Programming Assignment - 3
Panoramic Stitching & Stereo Matching
EE6132

TAs
Gopi Raju Matta, ee17d021@smail.iitm.ac.in
Salman Siddique Khan, ee18d406@smail.iitm.ac.in

Due date: 07/11/2020 11:55pm

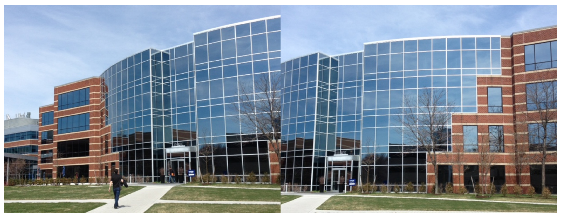
Notes:

1. Please use moodle dicussion threads for posting your doubts.
2. Before posting any question check it before , if the same question has been asked earlier.
3. Submit a single zip file in the moodle named as PA3_Rollno.zip containing report and folders containing corresponding codes.
4. Read the problem fully to understand the whole procedure.
5. Comment your code generously.
6. Put titles to all of the figures shown.
7. Save your iPython notebook as an html file with all the figures and results and submit it along with the iPython notebook containing your code.

---

1 Panoramic Stitching

In this problem we will develop an algorithm for stitching a panorama from overlapping photos (Figure 1), which amounts to estimating a transformation that aligns one image to another. To do this, we will compute SURF features in both images and match them to obtain correspondences. We will then estimate a homography from these correspondences, and we'll use it to stitch the two images together in a common coordinate system. In order to get an accurate transformation, we will need many accurate feature matches. Unfortunately, feature matching is a noisy process: even if two image patches (and their SURF descriptors) look alike, they may not be an actual match. To make our algorithm robust to matching errors, we will use RANSAC, a method for estimating a parametric model from noisy observations. We will use the obtained homography transformation to do panoramic stitching. We have provided you with two input images and also starter code.


(a) Image pair

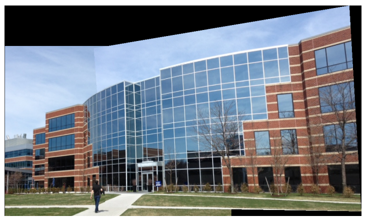

(b) Stitched panorama

Figure 1: Panorama produced using our implementation

Tasks

1. Implement **get_surf_features(img)** to compute SURF features for both of the given image. Implement **match_keypoints(desc1, desc2)** to compute key-point correspondences between the two source images using the ratio test. Run the plotting code to visualize the detected features and resulting correspondences. *(Hint: You can use existing libraries)*
2. Write a function **find_homography(pts1, pts2)** that takes in two N×2 matrices with the x and y coordinates of matching 2D points in the two images and computes the 3×3 homography H that maps pts1 to pts2. You can implement this function using direct linear transform (DLT). *(Hint: You should implement this function on your own)*
3. Your homography-fitting function from (2) will only work well if there are no mismatched features. To make it more robust, implement a function **transform_ransac(pts1, pts2)** that fits a homography using RANSAC. *(Hint: You should implement this function on your own)*
4. Write a function **panoramic_stitching(img1, img2)** that produces a panorama from a pair of overlapping images using your functions from the previous parts. Run the algorithm on the two images provided. *(Hint: You should implement this function on your own)*
5. Extend the algorithm to handle n(>2) images, and run it on your own photos, or photos you found online. *(Hint: You can use existing libraries)*

2 Rectified Stereo Matching

In this assignment, you are required to implement a two-view rectified stereo matching algorithm for disparity estimation. The rectified image pairs are provided in the shared folder linked below. Use a window based based stereo matching algorithm to find the disparity with respect to the reference view.
Link to data folder

Tasks

1. Implement a function called **get_costVol(img1,img2)** that takes in the rectified image pair and outputs a cost volume using window matching. The dimension of cost volume should be $M \times N \times D$ where M, N are the spatial dimension of the input images and D is the number of disparity levels. You can appropriately pad the images depending on the maximum disparity value prior to sending them as an input to the function to have an estimate of the cost for the boundary pixels. Along with the images, you can also send in window size, number of disparity levels, direction of search, maximum disparity etc. as arguments to the function.
2. Implement a function called **get_Disparity(cost_vol)** that takes in the cost volume obtained from the previous step and outputs a disparity map using winner takes all approach. Generate multiple cost volumes for different windows sizes: 3×3, 7×7, 11×11 and maximum disparity levels of 10 and 50 pixels and report the disparity estimates.

3. Implement a function called **get_Uncertainty(cost_vol)** that again takes in the cost volume and outputs the uncertainty in disparity estimate for each pixel. To calculate the uncertainty, use the following approach:
   If we assume $C(p)$ is the cost vector for pixel p, then we can define uncertainty for each pixel as
   $$U(p) = \frac{min(C(p))}{min^*(C(p))} \tag{1}$$
   Here $C(p)$ is a D-dimensional vector where D is the number of disparity levels assumed between maximum and minimum disparity values. $min(.)$ finds the minimum of the vector while $min^*(.)$ finds the second minimum of the vector.
   Similar to disparity estimate, find the uncertainty for different windows sizes: 3×3, 7×7, 11×11 and maximum disparity levels of 10 and 50 pixels.

Note: You are supposed to implement the stereo matching algorithm from scratch. Using predefined matching functions from existing libraries will not earn you any points.