2015

POSE ESTIMATION AND 3D RECONSTRUCTION USING SENSOR FUSION

Anuj S. Potnis
Michigan Technological University

POSE ESTIMATION AND 3D RECONSTRUCTION USING SENSOR FUSION

By

Anuj S. Potnis

A REPORT

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

In Electrical Engineering

MICHIGAN TECHNOLOGICAL UNIVERSITY

2015

This report has been approved in partial fulfillment of the requirements for the Degree of MASTER OF SCIENCE in Electrical Engineering.

Department of Electrical & Computer Engineering

Report Advisor:	*Dr. Timothy C. Havens*
Committee Member:	*Dr. Michael C. Roggemann*
Committee Member:	*Dr. Gowtham*
Department Chair:	*Dr. Daniel R. Fuhrmann*

Dedication

To my parents, teachers and friends

Contents

List of Figures

List of Tables

Acknowledgments

First and foremost, I thank my advisor Dr. Timothy Havens for coming up with an interesting research idea, and allowing me to work on it at the Intelligent Robotics Laboratory (IRL). I am indebted to him for his support and input throughout.

Special thanks to Dr. Gowtham, for his advice on Linux OS, which was integral to the functioning of this project. I would like to thank Dr. Michael C. Roggerman for being a part of my committee and offering advice.

Dr. Yushin Ahn had valuable inputs for me in the field of computer vision. Dr. Scott Kuhl trained me on using the Immersive Visual Studio, which I used for collecting data for this report. Pierre Moulon, author of OpenMVG library for computer vision has always promptly replied to every question I had.

Pranav Bhatkhande, introduced me to the wonderful environs of the IRL and motivated me at every step. I thank Dereck Wonnacott, Joshua Manela and Tim Bradt for making the data acquisition hardware which made data collection simple. Special thanks to Joshua Manela for allowing me to use the results of his LIDAR algorithm for comparison. I also thank the rest of my research team consisting of Husam Sweidan and Hanieh Deilamsalehy for their support in the field of computer vision.

My family and friends, have always been very supportive and I cannot thank them enough.

Abstract

A camera maps 3-dimensional (3D) world space to a 2-dimensional (2D) image space. In the process it loses the depth information, i.e., the distance from the camera focal point to the imaged objects. It is impossible to recover this information from a single image. However, by using two or more images from different viewing angles this information can be recovered, which in turn can be used to obtain the pose (position and orientation) of the camera. Using this pose, a 3D reconstruction of imaged objects in the world can be computed. Numerous algorithms have been proposed and implemented to solve the above problem; these algorithms are commonly called Structure from Motion (SfM). State-of-the-art SfM techniques have been shown to give promising results. However, unlike a Global Positioning System (GPS) or an Inertial Measurement Unit (IMU) which directly give the position and orientation respectively, the camera system estimates it after implementing SfM as mentioned above. This makes the pose obtained from a camera highly sensitive to the images captured and other effects, such as low lighting conditions, poor focus or improper viewing angles. In some applications, for example, an Unmanned Aerial Vehicle (UAV) inspecting a bridge or a robot mapping an environment using Simultaneous Localization and Mapping (SLAM), it is often difficult to capture images with ideal conditions. This report examines the use of SfM methods in such applications and the role of combining multiple sensors, viz., sensor fusion, to achieve more accurate

and usable position and reconstruction information.

This project investigates the role of sensor fusion in accurately estimating the pose of a camera for the application of 3D reconstruction of a scene. The first set of experiments is conducted in a motion capture room. These results are assumed as ground truth in order to evaluate the strengths and weaknesses of each sensor and to map their coordinate systems. Then a number of scenarios are targeted where SfM fails. The pose estimates obtained from SfM are replaced by those obtained from other sensors and the 3D reconstruction is completed. Quantitative and qualitative comparisons are made between the 3D reconstruction obtained by using only a camera versus that obtained by using the camera along with a LIDAR and/or an IMU. Additionally, the project also works towards the performance issue faced while handling large data sets of high-resolution images by implementing the system on the Superior high performance computing cluster at Michigan Technological University.

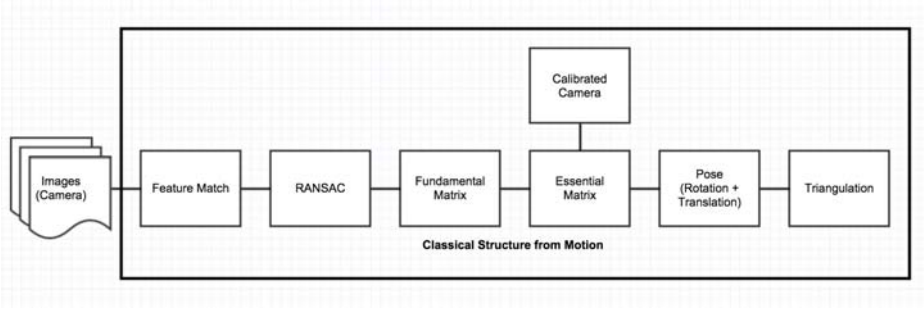
Chapter 1

Introduction

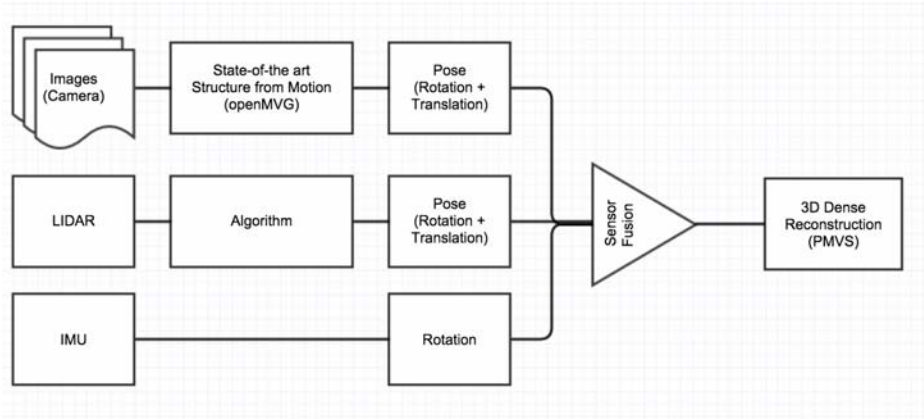
1.1 Motivation

The Intelligent Robotics Laboratory (IRL) at Michigan Technological University is working towards accurately estimating the pose (position and orientation) of an Unmanned Aerial Vehicle (UAV) which would help create a 3D reconstruction of the surroundings. Applications range from inspection of infrastructure to mapping of unknown terrain using Simultaneous Localization and Mapping (SLAM).

The sensors used by the UAV are camera, LIDAR, Inertial Measurement Unit (IMU) and GPS. While the GPS and IMU give position and orientation directly, the camera and LIDAR do not. This project focuses on obtaining the pose of a single camera



(a) Classical Structure from Motion



(b) Sensor Fusion

Figure 1.1: Figure 1.1(a) shows the first part of the project wherein we implement an algorithm for Structure from Motion. Figure 1.1(b) depicts the second part wherein we show an overview of a sensor fusion system.

and hence the UAV using Structure from Motion (SfM).

1.2 Report Overview

Chapter 2 explains the theory behind structure from motion using the concepts of multiple view geometry. It talks about the classical SfM pipeline and describes some robust techniques to reduce outliers. A basic model of a structure from motion is

implemented. We go step-by-step into explaining the model and its implementation as shown in Figure 1.1(a). In the remaining Chapters, we propose a sensor fusion model as shown in Figure 1.1(b).

Chapter 3 gives an overview of the state-of-the-art packages for structure from motion and dense 3D reconstruction. The rest of the chapter is focused on one such package called openMVG [1] by Pierre Moulon. A detailed discussion of the algorithms published in [2, 3] forms the basis of the chapter.

Chapter 4 talks in detail about the implementation and computing issues faced while running openMVG for large image datasets. It compares different computing platforms and explains the implementation of the High Performance Cluster - Superior at Michigan Technological University.

Chapter 5 describes the different sensors used, their strengths and weaknesses in obtaining data and their coordinate systems. The Immersive Visualization Studio (IVS), a motion capture room at Michigan Technological University, is described. It was used to obtain an extremely accurate pose and was assumed ground truth in the experiments. The process of data collection is documented here.

Chapter 6 compares the strengths and weaknesses of different sensors on three different datasets. It describes the algorithm used to combine data obtained from the different sensors. Fusion of data from an Inertial Measurement Unit with the output

from openMVG has resulted in an improvement in 3D reconstruction.

Chapter 2

Structure from Motion (SfM)

2.1 Chapter Goals

This chapter introduces the concept of SfM theoretically and mathematically. SfM is a process of reconstructing the structure of an object from a sequence of images taken from views separated by an optimum baseline distance. SfM simultaneously estimates motion of the camera and the structure of the object. It involves two major steps viz., pose estimation and triangulation.

2.2 Pose Estimation

2.2.1 Camera Calibration

Camera calibration is a process of finding intrinsic parameters of the camera. These include the lens distortion coefficients, focal length, principal point and skew. Out of these, the lens distortion coefficient is used to eliminate distortion of the image using Equation 2.1. The effect of undistorting the images using the lens equations can be seen in Figure 2.1(a) and 2.1(b). Other parameters form the intrinsic camera calibration matrix as given in Equation (2.3).

$$\begin{aligned} x_{dist} &= x(1 + k_1 r^2 + k_2 r^4) \\ y_{dist} &= y(1 + k_1 r^2 + k_2 r^4) \\ x &= \text{undistorted pixel} \\ y &= \text{undistorted pixel} \\ k_1, k_2 &= \text{radial distortion coefficients} \\ r^2 &= x^2 + y^2 \end{aligned} \tag{2.1}$$

(a) Radial Barrel distortion removal of door and wall in a building

(b) Radial Barrel distortion removal of ladder in Motion Capture room

Figure 2.1: The left halves of Figure 2.1(a) and 2.1(b) show the radial Barrel distortion introduced by Point Grey camera. Using the distortion coefficients obtained from camera calibration, the distortion is removed as seen in the right halves.

$$\begin{aligned} \mathbf{x} &= P \cdot \mathbf{X} \\ P_{3\times4} &= \text{camera matrix} \\ \mathbf{X}_{4\times1} &= \text{homogeneous 4-vector [x y z 1]}' \text{ world point} \\ \mathbf{x}_{3\times1} &= \text{homogeneous 3-vector [x y 1]}' \text{ image point} \end{aligned} \tag{2.2}$$

$$K = \begin{bmatrix} \alpha_x & s & x_0 \\ & \alpha_y & y_0 \\ & & 1 \end{bmatrix}$$

$$\alpha_x = f \cdot m_x \rightarrow \text{focal length in pixels}$$

$$\alpha_y = f \cdot m_y \rightarrow \text{focal length in pixels}$$

$$f = \text{focal length in mm}$$

$$m_x = \text{number of pixels per mm along x direction} \tag{2.3}$$

$$m_y = \text{number of pixels per mm along y direction}$$

$$x_o = p_x \cdot m_x \rightarrow \text{principal point in pixels}$$

$$y_o = p_y \cdot m_y \rightarrow \text{principal point in pixels}$$

$$p_x = \text{pixel dimension along x diection}$$

$$p_y = \text{pixel dimension along y diection}$$

$$s = skew$$

$$
\begin{aligned}
P &= K[R|t] \\
K_{3\times3} &= \text{camera intrinsic matrix (Equation (2.3))} \\
R_{3\times3} &= \text{orientation of camera coordinate frame} \\
t_{3\times1} &= \text{position of camera coordinate frame}
\end{aligned}
\tag{2.4}
$$

$$
\tilde{C} = -R^{\intercal}t \tag{2.5}
$$

A camera as given in Equation (2.2) is defined as a matrix $P_{3\times4}$ which maps a 3-space point $\mathbf{X}$ to image point $\mathbf{x}$. We begin by assuming that the camera model we are using is a basic pinhole camera. Then we generalize it to a CCD camera model in which pixel dimensions are different along the x and y directions. For this we consider different focal lengths (α_x, α_y) and principal points $(x_o,\ y_o)$ along x and y directions. In most of the cases the CCD camera model generalization is sufficient. However, after we calibrated the Point Grey it was found that it had a non-zero skew parameter as well. Therefore we further generalize it to a category known as finite projective camera. Mathematically it implies that there is a need to include the values of these parameters in the camera calibration (intrinsic) matrix K, which is related to the camera matrix P as given in Equation (2.4).

Camera calibration is carried out by imaging world coordinates of known metric dimensions. By this we calculate the value of this mapping. Therefore when we use the same camera for observing unknown world points we are able to estimate the Euclidean geometry in the image. For a more detailed procedure on camera calibration the reader may refer to the standard checker calibration procedure which can be implemented using [4].

Table 2.1
Intrinsic camera parameters of Point Grey camera obtained after calibration

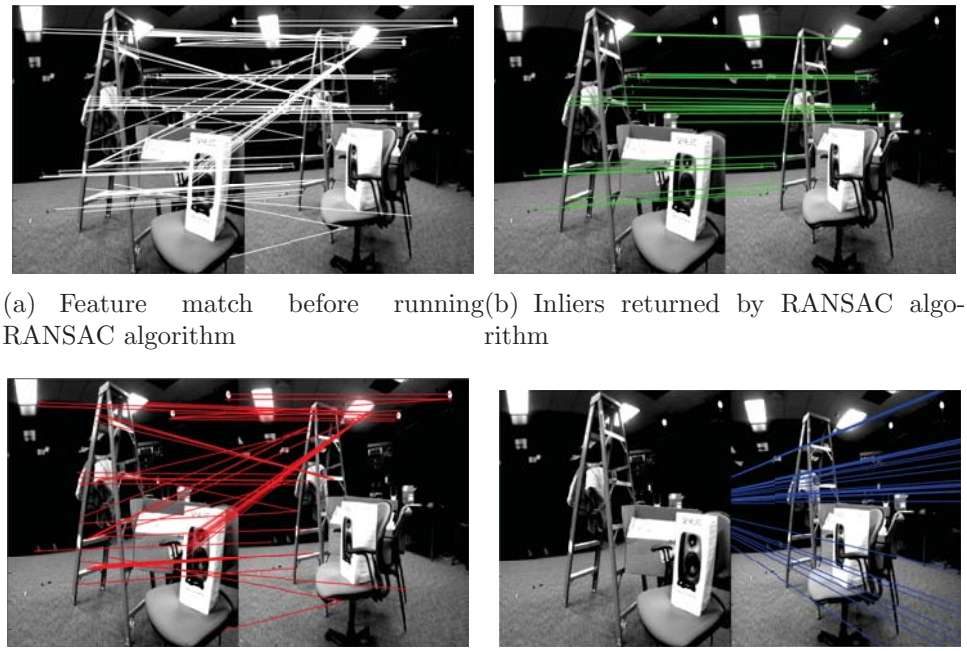
Camera Parameters	**Values (in pixels)**
Radial lens distortion coefficients [k1 k2]	[-0.34608 0.13639]
Focal length $[\alpha_x\ \alpha_y]$	[884.39 884.86]
Principal point $[x_o\ y_o]$	[619.53 485.87]
Skew	0.7857

Table 2.1 lists the camera parameters of Point Grey after calibrating it using the standard checkerboard procedure. Therefore, the K matrix is obtained as:

$$K = \begin{bmatrix} 884.39 & 0.7857 & 619.53 \\ & 884.86 & 485.87 \\ & & 1 \end{bmatrix}$$

2.2.2 Feature Matching

Images taken of outdoor objects like statues or buildings had large variations in intensity and rotation. After comparing the SIFT and SURF feature extractors and descriptors, SIFT was found to be more robust to changing lighting and random rotational shifts. SIFT also robustly matched repeating patterns, for example, bricks of a building with fewer outliers. [5] implementation of the SIFT was used. Figure 2.2(a)



(a) Feature match before running RANSAC algorithm

(b) Inliers returned by RANSAC algorithm

(c) Outliers returned by RANSAC algorithm

(d) Family of epilines converging towards position of first camera

Figure 2.2: Figure 2.2(a) shows the result of feature matching before the application of RANSAC algorithm. Figure 2.2(b) shows the inliers after application of RANSAC algorithms and Figure 2.2(c) shows only the outliers. Figure 2.2(d) shows a family of epipolar lines. These lines are functions of the first camera center. If the epipolar lines were extended they would converge at the center of the first camera

shows the result after feature matching. Despite robust performance we see many outliers that can have a negative impact on the geometry of images. The algorithm to remove the outliers is explained in Section 2.2.3.

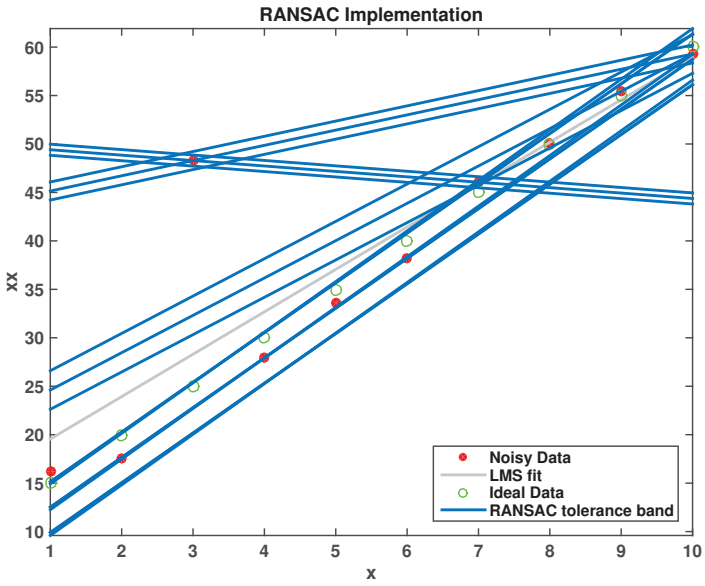


Figure 2.3: The LMS fitted lines (blue) are shown at every iteration of the RANSAC algorithm. We see that within a few iterations the algorithm converges and the blue line fits along the true data while ignoring the outlier.

2.2.3 RANdom SAmple Consensus (RANSAC)

The RANSAC [6] algorithm randomly samples the data and based on a voting metric comes to a consensus on the model parameters. The features which match after implementing RANSAC are shown in Figure 2.2(b) and the outliers in 2.2(c). It can be seen that over 50% outliers have been removed. To give an understanding of how RANSAC works we will use the simplified algorithm as given in [7] and demonstrate the model parameter selection where the model is a line. We see in Figure 2.3 a set of noisy points with ideal data in the background. We also see an outlier. Then we fit a line (shown in blue) such that the Least Mean Square error is minimum. There is

a shift in the resulting line - away from the ideal points and towards the outlier. The RANSAC method is seen to converge after a few iterations to accommodate only the inliers.

2.2.4 Epipolar Geometry

2.2.4.1 Fundamental Matrix (F)

If a point X in 3D space is observed from a camera from two different views such that x and $x^{'}$ are the projections of X on their respective image planes, the geometry of the two views is governed by epipolar geometry and is completely captured by a 3×3 matrix, called a Fundamental Matrix (F matrix) given by Equation (2.6).

$$\begin{aligned} x'^{\mathsf{T}} F x &= 0 \\ F_{3\times 3} &= \text{fundamental matrix} \\ x &= \text{3-space } \mathbf{X} \text{ imaged from first view} \\ x' &= \text{3-space } \mathbf{X} \text{ imaged from second view} \end{aligned} \tag{2.6}$$

In Figure 2.4 we see that the point X is projected as x when from viewed from the camera at pose 1. When camera moves to pose 2 the same point X is projected as point x' on the image plane. The position of this projection x' is restricted along a

single line known as the epiline. However since in our application, we already know the position of x' as given by feature matching, and using this, we can calculate the new pose. In Figure 2.2(d) we can see a family of epilines corresponding to the feature points in Figure 1. Also for more than one point we get a family of epipolar lines and they all pass through the epipole. The epipole is nothing but the projection of the first camera center on the second image. If the epilines in the images were to be extended such that they converge to a point, that location would be the location of the first camera center. Had the image been taken such that the camera center during the first take is visible in the second, we could have seen the epipole in the second image.

The F matrix has 9 elements and has rank 2. It has 8 independent ratios (with scale uncertainty) and an additional constraint of $\det F = 0$. This gives it 7 degrees of freedom. Thus we can calculate F matrix by knowing only 7 corresponding points. However by adding another pair of points we obtain 8 linear equations.

As given in [7] we have the 8 - point algorithm for computing F : For a pair of points $(x, y, 1)$ and $(x', y', 1)$ we have

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

which can we expressed as (2.7)

$$(x'x, x'y, x', y'x, y'y, y', x, y, 1) \cdot f = 0$$

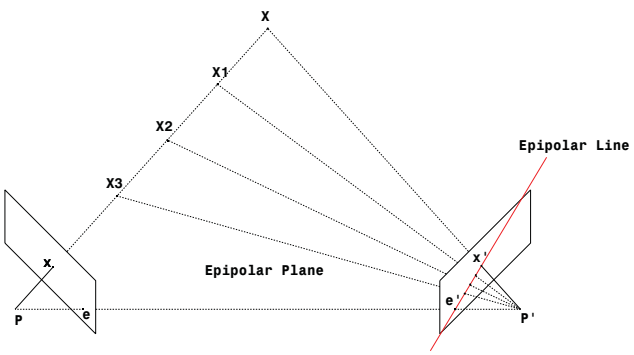


Figure 2.4: The epipolar geometry is shown in the above figure. The 3-space point X is projected as point x by the first camera in the first view. The projection of point x is constrained by the epiline in the view from the second camera. i.e. the projection of X could lie anywhere along the epiline when viewed from the second camera.

Thus for n points we have $A \cdot f = 0$

Solving for the null space of A, we get the solution to the equation. In a practical scenario due to noise, the rank is almost never 2. Also we have more than 8 points making it an overdetermined solution. In such a case we find the least square solution. This solution is the singular vector corresponding to the smallest singular value of the SVD of A i.e. last column of V in $SVD\ A = UDV^{\intercal}$. We thus obtain F but we need to ensure that its rank is 2 for it to be a Fundamental Matrix. This is done by finding F' such that the Frobenius norm $||F - F'||$ is minimized subject to $\det F = 0$. Now that we have the F matrix, we can decompose it to determine the pose i.e., R and t. However the reconstruction obtained from this would have a projective ambiguity. In order to obtain a metric transform (a true Euclidean reconstruction would require additional information like GPS or some known dimensions in world units) we need exploit the intrinsic parameters of the camera i.e., use a calibrated camera.

2.2.4.2 Essential Matrix (E)

We assume that at all times the camera will be calibrated and hence we will have the intrinsic calibration parameters. Therefore we can exploit the properties of the E matrix which is given as Equation 2.8, in obtaining the pose.

$$\begin{aligned} E &= K'^{\intercal} F K \\ K', K &= \text{camera intrinsic matrix (Equation (2.3))} \\ F &= \text{fundamental matrix (Equation (2.6))} \end{aligned} \tag{2.8}$$

The essential matrix on decomposition gives rise to 2 solutions. However both E and -E are results of the same epipolar geometry and hence we have a total of 4 solutions. Mathematically [7] gives the 4 solutions as shown in Equation (2.9). For essential matrix $E = Udiag(1,1,0)V^{\intercal}$ and first camera matrix $P = [I|0]$, we have second camera matrix as:

$$\begin{aligned} P' &= [UWV^{\intercal}| + u_3] \quad \text{or} \\ &= [UWV^{\intercal}| - u_3] \quad \text{or} \\ &= [UW^{\intercal}V^{\intercal}| + u_3] \quad \text{or} \\ &= [UW^{\intercal}V^{\intercal}| - u_3] \end{aligned} \tag{2.9}$$

The correct solution is determined by the fact that the test point is in view of both the cameras. As seen in Figure 2.5, the black camera is the first camera and the blue camera is the true second camera, and both cameras view the magenta colored test point. In the process of estimating pose of the second camera, we obtain 4 solutions (or poses). The test point is in view of only the green camera. The three remaining solutions (red cameras) are not correct. The incorrect solutions are incorrect either because the camera has translated to the other side of the first camera or it has been reversed by 180° about the line joining the centers of the first and second cameras. Note: the blue and green cameras are offset by a translation because the E matrix can be decomposed only upto a scale.

2.3 Triangulation

At this point we have the initially obtained camera poses and the feature points. We back project these points as viewed from both the cameras and allow them to intersect. Due to the noise and inaccuracy in the estimation of the poses the rays will almost always never meet. We set an error threshold for which we consider them as intersecting rays.

We see in Figure 2.6 the results of triangulation. Each of the points is the original feature point detected. This is known as sparse reconstruction since not every pixel is taken into consideration.

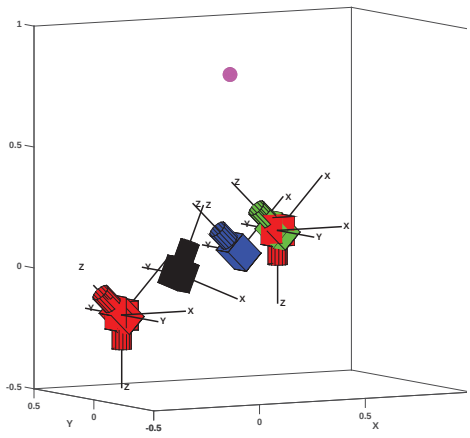
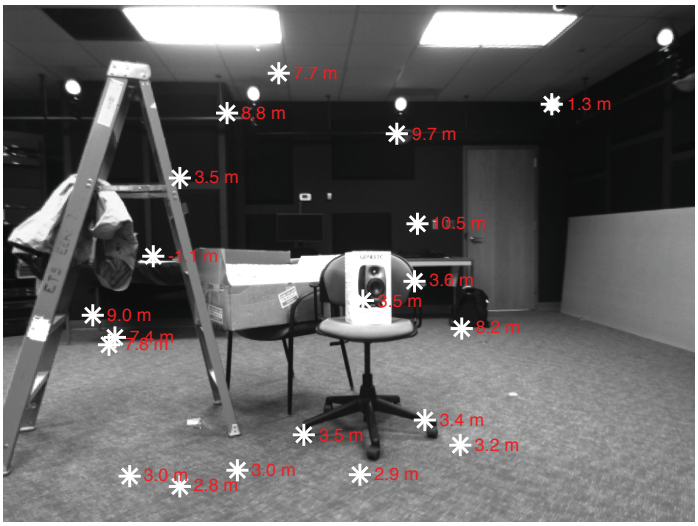


Figure 2.5: The black camera is the first camera. The blue camera is the second camera (true value). Both view the magenta colored test point. In the process of estimating the pose of the second camera, we obtain 4 solutions. The magenta test point is in front of the green camera only (in addition to the black camera which could previously view it too). It cannot be viewed from the red cameras either because they have translated to the other side of the black camera or have been rotated by 180° about the baseline connecting the black camera to itself. Note that the blue and green cameras are offset by a translation because the E matrix can be decomposed only upto a scale.

2.3.1 Scaled Translation and need for Sensor Fusion

One important point to note here is that one of the ground truth values was physically measured. For example in Figure 2.6(b) this is the distance from the camera to the box kept on the chair. This is due to the fact that although we can obtain a Euclidean metric reconstruction, it is true only upto scale. This relates to the F matrix and the E matrix being transforms only upto scale. However once we know one single point



(a) Building door and wall

(b) Motion Capture room

Figure 2.6: The images show the result of triangulation. Feature points are marked with the estimated depth (in meters) with respect to the camera. We see that for most of the points the relative depth metric conforms with the image. Note: One of the points had to be measured physically in order to get an absolute value of the estimated translation vector.

the rest of the points can be scaled accordingly.

2.4 Chapter Conclusion

In this chapter, we implemented a classical model of SfM. We can now see the importance of camera calibration on the structure of the 3D reconstruction. In addition to having robust feature extractors and detectors like SIFT it is necessary to have robust algorithms which can remove outliers. We implement SfM at the block level in order to obtain insight to the process and to get an understanding of the factors which impact the reconstruction. For example, the fact that SIFT is invariant to intensity means that we do not need to perform any image enhancement techniques to get a better structure.

Chapter 3

SfM and Dense 3D Reconstruction Pipelines

3.1 Chapter Goals

In this chapter we give an overview of these state-of-the-art packages for SfM pipelines. The classical SfM algorithm, when applied to a sequence of images, accumulates the error for every image. Moreover the translation we obtain is a scaled value. These problems have been solved in [2, 3, 8, 9, 10]. Some of this literature has been implemented in the form of state-of-the-art packages. For a more analytic comparison please refer to [11, 12, 13].

3.2 State-of-the-art SfM Pipelines

3.2.1 VisualSfM

VisualSfM [10] is one of the most popular SfM pipelines. It implements the Incremental SfM pipeline [9] with $O(n)$ time complexity (as compared to the $O(n^4)$ and $O(n^3)$ previously achieved) for most of its major processing steps. VisualSfM exploits CPU and GPU parallelism by using Multicore Bundle Adjustment [14].

3.2.2 Samantha

Samantha uses a Hierarchical Cluster Tree approach in contrast to the Incremental approach, which makes it inherently parallel and scalable [8, 15, 16, 17]. It also implements auto-calibration of images, which removes dependency on the EXIF data embedded in images. Hence, we need to know the internal camera parameters. Moreover, these parameters do not have to remain constant throughout all images.

3.2.3 Autodesk 123d Catch

Autodesk has recently launched the 123d Catch [18] mobile application. It allows the user to upload a sequence of images on cloud servers that handle the processing. A detailed accuracy analysis of 123d Catch is provided in [13].

3.3 open Multiple View Geometry (openMVG)

openMVG implements a Global SfM pipeline [3]. We choose openMVG as against the other packages because of a number of reasons. It is a free open source software with active development taking place on Github [1]. It is Linux based and hence could be implemented on the high performance clusters at Michigan Technological University. The entire code could be cloned to a local machine and modified easily.

Thus, although openMVG did not always give the most accurate or the fastest results, it is the most convenient option for the purpose of this research.

3.4 Patch-based Multi-view Stereo Software (PMVS)

The output of the SfM pipeline is given to PMVS. It converts the sparse 3D model to a dense 3D model. We do not go into detail with the process of dense 3D reconstruction. In our research, the result of sensor fusion is converted to a format accepted by PMVS and then processed to create the 3D reconstructions.

3.5 Chapter Conclusion

After seeing a comparison of the various packages and evaluating the benefits of openMVG and PMVS over others, we focus the next chapter on making openMVG and PMVS run more efficiently on high performance clusters.

Chapter 4

Computing Platforms and Performance Comparison

4.1 Chapter Goals

The 3D reconstruction pipeline consists of computationally expensive operations. These include the SIFT feature and descriptor calculation, SfM and dense reconstruction. This chapter describes the hardware and software of the various computing platforms the openMVG was run on, and highlights the challenges faced on each platform. It tabulates the time taken by some of the processes as a performance metric for each of the platform.

4.2 Computing Platforms

Table 4.1
Hardware and software specifications of the computing platforms used for running openMVG

Specifications	MacBook Pro	HPC Cluster	HPC Superior (8 cores)	HPC Superior (16 cores)
Compute Nodes	1	2	1	1
CPU (Intel)	i5	E5405	E5 2670	E5 2670
CPU Speed / Core (in GHz)	2.4	2	2.6	2.6
CPU Cores	2	4	8	16
CPU Instructions / Cycle	2	4	8	8
Performance (in GFLOPS)	9.6	64	166.4	332.8
OpenMP (Parallel Computing)	No	Yes	Yes	Yes
RAM (GB)	8	8	64	64
Operating System	Mac OS X	CentOS 6.3	CentOS 6.3	CentOS 6.3
Root Access	Yes	Yes	No	No
Cmake Version	3.1.3	3.0.2	3.0.2	3.0.2
GCC Compiler Version	4.2.1	4.8.0	4.8.0	4.8.0

4.2.1 MacBook Pro

OpenMVG was first built and run on a MacBook Pro (Mac). The Mac having a BSD UNIX flavor operating system, openMVG that was originally designed for Linux was compatible with it. One of the biggest advantages of the Mac was that it had administrator rights. Hence, dependency libraries could be installed using Homebrew [19]. The Mac could process upto 50 images in a reasonable amount of time beyond which it took time of the order of hours. One of the reasons was that the architecture did not support OpenMP [20], which specifies high-level parallelism in C++ programs. For outdoor applications like infrastructure inspection or performing SLAM in a building it was common to work with 300 to 500 images. Processing these on the Mac would

take time of the order of 10 to 20 hours.

4.2.2 High Performance Cluster (HPC)

It was necessary to migrate to a system with better computing performance. At the same time it was essential that the system have root access so that dependency libraries could be easily installed. A high performance cluster was designed and built having one front end and two compute nodes of 4 cores and 8GB RAM each. It was built using Rocks Cluster Distribution 6.1 (CentOS 6.3) [21]. The configuration was a scaled down version of the Superior High Performance Cluster. This later helped in migrating the software system to Superior. The HPC cluster proved to be a test ground for prototyping. It had marginal improvement in serial computing as compared to the Mac. However as the architecture supported OpenMP, it gave about 400% performance boost for those processing runs of openMVG which exploited parallelism.

Yet for 300 to 500 images it would take around 2 hours. This added to CMVS/PMVS would give a total turnaround time of 4 hours which was considered slow for rapid prototyping of different data sets.

4.2.3 High Performance Cluster - Superior (HPC - Superior)

HPC - Superior [22] had the best computing performance available. One of the major challenges of Superior was that it did not have root access, which was required whenever scripts were written to compile packages and libraries from source.

4.3 Performance Comparison

The peak theoretical performance was calculated in FLoating-point Operations Per Second (FLOPS):

$$\text{Performance} = \text{Compute Nodes} * \text{CPU Cores} * \text{Clock (GHz)} * \frac{\text{CPU Instructions}}{\text{Cycle}}$$

For example : To compute peak theoretical performance of the HPC Cluster, we have 2 compute nodes, 4 CPU cores, 4 CPU instructions per cycle and 2 GHz clock speed. Therefore,

$$\text{Performance} = 2 * 4 * 2 * 4 = 64 \text{ GFLOPS}$$

As seen in Table 4.2 and Figure 4.1, the HPC Superior exploits the openMVG parallelism making it up to 20 times faster than the MacBook. For example, processing

Table 4.2

Performance comparison of running openMVG on different computing platforms. Data sets increasing in number of images were run and timings were recorded. Superior - 8 and Superior - 16 stand for the 8 core and 16 core configuration of the Superior used.

Extract Features				
Number of Images	MacBook (hh:mm:ss)	HPC Cluster (hh:mm:ss)	Superior - 8 (hh:mm:ss)	Superior - 16 (hh:mm:ss)
25	0:01:22	0:01:45	0:00:57	0:00:50
50	0:02:48	0:03:29	0:02:04	0:01:42
100	0:05:39	0:07:09	0:03:28	0:03:24
300	0:19:08	0:21:02	0:10:06	0:10:01
500		0:35:28	0:20:00	0:16:49
Performance of Superior - 16 w.r.t Performance of MacBook = 1.72				
Performance of Superior - 16 w.r.t. performance of HPC Cluster = 2.09				

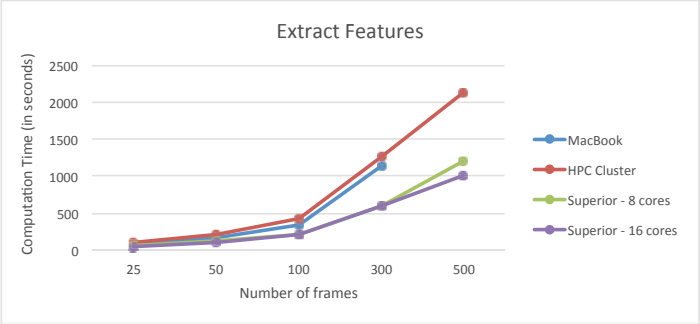
Putative Matches				
Number of Images	MacBook (hh:mm:ss)	HPC Cluster (hh:mm:ss)	Superior - 8 (hh:mm:ss)	Superior - 16 (hh:mm:ss)
25	0:01:54	0:00:33	0:00:16	0:00:13
50	0:07:20	0:01:50	0:00:51	0:00:40
100	0:30:23	0:07:06	0:02:17	0:02:19
300	4:17:44	0:55:10	0:21:01	0:17:23
500		2:05:32	0:54:02	0:45:16
Performance of Superior - 16 w.r.t. Performance of MacBook = 11.79				
Performance of Superior - 16 w.r.t. Performance of HPC Cluster = 2.84				

Geometric Filtering				
Number of Images	MacBook (hh:mm:ss)	HPC Cluster (hh:mm:ss)	Superior - 8 (hh:mm:ss)	Superior - 16 (hh:mm:ss)
25	0:02:01	0:00:23	0:00:12	0:00:06
50	0:07:58	0:01:32	0:00:48	0:00:23
100	0:36:00	0:06:42	0:03:52	0:01:39
300	5:07:38	1:00:12	0:32:16	0:15:00
500		2:41:42	1:29:19	0:40:36
Performance of Superior - 16 w.r.t. Performance of MacBook = 20.77				
Performance of Superior - 16 w.r.t. Performance of HPC Cluster = 3.98				

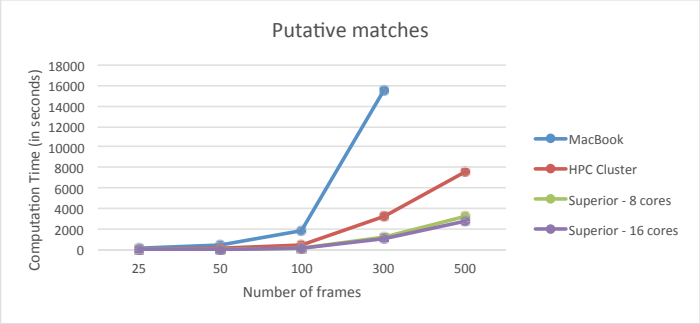
300 images would take 42 minutes on the Superior using all 16 cores as compared to 10 hours on the MacBook Pro. For UAV or robot SLAM applications it is common to have around 500 to 1000 images. In such a scenario, the use of the Superior is completely justified.

4.4 Chapter Conclusion

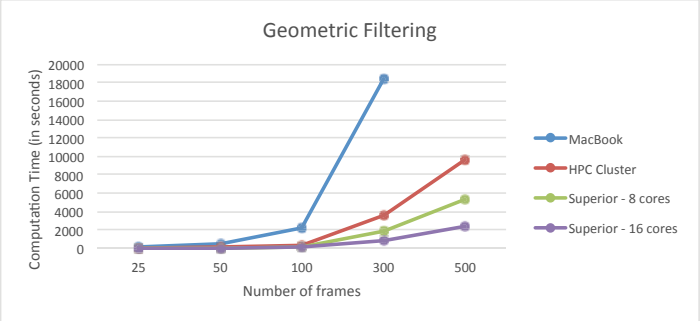
Successful implementation of openMVG on the HPC Superior was demonstrated through execution times of serial and parallel operations of SfM. The scripts and other implementation details can be found in Appendix B.



(a) Extract Features



(b) Putative Matches



(c) Geometric Filtering

Figure 4.1: Performance comparison graphs of the computing platforms for increasing number of image data sets. Figure 4.1(a) shows that since Extract Features is a serial process there is no significant difference in the performance. Figure 4.1(b) uses openMP parallelism and hence significant performance boost is observed on HPC Cluster and HPC Superior. Figure 4.1(c) being a highly parallel process exploits the 16 cores of HPC Superior making it about 20 times faster than the MacBook.

Chapter 5

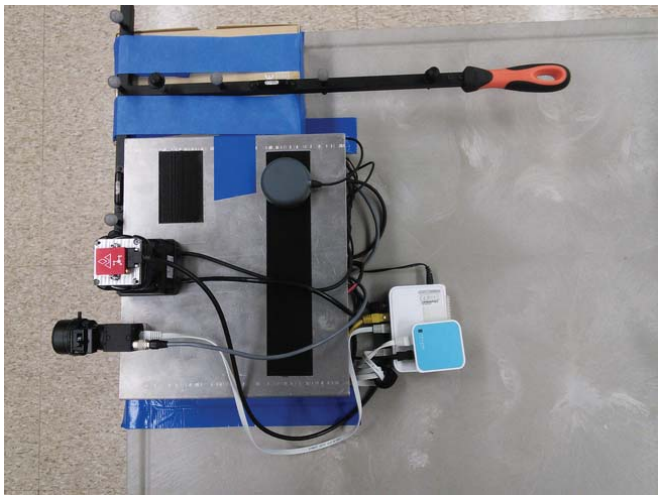
Experimental Setup

5.1 Chapter Goals

The goals of this chapter are i) to describe the sensors used to acquire data, ii) to describe the Immersive Visualization Studio, iii) to compare data from the Immersive Visualization Studio to that from the sensors for accuracy, iv) to establish relationships between coordinate systems.

(a) Motion Capture Room

(b) Wand



(c) Sensors

Figure 5.1: Figure 5.1(a) shows the motion capture room. The wand is kept in the center which is calibrated as the world origin. Figure 5.1(b) shows the wand which is a T-shaped object with reflectors attached asymmetrically. Figure 5.1(c) shows the sensors mounted on a cart system. The sensor mounting and data acquisition system was prepared by the team members at IRL.

Table 5.1
Sampling rates of each sensor

Sensor	Sampling Rate
IVS	100 Hz
Camera	1 to 2 frames/s
LIDAR	40 Hz
IMU	100 Hz

5.2 Sensors

Four sensors were used for data collection. Their sampling frequencies are shown in Table 5.1. The camera, LIDAR and IMU had the same time system and data could be easily compared. The IVS had its own time system and we had to use indirect ways to synchronize the time with the rest of the sensors. Figure 5.1(a) shows the IVS with the wand kept at the global origin. Figure 5.1(b) shows the wand which is the pre-defined object tracked by the IVS. Figure 5.1(c) shows the sensor mounted along with the wand.

5.2.1 Camera

We are using the camera by Point Grey has a specified focal length between 2.5mm and 8mm. However we calibrate is using the standard checkerboard pattern every time we collect data. The approximate focal length after calibration is 895. It has a

wide angle lens which allows us to capture wider imagery from close distances. This introduces radial distortion. The procedure to calibrate and determine the intrinsic parameters was described in Chapter 2. The Point Grey allows us to collect video at 5fps however for SfM, we use every 5^{th} frame. Thus we consider the camera to have sample frequency of 1 frame per second.

5.2.2 LIDAR

The LIDAR by Hokuyo is a 2D LIDAR, which can scan its plane through an angle of 270° (a sweep). It is able to measure with accuracy the points which lie anywhere between $0.75m$ and $30m$ from the LIDAR. Each sweep is of 1081 points. It has a sampling frequency of 40Hz.

5.2.3 Inertial Measurement Unit (IMU)

We use the VectorNav VN-200 IMU. The output we obtain is in quaternions which we convert to Euler angles and rotation matrices.

5.3 Immersive Visualization Studio (IVS) - Motion Capture Room

The Immersive Visualization Studio (IVS), also known as the motion capture room is a setup of 12 cameras which collectively tracks the motion i.e., the trajectory of the pose of a predefined object - the wand (described in detail in 5.3.1).
The IVS provided position readings accurate to the order of a few millimeters and orientation within a degree of accuracy, thus allowing us to assume it as ground truth.

5.3.1 Wand

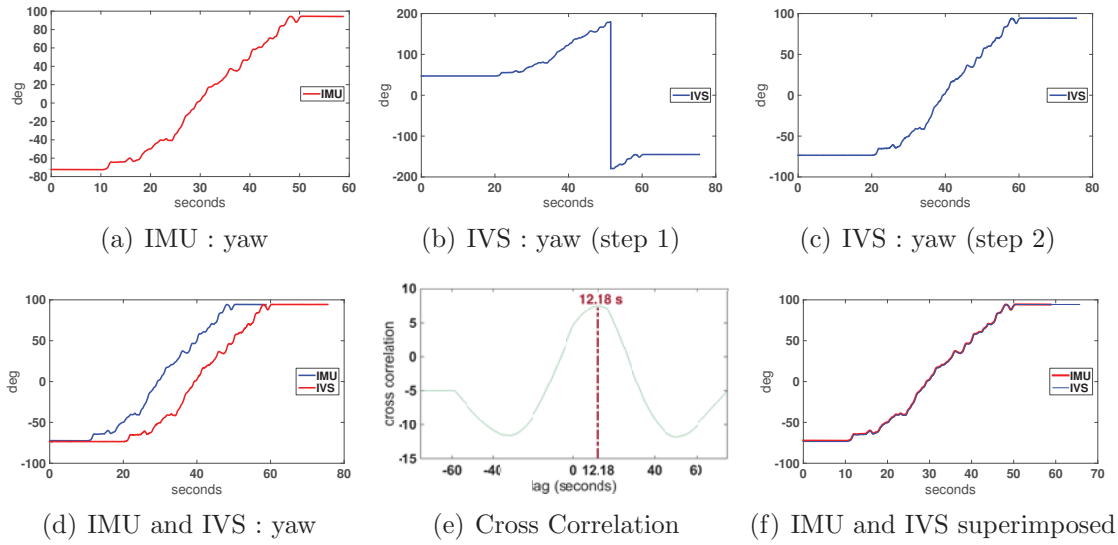
The tracking software tracks a pre-defined object called the ‘wand‘ which is shown in Figure 5.1(b). It is a T-shaped object with 5 reflectors attached asymmetrically to it. The center of the object was the centroid of these 5 reflector nodes.

5.3.2 Output from the IVS system

Output from the IVS is the pose of the wand. The position is given as global translation i.e. a vector $[xyz]$ w.r.t. to the world origin. The orientation is given in the form of Euler angles, rotation matrix or quaternions. Each one has its own advantages and disadvantages. The Euler angles can be easily related to the orientation as they are in the form of roll, pitch and yaw. The quaternions do not have singularity where the Euler can be faced with a gimbal lock. The rotation matrix could be used for comparison with the output from the SfM.

5.4 IVS and IMU Timestamp Synchronization

One the issues with using the IVS data was that it had a different time stamp compared to the rest of the sensors. It order to make use of the data from the IVS, it was essential to convert the readings such that they align with those of the IMU. Figure 5.2(a) shows the yaw reading from the IMU. Figure 5.2(b) shows the yaw reading from the IVS. However we observe that although it looks similar to that from the IMU, we are not able to compare it one to one. As per the IVS convention, the range for angle is from $-180°$ to $180°$ and hence we can see the discontinuity. In Figure 5.2(c) we show the adjusted plot. At this point we have the y-axis of the IVS

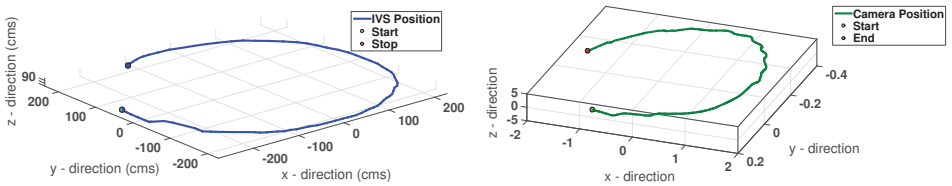


(a) IMU : yaw (b) IVS : yaw (step 1) (c) IVS : yaw (step 2)

(d) IMU and IVS : yaw (e) Cross Correlation (f) IMU and IVS superimposed

Figure 5.2: Figure 5.2(a) shows the yaw angular readings of the IMU. Figure 5.2(b) shows the yaw angular readings of the IVS. Figure 5.2(c) shows the angle adjustment required to bring it within range of the IMU. Figure 5.2(d) shows the IVS data aligned on the IMU data. Figure 5.2(e) shows the cross correlation performed. Figure 5.2(f) shows the superimposed data after cross correlation.

aligned with that of the IMU as seen in Figure 5.2(d). Also we note that the sampling frequency of the IMU and the IVS is the same i.e. 100Hz. In order to align the x-axis, we cross correlate the two signals. Cross correlation gives us the time stamp difference between the IMU and IVS systems. Also we can now see in Figure 5.2(f) the IVS reading superimposed on the IMU. We observe a very high level of accuracy. This is important because now we can assume that the orientation obtained from the IMU to be the ground truth for data collected outside the IVS.



(a) IVS Position (b) Camera Position

Figure 5.3: Figure 5.3(a) shows the position trajectory of the sensors using the IVS system. This we assume to be the ground truth. Figure 5.3(b) shows the position trajectory of the sensors using the camera system. Notice the camera coordinate axes and their directions w.r.t. that of the IVS system.

5.5 Coordinate Systems of Sensors

The IVS helped understand the relation between the coordinate systems of the various sensors. For example, as shown in Figure 5.3(b) we can see how the IVS and the camera uses different coordinate system. Although the wand, camera, LIDAR and IMU were placed close to each other on a single unit, it was important to make note of the initial pose of each sensor. We position the wand such that the centroid coincides with the camera center. The pose of the camera with respect to the origin of the world coordinate system is noted. We repeat the same for the other sensors. Then the wand is fixed at its own position and this pose is once again noted with respect to the world origin. Thus we obtained the poses of the sensors w.r.t. each other and w.r.t. to the IVS system. We use the notation and convention for pose as given by [23]. For example, to obtain the pose of the camera w.r.t. to LIDAR we use Equation (5.1).

$$
{}^{origin}(\zeta)_{LIDAR} = \begin{bmatrix} I_{3\times3} & 0_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix} \begin{bmatrix} R_{LIDAR} & t_{LIDAR} \\ 0_{1\times3} & 1 \end{bmatrix}
$$

$$
{}^{origin}(\zeta)_{camera} = \begin{bmatrix} I_{3\times3} & 0_{3\times1} \\ 0_{1\times3} & 1 \end{bmatrix} \begin{bmatrix} R_{camera} & t_{camera} \\ 0_{1\times3} & 1 \end{bmatrix} \tag{5.1}
$$

$$
{}^{LIDAR}(\zeta)_{camera} = {}^{origin}(\zeta)^{-1}_{LIDAR} \cdot {}^{origin}(\zeta)_{camera}
$$

$$
= {}^{LIDAR}(\zeta)_{origin} \cdot {}^{origin}(\zeta)_{camera}
$$

5.6 Chapter Conclusion

As a result of the experimentation in the IVS, we could verify the convention of the global and local coordinate systems (of each sensor). We established the relation to describe pose of a sensor relative to another sensor. We successfully compared our sensor results (both orientation and position) with the IVS which acted as the ground truth. Based on the comparison results of the IVS with the IMU, we can for all practical purposes assume the IMU orientation to act as ground truth for any data

set collected outside the IVS.

Chapter 6

Sensor Fusion

6.1 Chapter Goals

The goals of this chapter are i) to explain what sensor fusion is, ii) to compare the observed strengths and weaknesses of sensors after running them on datasets, iii) to explain the need for sensor fusion and how it will provide a better solution to the problem of 3D reconstruction of images, iv) to explain through experiments how sensor fusion worked in some cases, how and why it failed in others.

6.2 Sensor Fusion - An Introduction

Sensor fusion is the technique of combining the outputs of multiple sensors measuring the same value (measurement could be direct, or indirect after processing the raw measurement using some algorithm). For example, the IMU measures orientation directly while the camera measures orientation indirectly after running the SfM algorithm on the captured images. Also the final value we need to measure could be a combination of more than one value. For example, the final value we need is the pose which is the combination of the orientation and position.

The fusing in sensor fusion can be at multiple levels. For example, the orientation obtained from the camera could be replaced entirely by that from the IMU while retaining the position from the camera. Also, we could just use those orientations from IMU which lie within an error threshold from the camera. Furthermore, we could use the IMU orientation data in the very primitive stages as an initial condition or a priori information.

6.3 Sensor Comparison before Data Collection

It was observed that while each of the sensors (camera, IMU and LIDAR) are capable of providing the pose, each sensor has its own strength and weaknesses. The data from the IVS is assumed to be accurate and is used as ground truth to compare with each of the sensors. We also established the fact that outside the IVS, the orientation from the IMU could be considered as ground truth.

6.4 Datasets and Results

In this section we show our results on 3 different datasets. In order to best explain the 3D point clouds on paper, we show a set of four 2D images and their corresponding snapshots of the 3D point cloud. Also in order to explain the specifics within any image, we implement a simple procedure. Imagine a grid as shown in Figure 6.1 to be placed over every image dividing the image into nine parts and labeled as shown. We would then refer to any sub-part of an image using N-E-W-S terminology. For example, the ladder can be seen in parts N,C and S of Figure 6.1.

We also describe our datasets by a list of keywords given in Table 6.1.

Table 6.1
Following keywords are used to describe datasets and their motions

Keyword	Explanation
target	the object which we intend to reconstruct in 3D
indoor	inside a building or a room, furthest distance not more than 10m
outdoor	outside environments where the target object could be up to 50m, other object could go up to infinity
gloss	shiny objects not depicting true color or direct source of light like bulbs
matte	objects whose brightness does not change with viewing angle
yaw rotation	the motion in which the sensors rotate while being parallel to the ground at all times, sensors inscribe a circular figure
outer yaw rotation	yaw rotation around the target, sensors face normally inwards while inscribing the circle (includes significant translation)
inner yaw rotation	yaw rotation, sensors face outwards towards target while inscribing the circle (includes significant translation)
point yaw rotation	inner yaw rotation but at the same point (includes minimal translation)
translation motion	a linear motion parallel to the target

Figure 6.1: A grid to explain within image specifics.

6.4.1 Dataset 1 : IVS

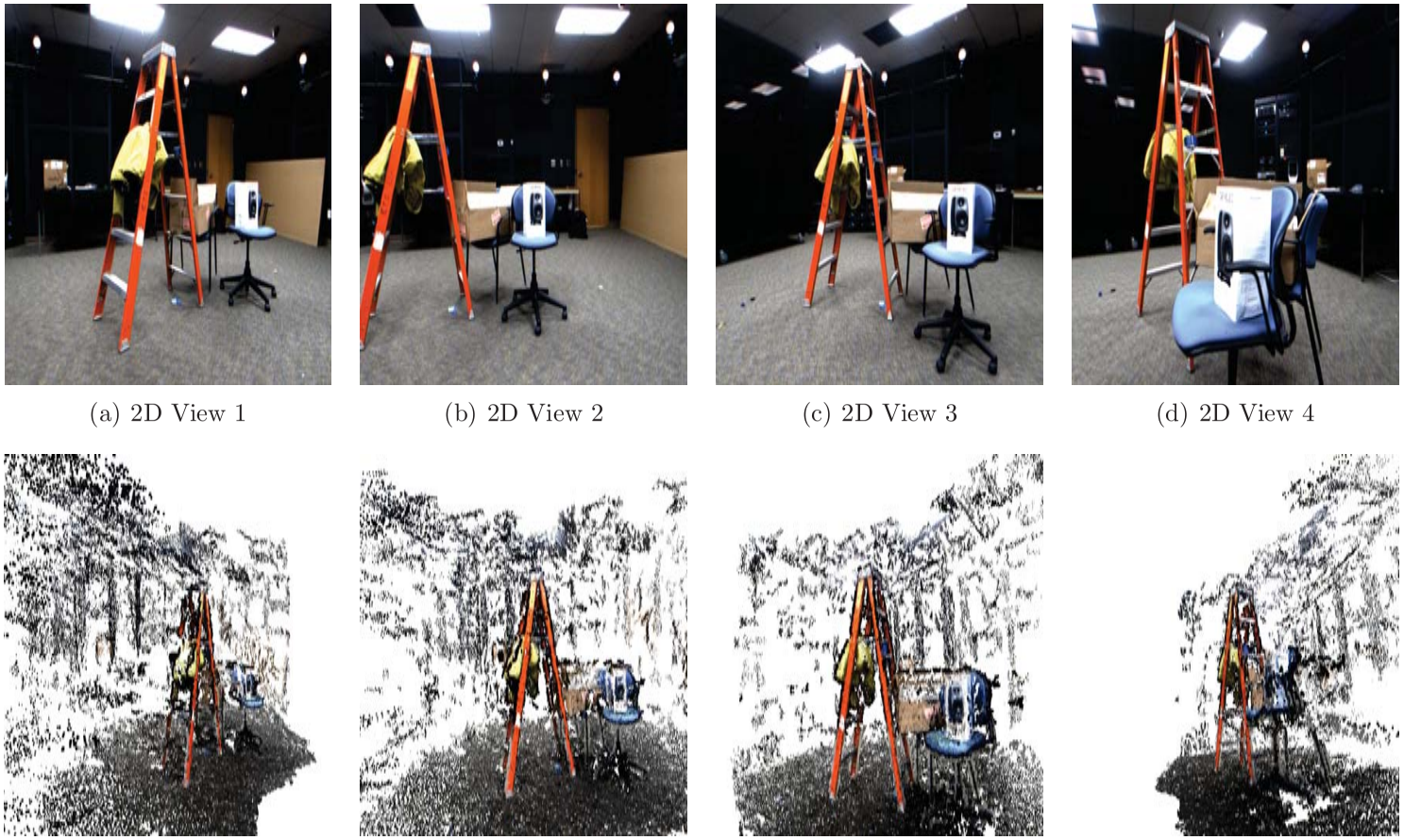
The dataset shown in Figure 6.2 is from the IVS. We would like to describe the IVS environment by the following keywords - indoor, matte target, glossy background. We describe the dataset collection as an outer yaw motion.

Figure 6.2 shows the outer yaw motion with images and corresponding point clouds. Overall the 3D reconstruction has been successful since it visually resembles the actual target in structure and color. In Figure 6.2(a), we can see the overall structure of the target. This structure is maintained in the reconstruction as can be seen in Figure 6.2(e). Part N of Figure 6.2(b) is glossy. We see that there has been no reconstruction in the N part of Figure 6.2(f). In E part of Figure 6.2(c) we can see a box kept on the chair. The box has a picture of a speaker on it. Figure 6.2(g) show the clarity with which this picture of the speaker can be seen. Thus the 3D reconstruction takes place with true reproduction of the color. We can therefore say that for the given dataset environment and motion description, the camera has performed well in reproducing the structure and color of the target.

We now look at the map created by the LIDAR. As seen in Figure 6.3(a), the indoor environment is clearly mapped with the target in the center. The pink dots clearly show the trajectory of the sensor. The room boundaries and the target can be seen in blue. As compared to the camera output, the LIDAR has produced an excellent map of the target with respect to the entire surrounding area. Thus for the given

dataset the LIDAR too has performed well.

In order to look at it more analytically we plot the position and orientation as a function of time. As previously established in Chapter 5, we assume the IMU readings to be ground truth. We can see in Figure 6.3(b) that both the LIDAR and camera have readings close to that of the IMU. Figure 6.3(d) and Figure 6.3(c) show the position trajectories of the LIDAR and camera respectively. As we can see they are similar to each other, and this is a confirmation of their correctness. They depict the outer yaw motion. The only difference is that LIDAR maps in a 2D plane and the camera in 3D space.



(a) 2D View 1 (b) 2D View 2 (c) 2D View 3 (d) 2D View 4

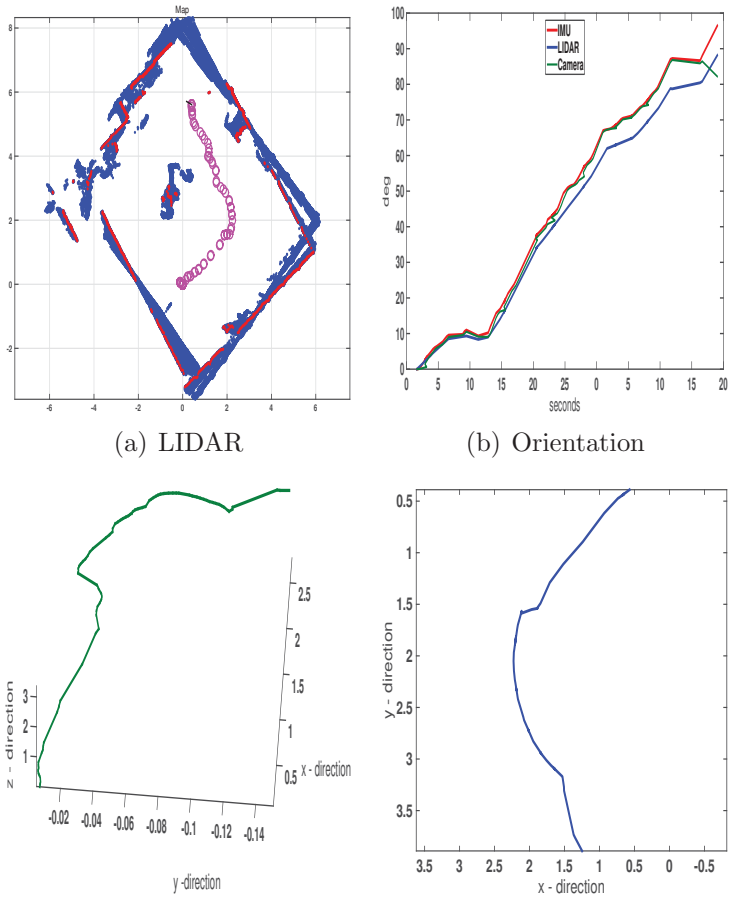
(e) 3D View 1 (f) 3D View 2 (g) 3D View 3 (h) 3D View 4

Figure 6.2: Figure 6.2(a) to 6.2(d) show the 2D views of the dataset in the IVS. Figure 6.2(e) to 6.2(h) show the corresponding 3D point clouds. This dataset is indoor, matte target, gloss background with an outer yaw rotation

6.4.2 Dataset 2 : Materials and Minerals Building

This is the second dataset. We describe this dataset as outdoor and matte. The sensor measuring motion is once again outer yaw.

We see in the NE part of Figure 6.4(a) that there are trees. Now trees are considered as non-rigid objects in the sense that they move across different frames. As can be seen in Figure 6.6(e) the trees are not reconstructed. In Figures 6.4(b) and 6.4(c) we can see a board in the C part and W part of the images respectively. This board is on the lawn in front of the main structure. We see from part S of Figure 6.4(f) and part E of Figure 6.4(g) that is had been recreated such that it accurately shows how it is in front of the main structure. Finally in Figure 6.4(d) we see a pole which is again in front of the main target. This too is seen clearly in E part of Figure 6.4(h). The LIDAR on the other hand, as seen from Figure 6.5(d) has not performed well on this dataset, the important dataset descriptor being outdoors. We see the orientation plots Figure 6.5(b) to see that the camera and IMU (assumed ground truth) have very similar plots. The position plot for camera Figure 6.5(c) shows that outer yaw trajectory. The LIDAR plots for orientation and position are incorrect.



(a) LIDAR (b) Orientation

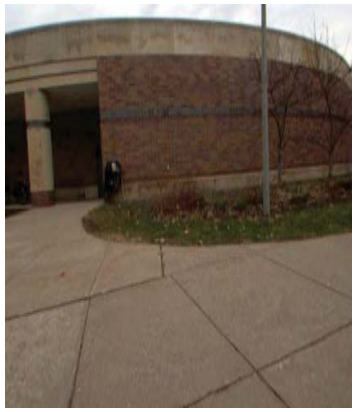
(c) Position - Camera (d) Position - LIDAR

Figure 6.3: Figure 6.3(a) shows the trajectory of the LIDAR. It show that the LIDAR has successfully mapped the surrounding area. Figure 6.3(b) shows that the orientation by the IMU, LIDAR and camera are almost overlapping. Figure 6.3(d) and Figure 6.3(c) show that the trajectories plotted by the LIDAR and camera respectively.

(a) 2D View 1

(b) 2D View 2

(c) 2D View 3



(d) 2D View 4

(e) 3D View 1

(f) 3D View 2

(g) 3D View 3

(h) 3D View 4

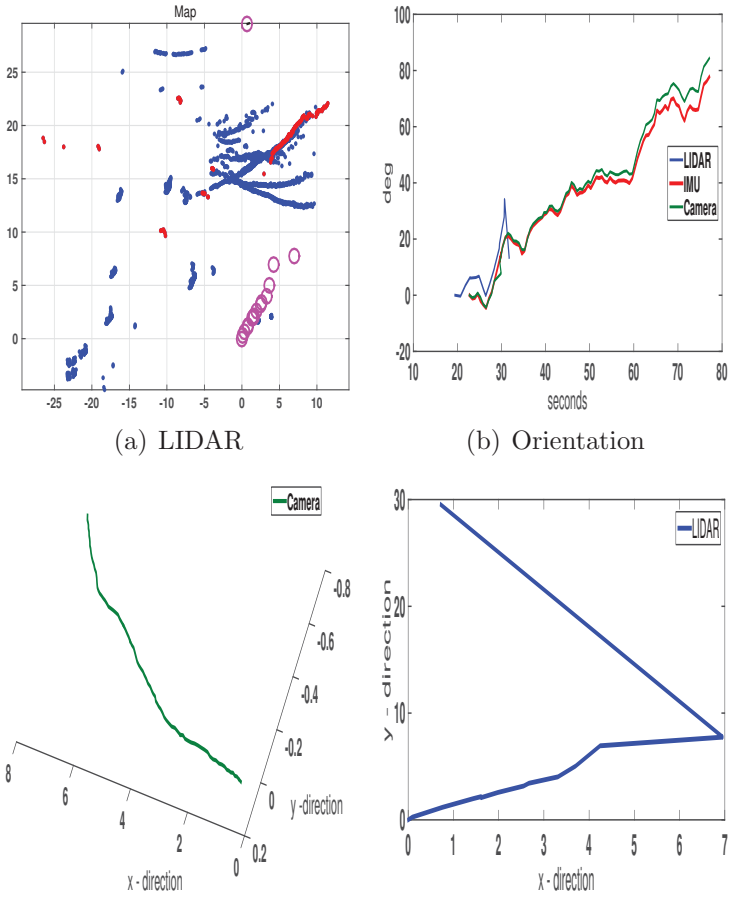
Figure 6.4: Figure 6.4(a) to 6.4(d) show the 2D images and Figure 6.4(e) to 6.4(h) show the 3D views. This dataset has been described as outdoor, matte with the motion as outer yaw rotation.

6.4.3 Dataset 3 : Dow Building

This dataset was very instrumental in the understanding of where SfM fails. Again, we describe the environment as indoor and matte which is perfect for the camera. However this time the motion in the first few frames is point yaw rotation and then it becomes translation. Specifically the point yaw rotation is when we come to the pillar shown in Figure 6.6(c) and 6.6(d).

As shown in the point cloud, parts NW, W, SW of Figure 6.6(e) and part NW, W 6.6(f) the pillar is not reconstructed. Further in the Figures 6.6(g) and 6.6(h) where the motion is translational, the reconstruction is perfect.

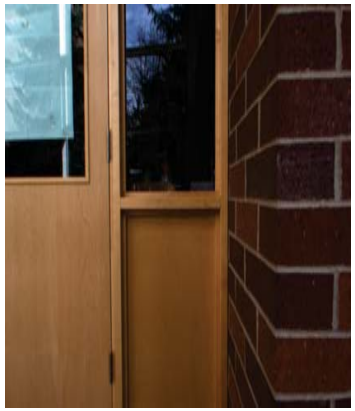
The LIDAR algorithm on the other hand works flawlessly in this environment. This can be seen from the LIDAR map in Figure 6.7(a) and the plots in Figure 6.7(b)and 6.7(d).



(a) LIDAR (b) Orientation

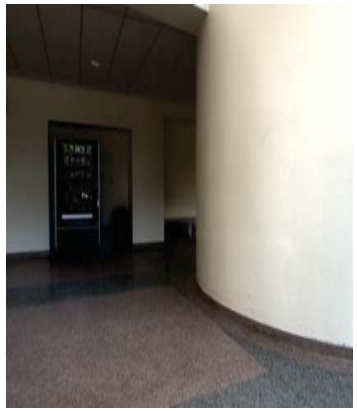
(c) Position - Camera (d) Position - LIDAR

Figure 6.5: Figure 6.5(a) shows that the LIDAR has not performed as expected for dataset 2. Figures 6.5(b), 6.5(d) and 6.5(c) show the plots of the LIDAR and camera. The camera can be seen to have performed better.

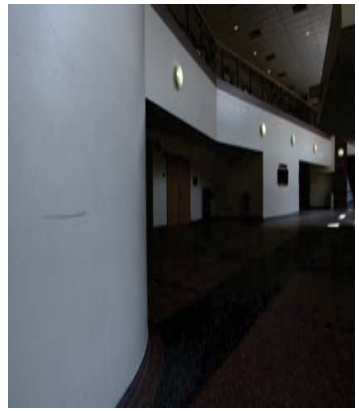


(a) 2D View 1

(b) 2D View 2



(c) 2D View 3



(d) 2D View 4

(e) 3D View 1

(f) 3D View 2

(g) 3D View 3

(h) 3D View 4

Figure 6.6: Figure 6.6(c) and 6.6(d) show the pillar about which motion was point yaw. As can be seen in Figure 6.6(e) and 6.6(f) the pillar has not been reconstructed. Figure 6.6(d) E part shows the path at which motion was translational. This part has been reconstructed in Figure 6.6(g).

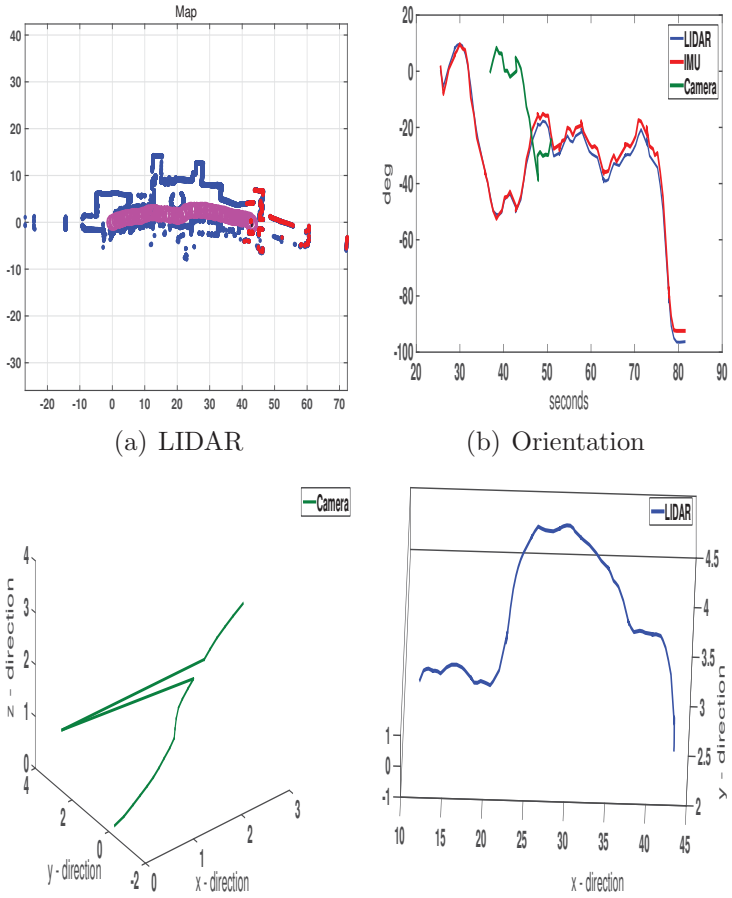
6.5 Methodology

6.5.1 Experiment 1

While the IMU provides excellent orientation it does a poor job of providing the position. This is primarily due to the integration errors which go on accumulating as the time progresses. The camera on the other hand provides fair orientation but excellent position.

In the most simple model, we create a pose by taking the position obtained from the camera and the orientation from the IMU. However this did not work and the results can be seen in Figure 6.8. The reason for this is that the orientation and position obtained from SfM are coupled together and one cannot be replaced directly without changing the other. This can be seen in Equation (6.1).

$$\tilde{C} = -R^{\intercal}t \tag{6.1}$$



(a) LIDAR (b) Orientation

(c) Position - Camera (d) Position - LIDAR

Figure 6.7: Figure 6.7(a) shows that the LIDAR has done the perfect job of mapping the environment. This can also be seen from the plots in Figure 6.7(b), 6.7(d).

6.5.2 Experiment 2

In this experiment taking the previous Equation (6.1) into consideration, we propose that the rotation can be replaced without changing the translation only if it is below some threshold.

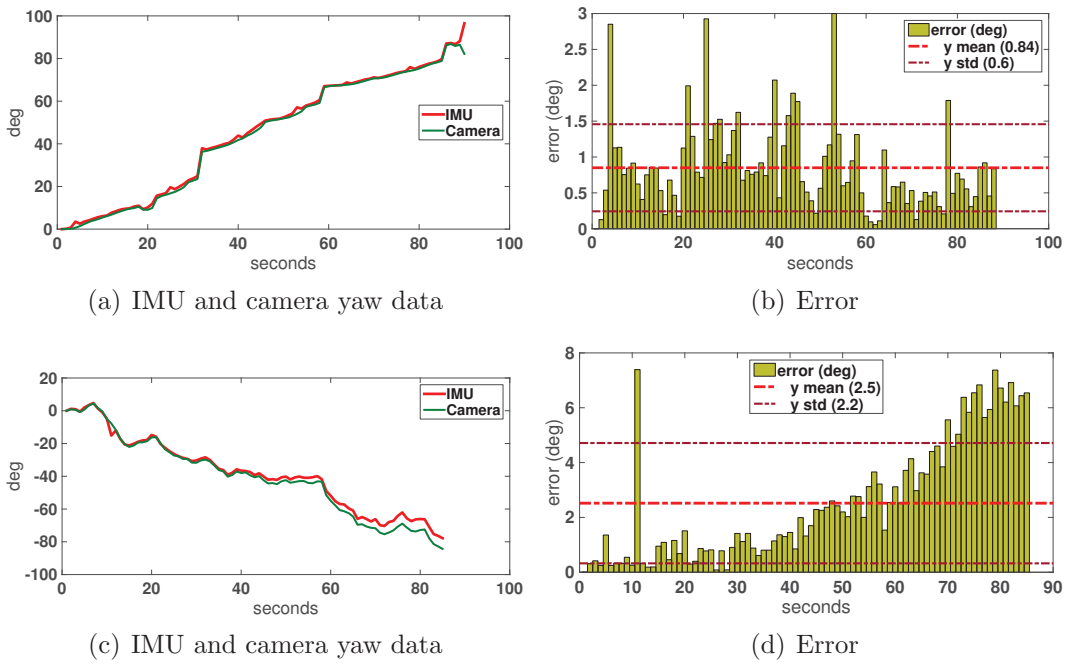
Figure 6.8: A failed example of sensor fusion as a result of replacing all camera rotations by IMU rotations

We once again superimpose the readings of the IMU with those from the camera as shown in Figure 6.9(a). This time however we plot a graph of the difference in the readings for every time step. Figure 6.9(b) shows that the mean error is 0.84° with a standard deviation of 0.6° which is excellent considering that the camera outputs the orientation as a result of SfM and not directly.

Now as compared to the previous experiment instead of replacing all values, we replace only those up to an error threshold. In order for this to work, the mean error should be not more than 2°. Figure 6.10(a) shows the original reconstruction using only the camera. Figure 6.10(b) shows the camera and IMU fusion for a threshold less than the positive standard deviation. Thus

$$\text{threshold} \leq \text{mean} + \text{std}$$

i.e., in our example all values where the error is less than $0.84° + 0.6° = 1.44°$. In Figure 6.10(b), we see an improvement in the jacket which is kept on the ladder in



(a) IMU and camera yaw data

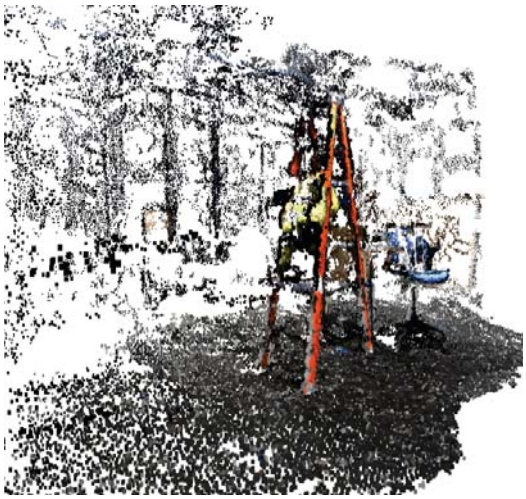
(b) Error

(c) IMU and camera yaw data

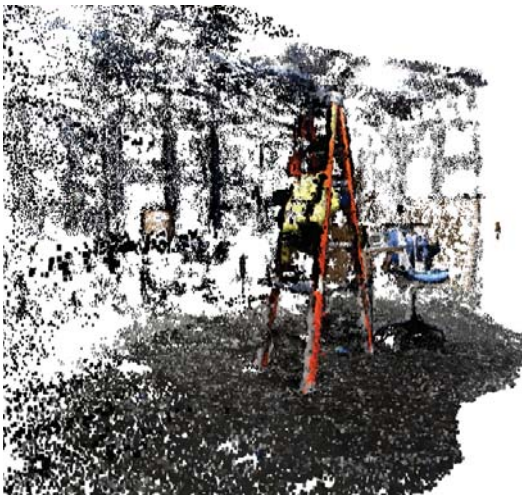
(d) Error

Figure 6.9: Figure 6.9(a) shows the output of the camera superimposed on that of the IMU. Figure 6.9(b) shows how the readings from the camera differ from that of the IMU as a function of time. Figure 6.9(c) shows the plot for another dataset. From Figure 6.9(c) we can see that the mean error value is very high and that it increases with time. Experiment 2 does not perform well for second dataset.

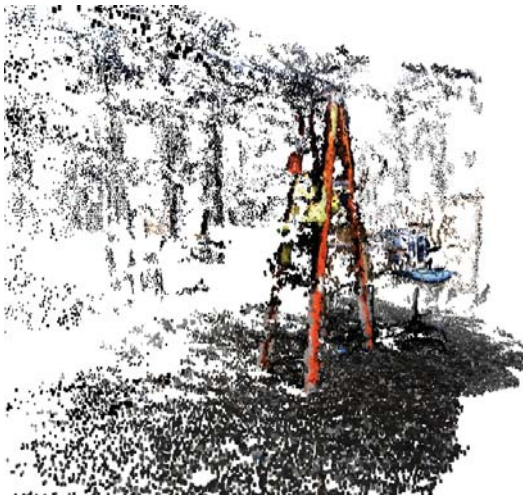
part C. Also the box to the left of the ladder and the frame behind it in part NW and W appear more prominent. The box kept on the chair to the right of the ladder in part E shows better perspective. In Figure 6.10(c) the threshold is high and the result is poorer than that taken by the camera alone. Finally in Figure 6.10(d) we see the result without having any threshold. We perform Experiment 2 on dataset 2 (Figure 6.4). As seen in Figure 6.9(c) and 6.9(d) the error is significant and goes on increasing. Thus Experiment 2 does not perform as well as seen in Figure 6.11.



(a) Camera only



(b) Camera and IMU (threshold less than positive standard deviation)



(c) Camera and IMU (high threshold)

(d) Camera and IMU (no threshold)

Figure 6.10: Sensor fusion for the rotation yaw

6.5.3 Experiment 3

We propose Experiment 3 which could serve as an improvement to Experiment 2. Although we have not been able to implement it, we formulate the problem and

Figure 6.11: Figure shows that Experiment 2 did not perform well on Dataset 2 as the mean error was very high.

derive the mathematical equation.

One observation as a result of our experimentation was that SfM did not perform well in cases where the translation (baseline) was small. We use this fact to our advantage. Thus instead of directly replacing the rotation obtained from the SfM by that from the IMU, we now replace only those values where the rotation error is high and the translation is small.

Thus we can now formulate the problem we are about to solve : Given a start X_A and an end X_B point, we define a trajectory of 5 vectors and 5 rotations as shown in Figure 6.12. We assume t_A and t_B and hence the translation vectors from SfM are accurate. We assume that t_1, t_2, t_3 are small and we have to estimate these translation vectors. Also the vector t_{AB} is accurately known. Our third assumption is that the rotations are obtained from the IMU and are accurate at all instances.

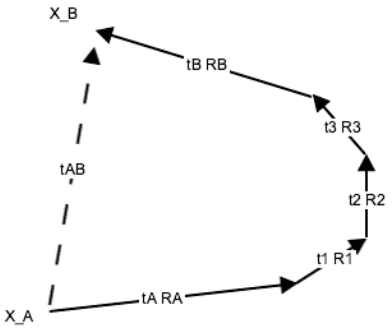


Figure 6.12: Trajectory of 5 translations and 5 rotations

Thus we obtain Equation (6.2).

$$\begin{aligned} t_{AB} &= R_B(R_3(R_2(R_1(R_A(X_A) + t_A) + t_1) + t_2) + t_3) + t_B \\ &= R_B R_3 R_2 R_1(X_A) + R_B R_3 R_2 R_1 t_A + R_B R_3 R_2 t_1 + R_B R_3 t_2 + R_B t_3 + t_B \\ &= A + B + Ct_1 + Dt_2 + Et_3 + t_B \end{aligned} \tag{6.2}$$

We wish to minimize the error of the estimated point $\hat{X_B}$ for the values of t_1, t_2, t_3. Using this we will be able to include rotations obtained from IMU while adjusting the translations.

6.6 Chapter Conclusion

In this chapter we analyzed different datasets. We found that in some cases the LIDAR performs better and in the others, the camera. We also showed how it is not only the type of environment but also the manner in which the data is collected that makes a difference. We then proposed the simplest form of sensor fusion and showed that it failed, while explaining the tight coupling between rotation and translation. We then also proposed another method which worked for minor errors in the rotation.

Chapter 7

Results

The UAV at the Intelligent Robotics Lab at Michigan Technological University can now successfully obtain its pose from a sequence of images taken from the camera mounted on it. Camera calibration procedure has been explained. Camera parameters for the Point Grey camera has been obtained and their significance and impact on 3D reconstruction has been detailed through experimentation. The algorithm for Structure from Motion has been explained and implemented successfully for an image pair from the data set.

State-of-the-art Structure from Motion packages were introduced. The complete installation procedure and usage of one such package - openMVG has been consolidated in the form of shell scripts attached in the appendix. The scripts include the installation of CMake and other dependencies required for openMVG. Special attention was

given to the installation procedure to ensure their installation without the need for root or admin access. This enables the use of High Performance Computing Clusters like the Superior at Michigan Technological University. The Superior has made it possible to process a large data set (around 500) of images within an hour as compared to 15 to 20 hours on other local systems.

The sensors, their coordinate systems and accuracy was verified at the Immersive Visual Studio at Michigan Technological University. The orientation readings of the Inertial Measurement Unit were established to be very close to that obtained from the Immersive Visual Studio, hence giving a ground truth for outdoor data sets with no access to the Immersive Visual Studio. The sensors were tested on various data sets and their strengths and weaknesses were found out. Pose obtained from the camera was compared with that obtained from the LIDAR. It was concluded that the pose estimate from the sensors depended not only on the environment type but also on the trajectory of the sensors during data collection.

Different methods were proposed to fuse the pose estimate obtained from the different sensors. Through experimentation it was concluded that the rotation and translation obtained from Structure from Motion is coupled and hence it is not possible to replace any one of them without modifying the other. Two workarounds for this were proposed.

References

[1] P. Moulon, "openmvg: "open multiple view geometry"."

[2] P. Moulon, P. Monasse, and R. Marlet, "Adaptive structure from motion with a contrario model estimation," in *Computer Vision–ACCV 2012*, pp. 257–270, Springer, 2013.

[3] P. Moulon, P. Monasse, and R. Marlet, "Global fusion of relative motions for robust, accurate and scalable structure from motion," in *Computer Vision (ICCV), 2013 IEEE International Conference on*, pp. 3248–3255, IEEE, 2013.

[4] "Matlab and computer vision system toolbox release 2014b, the mathworks, inc., natick, massachusetts, united states.."

[5] A. Vedaldi and B. Fulkerson, "VLFeat: An open and portable library VLFeat: An open and portable library of computer vision algorithms." `http://www.vlfeat.org/`, 2008.

[6] 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*, vol. 24, no. 6, pp. 381–395, 1981.

[7] R. Hartley and A. Zisserman, *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[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*, pp. 1489–1496, IEEE, 2009.

[9] C. Wu, "Towards linear-time incremental structure from motion," in *3D Vision-3DV 2013, 2013 International Conference on*, pp. 127–134, IEEE, 2013.

[10] C. Wu, "Visualsfm: A visual structure from motion system (2011)," *URL http://www.cs.washington.edu/homes/ccwu/vsfm*.

[11] A. Strupczewski and B. Czupryński, "3d reconstruction software comparison for short sequences," in *Symposium on Photonics Applications in Astronomy, Communications, Industry and High-Energy Physics Experiments*, pp. 929030–929030, International Society for Optics and Photonics, 2014.

[12] C. Santagati, L. Inzerillo, and F. Di Paola, "Image-based modeling techniques for architectural heritage 3d digitalization: Limits and potentialities," *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 5, no. w2, pp. 555–560, 2013.

[13] M. L. Brutto and P. Meli, "Computer vision tools for 3d modelling in archaeology," *International Journal of Heritage in the Digital Era*, vol. 1, pp. 1–6, 2012.

[14] C. Wu, S. Agarwal, B. Curless, and S. M. Seitz, "Multicore bundle adjustment," in *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pp. 3057–3064, IEEE, 2011.

[15] R. Toldo, *Towards automatic acquisition of high-level 3D models from images*. PhD thesis, Universita' degli Studi di Verona, Strada le Grazie 15, 37134 Verona Italy, 2013.

[16] 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*, pp. 1594–1600, IEEE, 2010.

[17] R. Gherardi and A. Fusiello, "Practical autocalibration," in *Computer Vision–ECCV 2010*, pp. 790–801, Springer, 2010.

[18] Autodesk, "123d catch (http://www.123dapp.com/gallery/catch)."

[19] "Homebrew (http://brew.sh/)."

[20] OpenMP Architecture Review Board, "OpenMP application program interface version 3.0," May 2008.

[21] "Rocks cluster distribution (http://www.rocksclusters.org/wordpress/)."

[22] "Superior - a shared hpc cluster at michigan technological university."

[23] P. I. Corke, *Robotics, vision and control : fundamental algorithms in MATLAB*. Berlin: Springer, corrected 2nd printing. ed., 2013.

Appendix A

Code

A.1 RANSAC.m

```
%% RANSAC
% Author: Anuj Potnis
% Date: 11 December 2014
% HZ Algorithm 4.4 (p.118)
% This is a standalone RANSAC code for a straight line

clear;clc;close all
% Sample size S
S = 10;
x = 1:S;
mtrue = 5; ctrue = 10;
xx = mtrue*x + ctrue;
```

```
rng;
xx_noise = xx + 5*rand(1,10) - mean(5*rand(1,10));
xx_noise(3) = xx_noise(3)+25;
scatter(x,xx_noise,55,'r','filled');
axis tight;
hold on
lsline
scatter(x,xx,55,'g');

%% Randomly select a sample of s data points from S
% and instantiate the model from this subset.

%% Select s as the minimum sample size (necessary) from ↩
   S
s = 2;
iter = 10;
i = 1;
while i < iter
    % Generate any two random integers and sort them to ↩
       simplify line formula
    ind = sort(randperm(S,s));

    x2 = x(ind(2));
    xx2 = xx_noise(ind(2));

    x1 = x(ind(1));
    xx1 = xx_noise(ind(1));

    m_est = (xx2-xx1)/(x2-x1);
    c_est = xx2-m_est*x2;

    t = 0.5;
    th = atand(m_est);
    h = t/sind(90-rad2deg(atan(m_est)));
```

```
    refline(m_est,c_est)
    refline(m_est,c_est+h)
    refline(m_est,c_est-h)

    % Check if point is within threshold t
    % Distance of point p(m,n) from line Am + Bn + C = 0
    % is d = |Am + Bn + C|/sqrt(A^2+B^2)
    A = m_est;
    B = -1;
    C = c_est;

    d = abs(A*x + B*xx_noise + C)/sqrt(A^2+B^2);
    inlier_idx = find(d<t)

    Ssub(i) = numel(x(inlier_idx))

    % If the size of Si (the number of inliers
    % is greater than some threshold T
    T = 7;
    if Ssub(i) > T
        break
    end
    i = i+1;
end

title('RANSAC Implementation')
legend('Noisy Data','LMS fit','Ideal Data','RANSAC ↩
   tolerance band', ...
    'Location','southeast')
xlabel('x'); ylabel('xx')
print('-depsc','RANSAC')
%% Determine the set of data points Si which are within
% a distance threshold t of the model
```

A.2 RQ_decomposition.m

```
%% Given rotation and RQ decomposition

clear; clc;close
format long
P = [ 3.53553*10^2 3.39645*10^2 2.77744*10^2 ↩
   -1.44946*10^6;
     -1.03528*10^2 2.33212*10^1 4.59607*10^2 ↩
        -6.32525*10^5;
     7.07107*10^-1 -3.53553*10^-1 6.12372*10^-1 ↩
        -9.18559*10^2]
%%
A = P(:,1:3)
%A = [1 4 7;2 5 8;3 6 9]; %-- DANGEROUS
%A = eye(3)
%A = [1 0 0; 1 1 0; 1 1 1];
%A = magic(3)

%%

cx = -A(3,3)/sqrt( A(3,2)^2 + A(3,3)^2);
sx = A(3,2)/sqrt( A(3,2)^2 + A(3,3)^2);
Qx = [1 0 0;
     0 cx -sx;
     0 sx cx];
Ax = A*Qx
%%
cy = Ax(3,3)/sqrt( Ax(3,1)^2 + Ax(3,3)^2);
sy = Ax(3,1)/sqrt( Ax(3,1)^2 + Ax(3,3)^2);
Qy = [cy 0 sy;
```

```
    0 1 0;
    -sy 0 cy];
Ay = Ax*Qy
%%
cz = -Ay(2,2)/sqrt( Ay(2,2)^2 + Ay(2,1)^2);
sz = Ay(2,1)/sqrt( Ay(2,2)^2 + Ay(2,1)^2);
Qz = [cz -sz 0;
    sz cz 0;
    0 0 1];
Az = Ay*Qz
%%
Q = Qz'*Qy'*Qx';
R = A*Qx*Qy*Qz

Acomposed = R*Q;
Recomposition_Error = abs(A - Acomposed)
format
```

A.3 homography_DLT.m

```
%% Homography
% 10th December 2014
% Hartley and Ziesserman : Multiple View Geometry
% 4.1 The Direct Linear Transformation (DLT) algorithm (↩
   p.88)
clear; clc; close


%%

x1 = [1 1 1]';
```

```
xx1 = [10 10 1]';
A1 = [0 0 0 -xx1(3)*x1' xx1(2)*x1';
    xx1(3)*x1' 0 0 0 -xx1(1)*x1'];

x2 = [5 1 1]';
xx2 = [15 10 1]';
A2 = [0 0 0 -xx2(3)*x2' xx2(2)*x2';
    xx2(3)*x2' 0 0 0 -xx2(1)*x2'];


x3 = [5 5 1]';
xx3 = [15 15 1]';
A3 = [0 0 0 -xx3(3)*x3' xx3(2)*x3';
    xx2(3)*x3' 0 0 0 -xx3(1)*x3'];


x4 = [1 5 1]';
xx4 = [10 15 1]';
A4 = [0 0 0 -xx4(3)*x4' xx4(2)*x4';
    xx4(3)*x4' 0 0 0 -xx4(1)*x4'];

% For over-determined
% This is an additional point we add to create an over-↩
   determined system.
% Since the points are accurate (noise-free), the rank ↩
   of A remains 8,
% and a null space of 1-dimension exists.

x5 = [3 3 1]';
xx5 = [12.5 12.5 1]';
A5 = [0 0 0 -xx5(3)*x5' xx5(2)*x5';
    xx5(3)*x5' 0 0 0 -xx5(1)*x5'];

A = [A1;A2;A3;A4]
```

```
% For over-determined
%A = [A1;A2;A3;A4;A5]
H = null(A)
H = reshape(H,[3,3])'

x = [x1 x2 x3 x4]
xx = [xx1 xx2 xx3 xx4]
xx_est_sc = H*x;
xx_est = xx_est_sc./xx_est_sc(3,1)

scatter([x1(1) x2(1) x3(1) x4(1) xx1(1) xx2(1) xx3(1) ↩
   xx4(1) ], ...
     [x1(2) x2(2) x3(2) x4(2) xx1(2) xx2(2) xx3(2) xx4(2)↩
        ]); hold on
scatter(xx_est(1,:), xx_est(2,:), 'filled')

x_test = [4.5 1.5 1]';
xx_test_sc = H*x_test
xx_test = xx_test_sc./xx_test_sc(3)
scatter([x_test(1) xx_test(1)], [x_test(2) xx_test(2)], ↩
   'filled')

title('Homography mapping and projection')
legend('True value', 'Calculated', 'Test points','↩
   Location','southeast')
axis equal
print('-depsc','homography')
```

A.4 cameramodel_ideal.m

```
close
```

```
%K = [fx 0 px; 0 fy py; 0 0 1]
th = 0;
R = [1 0 -sind(th);0 cosd(th) 0;0 sind(th) cosd(th)];
fx = 1; fy = 1; px = 0; py = 0;
K = [fx 0 px; 0 fy py; 0 0 1]

%R = [1 0 0;0 1 0;0 0 1];
C = [0 0 0]';
t = -R*C;

P = K*[R t]

x = P*X; % 2D points
I = find(x(3,:) == 0, length(X));
x = [x(1,:)./x(3,:); x(2,:)./x(3,:) ];
figure(2)
scatter(x(1,:),x(2,:),50,'filled')
%plot(x(1,:),x(2,:))
axis equal
```

A.5 camerapose.m

```
clear;clc;close all;

%url =  'http://141.219.218.16:8080/shot.jpg';
img_old  = imread('temple0045.png');
fh = image(img_old);
cam = CentralCamera('image', img_old, 'focal', 0.015204,...
    ...
'resolution', [640 480], 'centre', [302.32 246.87]);
Rold = [1 0 0 ;0 1 0 ; 0 0 1];
```

```
i=1;
for inum = 44:-1:40;
img_name = sprintf('temple00%d.png',inum);
%while(1)

    %img_new  = imread('HouseBlack__Curve_020_Rot_140.↩
       png');
    img_new = imread(img_name);
    set(fh,'CData',img_new);
    drawnow;

    img_new=single(rgb2gray(img_new));
    sz=size(img_new);
    [f1,d1] = vl_sift(img_new);

    if(~ismatrix(img_old))
        img_old=single(rgb2gray(img_old));
    end
    img_new=imresize(img_new,[sz(1),NaN]);
    [f2,d2] = vl_sift(img_old);

    img_old = img_new;

    thresh=5;
    [matches, scores] = vl_ubcmatch(d1,d2,thresh);
    fprintf('Number of Matches: %d\n',size(matches,2))

    indices1=matches(1,:);
    f1match=f1(:,indices1);
    d1match=d1(:,indices1);

    indices2=matches(2,:);
    f2match=f2(:,indices2);
    d2match=d2(:,indices2);
```

```
    u1 = f1match(1:2, :);
    u2 = f2match(1:2, :);

    F = estimateFundamentalMatrix(u1',u2', ...
        'Method','RANSAC', 'NumTrials', 2000,...
        'DistanceThreshold', 1e-4);

    E = cam.E(F);
    sol = cam.invE(E, [0,0,10]');
    [Rnew,t] = tr2rt(sol);
    Rpresent = Rold*Rnew;

    RPY(:,i) = tr2rpy(Rpresent, 'deg');
    Roll(i) = RPY(1);
    Yaw(i) = RPY(2);
    Pitch(i) = RPY(3);

    i=i+1;
    Rold = Rnew;

%end
end
i = 1:5;
plot(i,RPY(2,:))
title('Camera Pose')
xlabel('Frame number')
ylabel('Angle (deg)')
legend('Pitch','location','northeast')
print('-depsc','camerapose_RPY')
```

Appendix B

BASH Shell Scripts

B.1 CMake_build.sh

```
#!/bin/bash

# BASH script to install CMAKE version 3.0.2 on HPC ↩
   Superior (without root access)
#
# Usage: sh CMake_build.sh
#
# Tested on :
# MacBook - Mac OS X 10.9.5 (execution time 20 mins)
# HPC Superior - CentOS 6.3 (execution time from 5 mins ↩
   to 15 mins)
#
# NOTE:
# No root access is required
# Use absolute path of cmake while using it. For eg: ↩
   $HOME/research/apps/cmake-3.0.2/bin/cmake
```

```
#
# This is a modified version of a script provided by Dr.↩
    Gowtham (Michigan Tech University)


# Necessary variables
export TODAY=`date +"%Y%m%d_%H%M%S"`
echo $TODAY
export CMAKE_VERSION="3.0.2"
echo $CMAKE_VERSION

# Create necessary directories if they do not already ↩
   exist
mkdir -p $HOME/research/src/
mkdir -p $HOME/research/apps/

# Download the source file here
cd $HOME/research/src/
wget http://www.cmake.org/files/v3.0/cmake-3.0.2.tar.gz

# Untar the tar here
tar -zxvf cmake-${CMAKE_VERSION}.tar.gz -C $HOME/↩
   research/apps

# Cmake compile
cd ../apps/cmake-${CMAKE_VERSION}/
./bootstrap --prefix=$HOME/research/apps/cmake-${↩
   CMAKE_VERSION} 2>&1 | tee $HOME/research/apps/cmake-${↩
   CMAKE_VERSION}/bootstrap_${TODAY}.txt
make          2>&1 | tee $HOME/research/apps/cmake-${↩
   CMAKE_VERSION}/make_${TODAY}.txt
make install 2>&1 | tee $HOME/research/apps/cmake-${↩
   CMAKE_VERSION}/make-install_${TODAY}.txt
```

B.2 openMVG_build.sh

```
#!/bin/bash

# BASH script to compile and build openMVG
#
# Usage : sh openMVG_build.sh
#
# Tested on :
# MacBook - Mac OS X 10.9.5 (execution time 35 mins)
# HPC Superior - CentOS 6.3
# (execution time from 5 mins to 35 mins depending on ↩
   number of cores selected)
#
# NOTE:
# Multicore compilation is supported and the number of ↩
   cores have to be specified as
# make -j NBcore (replace NBcore by the number of ↩
   threads)
# CMake : Specific version is used by giving absolute ↩
   path of cmake-3.0.2
#
# For more details visit the openMVG website :
# https://raw.githubusercontent.com/openMVG/openMVG/↩
   master/BUILD


cd $HOME/research/
git clone --recursive https://github.com/openMVG/openMVG↩
   .git
mkdir openMVG_Build
cd openMVG_Build
```

```
# Uncomment the below only if using a different version ↩
   of gcc from default
#export CC=$HOME/research/apps/gcc-4.8.0/bin/gcc
#export CXX=$HOME/research/apps/gcc-4.8.0/bin/g++
#export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/research/↩
   apps/gcc-4.8.0/lib/

# Use absoule path of CMake if not using default
# Refer to the script cmake_compile for more details
$HOME/research/apps/cmake-3.0.2/bin/cmake -↩
   DCMAKE_BUILD_TYPE=RELEASE . ../openMVG/src/
echo
echo "CMake executed"
echo

# Enter number of cores
make -j 2

# Minor modifications in openMVG file structure
cd $HOME/research/openMVG_Build/software/globalSfM
cp openMVG_main_GlobalSfM ../SfM
echo
echo "Copied openMVG_main_GlobalSfM to main directory"
echo

# Create directory for a Dataset
mkdir -p $HOME/research/3D_Reconstruction/Dataset

# Copy all binaries to a single folder
cd $HOME/research/openMVG_Build/software/SfM/
cp openMVG_main_computeMatches openMVG_main_CreateList ↩
   openMVG_main_GlobalSfM openMVG_main_openMVG2PMVS $HOME↩
   /research/3D_Reconstruction/
```

```
echo
echo "openMVG successfully installed"
echo
```

B.3 openMVG_run.sh

```
#!/bin/bash

# BASH script to run openMVG/PMVS/CMVS on a dataset of ↩
   images to create a 3D reconstruction
#
# Usage : openMVG_run.sh
#
# Tested on :
# MacBook - Mac OS X 10.9.5
# HPC Superior - CentOS 6.3
# Execution time : Largely
# NOTE:
# Assumes all images are in a folder called images
# For more details visit the openMVG website :
# http://openmvg.readthedocs.org/en/latest/software/SfM/↩
   intrinsicGroups/
#

# openMVG_main_CreateList -f 895 (focal length in pixels↩
   ) -i Dataset/images/ -o Dataset/matches/
./openMVG_main_CreateList -f 895 -i Dataset/images/ -o ↩
   Dataset/matches/
```

```
# openMVG_main_computeMatches [optional args] -i Dataset↩
   /images/ -o Dataset/matches/
./openMVG_main_computeMatches -g e -p 0.01 -r 0.8 -s 1 -↩
   i Dataset/images/ -o Dataset/matches/

# Run Structure from Motion on input images and previous↩
    output (select method 1 or 2 when prompted)
./openMVG_main_GlobalSfM -i Dataset/images/ -m Dataset/↩
   matches/ -o Dataset/outGlobalSfM

# Convert openMVG output to PMVS
./openMVG_main_openMVG2PMVS -i Dataset/outGlobalSfM/↩
   SfM_output/ -o Dataset/outGlobalSfM/SfM_output/

# Run CMVS to cluster images before input to PMVS
./cmvs Dataset/outGlobalSfM/SfM_output/PMVS/ 15
./genOption Dataset/outGlobalSfM/SfM_output/PMVS/

# Run PMVS to create dense 3D reconstruction
./pmvs2 Dataset/outGlobalSfM/SfM_output/PMVS/ ↩
   pmvs_options.txt

echo
echo "3D reconstruction completed"
echo
```

B.4 send.sh

```
#!/bin/bash
```

```
# Script to send data to a folder from a cluster from a ↩
   folder called project_X
# on local system ($HOME/project_X)
#
# Usage : send.sh
#
# This is a modified version of a script provided by Dr.↩
    Gowtham (Michigan Tech University)
#

# Uncomment the DESTINATION as per reuiqrement
export DESTINATION="superior-login1.research.mtu.edu"
#export DESTINATION="un5395-aspotnis.research.mtu.edu"
#export DESTINATION="portage-login.research.mtu.edu"

# Send files from local folder project_X to cluster
rsync -ave ssh -hPz $HOME/project_X/ ↩
   aspotnis@$DESTINATION:research/3D_Reconstruction/↩
   Dataset/

echo
echo "Sent"
echo
```

B.5 receive.sh

```
#!/bin/bash

# Script to receive data from a folder from a cluster to↩
    a folder called project_X
# on local system ($HOME/project_X)
```

```
#
# Usage : receive.sh
#
# This is a modified version of a script provided by Dr.↩
    Gowtham (Michigan Tech University)
#


# Uncomment the DESTINATION as per reuiqrement
export DESTINATION="superior-login1.research.mtu.edu"
#export DESTINATION="un5395-aspotnis.research.mtu.edu"
#export DESTINATION="portage-login.research.mtu.edu"

# Receive output of openMVG from cluster to local folder↩
    project_X
rsync -ave ssh -hPz aspotnis@$DESTINATION:research/3↩
   D_Reconstruction/Dataset/outGlobalSfM $HOME/project_X/

echo
echo "Received"
echo
```