Submission Requirements and Description (Very Important !)

Format requirements

(i) Please use the provided *Latex template* to write your report, and the report should contain your name, student ID, and e-mail address;

(ii) You should choose between Matlab and python to write your code, and provide a README file to describe how to execute the code;

(iii) Pack your **report**.pdf, **code** and **README** into a zip file, named with your student ID, like MG1833001.zip. If you have an improved version, add an extra '_' with a number, like MG1833001_1.zip. We will take the final submitted version as your results.

Submission Way

(i) Please submit your results to email cvcourse.nju@gmail.com , the email subject is "**Assignment 3**";

(ii) The deadline is **23:59 on June 1, 2019**. No submission after this deadline is acceptable.

About Plagiarize

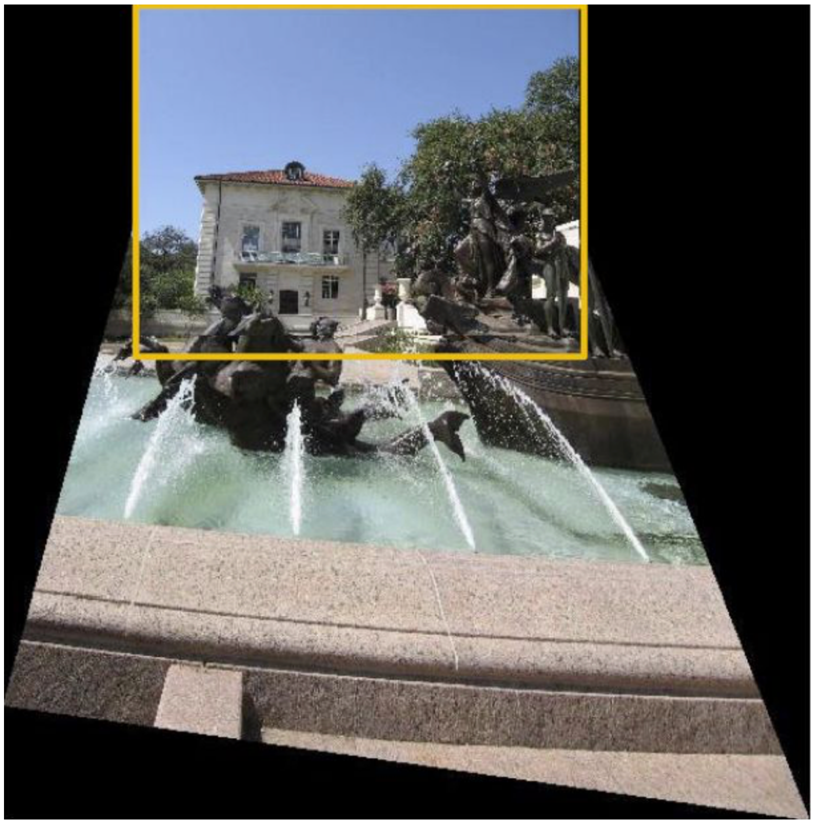
DO NOT PLAGIARIZE! We have no tolerance for plagiarizing and will penalize it with giving zero score. You may refer to some others' materials, please make citations such that one can tell which part is actually yours.

Evaluation Criterion

We mainly evaluate your submission according to your code and report. Efficient implementation, elegant code style, concise and logical report are all important factors towards a high score.

1 Image Mosaics (80 points)

In this exercise, you will implement an image stitcher that uses image warping 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.



Hint: There are some built-in Matlab functions that could do a lot of the work for this project. However, to get practice with the workings of the algorithms, we want you to write your own code. ***For example, you may not use any of these functions in your implementation: cp2tform, imtransform, tformarray, tformfwd, tforminv, imwarp, maketform****.*

Here are the main components you will need to implement:

(1) **Getting correspondences [5 points]**: first 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.

(2) **Computing the homography parameters [25 points]**: write a function that takes a set of corresponding image points and computes the associated 3×3 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 $\lambda p_i = H p'_i$. Note that p 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 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: $'\backslash'$ 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 sure to handle homogenous and non-homogenous coordinates correctly.

(3) **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.

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. Note that transforming all the points will generate an image of a different shape / dimensions than the original input.

[Useful Matlab functions: $round$, $interp2$, $meshgrid$, $isnan$].

As an example for the syntax of $interp2$, consider this form: $ZI = INTERP2(Z, XI, YI)$. If we call

$$output = interp2(double(input), x, y);$$

where x and y together specify a single position we want to inverse warp from, then we get the interpolated color for that single position sampled from $input$. If we call

$$output(:, :, i) = interp2(double(input(:, :, i)), X, Y);$$

where X and Y are 2D matrices of coordinates (X listing the x positions, Y listing the y positions), then we get a corresponding matrix of interpolated colors for all those positions at once.

(4) **Create the output mosaic: [5 points]** 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.

After writing and debugging your system:

1) **[5 pts]** Apply your system to the provided pair of images, and display the output mosaic.

2) **[5 pts]** Show one additional example of a mosaic you create using images that you have taken. You might make a mosaic from two or more images of a broad scene that requires a wide angle view to see well. Or, make a mosaic using two images from the same room where the same person appears in both. Or.... Hint: Keep in mind the constraints on the camera for the views that this technique assumes.

3) **[10 pts]** Warp one image into a "frame" region in the second image. To do this, let the points from the one view be the corners of the image you want to insert in the frame, and the let the corresponding points in the second view be the clicked points of the frame (rectangle) into which the first image should be warped. Use this idea to 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 replace a portrait on the wall with someone else's face, or paste a Powerpoint slide onto a movie screen, ...

You may want to play around a bit with the choice of points for the correspondence pairs until you get a reasonable alignment. Be sure to comment your code.

2 Automatic Image Mosaics [20 pts]

(1) **[10 pts]**Use VLFeat (URL below) to automatically obtain interest points and descriptors, and write code to automatically identify matching descriptors in the two input images. Now your system should be able to run in either of two modes — either with clicked correspondences or with automatically found correspondences.

http://www.vlfeat.org/overview/sift.html

You may use any of the interest point, descriptor, and matching code provided by VLFeat for this portion, or you may implement your own module for matching (which could facilitate answering number 3 below).

(2) **[10 pts]**(Depends on extra credit 1) Implement RANSAC for robustly estimating the homography matrix from noisy correspondences. Show 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.

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) order, i.e., (x,y).
- Check the order of the clicked corresponding points, to make sure your code uses the intended corresponding point pairs.
- As usual, be careful with how images are cast for computations and display ($double$ v.s. $uint8$). In particular, if you use the $interp2$ function, be sure to pass it 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.