Programming Assignment #3

Announcement: 28 February 2017

Submission Deadline: 14 March 2017

DESCRIPTION

In this assignment, you will write software which stitches images together to form panoramas. Your software will detect useful features in the images, find the best matching features in your other images, align the photographs, then warp and blend the photos to create a seamless panorama.

This assignment can be thought of as two major components:

1. Feature Detection and Matching [Quiz #1]
2. Panorama Mosaic Stitching [this assignment]

FEATURE DETECTION AND MATCHING [Steps 1 and 2]

Step 1: Compute the Harris corner detector using the following steps

A. Compute the x and y derivatives on the image
B. Compute the covariance matrix H of the image derivatives. Typically, when you compute the covariance matrix you compute the sum of Ix^2, Iy^2 and IxIy over a window or small area of the image. To obtain a smooth result for better detection of corners, use a Gaussian weighted window.
C. Compute the Harris response using determinant(H)/trace(H).
D. Find peaks in the response that are above the threshold, and store the interest point locations.

Required: Open "Boxes.png" and compute the Harris corners. Save an image "1a.png" showing the Harris response on "Boxes.png" (you'll need to scale the response of the Harris detector to lie between 0 and 255.) When you're debugging your code, I would recommend using "Boxes.png" to see if you're getting the right result.

Compute the Harris corner detector for images "Rainier1.png" and "Rainier2.png". Save the images "1b.png" and "1c.png" of the detected corners (use the drawing function cv::circle(...) or cv::drawKeypoints(...) to draw the interest points overlaid on the images)

Step 2: Matching the interest points between two images.

A. Compute the descriptors for each interest point.
B. For each interest point in image 1, find its best match in image 2. The best match is defined as the interest point with the closest descriptor (SSD or ratio test).
C. Add the pair of matching points to the list of matches.
D. Display the matches using cv::drawMatches(...). You should see many correct and incorrect matches.

Required: Compute the Harris corner detector and find matches for images "Rainier1.png" and "Rainier2.png". Save the image "2.png" showing the image with the found matches [use cv::drawMatches(...) to draw the matches on the two image].

PANORAMA MOSAIC STITCHING [Steps 3 and 4]

Step 3: Compute the homography between the images using RANSAC (Szeliski, Section 6.1.4). Following these steps:

A. Implement a function *project(x1, y1, H, x2, y2)*. This should project point (x1, y1) using the homography “H”. Return the projected point (x2, y2). Hint: See the slides for details on how to project using homogeneous coordinates. You can verify your result by comparing it to the result of the function *cv::perspectiveTransform(...)*. [Only use this function for verification and then comment it out.].
B. Implement the function *computeInlierCount(H, matches, numMatches, inlierThreshold)*. *computeInlierCount* is a helper function for RANSAC that computes the number of inlying points given a homography "H". That is, project the first point in each match using the function "project". If the projected point is less than the distance "inlierThreshold" from the second point, it is an inlier. Return the total number of inliers.
C. Implement the function *RANSAC (matches , numMatches, numIterations, inlierThreshold, hom, homInv, image1Display, image2Display)*. This function takes a list of potentially matching points between two images and returns the homography transformation that relates them. To do this follow these steps:
    a. For "numIterations" iterations do the following:
        i. Randomly select 4 pairs of potentially matching points from "matches".
        ii. Compute the homography relating the four selected matches with the function *cv::findHomography(...)**. Using the computed homography, compute the number of inliers using "computeInlierCount".

iii. If this homography produces the highest number of inliers, store it as the best homography.

b. Given the highest scoring homography, once again find all the inliers. Compute a new refined homography using all of the inliers (not just using four points as you did previously.) Compute an inverse homography as well, and return their values in "hom" and "homInv".

c. Display the inlier matches using cv::drawMatches(...).

Required: Compute the Harris corner detector, find matches and run RANSAC for images "Rainier1.png" and "Rainier2.png". Save the images "3a.png" and "3b.png" of the found matches (use cv::drawMatches(...) to show the inlier matches). You should only see correct matches, i.e. , all the incorrect matches from the previous step should be removed. If you see all or some incorrect matches try running RANSAC with a larger number of iterations. You may try tuning the other parameters as well.

*****NOTE: The function cv::findHomography(...) can optionally use RANSAC internally, if that is enabled. You should not have RANSAC enabled in this function i.e. third input parameter should be 0.**

Step 4: Stitch the images together using the computed homography. Following these steps:

A. Implement the function *stitch(image1, image2, hom, homInv, stitchedImage)*. Follow these steps:

a. Compute the size of "stitchedImage. " To do this project the four corners of "image2" onto "image1" using *project(...)* and "homInv". Allocate the image.

b. Copy "image1" onto the "stitchedImage" at the right location.

c. For each pixel in "stitchedImage", project the point onto "image2". If it lies within image2's boundaries, add or blend the pixel's value to "stitchedImage. " When finding the value of image2's pixel use bilinear interpolation [cv::getRectSubPix(...)].

Required: Compute the Harris corner detector, find matches, run RANSAC and stitch the images "Rainier1.png" and "Rainier2.png". Save the stitched image as "4.png". It should look like the image "Stitched.png".

EXTRA CREDIT

1. Create a panorama that stitches together the six Mt. Rainier photographs, i.e. , Rainier1.png, ... Painier6.png. The final result should look similar to "AllStitched.png".
2. Create your own panorama using three or more images. You must capture the images yourself, and not find them on the web. I would recommend downsampling them before stitching, i.e. , make them approximately the same size as the images in the homework assignment.

3. Implement a new image descriptor or detector that can stitch the images "Hanging1.png" and "Hanging2.png". Save the Stitched image and the "match" images. In case you're wondering - No, you cannot rotate "Hanging2.png" by hand before processing, that's cheating.
4. Do a better job of blending the two images. That is, make the seams between the two images invisible. One possible method to do this is to use center-weighting. See Szeliski, Sections 9.3.2 - 9.3.4.
5. Implement a new image descriptor and/or detector that can stitch the images ND1.png and ND2.png. Save the Stitched image and the "match" images.

Submission (electronic submission through EAS only)

Please create a zip file containing your C/C++ code and a readme text file (.txt).
In the readme file document the features and functionality you have implemented, and anything else you want the grader to know i.e. control keys, keyboard/mouse shortcuts, etc.

Additional Information

- The assignment's images can be found here

Credits

Based on the two assignments developed by Ali Farhadi and Steve Seitz, respectively.