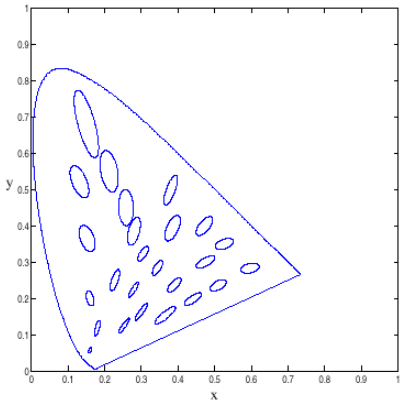
CS 378 Computer Vision
Problem set 2
Out: Thursday, October 2
Due: Tuesday, October 21, 11:59 PM

See submission instructions at the end of this document.

I. Short answer problems: clustering [30 points]

1. The figure to the right shows the variations in color matches on a CIE x,y color space. The center of each ellipse corresponds to the color of a test light, and the ellipse itself encompasses the scatter of lights that various human observers matched to that test color.

   You are designing a segmentation algorithm that will return image regions that are perceptually similar in color. Explain why using the CIE x,y color space together with k-means clustering may fail to capture very good groups. Assume a standard squared Euclidean distance for the k-means distance measure.

2. Consider the image on the right. Suppose we have successfully performed background subtraction in order to isolate one connected foreground component per fish, and now we want to use graph-based clustering to group together the different types of fish present in this scene. The input is the original image, plus the result of running background subtraction followed by connected components, so that each blob is associated with a single fish. The output should be clusters of similar-looking fish.

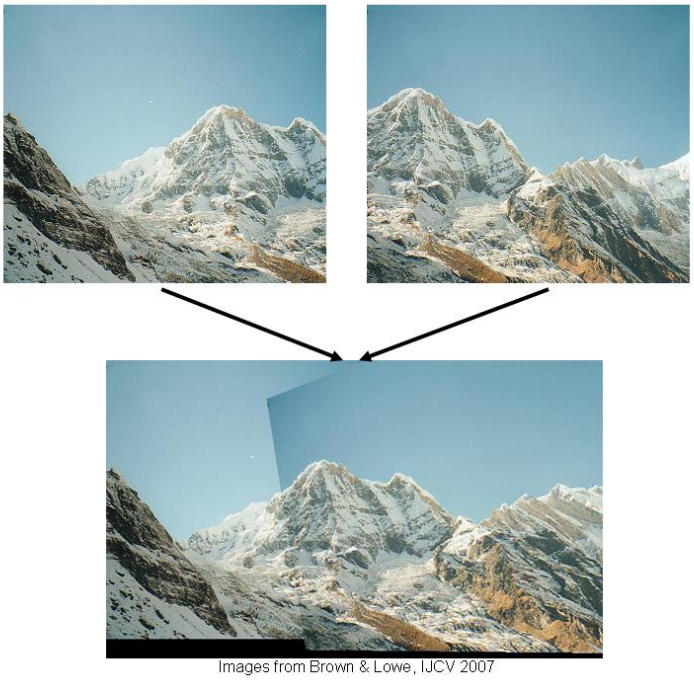
   

   a) To define the graph for this clustering problem, what are the nodes?

   b) Define a feature space and affinity measure that can be used to compute the edge weights for the graph. The affinity measure is a function that takes two features and returns their similarity score; it is used to assign the weight on an edge between two nodes. Be sure to choose features that will reflect the important differences between fish species in this scene.

3. What is the computational complexity of running the Hough transform to detect instance of a curve with P parameters on an image with E edge points?

II. Programming problem: image mosaics [70 points]



Images from Brown & Lowe, IJCV 2007

In this exercise, you will implement an image stitcher that uses alignment and homographies to automatically create an image mosaic. We will focus on the case where we have two input images that should form the mosaic, where we warp one image into the plane of the second image and display the combined views. This problem will give some practice manipulating homogeneous coordinates, computing homography matrices, and performing image warps. For simplicity, we'll specify corresponding pairs of points manually using mouse clicks.

Here are the main components you will need to implement:

- **Getting correspondences**: write code to get manually identified corresponding points from two views. Look at Matlab's `ginput` function for an easy way to collect mouse click positions. The results will be sensitive to the accuracy of the corresponding points; when providing clicks, choose distinctive points in the image that appear in both views.

- **Computing the homography parameters [25 points]**: write a function that takes a set of corresponding image points and computes the associated 3 x 3 homography matrix H. This matrix transforms any point p_i in one view to its corresponding homogeneous coordinates in the second view, p_i', such that $\lambda p_i = Hp_i'$. Note that p_i and p_i' are both 3-vectors. The function should take a list of $n \geq 4$ pairs of corresponding points from the two views, where each point is specified with its 2d *image* coordinates.

  We can set up a simple solution using a system of linear equations $Ax = b$, where the 8 unknowns of H are stacked into an 8-vector x, the $2n$-vector b contains image points from one view, and the $2n \times 8$ matrix A is filled appropriately so that the full system gives us $\lambda p_i = Hp_i'$. Let $H_{3,3} = 1$. Solve for the unknown homography matrix parameters. [Useful Matlab functions: '\' operator (help `mldivide`), `reshape`]

Verify that the homography matrix your function computes is correct by mapping the clicked image points from one view to the other, and displaying them on top of each respective image (`imshow`, followed by `hold on` and `plot`). Be careful to handle homogenous and non-homogenous coordinates correctly.

- **Warping between image planes [25 points]:** write a function that can take the recovered homography matrix and an image, and return a new image that is the warp of the input image using H. Since the transformed coordinates will typically be sub-pixel values, you will need to sample the pixel values from nearby pixels. For color images, warp each RGB channel separately and then stack together to form the output.

  The `interp2` function is useful for sampling all the positions required in one step. Note that transforming all the points will generate an image of a different shape / dimensions than the original input. Also, check for `NaN's` in the sampled output.

  To avoid holes in the output, use an inverse warp. Warp the points from the source image into the reference frame of the destination, and compute the bounding box in that new reference frame. Then sample all points in that destination bounding box from the proper coordinates in the source image. [Useful Matlab functions: `round`, `interp2`, `meshgrid`, `isnan`]

- **Create the output mosaic:** once we have the source image warped into the destination image's frame of reference, we can create a merged image showing the mosaic. Create a new image large enough to hold both (registered) views; overlay one view onto the other, simply leaving it black wherever no data is available. Don't worry about artifacts that result at the boundaries.

Tips:

- It can be useful when debugging to plot the corners and clicked points from one view on top of the second view after transforming them via H. Use `axis([minx, maxx, miny, maxy]);` to adjust the viewing window so that you can see all points. You will need the inverse of the homography matrix to transform "backwards".

- Be aware that Matlab's image (matrix) indices are specifed in (row,col) order, i.e., (y,x), whereas the `plot` and `ginput` functions use (col,row) i.e., (x,y).

- As usual, be careful with how images are cast for computations and display (`double` vs. `uint8`). In particular, be sure to pass the `interp` function a matrix of doubles for the image input.

- When collecting your own images, be sure to either maintain the same center of projection (hold the camera at one location, but rotate between views), or else take shots of a scene with a large planar component. In either case, use a static scene. Textured images that have distinctive points you can click on are good. Also ensure that there is an adequate overlap between the two views

After writing and debugging your system:

1. Apply your system to the provided pairs of images, and display the output mosaics. [10 points]
2. Also show at least one example using your own images. [5 points]
3. Explain how you could use RANSAC to compute the homography parameters in the presence of erroneous point correspondences. (Explain in words with a concise step by step description; you do not need to implement this.) [5 points]

For all examples, play around a bit with the choice of points for the correspondence pairs until you get a reasonable alignment.

III. [OPTIONAL] Extra credit [up to 10 pts each, max 20 pts]

1. Implement RANSAC for robustly estimating the homography matrix from noisy correspondences. Show with an example where it successfully gives good results even when there are outlier (bad) correspondences given as input. Compare the robust output to the original (non-RANSAC) implementation where all correspondences are used.

2. Refine the initial correspondences automatically by searching small patches near the clicked points for a good alignment, as measured by SSD between the pixels. Show an example where the refined points improve the mosaic alignment.

3. Do a creative combination: replace one surface in an image with an image of something else - for example – overwrite a billboard with a picture of your dog, or project a drawing from one image onto the street in another image, or stitch together two images from the same room where you or your friend appear in both. Use your imagination!

Submission instructions: what to hand in
Create a tar file with

- Your documented Matlab code. Include your name at the top of each file in a comment.
- A **single pdf** file containing the following
    - Your name at the top
    - Your answers to Part I, numbered.
    - A brief explanation of your implementation strategy for Part II: a short paragraph or two describing in English what you have computed.
    - Your responses and image results for each question in Part II, numbered. Insert image figures in the appropriate places.
    - (optional): any results and descriptions for extra credit portions.

1. Name the tar file with <username>_pset2.tar and submit via the turnin program: `turnin --submit harshd pset2 <filename>`

2. Also submit a hardcopy of the pdf file. You can submit the hardcopy in class. Otherwise, leave it at CSA 114.