Student: Li Hongguang (A0233309L)

Implementation: detect_keypoints()

1. Read image with **cv2.imread()**
2. Detect keypoints and descriptors using **cv2.SIFT_create()** followed by **sift.detectAndCompute**()

Implementation: create_feature_matches()

1. Run **cv2.BFMatcher()** and **matcher.knnMatch()** to match features
2. Iterate through pairs of matches {**m, n**}, if **m.distance < lowe_ratio * n.distance**, it is considered a good match and appended to **good_matches** to be returned

Implementation: create_ransac_matches()

1. Call **cv2.findEssentialMat()** to extract essential matrix and is_inlier mask

Implementation: create_scene_graph()

1. Loop through each set of image pair {**i, j**}, and check if numpy file exists for image pair using **np.load()**

1.1 If there are inlier correspondences between the two images, count the number of inliers and add edge between nodes **i** and **j**

1.2 If **np.load()** throws OSError (no correspondences), continue to next image pair.

Implementation: get_init_image_ids()

1. Initialize variable **max_inliers** to 0 to keep track of current highest number of inliers
2. Loop through every image pair {**i, j**}, get number of matches using **load_matches()**
3. If number of matches > **max_inliers**, set it as the new **max_inliers** value and overwrite previous **max_pair** and set {**i, j**} as the new **max_pair**

Implementation: get_init_extrinsics()

1. Call **cv2.recoverPose()** to retrieve relative rotation matrix **R** and translation vector **t** between initializing cameras
2. Call **np.concatenate()** to create 3x4 projection matrix [**R|t]** for one camera, assuming other is canonical

Implementation: get_next_pair()

1. Similar to **get_init_image_ids()**, except outer loop iterates through registered image ids **i**, and inner loop iterates through adjacent nodes **j**, checking that **j** is not registered before executing step 3 of **get_init_image_ids()**

Implementation: solve_pnp()

1. Call **cv2.solvePnP()** to extract rotation vector **R** and translation vector **tvec**
2. Convert **R** to rotation matrix **rotation_mtx** using **cv2.Rodrigues()**
3. Compute reprojection residuals using **get_reprojection_residuals()**

Implementation: get_reprojection_residuals()

1. Convert each 3D point to homogeneous coordinates
2. For each homogeneous 3D point coordinate, apply projection matrix to get projected homogeneous 2D coordinates
3. For each projected homogeneous 2D coordinate, we normalize the z-coordinate to 1
4. For each normalized projected homogeneous 2D coordinate, we take the difference with the ground truth homogeneous 2D coordinate and find euclidean distance using **np.linalg.norm()**

Implementation: add_points3d()

1. Call **triangulate()** to get 3d coordinates computed from triangulation

Student: Li Hongguang (A0233309L)

Implementation: compute_ba_residuals()

1. Similar to **get_reprojection_residuals()**, except with the use of **np.einsum()** to avoid the use of loops for doing matrix multiplication between camera intrinsics matrix and 3D tensor (array of projection matrices).

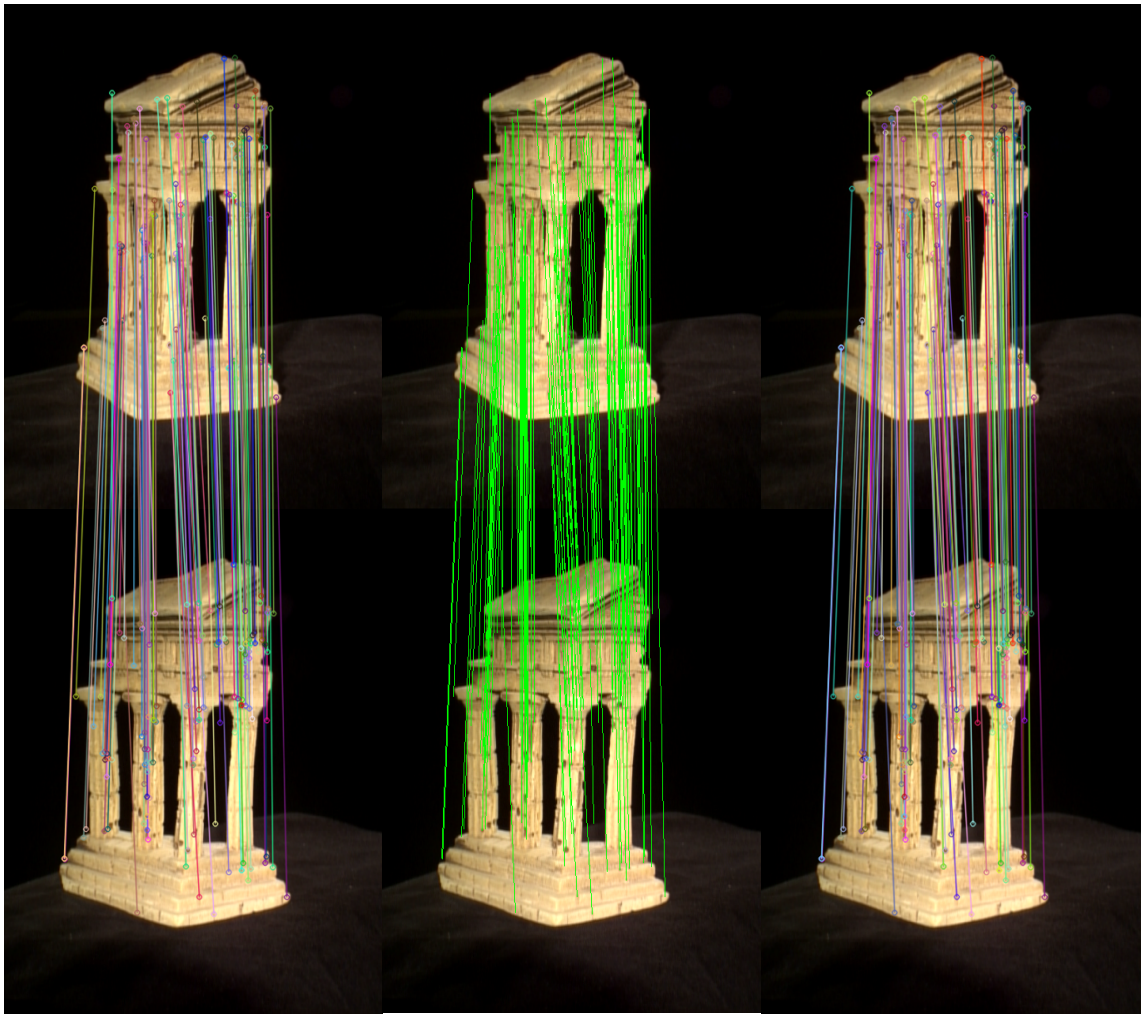

Fig 1. Mine(left) vs RANSAC(Middle) vs TA(right) *templeR0013_templeR0015.png*

Note: Color differences of the lines between my result and the TA might lead to test.py returning "**False"** for bf-match-images

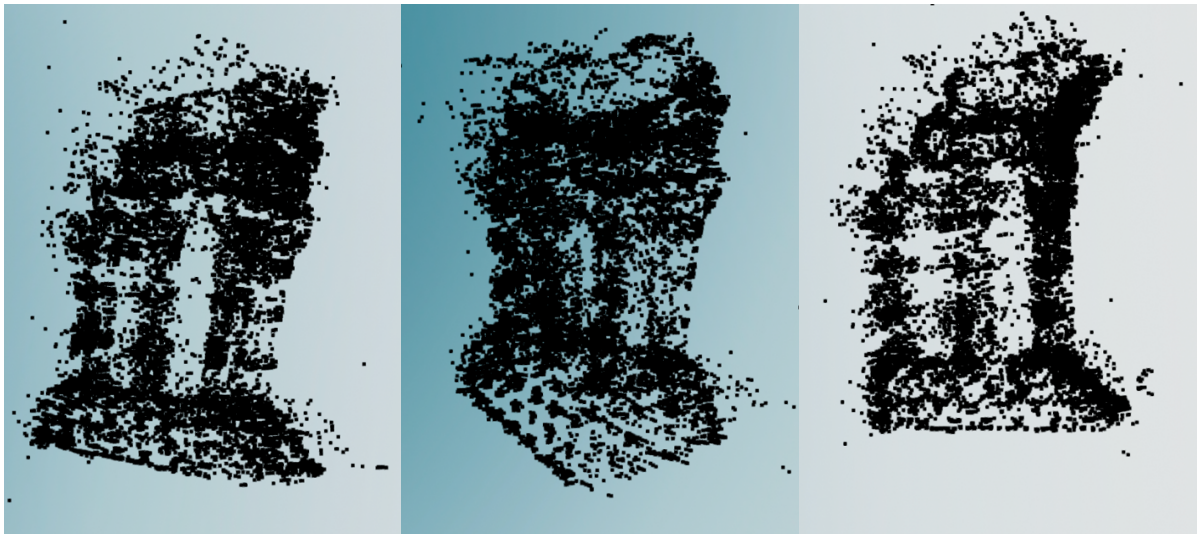

Fig 2. Results of incremental SfM (no BA) for temple dataset

Fig 3. Results of incremental SfM (no BA) for mini-temple dataset

Fig 4. Results of increamental SfM (with BA) for mini-temple dataset