Large Area 3D Reconstruction using Images, Maps and Location Data

Submitted in partial fulfillment of the requirements
for the degree of

Master of Technology

by

Srijit Dutt

Roll No: 10305056

under the guidance of

Prof. Parag Chaudhuri

Prof. J. Saketha Nath

Department of Computer Science and Engineering
Indian Institute of Technology Bombay
2012

Dissertation Approval Certificate

Department of Computer Science and Engineering

Indian Institute of Technology Bombay

The dissertation entitled **"Large Area 3D Reconstruction using images, maps and location data"**, submitted by **Srijit Dutt** (Roll No: **10305056**) is approved for the degree of **Master of Technology** in **Computer Science and Engineering** from **Indian Institute of Technology Bombay**.

Prof. Parag Chaudhuri **Dept CSE, IIT Bombay** **Supervisor**	**Prof. J. Saketha Nath** **Dept CSE, IIT Bombay** **Supervisor**
Prof. Sharat Chandran **Dept CSE, IIT Bombay** **Internal Examiner**	**Mr. Kiran Ambardekar** **Sark Infotech** **External Examiner**

Prof. K R Karunakaran
Dept ME, IIT Bombay
Chairperson

Place: IIT Bombay, Mumbai
Date: 2012

Declaration

I, Srijit Dutt, declare that this written submission represents my ideas in my own words and where others ideas or words have been included, I have adequately cited and referenced the original sources. I also declare that I have adhered to all principles of academic honesty and integrity and have not misrepresented or fabricated or falsified any idea/data/fact/source in my submission. I understand that any violation of the above will be cause for disciplinary action by the Institute and can also evoke penal action from the sources which have thus not been properly cited or from whom proper permission has not been taken when needed.

_____

Signature

_____

Name Of student

_____

Roll number

_____

Date

Abstract

With the advent and mass-penetration of smart phones which not only possess high mega-pixel cameras but also have various sensors like GPS, magnetometer, etc., it is possible now for users to take high quality photos which have geo-tagged information embedded in them in the form of *EXIF* tags. Together with the above development as well as publicly available map databases which have various places of interest outlined in them like *Wikimapia* and *Google Maps*, it is possible now to find out reliably which building or monument is being photographed by a user. We have build an online portal where various users can submit a collection of geo-tagged photographs and easily visualize three-dimensional dense point clouds reconstructed using those photographs.

We utilize an improved version of *Bundler Software* [33, 34] in the back-end. The key-point matching part of the *Bundler SFM* pipeline, which is the most computationally intensive stage of the pipeline, has been re-written to take advantage of the multi-core architecture of the current generation of CPUs giving about four to five times speed improvement over the original version. The outline of the building being viewed, which is queried from *Wikimapia*, is used to reduce the number of image matching. Also, the key-point matching is now incremental. allowing users to seamlessly add images to existing point cloud models.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

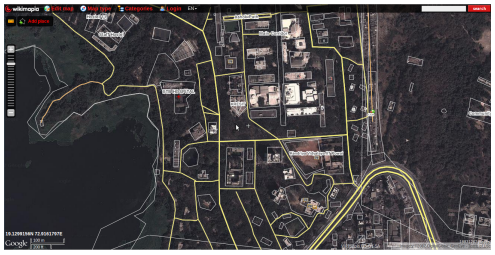
3D Reconstruction is the problem of constructing a three-dimensional model of an object from single or multiple two-dimensional views of the object under consideration. Human beings possess a remarkable ability to navigate and understand the visual world by solving the inversion problem of getting 3D information from 2D images. 3D Reconstruction is a major research area in computer vision that seeks to incorporate such abilities in computers. 3D Reconstruction has applications in diverse fields such as tourism, medical imaging, architecture, robotics, autonomous driving and exploration, photo organization, image or video retrieval, virtual reality, augmented reality, human-computer interaction, object recognition, *etc.*

In recent years, there has been a surge of interest in urban 3D modeling from large unordered collection of images [1, 10, 33]. With the high penetration of smart phones and digital cameras which come with various sensors to measure *GPS* and directional data, it is possible locate with precision the images on a map using geo-tagged data. If compass direction is available, then a viewing cone can be traced out. Together with the help of map databases, like *Wikimapia* and *Google Maps*, which have various urban buildings outlined in them, it is possible now to find out reliably which building or monument is being photographed by the user. We have built an on-line portal where various users can submit a collection of geo-tagged photographs and easily visualize dense point cloud 3D models generated by those photographs.

1.1 Problem Statement

The main aim of the project is to develop a system capable of doing large scale 3D reconstruction using sensor data from various sources like *GPS* and compass direction. We want to use the *GPS* and direction information to query the map database and find which building is being viewed and cluster the images using this information. 3D reconstruction procedure requires a large amount of images and these images should have good coverage of the building that we are trying to reconstruct. Our aim is to carry out campus wide reconstruction. As it is not possible for a single individual to be able to cover all the buildings, we have built an online portal where users can upload geo-tagged images and view the 3D model reconstructed using those images.

Images
Hampi Temple



Maps
courtesy wikimapia.org
last accessed on 20th June 2012

GPS and direction
courtesy wikipedia.com
last accessed on 20th June 2012

1.2 Contributions

In this thesis work, we have made following contributions:-

1. Online portal where users can upload images and view 3D model. The portal allows for users to track the status of the images submitted.
2. The geo-tagged images annotated with compass direction uploaded by the user are used to identify the building being viewed by the user and reconstruct the building.
3. The building outline information is used to reduce the number of image key-point matches.
4. The key-point matching stage is re-written to use the latest *FLANN Library*.
5. Multi-threading support is added to increase the speed of matching.
6. Support for incrementally adding images to an already reconstructed model.

The rest of this report is organized as follows. Chapter 2 gives an overview of single view metrology technique. Chapter 3 reviews the various algorithms used in the *Bundler* system. Chapter 4 gives brief system overview and describes the functionality of each module. Chapter 5 describes in detail the procedure to cluster images depending on the building being viewed and the key-point matching module. Chapter 6 outlines the formulation to incorporate *GPS* information into the *Bundle Adjustment* problem. In Chapter 7 we present our results and compare it with those of the pre-existing system. Chapter 8 concludes this report with a summary of key observations. We outline potential future research that can build upon the work in this thesis.

1.3 Related Work

In this section, we provide an overview of some of the state-of-the-art systems in 3D reconstruction. We start by describing a system that uses a single image as an input. We than progress towards describing systems that process a large number of images.

1.3.1 Reconstruction from a Single Image

Single View Metrology

Single View Metrology [7] uses only a single image to obtain a metric reconstruction from the image but restricts itself to scenes containing planes and parallel lines. The technique can used to get the distance between parallel planes, area ratio between parallel planes and camera position. The technique is described in detail in section 2.

Make3D

Make3D [28, 29, 30] uses only a single image to create a visually pleasing 3D model. The scene is assumed to be made up of a number of small plane patches. Discriminatively trained Markov Random Field (MRF) by supervised learning is used to infer a set of plane parameters that capture both the orientation as well as the location of a image patch. The training data consists of a large set of images and their corresponding ground-truth depth-maps. Make3D incorporates multi-scale local and global image features, The method does not generalize well to all types of images and gives good results only for image similar to those present in the training set.

Figure 1.1: Make3D Input Image (Left) and output model (Right)

Figure 1.2: Make3D output on a Hampi Temple Image

Rule based procedural modeling

Rule based procedural modeling technique [22] reconstructs facade of building with a single image. It uses shape grammars to get hierarchical subdivision of facades. It is useful for urban reconstruction using low resolution oblique aerial imagery and building facades.

1.3.2 Reconstruction from Multiple Images

Structure-and-Motion Pipeline on a Hierarchical Cluster Tree

Structure from motion refers to the process of finding the three-dimensional structure of an object by analyzing local motion signals over time. The computational complexity of the structure for motion pipeline is reduced by using hierarchical clustering [8, 12]. The images are organized into a tree with agglomerative clustering, using a measure of overlap *Geometric Robust Information Criterion (GRIC)* [36] as the distance. The reconstruction then follows this tree from the leaves to the root, with the partial reconstruction representing the internal nodes and the images representing the leaves. As a result, the problem is broken into smaller instances, which are then separately solved and combined. The scheme reduces the complexity of the problem if the tree is well balanced. Compared to the standard sequential approach, this pipeline boosts computational efficiency by one order of magnitude, is independent from the initial pair of views and copes better with drift problems.

Large Scale 3D Reconstruction

The recent development of techniques in 3D reconstruction have allowed for a large number of images (150,000 to 3 million images) [1, 10] to be processed for 3D reconstruction and large scale 3D models to be developed from community photo sharing websites. The major steps in large scale 3D reconstruction [10] are:-

1. **Appearance-based Clustering**: Using *GIST* [25] features to capture global image

similarity and *GPS* location information, iconic views are obtained and are clustered using K-Mediod algorithm [15].

2. **Geometric Cluster Verification**: *SIFT* [18] and *ARRSAC* [26] are used to identify mutually consistent core set of images. An iconic view of each cluster is also found.

3. **Local Iconic Scene Graph Reconstruction**: *Vocabulary Tree* [24] search and clustering based on geo-location is used to identify neighbouring iconics. Image registration are carried out for the iconic images.

4. **Dense Model Computation**: *Clustering Views for Multi-view Stereo (CMVS)* [11] is used for clustering the images in manageable cluster sizes. Any Multi-view stereo like *Patch-based Multi-view Stereo (PMVS)* can be used to process each cluster in parallel and then fuse the reconstructed model.

SFM with Duplicate Objects

Traditional structure for motion pipeline do not give good reconstructions when the scene contains duplicate structures or objects due to erroneous matching of point correspondences between the duplicate structure. *Expectation Maximization* (EM) based approach [27] is used to handle large duplicate objects in a scene using and estimate the camera parameters and detect erroneous point correspondence matches.

Linear SFM

Linear structure from motion technique [32] is based on vanishing point (VP) matches in images. A vanishing point is a point in a image to which parallel lines not parallel to the image plane appear to converge. Vanishing point are used to estimate global camera rotations as well as relative rotation estimates obtained from pairwise image matches. Camera translations and 3D points are simultaneously estimated using a multi-staged linear technique. Unlike sequential structure from motion the linear structure from motion method does not require intermediate bundle adjustments.

Fusion of GPS and Structure-from-Motion using Constrained Bundle Adjustments

Fusion of GPS and Structure from Motion [16] introduces two constrained Bundle Adjustment[38] formulation which enforce an upper bound for the re-projection error. The problem of drift in structure from motion pipeline is addressed by integrating the bundle adjustment procedure in an incremental structure from motion method based on local bundle adjustment.

Single View Metrology is explained in detail in chapter 2. Structure for motion pipeline stages are explained in greater detail in chapter 3. More details about all related work can be found in their respective chapters where we deal with the involved methods in greater detail.

Chapter 2

Single View Metrology

In general, a single view does not contain enough information for a complete 3D reconstruction of the scene. But with the knowledge of certain geometric information, we can achieve a metric reconstruction from the image.
In this chapter, we shall restrict ourselves to scenes containing planes and parallel lines. We assume that the vanishing line of a reference plane in the scene and the vanishing point are not parallel to the plane that is provided by the user. We can then measure the following three canonical quantities useful in single view metrology:-

1. Distance between planes parallel to the reference plane

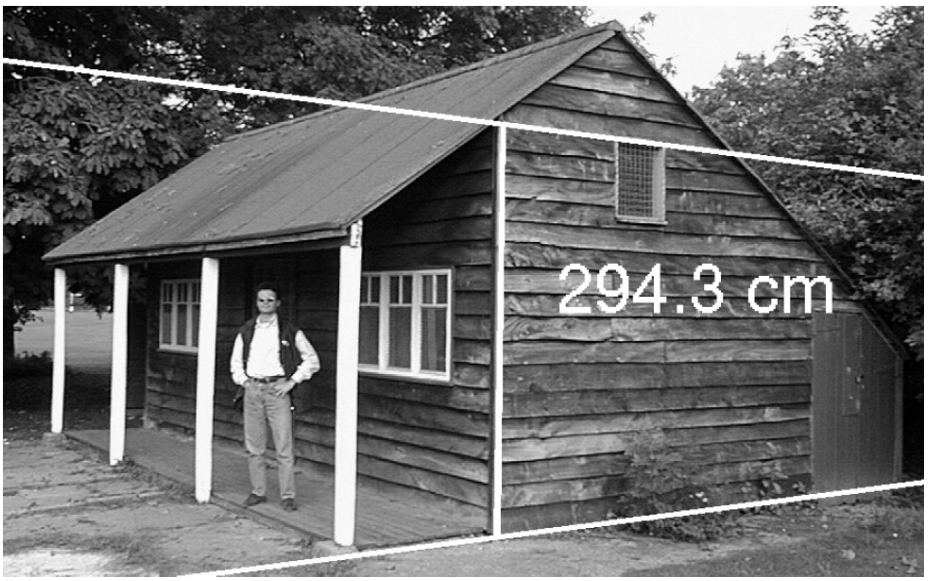


Figure 2.1: Distance between parallel planes [6]

2. Area and length ratios between planes parallel to the reference plane

Figure 2.2: Ratio of Area between parallel planes [6]

3. Camera position

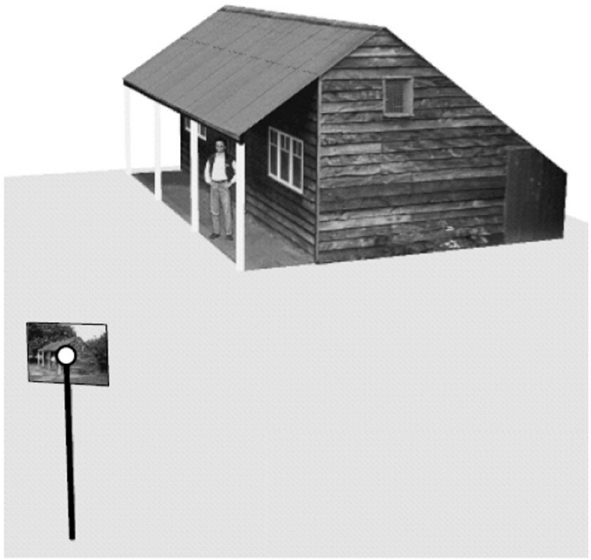


Figure 2.3: Camera Position [6]

These measurements are independent of the internal parameters of the camera.

2.1 Notation

Here we introduce the mathematical notation used throughout the report. We will be working with the homogeneous coordinate system.

1. Point (x, y) in $\mathbb{R}^2$ is represented as $(x, y, 1)$ in $\mathbb{P}^2$ projective space
2. Equality is defined only up to scale $(x, y, z) = k \cdot (x, y, z)$ for homogeneous coordinates
3. (x, y, z) in $\mathbb{P}^2 = \left(\frac{x}{z}, \frac{y}{z}\right)$ in $\mathbb{R}^2$ where $z \neq 0$
4. $(x, y, 0)$ represents points at infinity
5. Line in $\mathbb{R}^2 : ax + by + c = 0$
6. Line in $\mathbb{P}^2 : (a, b, c)$
7. $x^T l = l^T x = 0$ for all points on the line

2.2 Vanishing Points and Lines

A vanishing point is a point in a perspective drawing to which parallel lines not parallel to the image plane appear to converge.

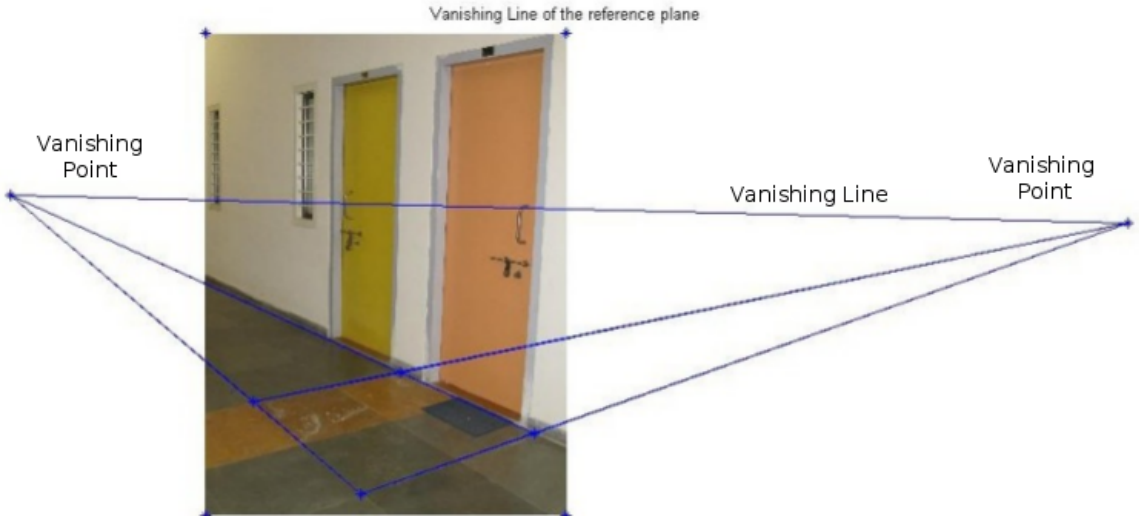


Figure 2.4: Vanishing Line and Vanishing Point [6]

The vanishing line is defined as the intersection of two vanishing points. The vanishing points and line can also be automatically detected [31].

2.3 Camera Model

We assume a pinhole camera model.

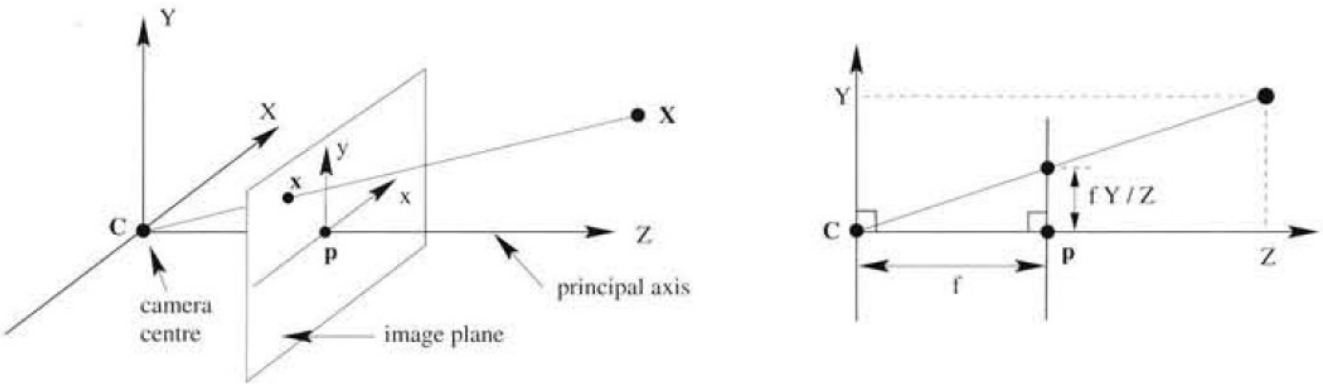


Figure 2.5: Pinhole Camera Model [13]

Pinhole Camera: C is the camera center which is placed at the origin. The line from the camera center perpendicular to the image plane is called the *principal axis* or *principal ray* of the camera and the point P where the principal axis meets the image plane is called the *principal point.* Any point on the line through camera center C and X (a point in 3D) is imaged at x (a point on the image plane).

2.3.1 Camera Projection Matrix

Camera projection matrix P is a 3×4 matrix. Some properties of projection matrix are :-

1. $\mathbf{x} = P\mathbf{X}$ where $\mathbf{x} = (x\ y\ 1)^T$ and $\mathbf{X} = (X\ Y\ Z\ 1)^T$
2. Columns of P are $p_i,\ i = 1, \ldots, 4$
3. $p_1,\ p_2, p_3$ are vanishing points of the world coordinate axes X, Y and Z
4. X-axis direction $\mathbf{D} = (1,\ 0,\ 0,\ 0)^T$ is imaged at $p_1 = PD$
5. p_4 is the image of world origin
6. $P = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \end{bmatrix} = [v_x \quad v_y \quad v \quad o]$

2.4 Planar Homography

$$x = P\mathbf{X} = \begin{bmatrix} p_1 & p_2 & p_3 & p_4 \end{bmatrix} \begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix} = \begin{bmatrix} p_1 & p_2 & p_4 \end{bmatrix} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}$$

$$x = \mathrm{H}x_\pi$$

H is the planar homography and is a 3×3 matrix of full rank where the columns of H are given by:-

$$\mathrm{H} = \begin{bmatrix} p_1 & p_2 & p_4 \end{bmatrix} = \begin{bmatrix} v_x & v_y & o \end{bmatrix}$$

p_4 must not lie on the vanishing line l. If it does, then the columns will not be linearly independent.

$$\mathrm{o} = p_4 = \frac{l}{\|l\|} = \bar{l}$$

$$P = \begin{bmatrix} v_x & v_y & \alpha v & \overline{l} \end{bmatrix}$$

where α is a scale factor
where v_x v_y v are the vanishing points in X, Y, Z direction and o is origin
The distance between the parallel planes is independent of first two columns.

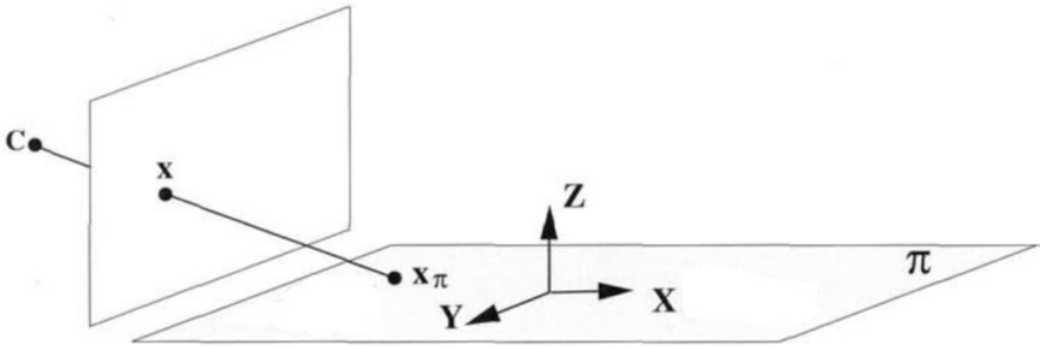


Figure 2.6: Planar Homography [13]

2.5 Camera Center

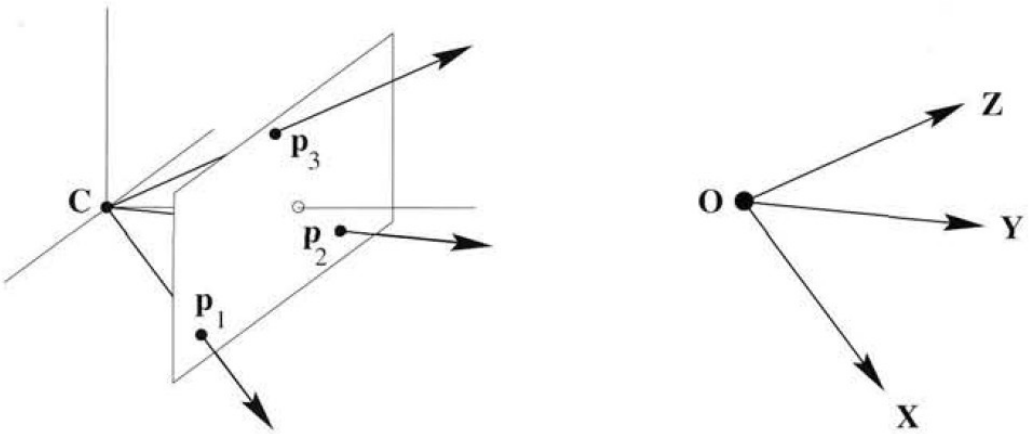


Figure 2.7: Camera center [13]

The camera center is the null-space of the camera projection matrix $PC = 0$
Line containing C and any other point A in 3-space:

$$X(\lambda) = \lambda A + (1 - \lambda)$$

Under projection

$$x = PX(\lambda) = \lambda PA + (1 - \lambda)PC = \lambda PA$$

All points on the line are mapped to same image point PA that means the line should be through the camera center.

2.6 Cross Ratio

Cross ratio is invariant under projective transformation of the line and is also independent of a particular homogeneous representation. The formula for cross ratio is given by :-

$$Cross(x_1, x_2, x_3, x_4) = \frac{|x_1 x_2|\,|x_3 x_4|}{|x_1 x_3|\,|x_2 x_4|}$$

$$|x_i x_j| = det \begin{bmatrix} x_{i1} & x_{j1} \\ x_{i2} & x_{j2} \end{bmatrix}$$

where if $x_{i2} = \ x_{j2} = 1$ then $|x_i x_j|$ is signed distance between x_i and x_j

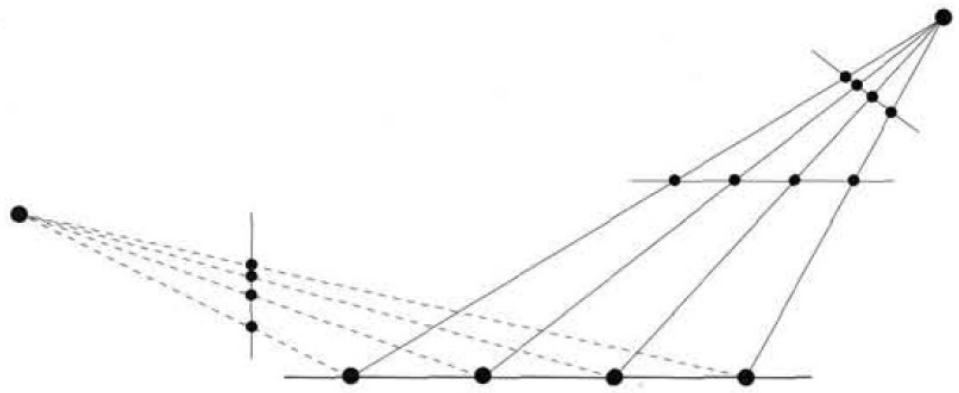


Figure 2.8: Cross Ratio [13]

2.7 Geometry

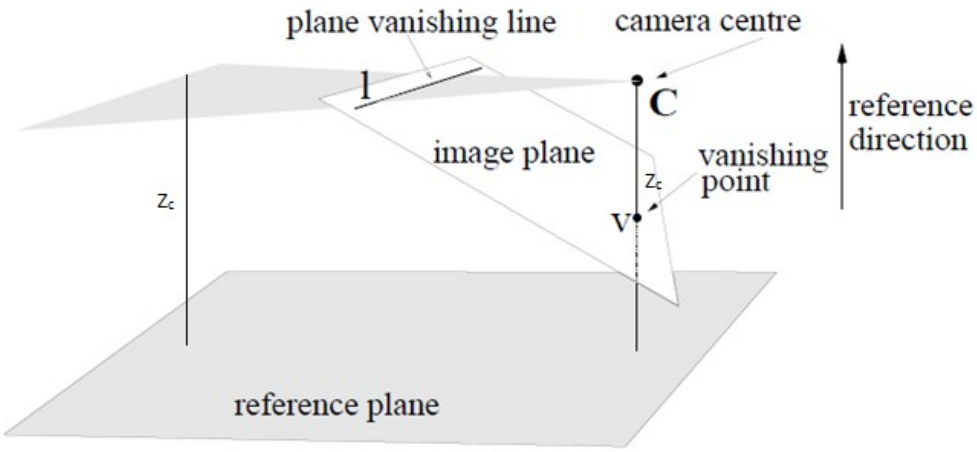


Figure 2.9: Basic Camera Geometry [7]

Any scene point which projects onto the vanishing line is at the same distance from the plane as the camera center

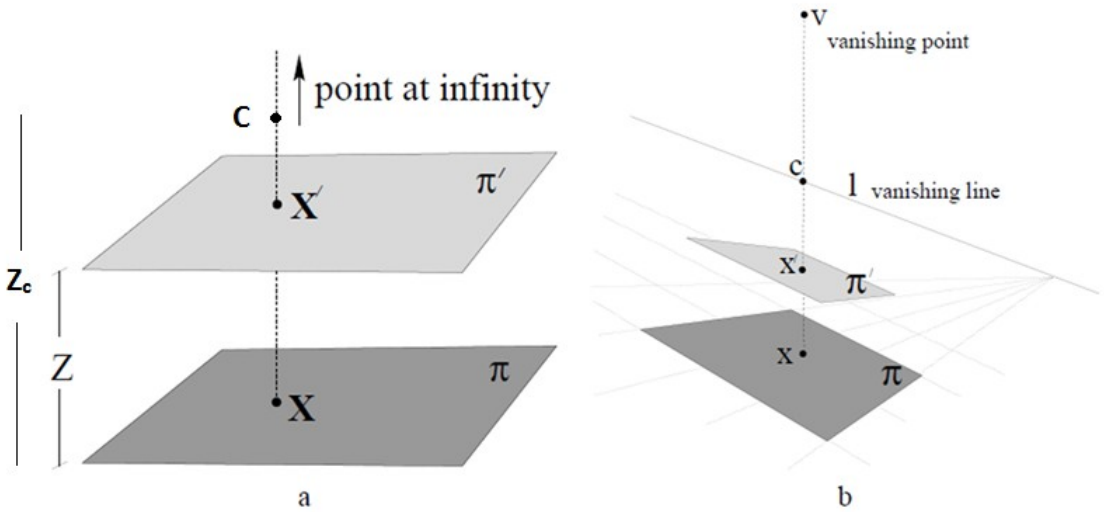


Figure 2.10: Distance between parallel planes [6]

The three aligned points x, x', v and the point of intersection c of the line joining them with the vanishing line define a cross-ratio. The value of the cross-ratio determines a ratio of distances between planes in the world.

Distance between two parallel planes is given by:-

$$\frac{Z}{Z_c} = 1 - \frac{d\left(x', c\right) * d(x, v)}{d\left(x, c\right) * d(x', v)}$$

where Z_c is the distance of the camera center from the reference plane
Distance between scene planes specified by base point $\mathbf{X} = (X,\ Y,\ 0)^T$ on the reference plane and top point $\mathbf{X}' = (X, Y, Z)^T$ is derived as:-

$$x = P \begin{pmatrix} X \\ Y \\ 0 \\ 1 \end{pmatrix}, \qquad \mathrm{x}^{'} = P \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

$$x = \ \rho \left(X p_1 + Y p_2 + p_4\right)$$

$$x' = \rho' \left(X p_1 + Y p_2 + Z p_3 + p_4\right)$$

$$P = \begin{bmatrix} v_x & v_y & \alpha v & \overline{l} \end{bmatrix}$$

$$p_1 \cdot \ \bar{l} = p_1 \cdot \ \bar{l} = 0$$

$$p_4 \cdot \bar{l} = 1$$

$$\rho = \bar{l} \cdot x$$

$$\alpha Z = -\frac{\|x \ \times x'\|}{(\bar{l}.x)\, \|v \ \times x'\|}$$

Projection matrix for parallel plane

$$P = \begin{bmatrix} \ p_1 & p_2 & p_3 & p_4 \ \end{bmatrix}$$

$$P' = \begin{bmatrix} \ p_1 & p_2 & p_3 & Z_r p_3 + p_4 \ \end{bmatrix}$$

The homology between the planes can be obtained from the two projection matrices

$$\mathrm{H} = \begin{bmatrix} \ p_1 & p_2 & p_4 \ \end{bmatrix}$$

$$\mathrm{H}' = \begin{bmatrix} \ p_1 & p_2 & p_3 & Z_r p_3 + p_4 \ \end{bmatrix}$$

Given the homology between two parallel planes we can transfer all points from one plane to the other and make affine measurements in either plane.

Homology $\widetilde{\mathrm{H}} = \mathrm{H}'\mathrm{H}^{-1}$ maps image points on the plane π onto points on π'

$$p_1 \cdot p_4 = 0 \ and \ p_2 \cdot p_4 = 0$$

$$\left(\mathrm{I} + Z_r p_3 {p_4}^T\right) \mathrm{H} = \mathrm{H}'$$

$$\widetilde{\mathrm{H}} = \mathrm{I} + Z_r p_3 {p_4}^T$$

$\widetilde{\mathrm{H}} = \mathrm{I} + \psi v \overline{l}^T$ with $\psi = \alpha Z_r = \alpha Z = -\frac{\|x \ \times x'\|}{(\bar{l}.x)\|v \ \times x'\|}$
Therefore we can calculate ψ if two corresponding points x and x' are known.

Algorithm for measurement between parallel planes

1. Calculate orthogonal vanishing points
2. Calculate the vanishing line
3. Calculate ψ from known point correspondences r and r'
4. Transfer each of the four corner using homology $\widetilde{\mathrm{H}}$
5. Affine rectify the plane
6. Calculate the ratio of the two areas

2.8 Experiments

Single View Metrology experiment to calculate the distance between parallel plane or height of an object

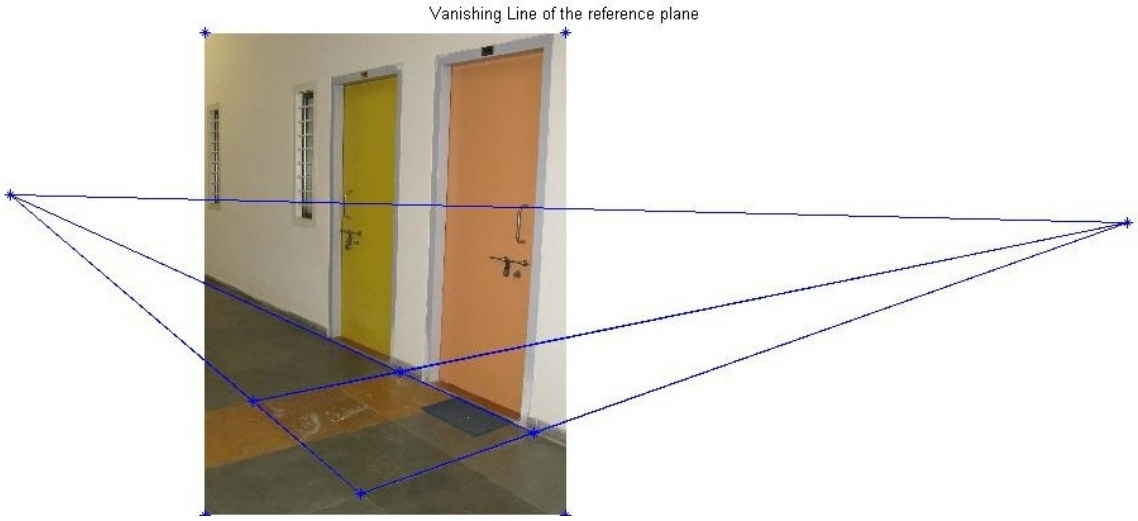


Figure 2.11: Vanishing Line of the reference plane

Figure 2.12: The height of the window is use as reference to measure the height of the door

Figure 2.13: Ratio of Area on Parallel Planes

The slab is of dimension $69cm * 78cm$ and the A4 is of dimension $21cm * 29.7cm$
Actual Area Ratio = $\frac{21*29.7}{69*78} = 0.1158$
Area Ratio predicted = 0.1123

Chapter 3

Bundler System

In this section, we describe some algorithms that are used in the *Bundler* [33, 34] system, which is a *Structure From Motion (SFM)* pipeline which takes mulitple images and gives a sparse point cloud. A modified version of the Bundler system is used in the project. The *RANSAC* algorithm [9] along with the *Five Point Algorithm* [23] are some of important algorithms used to initialize the *Bundle Adjustment (BA)* [38] procedure. RANSAC algorithm is used to select correct point correspondences between two images. The Five Point Algorithm is used to estimate the *essential matrix* between two images. The Triangulation Algorithm [14] is used to find the 3D points. We first describe Bundle Adjustment which is a key component of most SFM systems. We then describe *Levenberg-Marquadt* [20] algorithm which is used to solve the Bundle Adjustment problem. Lastly we describe various stages of a large scale 3D reconstruction system.

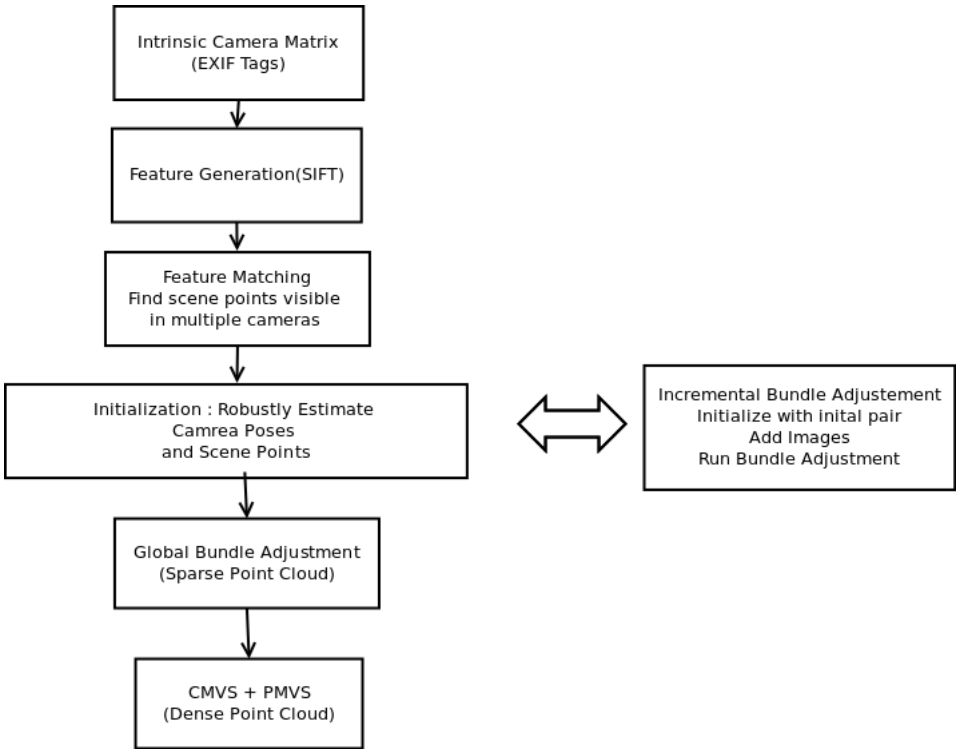


Figure 3.1: Bundler Pipeline

3.1 RANdom SAmple Consensus

RANSAC [9] is a non-deterministic iterative algorithm to estimate parameters of a mathematical model and is used when inlier to outlier ratio is generally high.
The input to the RANSAC algorithm is a set of point correspondences, a parametrized model which can explain or be fitted to the observations, and some threshold parameters.
RANSAC achieves its goal by iteratively selecting a random subset of the original data. These data are hypothetical inliers and this hypothesis is then tested as follows:

1. Select m data points randomly from the original data, where m depends on the minimum number of points required to generate the hypothesis/model, which is equivalent to the complexity of the geometric model

2. The selected points are checked for any degenerate configuration. If points are degenerate, we repeat step 1.

3. A hypothesis is generated from the selected points.

4. Hypothesis Testing: All other data points are then tested against the fitted model using some error metric. The set of data points S_i whose error is less than some given threshold T, is the support/consensus set for the model and the points are considered inliers for the model.

5. If the number of inlier points is greater than the best previous found model or is equal but the error is less than that of the best previous model, then the current model and the inliers are stored as the current best model.

6. Repeat step 1-6 until some termination criteria is met.

7. The model is re-estimated from all hypothetical inliers, because it has only been estimated from the initial set of hypothetical inliers.

The threshold is generally chosen empirically. The RANSAC algorithm terminates when at least one of the random samples is outlier free with probability p. Thus, the minimum number of samples that must be drawn in order to ensure that the probability of all samples being corrupted falls below $1 - p$ is

$$N \geq \frac{\log(1-p)}{\log\ (1-\varepsilon^m)}$$

where ε is the true inlier ratio
Since the true inlier ratio ε is unknown *a priori*, a lower bound on this ratio can be found by using the sample which currently has the largest support. This estimate is then updated as the algorithm progresses.

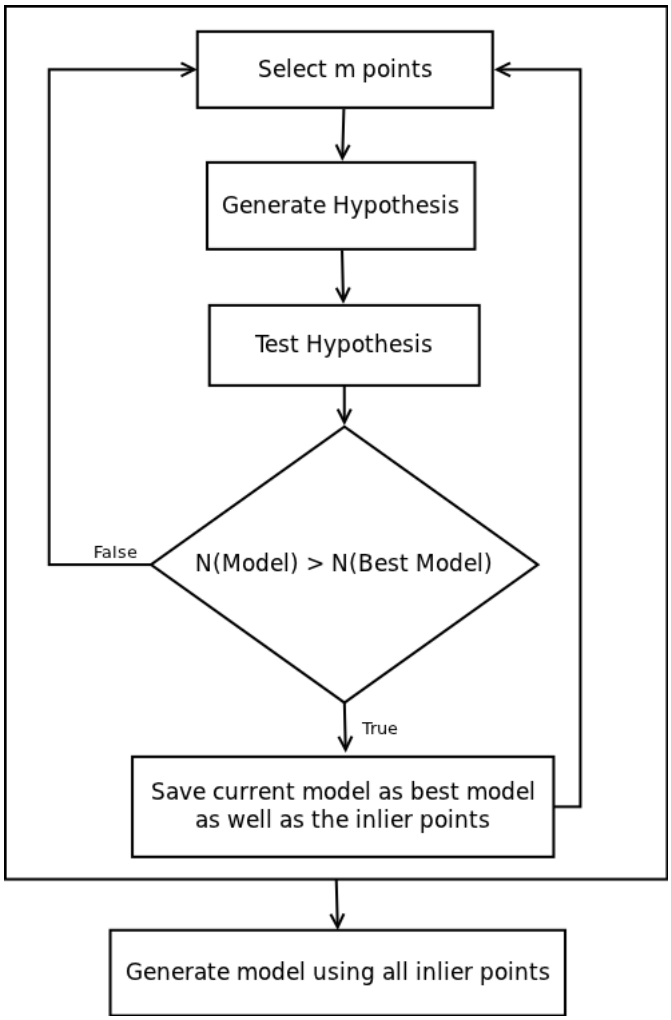


Figure 3.2: RANSAC Algorithm

Various improvements have been suggested to the basic RANSAC algorithm. These can be categorized into two classes:-

1. Improvements in the Hypothesis Generation Phase
   (a) *Progressive Sample Consensus (PROSAC)* algorithm [4]
   (b) *Maximum Likelihood Estimation Sample Consensus (MLESAC)* [37]
   (c) *Lo-RANSAC* [5]
2. Improvements in the Hypothesis Testing Phase Stage
   (a) The *Td,d* Test [3]
   (b) Bail-Out Test [2]
   (c) *WaldSAC* [19]

ARRSAC [26] a real time RANSAC algorithm combine PROSAC based sampling with partially depth-first evaluation technique to meet real time constraints

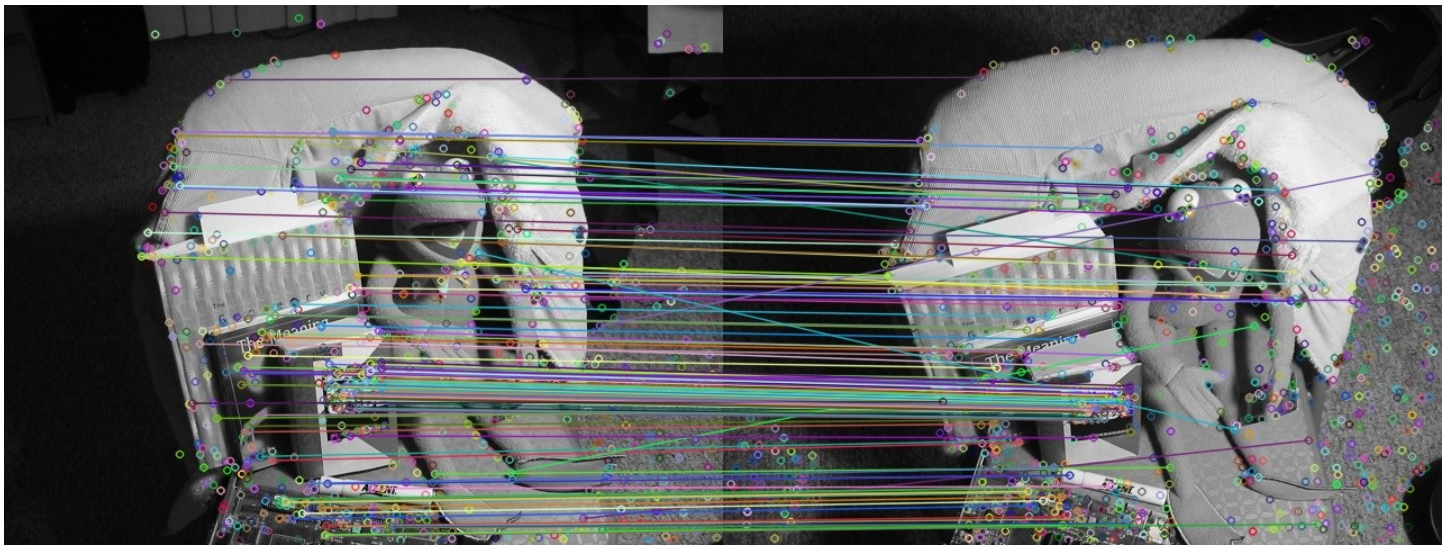


Figure 3.3: Point Correspondences using RANSAC and Five Point Algorithm on KERMIT data set

Figure 3.4: Point Correspondences using RANSAC and Five Point Algorithm on ET data set

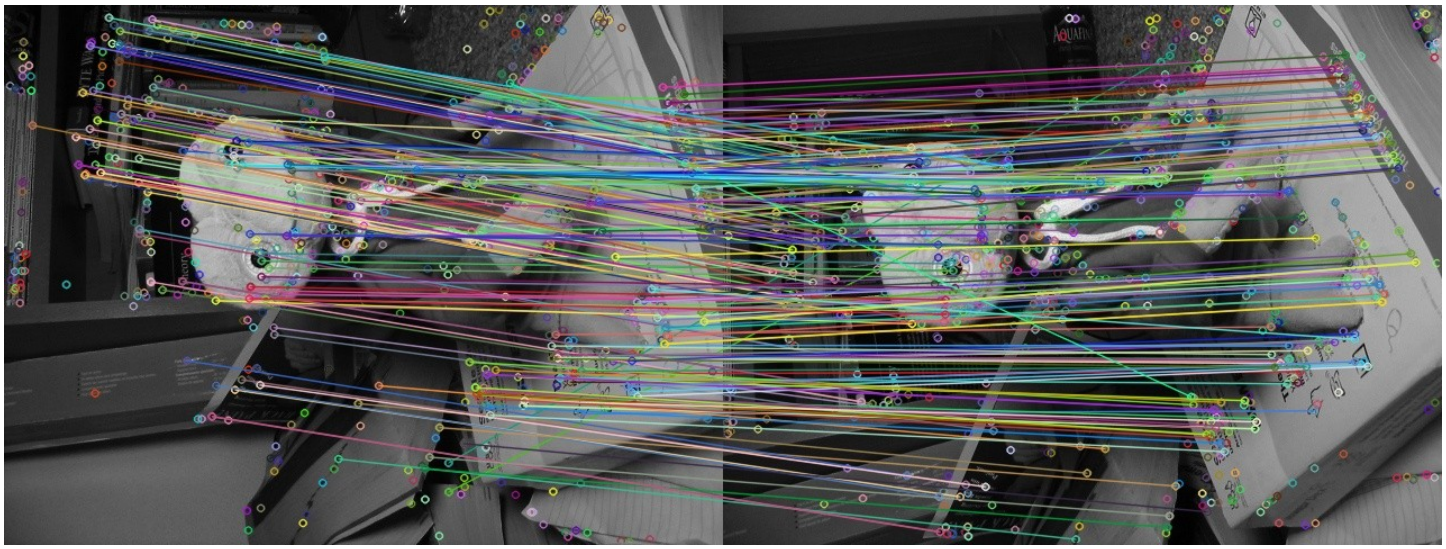


Figure 3.5: Point Correspondences using RANSAC and Eight Point Algorithm on ET data set

3.2 Fundamental matrix/Essential Matrix Estimation

The fundamental/essential matrix is the algebraic representation of epipolar geometry in the case of two images. Essential Matrix can be estimated when the cameras are calibrated. Fundamental Matrix is a generalization of the essential matrix. The camera matrices can be computed from both the fundamental as well as essential matrix. The major difference is that in the case of the fundamental matrix the camera matrices can only be known up to a projective ambiguity whereas in the case of essential matrices the camera matrices can be derived up to a scale and a four-fold ambiguity. Next we discuss the properties of fundamental and essential matrices followed by the various algorithms to compute the fundamental and essential matrix.

3.2.1 Properties of Fundamental Matrix

Given a pair of image for each point in one image there exists an epipolar line in the other image. The fundamental matrix gives a relation between a point and its epipolar line in the other image. The various properties of fundamental matrix are:-

1. Fundamental Matrix is a rank 2 matrix of dimension $3 * 3$ with seven degrees of freedom
2. Point Correspondences: If x and x' are corresponding image points the ${x'}^T F x = 0$
3. Epipolar Lines : $l' = Fx$ is the line corresponding to x
4. $l' = F^T x'$ is the line corresponding to x'
5. $Fe = 0$, where e is the epipole i.e. the image of the second camera center in the first view

6. $F^T e' = 0$ where e' is the image of the first camera center in the second view

7. Canonical Cameras $P = [I \mid 0]$, $P' = [M|m]$
   $F = [e']_\times M = M^{-T}[e]_\times$, where $e' = m$ and $e = M^{-1}m$

3.2.2 Fundamental Matrix Estimation Algorithm

The fundamental matrix is defined by the equation

$$x'^T F x = 0$$

for any pair of matching points $x \; and \; x'$
At least seven matching points are required to compute F
For $x = (x,\ y,\ 1)^T and\ x' = (x',\ y',\ 1)^T$ each point gives linear equation in unknown entries of F

$$x'xf_{11} + x'yf_{12} + x'f_{13} + y'xf_{21} + y'yf_{22} + y'f_{23} + xf_{31} + yf_{32} + f_{33} = 0$$

Where f_{ij} denotes the ith row and jth column unknown entry of the fundamental matrix
For a set of n points we stack the set of linear equations to obtain

$$Af = \begin{bmatrix} x'_1x_1 & x'_1y_1 & x'_1 & y'_1x_1 & y'_1y_1 & y'_1 & x_1 & y_1 & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x'_nx_n & x'_ny_n & x'_n & y'_nx_n & y'_ny_n & y'_n & x_n & y_n & 1 \end{bmatrix} f = 0$$

We can find the least square solution for

$$\min \|Af\| \;\; subject\ to \;\; \|f\| = 1$$

which is the singular vector corresponding to the last column of V where $SVD\ A = UDV^T$
To enforce the rank constraint on the fundamental matrix we can the least singular value of F to zero

$$F = UDV^T where\ D = diag\ (r,\ s,\ t)\ \ with\ r \geq s \geq t$$

Then $F' = U\ diag\,(r,\ s,\ 0)\,V^T$ is rank 2 matrix closest to F such that $\|F - F'\|$ is minimized where $\|\|$ represents the Forbenius Norm
Given N point correspondences $x_i \leftrightarrow x'_i$ the image points are normalized such that their centroid is the origin and their root mean squared distance is $\sqrt{2}$

$$\hat{x}_i = Tx_i \; and \; \hat{x}'_i = T'x'_i$$

where $T \; and \; T'$ are the normalization transformation and are of the form

$$T = \begin{bmatrix} scale & 0 & -scale * centroidx \\ 0 & scale & -scale * centroidy \\ 0 & 0 & 1 \end{bmatrix}$$

Where $(centroidx,\ centroidy)$ is the *centroid* of the points x_i and

$$scale = \frac{\sqrt{2} * N}{\sum_i (x_i - centroid)^2}$$

The algorithms to estimate require the points to be normalized and then the fundamental matrix to be denormalized

$$F = T'^T \hat{F} T$$

3.2.3 Seven Point algorithm

The seven point algorithm works when seven point correspondences are known. The matrix A is a $7 * 9$ matrix and we make use of the singularity constraint to solve for F.
The solution to the equation $Af = 0$ gives a two dimensional space with an unknown α. Using the singularity constraint we can write

$$\det(\alpha F_1 + (1 - \alpha) F_2) = 0$$

Solving the cubic polynomial in α can give either one or three real solutions. Thus the seven point algorithm gives one or three possible solution for the fundamental matrix

3.2.4 Normalized Eight Point Algorithm

When eight or more correspondences are available we apply the eight point algorithm.
The eight point algorithm involves the following steps:-

1. Normalize the point correspondences
2. Calculate $\hat{F}$ by solving $\hat{A}f = 0$ where f is the singular vector corresponding to the smallest singular value of $\hat{A}$ where is constructed from the matches $\hat{x}_i \leftrightarrow \hat{x}'_i$
3. Enforce the constraint that $\hat{F}$ is rank 2
4. De-normalize the fundamental matrix $F = T'^T \hat{F} T$

3.2.5 Essential Matrix Properties

The *essential matrix* properties [13] are:-

1. The determinant of the essential matrix is zero

$$\det(E) = 0$$

2. The essential matrix is 3×3 matrix with two singular values being equal and the third being zero.

$$2EE^T E - trace\left(EE^T\right) E = 0_{3\times 3}$$

3. For a given essential matrix $E = U\Sigma V^T$ where $\Sigma = diag(1, 1, 0)$, and the first camera matrix $P = [I|0]$, there are four possible choices for the second camera matrix P'

$$P' = \left[UWV^T \;\middle|\; +u_3\right] or \left[UWV^T \;\middle|\; -u_3\right] or \left[UW^TV^T \;\middle|\; u_3\right] or \left[UW^TV^T \;\middle|\; -u_3\right]$$

$$where\ W = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

In only one of the four solutions, the object is in front of the camera. In the other three solutions, the object is behind at least one of the cameras.
A single 3D point X is sufficient to decide between the four different solutions for the camera matrix P'

$$depth\left(X;P'\right) = \frac{sign\left(\det\left(M\right)\right)w}{T\left\|m^3\right\|}$$

where $P' = [M \mid t]$, $M = 3 \times 3$ matrix and m is the third row of M and $P'X = wx$. A point lies in front of a camera P' if and only if $depth\left(X;P'\right) > 0$. The point correspondences cannot be normalized as in the case of fundamental matrix [13] as they violate the singular value constraint of the fundamental matrix. The N point correspondences $x_i \leftrightarrow x_i'$ are normalized by

$$\hat{x}_i = K_1^{-1}x_i \; and \; \hat{x}_i' = K_2^{-1}x_i'$$

where K_1 *and* K_2 are the camera calibration matrices of first and second cameras. The point correspondence constraints are:-

$$\hat{x}_i^{'T}E\hat{x}_i = 0$$

The essential matrix is a special case of fundamental matrix when the camera calibration matrix is known. The essential matrix gives camera matrices only up to a scale ambiguity. The *Five Point Algorithm* [23, 35] is used to estimate the essential matrix from five known point correspondences.

3.2.6 Experiments

The kermit and ET data set used for experiments Epipolar Lines : A point in one image is related to a line containing the corresponding point in another image by the fundamental matrix

Figure 3.6: Fundamental matrix epipolar lines ET data set Image 6 and 7

Figure 3.7: Fundamental matrix epipolar lines kermit dataset Image 7 and 8

The variation of residual error as the number of point correspondences increase. Sampson Error was used for hypothesis testing

Sampson Error

$$\sum_i \frac{(x_i'^T F x_i)^2}{(Fx_i)_1^2+(Fx_i)_2^2+(F^T x_i')_1^2+(F^T x_i')_2^2}$$

Residual Error

$$\frac{1}{N}\sum_i^N d(x_i', Fx_i)^2 + d(x_i, F^T x_i')^2$$

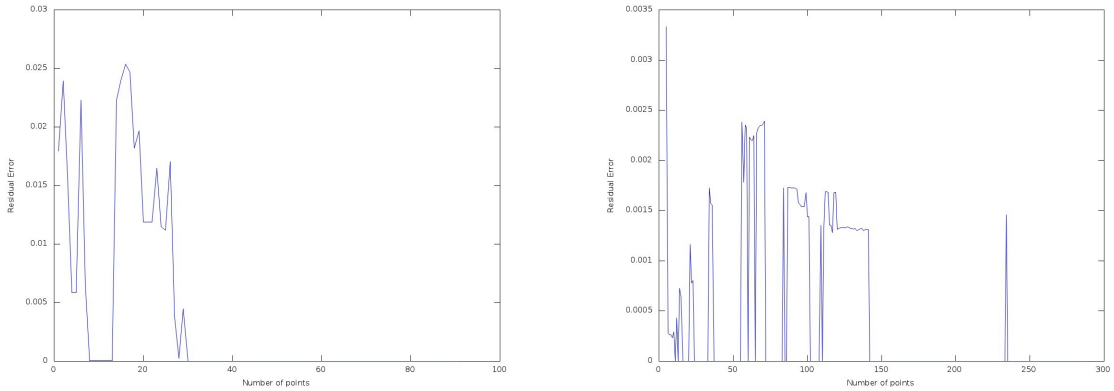


Figure 3.8: Residual Error Five Point Algorithm ET Dataset Image 6 and 7(left) and 1 and 2 (right)

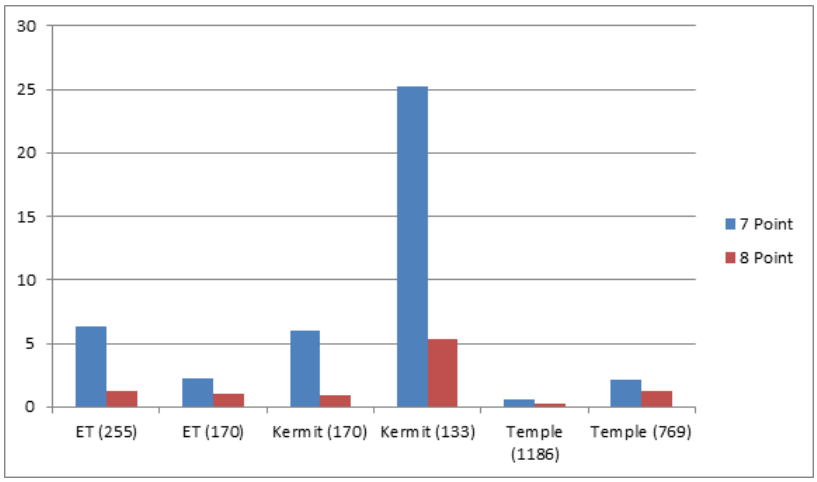


Figure 3.9: Comparison between 8 and 7 point algorithm based on Residual Error. The number in bracket is the number of point correspondences

3.3 Triangulation

Triangulation refers to the process of determining a point in 3D space given its projections onto two, or more, images. In order to solve this problem it is necessary to know the camera matrices. The camera matrices can be obtained from the fundamental or essential matrix with assuming the world origin at first camera $P = K[I|0]$
Each point in an image corresponds to a line in 3D space such that all points on the line are projected to that first point in the image. If a pair of corresponding points $x_i \leftrightarrow x_i'$ in two, or more images, can be found it must be the case that they are the projection of a common 3D point X.

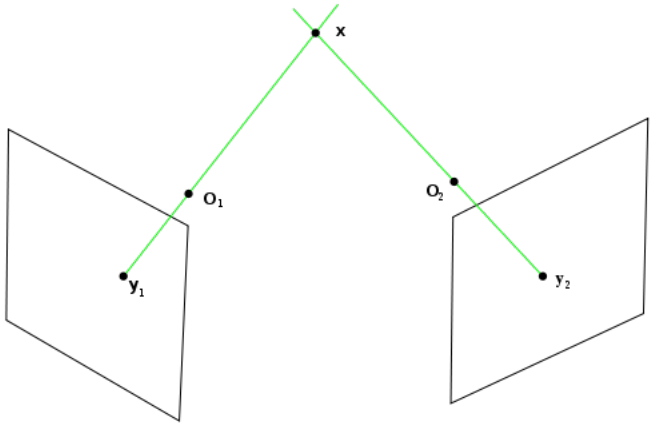


Figure 3.10: Triangulation of 3D point courtesy Wikipedia TriangulationIdeal.svg 13 Oct 2010. http://en.wikipedia.org/wiki/File:TriangulationIdeal.svg

We first present the linear triangulation algorithm:-
Find X given the camera matrices P_1 and P_2 and the point correspondences $x \leftrightarrow x'$

$$x = \ P_1 X \ and \ x' = \ P_2 X$$

The equality is only up to scale, expanding and including the scale factor w gives

$$w \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} P_1 \\ P_2 \\ P_3 \end{bmatrix} X \Rightarrow \begin{matrix} wx = P_1 X \\ wy = P_2 X \\ w = P_3 X \end{matrix}$$

Where P_i is the ith row of P and $\varkappa = (x, y, 1)$ homogeneous coordinate

$$\begin{matrix} P_3 X x = \ P_1 X \\ P_3 X y = \ P_2 X \end{matrix} \implies \begin{matrix} P_3 X x - \ P_1 X = 0 \\ P_3 X y - \ P_2 X = 0 \end{matrix} \implies \begin{bmatrix} P_3 x - P_1 \\ P_3 y - P_2 \end{bmatrix} X = 0$$

Similarly for the second camera , we get

$$\begin{bmatrix} P_3' x - P_1' \\ P_3' y - P_2' \end{bmatrix} X = 0$$

Stacking both the constraints gives us

$$\begin{bmatrix} P_3 x - P_1 \\ P_3 y - P_2 \\ P_3' x - P_1' \\ P_3' y - P_2' \end{bmatrix} X = 0$$

The equation of the form $AX = 0$ and can be easily solved , but we must first normalize the rows of the A matrix such that they are of unit norm

$$A_{norm} = \begin{bmatrix} \frac{A_1}{\|A_1\|} \\ \frac{A_2}{\|A_2\|} \\ \frac{A_3}{\|A_3\|} \\ \frac{A_4}{\|A_4\|} \end{bmatrix}$$

The optimal triangulation algorithm [14] is an non-iterative algorithm which involves solving an six degree polynomial and uses the linear triangulation algorithm as a subroutine.

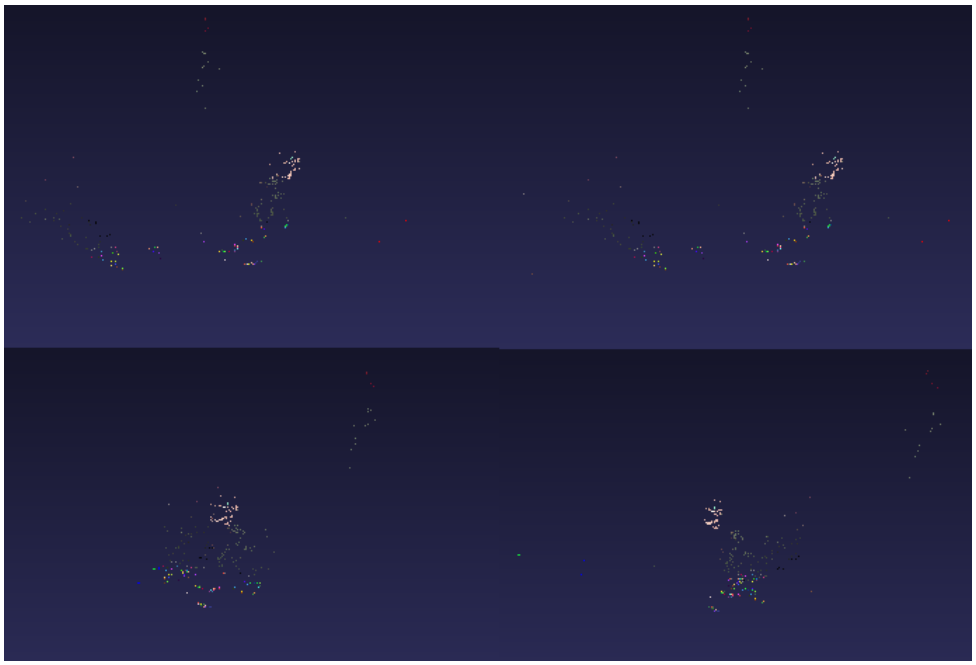


Figure 3.11: Point Cloud generated using Five Point and linear triangulation algorithm on ET data set

3.4 Bundle Adjustment

Bundle Adjustment [38] is the joint optimization of the camera parameters P and the 3D points X in one large optimization problem. *Global Bundle Adjustment* is used as the last step of the 3D reconstruction algorithm. *Incremental Bundle Adjustment* is used when we add a new image to the model. It amounts to an optimization problem on the 3D structure, intrinsic and extrinsic camera parameters and radial distortion coefficients. Bundle Adjustment is the *Maximum Likelihood Estimator* if the noise pertaining to the observed image features have a zero-mean Gaussian distribution. The objective of the Bundle Adjustment problem is to reduce the re-projection error:-

$$\underset{P_i,X_j}{\operatorname{argmin}} \sum_{i=1}^{i=n} \sum_{j=1}^{j=m} \|x_{ij} - Q(P_i, X_j)\|_2^2 * v_{ij}$$

where X_j is 3 dimensional vector representing a 3D points, x_{ij} is 2 dimensional vector representing the 2D point corresponding to camera i and 3D point j and v_{ij} is an indicator function and is one if point j is visible in camera i else zero. n is the number of camera and m is the number of 3D points

The camera model P used in the optimization has the following parameters:-

- focal length f,
- two radial distortion parameters k_1 and k_2
- Rotation R

- Translation t.

The formula for projecting a 3D point X into a camera $Q(P, X)$ is:

$$X_P = R * X + t$$

which converts from the world coordinate to camera coordinate. Then perspective division is carried out followed by conversion to pixel coordinate:

$$p = -P/X_P.z$$

$$p' = f * r(p) * p$$

where $X_P.z$ is the third coordinate of X_P. In the last equation, $r(p)$ is a function that computes a scaling factor to undo the radial distortion:

$$r(p) = 1.0 + k_1 * \|p\|^2 + k_2 * \|p\|^4$$

The equations above imply that the camera viewing direction is:

$$R^T * \begin{bmatrix} 0 & 0 & -1 \end{bmatrix}^T$$

(i.e. the third row of R)

and the 3D position of a camera is $-R^T * t$

The optimization problem is efficiently solved by using *Levenberg-Marquadt* algorithm and exploiting the sparsity pattern of the *Hessian* matrix, as only a subset of 3D points is visible in a particular camera.

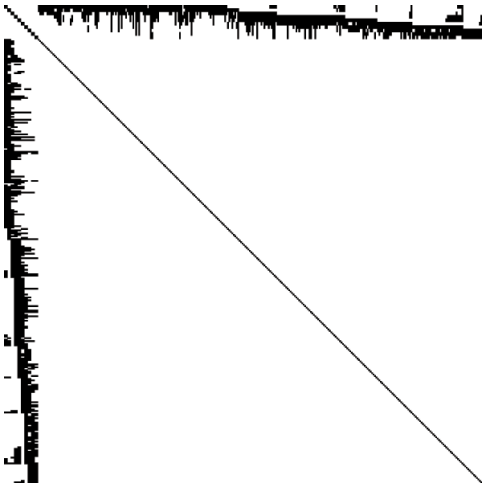


Figure 3.12: Arrow-Head Sparsity Pattern of the Hessian Matrix courtsey wikipedia

3.5 Levenberg-Marquadt Algorithm

The *Levenberg-Marquadt* method [20] is a simple variation on *Newton* iteration designed to provide faster convergence and regularization in the case of over-parametrized problems. It may be seen as a hybrid between *Newton* iteration and a *Gradient Descent* method.

Given a functional relation $X = f(P)$ where X and P are measurement and parameter vectors in Euclidean spaces $\mathbb{R}^N$ and $\mathbb{R}^M$ respectively. We need to find $\hat{P}$ such that $\|\epsilon\| = \|f\left(\hat{P}\right) - X\|$ is minimized. In each iteration, the parameter vector P is replaced by a new parameter vector $P + \Delta$. The function f is approximated by

$$f\left(P_0 + \Delta\right) = f\left(P_0\right) + J\Delta$$

Where P_0 is some initial estimate and $\epsilon_0 = f\left(P_0\right) - X$ and we wish to find $P_1 = P_0 + \Delta$,

$$f\left(P_1\right) - X = f\left(P_0\right) + J\Delta - X = \epsilon_0 + J\Delta$$

Thus we want to minimize $\|\epsilon_0 + J\Delta\|$. The normal equation $J^T J\triangle = -J^T\epsilon$ of *Gauss Newton* iteration method used to find Δ is replaced the augmented normal equation

$$(J^T J + \lambda I)\Delta = -J^T\epsilon$$

where J is the *Jacobian* matrix $J = \frac{\partial f}{\partial P}$, I is the identity matrix and λ is a scalar typically initialized to 10^{-3}.
If the value of Δ is obtained by solving the normal equation leads to a reduction of error, then the increment is accepted and λ is divided by a factor (usually 10). If the value of Δ leads to an increased error, then λ is multiplied by the same factor and the augmented normal equations are solved again.
The Levenberg-Marquadt algorithm moves seamlessly between *Gauss-Newton* iteration, which causes rapid convergence in the neighbourhood of the solution and a gradient descent approach which will guarantee a decrease in the cost function when we are far away from the minima.

Chapter 4

System Overview

We give a brief overview of the architecture of the system that is being used.

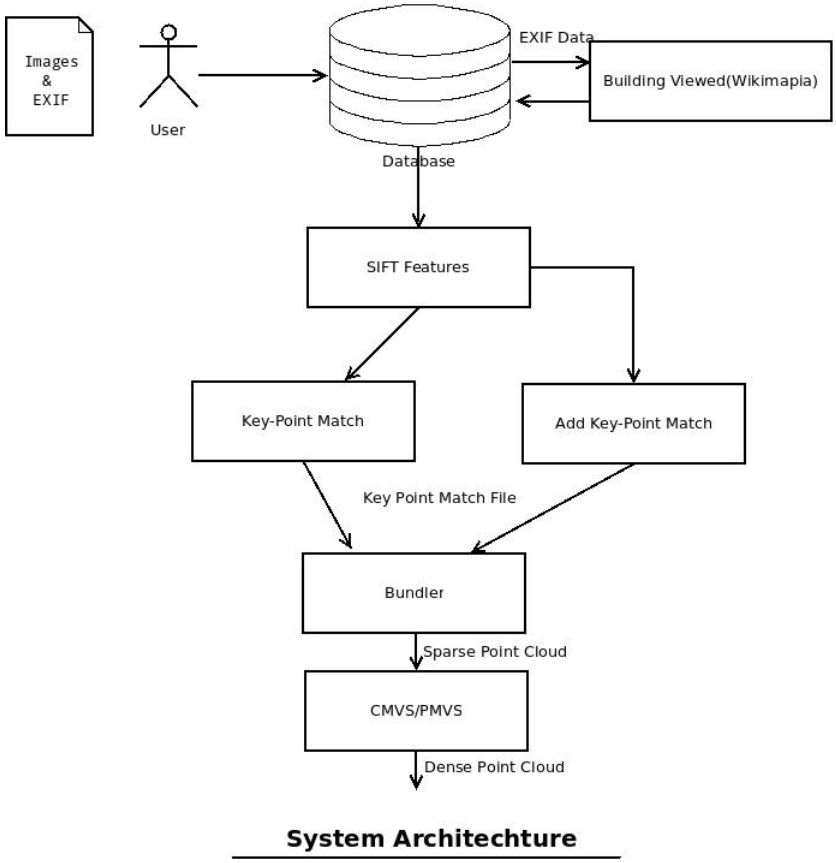


Figure 4.1: System Architecture

1. Upload Images: The website provides an interface for the user to upload multiple images. The *GPS* and direction can be manually entered by the user at the time of uploading the images, or they can be stored in the *EXIF* tag of the images being uploaded.

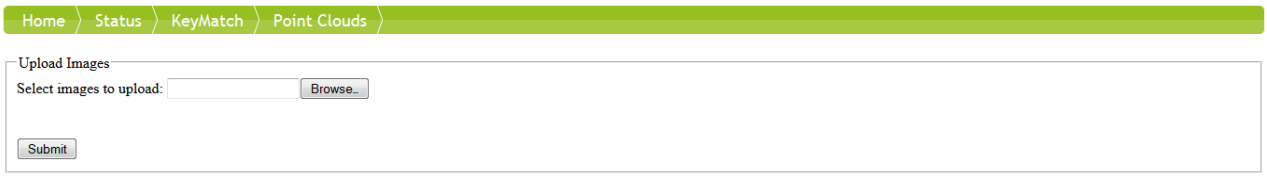


Figure 4.2: Interface for uploading images

Home | Status | KeyMatch | Point Clouds

S.No	Id	Filename	Type	Size	Longitude	Latitude	Direction	Located	Sift
1	178	et000.jpg	image/jpeg	171868	0.0	0.0	0.0	processed	processed
2	179	et001.jpg	image/jpeg	170941	0.0	0.0	0.0	processed	processed
3	180	et002.jpg	image/jpeg	166258	0.0	0.0	0.0	processed	processed
4	181	et003.jpg	image/jpeg	163036	0.0	0.0	0.0	processed	processed
5	182	et004.jpg	image/jpeg	174961	0.0	0.0	0.0	processed	processed
6	183	et005.jpg	image/jpeg	152260	0.0	0.0	0.0	processed	processed
7	184	et006.jpg	image/jpeg	156885	0.0	0.0	0.0	processed	processed
8	185	et007.jpg	image/jpeg	162004	0.0	0.0	0.0	processed	processed
9	186	et008.jpg	image/jpeg	162668	0.0	0.0	0.0	processed	processed
10	187	IMG_4460.JPG	image/jpeg	1069860	72.9160555555556	19.13445	200.0	processed	processed
11	188	IMG_4461.JPG	image/jpeg	1150798	72.9160555555556	19.13445	188.0	processed	processed
12	189	IMG_4462.JPG	image/jpeg	1048567	72.9160555555556	19.13445	170.0	processed	processed
13	190	IMG_4463.JPG	image/jpeg	883208	72.9160555555556	19.13445	163.0	processed	processed
14	191	IMG_4464.JPG	image/jpeg	1356330	72.9160555555556	19.13445	156.0	processed	processed
15	192	IMG_4465.JPG	image/jpeg	1763037	72.9160555555556	19.13445	190.0	processed	processed
16	193	IMG_4466.JPG	image/jpeg	2191695	72.9160555555556	19.13445	180.0	processed	processed

Figure 4.3: Interface for checking status of images

Home | Status | KeyMatch | Point Clouds

S.No	KeyMatchId	ImageId	EdgeId	Model Name	Longitude1	Latitude1	Longitude2	Latitude2	Processed
1	1491	178	134	Center of The Coordinate System, 0 Latitude 0 Longitude	-6.85E-5	9.07E-5	-8.32E-5	7.75E-5	processed
2	1500	179	134	Center of The Coordinate System, 0 Latitude 0 Longitude	-6.85E-5	9.07E-5	-8.32E-5	7.75E-5	processed
3	1509	180	134	Center of The Coordinate System, 0 Latitude 0 Longitude	-6.85E-5	9.07E-5	-8.32E-5	7.75E-5	processed
4	1518	181	134	Center of The Coordinate System, 0 Latitude 0 Longitude	-6.85E-5	9.07E-5	-8.32E-5	7.75E-5	processed
5	1527	182	134	Center of The Coordinate System, 0 Latitude 0 Longitude	-6.85E-5	9.07E-5	-8.32E-5	7.75E-5	processed
6	1536	183	134	Center of The Coordinate System, 0 Latitude 0 Longitude	-6.85E-5	9.07E-5	-8.32E-5	7.75E-5	processed
7	1545	184	134	Center of The Coordinate System, 0 Latitude 0 Longitude	-6.85E-5	9.07E-5	-8.32E-5	7.75E-5	processed
8	1554	185	134	Center of The Coordinate System, 0 Latitude 0 Longitude	-6.85E-5	9.07E-5	-8.32E-5	7.75E-5	processed
9	1563	186	134	Center of The Coordinate System, 0 Latitude 0 Longitude	-6.85E-5	9.07E-5	-8.32E-5	7.75E-5	processed
10	1490	178	133	Center of The Coordinate System, 0	-5.17E-5	1.013E-4	-6.85E-5	9.07E-5	processed

Figure 4.4: Interface for checking status of key-point matching of images

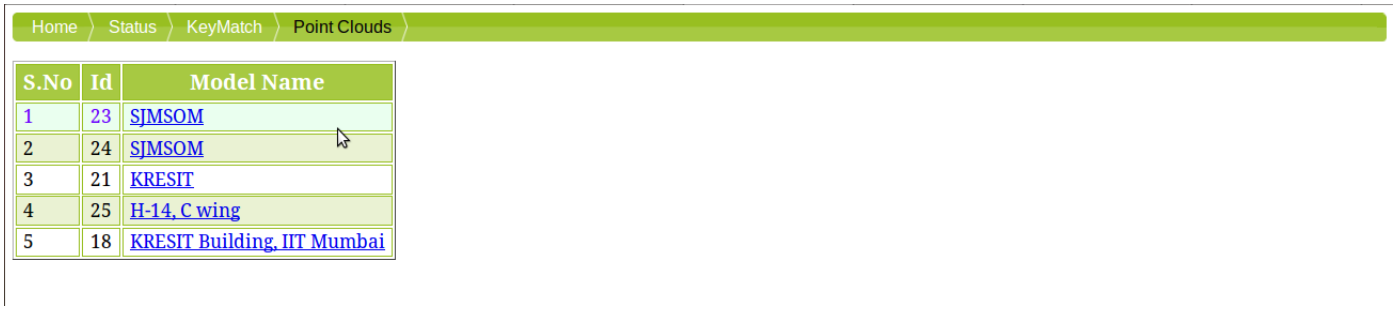


Figure 4.5: Interface for checking point clouds generated

2. Finding Buildings Visible: The longitude and latitude is used to query *Wikimapia* and get all places marked within a fixed window size. The places are marked as polygons on the maps and the longitude and latitude coordinates of these polygons are returned. Using the compass direction, a cone is traced and all polygon edges within the cone are selected. Next, we find which of the edges of the selected polygon edges are actually visible by tracing a fixed number of rays from the position of *GPS* coordinates and finding the nearest polygon edge. All edges which are visible in at least a pre-specified minimum number of rays are selected. The information is stored in the database.

3. Generation of local image features: *SIFT* [18] is used for local image features.

4. Matching of features: The features detected from the previous stage are matched to find out point correspondences between the images.

5. Bundler: Initializes the camera poses and 3D points using *RANSAC* [9, 26] and *Five Point Algorithm* [23]. It then applies *Incremental Bundle Adjustment* [38] and outputs a sparse point cloud.

6. Dense model reconstruction: *Clustering Views for Multi-view Stereo (CMVS)* [11] is used for clustering the images in manageable cluster sizes. *Patch-based Multi-view Stereo (PMVS)* is used to process each cluster in parallel and then fuse the reconstructed model.

Figure 4.6: Stone Chariot Hampi Temple Dense Point Cloud(468 images)

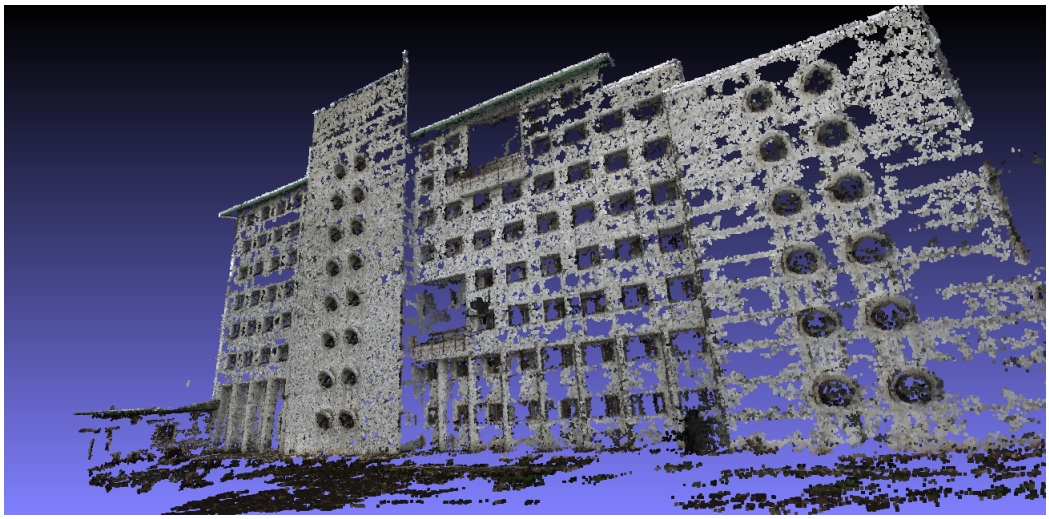


Figure 4.7: Hostel 14 B wing, IIT Bombay(413 images)

7. Displaying of 3D Models: The 3D model is displayed in the browser using *WebGL*. The position of the image and the compass direction is displayed using *Google Map API*.

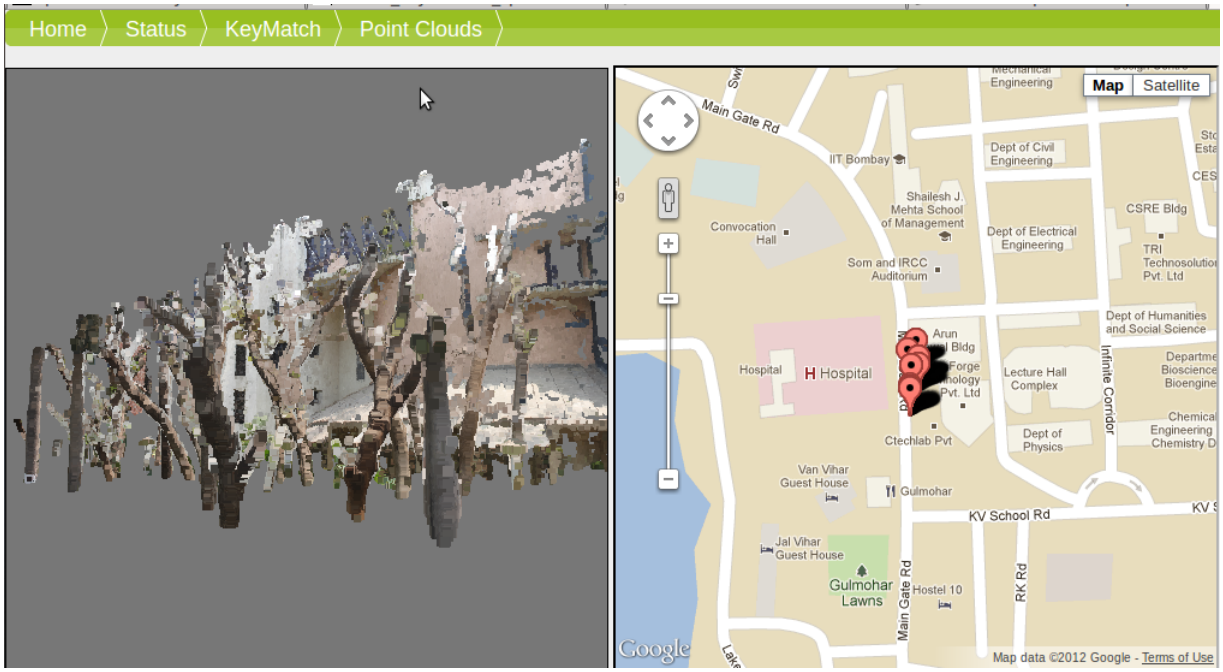


Figure 4.8: Kanwal Rekhi Building, IIT Bombay

4.1 Rules for starting various stages

In this section, we shall discuss the criteria for starting the various stages of the system. When an image is uploaded, we extract the longitude, latitude and direction information from the *EXIF* tags if they are not supplied in the upload form. We save the image along with the longitude, latitude and direction information in the 'upload' table as show in the figure 4.9. The fields 'located' and 'sift_state' field are set to their default values 'not_processed' when the image is uploaded.

The various stages of the pipeline are described below along with the prerequisites required for starting the stage.

- **Key-Point Generation Phase**: No prerequisite for starting of the phase for generation of *SIFT* features [18] of the images. The 'sift_state' field of the 'upload' table which is set to 'not_processed' when the image is uploaded, is set to 'processing' during the feature generation phase. Once the feature generation phase has been completed, the SIFT features are saved in the 'sift' field of the 'upload' table and the 'sift_state' field is set to 'processed'. If the feature generation phase fails, then the 'sift_state' field is set to 'not_processed'.

- **Locating Building Viewed**: There are no prerequisite for starting the phase for locating which building is being viewed by an image using the *GPS* and direction information stored in the 'longitude', 'latitude' and 'direction' fields of the 'upload' table. The 'located' field of the 'upload' table which is set to 'not_processed' when the image is uploaded is updated to 'processing' when we start this stage for an image. *Wikimapia* is queried and the edges of the building outline being viewed are found as described in Chapter 5.1. The

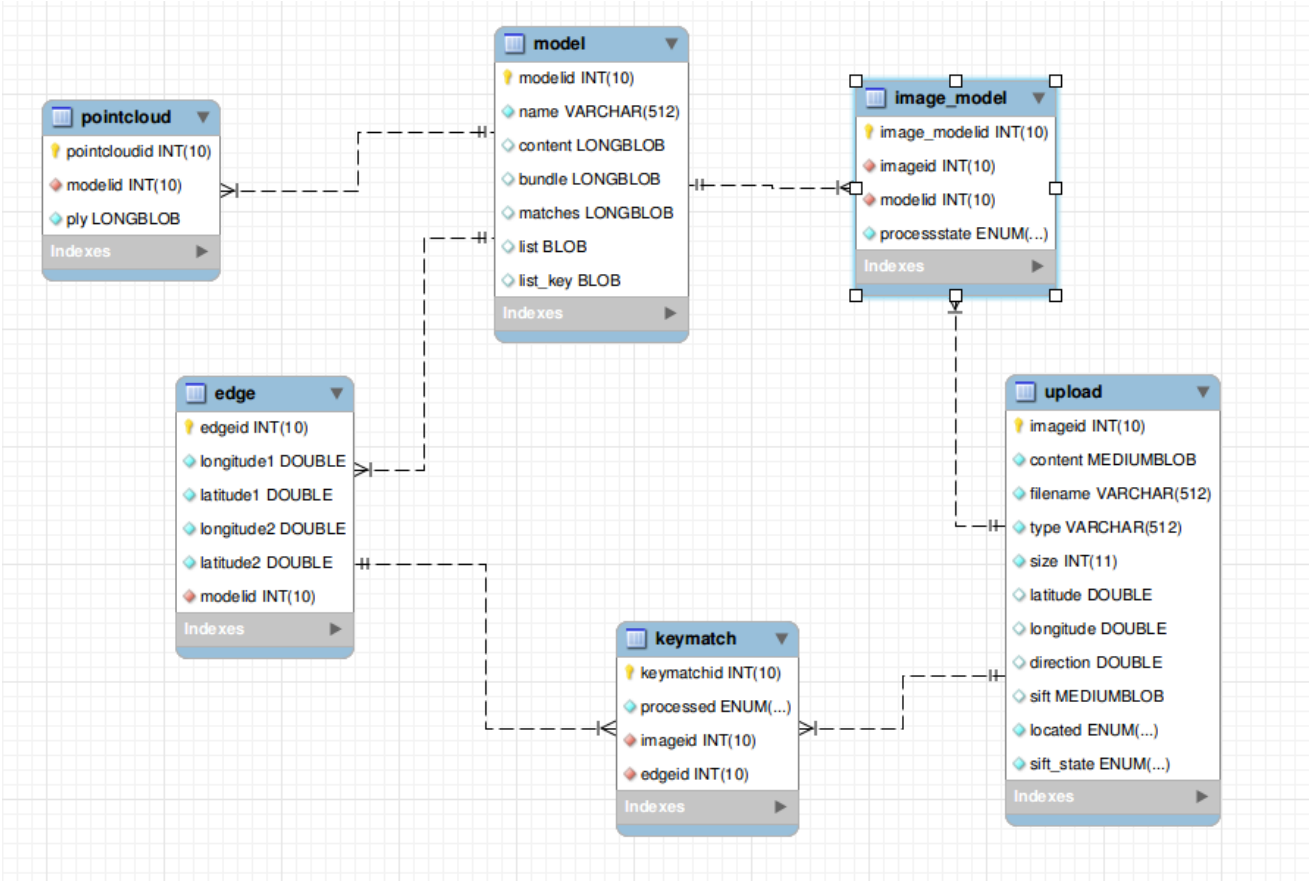


Figure 4.9: ER diagram

building information is added to the 'model' table if the building is already not present. The edge information of the building is added to the 'edge' table. The 'image_model' is updated, a tuple is added for each building image pair, to reflect that the building is viewed in the image. The 'processstate' field of the 'image_model' table is set to its default value 'uploaded'. For each edge which is visible, an entry is added to the 'keymatch' table. The 'processed' field of the 'keymatch' table is set to its default value 'not_processed'. After the successful addition of all the above entries in the table, the value of the 'located' field of the upload table is updated to 'processed'. If no building is visible in the image, then the value of the 'located' field is set to 'failed'. In case of some unknown error, the value is reverted to 'not_processed'.

- **Key-match**: After the above two stages for an image have been completed, an image is eligible for the key-match processing stage. Key-match processing is carried out for a model rather than an image. The following criteria must be satisfied for starting the key-match processing stage of a model:-
  - No key-match processing for the model/building is ongoing
  - *Bundler/PMVS* processing for the model/building should not be under progress
  - No image which has also been located to the same model/building is currently under or waiting for *SIFT* processing

– Number of images meeting above criteria is above a certain threshold. The threshold differs according to whether we are seeding a new model or adding to an existing model

When the key-match processing stage is underway, 'processed' field in the 'edge' table of the image edge pairs of the model being processed is updated to 'processing'. The 'image_model' table field 'processstate' is updated to 'keymatch_processing'. On the successful completion of the stage, the 'processed' field of the 'edge' table for the various image edge pairs being processed is set to 'processed'. The 'image_model' table's field 'processstate' is updated to 'keymatch_processed'. The output of this stage is the 'matches.init.txt' file containing the *SIFT* keypoint matching information about the images and it is saved in the 'matches' field of the 'model' table as a *BLOB*. If the stage cannot be completed 'processed' field the values of both the above fields are reverted back to their original state which is 'not_processed'.

- **Bundler/PMVS**: A model is eligible for Bundler/PMVS processing after the key-match processing stage has been successfully completed. The following criteria must be satisfied before we begin the Bundler/PMVS processing stage:-

  – Bundler/PMVS processing for the model is not currently under processing

  – *SIFT* processing for any image located with the model is not ongoing/pending

  – Key-match processing is not pending/ongoing

  – The number of images waiting to should be processed should greater than some threshold. The threshold varies depending on whether we seed a new model or we are incrementally adding to an existing model.

  The 'processstate' field of the 'image_model' table is updated to 'bundler_processing', before we begin the Bundler/PMVS processing stage. On successful completion of the stage, the 'processstate' field is updated to 'pmvs_processed'. If the execution of the Bundler/PMVS stage is unsuccessful, the original value of the 'processstate' is reverted back to its original value. The failure is noted in the log files. On successful completion, the dense point clouds are generated and these are processed and inserted into 'pointcloud' table. The Bundler/PMVS output is saved in the 'model' table.

Chapter 5

System Modules

5.1 Localization of Building being Reconstructed

In this section, we describe the procedure of finding which building is being currently photographed by the user. The latitude *lat*, longitude *lon* and compass direction *dir* are either extracted from the *EXIF* tag of the images using *Metadata Extractor Library* or manually entered by the user.

The direction is the angle from the longitude in clockwise direction. The direction *dir* is transformed so that we measure the angle θ from the latitude in counter-clockwise direction:

$$\theta = (((-((dir + 90) \bmod 360)) \bmod 360) + 180) \bmod 360$$

The axis a is defined as

$$a = \begin{bmatrix} cos(\theta) \\ sin(\theta) \\ 1 \end{bmatrix}$$

The cone angle $\theta_{coneangle}$ is a fixed parameter set to 40°. We rotate the axis with cone angle $\theta_{coneangle}$ and add it to the vertex v to get the cone point cp_1 as follows:-

$$cp_1 = v + \begin{bmatrix} cos(\theta_{coneangle}) & sin(\theta_{coneangle}) & 1 \\ -sin(\theta_{coneangle}) & cos(\theta_{coneangle}) & 1 \\ 0 & 0 & 1 \end{bmatrix} \cdot a$$

where the

$$v = \begin{bmatrix} lon \\ lat \\ 1 \end{bmatrix}$$

The second cone point cp_2 is similarly calculated

$$cp_2 = v + \begin{bmatrix} cos(-\theta_{coneangle}) & sin(-\theta_{coneangle}) & 1 \\ -sin(-\theta_{coneangle}) & cos(-\theta_{coneangle}) & 1 \\ 0 & 0 & 1 \end{bmatrix} \cdot a$$

The vertex v, cp_1 and cp_2 form a triangle which is used to estimate 2D viewing cone

The centroid c of the triangle is calculated

$$c = \frac{v + cp_1 + cp_2}{3}$$

We calculate the line equations of the sides of the triangle cl_1, cl_2, cl_3 as follows:-

$$cl_1 = v \times cp1$$

$$cl_2 = v \times cp2$$

$$cl3 = cp1 \times cp2$$

and the sign of the centroid w.r.t the cone lines is given as follows:-

$$sign_1 = sign(cl_1 \cdot centroid)$$

$$sign_2 = sign(cl_2 \cdot centroid)$$

$$sign_3 = sign(cl_3 \cdot centroid)$$

which will be used to check whether the edge lies inside the triangle.

The *Wikimapia* site is queried using the latitude *lat* and longitude *lon* coordinates and a window size and the count of number c of entries to be returned. If the number of buildings found is more than the number of entries, we query *Wikimapia* again with c set to number of buildings found.

Wikimapia returns an XML file with the latitude and longitude coordinates of polygon vertex of all the buildings in the window specified.


```
<?access-control allow="*"?>
- <folder language="en" version="1.0" found="6" page="1" count="50">
  - <place id="539375">
      <name>Main Corridor</name>
      <url>http://wikimapia.org/539375/en/Main_Corridor</url>
    - <location>
        <lon/>
        <lat/>
        <north>19.133816</north>
        <south>19.1299639</south>
        <east>72.917558</east>
        <west>72.9155731</west>
      </location>
    - <polygon>
        <point x="72.917129" y="19.129964"/>
        <point x="72.9169464" y="19.1336539"/>
        <point x="72.9155731" y="19.1336641"/>
        <point x="72.9155731" y="19.1337553"/>
        <point x="72.9169357" y="19.1337959"/>
        <point x="72.9170752" y="19.133806"/>
        <point x="72.9175365" y="19.1338161"/>
        <point x="72.917558" y="19.1337046"/>
        <point x="72.9170752" y="19.1336742"/>
        <point x="72.917268" y="19.129964"/>
        <point x="72.917268" y="19.129964"/>
      </polygon>
    </place>
  - <place id="43696">
      <name>Chemical Engineering &amp;amp; Chemistry Department</name>
    - <url>
        http://wikimapia.org/43696/en/Chemical_Engineering_&amp;_Chemistry_Department
      </url>
    - <location>
```


Figure 5.1: Sample XML file returned by Wikimapia

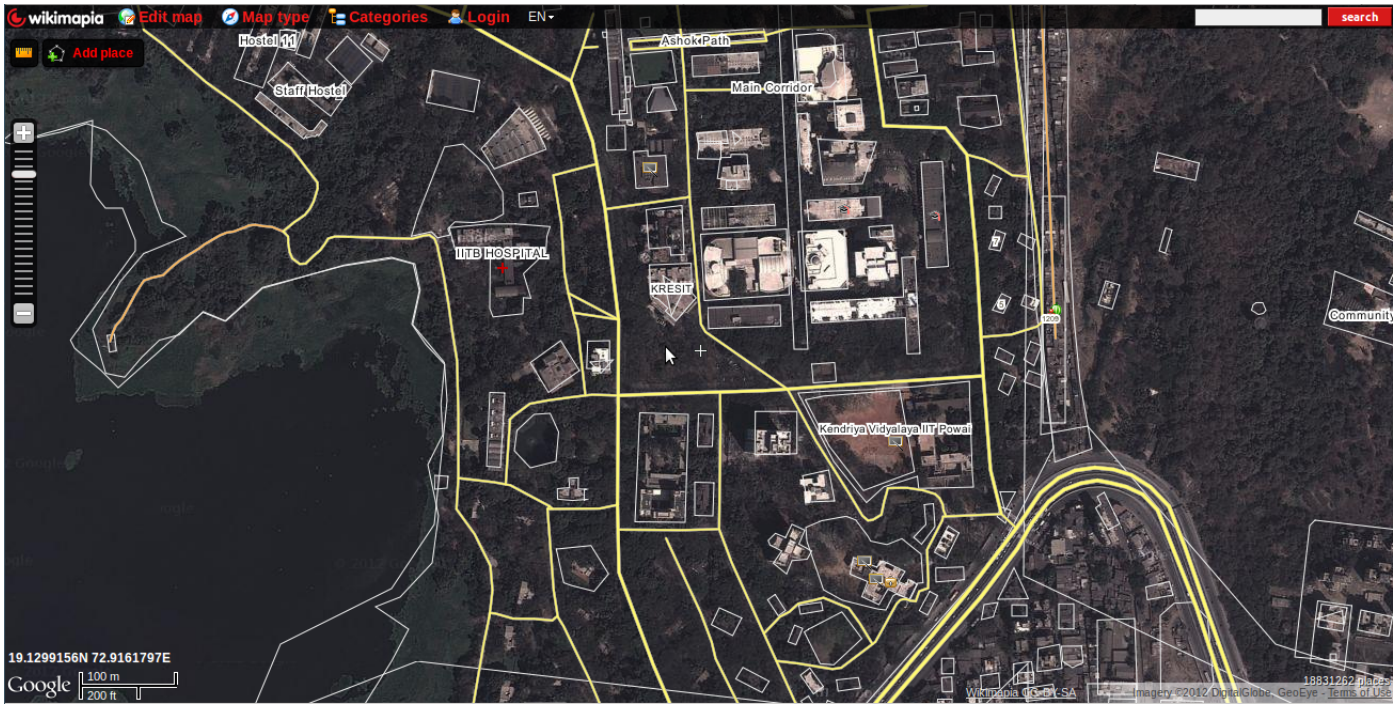


Figure 5.2: Wikimapia Image of IIT Bombay with buildings outlined

From the adjacent polygon vertices (say $pv1$ and $pv2$), we calculate the edge

$$e = pv_1 \times pv_2$$

We check whether each edge e lies within the viewing cone. There are three possible cases:-

1. The edge e intersects cl_1
2. The edge e intersects cl_2
3. The edge e completely lies with the triangle whose vertices are v, $cp1$, $cp1$

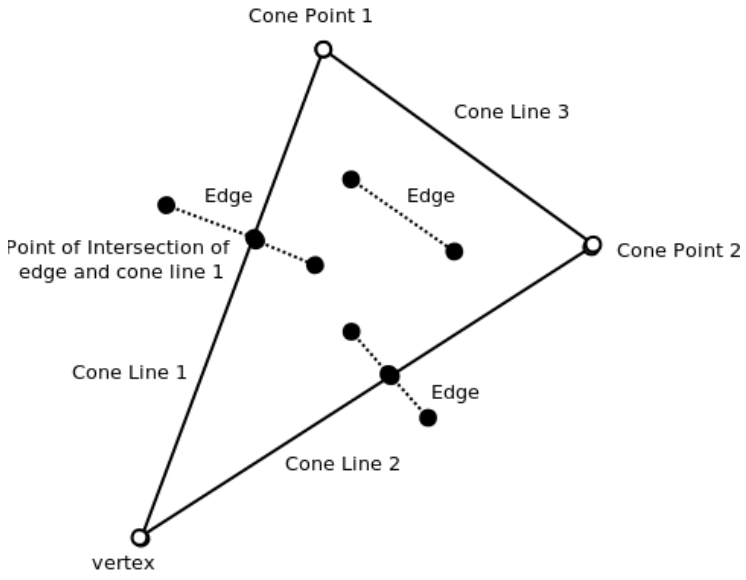


Figure 5.3: Three Conditions under which an edge is visible in the viewing cone

We do not need to check for intersection with cl_3 as it lies outside the window of buildings queried from Wikimapia. To check whether an edge e intersects with cone line cl_1, we take cross product of the edge e with cl_1 to get point of intersection p_1

$$p_1 = e \times cl_1$$

Now we need to check whether the point p_1 lies on the line segment defined by v and cp_1
We use the parametric equation of line segment

$$v + t * (v - cp_1), 0 \leq t \leq 1$$

to represent the cone line cl_1
We solve for t in

$$v + t(v - cp_1) = p_1$$

if $0 \leq t \leq 1$, then the point p_1 lies within the line segment v and cp_1.

Similarly we check that whether the point p_1 lies within the line segment pv_1 and pv_2. If the point of intersection p_1 lies within both the line segments cone line 1 cl_1 and edge e, then we conclude that the edge is visible in the viewing cone. If the above condition is not true then we need to check whether the cone line cl_2 intersects with the edge e. If cone line cl_2 intersects with the edge e, the edge is visible in the viewing cone, else we check for the last condition whether the edge e lies completely within the viewing cone. To check whether the edge e lies completely within the triangle, we take the two end points of the edge pv_1 and pv_2 and find their dot product with cone line cl_1, cl_2, cl_3 as follows:-

$$sign_{1pv_1} = sign(cl_1 \cdot pv_1)$$

$$sign_{2pv_1} = sign(cl_2 \cdot pv_1)$$

$$sign_{3pv_1} = sign(cl3 \cdot pv_1)$$

$$sign_{1pv_2} = sign(cl_1 \cdot pv_2)$$

$$sign_{2pv_2} = sign(cl_2 \cdot pv_2)$$

$$sign_{3pv_2} = sign(cl3 \cdot pv_2)$$

Then we check whether

$$sign_1 = sign_{1pv_1}$$

$$sign_2 = sign_{2pv_1}$$

$$sign_3 = sign_{3pv_1}$$

for pv_1 and similarly for pv_2 If all the conditions are true, then the edge e lies completely within the viewing cone.

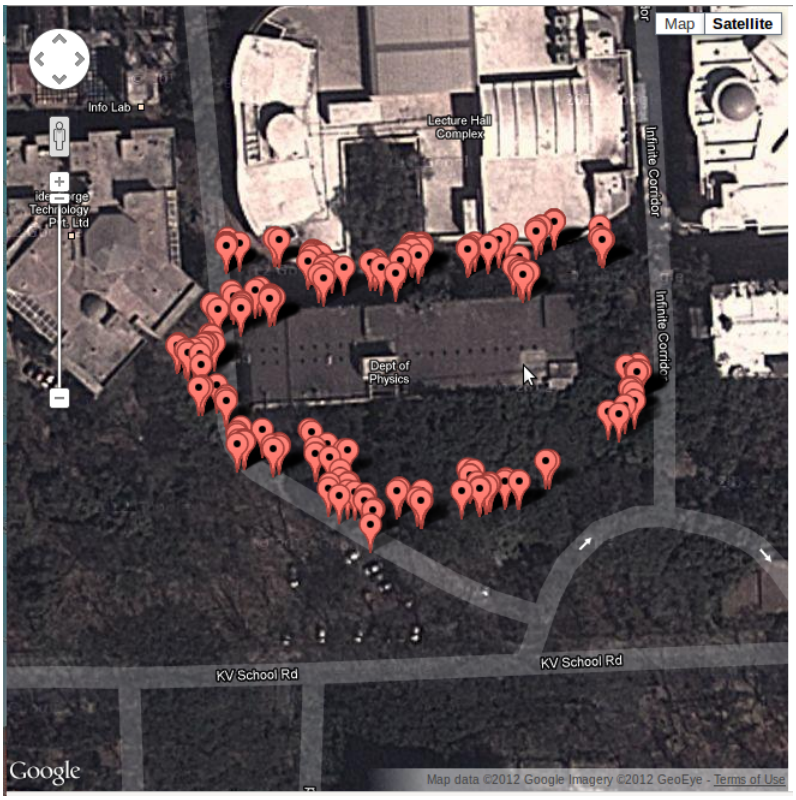


Figure 5.4: GPS positions of images taken around Physics Department IIT Bombay

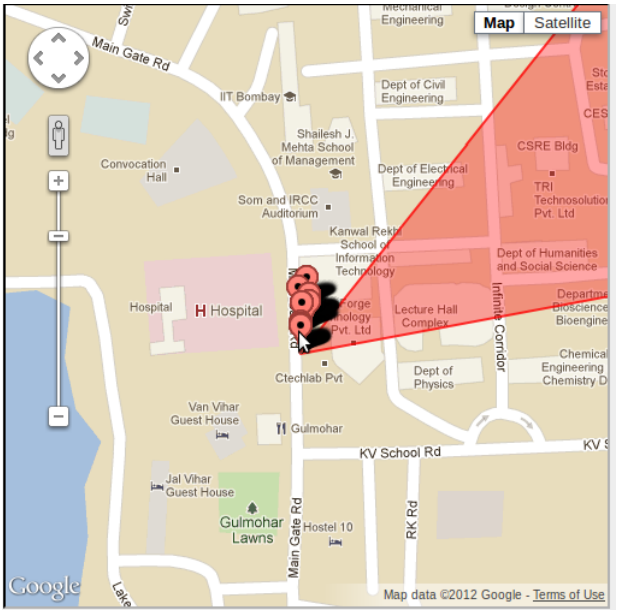


Figure 5.5: GPS positions of images taken around Kanwal Rekhi Building IIT Bombay with viewing cone of a photograph highlighted

We select all the edges which satisfy the above conditions. Now we find which of the edges in the viewing cone are visible i.e. not occluded by some other edge.

We trace rays from the vertex v with θ varying from $\theta_{coneangle}$ to $-\theta_{coneangle}$ with step size

1° and with respect to the axis a. A particular ray r is calculated as

$$r = v \times rp$$

where ray point rp is given by

$$rp = v + \begin{bmatrix} cos(\theta) & sin(\theta) & 1 \\ -sin(\theta) & cos(\theta) & 1 \\ 0 & 0 & 1 \end{bmatrix} \times a$$

We find the intersection of the ray r_i with the edge e_j

$$p_{r_i e_j} = r_i \times e_j$$

and check whether the point of intersection lies in the two line segment given by ray r_i and edge e_j

We solve for t in

$$v + t(v - rp_1) = p_{r_i e_j}$$

If $0 \leq t \leq 1$, then the point $p_{r_i e_j}$ lies within the line segment v and rp_1. Similarly solve for the other line segment.

We find the distance d between the points v and $p_{r_i e_j}$ using

$$d_{r_i e_j} = \sqrt{v^2 + p_{r_i e_j}^2}$$

For each edge we maintain a quantity known as visibility count and initialize it to zero. We find the closest edge amongst all the edges which intersects with the ray and increase visibility count of that edge by one. At the end of the process, any edge with a visibility count above a certain threshold (say 10) is said to be visible in the cone. Now for each image, we get a set of edges with their corresponding buildings which are visible in it. We save this information in the database. Using the stored information about the visible edges and buildings in each image we are able to reduce the number of matches which need to be performed in the key-matching phase. Only those images are matched to one another which share an edge i.e. in both of the images, the building edge is visible.

5.2 Feature Matching

The key-point matching module is the most computationally expensive and time taking of all the stages of the *Bundler* pipeline. As the number of images increases the complexity of the key-match point algorithm increases quadratically ($O(n^2)$).

	SIFT	Keymatch	Bundler	PMVS
SOM(376)	305	**2758**	95	112
H14-C(302)	186	**433**	19	414
H14-B(414)	244	**376**	253	143

Table 5.1: Computation time in minutes for various stages of the bundler pipeline(original) on different datasets

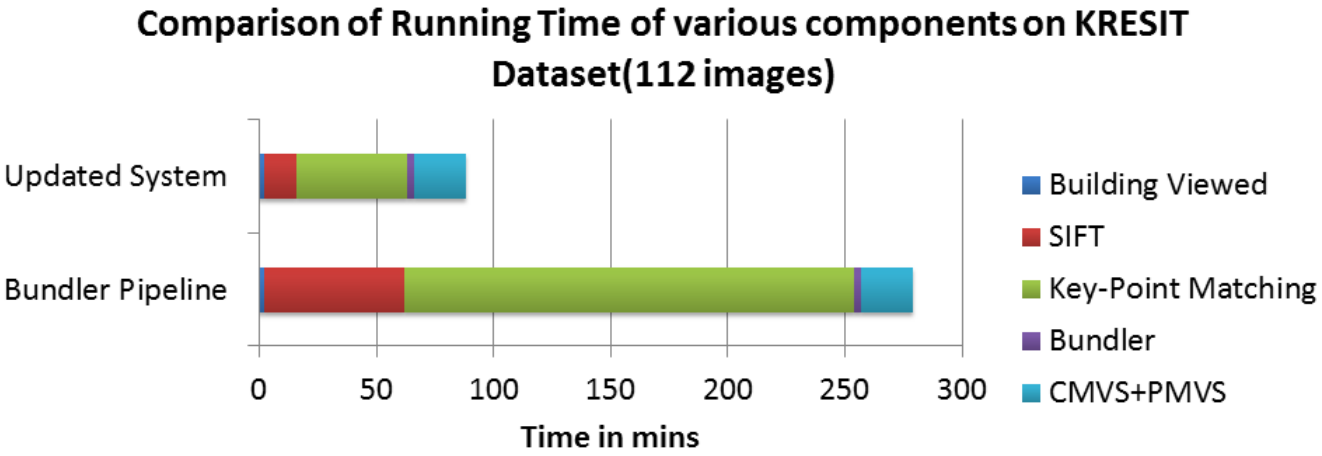


Figure 5.6: Comparison of Running Time of various component on KRESIT Dataset of 112 images

The original Bundler system used the *Approximate Nearest Neighbor(ANN)* library for matching of features in high dimensions. The *ANN* library is not thread safe. The matching module has been entirely replaced and now the *FLANN*[21] library is used which is thread safe and supports multi-threading.

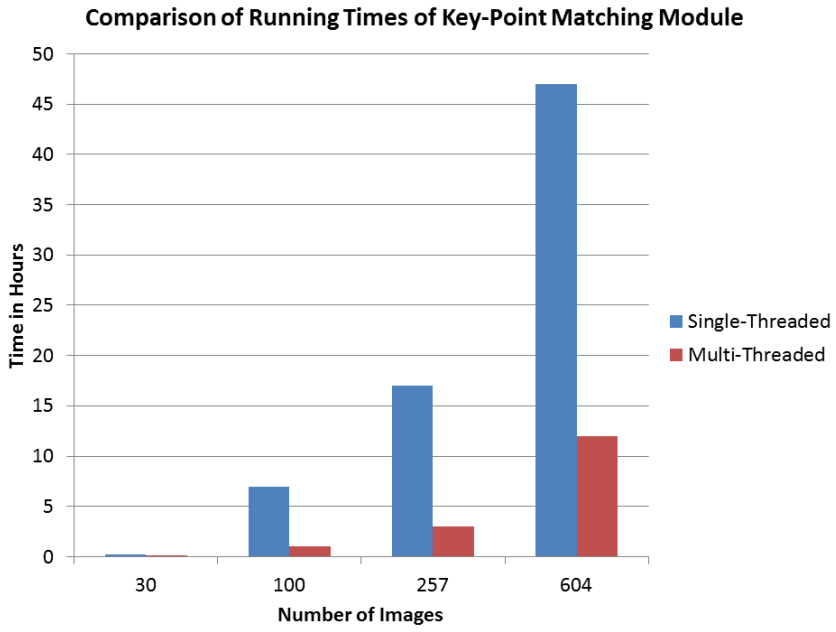


Figure 5.7: Comparison of key-point matching module

We use randomized kd-trees of the FLANN library for searching in high dimensional(128 length) feature space. Classical kd-trees performance degrades rapidly as the dimensionality of

the feature space increases. Randomized kd-tree overcomes this problem by maintaining multiple randomized kd-trees. Randomized kd-tree are built by choosing split dimension randomly from the first k dimensions. *Boost* multi-threading library is being used in the multi-threaded key-point matching module.

The key-match program consists of following stages

1. Load the *SIFT* key files
2. Build the k-d trees
3. Match the image pairs
4. Write the output to the file

5.2.1 Challenges

The keymatch algorithm requires us to match multiple pairs of images. Each image can be considered a node in the graph, where an edge between two nodes represent that the two images keypoints need to be matched. The graph is a complete graph, all pair of nodes are connected. The building outline information fetched from Wikimapia, can be utilized to remove some of the edges in the graph. some of the challenges that required to be addressed during the design of the module where to increase concurrency, reduce memory load and decrease contention between the threads. In the first version of multi-threaded key-matching that we implemented, we used a mutex to write output to a single file named *'matches.init.txt'*. *'matches.init.txt'* file must be written such that the key-match output of images is sorted by index of image i and j. The images are assigned indexes according to the order in which they appear in the *'list.txt'* file. There was also a common task pool from which the threads picked the next task i.e. image to be processed next. This lead to contention between the various threads and the performance did not improve.

In the second version we tried, we strived to reduce the contention between the threads. The output of matching each image pair was written to a separate file. The files where combined together using shell script after the execution of the keymatch phase was over. For a pair of image i, j the file written is *i:j.keyPoints*. Each thread loaded into its private memory the *SIFT* key files it was assigned to process, this lead to increase in memory requirements and loading time.

In the final version we load the data files and build indexes only once in the main program. We had two mutex arrays one for the keypoint data and the other for the kd-trees. A thread trying to match image i and j must acquire the kd-tree mutex of i and the data mutex of j to proceed. This granularity of mutual exclusion together with writing output to independent files lead to the performance improvement.

5.3 GPS Constrained Bundle Adjustment

We use the *GPS* information present in the images to constraint the *Bundle Adjustment* problem. We add to the objective of *Bundle Adjustment* a *GPS* error term

$$\underset{P_i,X_j}{\operatorname{argmin}} \sum_{i,j}^{n,m} \|x_{ij} - Q(P_i, X_j)\|_2^2 * v_{ij} + \lambda \sum_{i}^{n} \|x_{gps_i} - F(P_i)\|^2$$

where P_i is the i^{th} camera parameter, X_j is the j^{th} three-dimensional vector of 3D points, x_{ij} is two-dimensional vector of the 2D point corresponding to camera i and 3D point j. v_{ij} is an indicator function and is one if point j is visible in camera i else zero. n is the number of cameras. m is the number of 3D points. $Q(P, X)$ is the projection model. λ is a constant.

Camera position is given by

$$C = \begin{bmatrix} c_x \\ c_y \\ c_z \end{bmatrix} = -R' * t$$

where R is rotation matrix and t is the translation vector of the extrinsic camera matrix. We ignore the last coordinate and get 2D point 'c'. For a single camera we get the following relation between the camera center and the *GPS* co-ordinate of the camera

$$x_{gps} = \begin{bmatrix} x_{gps_1} \\ x_{gps_2} \end{bmatrix} = s * \begin{bmatrix} cos(\theta) & sin(\theta) & t_x \\ sin(\theta) & cos(\theta) & t_y \end{bmatrix} * \begin{bmatrix} c_x \\ c_y \\ 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix} * \begin{bmatrix} c_x \\ c_y \\ 1 \end{bmatrix} = B * c$$

$$a_{11} = a_{22}$$

$$a_{12} = -a_{21}$$

Using all the $2 * n$ equations from the n cameras

$$\begin{bmatrix} c_{x_1} & c_{y_1} & 1 & 0 \\ c_{y_1} & -c_{x_1} & 0 & 1 \\ c_{x_2} & c_{y_2} & 1 & 0 \\ c_{y_2} & -c_{x_2} & 0 & 1 \\ \vdots & \vdots & \vdots & \vdots \\ c_{x_n} & c_{y_n} & 1 & 0 \\ c_{y_n} & -c_{x_n} & 0 & 1 \end{bmatrix} * \begin{bmatrix} a_{11} \\ a_{12} \\ a_{13} \\ a_{23} \end{bmatrix} = \begin{bmatrix} x_{gps_{11}} \\ x_{gps_{12}} \\ x_{gps_{21}} \\ x_{gps_{22}} \\ \vdots \\ x_{gps_{n1}} \\ x_{gps_{n2}} \end{bmatrix}$$

We solve the above equation which is of the form

$$Ax = b$$

by taking the normal equation

$$x = (A^T A)^{-1} A^T * b$$

as A is $2n * 4$ and $2n \gg 4$ we first multiply $A^T A$ to get a $4 * 4$ matrix, then find its inverse $(A^T A)^{-1}$ we then multiply $A^T * b$ and get a $4 * 1$. We multiply the two matrices to get $4 * 1$ vector. The vector x is used to obtain the matrix B

$$\hat{x}_{gps_i} = B * -R'_i * t_i = F(P_i)$$

The software package *Scalable Bundle Adjustment SBA* [17] required that we use a single dummy 3D point, which is visible in all the cameras to incorporate the *GPS* error.

The above formulation has been tried, but the result were not encouraging. The main issue faced by us during the implementation was:-
The *GPS* data collected by us using smart-phone *A-GPS* has accuracy in meters(5m-30m), the data may be too noisy and can cause *Bundle Adjustment* to fail.
Some of the possible remedies could be:-

- We could incorporate the accuracy of the measured *GPS* co-ordinate as diagonal elements of the co-variance matrix.
- Use *Differential GPS(D-GPS)* which as an accuracy of 10cm. Such system are not available in smart-phones and thus is not practical.

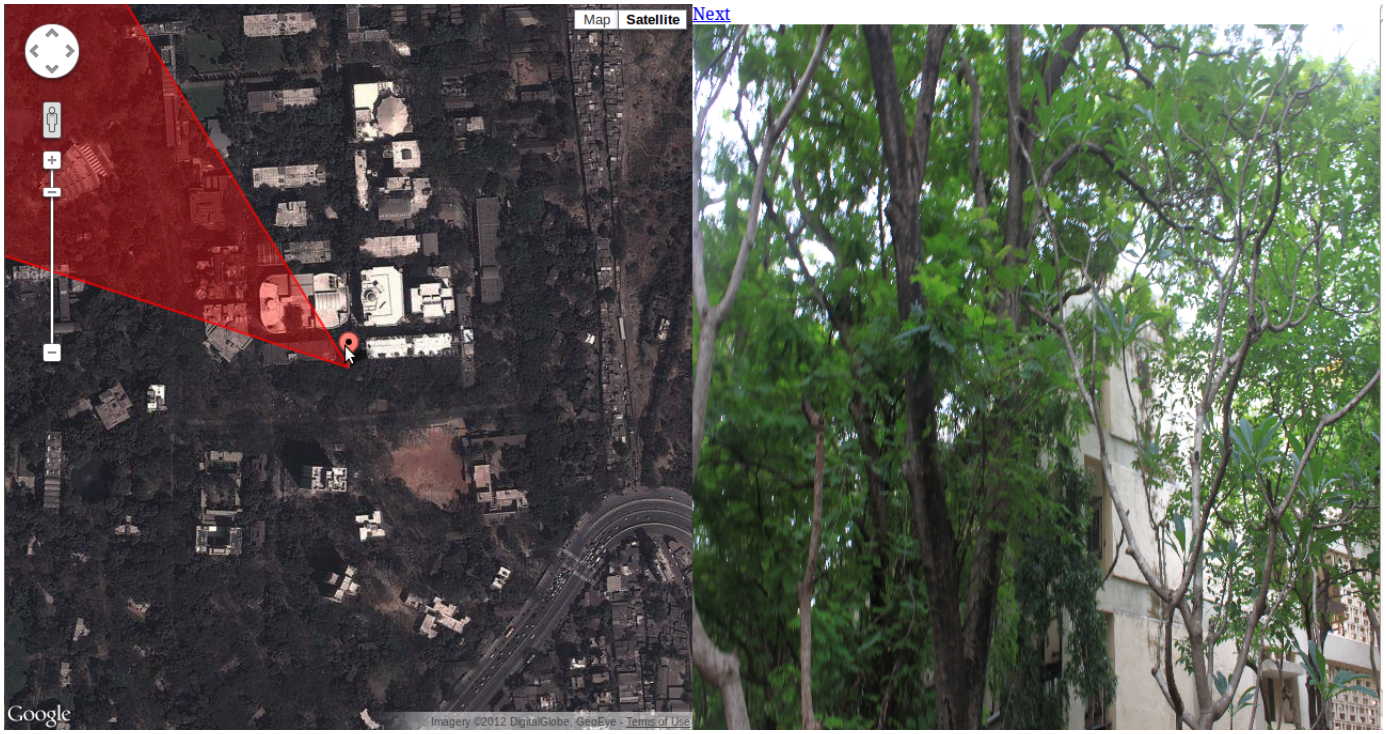


Figure 5.8: Correct GPS, Physics Department IIT Bombay

Image courtesy Google Maps last accessed 20th June, 2012

Figure 5.9: Incorrect GPS, Physics Department IIT Bombay

Image courtesy Google Maps last accessed 20th June, 2012

Figure 5.10: Totally Incorrect GPS, Physics Department IIT Bombay

Image courtesy Google Maps last accessed 20th June, 2012

Chapter 6

Results

In this section, we will discuss the various results obtained using the updated *Bundler* system. We look at the improvements in the computation time of the various modules of the *Bundler* pipeline on different datasets. We show around a 6x improvements in the *SIFT* key-point generation phase and also about 4-5x improvements in the running time of the key-match module.

	SIFT (Bundler)	SIFT (parallel)	Key-match (Bundler)	Key-match (multi-threaded)	Bundler	PMVS
SOM(376)	305	**55**	2758	**564**	95	112
H14-C(302)	186	**47**	433	**90**	19	414
H14-B(414)	244	**50**	376	**80**	253	143

Table 6.1: Computation time in minutes for various stages of the bundler pipeline(single/multi-threaded) on different datasets

The website can be accessed inside IIT Bombay using the following URL http://10.129.1.123:8080/vision
The project page can be accessed using the following URL `http://trellis.cse.iitb.ac.in/~srijit`

The figure 6.3 shows point-cloud generated for the datasets 6.1. *GPS* information of the various images in the *H-14 C wing* dataset is visualized in figure 6.2 :-

Figure 6.1: Hostel 14 C wing, IIT Bombay sample images

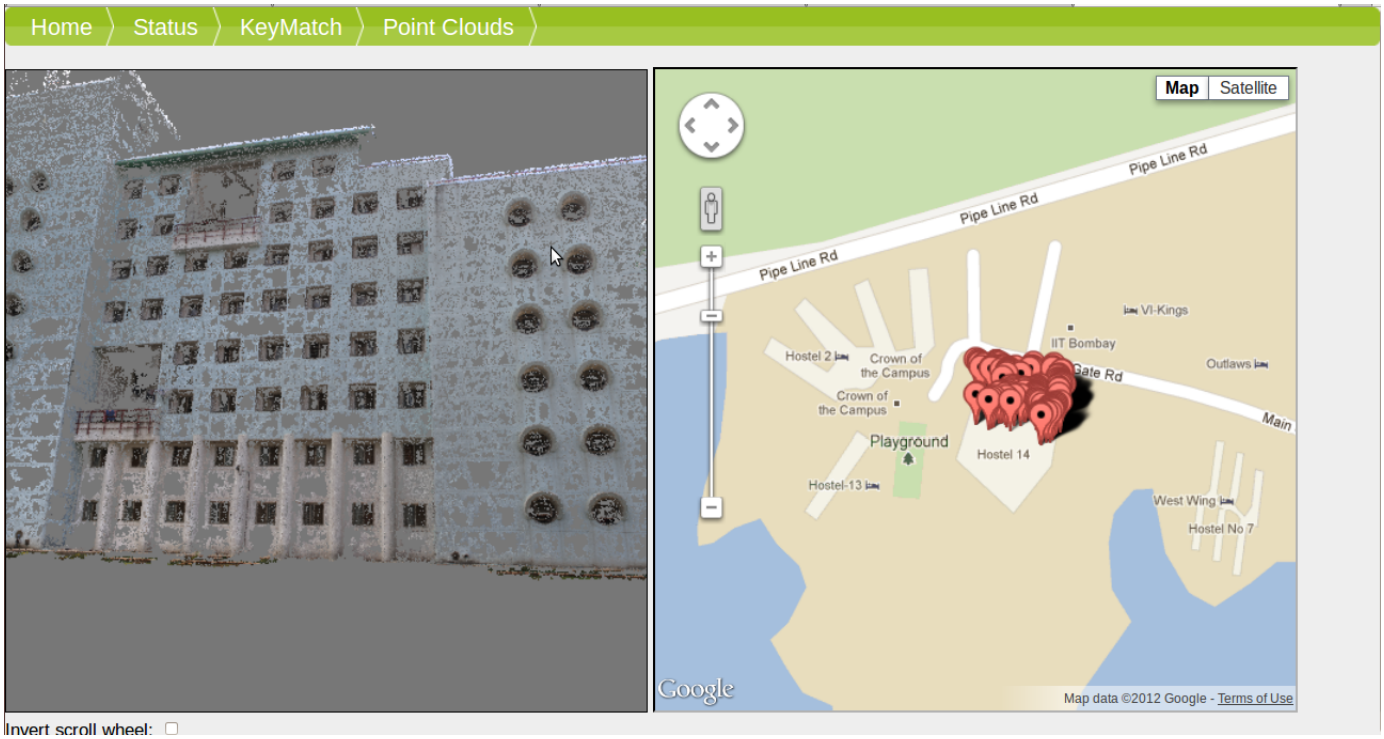


Figure 6.2: Hostel 14 C wing, IIT Bombay Point Cloud and GPS information(302 images)

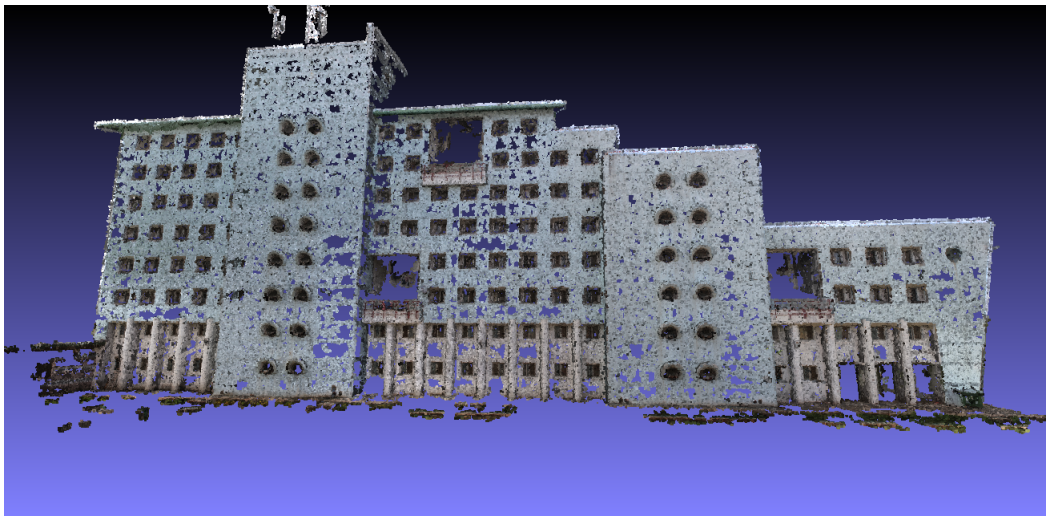


Figure 6.3: Hostel 14 C wing, IIT Bombay Point Cloud (302 images)

The figure 6.7, 6.6 shows point-cloud generated for the datasets 6.4. *GPS* information of the various images in the *Shailesh J. Mehta School of Management* dataset is visualized in figure 6.5 :-

Figure 6.4: Shailesh J. Mehta School of Management, IIT Bombay sample images

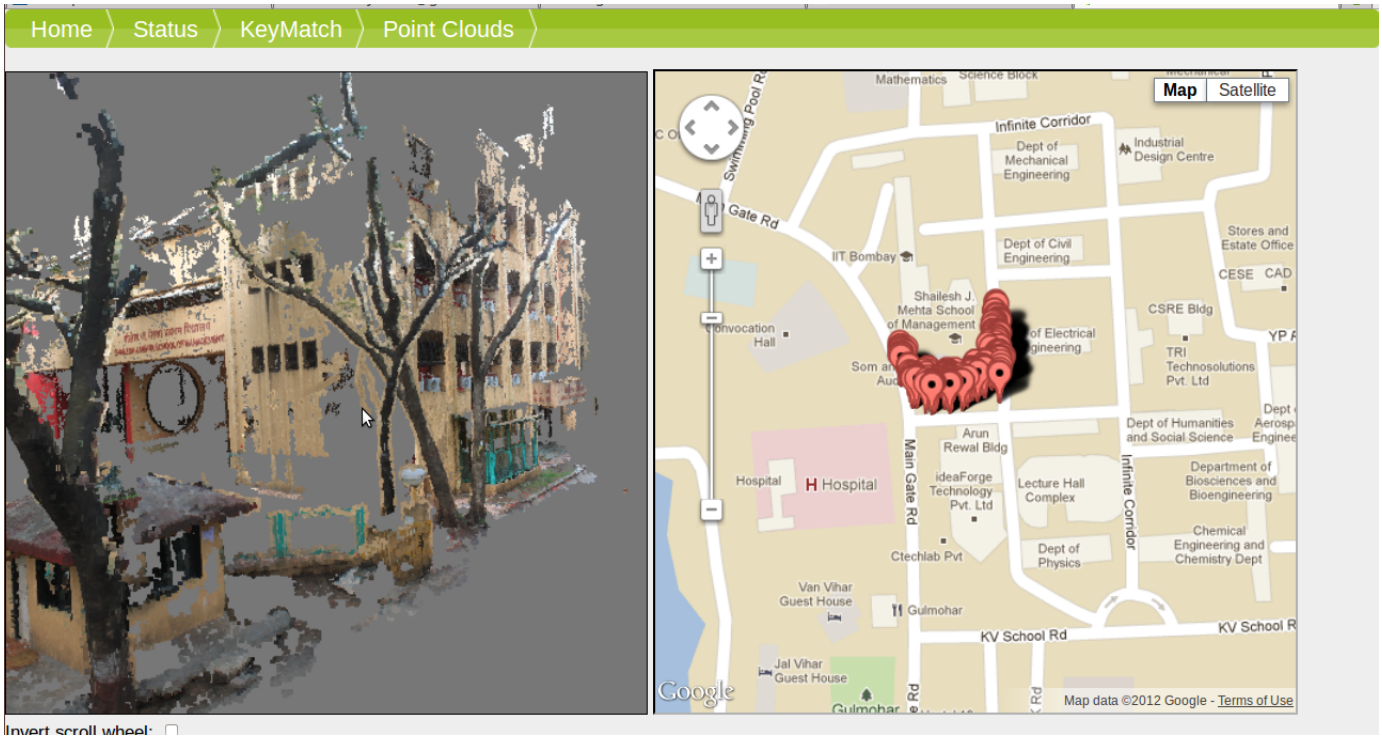


Figure 6.5: Shailesh J. Mehta School of Management, IIT Bombay Point Cloud and GPS information (345 images)

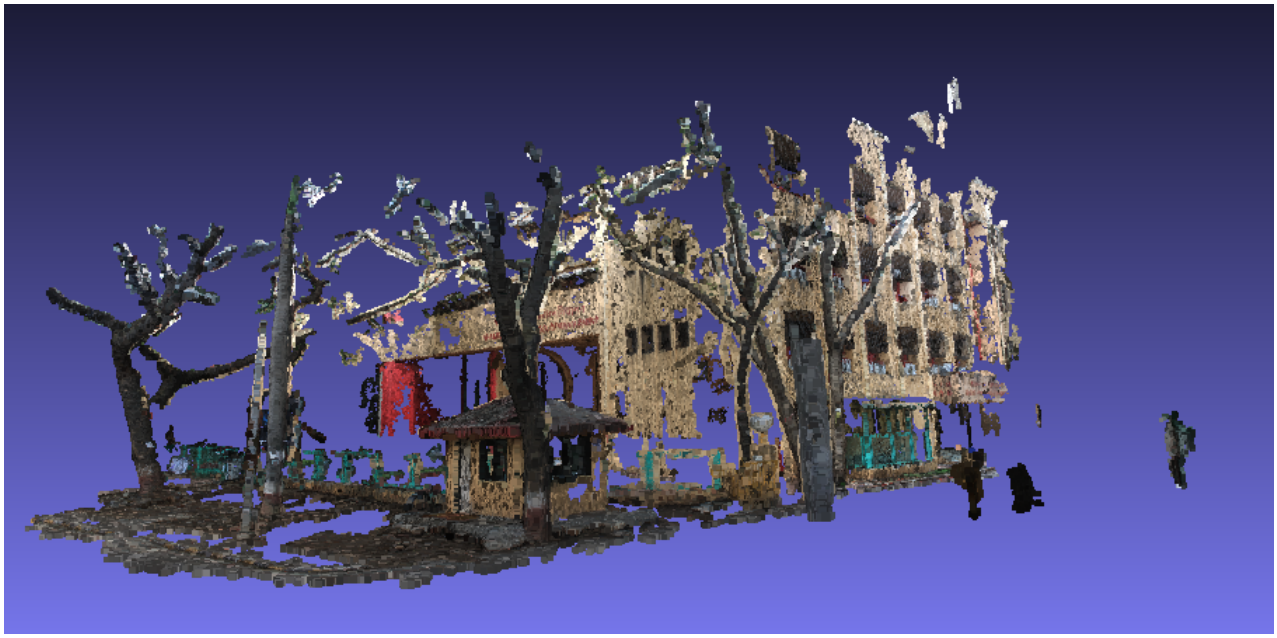


Figure 6.6: Shailesh J. Mehta School of Management, IIT Bombay(345 images)

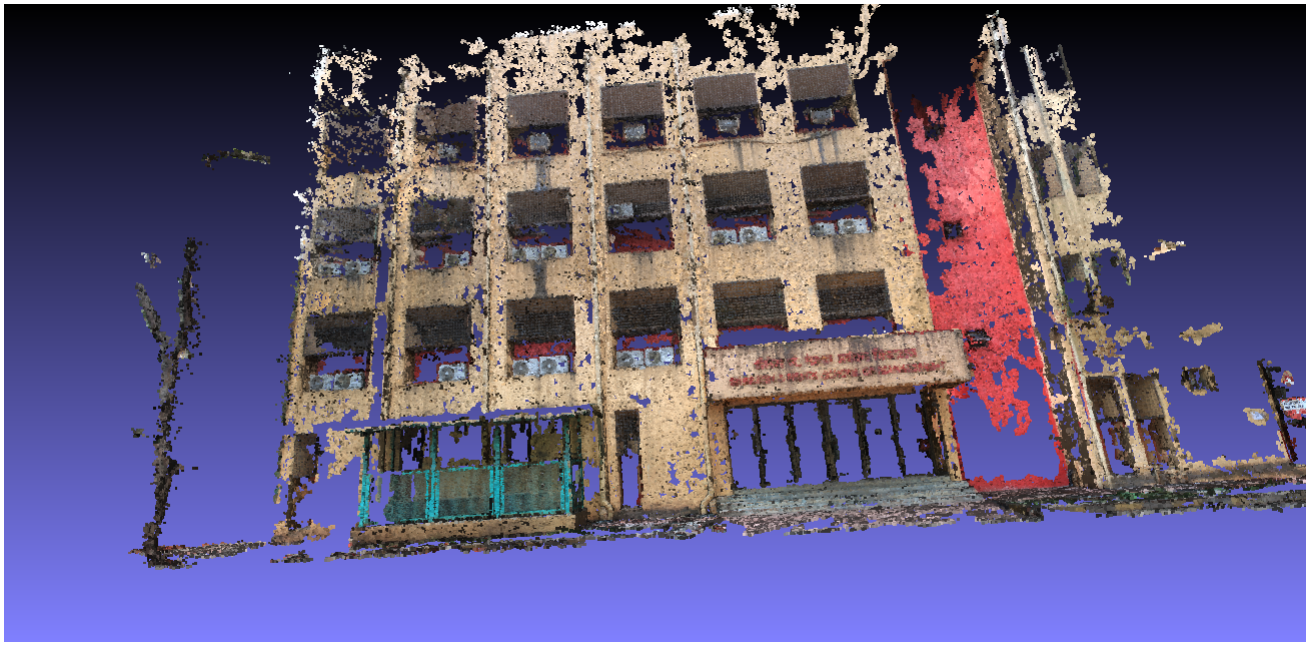


Figure 6.7: Shailesh J. Mehta School of Management, IIT Bombay(345 images)

The figure 6.9, 6.10, 6.11, 6.12 shows point-cloud generated for the datasets 6.8. *GPS* information of the various images in the *Physics Department* dataset is visualized in figure 6.14 :-

Figure 6.8: Physics Department, IIT Bombay sample images

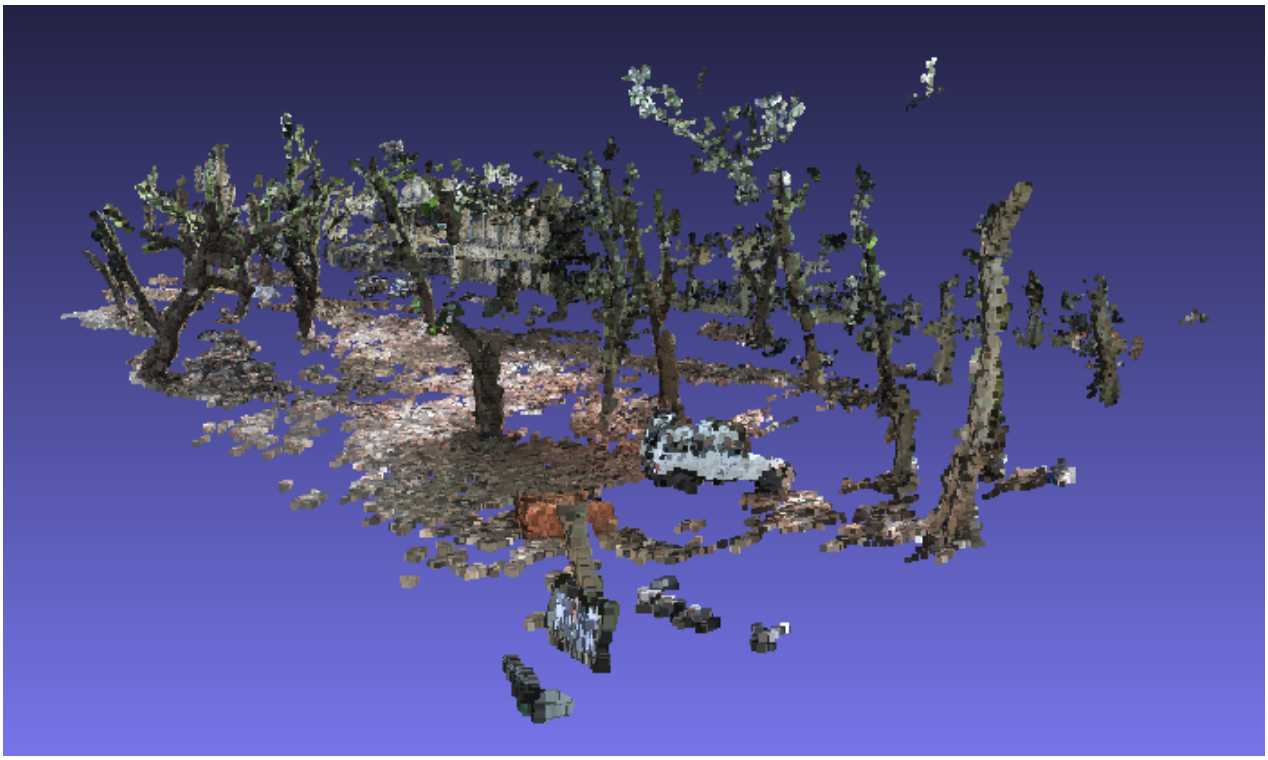


Figure 6.9: Physics Department(South Side) IIT Bombay, IIT Bombay(345 images)

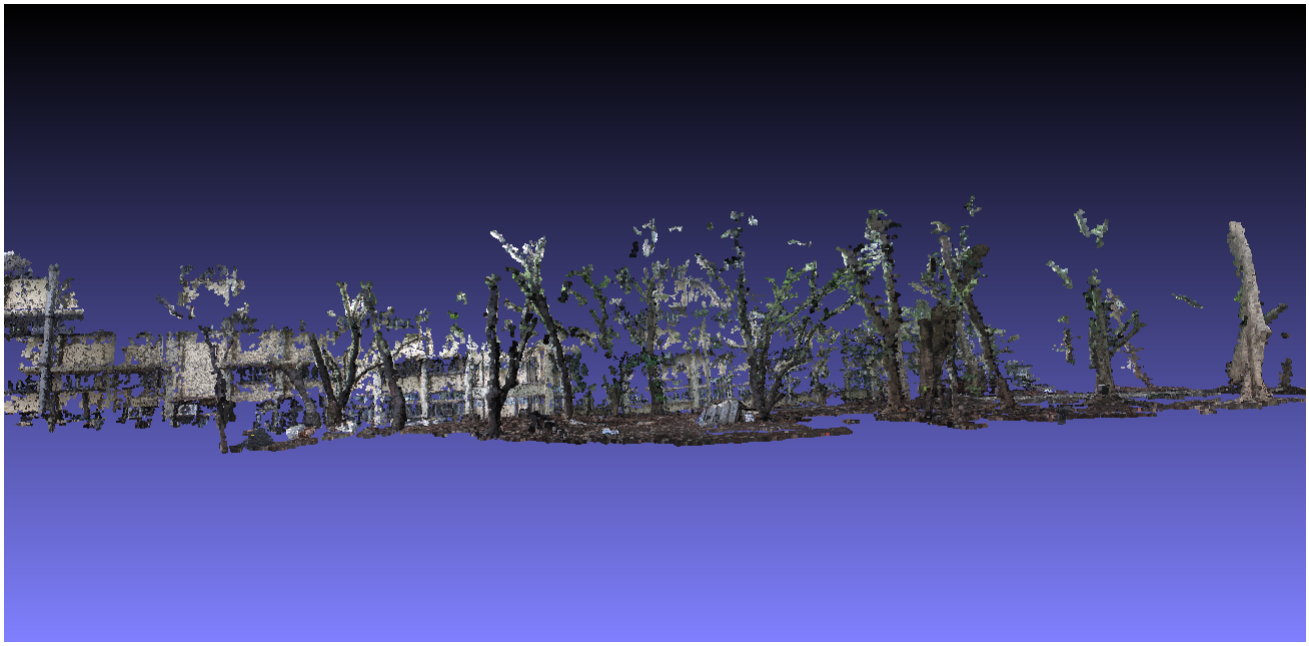


Figure 6.10: Physics Department(South Side) IIT Bombay, IIT Bombay(345 images)

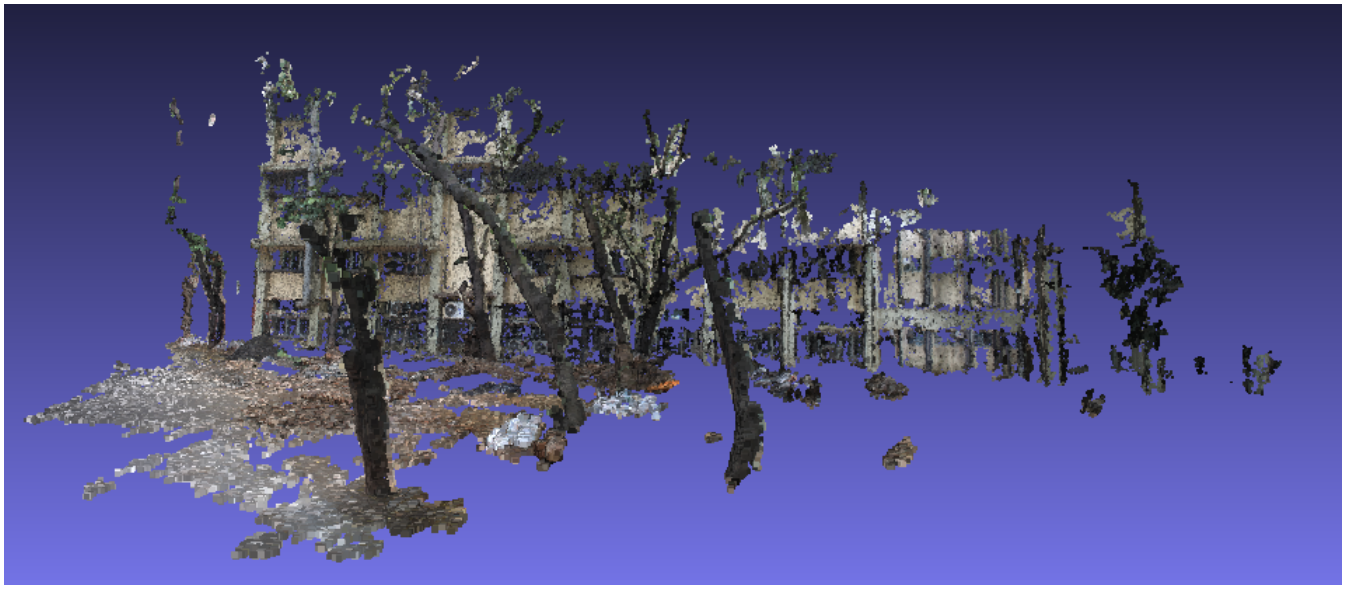


Figure 6.11: Physics Department(South Side) IIT Bombay, IIT Bombay(345 images)

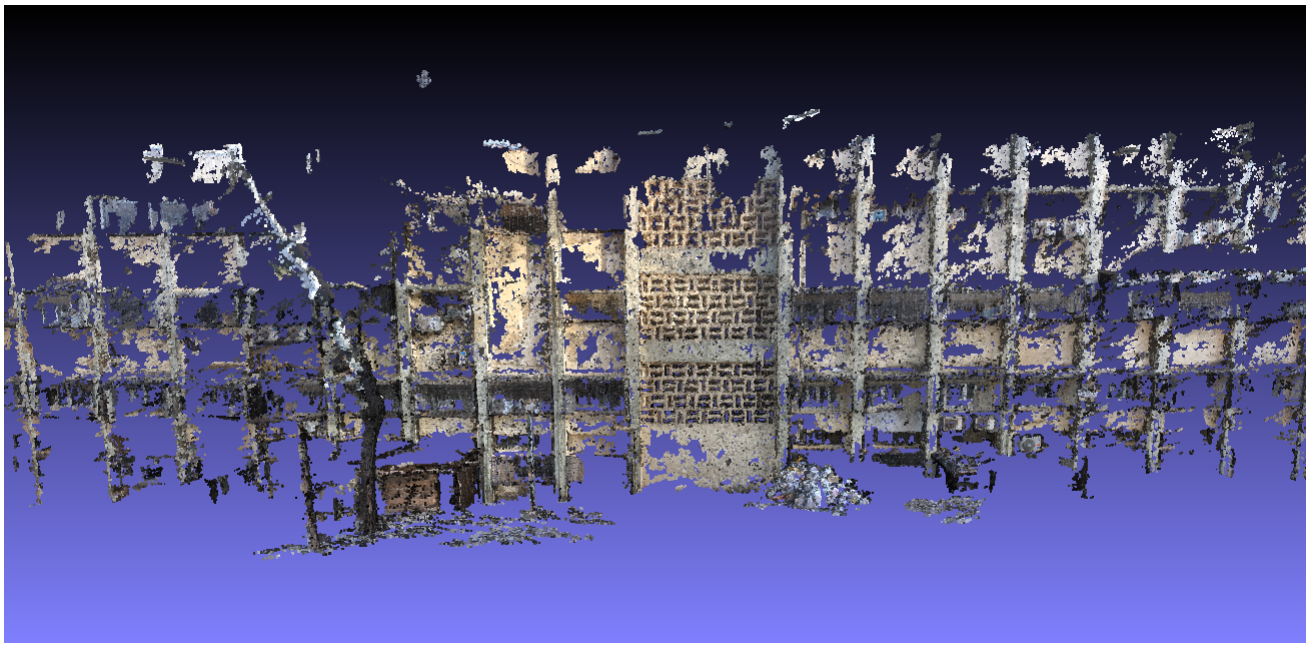


Figure 6.12: Physics Department(North Side) IIT Bombay, IIT Bombay(345 images)

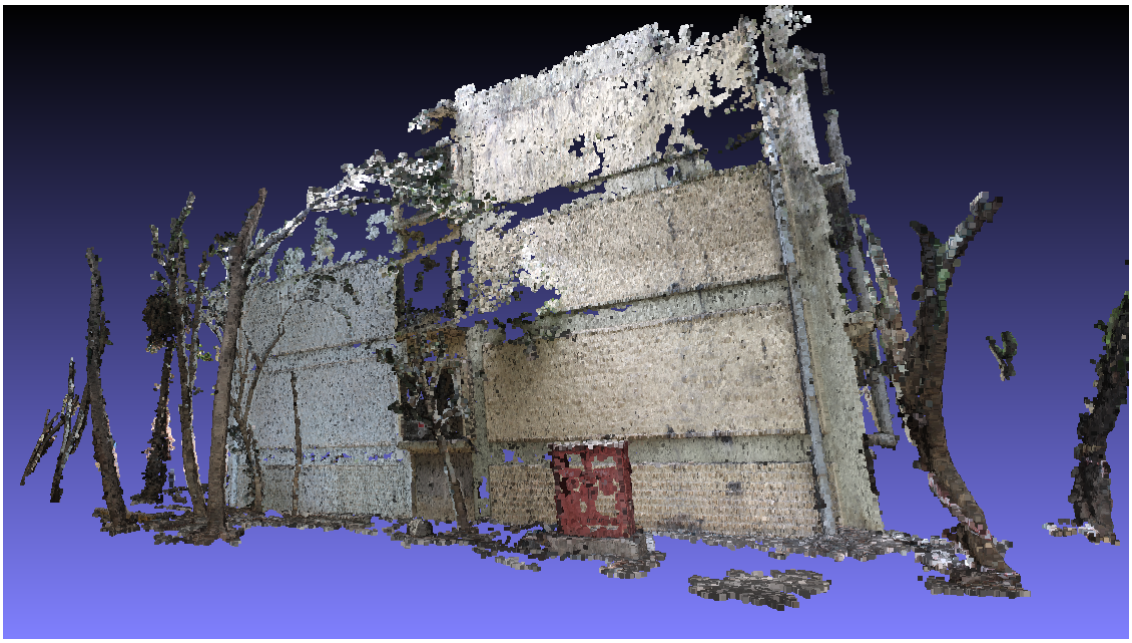


Figure 6.13: Physics Department(West Side) IIT Bombay, IIT Bombay(345 images)

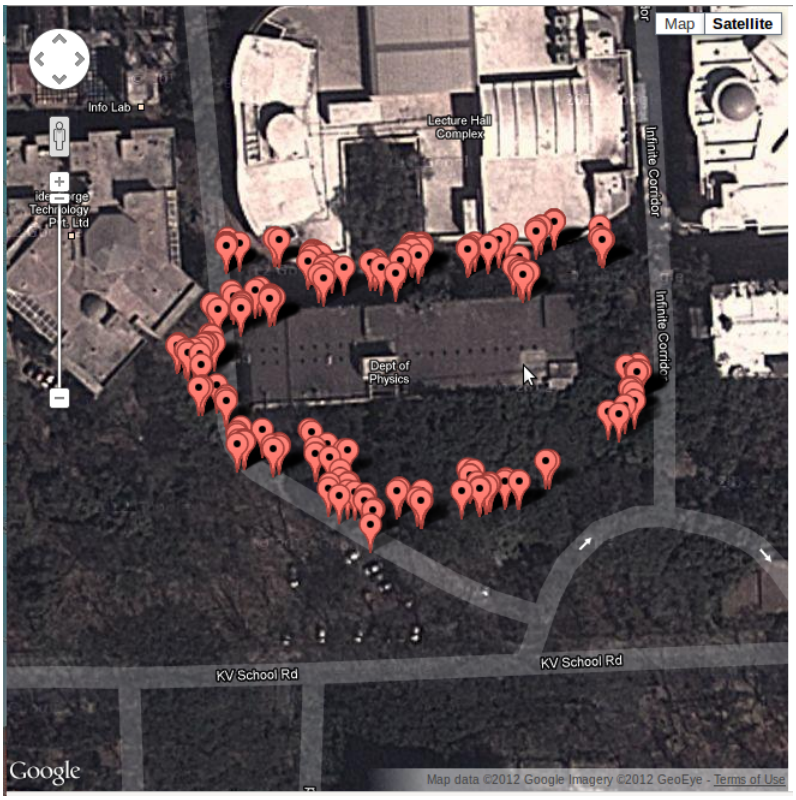


Figure 6.14: GPS positions of images taken around Physics Department IIT Bombay

Chapter 7

Conclusion and Future Work

7.1 Conclusion

In this report, we first discussed an approach of integrating data from varied sources like *GPS*, direction and map databases like *Wikimapia* containing building outline information and using this information to find which building is being imaged by a user. We have made a on-line portal for uploading images with *GPS* and direction information. Using the *GPS* and compass information we find which buildings are being viewed and cluster images according to the visible contour of the building. We then reconstruct the 3D model for the user and we provide an online interface for the user to monitor its progress as well as visualize the point cloud output. The website can be accessed inside IIT Bombay using the following URL http://10.129.1.123:8080/vision.

The various stages of the *Bundler* software is modified/replaced to improve its performance. Multi-Thread support is added to key-point/feature matching stage of the *Bundler* pipeline which results in three to four times performance increase. We tried to incorporate *GPS* constraints in the *Bundle Adjustment* problem but the results where not encouraging. The data collected using smart-phones of *GPS* co-ordinates are too noisy and are only accurate up-to a few meters, which are the primary reasons for the failure.

7.1.1 Contributions

In this thesis work, we make the following contributions:-

1. On-line portal where users can upload images and view 3D model The portal allows for users to track the status of the images submitted.
2. The geo-tagged images uploaded by the user along with compass direction is used to identify the building being viewed by the user and reconstruct the building
3. The building outline information is used to reduce the number of image key-point match.
4. The Key-point matching stage is re-written to use the latest *FLANN* library
5. Multi-threading support is added to increase the speed of matching
6. Support for incrementally adding images to an already reconstructed model

7.2 Future Work

Various improvements can be made to the existing system

- We could incorporate the accuracy of the measured GPS co-ordinate as diagonal elements of the co-variance matrix.
- The *GPS* data can be collected using *Differential GPS(D-GPS)* which as an accuracy of 10cm.
- The direction data can be incorporated in the *Bundle Adjustment* objective to get better reconstruction of the camera parameters.
- *GIST*[25] features can be used to cluster images which do not have *GPS* information.

Acknowledgments

I wish to acknowledge the help of many people who have directly or indirectly helped shape this thesis. I like to express my sincere gratitude and thanks to my guides *Prof. Parag Chaudhuri* and *Prof. J. Saketha Nath* for their constant encouragement and guidance. They has been my primary source of motivation and advice during my entire study. I have greatly benefited from the complementary guidance of both my advisors on this thesis.

I would like to thank all my friends in *IIT Bombay*. I would like to thank *Nimit Kalaria* for his expertise in system administration. I would like to acknowledge *Yogesh Kadke* and *Arindham Bhattacharya* for allowing me easy access to their smart-phones which was used to collect the *GPS* and direction data. I wish to express my love and gratitude to my beloved family for their understanding and endless love.

Bibliography

[1] S. Agarwal, N. Snavely, I. Simon, S.M. Seitz, and R. Szeliski. Building rome in a day. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 72–79. Ieee, 2009.

[2] D. Capel. An effective bail-out test for ransac consensus scoring. In *Proc. BMVC*, pages 629–638. Citeseer, 2005.

[3] O. Chum and J. Matas. Randomized ransac with td, d test. In *Proc. British Machine Vision Conference*, pages 448–457, 2002.

[4] O. Chum and J. Matas. Matching with prosac-progressive sample consensus. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 220–226. Ieee, 2005.

[5] O. Chum, J. Matas, and J. Kittler. Locally optimized ransac. *Pattern Recognition*, pages 236–243, 2003.

[6] A. Criminisi. Accurate visual metrology from single and multiple uncalibrated images. 1999.

[7] A. Criminisi, I. Reid, and A. Zisserman. Single view metrology. *International Journal of Computer Vision*, 40(2):123–148, 2000.

[8] M. Farenzena, A. Fusiello, and R. Gherardi. Structure-and-motion pipeline on a hierarchical cluster tree. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 1489–1496. IEEE, 2009.

[9] M.A. Fischler and R.C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[10] J.M. Frahm, P. Fite-Georgel, D. Gallup, T. Johnson, R. Raguram, C. Wu, Y.H. Jen, E. Dunn, B. Clipp, S. Lazebnik, et al. Building rome on a cloudless day. *Computer Vision–ECCV 2010*, pages 368–381, 2010.

[11] Y. Furukawa, B. Curless, S.M. Seitz, and R. Szeliski. Towards internet-scale multi-view stereo. 2010.

[12] R. Gherardi, M. Farenzena, and A. Fusiello. Improving the efficiency of hierarchical structure-and-motion. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 1594–1600. IEEE, 2010.

[13] R. Hartley and A. Zisserman. *Multiple view geometry*, volume 6. Cambridge university press, 2000.

[14] R.I. Hartley and P. Sturm. Triangulation. *Computer vision and image understanding*, 68(2):146–157, 1997.

[15] L. Kaufman and P.J. Rousseeuw. *Finding groups in data: an introduction to cluster analysis*, volume 39. Wiley Online Library, 1990.

[16] M. Lhuillier. Fusion of gps and structure-from-motion using constrained bundle adjustments. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3025–3032. IEEE, 2011.

[17] M. Lourakis and A. Argyros. The design and implementation of a generic sparse bundle adjustment software package based on the levenberg-marquardt algorithm. *Institute of Computer Science-FORTH, Heraklion, Crete, Greece, Tech. Rep*, 340, 2004.

[18] D.G. Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[19] J. Matas and O. Chum. Randomized ransac with sequential probability ratio test. 2005.

[20] J. More. The levenberg-marquardt algorithm: implementation and theory. *Numerical analysis*, pages 105–116, 1978.

[21] M. Muja and D.G. Lowe. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Applications (VISSAPP09)*, pages 331–340, 2009.

[22] P. Müller, G. Zeng, P. Wonka, and L. Van Gool. Image-based procedural modeling of facades. *ACM Transactions on Graphics*, 26(3):85, 2007.

[23] D. Nistér. An efficient solution to the five-point relative pose problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(6):756–770, 2004.

[24] D. Nister and H. Stewenius. Scalable recognition with a vocabulary tree. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2161–2168. IEEE, 2006.

[25] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *International Journal of Computer Vision*, 42(3):145–175, 2001.

[26] R. Raguram, J.M. Frahm, and M. Pollefeys. A comparative analysis of ransac techniques leading to adaptive real-time random sample consensus. *Computer Vision–ECCV 2008*, pages 500–513, 2008.

[27] R. Roberts, S.N. Sinha, R. Szeliski, and D. Steedly. Structure from motion for scenes with large duplicate structures. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 3137–3144. IEEE, 2011.

[28] A. Saxena, S.H. Chung, and A. Ng. Learning depth from single monocular images. *Advances in Neural Information Processing Systems*, 18:1161, 2006.

[29] A. Saxena, S.H. Chung, and A.Y. Ng. 3-d depth reconstruction from a single still image. *International Journal of Computer Vision*, 76(1):53–69, 2008.

[30] A. Saxena, M. Sun, and A.Y. Ng. Make3d: Learning 3d scene structure from a single still image. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 31(5):824–840, 2009.

[31] F. Schaffalitzky and A. Zisserman. Planar grouping for automatic detection of vanishing lines and points. *Image and Vision Computing*, 18(9):647–658, 2000.

[32] S.N. Sinha, D. Steedly, and R. Szeliski. A multi-stage linear approach to structure from motion. In *ECCV 2010 Workshop on Reconstruction and Modeling of Large-Scale 3D Virtual Environments*, volume 3002, pages 3003–3005, 2010.

[33] N. Snavely, S.M. Seitz, and R. Szeliski. Photo tourism: exploring photo collections in 3d. In *ACM Transactions on Graphics (TOG)*, volume 25, pages 835–846. ACM, 2006.

[34] N. Snavely, S.M. Seitz, and R. Szeliski. Modeling the world from internet photo collections. *International Journal of Computer Vision*, 80(2):189–210, 2008.

[35] H. Stewénius, C. Engels, and D. Nistér. Recent developments on direct relative orientation. *ISPRS Journal of Photogrammetry and Remote Sensing*, 60(4):284–294, 2006.

[36] PHS Torr. An assessment of information criteria for motion model selection. In *Computer Vision and Pattern Recognition, 1997. Proceedings., 1997 IEEE Computer Society Conference on*, pages 47–52. IEEE, 1997.

[37] P.H.S. Torr and A. Zisserman. Mlesac: A new robust estimator with application to estimating image geometry. *Computer Vision and Image Understanding*, 78(1):138–156, 2000.

[38] B. Triggs, P. McLauchlan, R. Hartley, and A. Fitzgibbon. Bundle adjustmenta modern synthesis. *Vision algorithms: theory and practice*, pages 153–177, 2000.