CMSC 733 Project 3: Report
Buildings Built in minuites - An SfM Approach

Abhishek Kathpal
M. Eng Robotics
University of Maryland
College Park, Maryland 20740
Email: akathpal@umd.edu

Darshan Shah
M. Eng Robotics
University of Maryland
College Park, Maryland 20740
Email: dshah003@umd.edu

Mayank Pathak
M. Eng Robotics
University of Maryland
College Park, Maryland 20740
Email: pathak10@umd.edu

I. Introduction

This project focuses on estimating a three dimensional structure from two-dimensional image sequences which are related to each other by change in camera motion (Orientation and translation). This problem is usually referred to as Structure from motion. There are several algorithms which achieve this. In this project, we aim to learn how to recreate 3D structures from given dataset of 2D images using traditional approaches to Sfm.
The following is the brief outline of the pipeline used in our project.

- Feature Matching, RANSAC based Outlier Rejection and Estimation of Fundamental Matrix.
- Estimate Essential Matrix based on Epipolar geometry.
- Estimate and refine Camera Pose
- Perspective-n-point Estimation
- Build visibility Matrix
- Bundle Adjustment

The structure of this project is such that each of these individual functionality are implemented in a different modules and the program flow is defines by the `wrapper.py` module.

II. Dataset

The following Figure 1 shows a set of 6 images of a building in front of Levine Hall at UPenn, taken using a GoPro Hero 3 with fisheye lens distortion corrected.These are the set of 2D images used for this project.

Fig. 1: Input Dataset

III. The Fundamental Matrix

A. Establishing Correspondences

First step towards SfM using any traditional method would be feature detection and establishing corresponding points between images. This process is usually accomplished using any feature detection and matching algorithms such as SIFT, SURF etc. For this project we used the given text files `matching<n>.txt` where n represents integer from (1,6).

The module `findCorrespondance.py` parses information from these text files and makes the data available for the rest of the program in appropriate formatting.

The figure 2 shows the output of the corresponding matching points. It can be evidently seen that there are a lot of Outliers falsely matched. To deal with this, we make use of the Random Sampling Consensus (RANSAC).
The RANSAC algorithm helps getting rid of outliers and at the same time also computes a better estimate of the fundamental matrix. This is done by iteratively, randomly selecting 8 point correspondences and computing fundamental matrix using those points. The points are rejected if the Fundamental matrix obtained is inaccurate beyond a certain threshold. This iterative process goes on either until a set number of cycles or until a satisfactory estimate is obtained.

Figure 3 shows the rejected outliers in red and the inliers in green. The number of inliers filtered between images are as follows:
Between image 1 and 2 Inliers found :631/1426

Fig. 2: Matching Point correspondences between image 1 and image 2

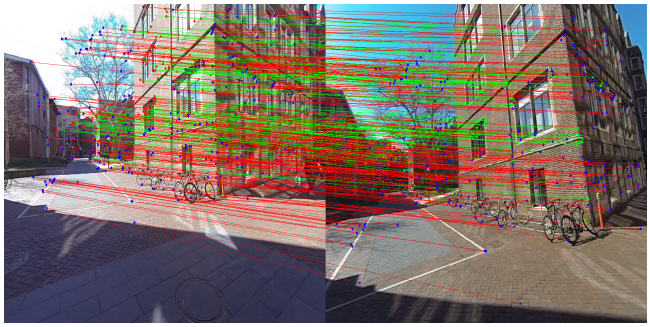

Fig. 3: Matching Point correspondences between image 1 and image 2. The red line represents the outliers and the green line represents inlers

Between image 1 and 3 Inliers found :300/608
Between image 1 and 4 Inliers found :205/469
Between image 1 and 5 Less than 8 inliers found
Between image 1 and 6 Less than 8 inliers found
Between image 2 and 3 Inliers found :907/2053
Between image 2 and 4 Inliers found :317/1075
Between image 2 and 5 Less than 8 inliers found
Between image 2 and 6 Less than 8 inliers found
Between image 3 and 4 Inliers found :679/2296
Between image 3 and 5 Inliers found :526/967
Between image 3 and 6 Inliers found :313/455
Between image 4 and 5 Inliers found :962/2166
Between image 4 and 6 Inliers found :661/1159
Between image 5 and 6 Inliers found :704/1971

The Fundamental Matrix obtained between image 1 and image 2 is given as follows:

$$F = \begin{bmatrix} \text{-7.452563e-07} & \text{-1.241800e-05} & \text{2.882683e-03} \\ \text{1.576234e-05} & \text{-1.030764e-06} & \text{-4.676318e-03} \\ \text{-4.583037e-03} & \text{1.984426e-03} & \text{1.000000e+00} \end{bmatrix}$$

B. Estimating Essential Matrix

The Fundamental Matrix is an algebraic representation of epipolar geometry and a more general form of Matrix. For a calibrated camera, we need to take the intrinsic and extrinsic parameters into consideration. The essential matrix E relates

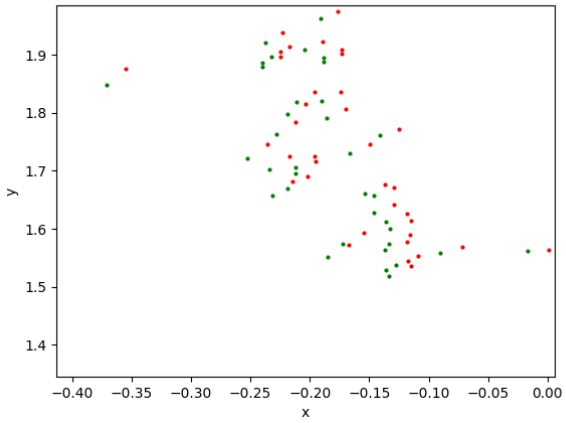

Fig. 4: Output of Linear vs Non-Linear triangulation between image 1 and 2

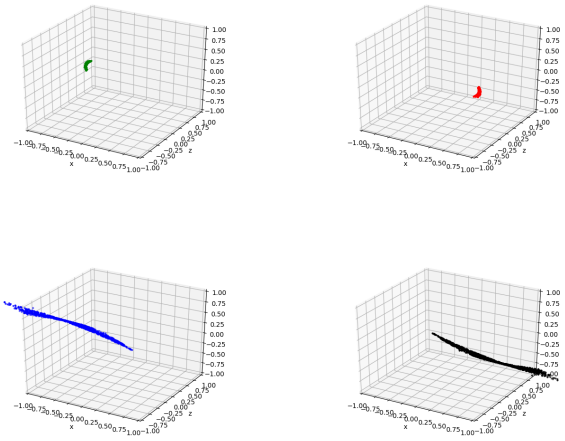

Fig. 5: 3D points generated between image pairs 1 and 2, by all 4 poses in R_set, C_set

corresponding image points between both cameras, given the rotation and translation.

$E = K^T F K$

where K is the camera Calibration matrix. The essential Matrix is extracted from F and K in the module `EssentialMatrixFromFundamentalMatrix.py`. The Essential Matrix E obtained by this method between image 1 and image 2 are as follows. The matrix E, just like Fundamental Matrix, is computed for every image pairs.

$$E = \begin{bmatrix} \text{-0.0362137} & \text{-0.68151242} & \text{-0.33796022} \\ \text{0.85017267} & \text{-0.05870702} & \text{0.46962839} \\ \text{0.23786413} & \text{-0.62720035} & \text{-0.152518} \end{bmatrix}$$

C. Estimating Camera Pose

The camera pose is defined by Rotation (Roll, Pitch, Yaw) and translation (x, y, z) of the camera with respect to the world coordinate system. Since we already computed the Essential

Matrix E, we can compute 4 camera pose configurations (C_i, R_i) where $i = 1, 2, 3, 4$. The camera pose is given by $P = KR[I_{3x3} - C]$.
The computed camera poses are corrected so as the $det(R) = 1$ always holds true. The `ExtractCameraPose.py` module computes the camera poses R and C given the Essential matrix.

The output of `EstimateCameraPose()` for images 1 and 2 is as follows:

Camera Centers C_{set} :

$C_1 =$ -0.76056042 0.0255314 0.64876497
$C_2 =$ 0.76056042 -0.0255314 -0.64876497
$C_3 =$ -0.76056042 0.0255314 0.64876497
$C_4 =$ 0.76056042 -0.0255314 -0.64876497

Rotation Matrices R_{set} :

$$R_1 = \begin{bmatrix} 0.99638539 & 0.06585649 & -0.05365701 \\ -0.05876457 & 0.9904841 & 0.12445066 \\ 0.0613423 & -0.12084768 & 0.99077392 \end{bmatrix}$$

$$R_2 = \begin{bmatrix} 0.99638539 & 0.06585649 & -0.05365701 \\ -0.05876457 & 0.9904841 & 0.12445066 \\ 0.0613423 & -0.12084768 & 0.99077392 \end{bmatrix}$$

$$R_4 = \begin{bmatrix} 0.09808373 & 0.09112491 & -0.99099739 \\ 0.02202412 & -0.99575384 & -0.08938245 \\ -0.99493443 & -0.01305889 & -0.09967419 \end{bmatrix}$$

$$R_4 = \begin{bmatrix} 0.09808373 & 0.09112491 & -0.99099739 \\ 0.02202412 & -0.99575384 & -0.08938245 \\ -0.99493443 & -0.01305889 & -0.09967419 \end{bmatrix}$$

Translation:

0.99638539 0.06585649 -0.05365701
-0.05876457 0.9904841 0.12445066
0.0613423 -0.12084768 0.99077392
0.76056042 -0.0255314 -0.64876497

The figure 5 shows the set of 3D points generated by each of the poses from R_set, C_set

D. Triangulation

1) Check for Cheirality Condition: In previous section, we got 4 possible camera poses using essential matrix. in order to find correct unique pose, we need to remove dis-ambiguity. We accomplished this by checking cheirality condition. The Cheirality Condition is that the reconstructed points must be in front of the cameras. We check this condition, the 3D points are triangulated using linear least square to check the sign of the depth Z in the camera coordinate system w.r.t. camera center. The best camera configuration, (C, R, X) is the one that produces the maximum number of points satisfying the cheirality condition. This module is implemented `DisambiguateCameraPose.py`

2) Linear Triangulation: The 3D world coordinate points are triangulated from the 2D image points using information from the point correspondences and Camera poses $(C_1, R_1$ and (C_2, R_2) using the method of linear least squares. Figure 4 shows the Triangulated points between images 1 and 2.

3) Non-linear Triangulation: We obtained two camera poses and linearly triangulated points. In this section we've refined the locations of the 3D points to minimize the reprojection error. To minimize this, we've made use of the `optimize.leastsq` function provided by `scipy` package. This function is implemented in the `NonlinearTriangulation.py` module.

The output comparison of Non-linear triangulation and linear triangulation is shown in figure 6.

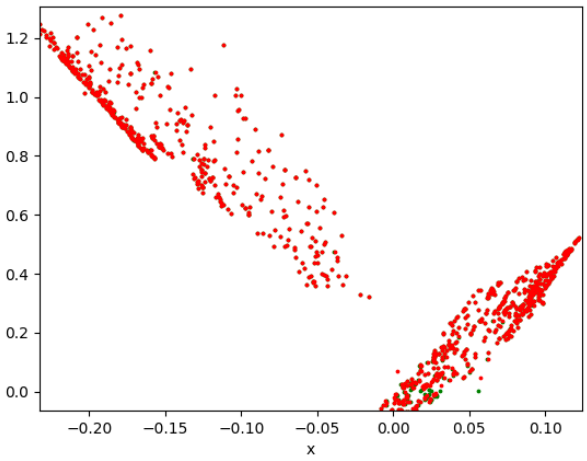

Fig. 6: Output of Non-Linear triangulation(image 1 and 2)

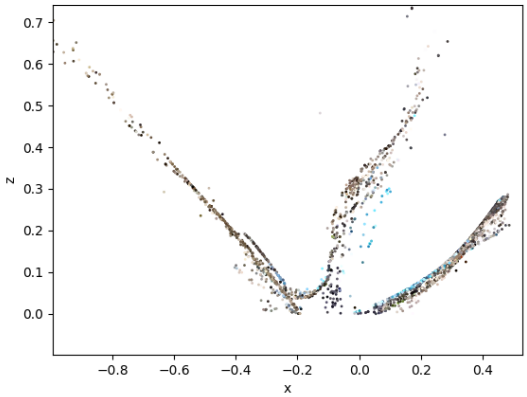

Fig. 7: Output of Non-Linear triangulation (Combined)

IV. Perspective-n-Projections

A. Linear PnP

Perspective-n-Point is the problem of estimating the pose of a calibrated camera given a set of n 3D points in the world and their corresponding 2D projections in the image. The camera pose consists of 6 degrees-of-freedom which are made up of the rotation (roll, pitch, and yaw) and 3D translation of the camera with respect to the world. When $n > 3$ the PnP problem is in its minimal form and can be solved with three point correspondences. However, with just three point correspondences, P3P yields many solutions, so a fourth correspondence is used in practice to remove ambiguity. [7]

This part of the program is implemented in the module `LinearPnP.py`.

B. PnP RANSAC

The PnP is prone to error if there are outliers in the set of point correspondences. Thus, RANSAC is used in conjunction with existing solutions to make the final solution for the camera pose more robust to outliers. This technique is very much similar to the RANSAC we implemented previously to find the Fundamental matrix. This function is implemented in `PnPRANSAC.py` and it iteratively calls the `LinearPnP` function to compute PnP for random correspondence points.

C. Nonlinear PnP

In this section we make use of 3D-2D correspondences, X $\leftrightarrow$ x and the linearly estimated pose, (C,R) and further refine the camera pose in order to minimize re-projection error. We use the output of the estimate from `LinearPnP` and minimize the cost function. This is implemented in the `NonlinearPnP.py`.

V. Bundle Adjustment

Now that we finally have the set of refined 3D points from various perspective and camera poses, we are one step away from having the 3D reconstructed output of the scene. Bundle Adjustment is defined as the problem of simultaneously refining the 3D coordinates describing the scene geometry, the parameters of the relative motion, and the optical characteristics of the camera(s) employed to acquire the images, according to an optimality criterion involving the corresponding image projections of all points.

A. Visibility Matrix

We created a visibility matrix which is a boolean matrix of the size Nxi where N is the total number of inliers and i = 6 is the number of images. The $[N,i]^th$ value of the matrix is flagged 1 if the N^th feature is visible in the i^th image. Else, it is flagged Zero.

B. Bundle Adjustment

As described earlier, in this step we refine camera poses and 3D points simultaneously. We do this by minimizing the reprojection error over $C^I_{i_{i=1}}, q^I_{i_{i=1}} and X^J_{j_{j=1}}$. Bundle Adjustment is implemented using the Large-scale bundle adjustment in scipy [2]. This is implemented in the `BundleAdjustment.py` module. The results after the Bundle adjustment stage are discussed and shown in the next section.

VI. Results

A. Our Results

The results as seen in Fig. 8 through Fig. 12 are the results obtained from our algorithm discussed in this report thus far.

B. Visual SfM

The outputs from figures 13 through 15 are the results obtained from Visual SfM [5] [6] [4] [3]

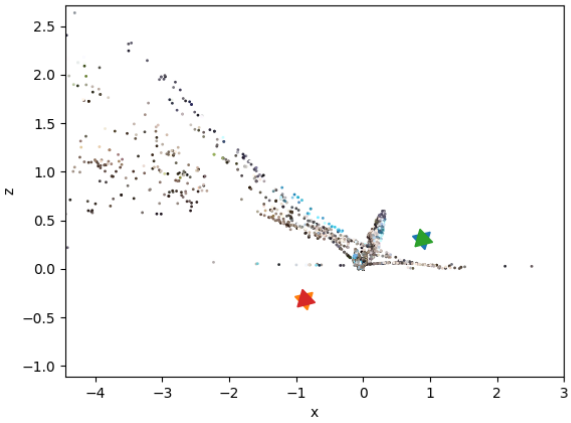

Fig. 8: Intermediate PnP output. (one of the views)

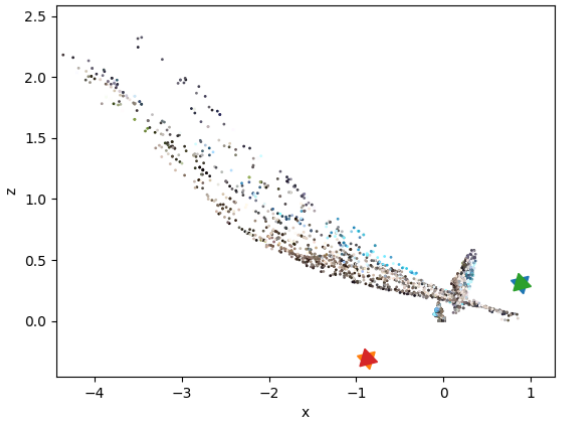

Fig. 9: Final PnP output. (one of the views). The Triangle mark indicates the camera poses.

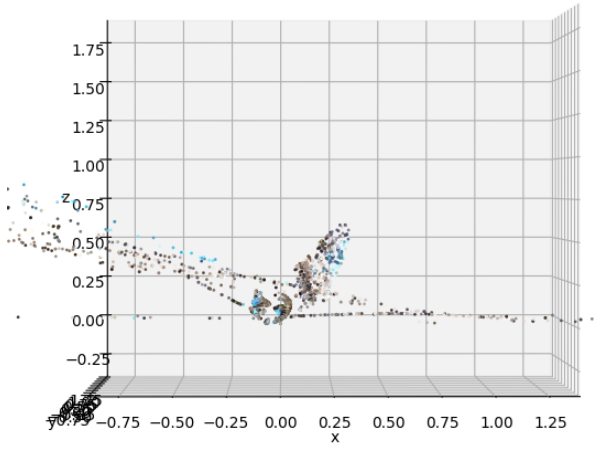

Fig. 10: Final output after Bundle Adjustment. (one of the views)

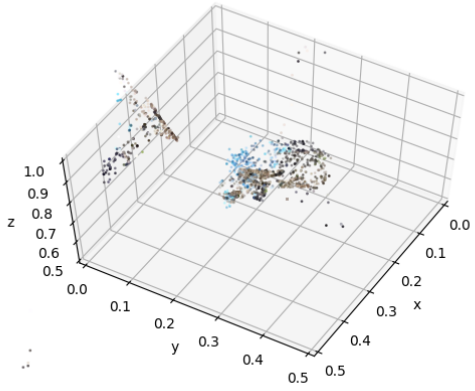

Fig. 11: Final output after Bundle Adjustment. 2D view

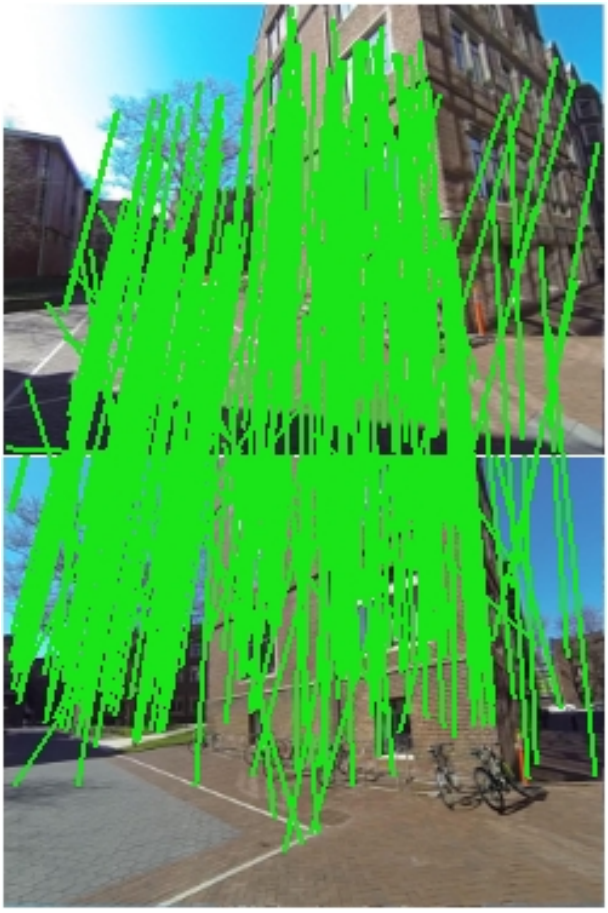

Fig. 12: Feature Matching using V-SfM

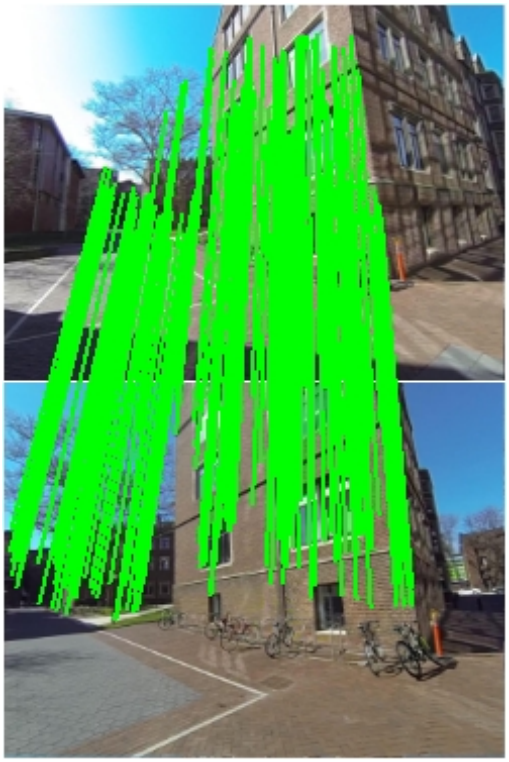

Fig. 13: Feature inliers obtained using V-SfM

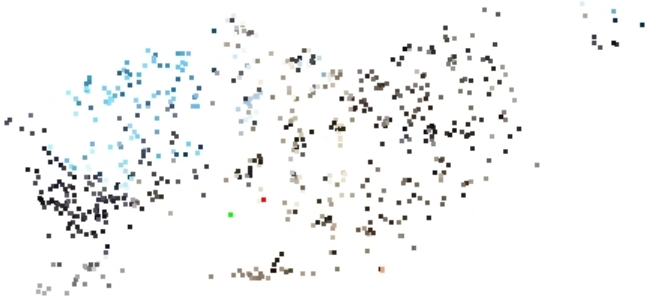

Fig. 14: 3D point plots obtained using V-SfM

Fig. 15: Final output obtained using V-SfM

VII. Extra Credits

As a part of the Extra credit problem statement, we followed the following workflow:

- Data Collection
- Camera Calibration
- Feature Matching
- SfM Wrapper

1) Data Collection: The images used for custom dataset are taken near McKeldin Library, University of Maryland. We have clicked 6 images for this implementation. These images are as shown in fig. 17

2) Camera Calibration: We used the camera calibration program implemented in our previous homework to obtain camera calibration matrix K. We clicked images of checkerboard as shown in fig. 16

3) Feature Matching: The Unique keypoints are obtained using SIFT algorithm [1]. We implemented SIFT using the `cv2.xfeatures2d` module. and further rejected as many outliers as possible using the `GetInliersRANSAC.py` module. Figure 18 shows the output after filtering Inliers between images 1 and 2.

A. Computing SfM

The rest of the algorithm to compute SfM is implemented using the same program and modules used to compute the earlier dataset. The file `wrapper.py` consists of the main function that controls the program flow. The final results are as shown in fig. 19

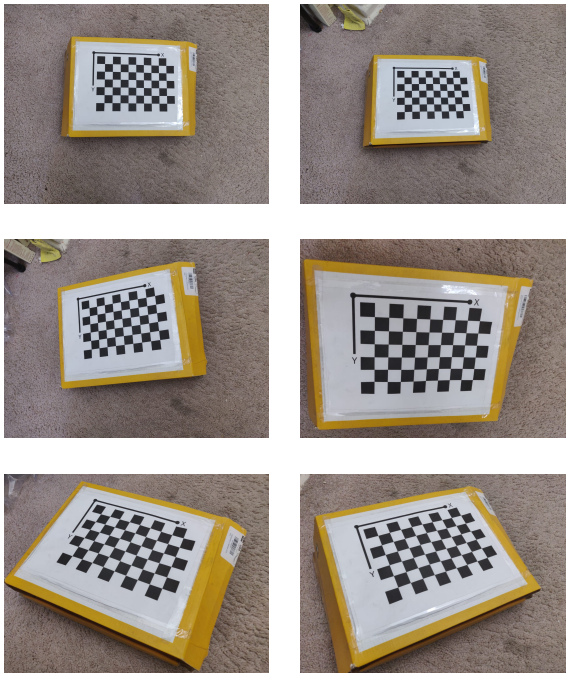

Fig. 16: Camera Calibration Checkerboard (displaying 6 out of 12 images taken)

Fig. 17: Custom Input Dataset

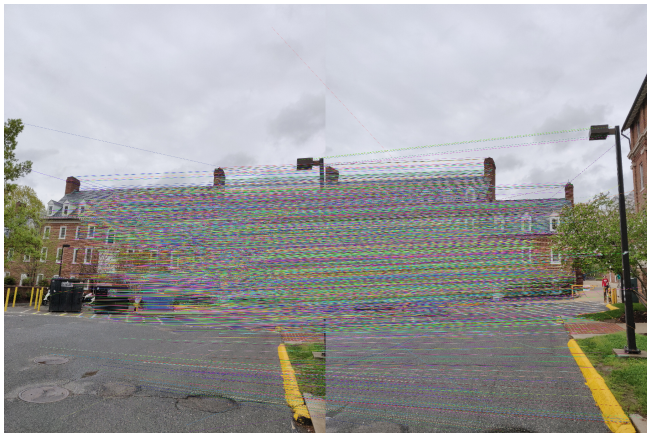

Fig. 18: Filtered Inliers between images 1 and 2 of the custom dataset

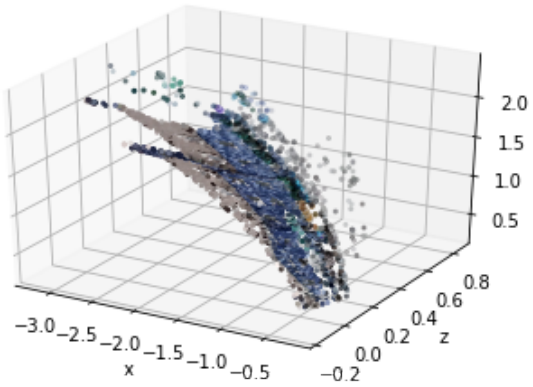

Fig. 19: Final output of the custom dataset from images 1 and 2 (Zoomed In)

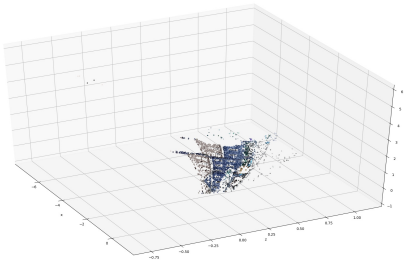

Fig. 20: Final output of the custom dataset from images 1 and 2 (Zoomed Out)

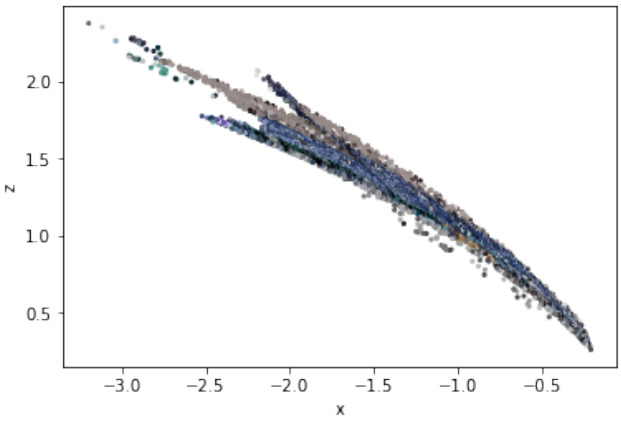

Fig. 21: Final output in 2D of the custom dataset from images 1 and 2 (Zoomed In)

References

[1] David G. Lowe. "Distinctive Image Features from Scale-Invariant Keypoints". In: *International Journal of Computer Vision* 60.2 (Nov. 2004), pp. 91–110. ISSN: 1573-1405. DOI: 10.1023/B:VISI.0000029664.99615.94.

[2] Nikolay Mayorov. *Large-scale bundle adjustment in scipy*. https://scipy-cookbook.readthedocs.io/items/bundle_adjustment.html. Oct. 2016.

[3] C. Wu. "Towards Linear-Time Incremental Structure from Motion". In: *2013 International Conference on 3D Vision - 3DV 2013*. June 2013, pp. 127–134. DOI: 10.1109/3DV.2013.25.

[4] Changchang Wu. "SiftGPU: A GPU Implementation of Scale Invariant Feature Transform SIFT". In: (Jan. 2013).

[5] Changchang Wu. *VisualSFM : A Visual Structure from Motion System*. http://ccwu.me/vsfm/.

[6] C. Wu et al. "Multicore bundle adjustment". In: *CVPR 2011*. June 2011, pp. 3057–3064. DOI: 10.1109/CVPR.2011.5995552.

[7] Jianliang Tang Xiao-Shan Gao Xiao-Rong Hou and Hang-Fei Cheng. "Complete solution classification for the perspective-three-point problem". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 25.8 (Aug. 2003), pp. 930–943. ISSN: 0162-8828. DOI: 10.1109/TPAMI.2003.1217599.