Preface

It is difficult, if not impossible, to imagine the state of modern medicine if not for non-invasive imaging techniques. From the humble beginnings in December of 1895 when Wilhem Röntgen took the first published X-Ray of his wife’s hand, the X-Ray has become the de facto standard when assessing internal injuries. Of course, the X-Ray evolved into a much more sophisticated tomographic imaging tool with the aid of the computer. Today, images are routinely performed using magnetic resonance imaging (MRI), computed tomography (CT) scans, positron emission tomography (PET) scans, and certainly ultrasound and X-Ray. Obtaining the image is only half the equation. To be of therapeutic or medical value, it must be processed and analyzed.

In this laboratory, you will learn how to create a graphical user interface (GUI) that can be compiled on most operating systems in use today (i.e. Linux, Mac®, and Windows®). We will be using the wxWidgets open source C++ GUI toolset. In addition, you will build a GUI using MATLAB®. While it is difficult to compete with the robust and near universal applicability of C++ programming, it is equally as difficult to build fully functional GUIs as easily as it can be done with MATLAB®. To be sure, there is a need to be fluent in both programming languages.
# Contents

1 Introduction to GUI Construction Using the wxWidgets Toolkit ............... 6  
   1.1 The MinGW IDE ................................................................ 7  
   1.2 Image Processing 101: Filters ........................................... 8  
   1.3 Summary ........................................................................... 13  
   1.4 Questions .......................................................................... 14  

2 Nonlinear Filters ........................................................................... 15  
   2.1 The Median Filter .................................................................. 16  
   2.2 Questions ............................................................................. 17  

3 Edge Detection and Thresholding ................................................... 18  
   3.1 Edge Detection ..................................................................... 19  
   3.2 Thresholding ......................................................................... 21  
   3.3 Questions .............................................................................. 22  

4 User Input - Sliders ....................................................................... 23  
   4.1 Sliders .................................................................................. 24  
   4.2 Questions .............................................................................. 28  

5 Interactive Features - Area Calculation ........................................... 29  
   5.1 GUI Input: Mouse Clicks ....................................................... 29  
   5.2 Area Calculation .................................................................... 31  
   5.3 Summary .............................................................................. 35  
   5.4 Questions .............................................................................. 35  

6 Accessorizing Your GUI - Icons and Buttons ................................. 36  
   6.1 The GIMP example ............................................................... 36  

7 Introduction to MATLAB’s® GUI Design Environment (GUIDE) .... 40  
   7.1 GUIDE ................................................................................. 40  
   7.2 Summary .............................................................................. 46  
   7.3 Questions .............................................................................. 46  

8 The MATLAB® Image Processing Toolbox (IPT) ............................. 47
8.1 Region of Interest (ROI) Properties .................................................. 47
8.2 Radon Transform .............................................................................. 50
List of Figures

1.1 The project with all folders expanded. .................................................. 7
1.2 All nine pixels inside the highlighted box are averaged and the result is returned to the center. The box slides through each pixel in the image. Edge effects will be discussed in the lab. ................................................................. 8
1.3 Building your project is necessary before being able to view your new additions. ... 13
1.4 Building, compiling and linking your project. ........................................ 14
   (a) The compiler had an issue with a conversion from a floating point number (float) to an integer (int). ................................................................. 14
   (b) Still, the compiler linked all the files together and successfully built the project. 14
3.1 Making edges more pronounced aids detection and classification. ............. 19
3.2 Again we have our standard filter mask except this time we are scaling the values on either side of the pixel. If it is an edge, one side will be validated, the other will be negated. 20
4.1 Add both files to their respective folders. ............................................. 24
   (a) Right click the "Source Files" folder to add the sliderFrame.cpp file. ....... 24
   (b) Right click the "Header Files" folder to add the sliderFrame.hpp file. (Note the .cpp file appears in the Source Folder above.) .................. 24
5.1 Area calculation. ................................................................................. 34
   (a) If all went well, you should be able to launch the interactive GUI. ......... 34
   (b) In the status bar, the text has change to "in area". ............................. 34
   (c) Notice the color of the line and the status bar showing the location of the current point and the distance away from the first point. .......... 34
   (d) Now that we are within three pixels of the first point (zero actually) the program ends showing the final area. Pretty close! 34
6.1 Open a new image file in GIMP. ......................................................... 37
6.2 Make sure the image size is 16 by 16. .............................................. 37
6.3 Changing foreground and background colors. ..................................... 37
   (a) Open the color palette by left-clicking. ............................................ 37
   (b) Change the color from white something, say, gold. ................. 37
6.4 Fill the layer with the color. ............................................................... 38
6.5 Your first icon. ................................................................................. 38
(a) Choose a new color to make a pattern. ............................................. 38
(b) Aim for something brief but true! ................................................... 38

6.6 Some ideas for icon design. .......................................................... 39
(a) This is a floppy disk. While you may not know what it is, it has been the universal Windows® icon for save for many years. ....................... 39
(b) A question mark works well for the Help icon. .............................. 39
(c) This is a magnifying glass for Preview. ........................................ 39
(d) Fading black to white for threshold. .......................................... 39

7.1 Open the GUIDE editor in MATLAB. ............................................. 41
7.2 Blank GUI template. ................................................................. 41
7.3 Layout the axes for the image. ..................................................... 42
7.4 Rename the push buttons. .......................................................... 43
7.5 The basic template. ................................................................. 43
7.6 Access the code associated with the push button. ......................... 44
7.7 The low pass filter. ................................................................. 46

8.1 These are the general steps to binarize an image in MATLAB. im2bw takes in an optional argument to control the threshold setting. If you want 2 or more images displayed in MATLAB, use the figure command to open a new window. Otherwise MATLAB will overwrite the figure. .............................................................. 48
8.2 Effectively isolating the skull. Type help bwconncomp and help labelmatrix for more information about those functions. ismember is exactly that - it checks to see if one thing (value) is a member of another thing (array of values). ....................... 49
8.3 .................................................................................................. 49
8.4 .................................................................................................. 50
Lab 1: Introduction to GUI Construction Using the wxWidgets Toolkit

PURPOSE: To introduce the functional components of wxWidgets, basic C++ programming techniques, and the course project.

GOAL: Most C++ programming environments are wrapped in an integrated development environment (IDE). The IDE is simply a way for you to type code, compile it, and turn it into an executable. The IDE we will be using is MinGW, or Minimalist GNU for Windows. (Incidentally, GNU is a somewhat cryptic acronym that stands for GNU is Not Unix.) This environment is fairly easy to use and has all the compilers we need to turn our code into a working executable. It is also conveniently bundled in a suite should you decide to put it on your personal computer. You will be given stock code which you will then modify to provide additional functionality. In fact, over the course of the semester, we will simply be adding to changes we’ve made from the previous week. **It is critical that you stay current with the labs from week to week.** All the labs are designed to be completed easily within the lab period time frame.

For lab 1, we will be working through the IDE, introducing the project, and making changes to the stock code.
1.1 The MinGW IDE

Just like any other IDE, the MinGW environment allows you to create a project. A project is just that - a project. It is a collection of files that all work together to complete the project. The base project which we will be modifying consists of nine files and a resource folder. The resource folder is where we keep things like icons that we’ll use for on-screen button buttons. (Like the folder icon that allows you to open a folder or the old floppy disk that prompts you to save.) If the BME465 folder is not on your computer then simply download it from the website http://www.ele.uri.edu/courses/bme465/.

The following are the seven files that make up the project:

- main.cpp
- BME465_Template.cpp
- BME465_Template.hpp
- image_processor.cpp
- image_processor.hpp
- BME465_Template.rc
- BME465_Template_private.rc

If you have a MinGW project file in your folder, and you should, click on the file - it has extension .MDSP, for MinGW Development Suite Project. You should see something very similar to figure 1.1.

![Figure 1.1: The project with all folders expanded.](Image)

There will never be a reason to change the resource files - the ones with the .rc extension. These are needed for the use of wxWidgets. The other file you will not have to change is the main file. If you open it, you’ll see that all it does is launch the application. That means that all the work is done by
four files, namely the .cpp files (for C++) and the corresponding header files .hpp. Also, if you open
the resources folder, you’ll notice .xpm files. These are images that are very small in size which can be
created with a program called Gimp (we’ll cover this in a later lab.) You are free to add as many icons
as you want to your project. This is where they will be stored for later use.
Essentially, that’s the nuts and bolts of the program. Now comes the heavy lifting.

1.2 Image Processing 101: Filters

There is a Debug folder that will contain the executable that is generated after a successful build of the
program. There is also an Images folder which contains the images we’ll be working with for most of
the semester (You are free to choose others.) Launch the executable in the Debug folder and open any
of the images you’d like. Don’t worry if you open a .tiff file and see a “field tag” error - it is generated
because of the tiff structure. It is not harmful but you can open a gif, bmp, or jpeg if you’d prefer.
Now click on the first icon - it looks the Bode plot of a low pass filter. Did the image get fuzzy? Click
it again. If you do this repeatedly, the image will turn into a giant fuzzy mess, eventually converging
to a black image.

The low pass filter works exactly the same in the spatial domain as it does in the time domain. It
averages a bunch of samples (in this case pixels) and returns the value into on pixel. It does this by
creating a 3 x 3 (or larger) box of pixels, often referred to as the filter mask, averages them all together,
and returns the value to the center pixel (figure 1.2). The box slides over one and the process repeats.
This is how the overwhelming majority of image processing filters are created.

Figure 1.2: All nine pixels inside the highlighted box are averaged and the result is returned to the
center. The box slides through each pixel in the image. Edge effects will be discussed in the lab.

Your task is to take the code listed as a low pass filter, found in the image_processing.cpp file and adapt
it to be a high pass filter. DO NOT DELETE THE LOW PASS FUNCTION!!! Simple copy and past
the function below the low pass filter then rename it something clever like \texttt{wxImage \ast \ \text{HighPass}(\texttt{wxImage \ast \ pImage})}. Of course there are a number of bookkeeping items that need to be resolved like changing the code to actually perform a high pass filter function instead of the low pass filter. Plus, changes in the code need to be made to allow this function to be called.

You may already have your own way of making changes to code. If you have C++ programming experience, feel free to keep your own method. However, if you are a little shaky with C++, try to stay calm and follow these simple rules. As stated there are only 4 files we will ever need to change. Only keep those files of your project open. Next, since header files are generally more compact that c-files, start making changes there first.

Let's start with the BME465_Template.hpp file. VERY IMPORTANT - DO NOT MODIFY ANY CODE THAT IS BEIGE COLOR! The header file is generally classes are defined and constructors are created. Note everywhere there appears to be something related to the Low Pass filter (LPF). In this case, the only entry for the LPF is in the enumeration table:

```cpp
// constants
enum
{
    // menu items
    MENU_FILE_OPEN = \texttt{wxID\_OPEN},
    MENU_FILE_QUIT = \texttt{wxID\_EXIT},
    MENU_FILTER = 100,
    MENU_FILTER_LP = 101,
    MENU_FILTER_UNDO = 108,
    ID_ToGray = 200,

    // it is important for the id corresponding to the "About" command to have
    // this standard value as otherwise it won't be handled properly under Mac
    // (where it is special and put into the "Apple" menu)
    MENU_HELP_ABOUT = \texttt{wxID\_ABOUT}
};
```

The enumeration table assigns values to events that will be called during the use of a program. It’s how the computer keeps track of events that occur during the execution of a program. Of course, if we needed to have an enumeration for the LPF (line 10 in the above code), we’ll need one for the HPF.
That’s it for that file.

Since we just modified a header file, it makes sense to now modify the corresponding .cpp file. Open BME465_Template.cpp. You’ll notice straight away that there is quite a bit more going on in this file than there was in the header file. Keep in mind, you are only looking for very specific things right now; you will only be adding a few lines of code. Notice right off that we include the header file and two of the icons - the lp.xpm and the undo.xpm. Remember the icons on the GUI? This is where we include them.

The first location where we see reference to the LPF is in the event table. You will basically just copy and paste the line of code that deals with the LPF and modify the copied line to correspond to the HPF. The event table is responsible for keeping track of which events have been instantiated.

```cpp
BEGIN_EVENT_TABLE(MyFrame, wxFrame)
    EVT_MENU(MENU_FILE_OPEN, MyFrame::OnLoad)
    EVT_MENU(MENU_FILE_QUIT, MyFrame::OnQuit)
    EVT_MENU(MENU_FILE_ABOUT, MyFrame::OnAbout)
    EVT_MENU(ID_ToGray, MyFrame::OnToGray)
    EVT_PAINT(MyFrame::OnPaint)
    EVT_MENU(MENU_FILTER_LP, MyFrame::OnFilter)
    EVT_MENU(MENU_FILTER_UNDO, MyFrame::OnFilter)
END_EVENT_TABLE()
```

The next listing is in the frame constructor. We need to append the frame to allow the HPF to be added to drop-down as well as an added icon (if you choose to add an icon).

```cpp
//build filterMenu
    filterMenu->Append(MENU_FILTER_LP, _T("&Lowpass Filter\tAlt-L"), _T("Lowpass Filter"));
```

As before, copy, paste, modify. Starting to see a pattern here? Scrolling down you’ll see a a toolbar menu. The toolbar is that strip in the window frame where icons appear. If you want an icon for the HPF, you’ll need to add it here.
#if wxUSE_TOOLBAR
wxToolBar *MyToolBar = new wxToolBar(this, wxID_ANY);

MyToolBar->AddTool(MENU_FILTER_LP, lp_xpm, _T("Low Pass Filter"));
MyToolBar->AddTool(MENU_FILTER_UNDO, undo_xpm, _T("Undo"));

MyToolBar->Realize();
SetToolBar(MyToolBar);
#endif

At the very end of this file, you'll see the most important part of the process. You may have noticed that there is a general structure to the code where there is a the class constructor followed by an object. For instance, at the end of the code you will see:

void MyFrame::OnFilter( wxCommandEvent& event )
{

  if( pImage == NULL )
  {
    wxMessageBox("Image is not loaded yet!", _T("Error"), wxOK | ←
    wxICON_INFORMATION,this);
  }
  else
  {
    wxImage *Filtered = NULL;

    switch(event.GetId())
    {
      case MENU_FILTER_LP: Filtered = LowPass(pImage); break;
      case MENU_FILTER_UNDO: Filtered = copy(masterImage); break;
    }

    pImage = Filtered;
  }
  Refresh();
  return;
}

The line void MyFrame::OnFilter( wxCommandEvent& event ) is basically saying you would like to create an object with the properties of the MyFrame class. Said another way, it is an object of the MyFrame class. This is where the object in object-oriented programming comes from. It is the same syntax used in JAVA programming. For now, you simply need to know that you can create a class then
create an object of that class. We will deal with creating other classes in later labs.

Copy, paste, and modify the line that deals with calling the LPF (line 15 in the above code.)

Now open the image_processor.hpp file. See the line that deals with the LPF? Copy, paste, and modify. When functions are declared in the header file, it is typically referred to as prototyping. You make a prototype function in the header file to tell the compiler that it can expect to see a function with the same name in the .cpp file. This is a hold over from the C programming language.

Almost done! Now open the image_processor.cpp. This is where the actual function performs the operation. You’ll see the low pass filter function, along with a number of others. Right now, focus on the low pass function. We’ll talk about everything else that is happening in this file during the lab period but the part that is of immediate concern is this snippet of code:

```c++
for(x=0;x<width;x++)
{
    for(y=0;y<height;y++)
    {
        if(x != 0 && y != 0 && x != (width -1) && y != (height -1))
        {
            unsigned long index;
            index=(unsigned long)y*width+x;
            double value=1.0/9.0*
( float(pTemp[index-width-1])+float(pTemp[index-width])+float(pTemp[index-width+1])
+float(pTemp[index-1])+float(pTemp[index])+float(pTemp[index+1])
+float(pTemp[index+width-1])+float(pTemp[index+width])+float(pTemp[index+width+1])
);
        pResult[(y)*width+(x)]=(int)(value);
    }
    else
    {
        pResult[y*width+x]=pTemp[y*width+x];
    }
}
```

We are starting at the edge of the image, creating a 3 x 3 box around each pixel, summing up those nine values, and dividing by 9, i.e. taking an average. We then move to the next pixel. That’s why we need two loops: one for the width of the image and one for the height. While we are going to copy and paste this whole function, we will only modify this part of it.
It is critical that you continue with the same naming convention you used with each step along the way. If you went the easy route, you would have just changed the word "Low" to "High". But you could have named it anything. As long as it is consistent, it will work just fine.

Now, you need to think about the mathematics of creating a high pass filter. You want to emphasize changing values, because when a value from one pixel changes considerably from one pixel to another, that is considered high frequency, just as if the image didn’t change much from pixel to pixel would be considered low frequency. In fact, if it didn’t change at all, it would be DC.

At the end of all your modifications, you will need to build your project. This step handles the compiling and linking all in one step. If all goes well, there will be no errors. You may have a warning or two - generally that is OK. Errors, however, need to be fixed. It’s usually something trivial, like forgetting a semicolon at the end of a statement or not declaring a variable before you’ve used it. If you get an error, the compiler will usually tell you approximately where it ran into trouble. Try to fix your errors before asking for help, it will make you a better programmer.

![Image of MinGW Developer Studio](image.png)

Figure 1.3: Building your project is necessary before being able to view your new additions.

### 1.3 Summary

We have taken an extended route to make sure you understand how to make changes to the code to perform different functions. In future labs, we will spend less time on syntax and much more time on algorithm development. If you are having trouble trying to figure out how to make a high pass filter,
The compiler had an issue with a conversion from a floating point number (float) to an integer (int).

Still, the compiler linked all the files together and successfully built the project.

Figure 1.4: Building, compiling and linking your project.

don’t worry. The instructor or TA will help you. Also, we have glanced over some topics in the manual which we’ll cover in the lab as well as the lecture. By the end of the course, you’ll be very competent C++ programmers.

1.4 Questions

1) What is the function of a header file?

2) In the C++ programming language (as well as C) when a command or file is in all capital letters it implies a MACRO, a set of rules or instructions that determine how to return an output. In our header files, the name of the header is in all capital letters in the definition check. Why?

3) What does your highpass filter do and why would you ever need to use it?
Lab 2: Nonlinear Filters

PURPOSE: To introduce filter types that are order dependent.

GOAL: Whereas linear filters can be applied repeated or in different combinations with respect to the order of operation without affecting the final result, nonlinear filters do not offer the same flexibility. A lowpass filter followed by a highpass filter will produce the same result as a highpass filter followed by a lowpass filter. However, a lowpass filter (linear) followed by a median filter (nonlinear) will not produce the same result as a median filter followed by a lowpass filter.

In this lab, we will create a handful of nonlinear filters including the median filter, the minimum filter, and the maximum filter.
2.1 The Median Filter

The median filter is ubiquitous in image processing. It is often called an edge preserving filter due to its ability to filter without smearing or blurring edges in the image. This, of course, is a highly desirable feature as it allows one to remove speckle noise without affecting the rest of the image.

The key to the median filter is sorting the values in the 3 x 3 filter mask. Once upon a time, this required what is called a "bubble sort", so-called due to the process. The first value is held in a variable and tested against the second. If the second value is lower than the first, it becomes the tested value and the process is repeated until the lowest value "bubbles" to the top. Almost every standard math library includes a sort function. While we could go through the steps of actually sorting the values, we can simply include the a math library and call the sort function.

The median filter simply takes the middle value of the 9 sorted values and returns the value to the pixel. Likewise, the minimum filter takes the lowest value and the maximum pixel takes the largest.

In actuality all three filters are only one filter. This is the perfect place to enact a switch-case statement. You’ve probably already noticed this type of statement in the code already. In case you missed it, this is basic gist:

```cpp
std::vector<int> valArray(9);
double value;
for (x = 0; x < width; x++) {
  for (y = 0; y < height; y++) {
    unsigned long index;
    index = (unsigned long)(y * width + x);
    if (y!=0 && x!=0 && x!= (width -1) && y!=(height -1)) {
      valArray[0] = pTemp[index - width - 1];
      valArray[1] = pTemp[index - width];
      valArray[2] = pTemp[index - width + 1];
      valArray[3] = pTemp[index - 1];
      valArray[4] = pTemp[index];
      valArray[5] = pTemp[index + 1];
      valArray[6] = pTemp[index + width - 1];
      valArray[7] = pTemp[index + width];
      valArray[8] = pTemp[index + width + 1];
      sort(valArray.begin(), valArray.end());
    }
  }
}
switch (type) {
    case MEDIAN_FILTER:  value = valArray[4];  break;
    case MINIMUM_FILTER: value = valArray[0];  break;
    case MAXIMUM_FILTER: value = valArray[8];  break;
}
if (value > 255)
    value = 255;
else if (value < 0)
    value = 0;
pResult[index] = (int)value;
else {
    pResult[index] = pTemp[index];
}
}

We’ll discuss all the things that need to be included to use the functions in this code.

2.2 Questions

1) The term nonlinear filter comes from the basic premise that the output is not a linear combination of the input. Discuss why this is a better alternative for specific types of filtering applications. (Hint: Discuss those applications!)
2) Besides the three types of nonlinear filters referenced in this lab, discuss other types of morphological nonlinear filters.
3) Why might biomedical image processing benefit more from median filtering than, say, facial recognition image processing?
4) In words, explain how a bubble sort works.
Lab 3: Edge Detection and Thresholding

PURPOSE: To reinforce the lessons in Lab 1 and 2 by creating new filters with specific functionality.

GOAL: In medical imaging, we are often most interested in boundary between two objects, whether soft tissue or bone. In the case of soft tissue, we may be interested in finding a mass, whereas the case of bone we may be interested in finding a break or a joint injury. Enter the techniques of edge detection and thresholding.

In this lab, we will repeat many of the steps of labs 1 and 2 while covering the mathematic principles behind edge detection. Thresholding is as it sounds: pick an arbitrary value and if the pixel is greater than that value, do something. The "something" is what we will emphasize.
3.1 Edge Detection

Consider the image in figure 3.1. It should be clear that with the edges highlighted, structures become more easily identifiable. (Incidentally, the split-screen view is something you may want to consider adding to your project as part of the final version - hint.)

![Image of edge detection](image)

Figure 3.1: Making edges more pronounced aids detection and classification.

While this may not be the exact result someone is seeking, it does highlight an important point: in biomedical image processing, edges are very important.

Often in image processing, we are interested in detecting the edges of a region of interest (ROI). There are a number of ways to perform such detection, but ultimately it remains a function of determining the two sides of the edge. The Sobel operator for instance uses the approximation to the derivative to find difference in gradients of a particular sized mask. Other similar methods include Robert’s Cross and Prewitt’s method. Another technique is to look for zero crossings after the application of a particular type of filter, as this would indicate a transition in frequency. For instance, if there were an image that was one color (low frequency) over most of the image and only one region with a different color, applying a low pass filter would reveal zeros at the point where the two colors met, which is the edge. All of these algorithms are straight forward to implement and require simply the construction of the appropriate mask to approximate the mathematics. As an example, the Sobel operator can be set up as two masks, one to handle the vertical edge detection and one to handle the horizontal (figure 3.2).
Figure 3.2: Again we have our standard filter mask except this time we are scaling the values on either side of the pixel. If it is an edge, one side will be validated, the other will be negated.
As before, you will need to make a new filter for the Sobel operator. If Lab 1 is a distant memory, just review it - the steps are identical. To help you with the code a bit, here is the snippet:

```c
for(x=0;x<width;x++)
{
    for(y=0;y<height;y++)
    {
        if(y!=0 && x!=0 && x!=(width-1) && y!=(height-1))
        {
            unsigned long index;
            index=(unsigned long)y*width+x;
            double value=
            (abs(pTemp[index-width-1]*(-1) + pTemp[index-width+1]) +
             pTemp[index-1]*(-2)+pTemp[index+1]+
             pTemp[index+width-1]*(-1)+pTemp[index+width+1]) +
             pTemp[index-width-1]*(-1)+pTemp[index+width-1]+pTemp[index+width]*2+pTemp[index+width+1]) +
             pTemp[index+width+1]*(-1)+pTemp[index+width-1]+pTemp[index+width]*2+pTemp[index+width+1])
             )
            ;
        
        if(value<0)
            value=0;
        if(value>255)
            value=64;

        pResult[y*width+x]=(int)value;
    }
    else
    {
        pResult[y*width+x]=pTemp[y*width+x];
    }
}
```

### 3.2 Thresholding

Thresholding simply provides a way to make an image that was once composed of many values (i.e. 256 possible values for an 8 bit per pixel image) have only a handful of possible values. The idea is simple: if a pixel is greater than a certain number, say 128, we force it to be 256, otherwise it is 0. This means that a grayscale image will now be black and white. Of course, we could just as easily make the image 4 colors, say black, dark gray, light gray, and white, we’d just need to add several more conditions.

The images we are working with are all converted to 8 bit images. That is, the values of the pixels can only take on values between 0 and 255. You’ll notice, if you haven’t already, that when we have
conditional statements in our code (the if statements), the numbers are bounded by 0 and 255. In the sobel edge detector, we set the pixel value to 64 if it was over 255 and we set it to 0 if it was less than 0. The reason for that is that we were multiplying by 1 and -1 then summing up the three value on either side of the pixel. That is how we determined if the gradient was significant enough to be an edge.

For our purposes, we’ll create a filter called binarize where we will simply set the pixel value to 255 if it’s greater than some value (you pick the number, it’s your project) and 0 if it is less than. You may need to figure out what to do about the condition when the pixel value is exactly equal to your threshold.

Next lab we will use a slider bar to allow the user to determine what value the threshold for binarizing the image should be.

3.3 Questions

1) Earlier in the semester, you created a high pass filter that had the tendency to perform edge detection. How is this edge detection algorithm different than that initial attempt at a high pass filter?
2) Are there any similarities between edge detection and binary filters? If so, what are they? If not, why not?
3) While binarizing implies making the image black or white (i.e. two possible outcomes) thresholding can be many different shades of grey or even color. We generally call this color mapping. Describe a situation that would benefit from having the output of a thresholding filter be several different shades of grey.
Lab 4: User Input - Sliders

PURPOSE: To introduce fundamental aspects of user interface coding.

GOAL: In this lab, you will take the critical step of allowing the user to have much more control over the image processing routines. While allowing the user to have access to filtering commands is important, variable input is key to the graphical user interface experience. All good image processing GUI’s allow the user to select their own level of input. Having many canned options is nice but not all image processing tasks will fit into a pre-selected group of action items. In this lab, you will make a slider bar which automatically updates the program with the new value on the slider to vary the way in which the image is processed. While this lab is focused on slider bars, all user inputs are handled in a very similar fashion.

At the end of this lab, you will be able to create a slider and make the value dynamically available to the program for real-time processing.
4.1 Sliders

While not terribly difficult to make user input available to the program in real-time, there are a great many pieces of that all need to come together in the right combination to do it successfully. Luckily, the wxWidgets toolbox takes care of much of the event handling. Still, as you will see, there is work to be done to make things work in a way that is both functional and intuitive to the user.

The slider is perhaps the most friendly of all user inputs. It provides a range of values so the user doesn’t have to keep entering numbers and the feedback is almost instantaneous. That is, the user can tell instantly what the effect of increasing or decreasing values has on the image.

On the website (http://www.ele.uri.edu/courses/bme464/f11/) under "Download project files" download the following two files: sliderFrame.hpp and sliderFrame.cpp. Save them into whatever folder you’re using for your project and add them to the project.

We will be going through these files in lab so that you understand how each file functions.

Next, we need to incorporate these files into the program, which is markedly different and more involved than simply adding them to the project. We will continue to follow the convention of modifying the template header file followed by the template source file followed by the image processing header file and finally the image processing source file.

First, we need to include the sliderFrame.hpp in the template header file.

```cpp
#include sliderFrame.hpp
```

(a) Right click the "Source Files" folder to add the sliderFrame.cpp file.

(b) Right click the "Header Files" folder to add the sliderFrame.hpp file. (Note the .cpp file appears in the Source Folder above.)

Figure 4.1: Add both files to their respective folders.

We will be going through these files in lab so that you understand how each file functions.

Next, we need to incorporate these files into the program, which is markedly different and more involved than simply adding them to the project. We will continue to follow the convention of modifying the template header file followed by the template source file followed by the image processing header file and finally the image processing source file.

First, we need to include the sliderFrame.hpp in the template header file.
Next, we need to append the enumeration table in the template header file with event enumeration types. (We’ll talk a bit about enumeration in lab.)

```c
enum {
    MENU_FILTER_2BIN = 110, 
    SLIDER_ID = 500, 
    BIN_SLIDER 
};
```

Notice how the type BIN_SLIDER is not assigned a number. You can if you wish, but it’s not relevant here since all we want to do is check to see whether the slider has been called, not necessarily to keep track of events.

In the public section of the header file (again, we’ll talk about public and private in lab), add the prototype functions for the slider functions we’ll call in the template source file.

```c
void makeSlider(int sliderID, int deflt, int low, int high);
void sliderHandler(void);
void sliderWindowCloses(void);
```

In the private section of the template header file, add the instantiation of the the sliderFrame, the actual slider itself, and variables that hold the slider ID (in case you want more than one slider) and the value the slider bar will return.

```c
MySliderFrame *sliderFrame;
wxSlider *slider;
int sliderID;
int sliderValue;
```

We have created a member of the sliderFrame class (which is itself a member of the wxFrame class), a slider which is a member of the wxSlider class, and two integers for the slider ID and the slider value. The star we’ve already talked about and just refers to the fact that we have created a pointer.

That’s all for the BME465_Template.hpp. Now we move on to the BME465_Template.cpp.

While it is not necessary to include "sliderFrame.hpp" because it was included in BME465_Template.hpp, it won’t hurt to include it here. Plus, it will help reinforce the habit that when you work from header file from source file you should always be conscious of including your header files.

What you do need to include is the event that will trigger the binarize filter:
EVT_MENU(MENU_FILTER_2BIN, MyFrame::OnFilter)

Remember that we will continue to call the filter everytime we slide the slider so it is critical that we keep track of the events, i.e. the calls to the function.

Of course, we’ll need to append the drop down list from the filter listing so we can call the binary filter. You’ll find this in the MyFrame constructor.

filterMenu->Append(MENU_FILTER_2BIN, _T("Binarize"), _T("Binarize"));

(You may chose to call your binary filter something else which is perfectly fine.)

At the end of the MyFrame constructor, you’ll see a little note prompting you to set the slider stuff to 0. This is where you will null the pointers we created in the .hpp file.

sliderFrame = NULL;
slider = NULL;

Recall that both sliderFrame and slider were declared as pointers not scalars so we can’t ”zero” them out. We need to null the pointer, which is to say, the pointer is empty.

Now, on the function calls. After the constructor, you’ll see a commented section where you can place the slider frame constructor. First, we need to make the frame where the slider will be placed.

```cpp
void MyFrame::makeSlider(int sID, int deflt, int low, int high) {
    if (sliderFrame != NULL) {
        delete slider;
        delete sliderFrame;
    }
    sliderID = sID;
    sliderFrame = new MySliderFrame(_T("Slider"), wxPoint(10, 10), wxSize(200, 200));
    sliderFrame->MySliderSetParent(this);
    slider = new wxSlider(sliderFrame, SLIDER_ID, deflt, low, high,
                          wxDefaultPosition, wxSize(160, wxDefaultCoord),
                          wxSL_AUTOTICKS | wxSL_LABELS);
    sliderValue = slider->GetValue();
    sliderFrame->Show(TRUE);
}
```

We’ll go through each line of this in lab. Next, we need to construct a handler that will fetch the value from the slider bar and make an initial call to the binarize filter.
void MyFrame::sliderHandler() {
    sliderValue = slider->GetValue();
    switch (sliderID) {
    case BIN_SLIDER: pImage = binarize(masterImage, sliderValue); break; }
    Refresh();
}

Finally, we need to create a function to close the slider frame, unless you want it to hang around even after you are done sliding.

void MyFrame::sliderWindowCloses(void) {
    sliderFrame = NULL;
    slider = NULL;
}

What is particularly important to remember here is that each of these functions, makeSlider, sliderHandler, and sliderWindowCloses, are all members of the MyFrame class. More on this in the lab.

Finally, all that’s left to do in the template source file is to add the function to our OnFilter call.

case MENU_FILTER_2BIN: { makeSlider(BIN_SLIDER, 128, 0, 255); Filtered = binarize(masterImage, sliderValue);} break;

We are now ready to move on to the image processing header file. Just like in previous labs, we simply need to create the prototype function.

wxImage* binarize(wxImage *pImage, int threshold);

Remember, if you changed the name of the function call back in the template file, you’ll need to be consistent with the naming here.

The last thing you’ll need to do is to write the actual function in the image processing source file. You’ve done this several times already so you probably don’t need much help. However, consider the following line of code:

pResult[index] = (pTemp[index] >= threshold) ? 255:0;

This one line of code is a conditional evaluation. It says if the value in the array pTemp at location index is greater than or equal to threshold, the value 255 is returned to pResult, otherwise 0 is returned. There are less artful ways of writing this, but this is probably the most efficient.
4.2 Questions

1) In the previous lab, it was suggested that thresholding could map to multiple different shades of grey or color. In words, how would you go about making the slider bar control the number of shades in an image?

2) Why did we create two new files to add to the project? That is, couldn’t we have just added all the code to the existing files? And if so, why didn’t we do that?

3) Describe a situation where being able to actively select a threshold value for a binary filter might come in handy.
Lab 5: Interactive Features - Area Calculation

PURPOSE: To begin the process of extracting meaningful data from images based on user input, which is fundamentally different from imposing input onto the data.

GOAL: There are many times in non-invasive imaging where the underlying pathology is known but the degree to which the pathology has advanced is unknown. This could be the result of shrapnel or debris from a violent impact, a metastasizing tumor, or simply a broken bone that needs to be reset. Being able to quantify the area, or volume, is a critical first step in determining a course of treatment. This lab will introduce the basic concepts necessary to allow a user to define points using a mouse click, fill in the line between those points, and calculate the enclosed area when the geometry is defined.

5.1 GUI Input: Mouse Clicks

As you are all aware, the mouse (or laptop touch pad) has become a virtually indispensable as a user input device. The ability to scroll anywhere on a screen using an input device is the basis for every capacitive touch-screen device. It turns out that controlling that input, whether it is getting x-y coordinates of the cursor or the pixel value at that point, is really quite simple. Again, we have wxWidgets to thank for many of the mouse commands that make this a much more manageable affair. Tiresome though it may be, we start with headers and work to source code. The difference here is that we will confine all of our work on the two template files. No changes to the image processing files need to be made since we are not doing anything to the image, simple extracting information from it. In the template header file, you’ll need to add some variables for us to access.

```c
//AREA PROTOTYPES
void OnAreaCalculation(wxCommandEvent& event);
void OnLButton(wxMouseEvent& event);

//AREA VARIABLES
```
int areaindex;
wxPoint* areaborder;
int pointNumber;
double dist;

bool bAreaCalculation;
bool bleftDown;

bool bRed;

//AREA
ID_AreaCalculation = 300,

We have two new functions OnAreaCalculation and OnLButton. (I hope it’s obvious what function
they will perform). We need to keep track of the number of points around the area we wish to define
(areaindex), the actual points that have been clicked (areaborder), the total number of points we
would like to limit people to use (pointNumber), and the distance from one point to another (hint:
point-distance formula). We also need to set some flags to indicate that area calculation has been
selected and when the left button is clicked. (Incidentally, it should be VERY clear how to do this
using the right button instead of the left!). The last bool (short for Boolean, i.e. TRUE or FALSE),
is to change the color of the outline you’d like to use. Default is black which doesn’t always work so
well with grayscale or black and white images. And of course, don’t forget to augment the enumeration
table to account for the new function. By now, you should know where all this goes.
The source file is a bit more involved since we have to actually write new code to perform the area
calculation. Plus we need to draw the lines on the image. Then finally, it would be nice if it returned
some information to us, either in the form of a new window or the status bar (the one at the bottom).
In the event table, there should be two entries:

EVT_LEFT_DOWN(MyFrame::OnLButton)
EVT_MENU(ID_AreaCalculation, MyFrame::OnAreaCalculation)

Note that the OnAreaCalculation event also takes in the MACRO ID_AreaCalculation. That should
be a familiar trick, i.e. the nonlinear filter. One way to handle the new functionality is to give it it’s
own menu heading in the menu bar

//AREA
menuProcess->Append(ID_AreaCalculation, _T("&Area Calculation\tAlt-A"), _T("⇒
Apply area calculation"));

but this is highly subjective. You may do whatever you’d like, i.e. add to one of the other menu items,
make an icon, etc. Also in the frame constructor, initialize all the variables we created in the header
file:
```c
//AREA
pointNumber = 99;
areaborder = new wxPoint[pointNumber];
areaindex = 0;
bAreaCalculation = FALSE;
bleftDown = FALSE;

SetStatusText("Out of Area", 1);
dist = 9999;
```

### 5.2 Area Calculation

The really big change occurs in the OnPaint function, where we need to account for the lines we’d like to draw. As a matter of convenience, you can also perform all the calculations right there:

```c
//AREA
if(bleftDown)
    pointNumber = areaindex;
else if(dist>=0 && dist<5){
    pointNumber = areaindex;
    int idist = (int)(floor(dist));
    SetStatusText( wxString::Format("The distance is %d", idist),0);
}

//AREA
if((areaindex >0) && (areaindex <=pointNumber)) // If in area calculation ↔ mode
{
    //wxPoint pt(event.GetLogicalPosition(dc));
    wxPoint* acborder_pt=new wxPoint[pointNumber]; // the points coord to ↔ be drawn
    for(int i = 0; i< areaindex;i++)
    {
        acborder_pt[i].x = dc.LogicalToDeviceX(areaborder[i].x);
        acborder_pt[i].y = dc.LogicalToDeviceY(areaborder[i].y);
    }
    dc.SetPen(*wxRED_PEN);
    dc.DrawLine( acborder_pt[pointNumber-1].x, acborder_pt[pointNumber-1].y,↔
                acborder_pt[0].x, acborder_pt[0].y);
```

University of Rhode Island, Department of Electrical, Computer, and Biomedical Engineering copyright © 2011 All rights reserved. Duplication for purposes of any kind is strictly forbidden.
float farea = 0;
int area;
int minborder = 9999;
for (int i = 0; i < pointNumber; i++){
    if (areaborder[i].x< minborder)
        minborder = areaborder[i].x;
}
for (int j = 0; j<pointNumber-1; j++){
    farea = ((float)((areaborder[j].x - minborder) + (areaborder[j+1].x - minborder)) / 2.0)
        * (areaborder[j+1].y - areaborder[j].y) + farea;
}
farea = ((float)((areaborder[pointNumber-1].x - minborder) + (areaborder[0].x - minborder)) / 2.0)
        * (areaborder[0].y - areaborder[pointNumber-1].y) + farea;
if(farea<0) farea = -farea;
area = (int)farea;
SetStatusText( wxString::Format("The area is %d", area),1);

areaindex = 0;
bleftDown = !bleftDown;
bred = FALSE;
pointNumber = 99;
dist = 9999;
}
delete acborder_pt;
}

I will be going over all this in lab. All that is left to do is build the two functions for which we made prototypes:

void MyFrame::OnAreaCalculation( wxCommandEvent& WXUNUSED(event) )
{
    bAreaCalculation = !bAreaCalculation;
    if( bAreaCalculation )
        SetStatusText("In Area", 1 );
    else
        SetStatusText("Out of Area", 1 );
    Refresh();
    return;
}

void MyFrame::OnLButton( wxMouseEvent& event)
wxClientDC dc(this);
wxPoint area_pt(event.GetLogicalPosition(dc));

SetStatusText( wxString::Format("(%d,%d)", area_pt.x,area_pt.y), 0 );

if(!bAreaCalculation)
    return;

if( pImage == NULL )
{
    wxMessageBox("Image is not loaded yet!", _T("Error"), wxOK | wxICON_INFORMATION, this);
    return;
}

areaborder[areaindex].x= area_pt.x;
areaborder[areaindex].y= area_pt.y+27;

if (areaindex>0)
    dist = sqrt( (pow((double)(area_pt.x- areaborder[0].x),2)+pow((double)(area_pt.y + 27 - areaborder[0].y),2)));
SetStatusText( wxString::Format("(%d,%d,%d)", area_pt.x,area_pt.y,(int)dist), 1 );
areaindex++;
Refresh();

I will be going over this in lab too!
(a) If all went well, you should be able to launch the interactive GUI.

(b) In the status bar, the text has change to "in area".

(c) Notice the color of the line and the status bar showing the location of the current point and the distance away from the first point.

(d) Now that we are within three pixels of the first point (zero actually) the program ends showing the final area. Pretty close!

Figure 5.1: Area calculation.
5.3 Summary

We have created a user interface that accepts user defined cursor inputs of locations on an image and allows the user to draw a polygon. The area of the polygon is calculated as the sum of multiple triangles defined by the points of the polygon. This is the basis of all image measurement tools, regardless of whether they use circles of varying radii or simple line lengths.

5.4 Questions

1) Why do we use booleans (TRUE, FALSE) to set flags?
2) Why did we put the calculation of the area in the OnPaint function?
3) We wrote a function called OnLButton, which does something when you click the left mouse button. What does the line event.GetLogicalPosition(dc) mean?
4) What controls the selection of the left button instead of the right button in the code? Hint: It’s not because we named the function OnLButton.
Lab 6: Accessorizing Your GUI - Icons and Buttons

PURPOSE: To give the GUI a more professional feel by creating intuitive action buttons and icons.

GOAL: In this lab, there will be no discussion or adaptation of the functionality of the application. While you still may choose the wxWidgets format for your final project, this marks the end of the C++ instruction. We have, however, been adding functionality for several weeks without providing the basic user interface buttons. When you look at the menu bar of any window, you will always see little icons that serve as interactive buttons. They could be a floppy disk for the ’save’ function (might be time to leave the floppy behind, but that is a different topic for a different time), it could be a manila folder for the ’open’ function, or even just a question mark for the ’help’ function. The point is the user has grown accustomed to using these buttons and we should know how to make them.

At the end of this lab, you will be able to create icons. Of course, there’s no telling what they’ll look like!

### 6.1 The GIMP example

There are many software packages out there for making icons. You may be aware of the .ico extension for icons but we will be using the .xpm (X PixMap) format (do a Google® for xpm to find out a bit more about the format). The reason for this is that while we can create the image using an image processing software environment, i.e. GIMP, the image is actually saved as an ASCII file. To verify this, simply right-click an XPM file and open it with Notepad.

But we’re here to create XPM files. Launch GIMP and create a new image file (figure 6.1.) You will need to change the size of the image to 16 by 16 pixels. This is the standard icon size.

Just like most photo editing tools, you can create multiple layers from which to work. While I highly recommend you explore this option, for now we’ll focus on the foreground and background. You’ll be surprised how much you can accomplish with just two layers. You certainly can work with black and white but lets just assume you have a bit more color in mind. You’ll notice the two rectangular palettes under all the icons; the top one is the foreground color and the background (figure 6.3a). Left click on the foreground and a color square will appear that will allow you to change the color (figure 6.3b).

Now if you want to make this the color of one layer of the image, simply fill the layer with the color.
Figure 6.1: Open a new image file in GIMP.

Figure 6.2: Make sure the image size is 16 by 16.

(a) Open the color palette by left-clicking. (b) Change the color from white something, say, gold.

Figure 6.3: Changing foreground and background colors.
You can click the rounded, two-sided arrow to send the foreground to the background. Simply choose another color for the foreground to make a design on the new background. REMEMBER, once you send the foreground to the background, the background becomes the new foreground. Instead of using the fill function, you can choose a pen or airbrush to make a pattern.

While this is certain an acceptable icon and worthy of a spot in your GUI, you may want to do something
a bit more polished. Here are few ideas.

(a) This is a floppy disk. While you may not know what it is, it has been the universal Windows® icon for save for many years.

(b) A question mark works well for the Help icon.

(c) This is a magnifying glass for Preview.

(d) Fading black to white for threshold.

Figure 6.6: Some ideas for icon design.
Lab 7: Introduction to MATLAB’s GUI Design Environment (GUIDE)

PURPOSE: To introduce the MATLAB® (Mathworks, Inc.) programming language GUI construction integrated development environment (IDE).

GOAL: MATLAB® is a very common and highly efficient programming tool that allows users to quickly prototype algorithms and GUIs. While it may not be able to compete with embedded systems and PC/MAC/LINUX native C-programming speed, what it lacks in execution speed it makes up for in design to proof-of-concept speed. That is, you can do really cool stuff very quickly with MATLAB. With that in mind, the goal of this lab is to show just how quickly someone can make an image processing GUI.

7.1 GUIDE

GUIDE is MATLAB’s graphical user interface - integrated development environment. It is very easy to use and has extensive user support through MATLAB’s online File Exchange. (Incidentally, there’s a really great (FREE) CT/MRI reading GUI available through UCLA http://www.ee.ucla.edu/~spapl/CTMRedit/index.html). Often, you will find free code on the Exchange that does much of what you want. You can download it, keep what you want, and modify it to do exactly what your project needs to do. Sound familiar?

We will spend a brief period of time making a very simple GUI that allows you to load an image and perform lowpass filtering. It will take about thirty (30) minutes. First, start by opening MATLAB and typing guide on the command line (figure 7.1). Choose the Blank GUI option. It will look like figure 7.2.

There are a number of predefined blocks on the left side pane. Familiarize yourself with all of them at some point - maybe later after lab. For now, we’ll concentrate on two but the ideas will be fairly straightforward to apply to all of the functions. First, create a set of axes by choosing the button that looks like a graph (figure 7.3). Stretch it out but leave room at the top for a couple of push buttons. Go back to the functions to the left and choose a Push Button (it’s the one at the top-left that reads
Figure 7.1: Open the GUIDE editor in MATLAB.

Figure 7.2: Blank GUI template.
Place two of these right over the top of the image. We will make one of these push buttons the Load button and the other will be our ever-popular low pass filter. Right click on either of the buttons and choose the Property Inspector option to rename the buttons something more appropriate, say Load Image and Low Pass Filter. You can do this with the String field (figure 7.4).

Figure 7.3: Layout the axes for the image.

If you’d like, go ahead and press the green Run arrow at the top of the GUI to see what it will look like. Should be something similar, though maybe not exactly, as figure 7.5. You will be prompted to save the file. You can leave it 'untitled' or name it. Either way, MATLAB will generate the code version of this building block version and display it. Don’t be alarmed, we’ll get to it.

Now comes the only real tricky part. We need to establish the callback functions associated with the push buttons. Again, right click one of your buttons and you will see View Callbacks. Select Callback and it will take you to the portion of the text file (which we’ll refer to as an ’m’ file, due to the extension type), figure 7.6.

The code for the two buttons should be modified as follows (my buttons are reversed in number, yours may be different):
Figure 7.4: Rename the push buttons.

Figure 7.5: The basic template.
Figure 7.6: Access the code associated with the push button.
% —— Executes on button press in pushbutton2 — **this** is the LOAD button
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton2 (see GCBO)
% eventdata reserved — to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

[FileName,PathName] = uigetfile({'*.'},'Load Image File');

if (FileName==0) % cancel pressed
    return;
end

handles.fullPath = [PathName FileName];
a. b. Ext] = fileparts(FileName);
availableExt = {'.bmp','.jpg','.jpeg','.tiff','.png','.gif'};
FOUND = 0;
for (i=1:length(availableExt))
    if (strcmpi(Ext, availableExt{i}))
        FOUND=1;
        break;
    end
end

if (FOUND==0)
    h = msgbox('File type not supported!','Error','error');
    return;
end

RGB = imread(handles.fullPath);
handles.im = RGB;
axes(handles.axes1); cla; imshow(RGB);
guida(hObject,handles);%%VERY IMPORTANT FOR SHARING!!

% —— Executes on button press in pushbutton1 — **this** is the LOW PASS
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject handle to pushbutton1 (see GCBO)
% eventdata reserved — to be defined in a future version of MATLAB
% handles structure with handles and user data (see GUIDATA)

mask = fspecial('average');

filtered = imfilter(handles.im,mask);
axes(handles.axes1); cla; hObject = imshow(filtered);
handles.im = filtered;
guida(hObject,handles);
I will be going over this code in lab pretty much line by line. Once you have made the changes to the code, run the GUI from the m file. Load an image and keep clicking through the Low Pass Filter. Does it work?

![GUI Image](image.png)

Figure 7.7: The low pass filter.

### 7.2 Summary

While there is a certain startup cost associated with becoming familiar with MATLAB, it may be a good bet that it is much easier to prototype an image processing GUI using MATLAB than it is to using C++. The big drawback here, of course, is that in order to use the GUI, you need to have a copy of MATLAB. We’ll talk about this a bit.

### 7.3 Questions

1) What is a handle? Is it similar to a pointer? If so, how?
2) What is a callback function? What is the C++ corollary?
3) Compare and contrast the two methodologies for creating GUIs you have learned to date.
Lab 8: The MATLAB® Image Processing Toolbox (IPT)

PURPOSE: To introduce several of the more powerful image processing tools in the IPT.

GOAL: There is more to image processing than filters and edge detection. Often, we are interested in being able to use the relationships of regions in an image to determine something meaningful about the image. For instance, maybe you’d like to devise an automatic feature that detects circular regions in an image and colors them. The goal of this lab is to explore the regionprops function in Matlab.

While not terribly related to region properties, we will also explore the Radon transform specifically due to it’s ubiquity in medical imaging. This lab will conclude the organized lab portion of the course.

8.1 Region of Interest (ROI) Properties

Like most functions in MATLAB, regionprops offers a very detailed help document. On the command line type help regionprops and open the reference page in the help browser. Most of the image processing tools in this function will work only on black and white images. In MATLAB, there is a nice function called im2bw which converts color and grayscale images to binary images. This is essentially the same function you wrote in C++ in lab 3.

Pick any picture you’d like to use from our medical images (remember to load the image you’ll need to use imread) and perform a binarization filter using im2bw in MATLAB. Of course, you could use the C++ program to do it with your slider and save the image - if you’ve included a save function.

In figure 8.1, you’ll see the binarized version of the CTHemorrhage image, but the same principles apply to all images. The first regionprops function we’ll use is the Area parameter.

The nice property of the Area option in regionprops is that it will find all the areas of the image that have a connectivity. That is, it counts the 1’s in the image and returns a list of all the regions in the image that have 1’s. There is no ranking performed at this step meaning that the number of connected pixels is returned as MATLAB finds them. The values are stored as a struct so if you want all of them, you have to collect them in an array, i.e.:
Figure 8.1: These are the general steps to binarize an image in MATLAB. `im2bw` takes in an optional argument to control the threshold setting. If you want 2 or more images displayed in MATLAB, use the `figure` command to open a new window. Otherwise MATLAB will overwrite the figure.

```matlab
>> stats = regionprops(BW, 'Area');
>> statsArray = [stats.Area];
```

Now we can do some selective processing. Suppose we want to find the region with most connected white pixels (that would be the area around the brain - the skull). We simply type

```matlab
>> CC = bwconncomp(BW);
>> idx = find([stats.Area]==max([stats.Area]));
>> BW2 = ismember(labelmatrix(CC), idx);
>> imshow(BW2)
```

and we have the portion of the image that we want (figure 8.2).

Now there are a number of things we can do. Of course, the next largest area of connected pixels would be the hemorrhage and we certainly could just ask MATLAB to give us the next one in the list (i.e. sort them and take the second entry.) However, images are simply matrices of numbers so we can add or subtract them. Further, binary images are 1’s and 0’s so we can do logical operations like `and`, `or`, `xor`, etc. We could type `sub = xor(BW,BW2);` and display the result and it will be everything except the skull. The only problem here is that it still has the text and other smaller connected white pixels. But we certainly can handle that easily with another call to `regionprops` and `Area` as the hemorrhage will certainly be the largest area now. There are a number of ways to handle this and no one way is necessarily better.
Figure 8.2: Effectively isolating the skull. Type `help bwconncomp` and `help labelmatrix` for more information about those functions. `ismember` is exactly that - it checks to see if one thing (value) is a member of another thing (array of values).

Figure 8.3

Suppose instead you’d like to color the top 5 connected regions different colors. This is very easy to do in MATLAB. You simply need to find the 5 largest regions and store their locations:
sorted = sort(statsArray, 'descend');
for k = 1:5
    idx(k) = find([stats.Area] == sorted(k));
end

Then simply re-label the image with the new index:

CC = bwconncomp(BW); %Find the connected pixels in the original
BW2 = ismember(labelmatrix(CC), idx); %Reduce the image to only those listed in idx
CC = bwconncomp(BW2); %Find connected pixels in the new image
L = labelmatrix(CC); %Lable them
RGB = label2rgb(L); %Make them all different colors
imshow(RGB) %Display

Figure 8.4

This will work on any image. Now, pick 5 additional regionprops properties to explore on your own.

8.2 Radon Transform

Again relying on the help document, look-up the Radon transform in MATLAB by typing help radon. The example they list is the Radon transform on a box. Type figure, imshow(I) to see it. Next, look at the inverse Radon transform by typing figure, imshow(iradon(R,theta)). Notice anything peculiar? Why?

Work through the examples on the Radon transform and inverse Radon transform in MATLAB and comment on what you see.