Assignment 5: Sparse 3D Reconstruction

Due Date: 13 December 2023, 23:59

Extension granted until: 20 December 2024, 23:59

Introduction

For the last assignment, you will reconstruct a 3D scene based on a stereo pair of images. First, you will revisit the RANSAC algorithm to estimate the fundamental matrix between the two images, filter out the outlier correspondences and calculate the epilines on both images. During the RANSAC process, you will need to pre-condition the linear system that calculates the epipolar constraint, so as to get a better estimation of the fundamental matrix and more accurate epilines. Finally, given the camera intrinsics you will estimate the essential matrix and the camera extrinsics in order to reconstruct the 3D scene by triangulating the given pairs of good 2D correspondences between the two images.

Preliminary step

For all checks that are needed for the following questions, you should use the `assert()` function. All intermediate and final results should be plotted using the `matplotlib` package. In order for your results to be reproducible use the following seed `np.random.seed(123)`. For visualizing your results you can use any function found in the provided `helper.py` script.

1 Fundamental Matrix Linear System (10 points)

For this question you are asked to implement the following function:

```
def fundamental_matrix_linear_system(pts1, pts2):
    """
        Create linear equations for estimating the fundamental matrix in matrix form
    :param pts1: numpy.array(float), an array Nx2 that holds the source image points
    :param pts2: numpy.array(float), an array Nx2 that holds the destination image points
    :return:
        A: numpy.array(float), an array Nx8 that holds the left side coefficients of the linear equations
        b: numpy.array(float), an array Nx1 that holds the right side coefficients of the linear equations
    """
```

Using a number of corresponding points from the source (`pts1`) and destination (`pts2`) images as input, this function will return the array $\boldsymbol{A} \in R^{N\times 8}$ and vector $\boldsymbol{b} \in R^{N\times 1}$ of the linear system $\boldsymbol{Ax} = \boldsymbol{b}$, where $\boldsymbol{x} \in R^{8\times 1}$ is a vector that holds the parameters of the fundamental matrix (remember that f_{33} can be set to 1). You should check that N is at least 8 and that both `pts1` and `pts2` are two dimensional arrays and that their first dimension is equal to N and the second is equal to 2.

2 Compute Epipolar Lines (10 points)

In this part of the assignment you need to implement a function that calculates the epipolar lines that correspond to a set of points, under the epipolar constraint. This function should follow the function signature below:

```
def compute_correspond_epilines(points, which_image, F):
    """
        For points in an image of a stereo pair, computes the corresponding epilines in the other image
    :param points: numpy.array(float), an array Nx2 that holds the image points
    :param which_image: int, index of the image (1 or 2) that contains the points
    :param F: numpy.array(float), fundamental matrix between the stereo pair
    :return:
        epilines: numpy.array(float): an array Nx3 that holds the coefficients of the corresponding
                                      epipolar lines
    """
```

According to which image the `points` belong to, it should calculate the corresponding epilines under the epipolar constraint expressed by the fundamental matrix. You should check that `points` is a two dimensional array where the second dimension is equal to 2. Also, the `which_image` parameter should take only one value from the set $\{1,2\}$ and the fundamental matrix should be a 3×3 array. The epilines should be returned in the format (a,b,c), which are the coefficients of the line $l_i : ax+by+c = 0$ that correspond to the point $\boldsymbol{p}_i \in$ `points`. Note that the coefficients should be normalized so that $a^2 + b^2 = 1$.

3 Estimate Fundamental Matrix (30 points)

For this part, you need to calculate the fundamental matrix for a stereo pair of images. First you need to load the two images `image1.png` and `image2.png` that are found under the `data/` directory. Then you will need to extract SIFT features from their grayscale counterparts and find some good correspondences between the left and right image, by applying the ratio test with a `threshold=0.75`. Using these correspondences you will estimate the fundamental matrix between the stereo pair using the RANSAC algorithm. In order to do this you will need to implement the following function:

```
def ransac(src_points, dst_points, ransac_reproj_threshold=2, max_iters=500, inlier_ratio=0.8):
    """
        Calculate the set of inlier correspondences w.r.t. fundamental matrix, using the RANSAC method.
    :param src_points: numpy.array(float), an Nx2 array that holds the coordinates of the points in the
                                           source image
    :param dst_points: numpy.array(float), an Nx2 array that holds the coordinates of the points in the
                                           destination image
    :param ransac_reproj_threshold: float, maximum allowed reprojection error to treat a point-epiline pair
                                           as an inlier
    :param max_iters: int, the maximum number of RANSAC iterations
    :param inlier_ratio: float, ratio of inliers w.r.t. total number of correspondences
    :return:
        F: numpy.array(float), the estimated fundamental matrix using linear least-squares
        mask: numpy.array(uint8), mask that denotes the inlier correspondences
    """
```

Your implementation should do the followings checks:

- `src_points` and `dst_points` should have the same dimensions
- `ransac_reproj_threshold`: should be a positive number (zero included)
- `max_iters`: should be a non-zero positive number
- `inlier_ratio`: should lie within the range $[0,1]$

First by selecting a random seed group of corresponding points you will need to construct the linear equations system that is required to solve for the fundamental matrix. For this you should use the function that you implemented in Question 1. After acquiring the linear system you need to solve for $\boldsymbol{F}$ - the fundamental matrix. There will be some cases where $\boldsymbol{A}$ is singular so you will not be able to calculate its inverse matrix. In order to overcome this you need to try to solve for $\boldsymbol{F}$ within a `try-except` block. If `np.linalg.LinAlgError` is raised you should check if `'Singular matrix'` is in the error message. If yes, then you should skip that particular iteration of RANSAC and continue to the next one. If this is not the case then you should `raise` the error. Then you will calculate the corresponding epipolar lines in the destination image using your function (not OpenCV's) from Question 2. For each point in the destination image you will need to check if it agrees with its corresponding epiline, i.e., check if their distance is smaller than the reprojection threshold (for calculating the distance from a point to a line you can use this formula). Then you should follow the rest steps of RANSAC in order to decide if you have a good consensus set or not. The returned fundamental matrix should be refined using the inliers set (along with the seed group that produced it) in a linear least square manner.

You should compare your implementation with OpenCV's using the following input parameters

```
cv2.findFundamentalMat(<source_points>, <dst_points>, cv2.FM_RANSAC, ransacReprojThreshold=0.5)
```

For your implementation the reprojection threshold should be set to 0.5, the maximum number of iterations should be equal to 5000 and the inlier ratio should be equal to 0.9. After you estimate the fundamental matrix using both implementations, you will need to calculate the epipolar lines for both images using your function from Question 2 (do not use `cv2.computeCorrespondEpilines()`).

For visualizing your results, first you will need to plot the input images in a 1×2 grid and the correspondences after the ratio test in a separate figure. Then plot the inlier correspondences from your and OpenCV's implementation of RANSAC in a 2×1 grid. Finally, plot the epipolar lines of the image 2 along with their corresponding points in image 1 (use the same colors for each line-point pair) from both implementations in a 2×2 grid (see the `drawlines()` function in the provided `helper.py` script). Do the same for the epipolar lines of the image 1 and their corresponding points on image 2.

4 Normalize Points (20 points)

In many cases the estimated fundamental matrix is not precise, since the distance of a point from its epiline is large or the point of intersection of the epilines is not close to their corresponding epipole. In order to reduce this error we need to normalize the points that are used to construct the $\boldsymbol{A}$ matrix (either in the linear or linear least squares case). This means to pre-condition $\boldsymbol{A}$ by applying a translation and scaling transformation on the image coordinates. First, the origin of the new coordinate system should be located at the centroid of the image points i.e, translate the set of points $\boldsymbol{w}$ for each image by their average point - centroid $\bar{\boldsymbol{w}}$. Then the mean square distance of the translated image points from the origin should be 2 pixels i.e., scale the translated points $\boldsymbol{w}_i$ by a scaling factor:

$$s = \left(\frac{2N}{\sum_{i=1}^{N} ||\boldsymbol{w}_i - \bar{\boldsymbol{w}}||_2^2} \right)^{1/2} \tag{1}$$

where N is the number of correspondences that are used to calculate $\boldsymbol{F}$. These transformations for both set of points (source points $\boldsymbol{w}$ and destination points $\boldsymbol{w}'$) can be expressed by two transformation matrices $\boldsymbol{M}$ and $\boldsymbol{M}'$ and the coordinates can be normalized as:

$$\boldsymbol{q}_i = \boldsymbol{M}\boldsymbol{w}_i \quad \boldsymbol{q}'_j = \boldsymbol{M}'\boldsymbol{w}'_i \tag{2}$$

Keep in mind that $\boldsymbol{M}$ and $\boldsymbol{M}'$ should consider first the translation transformation and then the scaling transformation. Using the normalized points you can create the linear system (or linear least squares) and solve for $\boldsymbol{F}_q$ which is pre-conditioned on the transformations $\boldsymbol{M}$ and $\boldsymbol{M}'$. In order for $\boldsymbol{F}_q$ to be usable on the original coordinates you can de-normalize it as:

$$\boldsymbol{F} = \boldsymbol{M}'^{\top} \boldsymbol{F}_q \boldsymbol{M} \tag{3}$$

Using $\boldsymbol{F}$ you can estimate the epipolar lines on both images using the original point coordinates. For this you need to implement the following function:

```
def points_normalization(pts1, pts2):
    """
        Normalize points so that each coordinate system is located at the centroid of the image points and
        the mean square distance of the transformed image points from the origin should be 2 pixels
    :param pts1: numpy.array(float), an Nx2 array that holds the source image points
    :param pts2: numpy.array(float), an Nx2 array that holds the destination image point
    :return:
        pts1_normalized: numpy.array(float), an Nx2 array with the transformed source image points
        pts2_normalized: numpy.array(float), an Nx2 array with the transformed destination image points
        M1: numpy.array(float), an 3x3 array - transformation for source image
        M2: numpy.array(float), an 3x3 array - transformation for destination image
    """
```

You should check that `pts1` and `pts2` are two dimensional arrays, that they hold the same number of points and that those points are in $2D$ space. This function will be used before estimating the fundamental matrix during the RANSAC iterations and at the least squares refinement step. The estimated fundamental matrix using the normalized points should be de-normalized with the use of transformations `M1` and `M2`. Finally, you need to update your RANSAC implementation in order for the user to be able to choose if the points are going to be normalized or not, by adding an extra input argument (`normalize=False`).

5 2D Points Triangulation (30 points)

For the last question you will triangulate 2D corresponding points from the stereo pair of images, in order to partially reconstruct the 3D scene. First, you will need to estimate the essential matrix based on the fundamental matrix that you have calculated from your implementation and OpenCV. For this step, the intrinsics matrices $\boldsymbol{K}_1$ and $\boldsymbol{K}_2$ for both cameras are given in the `intrinsic.npz` file. You can use the `numpy.load()` function in order to load the camera intrinsics. Next, you will decompose both essential matrices (from your implementation and from OpenCV) to possible rotations $\boldsymbol{R}_1, \boldsymbol{R}_2$ and translation $\boldsymbol{t}$ using the `cv2.decomposeEssentialMat()` function.

Based on these, you will form four possible extrinsics matrices for camera 2 $\boldsymbol{E}_2 \in R^{3\times4}$ for each essential matrix $([\boldsymbol{R}_1,\boldsymbol{t}],[\boldsymbol{R}_1,-\boldsymbol{t}],[\boldsymbol{R}_2,\boldsymbol{t}],[\boldsymbol{R}_2,-\boldsymbol{t}])$. Finally, using each possible extrinsic matrix you will triangulate 2D point correspondences found in the images of the stereo pair in order to reconstruct their corresponding 3D points. Since the SIFT correspondences are quite sparse, you should load and use the correspondences that are found in the provided `good_correspondences.npz` file (use the `drawcorrespondences()` function from the `helper.py` script to visualize them). For the triangulation process, you will implement the following function:

```
def triangulation(P1, pts1, P2, pts2):
    """
        Triangulate pairs of 2D points in the images to a set of 3D points
    :param P1: numpy.array(float), an array 3x4 that holds the projection matrix of camera 1
    :param pts1: numpy.array(float), an array Nx2 that holds the 2D points on image 1
    :param P2: numpy.array(float), an array 3x4 that holds the projection matrix of camera 2
    :param pts2: numpy.array(float), an array Nx2 that holds the 2D points on image 2
    :return:
        pts3d: numpy.array(float), an array Nx3 that holds the reconstructed 3D points
    """
```

You should check that `pts1` and `pts2` are two dimensional arrays, that they hold the same number of points and that those points are in $2D$ space. Additionally, you should check that `P1` and `P2` are 3×4 arrays. This function will iterate over the point correspondences, and for each pair of 2D points it will reconstruct their 3D corresponding point via triangulation given the two projection matrices $\boldsymbol{P}_1$ and $\boldsymbol{P}_2$. You should concider that camera 1 coincides with the world origin, so its projection matrix can be expressed as $\boldsymbol{P}_1 = \boldsymbol{K}_1[\boldsymbol{I}_{3\times3},\boldsymbol{0}_{3\times1}]$. Since each essential matrix is decomposed into four possible extrinsics matrices for camera 2, you will need to triangulate point correspondences using each possible projection matrix $\boldsymbol{P}_2 = \boldsymbol{K}_2\boldsymbol{E}_2$ and retain the set of 3D reconstructed points that are in front of both cameras. Finally, you should visualize the correct set of 3D reconstructed points for both implementations (yours and OpenCV's), in two separate 3D plots, by using the `plot_3d_points()` (see `helper.py` script).

Instructions

- The assignment is due by **Friday December 13th, at 23:59** (see the course's GitHub repository for more details regarding the late assignment policy).
- The assignment should be submitted **only** through Moodle. Compress all the needed **source code** files, e.g. all *.py files into a .zip file and name it as follows "**Lab_Assignment_4_<UC-ID-Number>.zip**".
- For further questions you can create an issue on the course's GitHub repository.
- All lab assignments should be conducted **independently**.
- The Moss system will be used to detect code similarity.
- Plagiarism and/or cheating will be punished with a grade of zero for the current assignment. A second offense will carry additional penalties.