Hochschule Ravensburg-Weingarten

Computer Vision Project Report

3D Point Could Generation using Structure from Motion

Created by:

Sahil Arora
(Matriculation Number: 32953)

June 29, 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 (https://en.wikipedia.org/wiki/3D). 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 angles by different cameras and the collective information of transformations from one view to other is determined by finding camera extrinsic parameters. This involves finding features in a pair of images and registering those set of matching features. Further, using concept of Triangulation, 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 machine with dedicated 4GB NVIDIA GPU and 16GB RAM. Source code is written in Python 3.6 with support libraries like NumPy and MatPlotLib. Additionally, OpenCV library is set up which provides access to various function from the computer vision library. For evaluation of the algorithm with the trusted truth, we use a famously-known tool Visual SFM to generate sparse cloud and then use Cloud Compare to evaluate our results against those from Visual-SFM..

2 Camera Intrinsic Parameters and Undistorting Images

2.1 What is Camera Matrix ?

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 (x_1, x_2, x_3) to 2D points in pixel co-ordinate system of the said camera (z_1, z_2).

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

$$\begin{pmatrix} z_1 \\ z_2 \\ 1 \end{pmatrix} = \frac{1}{\tilde{z_3}} \begin{pmatrix} \tilde{z_1} \\ \tilde{z_2} \\ \tilde{z_3} \end{pmatrix} \tag{2}$$

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

Matrix K is defined as:

$$\begin{pmatrix} fk_{p1} & 0 & s_1 \\ 0 & fk_{p2} & s_2 \\ 0 & 0 & 0 \end{pmatrix} \tag{3}$$

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.

2.2 Finding Camera Matrix by Checkerboard

We estimate the camera matrix based on library functions in OpenCV. A check-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 world 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:

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = c \begin{pmatrix} m_1 \\ m_2 \end{pmatrix} \tag{4}$$

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

$$\begin{pmatrix} z_1 \\ z_2 \\ 1 \end{pmatrix} = P \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ 0 \\ 1 \end{pmatrix} = K(R, \overrightarrow{t}) \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ 0 \\ 1 \end{pmatrix} \tag{5}$$

We already have denoted x_1 and x_2. Using corner detection z_1 and z_2 are found but in this equation (R,t) is not known. We denote the product of K $\times$ (R, $\overrightarrow{t}$) as H matrix (called Homography) and then matrix K is estimated based on Cholesky Factorization of linear equations formed by H matrix and above mentioned equation. Fig 2.2 shows image from source code where corners points are detected and drawn for 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. 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 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 expanded with Taylor Series and written as :

$$L(r) = 1 + k_1 r + k_2 r^2 + k_3 r^3 + k_4 r^4 + ... \tag{6}$$

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. 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.

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 pair 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:

$$f_H(u, v) = \sum_{(x_k, y_k) \in N(x,y)} (I(x_k, y_k) - I(x_k + u, y_k + v))^2 \tag{7}$$

3.2 Finding Features

There exist multiple techniques for Feature Matching, for this project we will use ORB (Oriented FAST and BRIEF). 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 the same point or not. 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 don't really are 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. This technique is easy to implement and promises good results. But since one feature point A can be matched t various points B, we have used k-Nearest Neighbour matching with k = 2. Since there can also be outlier points in feature matching, we use Ratio Test explain by D. Lowe in his paper. Fig 3.3 shows two images with straight lines connecting matching feature points.

4 Fundamental Matrix

4.1 What is Fundamental Matrix ?

By definition, it is a matrix (denoted here by F) which 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 : We already know that 3D points $(x_1, x_2, x_3))$ can be converted into pixel co-ordinates (z_1, z_2) by equation 1 and 2.
We know that 3D world co-ordinates p = $(\lambda_1, \lambda_2, \lambda_3)^T$ can be converted into 3D camera co-ordinates by below equation.

$$\begin{pmatrix} \tilde{z_1} \\ \tilde{z_2} \\ \tilde{z_3} \end{pmatrix} = P \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ 1 \end{pmatrix} = K(R, \overrightarrow{t}) \begin{pmatrix} \lambda_1 \\ \lambda_2 \\ \lambda_3 \\ 1 \end{pmatrix} \tag{8}$$

This can be written in shorthand notation: $\overrightarrow{z}$ = P $\cdot$ $\overrightarrow{p}$
So, in other words if we know $\overrightarrow{z_0} = (z_1, z_2, 1, d)$ where d is the disparity value, we can compute any point p by below relation:

$$\left(\overrightarrow{p_0} = P_0^{-1} \cdot \overrightarrow{z_0} \right) \tag{9}$$

The mentioned P matrix is not a sqaure matrix yet but we can always augment zero vector $(0, 0, 0)^T$ to make P invertable. The major problem during reconstruction with just digital images is that the disparity is not known. So for any arbitrary value of disparity, we will get multiple points in 3 dimension. See fig 4.1 as an illustration of this. However, these points when projected onto the 2nd Image plane by pre-multiplying with Camera Matrix P_1, a line if formed for various possibilities of p_1. This line is termed as Epipolar line. We can use the fundamental matrix F to find any Epipolar line corresponding to a point x_0.

$$\left(L_1 = F x_0\right) \tag{10}$$

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

So to calculate the Fundamental Matrix we need to have at most 9 pair of matching points x0 and x1, 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 F. We have used with RANSAC filter approach to avoid outliers. 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.