Hochschule Ravensburg-Weingarten

Computer Vision Project Report

3D Point Could Generation using Structure from Motion

HOCHSCHULE
RAVENSBURG-WEINGARTEN
UNIVERSITY
OF APPLIED SCIENCES

Created by:

Sahil Arora
(Matriculation Number: 32953)

July 3, 2020

Contents

1 Introduction

The scope of this project is to reconstruct the scene using a pair of images captured by a smartphone camera. This process of re-generating scene from multiple 2D images is called 3D reconstruction which is helpful in application areas like Robotic Mapping, Medicine and more [Wik]. The concept of Structure from Motion is used here in which we capture images from a two or more angles around the scene. The same point in space is viewed from different perspectives by different cameras and the collective information of transformations from one view to other is determined by estimating camera parameters. This involves finding features in a pair of images, registering those set of matching features, then finding Essential Matrix. Using concept of Triangulation, un-scaled depth of a point relative to a camera scene is computed.

The process requires a few tools and libraries which need to be set up. This whole project is developed on Ubuntu 18.4 platform on a Intel i7 personal computer with dedicated 4GB NVIDIA GPU and 16GB RAM. Source code is written in Python 3.6 with support libraries like NumPy and MatPlotLib [Num] [M D]. Additionally, OpenCV library is set up which provides access to various ready-to-use functions from the computer vision community [Opea]. For evaluation of the algorithm with the trusted software results, we use a famously-known tool VisualSFM to generate sparse cloud with same pair of images and then use CloudCompare to evaluate our results against those from VisualSFM [Wu] [Clo].

2 Camera Intrinsic Parameters and Undistorting Images

2.1 Camera Matrix influences 3D to 2D projection

As soon as we decide which camera is being used for capturing we would want to calibrate the camera and find Camera Matrix (K). K matrix is a 3 × 3 matrix which converts the 3D co-ordinates in camera system (x_1, x_2, x_3) to 2D points in pixel co-ordinate system of the said camera (z_1, z_2).

$$\begin{pmatrix} \tilde{z_1} \\ \tilde{z_2} \\ \tilde{z_3} \end{pmatrix} = K \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} ; further \begin{pmatrix} z_1 \\ z_2 \\ 1 \end{pmatrix} = \frac{1}{\tilde{z_3}} \begin{pmatrix} \tilde{z_1} \\ \tilde{z_2} \\ \tilde{z_3} \end{pmatrix} \tag{1}$$

where $\tilde{z_1}, \tilde{z_2}, \tilde{z_3}$ are called helper co-ordinates in camera system.

Matrix K is defined as:

$$\begin{pmatrix} fk_{p1} & 0 & s_1 \\ 0 & fk_{p2} & s_2 \\ 0 & 0 & 0 \end{pmatrix} \tag{2}$$

where f is focal length of the lens and k_{p1}, k_{p2} are parameters of pixels per unit mm s_1 and s_2 are difference between Optical Center and Image Center (Sec. 2.1.5 of [Sze10]).

2.2 Finding Camera Matrix by Checkerboard

We estimate the camera matrix based on library functions in OpenCV [Opec]. A checker-board as illustrated in Fig 2.1 is printed on a piece of paper. With the camera we capture few images of this checker-board from different perspectives. In 3D camera co-ordinate system, we denote the co-ordinates as (x_1, x_2, x_3). Since, the checker-board was pasted on a flat wall we can assume that $x_3 = 0$. Size of the squares will remain constant through out so we can denote:

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = c \begin{pmatrix} m_1 \\ m_2 \end{pmatrix} \tag{3}$$

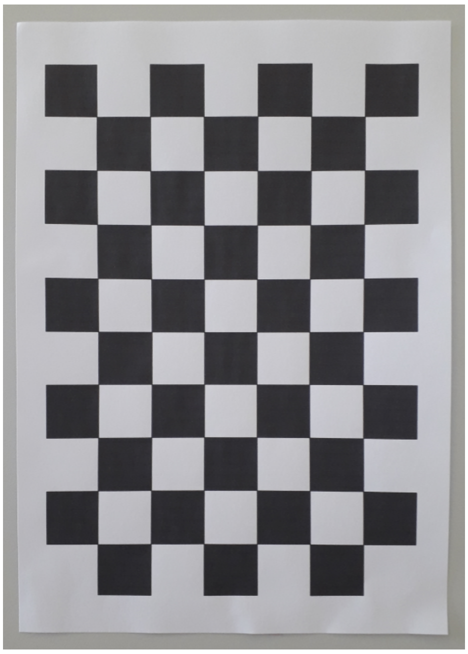
where m_1, m_2 are Natural numbers and c is a constant factor. The library function uses above approach to find the corner points in images through corner detection algorithm and Camera Matrix is then estimated by solving the below equation for K:

$$\begin{pmatrix} z_1 \\ z_2 \\ 1 \end{pmatrix} = P \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ 0 \\ 1 \end{pmatrix} = K(R, \overrightarrow{t}) \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ 0 \\ 1 \end{pmatrix} \tag{4}$$

We already have denoted x_1 and x_2. Using corner detection z_1 and z_2 are found but in this equation (R, $\overrightarrow{t}$) is not known. Mathematically, we denote the product of K × (R, $\overrightarrow{t}$) as H matrix (called Homography) and then the function extracts matrix K based on Cholesky Factorization of linear equations formed by H matrix and above mentioned equation [Zha00]. Fig 2.2 shows image from source code where corners points are detected and drawn for visualization (lines have been enhanced for better visualization).

Re-Projection Error: It is better to use images having lot of different perspectives to get better approximation of K. In our setup we have used 16 images and it took the program 12 seconds to run and detect all images. The output of K is generated as a numpy array and is stored in local directory for usage. As a checkpoint, we try to re-project the world co-ordinates mathematically with the calculated matrix K and compare it with the point co-ordinates detected as corners [Opec]. This calculation gave a re-projection error of 26 % error in our image set.

Undistorting Images: Apart from Camera Matrix (K), another parameter is unknown, namely the Lens-Distortion coefficients. Ideally, straight lines in real world should also appear straight in images but this is hardly true. Lenses produce effect like Pincushion Distortion, Barrel Distortion etc (Sec 2.1.6 of [Sze10]) which has to be undone to carry out further procession with minimal errors. The function L that distorts the point (z_1, z_2) is depending upon distance (r) of this point from optical center of camera. Function L can be



(a) Fig 2.1 Checkerboard pattern 9×6

(b) Fig 2.2 lines showing detected corners

expanded with Taylor Series and written as :

$$L(r) = 1 + k_1 r + k_2 r^2 + k_3 r^3 + k_4 r^4 + ... \tag{5}$$

where $k_1, k_2, k_3...$ are distortion coefficients. Checker-board function again uses sub-pixel accuracy to improve the position of all corners on a straight line and gives the output as an array of Distortion Coefficients which is again stored as a numpy array. The set of images used and source code can be found on github [Git]. To undistort an image we simply use the previously estimated Camera Matrix and Distortion Coefficients to call the function. The function transforms the given image to rectify the distortions caused by lens [Opec].

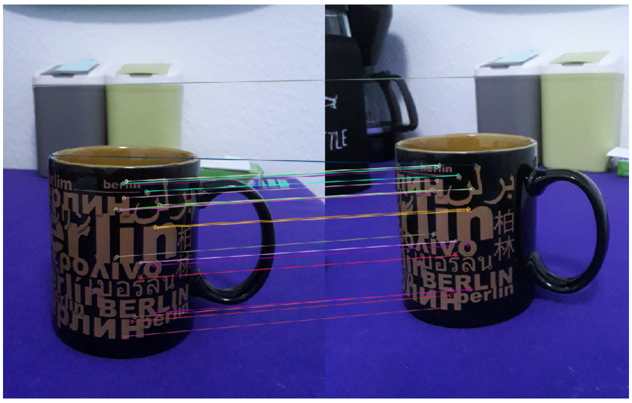
3 Feature Matching in pair of Images

3.1 What is Feature ?

The first step in reconstructing a scene would be to find Fundamental Matrix (F). More explanation about Fundamental Matrix will follow in next section. But, to find the Fundamental matrix we have to find pairs of points that are visible in both the images. Feature points are defined by pixel areas in a image in the neighborhood of which the sum of squared difference (SSD) in x- direction and y-direction change rapidly. SSD is defined as equation below [ER20b]:

$$f_H(u,v) = \sum_{(x_k,y_k)\in N(x,y)} (I(x_k,y_k) - I(x_k+u, y_k+v))^2 \tag{6}$$

(a) Fig 3.1 features found on Mug



(b) Fig 3.2 lines showing matched features

3.2 Finding Features

There exist multiple techniques for Feature Description, for this project we will use ORB (Oriented FAST and BRIEF) [Rub+11].This is a combination of feature finder technique FAST (Features from Accelerated Segment Test) which classifies a point as corner if $\boldsymbol{n}$ out of 16 neighbouring pixels are brighter or darker than the pixel under consideration (default $\boldsymbol{n}$ is taken as 12 and naturally a threshold t_h is defined to classify any pixel as brighter or darker). Further BRIEF (Binary robust independent elementary features) is a feature descriptor which defines a feature as a vector which can later be compared to other vectors using Euclidean Distance / Hamming Distance to decide if it is the same point or not [Cal+10]. We use OpenCV library function to find the feature points in both our images [Opef]. Fig 3.1 shows an illustration of found features in one of the images.

Now we need to match features in two corresponding images. Since, we are using a powerful computer processor we aren't really much concerned about time. Therefore, we choose Brute Force matcher as the first choice. It simply compares every single feature description A from first image to every single feature descriptor B in other image and chooses the one with lowest Hamming Distance [ER20b]. This technique is easy to implement and promises good results. We are using OpenCV library function to match feature points from one image to other [Opeb]. Till now, we have a set of points whose location is known in the corresponding images. Since there can also be outlier points in feature matching which can really mess up further computations, we use Ratio Test with threshold = 0.8 as explained by D. Lowe in his paper [Low04]. Fig 3.2 shows two images with straight lines connecting matching feature points.

4 Fundamental Matrix

4.1 How is Fundamental Matrix helpful ?

By definition, it is a matrix (denoted here by F) which constraints the matching feature points in two images. It can be used to calculate any Epipolar line in 2nd image for any given point x_0 in first image. A slight explanation about Epipolar lines : Given for every feature point in image 1, we can reduce our search for a matching point in the second image. There is no need to look through each pixel in the second image. Rather we can just search along the Epipolar Line to find the match (Sec 9.2 of [HZ03]). We know that 2D image co-ordinates can be reconstructed into 3D world co-ordinates p = $(\lambda_1, \lambda_2, \lambda_3)^T$ by below equation [ER20c]:

$$\begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \end{pmatrix} = cR^T K^{-1} \begin{pmatrix} z_1 \\ z_2 \\ 1 \end{pmatrix} - R^t \overrightarrow{t} \tag{7}$$

where **c** is a scalar. This equation (8) is expression of a 3D line. This means for any arbitrary 2D point, we will get multiple re-projected points in three dimension. See fig 4.1 (Sec 11.1 of [Sze10]) as an illustration of this. However, these multiple points when projected onto the 2nd Image plane by pre-multiplying with Camera Matrix P_1, we get again a series of 2D points in image 2 that lie on a straight line. This line is termed as Epipolar line. We can use the fundamental matrix F to denote any Epipolar line corresponding to a point x_0.

$$\big(L_1 = Fx_0\big) \tag{8}$$

since x_1 lies on L_1 , we can say $x_1^T L_1 = 0 \implies \mathrm{x}_1^T F x_0 = 0$.

4.2 Compute Fundamental Matrix

To calculate the Fundamental Matrix we need to have 9 pair of matching points, so that we can set up 9 linear equations and solve for F matrix. We have used our matched points (shown in fig 3.2) and used the OpenCV library function to compute Fundamental Matrix [Opee]. Since there can be outlier points, we have used RANSAC filter approach with 99% confidence interval. Our results shows that the rank of F matrix is 2, which confirm that it is not a full rank matrix and has 1 singular value equal to 0 (Sec 9.2.4 of [HZ03]).

5 Essential Matrix

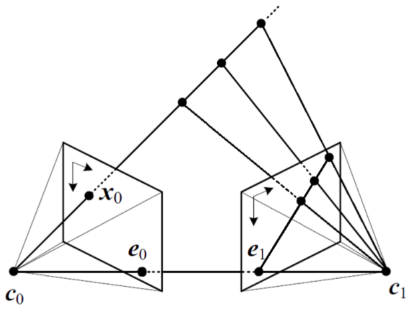
5.1 Essential Matrix: Calibrated form of Fundamental Matrix

Essential matrix satisfies the same constraint of Epipolar line as the Fundamental matrix did, but it does so on normalized image co-ordinates. We denote E as essential matrix, then:

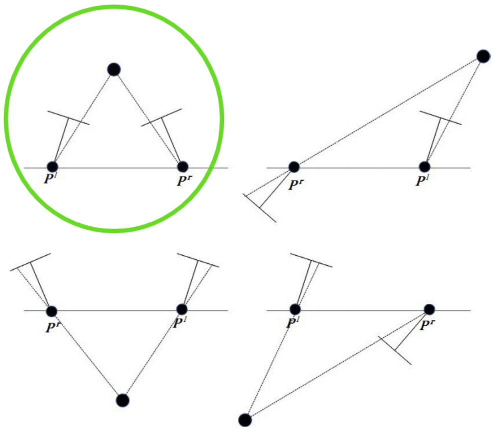
$$\big(\hat{x}_1^T \cdot E \cdot \hat{x}_0 = 0\big) \tag{9}$$

where $\hat{x}_0 = K^{-1} x_0$

So to find the Essential Matrix we use the same approach as we did in Fundamental Matrix. We use the library function (with RANSAC filter approach to avoid outliers) to estimate the value of E [Oped]. Our results shows that the rank of E matrix is 2, which confirm that its properties are similar to that of Fundamental Matrix (Sec 9.6.1 of [HZ03])



(a) Fig 4.1 3D re-projection line of 2D point



(b) Fig 5.1: 4 Camera positions wrt (R, $\overrightarrow{t}$)

5.2 Compute Rotational and Translation Matrices

This essential matrix is quite useful for us to find the Camera Extrinsic. E can be written as cross product of $\overrightarrow{t}$ and R. We can safely denote :

$$\big(E = [t] \times R\big) \tag{10}$$

Now the whole point is to extract $\overrightarrow{t}$ and R from known E matrix so that we can have the Camera Matrix Fully defined. We use the Singular Value Decomposition to get two orthogonal matrices (U and V) and a diagonal matrix (Σ) : $E = [t] \times R = U\Sigma V^t$ (Sec 9.6.2 of [HZ03]). To find out T and R matrix we define:

$$W = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{11}$$

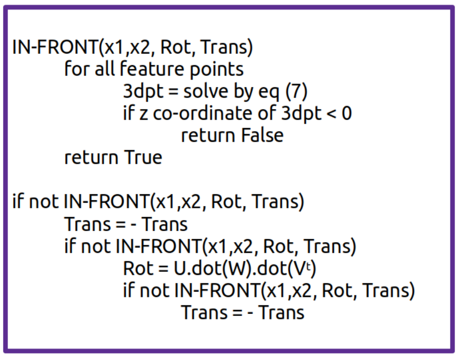
The T and R matrix as calculated as per: $T = UW\Sigma U^T; R = UW^{-1}V^t$ where T is the matrix representation of cross product.

$$T = \begin{pmatrix} 0 & -t_3 & t_2 \\ t_3 & 0 & -t_1 \\ -t_2 & t_1 & 0 \end{pmatrix} \tag{12}$$

if we know T we can generate our $\overrightarrow{t}$ vec using above notation [ER20a].

$$\begin{pmatrix} t_1 = su_{21}u_{32} - su_{22}u_{31} \\ t_2 = su_{11}u_{32} - su_{12}u_{31} \\ t_3 = su_{11}u_{22} - su_{12}u_{11} \end{pmatrix} \tag{13}$$

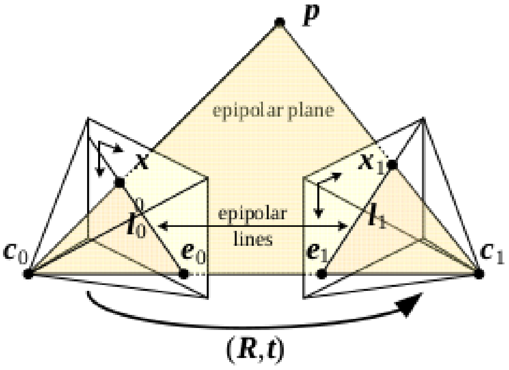
But the obtained $\overrightarrow{t}$ vec and R matrix are not unique solutions. The right hand side of Epipolar equation (9) is zero and thus any scalar multiplied to E would also work. This will directly affect the translation vector and any scalar multiple of $\overrightarrow{t}$ would then also be a valid solution. There can be 4 possible solution if we work with normalized $\overrightarrow{t}$ vectors. With these 4 possible sets of R and $\overrightarrow{t}$ we can calculate 4 different 3D positions of a point but all of them will not make sense physically. See fig 5.1 illustrating 4 different camera positions. In only 1 of the case the 3D point will lie in front of both the cameras (Sec 9.6.3 of [HZ03]).



```
IN-FRONT(x1,x2, Rot, Trans)
        for all feature points
                3dpt = solve by eq (7)
                if z co-ordinate of 3dpt < 0
                        return False
        return True

if not IN-FRONT(x1,x2, Rot, Trans)
        Trans = - Trans
        if not IN-FRONT(x1,x2, Rot, Trans)
                Rot = U.dot(W).dot(V^t)
                if not IN-FRONT(x1,x2, Rot, Trans)
                        Trans = - Trans
```

(a) Fig 5.2 Pseudo Code to check physical sense of camera



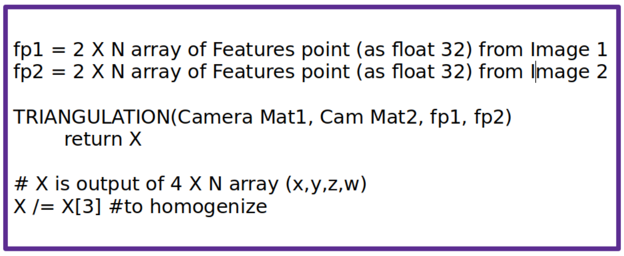
(b) Fig 6.1 Epipolar Geometry, pg348 [Sze10]

We do it by hit and trial method to check which combination of R and $\vec{t}$ is suitable. We take all the feature matched points in Section 3 and use the full Camera matrix to re-project them in 3D using equation (7). Then we ensure that **z** co-ordinate of both these points shall be positive. If one of the points come out to negative we change the Rotation or Translation Matrix and recalculate the points. Fig 5.2 is a pseudo code for explanation. Now we have our Camera Intrinsic and Camera Extrinsic we can re-project all our feature points to 3D co-ordinates using **Triangulation**.

6 Triangulation

6.1 Finding 3D co-ordinates by intersection of 2 rays

We have previously seen with equation (7) that re-projection doesn't give us a unique point but a 3D line consisting infinite possibilities. We know the position of same point in image co-ordinates of 2nd camera and we could also get a 3D line on which the said point can lie. We do not have information about world co-ordinate system but we can choose Camera center C_0 as world position [ER20a]. So, first camera matrix $\tilde{P}_0 = \tilde{K}_0 \tilde{R}_0$ where

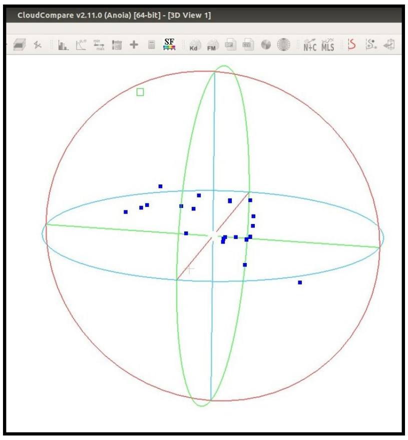


```
fp1 = 2 X N array of Features point (as float 32) from Image 1
fp2 = 2 X N array of Features point (as float 32) from Image 2

TRIANGULATION(Camera Mat1, Cam Mat2, fp1, fp2)
        return X

# X is output of 4 X N array (x,y,z,w)
X /= X[3] #to homogenize
```

(a) Fig 6.2: Psuedo Code Triangulation



(b) Fig 7.1: Isometric View of generated point cloud

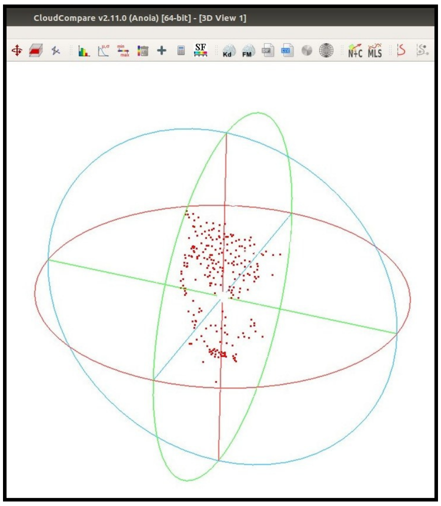
$$\tilde{K_0} = \begin{pmatrix} K_0 & 0 \\ \vec{0}^T & 1 \end{pmatrix} \textbf{ and } \tilde{R_0} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{14}$$

We have already computed P_1 from Essential Matrix decomposition. Ideally, these two 3D lines should intersect and we can find the exact location of a point in world co-ordinate, but this is hardly the case. There are multiple techniques available but we use OpenCV library function that uses Least-Square method [Opeg]. It computes the error between two depth-less rays and minimizes the error. This may not give an exact solution. Refer to Fig 6.1 for illustration of rays and their intersection. Fig 6.2 is a pseudo code for explanation of used function.

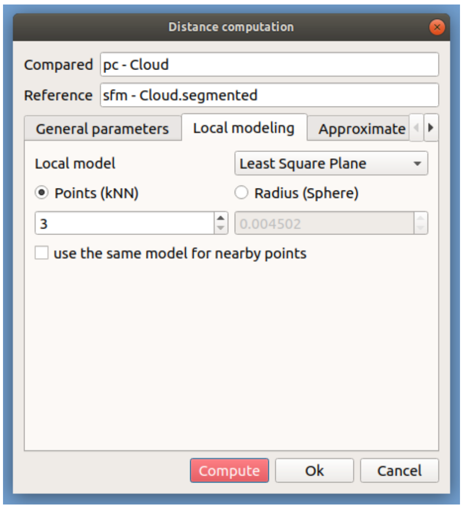
7 Result and Evaluation

The output of OpenCV function as illustrated in pseudo code (Fig 6.2) is an numpy array of type $\vec{f} = (x, y, z, w)$. This is converted into homogeneous co-ordinates of the type $\tilde{f} = (\tilde{x}, \tilde{y}, \tilde{z}, 1)$ by dividing vector by its 4th element. This array is then stored as a text file using python numpy so that it can be directly imported in Cloud Compare. Fig 7.1 shows the screenshot of point cloud obtained from Triangulation. Notice that only matching feature points are reconstructed. We had a very few matching pair (23 in number) and it is one of the reasons that the output figure looks almost unlike a cup.

Fig 7.2 shows the point cloud obtained from Visual SFM. Using VisualSFM is quite easy and guidelines can be found on their homepage. This software uses a robust technique of finding features: SIFT. It could also find features varying in scale and rotation [Wu]. The results of Visual SFM are more promising than our results. To set a quantitative metric we are using CloudCompare to evaluate our results with that of VisualSFM [N+18].



(a) Fig 7.2: VisualSFM Results



Compared: pc - Cloud

Reference: sfm - Cloud.segmented

Local model: Least Square Plane

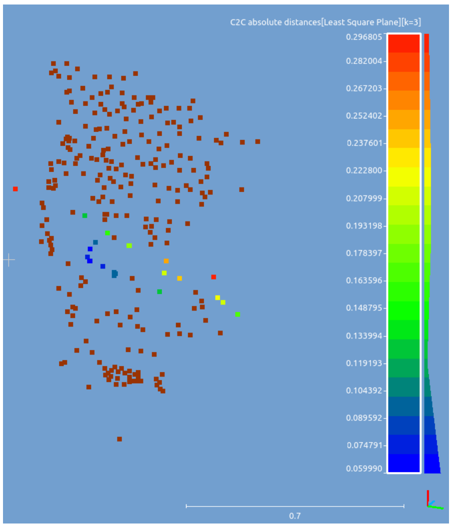
● Points (kNN) ○ Radius (Sphere)

3 0.004502

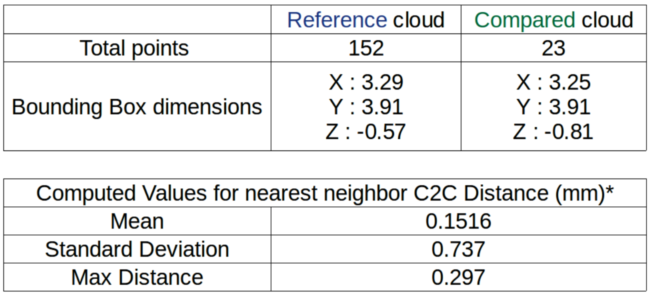
□ use the same model for nearby points

(b) Fig 7.3: Comparison parameters

Comparing Results: We are computing the distance between the two clouds. We set the VisualSFM result as reference and set our result as 'compared cloud'. The algorithm of



(a) Fig 7.4: Nearest Distance colormap



	Reference cloud	Compared cloud
Total points	152	23
Bounding Box dimensions	X : 3.29 Y : 3.91 Z : -0.57	X : 3.25 Y : 3.91 Z : -0.81

Computed Values for nearest neighbor C2C Distance (mm)*	
Mean	0.1516
Standard Deviation	0.737
Max Distance	0.297

(b) Table 7.A: Statistical results of comparison

software finds the nearest point of the reference cloud for each point in compared cloud and computes their Eucledian distance [Com]. We have used the k-Nearest Neighbour approach with k = 3 to average out the effect of far distant outlier points. Fig 7.3 shows the parameters used to compare the two clouds. For visualization, a color-chart is shown in Fig 7.4. Brown colored dots are from VisualSFM result and points from our results are color coded based on their closest distances. CloudCompare provides also provides the statistical results which are listed in Table 7.A.

8 Future Work

We have used only 2 images to generate the point cloud and very few feature points are detected in these images. As a result, what we get is a scattered Point Cloud. Firstly, we can capture more pictures from different perspectives. This will develop a relation between different views of scene and will give it a holistic sense. Recommendation by Carnegie Mellon University, USA are a good explanation of how to efficiently capture the scene [Uni]. Secondly, to find more corresponding feature points we can modify source code to use a much simpler technique like Harris Corner and constrain searching on Epipolar line. We could also follow another method for regeneration of point cloud. Since we have already computed the Camera Matrices we can go for Rectification of the images. One of the images is adjusted and remapped in such sense that the pair has same orientation but with a baseline translation. This is very similar to Stereo geometry and thus pixel-wise depth can be computed and stored as disparity map. Each source code for this project is uploaded to Github for reference [Git].

References : Books

[HZ03] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision, Second Edition*. Cambridge University Press, 2003.

[Sze10] R. Szeliski. *Computer Vision: Algorithms and Applications, 1st ed.* Springers, 2010.

Articles

[Zha00] Z. Zhang. “A flexible new technique for camera calibration”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 22.11 (2000), pp. 1330–1334.

[Low04] D.G Lowe. “Distinctive Image Features from Scale-Invariant Keypoints”. In: *International Journal of Computer Vision 60, 91–110 (2004).* (2004).

[Cal+10] Michael Calonder et al. “BRIEF: Binary Robust Independent Elementary Features”. In: *Computer Vision – ECCV 2010* (2010), pp. 778–792.

[Rub+11] E. Rublee et al. “ORB: An efficient alternative to SIFT or SURF”. In: *2011 International Conference on Computer Vision* (2011), pp. 2564–2571.

[N+18] Ahmad Fuad N. et al. “Comparing the Performance of Point Cloud Registration Methods For Landslide Monitoring Using Mobile Laser Scanning Data”. In: *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Volume XLII-4/W9* (2018).

Lecture Notes

[ER20a] S. Elser and Hochschule Ravensburg Weingarten. *Lecture notes in Computer Vision, Chapter 10: Structure From Motion and 3d Reconstruction*. June 2020.

[ER20b] S. Elser and Hochschule Ravensburg Weingarten. *Lecture notes in Computer Vision, Chapter 3:feature Detection, Description and Matching*. May 2020.

[ER20c] S. Elser and Hochschule Ravensburg Weingarten. *Lecture notes in Computer Vision, Chapter 7: 3d to 2d Projections, 2d to 3d Projections and 3d Rotations.* June 2020.

Webpages

[Clo] CloudCompare. *CloudCompare*. URL: `https://www.danielgm.net/cc/`. (accessed: 13.06.2020).

[Com] Cloud Compare. *Cloud-to-Cloud Distance*. URL: `http://www.cloudcompare.org/doc/wiki/index.php?title=Cloud-to-Cloud_Distance`. (accessed: 22.06.2020).

[Git] GitHub. */sa32953*. URL: `https://github.com/sa32953/3d_reconstruction`. (accessed: 09.06.2020).

[M D] et al M. Droettboom. *MatplotLib*. URL: https://matplotlib.org/. (accessed: 10.06.2020).

[Num] Numpy. *NumPy*. URL: https://numpy.org/. (accessed: 01.07.2020).

[Opea] OpenCV. *OpenCV*. URL: https://opencv.org/. (accessed: 01.07.2020).

[Opeb] Brute Force OpenCV. *Brute Force*. URL: https://docs.opencv.org/3.4/dc/dc3/tutorial_py_matcher.html. (accessed: 17.05.2020).

[Opec] Camera Calibration : OpenCV. *Camera Calibration*. URL: https://docs.opencv.org/3.4/dc/dbb/tutorial_py_calibration.html. (accessed: 05.06.2020).

[Oped] Essential Matrix OpenCV. *Essential Matrix*. URL: https://docs.opencv.org/3.4/d9/d0c/group__calib3d.html#ga13f7e34de8fa516a686a56af1196247f. (accessed: 23.06.2020).

[Opee] Fundamental Matrix OpenCV. *Fundamental Matrix*. URL: https://docs.opencv.org/3.4/da/de9/tutorial_py_epipolar_geometry.html. (accessed: 20.05.2020).

[Opef] ORB OpenCV. *ORB OpenCV*. URL: https://docs.opencv.org/3.4/d1/d89/tutorial_py_orb.html. (accessed: 04.05.2020).

[Opeg] Triangulation OpenCV. *Triangulation*. URL: https://docs.opencv.org/3.4/d0/dbd/group__triangulation.html. (accessed: 25.06.2020).

[Uni] Carnegie Mellon University. *Instruction on Reconstruction Workflow*. URL: https://www.cs.cmu.edu/~reconstruction/basic_workflow.html. (accessed: 28.05.2020).

[Wik] Wikipedia. *Wikipedia*. URL: https://en.wikipedia.org/wiki/3D_reconstruction#cite_note-18. (accessed: 01.07.2020).

[Wu] C. Wu. *VisualSFM: A Visual Structure from Motion System*. URL: http://ccwu.me/vsfm/. (accessed: 24.06.2020).