ETH Zürich
Eidgenössische Technische Hochschule Zürich

Computer Vision
Department of Computer Science

Lab 7:
Structure from Motion & Model Fitting

Christos Dimopoulos - 23-941-834
cdimopoulos@student.ethz.ch

December, 2023

Contents

Task 1: Structure from Motion

In this assignment, we will produce a reconstruction of a small scene using Structure from Motion (SfM) methods. We combine realtive pose estimation, absolute pose estimation and point triangulation to a simplified Structure from Motion pipeline. The code framework for this is provided in the file sfm.py. We are also provided with extracted keypoints and features matches. These matches are already geometrically verified and do not contain outliers since we would need extra steps to handle outliers in the pipeline.

1. Essential matrix estimation

We are provided with the camera intrinsics matrix K, so we can compute the relative pose between the first images from the essential matrix E. We use DLT to reformulate the constraint $\hat{x}_1^T E \hat{x}_2 = 0$:

$$[\hat{x}_1^1\hat{x}_2^1, \hat{x}_1^1\hat{y}_2^1, \hat{x}_1^1, \hat{y}_1^1\hat{x}_2^1, \hat{y}_1^1\hat{y}_2^1, \hat{y}_1^1, \hat{x}_2^1, \hat{y}_2^1, 1][e_{11}, e_{12}, e_{13}, e_{21}, e_{22}, e_{23}e_{31}, e_{32}, e_{33}]^T = 0 \tag{1}$$

We then apply Singular Value Decomposition on our estimated Essential matrix and set the singular values accordingly, i.e., make the first two singular values to be equal to 1 (since E is up to scale) and the third singular value as 0.

$$SVD(\hat{E}) = U\sigma V^T = U \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T \tag{2}$$

2. Point Triangulation

To screen out incorrect 3D points, a straightforward validation involves examining the Z coordinates (which should be >0) of the points when projected onto each camera frame. By removing the associated points and correspondences through a mask, the task is accomplished.

3. Finding the correct decomposition

The essential matrix that is calculated is then decomposed into an (R, t) pose by using SVD and gives four possible solutions (four poses), that all fulfill the requirements of the essential matrix. For each pose (R, t), we do triangulation, and ultimately pick the pose with the greatest number of points in front of the camera. The 3D points are later tied with indices to both images as correspondences.

4. Absolute pose estimation

We fill the missing part of the function *EstimateImagePose* to get the constraint matrix and iteratively use calculated 3D points to calibrate the next camera.

5. Map extension

Now, each additional image can be employed to deduce its orientation by utilizing the 2D-3D correspondences it shares with previously established points. Subsequently, fresh 3D points can be determined by triangulating them with an already registered image. These newly obtained points are then stored globally to expand the point cloud and cataloged as new correspondences for the images.

6. Result

We save the final result, after using pairs of images 3-4 and 5-6 as initial images:

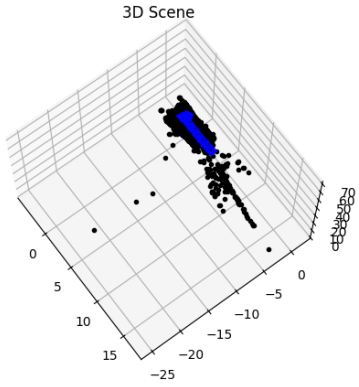

Figure 1: Initial Images 3 and 4.

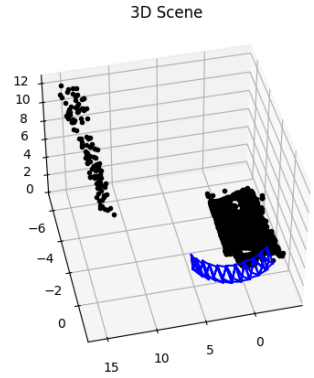

Figure 2: Initial Images 5 and 6.

Model Fitting

In this assignment, we will learn how to use RANSAC (RANdom SAmple Consensus) for robust model fitting. We will work on the simple case of 2D line estimation. With a ground truth linear model y = kx + b, we generate a point set (x noisy, y noisy) with the linear model and add noise to it. Besides, we also add outliers to the point set.

1. Least-squares Solution

First of all, we implement the function that calculates the least square solution with the help of numpy:

```
def least_square(x, y):
    A = np.vstack([x, np.ones(len(x))]).transpose()
    k, b = np.linalg.lstsq(A, y, rcond=None)[0]
    return k, b
```

2. RANSAC

For the following parameter values iter = 300, thres_dist = 1, n_samples = 500, n_outliers = 50 and num_subset = 5 we implement the RANSAC algorithm with the following steps:

- randomly choose a small subset, with num_subset elements, from the noisy point set, by using *random.sample*.
- compute the least-squares solution for this subset, using the function we implemented above.
- compute the number of inliers and the mask denotes the indices of inliers. To see if a point is an inlier, we simply check if the distance $|y - (kx + b)|$ is smaller than a threshold value.
- lastly, if the number of inliers is larger than the current best result, we simply update the parameters and the inlier mask.

3. Result

The following image shows the final result of linear regression and RANSAC, for ground truth values $k = 1$ and $b = 10$:

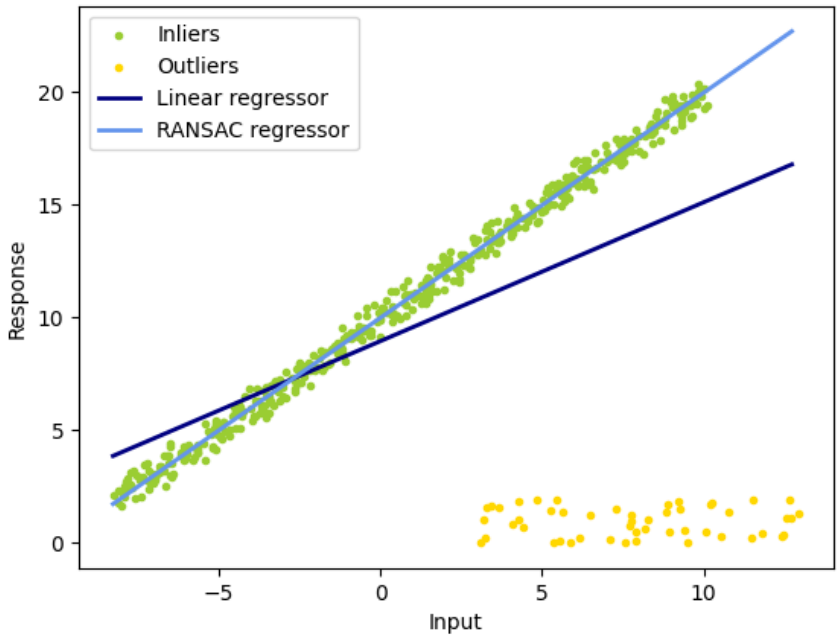

Figure 3: Model Fitting Results

k_{gt}	b_{gt}	k_{lr}	b_{lr}	k_{RANSAC}	b_{RANSAC}
1	10	0.6159656578755459	8.96172714144364	0.9987449792570882	9.997009758791009

Table 1: Comparison Results between Linear Regressor and RANSAC.

As it is apparent, RANSAC succeeds much more than Linear Regressor in the task of line fitting with outliers.