16-720 Computer Vision: Homework 2
Feature Descriptor& Homographies & RANSAC

Instructors: Srinivasa Narasimhan, Abhinav Gupta
TAs: Chao Liu, Lerrel Pinto, Dawei Wang
Gunnar Sigurdsson, Vivek Krishnan

Due: Monday, Oct.12, 11:59:59 p.m.

1. Please pack your system and write-up into a single file named <**AndrewId**>**.zip**, see the complete submission checklist in the overview.
2. All questions marked with a **Q** require a submission.
3. **For the implementation part, please stick to the headers, variable names, and file conventions provided.**
4. **Start early!** This homework will take a long time to complete.
5. **Attempt to verify your implementation as you proceed:** If you don't verify that your implementation is correct on toy examples, you will risk having a huge mess when you put everything together.
6. If you have any questions, please contact TAs - Chao (`chao.liu@cs.cmu.edu`), Gunnar (`gsigurds@cs.cmu.edu`), Lerrel (`lerrelp@andrew.cmu.edu`), Dawei (`daweiwan@andrew.cmu.edu`) and Vivek (`vrkrishn@andrew.cmu.edu`) .

Introduction

In this homework, you will implement an interest point (keypoint) detector, a feature descriptor and an image stitching tool based on feature matching and homography.

Interest point (keypoint) detectors find particularly salient points in an image. We then can then extract a feature descriptor that helps describe the region around each of the interest points. SIFT, SURF and BRIEF are all examples of commonly used feature descriptors. Once we have extracted the interest points, we can use feature descriptors to match them (find correspondences) between images to do neat things like panorama stitching or scene reconstruction.

For the feature descriptor, we will be implementing the BRIEF descriptor in this homework. It has a very compact representation, is quick to compute, and has a discriminative yet easily computed distance metric, and is relatively simple to implements. This allows for real-time computation, as you have seen in class. Most importantly, as you will see, it is also just as powerful as more complex descriptors like SIFT, for many cases.

After the extracted features have been matched, we will begin to explore the homography between images based on the locations of the matched features. Specifically, we will be looking at the planar homographies. Why is this useful? In many robotic applications, robots often must deal tabletops, walls, and the ground among other flat planar surfaces. When two cameras observe a plane, there exists a relationship between the images captured. This relationship is defined by a 3×3 transformation matrix, called a planar homography. A

Figure 1: Example Gaussian pyramid for `model_chickenbroth.jpg`

planar homography allows us to compute how a planar scene would look from second camera location, given only the first image. In fact, we can compute how images of the plane will look from any camera at any location without knowing any internal camera parameters and without actually taking the pictures, all with the planar homography.

1 Keypoint Detector

For our implementation, we will be using the interest point detector similar to the one introduced in class. A good reference for its implementation can be found in [3]. Keypoints are found by using the Difference of Gaussian (DoG) detector. This detector finds points that are extrema in both scale and space of a DoG pyramid. This is described in [1], an important paper in our field. Here, we will be implementing a simplified version of the DoG detector described in Section 3 of [3].

NOTE: The parameters to use for the following sections are:
$\sigma_0 = 1$**,** $k = \sqrt{2}$**,** `levels` $= [-1, 0, 1, 2, 3, 4]$**,** $\theta_c = 0.03$ **and** $\theta_r = 12$.

1.1 Gaussian Pyramid

In order to create a DoG pyramid, we will first need to create a Gaussian pyramid. Gaussian pyramids are constructed by progressively applying a lowpass Gaussian filter to the input image. This function is already provided to your in `matlab/createGaussianPyramid.m`.

```
GaussianPyramid = createGaussianPyramid(im, sigma0, k, levels)
```

The function takes as input a grayscale image `im` with values between 0 and 1 (hint: `im2double()`, `rgb2gray()`), the scale of the zeroth level of the pyramid `sigma0`, the pyramid factor `k`, and a vector `levels` specifying the levels of the pyramid to construct.

At level l in the pyramid, the image is smoothed by a Gaussian filter with $\sigma_l = \sigma_0 k^l$. The output `GaussianPyramid` is a $R \times C \times L$ matrix, where $R \times C$ is the size of the input image `im` and L is the number of levels. An example of a Gaussian pyramid can be seen in Figure 1. You can visualize this pyramid with the provided function

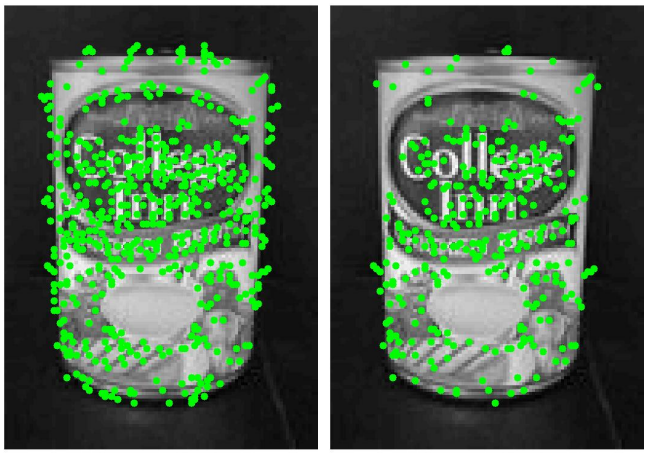
```
displayPyramid(pyramid)
```

1.2 The DoG Pyramid (5 pts)

The DoG pyramid is obtained by subtracting successive levels of the Gaussian pyramid. Specifically, we want to find:

$$D_l(x, y, \sigma_l) = (G(x, y, \sigma_l) - G(x, y, \sigma_{l-1})) * I(x, y) \tag{1}$$

Figure 2: Example DoG pyramid for `model_chickenbroth.jpg`



(a) Without edge suppression (b) With edge suppression

Figure 3: Interest Point (keypoint) Detection without and with edge suppression for `model_chickenbroth.jpg`

where $G(x, y, \sigma_l)$ is the Gaussian filter used at level l and $*$ is the convolution operator. Due to the distributive property of convolution, this simplifies to

$$D_l(x, y, \sigma_l) = G(x, y, \sigma_l) * I(x, y) - G(x, y, \sigma_{l-1}) * I(x, y) \quad (2)$$

$$= GP_l - GP_{l-1} \quad (3)$$

where GP_l denotes the l level in the Gaussian pyramid.

Q 1.2: Write the following function to construct a Difference of Gaussian pyramid:

```
[DoGPyramid, DoGLevels] = createDoGPyramid(GaussianPyramid, levels)
```

The function should return `DoGPyramid` an $R \times C \times (L-1)$ matrix of the DoG pyramid created by differencing the `GaussianPyramid` input. Note that you will have one level less than the Gaussian Pyramid. `DoGLevels` is an $(L-1)$ vector specifying the corresponding levels of the DoG Pyramid (should be the last $L-1$ elements of `levels`). An example of the DoG pyramid can be seen in Figure 2.

1.3 Edge Suppression (10 pts)

The Difference of Gaussian function responds strongly on corners and edges in addition to blob-like objects. However, edges are not desirable for feature extraction as they are not

as distinctive and do not provide as stable of a localization for keypoints. Here, we will implement the edge removal method described in Section 4.1 of [3], which is based on the principal curvature ratio in a local neighborhood of a point. The paper makes the observation that edge points will have a "large principal curvature across the edge but a small one in the perpendicular direction."

Q 1.3: Implement the following function:

```
PrincipalCurvature = computePrincipalCurvature(DoGPyramid)
```

The function takes in `DoGPyramid` generated in the previous section and returns `PrincipalCurvature`, a matrix of the same size where each point contains the curvature ratio R for the corresponding point in the DoG pyramid:

$$R = \frac{\text{Tr}(H)^2}{\text{Det}(H)} = \frac{(\lambda_{min} + \lambda_{max})^2}{\lambda_{min}\lambda_{max}} \tag{4}$$

where H is the Hessian of the Difference of Gaussian function (i.e. one level of the DoG pyramid) computed by using pixel differences as mentioned in Section 4.1 of [3]. (hint: Matlab function `gradient()`).

$$H = \begin{bmatrix} D_{xx} & D_{xy} \\ D_{yx} & D_{yy} \end{bmatrix} \tag{5}$$

This is similar in spirit to but *different* than the Harris matrix you saw in class. Both methods examine the eigenvalues λ of a matrix, but the method in [3] performs a test **without** requiring the direct computation of the eigenvalues. Note that you need to compute each term of the Hessian before being able to take the trace and determinant.

We can see that R reaches its minimum when the two eigenvalues λ_{min} and λ_{max} are equal, meaning that the curvature is the same in the two principal directions. Edge points, in general, will have a principal curvature significantly larger in one direction than the other. To remove edge points, we simply check against a threshold $R > \theta_r$. Fig. 3 shows the DoG detector with and without edge suppression.

1.4 Detecting Extrema (10 pts)

To detect corner-like, scale-invariant interest points, the DoG detector chooses points that are local extrema in both scale and space. Here, we will consider a point's eight neighbors in space and its two neighbors in scale (one in the scale above and one in the scale below).

Q 1.4: Write the function :

```
locs = getLocalExtrema(DoGPyramid, DoGLevels, PrincipalCurvature,
                       th_contrast, th_r)
```

This function takes as input `DoGPyramid` and `DoGLevels` from Section 1.2 and `PrincipalCurvature` from Section 1.3. It also takes two threshold values, `th_contrast` and `th_r`. The threshold θ_c should remove any point that is a local extremum but does not have a Difference of Gaussian (DoG) response magnitude above this threshold (i.e. $|D(x, y, \sigma)| > \theta_c$). The threshold θ_r should remove any edge-like points that have too large a principal curvature ratio specified by `PrincipalCurvature`.

The function should return `locs`, an $N \times 3$ matrix where the DoG pyramid achieves a local extrema in both scale and space, and also satisfies the two thresholds. The first and second column of locs should be the (x, y) values of the local extremum respectively and the third column should contain the corresponding level of the DoG pyramid where it was detected. (Try to eliminate loops in the function so that it runs efficiently.)

NOTE: In all implementations, we assume the x coordinate corresponds to columns and y coordinate corresponds to rows. For example, the coordinate (10,20) cooresponds to the (row 20, column 10) in the image.

1.5 Putting it together (5 pts)

Q 1.5: Write the following function to combine the above parts into a DoG detector:

```
[locs, GaussianPyramid] = DoGdetector(im, sigma0, k, levels, th_contrast,
                                      th_r)
```

The function should take in a gray scale image, `im`, scaled between 0 and 1, and the parameters `sigma0`, `k`, `levels`, `th_contrast`, and `th_r`. It should call each of the above functions and return the keypoints in `locs` and the Gaussian pyramid in `GaussianPyramid`. Figure 3 shows the keypoints detected for an example image. Note that we are dealing with real images here, so your keypoint detector may find points with high scores that you do not perceive to be corners.

Include the image with the detected keypoints (similiar to the one shown in Fig. 3 (b)). You can use any of the provided images.

2 BRIEF Descriptor

Now that we have interest points that tell us where to find the most informative points in the image, we can compute descriptors that can be used to match to other views of the same point in different images. The BRIEF descriptor encodes information from a 9×9 patch p centered around the interest point at the *characteristic scale* of the interest point. See the lecture notes for Point Feature Detectors if you need to refresh your memory.

2.1 Creating a Set of BRIEF Tests (10 pts)

The descriptor itself is a vector that is n-bits long, where each bit is the result of the following simple test:

$$\tau(I; p_1, p_2) := \begin{cases} 1, & \text{if } I(p_1) < I(p_2). \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

where $I(p)$ is the pixel intensity at p in the patch. In the implementation, set the number of bits n to 256. There are many choices for the 256 test pairs (p_1, p_2) used to compute $\tau(I; p_1, p_2)$. The authors describe and test some of them in [2]. Read section 3.2 of that paper and implement one of these solutions. You should generate a static set of test pairs and save that data to a file. You will use these pairs for all subsequent computations of the BRIEF descriptor.

Q 2.1: Write the function to create the x and y pairs that we will use for comparison to compute τ:

```
[compareX, compareY] = makeTestPattern(patchWidth, nbits)
```

`patchWidth` is the width of the image patch (usually 9) and `nbits` is the number of tests in the BRIEF descriptor. `compareX` and `compareY` are linear indices into the `patchWidth` $\times$ `patchWidth` image patch and are each $n \times 1$ vectors. Run this routine for the given parameters `patchWidth` = 9 and $n = 256$ and save the results in `testPattern.mat`. **Include this file** in your submission.

2.2 Compute the BRIEF Descriptor (10 pts)

It is now time to compute the BRIEF descriptor for the detected keypoints.

Q 2.2: Write the function:

```
[locs,desc] = computeBrief(im, locs, levels, compareX, compareY)
```

Where `im` is a grayscale image with values from 0 to 1, `locs` are the keypoint locations returned by the DoG detector from Section 1.5, `levels` are the Gaussian scale levels that were given in Section 1, and `compareX` and `compareY` are the test patterns computed in Section 2.1 and were saved into `testPattern.mat`.

The function returns `locs`, an $m \times 3$ vector, where the first two columns are the image coordinates of keypoints and the third column is the pyramid level of the keypoints, and `desc`, an $m \times n$ matrix of stacked BRIEF descriptors. m is the number of valid descriptors in the image and will vary. You may have to be careful about the input DoG detector locations since they may be at the edge of an image where we cannot extract a full patch of width `patchWidth`. Thus, the number of output `locs` may be less than the input.

2.3 Putting it all Together (5 pts)

Q 2.3: Write a function :

```
[locs, desc] = briefLite(im)
```

,which accepts a grayscale image `im` with values between zero and one and returns `locs`, an $m \times 3$ vector, where the first two columns are the image coordinates of keypoints and the third column is the pyramid level of the keypoints, and `desc`, an $m \times n$ *bits* matrix of stacked BRIEF descriptors. m is the number of valid descriptors in the image and will vary. n is the number of bits for the BRIEF descriptor.

This function should perform all the necessary steps to extract the descriptors from the image, including

- Load parameters and test patterns
- Get keypoint locations
- Compute a set of valid BRIEF descriptors

2.4 Check Point: Descriptor Matching (5 pts)

A descriptor's strength is in its ability to match to other descriptors generated by the same world point, despite change of view, lighting, etc. The distance metric used to compute the similarity between two descriptors is critical. For BRIEF, this distance metric is the Hamming distance. The Hamming distance is simply the number of bits in two descriptors that differ. (Note that the position of the bits matters.)

To perform the descriptor matching mentioned above, we have provided the function in `matlab/briefMatch.m`):

```
[matches] = briefMatch(desc1, desc2)
```

Which accepts an $m_1 \times n$ stack of BRIEF descriptors from a first image and a $m_2 \times n$ stack of BRIEF descriptors from a second image and returns a $p \times 2$ matrix of matches, where the first column are indices into `desc1` and the second column are indices into `desc2`. Note that m_1, m_2, and p may be different sizes and $p \leq \min(m_1, m_2)$.

Q 2.4.1: Write a test script `testMatch` to load two of the **chickenbroth** images, compute feature matches. Use the provided `plotMatches` and `briefMatch` functions to visualize the result.

```
plotMatches(im1, im2, matches, locs1, locs2)
```

Where `im1` and `im2` are grayscale images from 0 to 1, `matches` is the list of matches returned by `briefMatch` and `locs1` and `locs2` are the locations of keypoints from `briefLite`.

Save the resulting figure and submit it in your PDF. Also, present results with the two `incline` images and with the computer vision textbook cover page (template is in file `pf_scan_scaled.jpg`) against the other `pf_*` images. Briefly discuss any cases that perform worse or better.

Figure 4 is an example result. Suggest for debugging: A good test of your code is to check that you can match an image to itself.

2.5 BRIEF and rotations (10 pts)

You may have noticed worse performance under rotations. Let's investigate this!

Q 2.5: Take the `model_chickenbroth.jpg` test image and match it to itself while rotating the second image (hint: `imrotate`) in increments of 10 degrees. Count the number of correct matches at each rotation and construct a bar graph showing rotation angle vs the number of correct matches. Include this in your PDF and explain why you think the descriptor behaves this way. **Create a script** `briefRotTest.m` that performs this task.

3 Improving Performance - (Extra Credit)

The extra credit opportunities described below are optional and provide an avenue to explore computer vision and improve the performance of the techniques developed above.

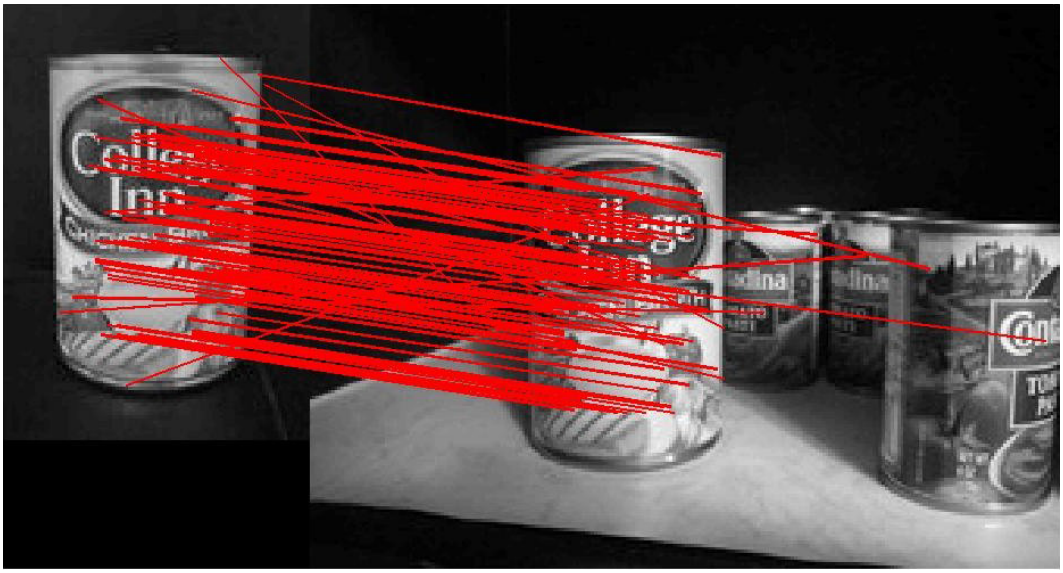


Figure 4: Example of BRIEF matches for `model_chickenbroth.jpg` and `chickenbroth_01.jpg`.

1. **(10 pts)** As we have seen, BRIEF is not rotationally invariant. Design a simple fix to solve this problem using the tools you have developed so far. Explain in your PDF your design decisions and how you selected any parameters that you use. Demonstrate the effectiveness of your algorithm on image pairs related by large rotation.

2. **(10 pts)** This implementation of BRIEF has some scale invariance, but there are limits. What happens when you match a picture to the same picture at half the size? Look to section 3 of [3] for a technique that will make your detector more robust to changes in scale. Implement it and demonstrate it in action with several test images. You may simply rescale some of the test images we have given you.

4 Planar Homographies: Theory (up to 35 pts)

Suppose we have two cameras $\mathbf{C}_1$ and $\mathbf{C}_2$ looking at a common plane Π in 3D space. Any 3D point $\mathbf{P}$ on Π generates a projected 2D point located at $\mathbf{p} \equiv (u_1, v_1, 1)^T$ on the first camera $\mathbf{C}_1$ and $\mathbf{q} \equiv (u_2, v_2, 1)^T$ on the second camera $\mathbf{C}_2$. Since $\mathbf{P}$ is confined to the plane Π, we expect that there is a relationship between $\mathbf{p}$ and $\mathbf{q}$. In particular, there exists a common 3×3 matrix $\mathbf{H}$, so that for any $\mathbf{p}$ and $\mathbf{q}$, the following conditions holds:

$$\mathbf{p} \equiv \mathbf{H}\mathbf{q} \tag{7}$$

We call this relationship *planar homography*. Recall that both $\mathbf{p}$ and $\mathbf{q}$ are in homogeneous coordinates and the equality $\equiv$ means $\mathbf{p}$ is proportional to $\mathbf{Hq}$ (recall homogeneous coordinates). It turns out this relationship is also true for cameras that are related by pure rotation without the planar constraint.

- **Q4.1 (20 pts):** We have a set of points $\mathbf{p} = \{p^1, p^2, \ldots, p^N\}$ in an image taken by camera $\mathbf{C}_1$ and corresponding points $\mathbf{q} = \{q^1, q^2, \ldots, q^N\}$ in an image taken by $\mathbf{C}_2$. Suppose we know there exists an unknown homography $\mathbf{H}$ between corresponding points for all $i \in \{1, 2, \ldots, N\}$. This formally means that $\exists\ \mathbf{H}$ such that:

$$p^i \equiv \mathbf{H} q^i \tag{8}$$

where $p^i = (x_i, y_i, 1)$ and $q^i = (u_i, v_i, 1)$ are homogeneous coordinates of image points each from an image taken with $\mathbf{C_1}$ and $\mathbf{C_2}$ respectively.

  (a) Given N correspondences in $\mathbf{p}$ and $\mathbf{q}$ and using Equation 8, derive a set of $2N$ independent linear equations in the form:

$$\mathbf{A}\mathbf{h} = \mathbf{0} \tag{9}$$

  where $\mathbf{h}$ is a vector of the elements of $\mathbf{H}$ and $\mathbf{A}$ is a matrix composed of elements derived from the point coordinates. Write out an expression for $\mathbf{A}$.
  Hint: Start by writing out Equation 8 in terms of the elements of $\mathbf{H}$ and the homogeneous coordinates for p^i and q^i.

  (b) How many elements are there in $\mathbf{h}$?

  (c) How many point pairs (correspondences) are required to solve this system?
  Hint: How many degrees of freedom are in $\mathbf{H}$? How much information does each point correspondence give?

  (d) Show how to estimate the elements in $\mathbf{h}$ to find a solution to minimize this homogeneous linear least squares system. Step us through this procedure.
  Hint: Use the Rayleigh quotient theorem (homogeneous least squares).

- **(extra credit) 4.2 (15 pts):** Prove that there exists an $\mathbf{H}$ that satisfies Equation 7 given two cameras separated by a pure rotation. That is, for $\mathbf{C_1}$, $\mathbf{p} = \mathbf{K_1}\begin{bmatrix}\mathbf{I} & 0\end{bmatrix}\mathbf{P}$ and for $\mathbf{C_2}$, $\mathbf{p'} = \mathbf{K_2}\begin{bmatrix}\mathbf{R} & 0\end{bmatrix}\mathbf{P}$. Note that $\mathbf{K}$ is not the same for the two cameras.

5 Planar Homographies: Implementation (20 pts)

Now that we have derived how to find $\mathbf{H}$ mathematically in **Q4.1**, we will implement in this section.

- **Q5.1 (10pts)** Implement the function

```
H2to1=computeH(p1,p2)
```

  Inputs: `p1` and `p2` should be $2 \times N$ matrices of corresponding $(x, y)^T$ coordinates between two images.
  Outputs: `H2to1` should be a 3×3 matrix encoding the homography that best matches the linear equation derived above for Equation 8 (in the least squares sense). *Hint:* Remember that a homography is only determined up to scale. The Matlab functions `eig()` or `svd()` will be useful. Note that this function can be written without an explicit for-loop over the data points.

(a) `incline_L.jpg (img1)` (b) `incline_R.jpg (img2)` (c) `img2` warped to `img1`'s frame

Figure 5: Example output for **Q6.1**: Original images `img1` and `img2` (left and center) and `img2` warped to fit `img1` (right). Notice that the warped image clips out of the image. We will fix this in **Q6.2**

6 Stitching it together: Panoramas (30 pts)

We can also use homographies to create a panorama image from multiple views of the same scene. This is possible for example when there is no camera translation between the views (e.g., only rotation about the camera center). First, you will generate panoramas using matched point correspondences between images using the BRIEF matching in Section. 2.4. We will assume that there is no error in your matched point correspondences between images (Although there might be some errors). In the next section you will extend the technique to deal with the noisy keypoint matches.

You will need to use the provided function `warp_im=warpH(im, H, out_size)`, which warps image `im` using the homography transform `H`. The pixels in `warp_im` are sampled at coordinates in the rectangle $(1, 1)$ to $(\texttt{out\_size(2)}, \texttt{out\_size(1)})$. The coordinates of the pixels in the source image are taken to be $(1, 1)$ to $(\texttt{size(im,2)}, \texttt{size(im,1)})$ and transformed according to `H`.

- **Q6.1 (15pts)** In this problem you will implement and use the function:

  ```
  [panoImg] = imageStitching(img1, img2, H2to1)
  ```

  on two images from the Dusquesne incline. This function accepts two images and the output from the homography estimation function. This function will:

  (a) Warps `img2` into `img1`'s reference frame using the provided `warpH()` function

  (b) Blends `img1` and warpped `img2` and outputs the panorama image.

  For this problem, use the provided images `incline_L` as `img1` and `incline_R` as `img2`. The point correspondences `pts` are generated by your BRIEF descriptor matching.

  Apply your `computeH()` to these correspondences to compute `H2to1`, which is the homography from `incline_R` onto `incline_L`. Then Apply this homography to `incline_R` using `warpH()`.
  Note: Since the warped image will be translated to the right, you will need a larger target image.

  Visualize the warped image and save this figure as `results/q6_1.jpg` using Matlab's `imwrite()` function and save *only* `H2to1` as `results/q6_1.mat` using Matlab's `save()` function.

- **Q6.2 (15pts)** Notice how the output from Q6.1 is clipped at the edges? We will fix this in this question. Implement a function

  ```
  [panoImg] = imageStitching_noClip(img1, img2, H2to1)
  ```

  that takes in the same input types and produces the same outputs as in Q6.1.

  To prevent this clipping at the edges, we instead need to warp *both* image 1 and image 2 into a common third reference frame in which we can display both images without having any clipping. Specifically we want to find a matrix M that *only* does scaling and translation such that:

  ```
  warp_im1 = warpH(im1, M, out_size);
  warp_im2 = warpH(im2, M*H2to1, out_size);
  ```

  This produces warped images in a common reference frame where all points in image 1 and image 2 are visible. To do this, we will only take as input either the width or height of `out_size` and will have to compute the other one based on the given such that the images are not squeezed or elongated in the panorama image. For now, assume we only take as input the width of the image in `out_size(2)` and must compute the correct `out_size(1)`.

  Hint: The computation will be done in terms of `H2to1` and the extreme points (corners) of the two images Make sure M includes only scaling (find the aspect ratio of the full-sized panorama image) and translation.

  Again, pass `incline_L` as `img1` and `incline_R` as `img2`. **Save** the resulting panorama in `results/q6_2_pan.jpg`.

7 RANSAC (25 pts)

The least squares method you implemented for computing homographies is not robust to outliers. If all the correspondences are good matches (like in the previous section), this is not a problem. But even a single false correspondence can completely throw off the homography estimation. When correspondences are determined automatically (using BRIEF feature matching for instance), some mismatches in a set of point correspondences are almost certain. RANSAC can be used to fit models robustly in the presence of outliers.

Q7.1 (15pts): Write a function that uses RANSAC to compute homographies automatically between two images:

```
[bestH] = ransacH(matches, locs1, locs2, nIter, tol)
```

The input and outputs of this function should be as follows:

- Inputs: `locs1` and `locs2` are matrices specifying point locations in each of the images and `matches` is a matrix specifying matches between these two sets of point locations. These matrices are formatted identically to the output of the provided `briefMatch` function.

- Algorithm Input Parameters: `nIter` is the number of iterations to run RANSAC for, `tol` is the tolerance value for considering a point to be an inlier. Define your function so that these two parameters have reasonable default values.
- Outputs: `bestH` should be the homography model with the most inliers found during RANSAC.

Q7.2 (10pts): You now have all the tools you need to automatically generate panoramas. Write a function that accepts two images as input, computes keypoints and descriptors for both the images, finds putative feature correspondences by matching keypoint descriptors, estimates a homography using RANSAC and then warps one of the images with the homography so that they are aligned and then overlays them.

```
im3 = generatePanorama(im1, im2)
```

Run your code on the image pair `data/incline_L.jpg`, `data/incline_R.jpg`. However during debugging, try on scaled down versions of the images to keep running time low. Save the resulting panorama on the full sized images as `results/q7_2.jpg`. (see Figure 6 for example output from Q7.2). Include the figure in your writeup.

Figure 6: Final panorama view. With homography estimated with RANSAC.

8 Extra Credit (Upto 10pts)

Take pictures with your own camera, separated by a pure rotation, and automatically construct a panorama *with 3 or more* images. Be sure that objects in the images are far enough away so that there are no parallax effects. Submit the original images as `results/ec8_1.jpg`, `results/ec8_2.jpg` etc., and a script `matlab/ec.m` that loads the images and assembles a panorama. Save the panorama as `ec8_pan.jpg`. We may even showcase some of the best panoramas in class! :)

9 What to Submit

Your submission should consist of only 1 file, a zip file named `<AndrewId>.zip`. This zip file should contain

- a folder `matlab` containing all the `.m` and `.mat` files you were asked to write and generate
- a `pdf` named `writeup.pdf` containing the results, explanations and images asked for in the assignment along with to the answers to the questions on homographies.

Submit all the code needed to make your panorama generator run. Make sure all the `.m` files that need to run are accessable from the `matlab` folder without any editing of the path variable. If you downloaded and used a feature detector for the extra credit, include the code with your submission and mention it in your writeup. You may leave the `data` folder in your submission, but it is not needed. Please zip your homework as usual and submit it using blackboard.

Appendix: Image Blending

Note: This section is not for credit and is for informational purposes only.

For overlapping pixels, it is common to blend the values of both images. You can simply average the values but that will leave a seam at the edges of the overlapping images. Alternatively, you can obtain a blending value for each image that fades one image into the other. To do this, first create a mask like this for each image you wish to blend:

```
mask = zeros(size(im,1), size(im,2));
mask(1,:) = 1; mask(end,:) = 1; mask(:,1) = 1; mask(:,end) = 1;
mask = bwdist(mask, 'city');
mask = mask/max(mask(:));
```

The function `bwdist` computes the distance transform of the binarized input image, so this mask will be zero at the borders and 1 at the center of the image. You can warp this mask just as you warped your images. How would you use the mask weights to compute a linear combination of the pixels in the overlap region? Your function should behave well where one or both of the blending constants are zero.

References

[1] P. Burt and E. Adelson. The Laplacian Pyramid as a Compact Image Code. *IEEE Transactions on Communications*, 31(4):532–540, April 1983.

[2] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. BRIEF : Binary Robust Independent Elementary Features . *ECCV*, 2010.

[3] David G. Lowe. Distinctive Image Features from Scale-Invariant Keypoints. *International Journal of Computer Vision*, 60(2):91–110, November 2004.