preprocessing.py

detect_keypoints: Make use of **cv2.SIFT_create()** and **cv2.detectAndCompute** to find all the keypoints

create_feature_matches: Make use of **cv2.BFMatcher** to find good matches

create_ransac_matches: Make use of **cv2.findEssentialMat** to filter out outliers.

create_scene_graph: For each pair of image id, we read the ransac match file to count the number of inliers and then, pick the pair that produce the most number of inliners to add into the scene graph.

sfm.py

git_init_image_ids: We loop though all the edges/pair of image and find the edges/pair that produce the most inliers and choose it to be our starting point.

get_init_extrinsics: For the first extrinsic matrix, we assume it to be $[I|0]$. For the second extrinsic matrix, We make use of **cv2.recoverPose** to calculate the rotation matrix and translation vector from the essential matrix. Then we concatenate the rotation matrix and translation vector to be the extrinsic matrix of the second matrix.

get_next_pair: Similar to **get_init_extrinsics**, we find the pair of registered image and unregistered image which produce the most inliers to be the next pair.

solve_PnP: We first get the rotation matrix and translation vector using **cv2.solvePnP** and **cv2.Rodrigues** and calculate the error. The way to calculate the error is to project the 3D points into an image using given intrinsic and extrinsic matrix and find the distance between the calculated 2d point and the corresponding 2d point.

add_points3d: Directly find the new 3d points using the triangulate function

bundle_adjustment.py

compute_ba_residuals: Similar to find the re-projection error, we project the given 3d points using correspondence matrix and then calculate the distance between the calculated 2d points and origin 2d points.

Note

If the code encounter type error when executing cv2.Rodrigues, please uncomment the code in line 376 in sfm.py.