ELEC 474 - Final Project

Autostich

Martin Maly - 20068784
December 15th, 2021.

Modules

The following modules were used in the development of my code:

- **cv2**

This library was used for basic machine vision functions such as creating my SIFT instance and BFMatcher, reading images from files, as well as performing perspective transformation math.

- **glob**

This library was used to retrieve images from my global directory.

- **os**

This library was also used to retrieve images from my global directory.

- **matplotlib - pyplot**

This library was used to plot my stitched images after the stitching was complete.

- **numpy**

This library was used for basic number conversion and math.

- **uuid**

This library was used to give my images unique identifiers so they could be referenced.

It should be noted that none of these modules are my work, but they are all very standard libraries.

Declaration of originality

I declare I am the sole author of the submitted solution.

Feature extraction

For my feature extraction, I opted to use SIFT. I created a SIFTImage data structure, which holds an image, key points, descriptors, as well as an ID. I found it quite handy to keep all of these related pieces of data in the same data structure. In terms of finding the features between images, in order to increase efficiency I created a nested for loop, but ensured to make sure that I was not attempting to find features between the same images. I also ensured not to find features between two images that features had already been calculated for. This is because the features between image ‘A’ and image ‘B’ would be equivalent to the features between image ‘B’ and image ‘A’. This ensured that the costly detectAndCompute SIFT method would not be unnecessarily run.

Next, on all of the image pairs, that now had their key points and descriptors calculated, I got the matches using a BFMatcher, utilizing the knnMatch method. Once the matches were determined, I performed Lowe’s ratio test on all my matches in order to determine which matches were of high enough quality to deem a “good match”. Finally, I counted the number of “good matches”, and if there were over 6000 good matches, I deemed it very likely for those two images to overlap. It should be noted I also created a MatchedImagePair data structure. This was used to store both SIFTImage’s for a pair as well as the “good matches” between those two images.

Transformation Estimation / Transformation Application and Image Composition
In terms of transformation estimation, the algorithm is as follows. First, we stitch together the two images in the MatchedImagePair, and keep track of the images that have been stitched in an array. Then we loop through the other MatchedImagePair’s that we had previously determined, and if any of the MatchedImagePair’s contain an image that is in the array of stitched images, we find the matches for the other image from the same MatchedImagePair since we know it’s partner is in the stitched image. We keep doing this until the MatchedImagePair array is empty, and the entire image has been stitched.

The bulk of the logic for the actual image composition and transformation application methods is found within ‘StitchGoodMatchPair’. This method finds the homography from the two inputted images and thus uses that to indicate via a matrix, what transformations would need to be applied for one image to be overlapped with the other. Then, using this matrix, we call the ‘warpPerspective’ method which overlays one of the images over top of the other image. Since technically the images could be just barley overlapping on any of the sides, we make sure the canvas that these images are displayed in, is double the width and size of the input images. That way we can ensure no image data is lost.

Tests

Due to time constraints I wasn't able to run as many tests as I would've hoped. I was able to test on some of the St. James church photos. I found that my feature matching took about 15 minutes for 4 photos, 6 minutes for 3 photos, 3 minutes for 2 photos. As such the time would exponentially increase. The times for stitching the actual photos were slightly quicker, with times of about 13 minutes for 4 photos, 5 minutes for 3 photos, 2 minutes for 2 photos.

Success of my solution

My solution worked, but was found to be quite inefficient. So inn-efficient that I was unable to test with more than five photos at a time. My code did work well for five photos, but I was unable to test with more photos, so perhaps more photos may have revealed certain bugs or issues. To the extent of my knowledge, the task was achieved with my code.

Improvements

Ways that I could improve my solution in the future would be to blend the seams between stitched images so that they appear more seamless. In addition to this, instead of choosing the first "StitchGoodMatchPair" I iterate through that has a member in the stitched image, I could perhaps choose the StitchGoodMatchPair with the highest number of "Good matches", and iterate through that way until all the images are in the stitched image. In addition to this, I could try other feature detection methods such as ORB, or SURF, or perhaps even a combination of them, and see how that changes my result. Finally, I could perhaps use EXIF data from the images themselves that may contain pitch, roll, and yaw which would perhaps allow me to more confidently know that two images are overlapping instead of just using matches.