ELEC 474 Final Project: Auto-stitch with OpenCV

William Kennedy
20120570
Dec 1st, 2021

Declaration Of Originality

I, William Kennedy, declare that this project and all its contents is my own work and to my knowledge contains no materials previously published online by any person or institution, or peer. All work contains content developed through personal research and course content (i.e.-labs).

Inventory Of Source Code

Libraries

- All libraries are used from previous labs due to my comfortability with their modules

Extra Modules to Limit Code Clutter

- def load_folder(folder): This function extracts all images from a folder and converts them to opencv friendly arrays to be interpreted as images (new - original)
- def convert_image_bw(img): This function converts images from jpg RGB to GRAY to be used in the feature extraction process (used in previous labs - original)
- def show_image_window(img, title): This function is used to display any image in a new window for better quality/view (used in previous labs - original)
- def show_image_plt(img, title): This function takes advantage of matplotlib's imshow() builtin function and displays images quickly in the console (used in previous labs - original) (used in previous labs - original)
- def show_row_image_plt(images): this function is the same as above however displays a list of images (used in previous labs - original)
- def Match_Metric(src_img, BM_img): This function is considered my MATCH METRIC --> by only considering good matches as those that pass LOWES RATIO TEST and that are to the LEFT in image position on comparison image compared to position of source image (new - original)
- def remove_element(my_list, target): This function removes elements (in this case images or indexes) from a list (used in previous labs - original)
- def remove_border(img): This function is used to remove the border around images - this is useful to remove unecessary blackness around warped images (new - original)
- def remove_border_list(list_of_img): This function is the same as above however applies it on a list of warped images (new - original)

STEP 01 MATCH FEATURES: Feature Extraction and Feature Correspondence

- def feature_extraction(img1, img2): Using Scale Invariant Feature Detection detection and returning best match points (used in previous labs - original)
- def match_features(folder): STEP 01: The first step is to extract features from the images, and automatically establish feature correspondences between image pairs. (new - original)

STEP 02 ESTIMATE TRANSFORMATION: Calculating Transformations Between Matched Features

- def determine_image_pairs(sorted_list, folder): Initializing pairs of best matches to begin the warping process (new - original)
- def inital_pair_warp_R(rightward_pairs, folder): Taking pairs and starting the warping process (new - original)
- def warping_to_right_pair(right_img, left_img): Modified rightward stitching warp function that takes current already warped images and applies warping again. Also stitches the right warped image to left (new - original)

STEP 03 MERGE IMAGES: Applying Geometric and Radiometric Transformations

- def recursive_warping_R(rightward_pairs, folder): # This is now the process of taking a list of best matches and continously warping them and their outputs until each half is reduced to one warped image in direction of center (new - original)
- def flip_image(img): Flipping all LEFTWARD_PAIR IMAGES TO APPLY RIGHTWARD STITCHING AND FLIPPING RESULT (used in previous labs - original)
- def inital_pair_warp_L(leftward_pairs, folder): Apply initial warps from leftward_pairs list and flip all images and send to recursive warp (new - original)
- def recursive_warping_L(leftward_pairs, folder): # Same as R but with flipped imagesd
- def save_to_D3(img, folder): Saving Image in D3 Folder (new - original)
- def Auto_Stitch_Panorama(folder): # combining all functions to one main (new - original)

Feature Extraction

The main goal of feature extraction in this project is to identify images of best match in the unordered list (i.e.- closest in world location) to then have the smoothest and most efficient overall warping and stitching effect.

The feature extracting method chosen in this project was SIFT (Scale Invariant Feature Transform). This transform was chosen initially due to its robustness and efficiency at a large scale. The SIFT algorithm is broken down to: Scale-space peak selection, Keypoint localization, orientation assignment, Keypoint descriptors and matching Keypoint. SIFT also brings a benefit over other detection algorithms such as corner detection algorithms do to its Scale Invariance.

Once SIFT detection takes place the key points and descriptors are returned and then a brute force matcher is used to iterate through all matches and return two best matches per Keypoint (using the KNN algorithm when comparing Euclidean distance) then they are evaluated through Lowe's ratio tests with a threshold of 0.6 to determine if they are a match of good quality.

The overall feature extracting approach taken in this project was to iterate through the folder of images and starting with the first one compare against all other images and extract the key points that pass the Lowes ratio test (keeping two best matches for each Keypoint). Then the images were considered best pairs if the image being compared to source had the most key points in negative x direction compared to source image key points.

Then the list of best matches is split into two list, leftward pairs and rightward pairs to then apply transformations onto into the direction of the center image to give a proper panorama effect.

Transformation Estimation

The transformation estimation that was incorporated in the project was the homography transformation. The Key points that are extracted from the SIFT feature detection are then applied to the cv2.findHomography(). This function returns a 3x3 matrix that will convert all points in right image to the plane of the left image (pL = M*pR). RANSAC (Random Sample Concensous) is used to determine the matrix with the most resulting inliers.

Transformation Application and Image Composition

The process of merging all the now ordered list of images involves around RIGHTWARD STITCHING, this is when the right image is warped to plane of left image then left image is overlayed from [0,0] to its dimensions to help create a seamless transition. Then a remove border function is applied which converts image to black, white extracts max contour dimensions and removes all pixels outside of those dimensions. This is used on the rightward list regularly and then the leftward list with all images flipped so that the warps can be layered, and direction is still towards the center anchor image.

This rightward stitching approach for each half list perform initial warping of perspectives of right image plane to left image plane using the homography matrix extracted from Step 2. With each initial warp list perform warping of images to the right which will create a gradual warp from center outward. This way each side-by-side portion grows with more layers added and is gradually warped towards center until each half is one resulting image.

Then finally each side is stitched together to create one full resulting panorama from an unordered list of images. This is obviously in an ideal scenario if images are merged and warped perfectly. There will be a series of blurring and pixel averaging approaches to help merge the seams of each image.

Further Relevant Details

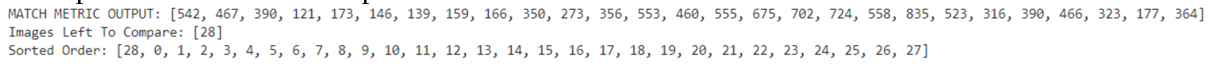
N/A

Testing

The tests executed were initially broken down into 3 specific rounds for each folder. Each round is from the 3 main stages of the project. The feature extraction, transformation estimation and image merging processes. These are all averaged and rounded since certain tests times fluctuated greatly when computer had more tasks

FOLDER	Feature Extraction	Transform Estimation	Image Merging
Office2	3000s	2000s	150s
StJames	3300s	1700s	100s
WLH	4200s	2200s	

Example of match metric output:



```
MATCH METRIC OUTPUT: [542, 467, 390, 121, 173, 146, 139, 159, 166, 350, 273, 356, 553, 460, 555, 675, 702, 724, 558, 835, 523, 316, 390, 466, 323, 177, 364]
 Images Left To Compare: [28]
 Sorted Order: [28, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27]
```

Reflection

I believe that my solution is effective in the execution of this project. I believe that my feature extraction and transformation implementations are both effective and robust for different image sets however, my overall solution is not very efficient in terms of time or memory. The main limitations I endured during the project was overall runtimes consuming a lot of time that could have been spent debugging. This also comes down to my choice of effective and detailed search methods to produce more accurate results rather than faster, less effective measures such as corner detection.

Improvements

To elaborate more on my reflection, I found that I was going in the right direction with my overall approach however I found that my overall solution has some repetitive parts that contribute to longer runtimes and more memory consumption. I wanted to ensure that my execution was broken down to simple steps that were easy to follow and easy to debug.

Possible improvements could have been to retain all the sift data between images, that way they would not have to be recomputed during the warping process. Another improvement would have been to compute the homography at the same time as the best feature points in previous improvement. So once again the computation would not have to be recomputed during each of the warping process. Therefore, the main takeaway is to improve overall data management and flow execution.

Another major improvement could have been more focus on just modifying the Homography matrix values rather than whole image computations to reduce runtime. On top of that to potentially improve runtime an option is to use more built-in functions such as cv2.flip() to flip images instead of traversing all pixels – once again a runtime improvement.