Functionalities implemented:

<u>File: A2code.py</u>

1. Harris corner detector
   i. ***harris_points (input_img,smallval,threshold,resize)***
   - *Used Sobel to created gradient matrix towards x and y axis.*
   - *Computed Ixx, Iyy and Ixy and further applied gaussian blur of 3x3 matrix.*
   - *Created a window of 3x3 to add the values Ixx, Iyy and Ixy respectively across the window to create Harris matrix.*
   - *Computed determinant, trace and corner strength. For trace, added a very small value to avoid division by zero.*
   - *<u>Formula</u>: Corner Strength= determinant/trace*
   - *Further applied maximum suppression and adaptive supression.*

   ii. ***max_suppression(input_img,threshold)***
   - *Created a window of 3x3 and moved it across the Corner Strength matrix created by the Harrix detection.*
   - *For each window movement, took the maximum across the window and supressed others.*
   - *While choosing the maximum also applied the check that the value is greater than threshold.*
   - *Returned the Matrix with local maximas and further applied adaptive non-maximum supression.*

   iii. ***adaptive_sus(points,resize)***
   - *Applied the algorithm given in **MOPS paper** to spread the points and matches in the pictures.*
   - *Calculated the distance between various and kept the value as radius if the point has corner strength smaller than 0.9 of other's point corner strength.*
   - *Sorted the points within decreasing order and resized the points array for the given value.*

2. Matching the interest points between two images
   i. ***create_mag_angle(inp_img)***
   - *Calculated gradients along x and y axis.*
   - *Computed magnitude and angle using these gradient values for each pixel along the matrix.*
   - *<u>Magnitude Formula</u>: ((g_x**2) + (g_y**2)) **0.5*
   - *<u>Angle Formula</u>: arctan2(g_y,g_x)*

   ii. ***rotateinvariance(mag,angle)***
   - *Used the concept listed in the SIFT paper given in lecture.*
   - *Created 36 bin histogram to get the dominant orientation in 16x16 matrix.*
   - *Took all the values or orientations 80% to 100% of the highest one to create multiple keypoints with different orientations.*
   - *Subtracted these oriented creating multiple matrices rotated along the x-axis by subtracting dominant orientation.*

iii. sift(input_img, points)

- *Used Gaussian blur to remove the noise and provide some scale invariance by taking sigma=1.5.*
- *Computed magnitude and angles along the matrix.*
- *Created a 16x16 matrix of both above and provided rotation invariance (given above).*
- *Created a descriptor for 128 size.*
- *Divided 16x16 matrix in 16 4x4 blocks.*
- *Created 8sized histogram for each block.*
- *Histogram calculated the orientations by adding magnitudes for angle bins 0-45, 45-90, 90-135, 135-180, 180-225 and 225-0.*
- *Histogram was further included in 128 sized descriptors.*
- *Descriptor was normalized and clipped to provide Contrast invariance with max value of 0.2.*

iv. create_matchings(pts_1, pts_2)

- *Calculated SSD for different combinations of feature descriptor from both images.*
- *Set a boundary condition for the SSD < 0.5.*
- *Further calculated Ratio Test: SSD(smallest)/SSD(second smallest)*
- *Set a boundary for Ratio Test: SSD1/SSD2 < 0.6.*

Extra Functions

i. a2start()

- *Runs all the functions to display the matches and interest points for the images.*
- *Threshold: 20000000*

ii. image_features(threshold, small_val, adapt_resize, imgName, var, outname)

- *Use the harris_points function to get the interest points for an image.*
- *Use sift function to get the descriptors for an image.*

File: code.py

3. Homography between the images using RANSAC

i. find_matches(inp_img1,inp_img2)

- *Used opencv Sift detector and descriptor to find the interest points and descriptors.*
- *Used BFMatcher to match the interest points using the above descriptors.*
- *Sorted the matches as per their distance.*
- *Referred to the pages given in references (1, 2 and 3).*

ii. get_first_second(mts, insPts1, insPts2)

- *From the given matches and interest points created 2 arrays.*
- *First array for the first image with the points (x, y) for each match (using query index).*
- *Second array for the second image with the points (x, y) for each match (using train index).*

iii. project(col1, row1, hom)

- *Created a floating-point array with x1(col), y1(row) values and 1.*
- *Used the dot product for multiplying homography and the above array to get the output array.*
- *Divided the third value in output array from 1st and 2nd to get x2 and y2.*
- *Added a small value in the denominator to avoid zero division error.*

iv. ***computeInlierCount(hom, matches, first, second, thresh)***

- *Points of image1 are projected using project function to get projected points on image2.*
- *Distance is calculated between actual points of image 2 matched with points of image 1 and projected points of image 2.*
- *Matches are appended into inliers_match list if the distance is smaller than inlier threshold.*
- *Inlier threshold taken is **5**.*

v. ***RANSAC (matches , numIterations, inlierThresh, insPts1, insPts2)***

- *Used Random lib to get 4 random matches from the list.*
- *Used get_first_second function to get the points for first and second keypoints per match.*
- *Used these arrays to find the homography.*
- *Used computeInlierCount function to get the inlier per homography.*
- *Best homography with highest inliers is selected.*
- *New inliers are selected as per the best homography.*
- *Using these new inliers, new homography is calculated.*
- *No of iterations are **1000**.*
- *Inverse homography is calculated using numpy.linalg.inv function of numpy and returned.*

4. Stitch the images

i. ***get_corners(inp_img1, inp_img2, hom, hom_inverse)***

- *Corner points of image 2 are projects over image 1.*
- *Size of the stitched image is calculated.*
- *Additional parameters to determine how to shift image 1 and image 2.*

ii. ***stitch(inp_img1, inp_img2, hom, hom_inverse)***

- *get_corners function is used to get the size of the stitch image.*
- *Image 1 is pasted over stitched image using additional parameters returned by get_corners function.*
- *All the pixel points in stitched image are projected over image 2.*
- *If these projected points lie within image 2 borders, then getRectSubPix function is used get the pixel value for image 2 using bilinear interpolation.*

5. Creating a panorama

- *All the images are extracted into an inp_img list.*
- *This list is iterated and one by one is stitched together and a panorama is created.*

6. Creating own panorama using three or more images

- *I have taken 2 samples for 3 images each.*
- *The outputs are attached below.*

Extra Functions

iii. ***rainerBoxRansac(inlierThresh, iterations)***

- *To get the outputs for Rainer1 and Rainer2 as required in step 3 and 4.*

iv. ***start()***

- *To run all the functions.*

Steps of Implementation:

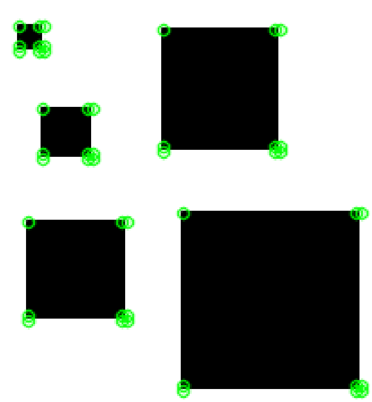
- **Step 1** is implemented in A2code file using above functions. The corner points for Boxes, Rainer1 and Rainer2 images are displayed in images **1a.png, 1b.png and 1c.png**. These outputs are attached below and are saved in **results folder**. Each time code runs they are saved in code folder as well with same names.
- **Step 2** is implemented in A2code file using above functions. The matches for Rainer1 and Rainer2 images is displayed in image **2.png**. The output is attached below and is saved in **results folder**. Each time code runs it is saved in code folder as well with same name.
- **Step 3** is implemented in code.py using functions listed above. The updated matches in Ransac for Rainer1 and Rainer2 images is displayed in image **3.png**. The output is attached below and is saved in **results folder**. Each time code runs it is saved in code folder as well with same name.
- **Step 4** is implemented in code.py using functions listed above. The stitched image for Rainer1 and Rainer2 images is displayed in image **4.png**. The output is attached below and is saved in **results folder**. Each time code runs it is saved in code folder as well with same name.
- The panorama for all Rainer images is saved as **Rainer Panorama.png**. The panorama for all *MelakwaLake* images are saved as **MelakwaLake Panorama.png**. These outputs are attached below and are saved in **results folder**. For panoramas, the matches both with & without RANSAC and stitched for each iteration are saved in “Extra output for panoroma” folder.
- The **input images** for own panoramas are saved in **building & road folders** and the panoramas with respective names are saved in **results folder** and attached below.

How to Run the code:

- The python version is 3.5.1 and open-contrib version is 3.3.1.
- Go to “code and input” folder, if you want to see Rainer panorama the simply run the code.py.
- Else open code.py, then, go to “Start” function select the list of images and uncomment it if required.
- Run the file.
- The 1a, 1b, 1c, 2, 3, 4 images are saved in results and “code and input” folders.
- RANSAC images are saved in “Extra output for panoroma” folder.
- Final stitched image is displayed and saved as panoroma.png.

Results:

i. 1a.png



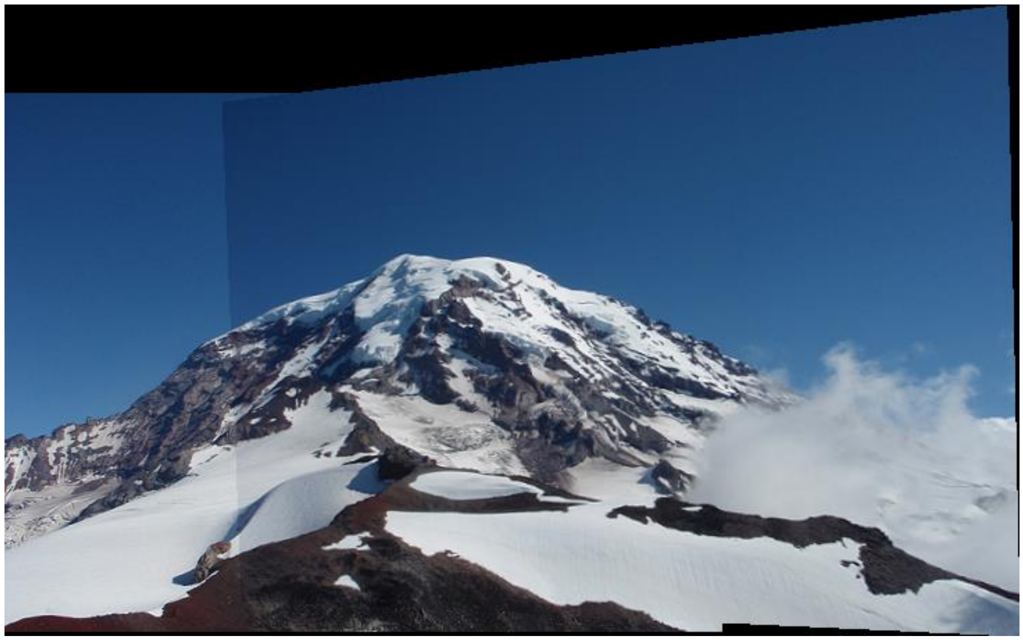
ii. *1b.png*

iii. *1c.png*

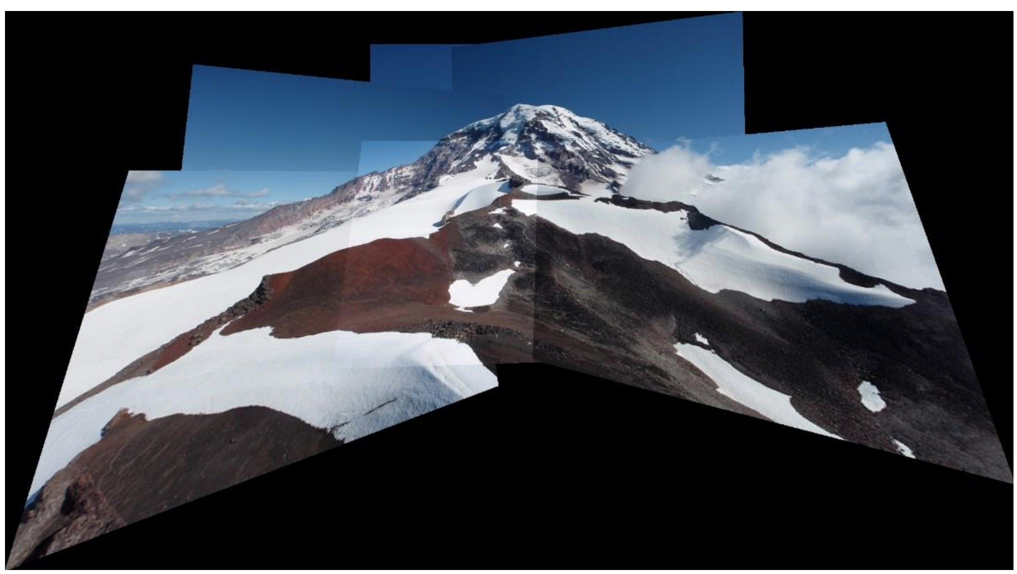
iv. 2.png

v. 3.png

vi. 4.png



vii. Rainer Panorama



viii. MelakwaLake Panorama

ix. Own Panorama 1: Building Panorama

x. Own Panorama 2: Road Panorama

References:

- https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html
- https://docs.opencv.org/master/dc/dc3/tutorial_py_matcher.html
- https://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_feature2d/py_matcher/py_matcher.html