Part C: Image Homography (50 points)

In this part, you will compute an image homography, from matching points between two images. Using this homography, you can tell where points from the first image appear in the second image. You can also compute a warp between the two images, stitching the two images into the same canvas.

You are required to write and submit the following:

1) [15 pts] Function `H = estimate_homography(PA, PB)` to compute a homography between the points from the first image (in matrix `PA`) and second image (in matrix `PB`). Inputs: `PA` and `PB` are 4x2 matrices (or they can have more than 4 rows); each row contains the *(x, y)* coordinates of matching points (a row in the first matrix should be the match for a row in the second matrix). Output: `H` is a 3x3 matrix. You need to set up a system of equations `A` as shown in **slide 34** here. Once you set up your system `A`, solve for `H` using: `[~, ~, V] = svd(A); h = V(:, end); H = reshape(h, 3, 3)';`.
2) [5 pts] Function `[p2] = apply_homography(p1, H)` to apply the homography and convert back from homogeneous coordinates, as shown in slide 34.
3) [30 pts] Script titled `mosaic.m` where you load images, select matching points, compute a homography, apply it to a new point from the first image, and stitch a mosaic from the two images:
    a) [1 pts] Use the following two images: `keble1.png`, `keble2.png` in the HW2 folder. Load them into Matlab and show them in separate figures, followed by the command `impixelinfo` after each figure. This will allow you to see pixel coordinates at the bottom of the figures, when you hover over the images.
    b) [4 pts] Examine the images, and manually determine at least four pairs of points (in each pair, one point should be from the first image, and one from the second image) that are distinctive. Write them down in matrix form in the script, with rows being the points and columns being the *x* and *y* locations. This will give you the `PA, PB` to use below.
    c) [2 pts] Use the function `H = estimate_homography(PA, PB)` you wrote that computes a homography between the points from the first image (in matrix `PA`) and second image (in matrix `PB`).
    d) [3 pts] Now pick one new point from the first image, write it down in your script, and use the computed homography to compute where it "lands" in the second image. Use your `apply_homography` function to do this. Create a 1x2 subplot which shows (1) the first image, with the `p1` point selected shown in green, and (2) the second image, with the `p2` point computed using the homography, shown in yellow. Save and submit your result as `keble_onept.png`.
    e) Now stitch a mosaic from the two images. Save and submit your result as `keble_mosaic.png`.

i) [2 pts] Create a new canvas which replicates the size of `image2` 3 times in the horizontal and 3 times in the vertical direction, and puts `image2` in the middle of this canvas.

ii) [8 pts] For each pixel at location `p1` in `image1`, apply the estimated homography to determine location `p2` where to send the pixel from `p1` into the canvas you created. The new location you compute might be negative, which means it indexes the black part of the canvas; you will need to add two values to the first and second component of `p2`, to convert from `image2`'s coordinate system to the large canvas' coordinate system. The location you computed might be a non-integer so round both the *x* and *y* components up and down (using `ceil` and `floor`), resulting in up to four locations in `image2`, for each pixel in `image1`.

iii) [3 pts] Add each pixel from `image1` to (four locations in) the large canvas.

iv) [2 pt] After iterating over all pixels in `image1`, show the stitched result.

4) [5 pts] Now apply your algorithm on these two images: `uttower1.jpg`, `uttower2.jpg` in the HW2 folder. You will need to select new matching points, and might need to use more than four pairs. Generate both visual results as above, and save and submit them as `uttower_onept.png` and `uttower_mosaic.png`.

Submission:
Part A:

- `detectCircles.m` and `convert_variables.m`
- demonstrations of your circle detection on the provided images `jupiter.jpg` and `egg.jpg`, for one radius (can be different for the two images), named `jupiter_circles.png` and `egg_circles.png`, respectively

Part B:

- `quantizeRGB.m`
- 3 different images with different *k* applied on `fish.jpg`, named `k?.png`, `k??.png`, `k???.png`, where the question marks are replaced with the values of *k* you used

Part C:

- `estimate_homography.m`
- `apply_homography.m`
- `mosaic.m`
- `keble_onept.png, keble_mosaic.png, uttower_onept.png, uttower_mosaic.png`

Acknowledgement: This assignment was adapted from Kristen Grauman's and Adriana Kovashka's original assignment.