I. Programming problem: image mosaics [100 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. For simplicity, we’ll specify corresponding pairs of points manually using mouse clicks. For extra credit, you can optionally implement an automated correspondence process with local feature matching.

** There are some built-in Matlab functions that could do much of the work for this project. However, to get practice with the workings of the algorithms, we want you to write your own code. Specifically, you may not use any of these functions in your implementation: cp2tform, imtransform, tformarray, tformfwd, tforminv, maketform.**
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 $\lambda p_i = H p'_i$. 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 solution using a system of linear equations $Ax = b$, where the 8 unknowns of $H$ are stacked into an 8-vector $x$, the $2n \times 8$ matrix $A$ is filled appropriately so that the full system gives us $\lambda p_i = H p'_i$. Let $H_{33} = 1$. 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 sure 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.

  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: $ZI = \text{INTERP2}(Z,XI,YI)$. If we call $\text{output} = \text{interp2}(\text{double}(\text{input}), \text{x}, \text{y})$; 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 $\text{output}(:, :, i) = \text{interp2}(\text{double}(\text{input}(:, :, i)), \text{X}, \text{Y})$; 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.]
• **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.

After writing and debugging your system:

1. **[15 pts]** Apply your system to the provided pair of images, and display the output mosaic.

2. **[20 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.

3. **[15 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, …

For all examples, 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.

II. **[OPTIONAL] Extra credit [up to 10 pts each, max 30 pts]**

1. Replace the manual correspondence stage with automatic interest point detection and local feature matching. Check out available code here to compute the local interest points and features:
   - [http://www.vlfeat.org/overview/sift.html](http://www.vlfeat.org/overview/sift.html)
   - [http://www.robots.ox.ac.uk/~vgg/research/affine/detectors.html](http://www.robots.ox.ac.uk/~vgg/research/affine/detectors.html)

2. 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.

3. Rectify an image with some known planar surface (say, a square floor tile, or the rectangular face of a building façade) and show the virtual fronto-parallel view. In this case there is only one input image. To solve for $H$, you define the correspondences by clicking on the four corners of the planar surface in the input image, and then associating them with hand-specified coordinates for the output image. For example, a square tile’s corners from the non-frontal view could get mapped to $[0 \ 0; 0 \ N; N \ 0; N \ N]$ in the output.

4. Make a short video in the style of the HP commercials. Building on #3 above, let the frame in the output video move to different positions over time, and warp the framed image into the correct position for every video frame in the sequence.
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 specified 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` vs. `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.

Submission instructions: what to hand in

**Electronically:**

- Your well-documented Matlab code .m files.
- A pdf file containing
  - Your name and CS login ID at the top.
  - Your image results and accompanying brief explanations. This file will be posted online; be sure to include credit for images if necessary.
  - (optional): any results and descriptions for extra credit portions in Section II.

Submit all the above with one call to turnin:

```
>> turnin --submit shalini pset3 file.pdf codeFileXYZ.m codeFileABC.m etc.
```

**Hardcopy:** Also drop a hardcopy in the CS homework dropbox in PAI 5.38. Attach a cover page with the course number CS376 and your name, to make it easy to find in the dropbox. Please save paper by concatenating shorter functions into a single page before printing.