'uint8') );

% Set up the push buttons
uicontrol('String', 'Start Preview', ...
    'Callback', 'preview(vid, image)', ...
    'Units', 'normalized', 'fontunits', 'normalized', 'FontSize', .4, ...
    'Position', [.05 0.1 .05]);
uicontrol('String', 'avimode', ...
    'Callback', ...
    'setappdata(image, 'UpdatePreviewWindowFcn', @preview_fcn2)', ...
    'Units', 'normalized', 'fontunits', 'normalized', 'FontSize', .4, ...
    'Position', [.2 .05 0.1 .05]);
uicontrol('String', 'Stop Record', ...
    'Callback', 'stoprec(fig)', ...
    'Units', 'normalized', 'fontunits', 'normalized', 'FontSize', .4, ...
    'Position', [.2 0 0.1 .05]);
uicontrol('String', 'Stop Preview', ...
    'Callback', 'stoppreview(vid)', ...
'Units', 'normalized', 'fontunits', 'normalized', 'FontSize', .4, ...
'Position', [.1 .0 .1 .05]);
uicontrol('String', 'Close', 'Callback', 'close(gcf)', ...
'Units', 'normalized', 'fontunits', 'normalized', 'FontSize', .4, ...
'Position', [0 0 .1 .05]);
uicontrol('String', 'rectangle', 'Callback', 'update2(hImage)', ...
'Units', 'normalized', 'fontunits', 'normalized', 'FontSize', .4, ...
'Position', [.1 .05 .1 .05]);

hPop = uimenu('Label', 'Material', 'Checked', 'on');
  uimenu(hPop, 'Label', 'TSIKN65', ...
    'Callback', '[IA AC]=popup(hImage);');
  uimenu(hPop, 'Label', '8422', ...
    'Callback', '[IA AC]=popup(hImage);');
  uimenu(hPop, 'Label', 'w415', ...
    'Callback', '[IA AC]=popup(hImage);');
  uimenu(hPop, 'Label', '8422[2f3]', ...
    'Callback', '[IA AC]=popup(hImage);');

hColor = uimenu('Label', 'color');
  uimenu(hColor, 'Label', 'Grayscale', ...
    'Callback', 'set(vid,''ReturnedColorSpace'',''grayscale'')');
  uimenu(hColor, 'Label', 'rgb', ...
    'Callback', 'set(vid,''ReturnedColorSpace'',''rgb'')');
  uimenu(hColor, 'Label', 'YCbCr', ...
    'Callback', 'set(vid,''ReturnedColorSpace'',''YCbCr'')');

myhandles = guihandles(hFig);
myhandles.IA=56;
myhandles.AC=26;
myhandles.gr=zeros(2,60);

mat2=[imWidth, imHeight, nBands];
myhandles.Amplitude=0;
myhandles.rec=1;

myhandles.mat=[imWidth, imHeight, nBands];
myhandles.rect=7;
%myhandles=guihandles(hImage);
%roi rectangle position
fileexist='empty';
try
  roi = csvread('roiposition.txt');
  fileexist = 'exist';
catch err1
  rethrow(err1)
end

if (strcmp(fileexist, 'exist') & (roi(3) ~= 0) & (roi(4) ~= 0))
  maxX=imWidth;
  maxY=imHeight;
  myhandles.X=roi(1);
  myhandles.Y=roi(2);
myhandles.W=roi(3);
myhandles.H=roi(4);
h=findobj(gca,'Type','rectangle','Tag','RectangleA');

X=floor(myhandles.X *maxX);
Y=floor(myhandles.Y *maxY);
W=floor(myhandles.W *maxX);
H=floor(myhandles.H *maxY);
delete(h);
h=rectangle('position',[X Y W H],'Tag','RectangleA');
myhandles.rect= h;
drawnow('expose')
guidata(hFig, myhandles);

else

myhandles.X=1;
myhandles.Y=1;
myhandles.W=imWidth-3;
myhandles.H=imHeight-3;
h=findobj(gca,'Type','rectangle','Tag','RectangleA');
delete(h);
h=rectangle('position',...
'Tag','RectangleA');

myhandles.rect= h;
drawnow('expose')
guidata(hFig, myhandles);
end

% Create the text label for the timestamp
hTextLabel = uicontrol('style','text','String','Amplitude', ... 
'Units','normalized','fontunits','normalized','FontSize',.25,... 
'Position',[0.3 0.4 .1]);
hTextLabel2 = uicontrol('style','text','String','ia',... 
'Units','normalized','fontunits','normalized','FontSize',.4,... 
'Position',[0.65 0.05 .15 .05]);
hTextLabel3 = uicontrol('style','text','String','ac',... 
'Units','normalized','fontunits','normalized','FontSize',.4,... 
'Position',[0.65 0 0.15 .05]);

% Specify the size of the axes that contains the image object
% so that it displays the image at the right resolution and 
% centers it in the figure window.
figSize = get(hFig,'Position');
figWidth = figSize(3);
figHeight = figSize(4);
set(gca,'unit','normalized','position',[ 0 .1 .75 .9 ]); 

% Set up the update preview window function.
setappdata(hImage,'UpdatePreviewWindowFcn',@mypreview_fcnmodified);
% Make handle to text label available to update function.
setappdata(hImage, 'HandleToAmplitude', hTextLabel);
setappdata(hImage, 'HandleToIA', hTextLabel2);
setappdata(hImage, 'HandleToAC', hTextLabel3);

setappdata(hImage, 'dataout', mat2);

%popup(hObject, eventdata, handles)
try
preview(vid, hImage);
catch err
    rethrow(err);
end

vidprocessing.m

This function analyzes each frame and returns the results to mypreivew_fcnmodified.

This function retrieves the location of the user selected region of interest from the application data structure and then performs several operations on the video frame data in the range. The most basic operation is an average that gives the user a measure of the amount of light transmitted through the cell.

function [A S I med mode2] = vidprocessing(vid, himage)
frame = getsnapshot(vid);
xd = get(himage, 'XData');
yd = get(himage, 'YData');
maxX = xd(2);
maxY = yd(2);
myhandles = guidata(himage);
X = floor(myhandles.X * maxX);
Y = floor(myhandles.Y * maxY);
W = floor(myhandles.W * maxX);
H = floor(myhandles.H * maxY);
frame = frame(Y:Y+H, X:X+W,:);

N = 50;
H = hist(sum(frame, 3), N);
b = sum(frame, 3);
I = sum(H, 2)./size(H, 2);
vec = reshape(b, 1, size(b, 1).*size(b, 2));
vec = sort(vec);
med = median(vec);
mode2 = mode(vec);
S = std2(frame);
A = sum(sum(frame))./(size(frame, 1).*size(frame, 2).*size(frame, 3));
This function refreshes the data on the GUI. This function calls the data analysis function (vidprocessing.m) and collects the data from the other functions and formats them into displayable forms before passing it back to the GUI. This function also creates the subplots generated from the data returned by vidprocessing.m.

```matlab
function mypreview_fcnmodified(obj, event, himage)

myhandles = guidata(himage);
[A S I med mod2]=vidprocessing(obj,himage);
myhandles.Amplitude=A;
ampstr=[
    {'Amplitude: ', num2str(myhandles.Amplitude)}, ...
    {'median: ', num2str(med)}, ...
    {'mode: ', num2str(mod2)}];
if myhandles.rec == 1
    myhandles.mat=[myhandles.mat;now, [A S]];
end
% Get handle to text label uicontrol.
ht = getappdata(himage,'HandleToAmplitude');
% Set the value of the text labels.
set(ht,'String',ampstr);
setappdata(himage,'dataout',myhandles.mat)
r=size(myhandles.mat);
points=200;
if r(1)<points
    s=1;
else
    s=r(1)-points+1;
end

subplot('position', [.82 .75 .17 .24]);
plot(myhandles.mat(s:r(1),1)-myhandles.mat(2,1),myhandles.mat(s:r(1),2));
%ylim([0 255])
ylabel('amplitude')
xlabel('time')
xlim([myhandles.mat(s,1)-myhandles.mat(2,1)],...
     (myhandles.mat(r(1),1)-myhandles.mat(2,1)))
subplot('position', [.82 .45 .17 .24]);
plot(myhandles.mat(s:r(1),1)-myhandles.mat(2,1), myhandles.mat(s:r(1),3));
ylim([0 50])
```
xlim([myhandles.mat(s,1)-myhandles.mat(2,1),...
    myhandles.mat(r(1),1)-myhandles.mat(2,1)])
ylabel('stdev')
xlabel('time')

subplot('position', [.82 .1 .17 .24]); plot(I);
title('histogram')
ylim([0 100])
ylabel('number of pixels')
xlabel('bin')
xlim([0 50])

guidata(himage,myhandles);
% Display image data.
set(himage, 'CData',event.Data)

return

update2.m

This function is called by the “rectangle” button in the main function. retrieves the
region of interest rectangle. The first step is to get a the location of the region of interest based
on user input. The second step is to print a rectangle on figure in the location of the region of
interest so that the user can view the region. The third step is to record the location of the
region of intrest to the application data structure.

function update2(himage)
rec = getrect(get(himage,'Parent'));
xd=get(himage,'XData');
yd=get(himage,'YData');
maxX=xd(2);
maxY=yd(2);
myhandles = guidata(himage);
myhandles.X = rec(1)./maxX;
myhandles.Y = rec(2)./maxY;
myhandles.W = rec(3)./maxX;
myhandles.H = rec(4)./maxY;
roi=[rec(1)./maxX, rec(2)./maxY, rec(3)./maxX, rec(4)./maxY];
csvwrite('roiposition.txt',roi);

X=floor(myhandles.X *maxX);
Y=floor(myhandles.Y *maxX);
W=floor(myhandles.W *maxX);
Height=floor(myhandles.H *maxY);
% save the changes to the structure
popup.m

This function is used to display the transition temperatures of our liquid crystal materials for easy reference. The function is called by the popup menu and passes the transition temperatures of the material the user selects on the menu to the figure handles so that it can be accessed by the window updating function.

```matlab
function [IA AC] = popup(hPop)
obj=gcb;
val=get(obj,'Position');
set(obj, 'Checked', 'on');
par=get(obj,'Parent');
m=max(size(get(par,'Children')));
children=get(par,'Children');
for i=1:m
    r=get(children(i),'Position');
    if r ~= val
        set(children(i),'Checked','off')
    end
end
if val == 1
    IA=56;
    AC=26;
elseif val == 2
    IA=71;
    AC=47;
elseif val == 3
    IA='?';
    AC='?';
elseif val == 4
    IA=91;
    AC=65;
end
%a=get(hPop, 'Parent');
%ht = getappdata(hPop,'HandleToTimestampLabel2');
%set(ht,'String',num2str(IA));
% get the structure in the subfunction
```
IA = {'Iso-SmA ' [num2str(IA) '°C']};
AC = {'SmA-SmC ' [num2str(AC) '°C']};
ht2 = getappdata(hPop,'HandleToIA');
ht3 = getappdata(hPop,'HandleToAC');
set(ht2,'string',IA);
set(ht3,'string',AC);

myhandles = guidata(hPop);
myhandles.IA = IA;
myhandles.AC = AC;
% save the changes to the structure
guidata(hPop,myhandles);

end