README

**Detect features.py**: use Harris corner detection algorithm to get the keypoints. I first used a 3\*3 gausian kernel to smooth the image (deal with noise) and used a 5\*5 sobel kernel to convolve the image to the x and y gradient. Then we calculate R value for each window(centered at a point, and we store R response in that specific point). Then we used nonmaxsuppts with radius = 7, threshold=0.8 maximum R.

**Match\_features.py**: match two points with NCC and mutual marriage algorithm. For window size, I use 15\*15 (depth = 7) window size. Which by test works good. Note: to eliminate weak correlation, we set a threshold of 0.2 that when NCC is less than 0.2, we considered 2 windows (centered at 2 points) not being matched. We then use mutual marriage to further guarantee matches.

**Compute\_affine\_xform.py:** calculate affine transformation matrix. We build a matrix (shown in code) and b matrix(shown in code), and calculate : , then use value from t to create a 3\*3 matrix (with last row on the bottom being [0, 0, 1]) as our affine matrix transformation. For inlier threshold: we use 10 which works good (inlier looks correct and we have good amount of inliers). To further guarantee output result, after getting inliers and transformation from RANSAC, we run 1000 additional times (shown in code) to randomly chose 3 matches (all chosen from inliers, not from outliers) each time and compute h, so we get 1000 matrix for h and we average the 1000 h and get average h, which is a good way to deal with noise (like gausian noise).

**Compute\_proj\_xform.py:** calculate projective transformation matrix. We randomly pick up 4 (distinct) pairs of matches, and build matrix (shown in code and lectures) and calculate eigenvector with min eigenvalue. Then we divide the min eigenvector with the last value (in order to make last value in eigenvector to become 1, which is projective transformation form).

**Ssift\_descriptor.py:** calculate the ssift descriptor using simplified sift as required. We first use 3\*3 gausian filter to convolve the image to suppress noise. Then we calculate the histogram (4\*4, grid, each with 100 elements, and vote for the “orientation” (pi/4 each unit) with weight = gradient). Also note we used 40\*40 gausian kernel to add weight for the 40\*40 big window. After the voting we will get a 128-d vector , which is 4\*4 grids \* 8-d (orientation) = 128-d vector. Then we normalize this 128-d vector, if value is >0.2, we set it as 0.2. Then we normalize it again, as our descriptor for feature point

**match\_features\_ratio.py**:as requested, we used SSD to calculate distance between 2 windwos (centered at 2 feature points). We do it (len(features1) \* (len(features2)) times, and for fi in features1, we use it to compare with all features in feature2, if best SSD distance / 2nd best SSD distance < 0.6, we see it as a match, otherwise not match.

Note: to make the hw2.py(test terminal) code cleaner, all the drawings are implemented in above functions and not shown in test terminal (hw2.py). Once you run it the images will pop up one by one (with titles), press any key to continue to next image.

**Result analysis:**

bikes1 bikes2, bikes1 bikes3, leuven1 leuven2, leuven1 leuven3, wall1 wall2, graf1 graf2 matches relatively well. For graf1 graf3, wall1 wall3 they match poorly. The possible causes are: NCC (or perhaps correlation in general) is sensitive to orientation and scale, when scale and orientation is changing a lot, we cannot get very good matches to compute affine or proj transformations, as it is happening in graf1 graf3 and wall1 wall3.

Also I found for some pairs (like bikes1 bikes2), proj transformation doesn’t get result as good as affine, this might be because project has higher degree of freedom and not as stable as affine with regard to some specific kinds of transformation (like translation).

Sift has better result than NCC, because it can more accurately characterize inliers. But because we didn’t do orientation matching and scale matching using NloG, it is still sensitive to rotation and scaling.