B490/659 Project 3: Edge and region-based recognition

Spring 2015
Due: Wednesday March 25, 11:59PM

Note that this assignment is due a few days after spring break. We expect the project to only take 2 weeks, so you have time to finish it before and after the break, but we are giving extra time to maximize your flexibility in managing your time. Please consult with your partner to develop a strategy; many people may be away for the break (including AIs!) and waiting until then to begin may be a recipe for disaster!

In class we discussed edge detection and image segmentation as two fundamental building blocks for computer vision. This project will give you experience in implementing these techniques for a simple but realistic problem.

We've once again assigned you to a team with another student in your section, using the preferences you gave us in your team feedback. We tried to accommodate as many preferences as possible. Please understand if we weren't able to accommodate yours, and you'll be able to change these teams in future assignments. You can once again find your assigned teammate(s) by logging into the IU Github website and looking for a repository called *userid1-userid2*-p3. Please contact them ASAP, and let us know if there is a problem (i.e., they have decided to drop the class).

We once again recommend using C/C++ for this assignment. We have not prepared skeleton code this time, but again recommend using CImg.For this project, you *may* use the image processing methods of the CImg class, instead of having to write everything from scratch. You may also use additional libraries for routines not related to image processing (e.g. data structures, sorting algorithms, etc.). Please ask if you have questions about this policy.

This project is a bit more open-ended than the first two, giving you some latitude to explore approaches that we have covered (or even not covered) in class. Your primary goal in this project is to get as good performance as possible on the recognition task we define (while still implementing, at a minimum, the specific steps discussed below). To reflect this goal, 90% of your grade will be based on completing the steps below and writing your report, while 10% will be based on the accuracy that you get, relative to that of other groups in your section (B490 or B659) of the class.

Please ask questions on the OnCourse Forum so that others can benefit from the answers.

Part 1: Getting started

Clone the github repository with the test images:

```
git clone https://github.iu.edu/cs-b490-b659/your-repo-name-p3
```

where *your-repo-name* is the one you found on the GitHub website.

Part 2: Detecting edges and circles

1. Implement an edge detector. A simple approach is to estimate partial derivatives in the x and y directions (I_x and I_y) using the Sobel operator, compute the magnitude of the gradient $I_m(x, y) = \sqrt{I_x(x, y)^2 + I_y(x, y)^2}$ at each pixel, and then threshold I_m to produce a binary image.
2. Implement a Hough transform to find circles in an image. A circle can be specified in terms of three

parameters: the row and column coordinates of its center and its radius. Your Hough accumulator space will thus be three-dimensional; to reduce the size of the space you can assume that we are only interested in circles having radius within a reasonable range, and you may want to discretize the space to further reduce its size. You may also want to reduce the size of the image to make computation less costly. As we discussed in class, in the Hough transform each image pixel votes for a set of points in the Hough space. Once the votes are accumulated, find peaks in the Hough space above some threshold to detect circles in the image.

Part 3: Finding circular regions

An alternative approach to that in Part 2 is to first run a segmentation algorithm, which finds areas of the image that have about the same color, and then choose the ones that have a circular shape.

1. Implement the following simple segmentation algorithm, based on the one we discussed in class. Given an image, define a graph where each vertex represents a pixel, and edges connect neighboring pixels (i.e., so that most pixels are incident on 4 edges, except for those on the image boundary). For each edge, define an edge weight given by the absolute color difference between the two pixels (i.e. $\sqrt{(r_1 - r_2)^2 + (g_1 - g_2)^2 + (b_1 - b_2)^2}$ for two pixels with RGB color values (r_1, g_1, b_1) and (r_2, g_2, b_2)). Define a set S of connected components, with each pixel initially being its own connected component. Sort all edges in increasing order of their weight. Then find the minimum-weight edge that spans two connected components. If this edge has weight less than a threshold, connect the two connected components together and update S. Keep iterating until all remaining edges have weight greater than the threshold.

2. Once you've implemented the basic segmentation algorithm, here are two simple modifications that may improve results significantly in practice. First, apply a Gaussian kernel to each RGB color plane of the image ahead of time, in order to reduce noise in the edge weights before segmentation. Second, instead of simply joining connected components based on edge weight, use a slightly more complicated rule. Consider the minimum edge $e = (v_1, v_2)$ connecting any two connected components C_1 and C_2, as before, but only join them if

$$w_e \leq \min(N(C_1) + \frac{k}{|C_1|}, N(C_2) + \frac{k}{|C_2|}),$$

where w_e denotes the weight of edge e, k is a constant parameter, $|C|$ denotes the size of component C (i.e., number of vertices in the component), and $N(C)$ denotes the maximum edge weight in the minimum spanning tree of C (i.e., the maximum over all edges that were used to create the component during the iteration so far).

3. For each region, evaluate how "circular" it is according to some rules of your choosing. A starting point might be to just check the aspect ratio (height of region over width) and fill factor (percentage of the region that is filled within the bounding box of the region). Chapter 2 of the second Burger & Burge book (listed on the OnCourse wiki) has some additional ideas. Use this result to select the circular regions of the image.

Part 4: Coin counting

Now combine the above techniques together to implement an object detector for a simple kind of object: U.S. coins. The goal is to accurately count the number of coins in a photo. We have included some sample test images (see examples in Figure 1). You also may want to use some of your own; testing vision algorithms

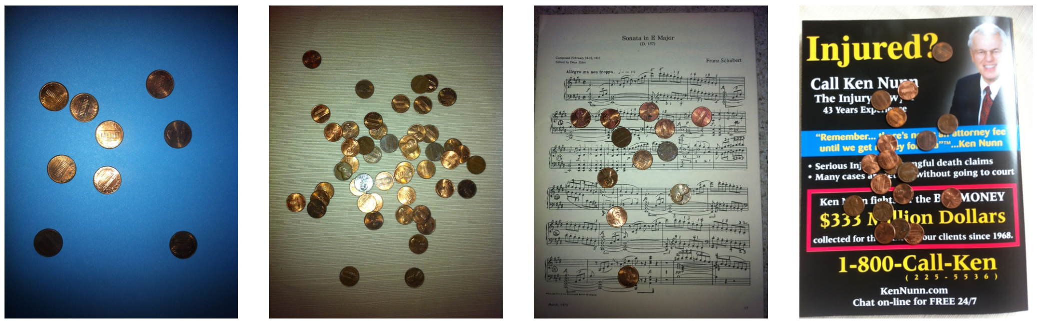

Figure 1: Sample test images.

on very simple synthetic images – e.g. white circles on black backgrounds created using PhotoShop – is often very helpful for debugging.

This task is purposely open-ended, with many details left unspecified. For example, you'll need to do some experimentation to find parameters and thresholds that work well for this task. You may need additional heuristics, like non-maximal suppresion to prevent the same coin from being detected multiple times. You may find that Part 2 works best for this task, or Part 3, or some combination of both of them. You may want to check that the average RGB color of a region is consistent with a true coin. As is usually the case in computer vision, it may not be possible to achieve 100% accuracy on the test images. Your goal is to do as well as possible, relative to the performance of the rest of the class. Note that we may use additional test images other than those we are providing to you, so you do not want to "overtrain" to this collection of images.

In addition to displaying the estimated number of coins for an image, your code should also output several intermediate images (which are useful both for your debugging purposes and our grading purposes):

- `edges.png`: Output of edge detector.
- `edges_detected.png`: Visualization of which circles the edge-based detector found. CImg has a method called `draw_circle` that may be helpful here.
- `regions.png`: Output of segmentation algorithm.
- `seg_detected.png`: Visualization of which circles the segmentation-based detector found.
- `detected.png`: Visualization of which coins your final detector found.

In your report, please describe your coin counting technique and present the recognition results for all of the test images. Also present quantitative results for each image, in terms of percentage error with respect to the correct answer (i.e., $|your\_estimate - correct\_answer|/correct\_answer$). We'll also give extra credit for additional work beyond that required by the assignment. For example, some of the test images include multiple kinds of coins; you might try calculating the total value of the coins instead of just counting them.

What to turn in

Make sure to prepare (1) your source code, and (2) a report that explains how your code works, including any problems you faced, and any assumptions, simplfications, and design decisions you made, and answers the questions posed above. To submit, simply put the finished version (of the code and the report) on GitHub (remember to `add`, `commit`, `push`) — we'll grade the version that's there at 11:59PM on the due date.

Important hints and warnings

Design decisions. You'll likely encounter some decisions as you write your programs that are not specified in this assignment. Feel free to make some reasonable assumptions in these cases, or to ask on the OnCourse forum for clarification.

Academic integrity. You and your partner may discuss the assignment with other people at a high level, e.g. discussing general strategies to solve the problem, talking about C/C++ syntax and features, etc. You may also consult printed and/or online references, including books, tutorials, etc., but you must cite these materials in the documentation of your source code. However, the code that you and your partner submit must be your own work, which you personally designed and wrote. You may not share written code with any other students except your own partner, nor may you possess code written by another student who is not your partner, either in whole or in part, regardless of format.