Due date: February 18, 2013. 11:59 PM (Madison time). You have three late days — use it at as you wish. Once you run out of this quota, the penalty for late submission goes as follows for each subsequent 24 hour period: {10%, 25%, 50%}. That is, if you submit this assignment on Sunday (2/17) at 4:00 AM in the morning, you lose two late days.

What to turn in:

1. Your submission should include your complete code base in an archive file (zip, tar.gz), example images you used to evaluate your implementation on (organized under different directories such as q1/, q2/, and so on), and a README describing how to run it.

2. A brief report (typed up, submit as a PDF file, NO handwritten scanned copies) describing what you implemented and known failure cases.

Notes from instructor:

- Start early!
- Stick with small images to start with to keep the running time manageable. There will be no penalty for (slow) running time. However, use this opportunity to learn about data structures that you think might be helpful. You may ask the TA or instructor for suggestions, and discuss the problem with others (minimally). But all parts of the submitted code must be your own. Matlab is slow with loops, and vectorized code is significantly faster, see http://www.mathworks.com/support/tech-notes/1100/1109.html.
- Submission instructions (for Moodle) and a number of images will be provided shortly. Remember that the performance of your submitted code on these ‘example images’ is not sufficient for full credit. Try thinking about and accounting for degenerate cases. You should generate new images to evaluate the robustness of your code. Data we use to grade your submitted code will be available to you after submission.

Problem 1

(Convex Hull, Perimeter, Distances, Dilate, We consider 8-connected for all problems, 20pts)

1. (10pts) Write a function called \texttt{imOut = myConvHull}(imIn). \texttt{imIn} will be a binary image (e.g., 0s or 1s) containing a single region (blob) as 1s. Your Matlab function should compute the convex hull of the region of interest outputted as 1s in image \texttt{imOut}. All other pixels in \texttt{imOut} will be zero. In addition, your function should return some basic statistics of the convex hull such as area (number of pixels) and diameter. You should write your own implementation for this functionality, usage of \texttt{regionprops}, \texttt{convhull}, \texttt{convhulln}, \texttt{qhull} from Matlab not permitted. If in doubt, ask!

As a precursor to this task, you must write a function, \texttt{myPerimeter()} to perform a perimeter extraction of the region, this should be called from within \texttt{myConvHull()}. The output of this function will be a \(m \times 2\) matrix, where \(m\) is the number of points (pixels) on the perimeter, \texttt{bwperim}, \texttt{regionprops} not permitted.

2. (5pts) The second part of this problem deals with computing the "distance" of every pixel in the image to its closest point on the perimeter – in the form of a function called \texttt{imOut = myDT}(imIn). The output from this process should be two colormap images: where the first corresponds to the distances for pixels outside the interior of the region and the second image will correspond to pixels which lie inside the interior of the region.
3. **(5pts)** The third part of this question is straightforward. Write a function \(\text{imOut} = \text{myDilate}(\text{imIn})\) that takes in an image dilates it with a structuring element incrementally (you are free to use \text{imdilate()} and \text{strel()} from Matlab). Compute and show perimeters of each of images (examples provided with this assignment) and also display their corresponding distance maps in a set of ten images. You can choose your any structuring element you want.

**Problem 2**

(Euler number, Region finding, Labeling, 20pts)

1. **(10pts)** Given a binary image, write a function which counts the number of connected components and the number of holes: \([\text{numObjects}, \text{numHoles}] = \text{imEuler}(\text{imIn})\). A example is shown in Figure 1.

2. **(10pts)** Write a function \(\text{imOut} = \text{myRegionFinder}(\text{imIn})\) to extract distinct regions from a binary image. The image will have more than one blob(s), your function must recognize the different blobs as distinct regions. The output of the function should be a gray scale image (same size as the input) where pixels in the same blob are assigned the same unique intensity value. Again, do not use \text{bwlabel}, \text{bwlabeln} explicitly but you are free to employ similar ideas as implemented in those functions. This should be based on a neighborhood system of your choice. (hint: look up the concept of “connected components”).

![Figure 1: The number of connected comments: 5, number of holes: 1.](image_url)

**Problem 3**

(Edge detector, Edge linking, 30pts)

1. **(15pts)** Write a gradient based edge detector. Your code should load in a gray-scale image (use \text{imread} and convert to a double array using \text{im2double}). You can display the image using functionality like \text{imagesc}. Once you have loaded in the image, you should smooth the image with a Gaussian filter and then compute horizontal and vertical derivatives using a derivative filter. The amount of smoothing is determined by the parameter \(\sigma\) of the Gaussian (which should be a parameter in your code). You can use \text{conv2} with the ‘same’ option to perform the required convolutions. Once you have computed the derivatives in the \(x\) and \(y\) directions, compute the gradient magnitude and orientation. Display the gradient magnitude as a image and the orientation using \text{quiver} functionality.

Bonus points: To get nice results, come up with ways to remove cluttered edges (like non-maximum suppression).
2. (15pts) We want to extract the outer boundary of each object contained in one image. First, use `edge()` function (or your own edge detector) to obtain the edge map. Then use `imview()` function to manually locate one pixel located on the outer boundary of each object. Next, apply any technique you can come up with to trace the boundary edge pixels. Try to get a boundary as complete and accurate as possible: it is OK if your results are not perfect (see below). Plot the final boundary you are able to detect and trace for each object. Discuss the success, failure, and possible ways for improvement in the report.

Note that boundary tracing is a non-trivial problem, involving many complicated issues such as noise, broken boundaries, ambiguous locations and directions, among others. Thus, do not worry if you are not able to get a ‘perfect’ result — we just want to see how hard you tried. We will grade this question on a curve.

Do not use the `bwtraceboundary()` function provided by Matlab image processing library. The idea is for you to gain familiarity with the algorithm through the experiments in this homework.

Problem 4

(Integral Images, 30pts)

In medical image analysis, one important application is to identify whether there is something interesting or abnormal in a given image. A common way to tackle this task is to extract local features from patches, followed by some statistical analysis. In this problem, we will compute certain simple summary measures from rectangular patches.

1. (10pts) Read the following paper before starting work on this problem. Paul A. Viola and Michael J. Jones, Robust Real-Time Face Detection. IJCV 2001. In your PDF report, summarize how you used ideas from this paper within your solutions to the following two sub-problems. Even if you ended up not implementing any functionality from this paper, you should nonetheless attempt to describe how you could have used certain concepts.

2. (10pts) Suppose we have a mask (divided into 2 rectangles) as shown in Figure 2: the summary measure we are interested in is the mean of the pixel intensities within the right rectangle subtracted from the mean of the pixel intensities within the left rectangle.

Write a function to compute the rectangular features: `feature = getFeature(imIn, i, j, m, n)`, where `imIn` is a gray-scale image, `(i, j)` is the pixel at which to evaluate the mask (see Fig. 2), and 2m and n give the width and height of the smallest rectangle as shown in Figure 2. The origin of the mask is defined to be the upper left corner.

After running your code on the example images, comment on your results. What can you say about the type of region that produces no response from the mask (i.e. it sums to approximately zero)? What can you say about the type of region that does produce a response from the mask?

3. (10pts) An important task for the problem of detection is how to choose the mask size. In this part, we will explore the influence of mask size with respect to rectangle feature.

Suppose for any fixed mask size, we want to compute rectangle features for masks at all possible positions. Write a function that compute the histogram for one mask size. `featureHist = getFeatureHist(imIn, m, n, numBins)`. `numBins` is the number of bins used in computing the histogram.

Plot five histograms with different mask size. If appropriate, summarize any observations you have about its behavior as a function of window size.
Figure 2: Illustration of integral image.