Structure from Motion & Model Fitting

Orion Junkins

December 21, 2023

1 Introduction

This is the final lab for the 2023 Computer Vision course at ETH. Section 2 below describes a Structure from Motion pipeline. Section 3 describes a basic model-fitting approach using RANSAC.

2 Structure From Motion

This section presents a basic Python implementation for a Structure From Motion Pipeline. The core pipeline skeleton is provided, and the following features are implemented.

2.1 Essential Matrix Estimation

Given the camera intrinsic matrix, k, and a set of images with identified key points and matches, the Essential Matrix, E, is calculated for specified images. First, the key points for both images are lifted to the normalized image plane. For each point, x, the normalized image plane point, $\hat{x}$, is calculated as follows:

$$\hat{x} = K^{-1}x$$

In Python, this requires inverting the provided K, converting all key points to Homogeneous coordinates, and multiplying the inverted K by each coordinate.

A constraint matrix is assembled with one row per provided match. Each image's corresponding normalized image plane key points are decomposed into u_1, v_1 and u_2, v_2. For match i, the constraint matrix row is constructed as follows:

$$\text{constraint\_matrix}[i] = [u_1u_2,\ u_1v_2,\ u_1,\ v_1u_2,\ v_1v_2,\ v_1,\ u_2,\ v_2,\ 1] \tag{1}$$

SVD solves for the null space of this constraint Matrix yielding a $U\Sigma V^T$ decomposition. The Essential matrix is estimated as the reshaped last row of the V^T Matrix yielding $\hat{E}$.

$$\text{SVD}(\textit{constraint\_matrix}) = U, S, V^T \tag{2}$$

$$\hat{E} = V^T[-1, :].\textit{reshape}(3, 3) \tag{3}$$

A refined estimation, E, is found by applying SVD again to $\hat{E}$ and manually setting the singular values.

$$\text{SVD}(\hat{E}) = U, S, V^T \tag{4}$$

$$\text{E} = U \cdot \text{diag}[1, 1, 0] \cdot V^T \tag{5}$$

This enforces the constraints that must be true for a valid essential matrix but may not be for the initial $\hat{E}$ estimation.

2.2 Point Triangulation

The implementation of DLT for point triangulation is provided. However, an additional filtering step is implemented to ensure the validity of generated points. All generated 3D points are made homogeneous. These homogeneous points are transformed into camera space for each camera. While in camera space, the z coordinate gives the points' depth into the scene relative to the camera location. A negative z value implies that the point is behind the camera and thus is not visible. Any point not visible in either or both cameras is excluded, and the resulting filtered list of valid coordinates is returned.

2.3 Finding the Correct Decomposition

Functionality for decomposing E into a relative pose is provided. The location of the first camera is the origin of the world coordinate system, and this decomposition defines valid poses for the second camera. However, a unique pose cannot be determined, and the provided function yields four possible candidates. Point triangulation is run to select the best candidate using each. The total number of valid points is counted, and the 'correct' pose is selected according to which yields the highest number of valid points.

For camera 1, the pose rotation is set as the identity matrix, and the pose translation is set as a 0 vector. For camera 2, the pose rotation and translation are set as the inverse of the identified best relative pose. This inversion is necessary because the pose is assumed to be the transformation from global space to image space, but the decomposition initially calculates it in the opposite direction.

2.4 Absolute pose estimation

To incorporate data from additional images, the location of each new camera must be found. This functionality is largely provided with the exception of the initial normalization. Normalization is performed similarly to the normalization in the initial estimation of E. The inverse of K is multiplied by the homogeneous form of all provided 2D points. The resulting coordinates are normalized to have a 1 in the last position using the provided *HNormalize()* function.

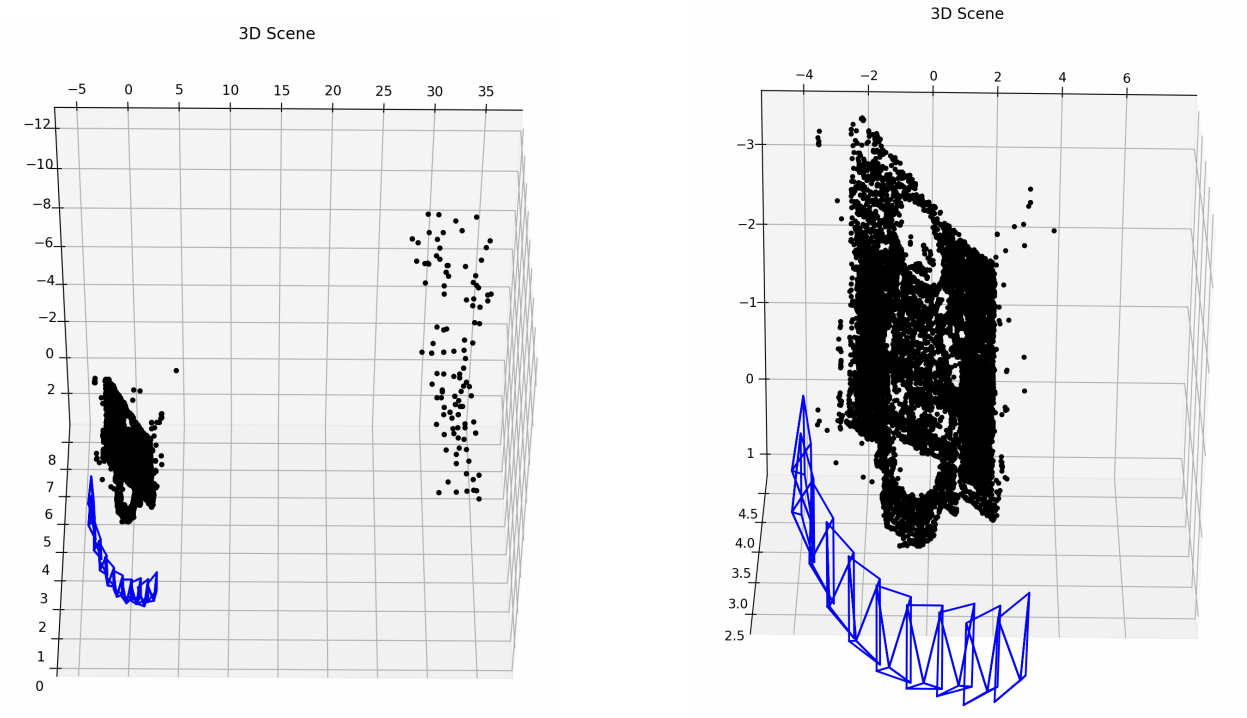
2.5 Map extension

Now, new images can be integrated to extend the point cloud further. Images are iteratively 'registered' until all images are incorporated. To register an image, points are triangulated between it and every previously registered image using the same procedure applied to the first two images. Any newly triangulated points that have not been seen before are returned, along with the correspondences for the images where those points are visible. These newly identified points are added to the overall reconstruction, and all images are updated with the new correspondences.

2.6 Result

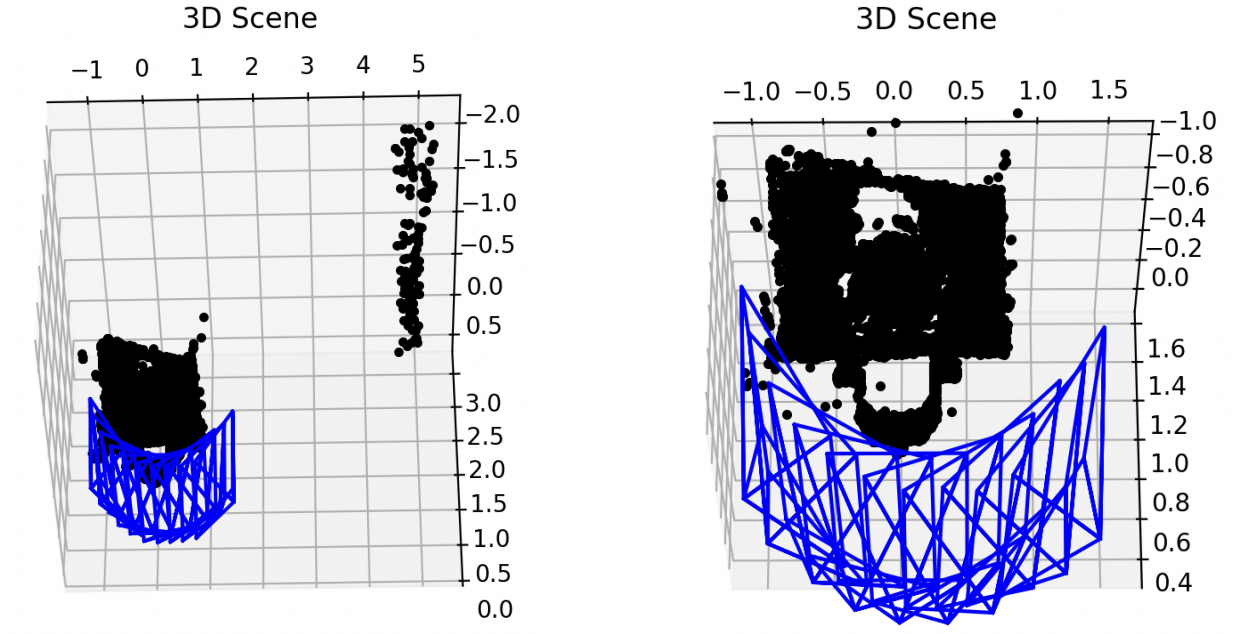
This approach is generally effective for rough reconstructions. However, there is a relatively high degree of noise, especially for points far away or only seen by a limited subset of the cameras. This can be seen clearly when using the default initialization from images 3 and 4 shown in figure 1. The reconstruction of the fountain shows some warping, noise, and outliers. But, the result is generally reasonable. Noise is much more extreme for the more distant points, such as those corresponding to the second building to the viewer's right of the main scene. While not ideal, this is to be expected; these points are only seen by cameras 8 and 9. These camera poses are relatively close, with a small baseline distance relative to the distance of the points. It is expected that the error in triangulation will be high under these conditions.

It is additionally worth noting that the choice of starting cameras heavily influences the resulting reconstruction quality. Certain camera combinations yield much worse results than the default selection of 3 and 4, while others offer substantial improvements. One pair, which shows improved reconstruction quality, 5 and 2, is shown in figure 2.



(a) Overall result (b) Zoomed result

Figure 1: Baseline result using the default initialization of images 3 and 4



(a) Overall result (b) Zoomed result

Figure 2: Improved result using the selected initialization of images 5 and 2

3 Model Fitting

Given a randomly generated data set, RANSAC fits a line to the data. Most of the data set is generated by adding noise to a linear function, but outliers are also included.

3.1 Least-squares Solution

A simple helper utility, *least_square(x,y)* is defined. It wraps the numpy *linalg.lstsq()* function, returning the least squares k, b for a line that fits the provided x and y vectors.

3.2 RANSAC

Rather than naively fitting all data, RANSAC is used to avoid overfitting to outliers. The following steps are performed for a fixed number of iterations according to the provided *iter* parameter. Each iteration performs the following steps.

3.2.1 Sampling

A random subset of size *n_samples* is drawn from the data. The *random.sample()* function generates a list of indices, and these indices are used to sample from x and y.

3.2.2 Least Square Calculation

For the current x, y subsets, the least squares solution is calculated with the *least_square()* helper. This yields a k and b value for the current subset.

3.2.3 Identify Inliers

All data points are compared to the identified line defined by the k and b values for the current subset. Those with a distance from the fit line less than the provided *thres_dist* threshold are regarded as inliers. The number of inliers and a mask specifying which indices contain the inlying elements are found.

3.2.4 Update Best

If the current number of inliers is the best found so far, the current k, b, number of inliers, and inlier mask are saved. The final result of the fitting is the best found across all iterations.

3.3 Results

Results are listed in Table 1. The performance of RANSAC is superior, with k and b values significantly closer to the Ground Truth values. This is more intuitively shown in Figure 3. The outliers heavily influence the Linear Regression line, while the RANSAC Regression effectively ignores the outliers, optimizing fit for the inliers. This shows that RANSAC can offer substantially better results than Linear Regression when modeling data sets with many outliers.

	Ground Truth	**Linear Regression**	**RANSAC**
k	1	0.616	0.964
b	10	8.962	9.983

Table 1: Values for Variables k and b

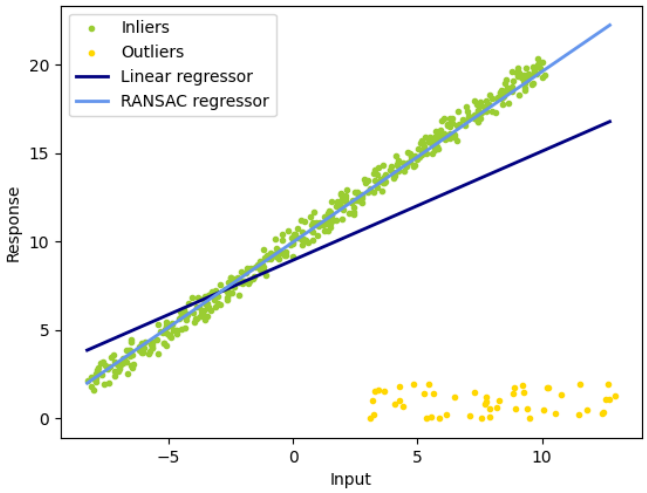


Figure 3: Visualization of data with Linear Regression and RANSAC regression lines