A REPORT

ON

TIME SYNCHRONISATION OF MOBILE MAPPING SYSTEM

BY

Name	ID
Sagnik Majumder	2014A8PS464P
Raunaq Majumdar	2014A8PS396P
Vema Teja Krishna	2014A3PS166P

PREPARED IN PARTIAL FULFILLMENT OF PS-1 COURSE

AT

INDIAN INSTITUTE OF REMOTE SENSING, DEHRADUN

A Practice School-I station of

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

(JULY 2016)

A REPORT

ON

TIME SYNCHRONISATION OF MOBILE MAPPING SYSTEM

BY

Name	ID	Branch
Sagnik Majumder	2014A8PS464P	B.E. (Hons.) Electronics and Instrumentation Engineering
Raunaq Majumdar	2014A8PS396P	B.E. (Hons.) Electronics and Instrumentation Engineering
Vema Teja Krishna	2014A3PS166P	B.E. (Hons.) Electronics and Electronics Engineering

Prepared in the partial fulfilment of the

Practice School-I Course

AT

INDIAN INSTITUTE OF REMOTE SENSING, DEHRADUN

A Practice School-I station of

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE, PILANI

(JULY 2016)

ACKNOWLEDGEMENTS

We would sincerely like to thank the Director of Indian Institute of Remote Sensing, Dr. A Senthil Kumar, for giving us an opportunity to work in this organization and gain a significant amount of exposure to corporate work culture, ethics, and etiquettes to be followed while working in a professional environment.

We would also like to thank the coordinator of the PS-1 program at IIRS, Mrs. Shefali Agarwal, Head of the Department, Photogrammetry and Remote Sensing Division (PRSD), for guiding us through the program and supporting us throughout.
We would like to extend our deepest gratitude to our project in-charge, Dr. Raghavendra S, Scientist at PRSD, IIRS, for providing us with the opportunity to work with him on this project, and his guidance and mentorship during the same.

We are highly indebted to Dr. Praveen Goyal, the faculty in charge of the PS-1 program at IIRS, for his guidance during the PS–1 program, and his helpfulness and responsiveness while addressing all the concerns we raised during the same. We also thank Mr.Siddhant Pundir, the co-instructor of the PS-1 program for all the help and suggestions that he gave in favour of our project.

We would like to extend our gratitude to all other members of the organization, who have been co-operative with us during the program while having us as interns, and to the members of the Practice School Division, who have worked very hard for making this program a very fruitful one in terms of the experience gained by the students.

We are also extremely grateful to Mr. Abhinav Gupta for helping us build our code on Linux and make executable files

Sagnik Majumder
Raunaq Majumdar
Vema Teja Krishna

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE

PILANI (RAJASTHAN)

Practice School Division

Station: Indian Institute of Remote Sensing **Centre**: Dehradun

Duration: From 23rd May, 2016 to 16th July, 2016
Date of Submission: 11 July, 2016

Title of the Project: Time Synchronisation in Mobile Mapping System

ID No.	Name of Student	Discipline
Sagnik Majumder	2014A8PS464P	B.E. (Hons.) Electronics and Instrumentation Engineering
Vema Teja Krishna	2014A3PS166P	B.E. (Hons.) Electrical and Electronics Engineering
Raunaq Nandy Majumdar	2014A8PS396P	B.E. (Hons.) Electronics and Instrumentation Engineering

Name of expert: Mrs. Shefali Agarwal

Designation: Head of the department, Photogrammetry and Remote Sensing Division

Name of PS Faculty: Dr. Praveen Goyal

Keywords:

Project Area: TLS (Terrestrial Laser Scanner), IRNSS Receiver, IMU (Inertial Measurement Unit)

ABSTRACT: The difficulty that IIRS Dehradun are facing presently in building their own Mobile Mapping System (MMS) is that the TLS and the GPS are from different manufacturers and cannot be easily time synchronised. Hence, geo-referencing of the acquired 3D scan with accuracy is not possible. We have used RiVLib libraries and have built executable files to extract the time stamp information along with other scan attributes from the '.rxp' file produced by the TLS and, automatically synchronise the GPS and the TLS and initiate a scan by pre-setting the scan parameters. We set up servers on our computer and could finally synchronise the TLS and the GPS. The computer can act both as a server and a client. When it becomes a client, it takes the accurate time information from the GPS receiver using Network Time Protocol and upon becoming a server it passes on that time information to the TLS called Dynamic Host Configuration Protocol. Thus the TLS indirectly gets the time information from the GPS receiver and they are time synchronised.

Signature of Student
Date:

Signature of PS Faculty
Date:

TABLE OF CONTENTS

Acknowledgement …… III

Abstract …… IV

List of Abbreviations …… VIII

List of Figures …….. IX

1. INTRODUCTION…….. 1

1.1. History of IIRS ……... 1

1.2. Mobile mapping system …..... 1

1.3. Report Structure ……..... 1

1.4. Main Objective ……... 2

1.4.1. Extraction of time stamp information of TLS..……..... 2

1.4.2. Time Synchronisation of GPS and TLS…….. 2

2. HARDWARE AND SOFTWARE USED …… 3

3. LITERATURE Review …….. 4

4. WORKING PRINCIPLES OF INSTRUMENTS IN BRIEF …… 5

4.1. A Short note on laser scanning…… 5

4.2. Classification of terrestrial laser scanners …….. 5

4.3. A brief overview of the principles and methods of laser scanning…….. 6

4.4. A brief overview of GPS ……..... 8

5. HARDWARE BASED APPROACH OF TIME SYNCHRONISATION …….. 9

5.1 Extraction of timestamp from .rxp file ……..... 9

5.2 Synchronisation by hard connection between GPS receiver and TLS …….. 12

6. SOFTWARE BASED APPROACH FOR TIME SYNCHRONISATION ……. 17

6.1 Solutions to our problems …….. 18

6.2 Extraction of timestamp from .rxp files and automatic initiation of scans using c code …… 18

6.2.1 Code #1 – C code for scan attributes extraction …….. 18

6.2.1.1 C code …….. 19

6.2.1.2 Building the code on Ubuntu: ……. 22

6.2.1.3 Running the executable on Ubuntu…….. 22

6.2.1.4 Output of code ……. 23

6.2.2 Code #2 - Code to synchronise with GPS, initiate and continue scan until the data limit is reached …….. 24

6.2.2.1 C++ code ……. 24

6.2.2.2 Building the code on Ubuntu …….. 29

6.2.2.3 Running the executable on Ubuntu …… 29

6.2.2.4 Output of code …… 30

6.2.3 Code #3 – Combined code to synchronise with gps, initiate and continue scan until the data limit is reached and extract the scan attributes from the '.rxp' file or live data stream…….. 30

6.2.3.1 C and C++ header……... 31

6.2.3.2 Building the code on Ubuntu…….. 39

6.2.3.3 Running the executable on Ubuntu…… 40

6.2.3.4 Output of code …… 40

6.3 Use of servers to time synchronise TLS and GPS …… 40

6.3.1 Using NTPD to set up NTP Communication …….. 42

6.3.1.1 Steps for installing NTPD and setting up NTP server/client on Ubuntu …… 43

6.3.1.1.1 Installation ……. 43

6.3.1.1.2 Running NTPD as server/client ……. 45

6.3.2 Using GPSD to set up communication between GPS receiver and NTPD …….. 49

6.3.2.1 Steps for installing GPSD and running it ……..... 49

6.3.2.1.1 Installation …….. 49
6.3.2.1.2 Running GPSD …… 52

6.3.3 Using DHCPD to set up DHCP communication ……57

6.3.3.1 Steps for installing GPSD and running it ……..57
6.3.3.1.1 Installation …… 57
6.3.3.1.2 Running the DHCP server ……... 58

7. WORK TO BE DONE FOR FUTURE DEVELOPMENT …… 71

8. CONCLUSION …….. 72

9. REFERENCES…… 73

9.1 Research papers ……... 73

9.2 Website links……. 74

LIST OF ABBREVIATIONS

1. TLS – Terrestrial Laser Scanner
2. GPS – Global Positioning System
3. IRNSS - Indian Regional Navigation Satellite System
4. NTP – Network Time Protocol
5. NTPD – Network Time Protocol Daemon
6. DHCP – Direct Host Control Protocol
7. GPSD – Global Positioning System Daemon
8. IMU- Inertial Measurement Unit
9. GUI – Graphical User Interface
10. NMEA- National Marine Electronics Association
11. HMI – Human Machine Interface

LIST OF FIGURES

1. Figure-1: External View of a TLS …… 7
2. Figure-2: Connection Ports of a TLS……..... 7
3. Figure-3: External View of an IRNSS Receiver …… 8
4. Figure-4: Human Machine Interface of IRNSS Receiver…… 9
5. Figure-5: Screen Capture of RiScan Pro……...…10
6. Figure-6: Screen Capture of text file exported by RiScan Pro…… 11
7. Figure-7: Screen Capture of text file exported by RiScan Pro…… 11
8. Figure-8: Screen Capture of text file exported by RiScan Pro……. 12
9. Figure-9: Intra GUI Command Panel of IRNSS Receiver……13
10. Figure-10: Intra GUI Command Panel of IRNSS Receiver……..14
11. Figure-11: Intra GUI Command Panel of IRNSS Receiver……..14
12. Figure-12: Intra GUI Command Panel of IRNSS Receiver……..15
13. Figure-13: A Single-Port-to-Dual-Port Connector……..……..15
14. Figure-14: HMI of TLS…….16
15. Figure-15: HMI of TLS…….16
16. Figure-16: Parts of the text file produced from '.rxp file'…….23
17. Figure-17: Representative Diagram of Synchronization……...…42
18. Figure-18: 'service ntp start' on Command Window……...46
19. Figure-19: 'ntpq-p' on Command Window……...…47
20. Figure-20: 'timedatecl status' on Command Window……...48
21. Figure-21: Using GPSD to set up Communication between GPS Receiver and NTPD……...49
22. Figure-22: '/usr/local/sbin/gpsd -n' on Command Window……. 54
23. Figure-23: 'xgps' on Command Window……. 55
24. Figure-24: 'xgpsspeed' on Command Window…….55
25. Figure-25: 'cgps' on Command Window……. 56
26. Figure-26: 'cgpsmon' on Command Window……. 56
27. Figure-27: 'service isc-dhcp server start' on Command Window…… 58
28. Figure-28: 'service isc-dhcp server stop' on Command Window…… 59
29. Figure-29: 'ifconfig -a' on Command Window…….60

Chapter-1: INTRODUCTION

1.1 HISTORY OF IIRS:

The Indian Institute of Remote Sensing is a premier institute for research, higher education and training in the field of Remote Sensing, Geoinformatics and GPS Technology for Natural Resources, Environmental and Disaster Management under the Indian Department Of Space (ISRO), which was established in the year 1966 Formerly known as Indian Photo-interpretation Institute (IPI), the Institute was founded on 21 April 1966 under the aegis of Survey of India (SOI). It was established with the collaboration of the Government of The Netherlands on the pattern of Faculty of Geo-Information Science and Earth Observation (ITC) of the University of Twente, formerly known as International Institute for Aerospace Survey and Earth Sciences, The Netherlands. The original idea of setting the Institute came from India's first Prime Minister Pandit Jawahar Lal Nehru during his visit to The Netherlands in 1957.

The Institute's building at Kalidas Road, Dehradun was inaugurated on 27 May 1972. Since its founding, the Institute has been playing a key role in capacity building in remote sensing and geoinformatics technology and their applications for the benefit of the user community from India and abroad. Today, it has programmes for all levels of users, i.e. mid-career professionals, researchers, academia, fresh graduates and policy makers.

1.2 MOBILE MAPPING SYSTEM:

Mobile mapping is collecting geo-spatial data from a mobile device. It consists of mainly a laser scanner (2D/3D), a GPS device an Inertial Measurement Unit (IMU). The system has various real life applications like analysing the road condition, analysing the building material quality etc. The use of MMS in road condition surveying (roads, structures, facilities, exclusive use) reduces exclusive road use time in site operations, and enables the acquisition of 3D data for roads and their surroundings in an efficient yet low-cost manner. Through this technology, it is possible to gain an accurate and detailed understanding of the current situation. There are many other uses of the Mobile Mapping System that we will be discussing in course of time.

1.3. REPORT STRUCTURE

This report consists of 4 sections each among which some have subparts. The first part describes the problem we were assigned and gives a brief explanation about it. The second part consists of the different research papers and literatures we went through and what we learnt from it regarding the current status of research on MMS and what we were supposed to do. The third part consist of the process we have followed and the problems we have faced and solutions to the problems, for future reference. The fourth part consists scope of improvement in future. The report is complemented with various screenshots depicting the progress of our work. The pictures are duly marked for readers' reference. We have used various abbreviations in the report and the acronyms have been mentioned separately. The report ends with a brief conclusion.

1.4. MAIN OBJECTIVE

1.4.1. EXTRACTION OF TIME STAMP INFORMATION OF TLS:

The time stamps or time information of the laser scans obtained directly from the '.rxp' file ,produced by the TLS, using RiScan Pro were showing anomalies in the sense that they were not increasing regularly in accordance with the order of scanning, they kept looping over and over again and they also did not start from zero value. So we were expected to somehow extract the correct timestamps of the scanned points which increased in accordance with the order of scanning.

1.4.2. TIME SYNCHRONIZATION OF GPS AND TLS:

We were also supposed to time synchronise the GPS and the TLS. The synchronisation could be either achieved by a software based approach or by a hardware based approach. The main impediment in synchronisation is that the manufacturers of the two systems and the data sampling rates of the two devices are different and can't be directly synchronised. But to geo reference the acquired scan data, we need to time synchronise both the device.

Chapter – 2:
HARDWARE AND SOFTWARE USED

We worked with the following instruments and devices

1. Riegl Terrestrial Laser Scanner(TLS) – VZ-400
2. IRNSS/GPS/SBAS Receiver – by Accord Software and Systems.
3. Inertial Measurement Unit (IMU) –by Microstrain

The software we used are as follows

1. IRNSS-UR GUI - For logging and processing GPS data and also for changing receiver settings
2. RISCAN-Pro – For configuring the TLS and for processing data obtained from the TLS
3. Code::Blocks IDE – For writing C and C++ code
4. RivLib Libraries
5. CMake – For building the code into executable files
6. Linux OS and related packages – For building and running soft servers

The TLS comes with internal GPS which is perfectly synchronised with the device. The TLS gives around 122000 measurements/second with an accuracy of 3mm and measurement range of 0-600m.

The IRNSS receiver served our purpose as it is similar to a GPS receiver .The GPS receiver gives about 1-4 data samples per second.

For the MMS to be properly implemented, the scans obtained from the TLS should be geo-referenced and time stamped in accordance with the external GPS receiver because the internal GPS gives imprecise and inaccurate values and is most likely to jeopardize the accuracy necessary for a highly efficient MMS.

The devices are physically disconnected and their data sampling rates are also different. There is an option in the TLS to connect an external GPS receiver to it and synchronise but it turned out that synchronisation couldn't be achieved even after following the instructions precisely. So we had to time synchronise the two devices in order to extract the accurate geo-referenced data for using it in building an efficient and precise MMS.

Chapter – 3:
LITERATURE Review

Before starting any actual work, we had to go through some research papers on MMS and time synchronisation in MMS so as to get familiarized with our objective and the approach that had been used already. The study gave us some idea about what we were expected to achieve and what methods we could adopt. We have studies a hardware manual on Mobile mapping by Joshua I. France. It mainly talked about how to connect a VZ-400 TLS to a computer, acquire scan data and process it. Among other papers, one was on 6DOF semi-rigid-SLAM for mobile mapping which was particularly helpful. It mainly spoke about acquisition of 3D point clouds and using six degrees of freedom to convert the point clouds into more accurate ones in order to get proper geo-referenced data.

We found one more good research paper, namely Registration of Long-Strip Terrestrial Laser Scanning Point Clouds Using RANSAC and Closed Constraint Adjustment by Li Zheng, Manzhu Yu , Mengxiao Song , Anthony Stefanidis , Zheng Ji and Chaowei Yang ,which is little above the scope of our project but was really beneficial. It mainly talked about the registration of long-strip, terrestrial laser scanning (TLS) point clouds which is a prerequisite for various engineering tasks, including building tunnels, bridges, and roads. An artificial target-based registration method is proposed in this paper to automatically calculate registration parameters (i.e., rotation, translation) of scanned pairs without initial estimations. The approach is based on the well-known Random Sample Consensus (RANSAC) method and effectively searches the point cloud for corresponding returns from a system of artificial targets. In addition, Closed Constraint Adjustment (CCA) is integrated into the registration method to significantly reduce the accumulative error. Experimental results demonstrate the robustness and feasibility of the proposed approach. It is a promising approach to register automatically long strips with limited external control points with satisfactory precision.

The research papers fairly gave us the idea how the TLS works and how it converts 3D data to 6DOF data and how to geo reference it. They also spokes about use of different geometric projections like geometric projections, rectangular projections, cylindrical projections, lambert cylindrical projections and errors related to it and ways to solve it. It also talked about the subtle features of Riegl vz-400 which user need to keep in mind while performing experiments.

A detailed description of different laser scanners and its uses are described in the further sections.

A voxel represents a value in rectangular grid in three dimensional space and is analogous to pixels in bitmap but little different in aspect of their position is not explicitly encoded but is represented with respect to other voxels.

We also studied the hardware manuals of Riegl VZ-400 TLS and IRNSS/GPS/SBAS Receiver and software manuals of RiScan-Pro, RiVLib Libraries, and IRNSS-UR GUI among others.

Chapter – 4:
WORKING PRINCIPLES OF THE INSTRUMENTS IN BRIEF

Before directly starting with the project we learnt about the various instruments, working principles and the variations in their configurations. So we tried to understanding our project, its main objective and attempted to figure out the methodologies that we could adopt by studying the working principles of the hardware devices. A brief description of the same is given below

4.1. A SHORT NOTE ON LASER SCANNING:

Nowadays, terrestrial laser scanning has become a very efficient technique for geodetic applications. The use of laser scanners is continuously increasing. Different classes of laser scanners from several manufacturers are available. However, a strict classification of the laser scanners is quite difficult: on which point of view should the classification be based? Conceivable classifications concern the range or the point density or the point accuracy or the field of view.

4.2. CLASSIFICATION OF TERRESTRIAL LASER SCANNERS:

Based on the question above, classification of terrestrial laser scanners will be discussed here. First of all, there is no single universal laser scanner for all conceivable applications. Some scanner is most suitable for indoor use and medium ranges (up to max. 100 m), while some other scanner is most suitable for outdoor use with long ranges (up to several 100 m). The decision to choose the appropriate scanner for a particular application depends on the type of application. Terrestrial laser scanners can be categorized mainly on the basis of their distance measurement system. The distance measurement system correlates both to the range and to the accuracy. Therefore, this categorization implies a categorization by range as well as by accuracy. Most of the laser scanners are based on the time of flight (TOF) principle. This technique allows measurements of distances up to several hundred of metres. Even ranges beyond one kilometre are achievable (e.g. Mensi, Riegl). The advantage of long ranges outweighs the disadvantage of lower accuracy in the distance measurement (approx. one centimetre). Apart from the time of flight principle, the phase measurement principle represents the other common technique used for medium ranges. In this case, the range is restricted to one hundred metres (e.g. Zoller+Froehlich, IQSun). In contrast to the time of flight principle, accuracy of the measured distances within a few millimetres is possible. For the sake of completeness, several close range laser scanners with ranges up to few meters are available. But, they are mostly used in industry applications and reverse engineering (online monitoring in construction processes). The used distance measurement principles are laser radar and optical triangulation. Accuracies less than one millimetre (from one-tenth up to one-thousandth of a millimetre) can be achieved. However, these scanners do not fit into the classification of "terrestrial" laser scanners. In addition, classifications by instrumental properties affect a more technical grade.

Differentiations of laser scanners in a technical way can concern − the way of scanning (360 ° scans, scanning specific sections because of limited fields of view, scanning of profiles etc.), the deflection system (sweeping or rotating mirrors), the combination with other devices, mounted on the laser scanner (e.g. camera, GPS), etc.

4.3. A BRIEF OVERVIEW OF THE PRINCIPLES AND METHODS OF LASER SCANNING:

Laser scanning means the deflection of a laser beam by moving (sweeping or rotating) mirrors, the reflection of the laser beam on object surfaces, and the receiving of the reflected laser beam. In opposite to measurements on reflectors, the accuracy of distance measurements depends on the intensity of the reflected laser beam. Physical laws describe the functionality between accuracy and intensity (Gerthsen 1993). Main parameters in these functions are the distance, the angle of incidence, and surface properties (Ingensand et al. 2003). For calibration of total stations, specific procedures were developed in the last years. Instrumental errors as well as angular resolutions and distance accuracies can be defined. Unfortunately, the mechanical design of laser scanners is different to total stations. Most investigations cannot be applied to laser scanners. The measurements in two faces as well as the repetition of single point measurements are impossible. Further, a complicating factor is that most of the manufactures of laser scanners have no experiences in how to construct "geodetic" instruments and how to minimize instrumental errors. This leads to the conclusion that laser scanners have to be investigated from the point of geodetic metrology. Mostly in ways which are optimised for the specific instrument and which are unstandardized.

In our project we used Riegl VZ-400 to serve our purpose (figure 1). It is a 3-D laser scanner that can be configured to use different modes of scanning. Our project mainly involved scanning in line scan mode. It comes with an internal GPS and has provisions for an external GPS too. It also consists of a mountable battery, an LCD screen, a Nikon camera for image acquisition, GPS receiver, antennae and class 1 laser emitter. It comes with a tripod stand for mounting it for field purpose. It brings along with it a software called RiScan pro for viewing and processing the scanned data and acquiring other necessary details from the scans. The socket panel has an input port for PPS and NMEA signals (Trigger socket), two charging ports (figure 2), an ethernet port and an USB port.

Figure 1 – External View of a TLS

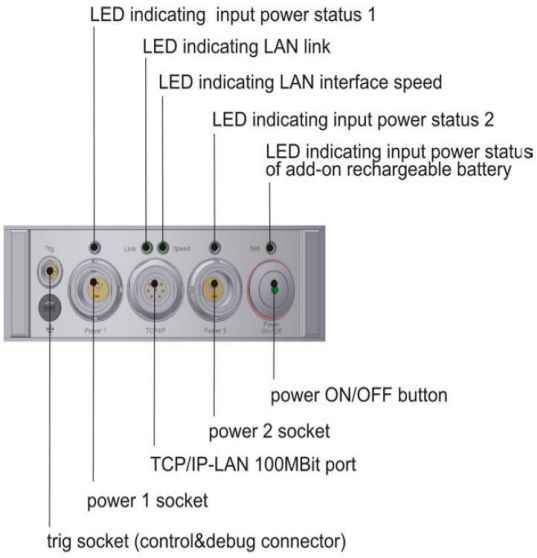


Figure 2- Connection Ports of a TLS

4.4. A BRIEF OVERVIEW OF GPS:

The Global Positioning System (GPS) is a space-based navigation system that provides location and time information in all weather conditions, anywhere on or near the Earth where there is an unobstructed line of sight to four or more GPS satellites. The system provides critical capabilities to military, civil, and commercial users around the world. The United States government created the system, maintains it, and makes it freely accessible to anyone with a GPS receiver. The US began the GPS project in 1973 to overcome the limitations of previous navigation systems, integrating ideas from several predecessors, including a number of classified engineering design studies from the 1960s. The U.S. Department of Defence (DOD) developed the system, which originally used 24 satellites. It became fully operational in 1995. Roger L. Easton, Ivan A. Getting and Bradford Parkinson of the Applied Physics Laboratory are credited with inventing it.

The process of fixing the position, velocity and altitude of a point is called as Trilateration. It generally uses 6 satellites for a precise fix and the accurate time provided by the satellites is maintained by a highly precise atomic clock on board the satellites which is further verified by a quartz crystal clock inside the GPS receiver.

We used an IRNSS/GPS/SBAS receiver which is an indigenous GPS receiver made by Accord Software and Systems in India. It comes with a monitor (figure 3), a receiver, a battery charger, an AC-DC adapter, a DC-DC adapter, several input and output ports and different wires for connection.

Figure 3 – External View of an IRNSS Receiver

Chapter – 5:
HARDWARE BASED APPROACH FOR TIME SYNCHRONISATION

5.1. EXTRACTION OF TIMESTAMPS FROM '.rxp' FILE:

We acquired the necessary hardware devices and started studying the respective user manuals. We began working with the IRNSS/GPS/SBAS receiver first and went for its field test and found that it was working fine and was receiving data properly from satellites. It even gave the information regarding its fix (figure 3a).

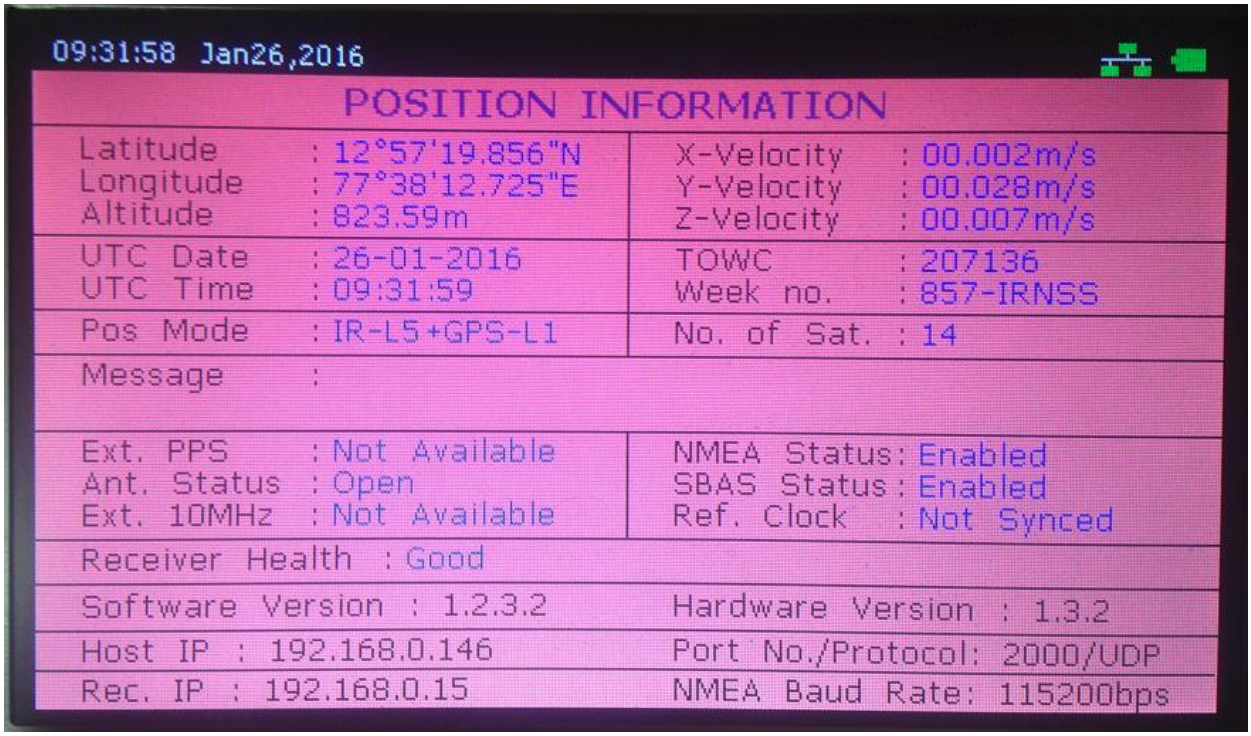


Figure 4 – Human Machine Interface of IRNSS Receiver

After that we started field testing the TLS. We took around 20 scans both indoor and outdoor, and processed the '.rxp' file generated by the TLS on the RiScan-Pro software (figure4). RiScan-Pro has the flexibility of obtaining data both in the Scanner's Own Coordinate System(SOCS) and the Global Coordinate System (GLCS) which follows the World Geodetic System 1984(WGS 84). But the problem with the internal GPS is that the time data from it is inaccurate as compared to IRNSS receiver. For our

convenience, we exported the linear space coordinates (x, y, z) and angular coordinates in SOCS or in GLCS from the processed data along with their I.D. numbers and time stamps as we needed to understand the time stamp pattern of the acquired data in order to synchronise it. But when we exported the data to an ASCII file we found that the timestamps written on to the file were irregular in nature and there was no particular pattern in them. The timestamps started from a particular value which was not zero, increased in magnitude, came back to a near about value after a certain number of scanned points, again increased for a different number of points and once again fell back to a similar value. This behaviour existed throughout the file and lacked any kind of regularity (figure 5, 6, 7).

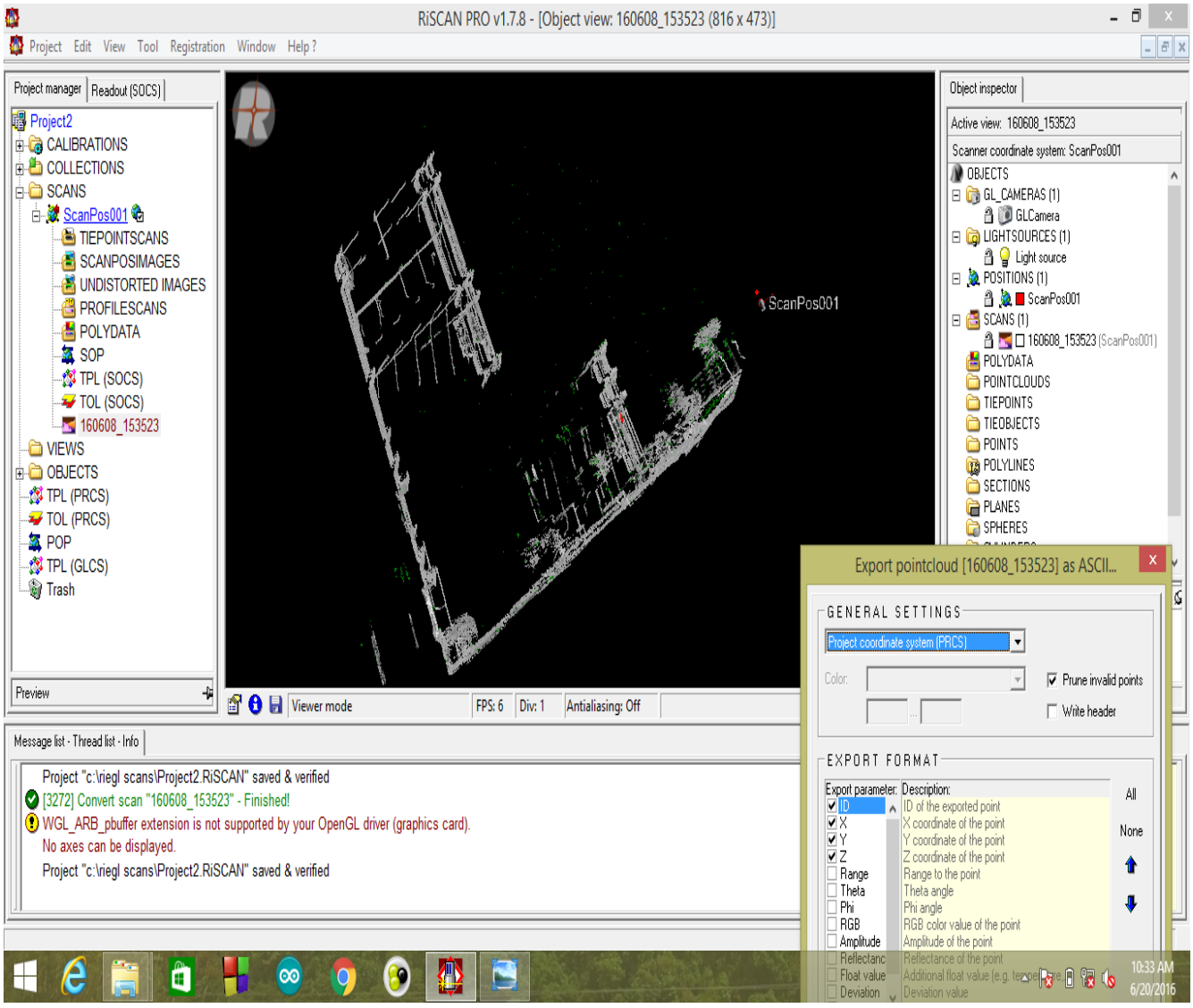


Figure 5 – Screen Capture of RiScan Pro

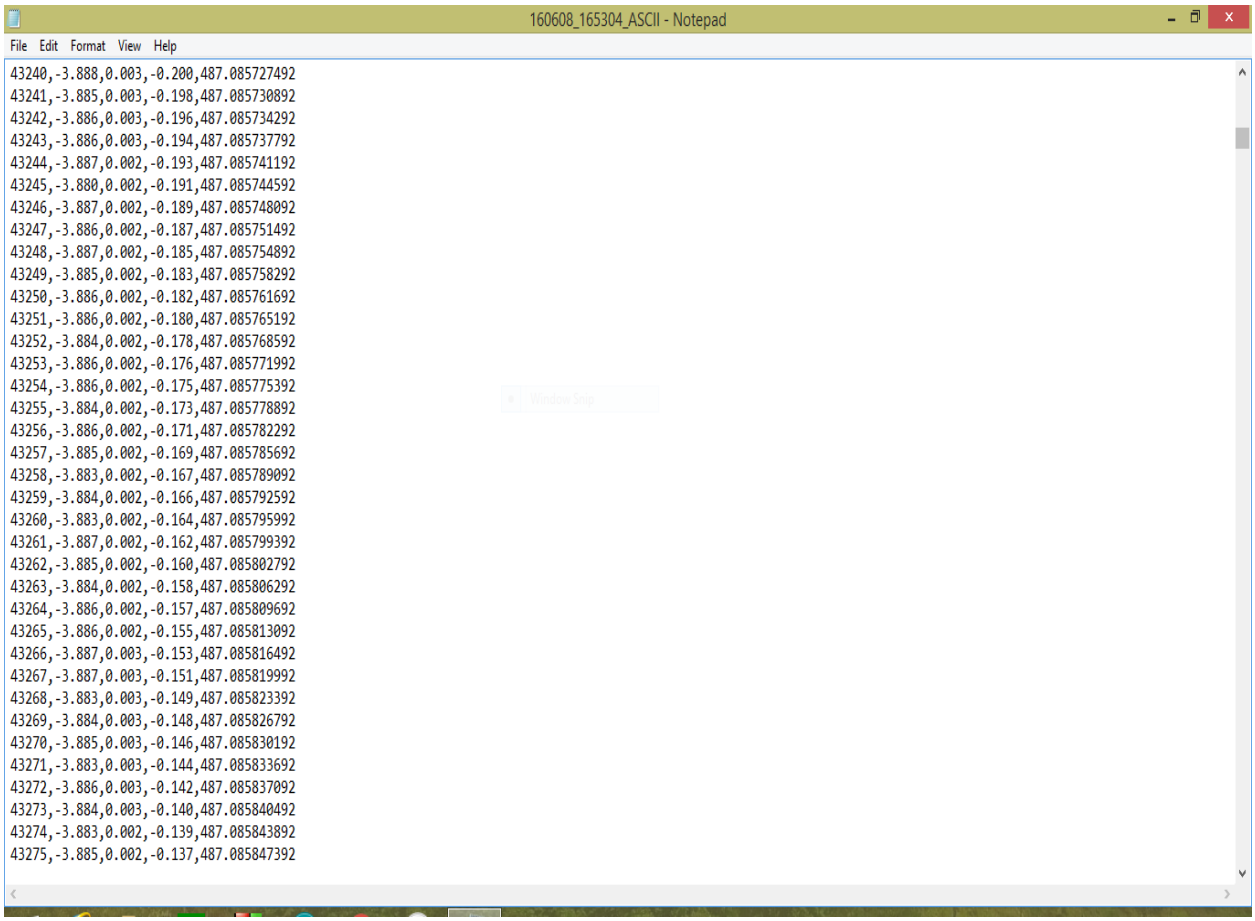


Figure 6 – Screen Capture of Text File Exported by RiScan Pro

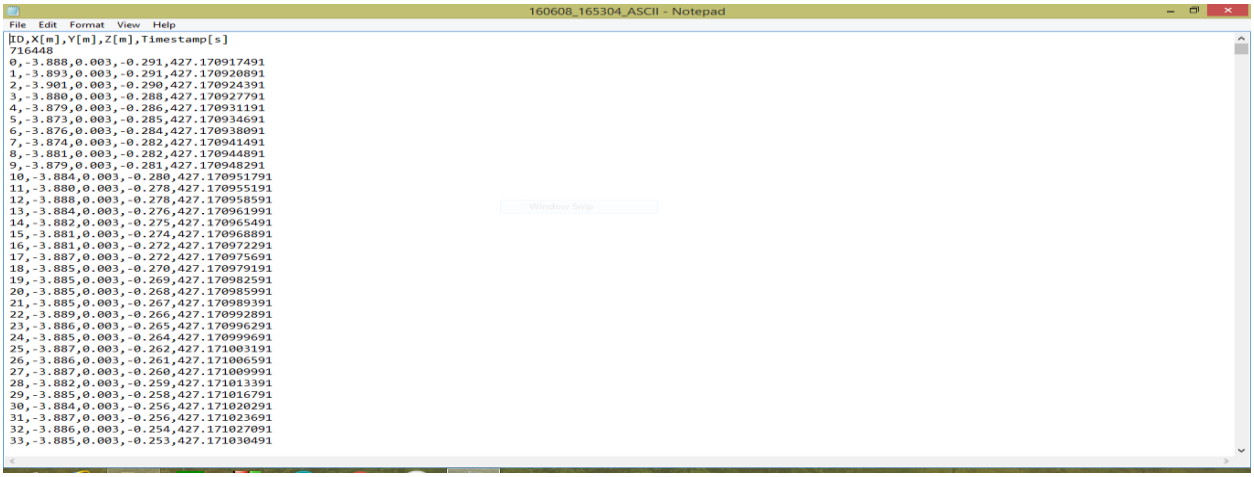


Figure 7 - Screen Capture of Text File Exported by RiScan Pro

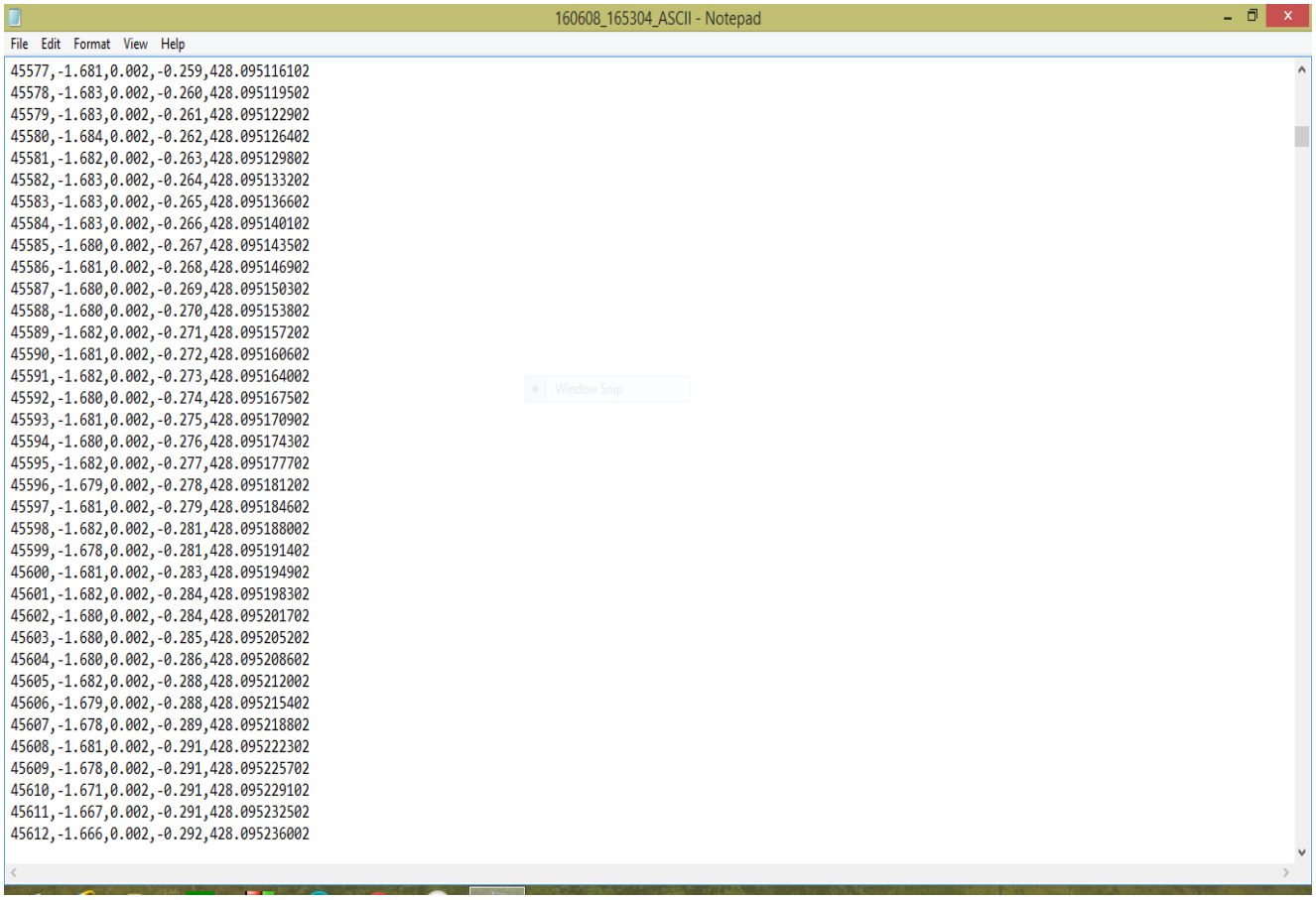


Figure 8 - Screen Capture of Text File Exported by RiScan Pro

If we compared the fourth column of figure 5, 6, 7 it was observed that the timestamp first increased and then decreased. The timestamps appeared to follow a more or less circular pattern. But even if they were following a circular pattern then they should have started from 000.0000 seconds when they started looping again. The initial deviation in seconds (in this case is 427.05 seconds) is also not the epoch time. We roughly guessed that is difference in the time of starting the scan and the time of the booting the scanner. This seemed to be a failed attempt to extract the timestamps and so we decided to hard connect the TLS with the GPS receiver and synchronise it.

5.2. SYNCHRONISATION BY HARD CONNECTION BETWEEN GPS RECEIVER AND TLS:

We connected the GPS receiver with computer and logged the GPS fix data onto the IRNSS-UR GUI. We first match the Subnet Mask, IP address domain of the receiver with the same of the computer (figure 8). We selected Transmission Control Protocol (TCP) at one time and User Datagram Protocol (UDP) at the

other for communication over Ethernet cable and RS232 for serial communication. The GPS receiver transmits NMEA-0183 messages.

We needed GPZDA and GPGGA Talker ID for getting position and time information respectively (figure 8, 9, 10, 11). We selected the Baud Rate as per the TLS configuration and connected the GPS receiver to the TLS (figure 12).

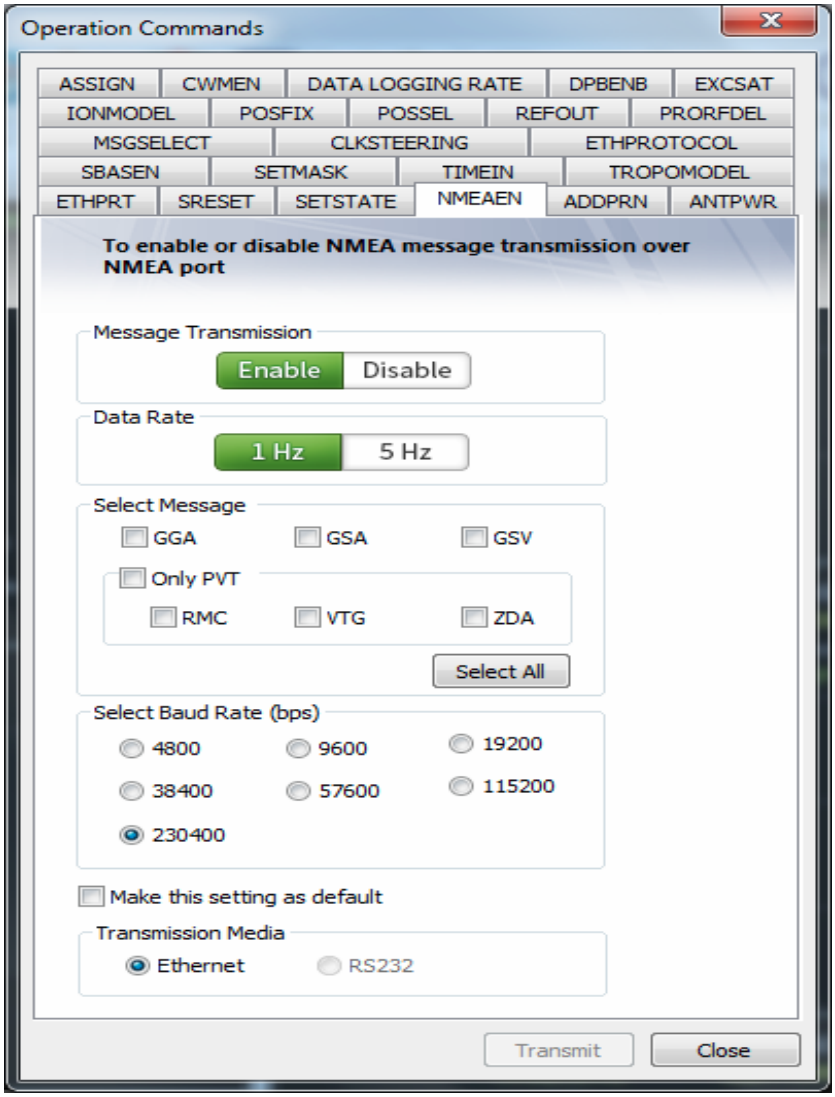


Figure 9 – Intra GUI Command Panel of IRNSS Receiver

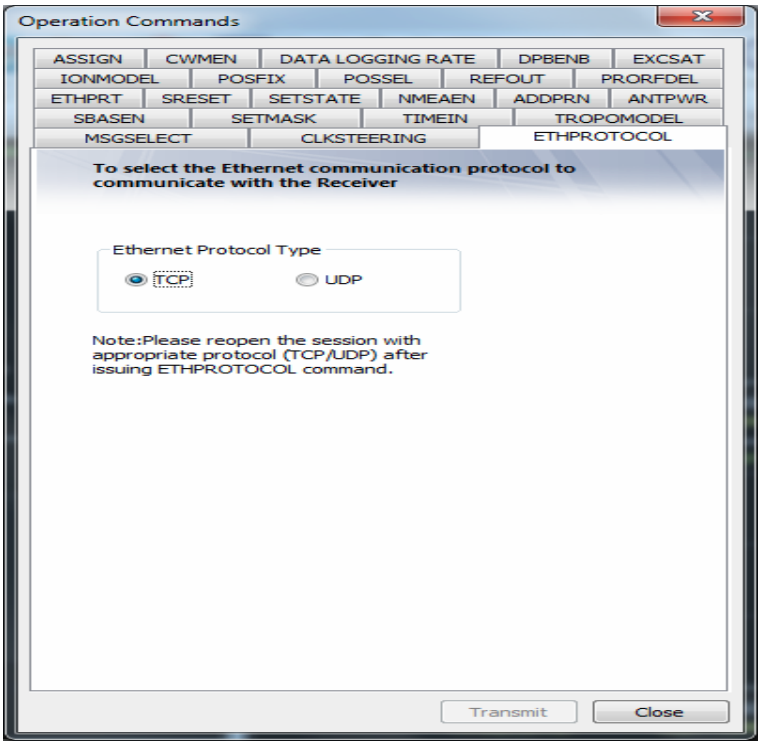


Figure 10- Screen Capture of Text File Exported by RiScan Pro

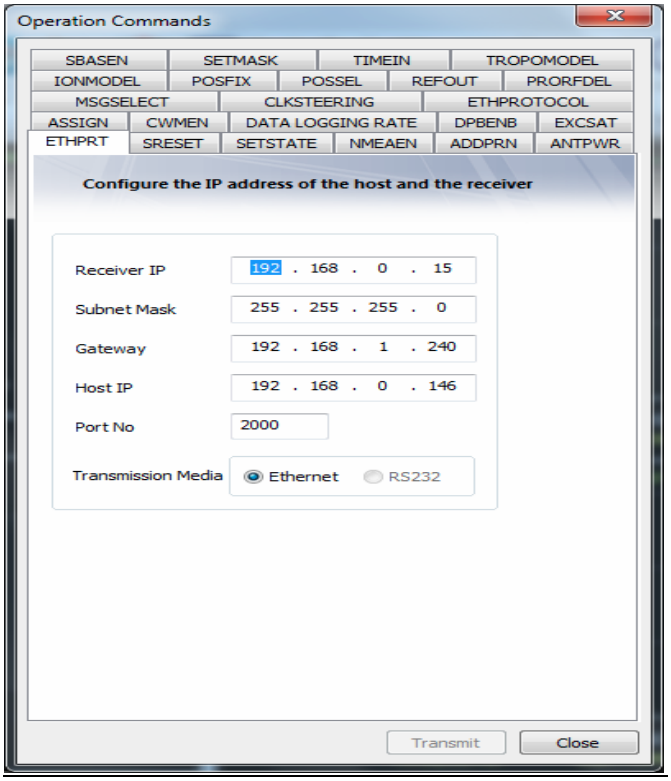


Figure 11 Screen Capture of Text File Exported by RiScan Pro

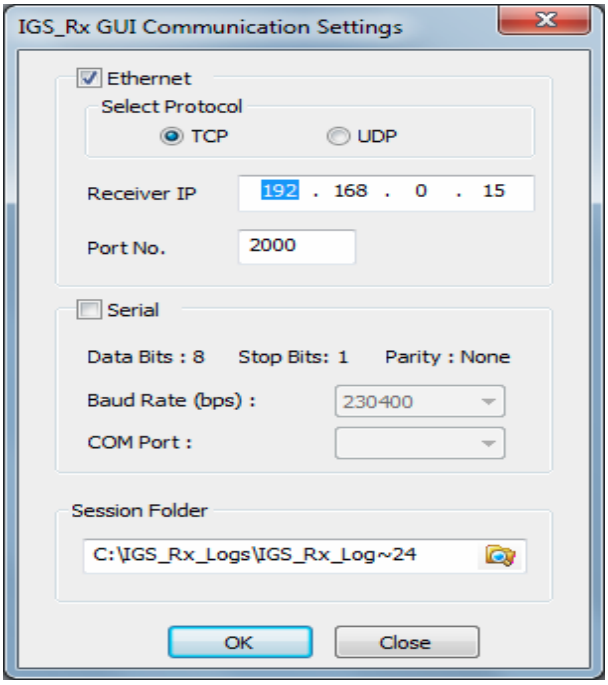


Figure 12 Screen Capture of Text File Exported by RiScan Pro

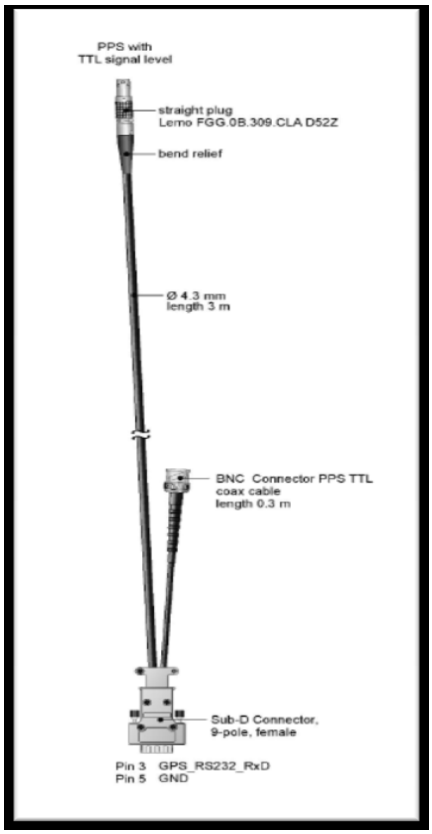


Figure 13- A Single-Port-To-Dual-Port Connector

After connecting, the TLS Human Machine Interface (HMI) or the display panel was supposed to be showing 'Synchronisation: OK' as shown in figure 13 but unfortunately it didn't show the same. We tried several other configurations like external GPS RS 232, external GPS top or external PPS receiver but in vain (figure 14).

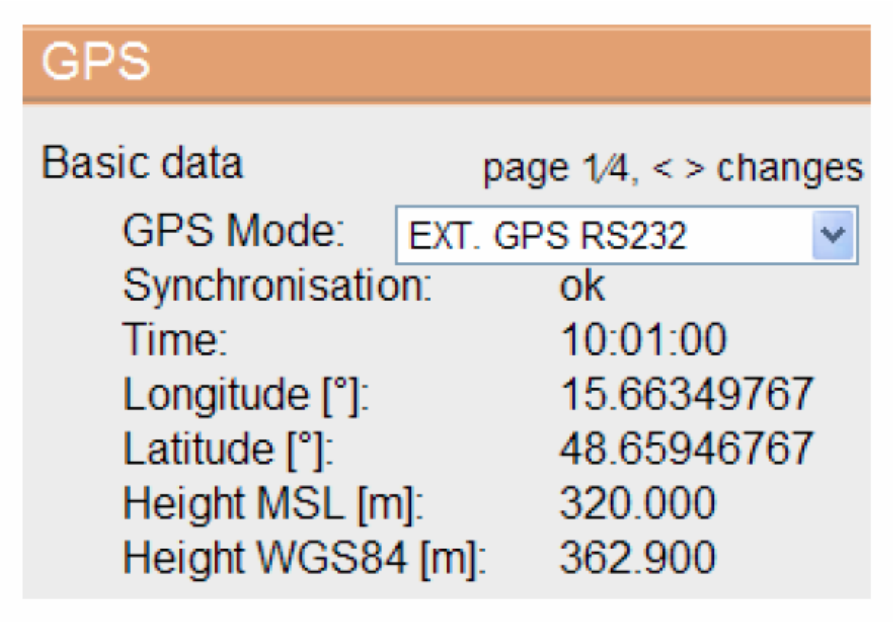


Figure 14 – HMI of TLS

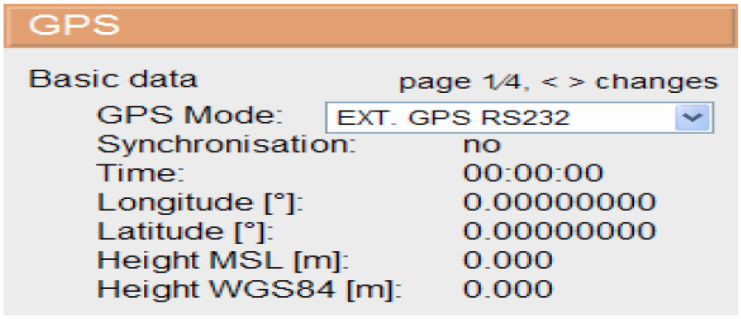


Figure 15 – HMI of TLS

We had done the connections by following the hardware manuals precisely and still failed to hard synchronise the TLS and the receiver. So we decided to switch over to the software approach.

Chapter – 6:
SOFTWARE BASED APPROACH FOR TIME SYNCHRONISATION

As per the instructions of our guide we downloaded and started exploring RivLib libraries. There were various versions of libraries available pertaining to different C/C++ compilers (gcc, mingw), processors(x86, ARM) and Operating Systems (Windows, Linux). After downloading quite a few versions of RiVLib, we found out that a particular version 'rivlib-2_3_0-x86_64-windows-mgw49' best suited our system. We started adding appropriate paths in the '# include <>' statement within the header files namely ctrlifc.h, ctrlib.h, pointcloud.h, ridataspec.h, rxpmarker.h, rxpstream.h , scanifc.h, scanlib.h, pointsifc_t.h etc.There were examples provided in the library which used the static header files and a few '.lib' files and '.dll' files for compiling .We built all those files into a '.proj' package using CMake .Before doing this ,we even linked Code::Blocks IDE on our system to the required library files using Linker Settings. Apparently, problem with the library was that the examples couldn't individually build and the error log showed error messages talking about undefined reference, the crux of which was that the examples couldn't find the implementations of the functions in the header files (See Figure 15) .Even the '.proj' package made from the example files on CMake didn't run due to some error though we have built it by following the instructions given in the 'CMakeLists.txt' file to the T. Moreover there were some inexplicable errors in the example code related to the namespace 'sharedptr'. Repeatedly we failed in our attempt to the build the individual example files and run the complete CMake Package built from the examples.

After consulting our guide, we wrote a detailed mail to the Riegl support team regarding our problem. There was a Delphi Code given in the library which converted an '.rxp' file into a '.txt' file containing different attributes of the scanned points. Unfortunately, the Delphi code also failed to run even after installing a one month trial version of a Delphi Editor named 'Rad Studio' by Embarcadero. Delphi is basically evolved Pascal with more advanced functionality and implementations.

6.1. SOLUTION TO OUR PROBLEMS:

After working on this for two long weeks, we came up with ideas both for timestamp extraction and time synchronization of the GPS Receiver with the TLS.
We decide to do the following-

i) Setup a servers on a PC which will connect to both scanner and GPS Receiver. We decided to feed the PC with the time from the GPS and feed the time to the TLS. This seemed to be a credible and accurate approach for time synchronisation because GPS time is known to be very precise to ten-sixth of a second as the time is maintained by crystal and atomic clocks.

ii) For time stamp extraction we decided to write our own code, as suggested by our guide, to extract the timestamps in a sorted order same as that of the points scanned, as opposed to the timestamps exported by RiScan-Pro in the form of a text file which were completely out of order. The main impediment in this approach was that the '.rxp' was a very hidden and editor selective type of format which refused to open in any other editor. So, extracting data from such a file format seemed to be quite difficult.

6.2. EXTRACTION OF TIMESTAMPS FROM '.rxp' FILE AND AUTOMATIC INITIATION OF SCANS USING C CODE:

After continuously working for about a week we finally came up with our own code. Writing the code was not easy particularly because the library was quite complicated and heavy. It comes with multiple static header files '.h' / '.hpp', '.lib' and dynamic '.dll' files. Each static header file had link to some other header file and the complete understanding of all these static header files cost us a lot of time.

We wrote two pieces of code:

i) A C code for time stamp extraction both from a '.rxp' file and live data streaming from the scanner.

ii) A C++ code for automatically initiating a scan and setting initial parameters for the scan.

6.2.1. Code #1 – C Code for Scan Attributes Extraction

The first code was a C code is meant for timestamp extraction along with other attributes like local coordinates of the points scanned , the pps sync bit of the points and the 'end of frame' bit. The code mainly used the 'scanifc.h' header file. The code self-builds a text file by the same name as the rxp file but with a '.txt' extension in place of '.rxp' and writes all these parameters onto the text file in the order of their scanning. The timestamps obtained on the text file are also supposed to be in increasing order, same as that of scanning.

6.2.1.1. C Code

```
/* START OF CODE */

/* This example uses the RiVLib as a DLL, so do not forget to copy
the scanifc.dll to a place where it can be found ( usually at the
same location as the executable when on windows). */

#include <riegl/scanifc.h>
#include <stdio.h>
#include <string>
#include <iostream>

using namespace std;

void print_last_error()
{
        char buffer[512];
        scanifc_uint32_t bs;
        scanifc_get_last_error(buffer,512,&bs); // actual size of error -> *bs , 512 size of buffer .. error truncated to 512 if longer than 512
        printf("%s\n",buffer);
}

#define N_TARG 1000000 // no of points per call
scanifc_xyz32       rg_xyz32[N_TARG];
scanifc_attributes  rg_attributes[N_TARG];
scanifc_time_ns     rg_time[N_TARG];


int main(int argc,char* argv[]) //argv[1] uri of the rxp stream
```

```
{
point3dstream_handle h3ds=0; //handle for data stream
int sync_to_pps=0; // since reading data with pps timestamp
scanifc_uint32_t count;   // type uint32_t
scanifc_uint32_t n;
int end_of_frame;
unsigned long int numbr_of_points=0;
            //scanifc_xyz32      rg_xyz32[N_TARG];    //scanifc_xyz32 -> structure having x,y,z coordinates
             //scanifc_attributes  rg_attributes[N_TARG]; //scanifc_attributes-> structure having attributes like amolitude,deviation,reflectance
             //scanifc_time_ns    rg_time[N_TARG];     // timestamp havin exrernal gps since sync_to_pps=1
if (argc==2)
{
        if (scanifc_point3dstream_open_with_logging(argv[1],sync_to_pps,"log.rxp",&h3ds))
            {  // !=0 -> failure...
                fprintf(stderr,"Cannot open: %s\n",argv[1]);
                print_last_error();
                return 1;
            }
         else
            {
                if (scanifc_point3dstream_add_demultiplexer(h3ds,"hk.txt",0,"status protocol")) //no package names, class names - 'status','protocol'
                    {
                        print_last_error();
                        return 1;
                    }
                string p(argv[1]);
                int i = p.find_last_of(".");
                p = p.substr(0,i);
                p = p + ".txt";
                FILE *f=fopen(p.c_str(),"w");
                do
                  {
```

```
            if
(scanifc_point3dstream_read(h3ds,N_TARG,rg_xyz32,rg_attributes,rg_time,&count,&end_of_frame))
                {
                   fprintf(stderr,"Error reading %s\n",argv[1]);      //count -> no of points actually read returned in count
                   return 1;
                 }
            for (n=0;n<count;++n)
                {
                  fprintf(f,"%f,%f,%f,%u,%lu,%d\n",rg_xyz32[n].x,rg_xyz32[n].y,rg_xyz32[n].z,(rg_attributes[n].fl
                  ags & 0x40)>>6,rg_time[n],end_of_frame);
                }
            numbr_of_points+=n;
          } while(count >0); // reads 'count' no of points
        fprintf(f,"The number of points obtained from rxp :%lu",numbr_of_points); //writing the number of points to the file

        if (scanifc_point3dstream_close(h3ds)) //closing the file handle
          {
             fprintf(stderr,"Error,closing %s\n",argv[1]);
             return 1;
           }
        fclose(f);
        return 0;
        }
}

fprintf(stderr,"Usage:%s <input-uri>\n",argv[0]);
return 1;
}
```

/* END OF CODE */

We could not build the code on Windows because we faced the same problems as mentioned earlier. The coder couldn't bind the dynamic libraries required and couldn't find the function implementations which are found in the '.a' files. So we decided to build the code on Linux Distribution Ubuntu 14.04 using command line arguments. We required CMake for building the code. Finally we used the RiVLib library version 'rivlib-2_3_0-x86_64-linux-gcc44'. We built the code into an 'attributes_output.exe' file for Ubuntu. It is also essential to put the file 'scanifc.dll' in a directory accessible by the code preferably the same directory as the code.

6.2.1.2. Building the Code on Ubuntu

To build the code on Ubuntu from the command line terminal, make the following changes and installations –

> No significant changes are required, if any modifications are needed for the particular system on which the code is being built, it is advisable that they are done in accordance with official Ubuntu documentation.

After that, execute the following commands from the terminal window-

> **g++ -g -std=c++11 attributes_output_linux.c -o attributes_output_linux -I/home/user/rivlib-2_3_0-x86_64-linux-gcc44/include -L/home/user/rivlib-2_3_0-x86_64-linux-gcc44/lib -lscanifc-mt -lscanlib-mt**

6.2.1.3. Running the Executable on Ubuntu

To build the code on Ubuntu from the command line terminal, make the following changes and installations –

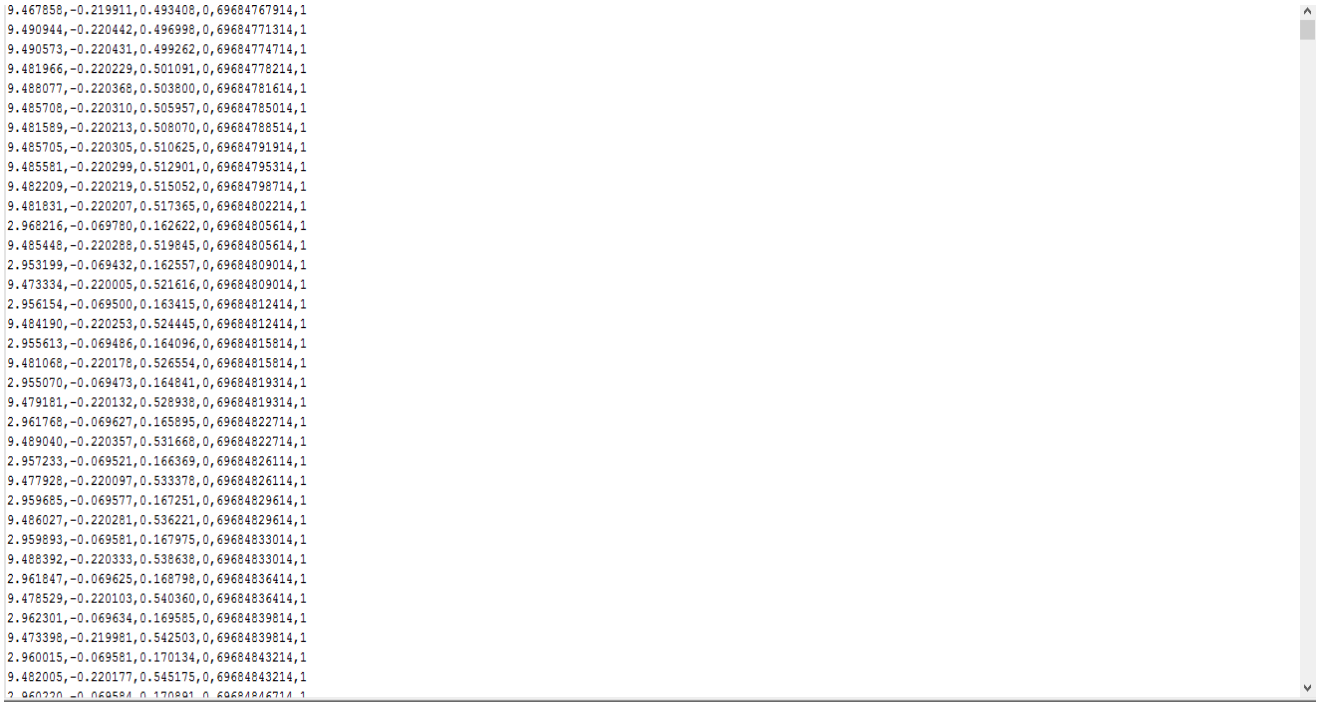
> No significant changes are required, if any modifications are needed for the particular system on which the code is being built , it is advisable that they are done in accordance with official Ubuntu documentation.

After that, execute the following commands from the terminal window-

> **g++ -g -std=c++11 attributes_output_linux.c -o attributes_output_linux -I/home/user/rivlib-2_3_0-x86_64-linux-gcc44/include -L/home/user/rivlib-2_3_0-x86_64-linux-gcc44/lib -lscanifc-mt -lscanlib-mt**

6.2.1.4. Output of Code

Finally we tested the code on laser scans to see if the code produced proper output in the form of a text file. We tested the code both on old scans taken in Panorama or Line Scan mode and even took new scans for testing. The code worked perfectly on all the scans and produced output having information about the local coordinates of each point , their timestamp, the value of their pps sync bits, the value of their end of frame-bits.



```
9.467858,-0.219911,0.493408,0,69684767914,1
9.490944,-0.220442,0.496998,0,69684771314,1
9.490573,-0.220431,0.499262,0,69684774714,1
9.481966,-0.220229,0.501091,0,69684778214,1
9.488077,-0.220368,0.503800,0,69684781614,1
9.485708,-0.220310,0.505957,0,69684785014,1
9.481589,-0.220213,0.508070,0,69684788514,1
9.485705,-0.220305,0.510625,0,69684791914,1
9.485581,-0.220299,0.512901,0,69684795314,1
9.482209,-0.220219,0.515052,0,69684798714,1
9.481831,-0.220207,0.517365,0,69684802214,1
2.968216,-0.069780,0.162622,0,69684805614,1
9.485448,-0.220288,0.519845,0,69684805614,1
2.953199,-0.069432,0.162557,0,69684809014,1
9.473334,-0.220005,0.521616,0,69684809014,1
2.956154,-0.069500,0.163415,0,69684812414,1
9.484190,-0.220253,0.524445,0,69684812414,1
2.955613,-0.069486,0.164096,0,69684815814,1
9.481068,-0.220178,0.526554,0,69684815814,1
2.955070,-0.069473,0.164841,0,69684819314,1
9.479181,-0.220132,0.528938,0,69684819314,1
2.961768,-0.069627,0.165895,0,69684822714,1
9.489040,-0.220357,0.531668,0,69684822714,1
2.957233,-0.069521,0.166369,0,69684826114,1
9.477928,-0.220097,0.533378,0,69684826114,1
2.959685,-0.069577,0.167251,0,69684829614,1
9.486027,-0.220281,0.536221,0,69684829614,1
2.959893,-0.069581,0.167975,0,69684833014,1
9.488392,-0.220333,0.538638,0,69684833014,1
2.961847,-0.069625,0.168798,0,69684836414,1
9.478529,-0.220103,0.540360,0,69684836414,1
2.962301,-0.069634,0.169585,0,69684839814,1
9.473398,-0.219981,0.542503,0,69684839814,1
2.960015,-0.069581,0.170134,0,69684843214,1
9.482005,-0.220177,0.545175,0,69684843214,1
2.960220,-0.069584,0.170891,0,69684846714,1
```

Figure 16 – Parts of the Text File Produced From '.rxp' File

At the end of the file the total number of points scanned is getting printed. We also notice that the difference between two consecutive points in the text file is 3400 and all the timestamps are arranged in increasing order of magnitude. Though the magnitude of the timestamp is peculiar because of its 15 bit decimal value and it is not the epoch or UNIX time as measured from January 1st, 1970 without the leap seconds. Another noticeable characteristic in the text file was that there was a difference of 3400 units between 2 consecutive points which showed that it was directly related to the time rate of scanning of the points. It also highlighted the fact that the points were being scanned by the scanner at a constant frequency. But both the timestamps and the time difference between two consecutive points was somewhat large in magnitude which may be because of some internal factor of multiplication by the scanner.

6.2.2. Code #2 - Code to Synchronise With GPS, Initiate and Continue Scan until the Data Limit Is Reached

The second code was written so as to automatically terminate a previous scan and start a new scan only if the scanner whose identity given as command line argument is connected to the system(computer in this case).
The program forces the scanner to abort the present scan and start a new scan with the range horizontal scan angle range set to 180° and the vertical angle range set to (30° to 130°) with a step increment of 0.9875°. It also sets various parameters related synchronisation

i) The Talker ID of NMEA Communication Protocol or GPS External Format (GPZDA in this case),

ii) The Baud Rate or External Communication Baud Rate (115200 bps in this case),

iii) GPS Mode (set to External GPS via RS232 in this case),

iv) GPS data sequence or GPS External Sequence (set to PPS FIRST in this case) ,

v) GPS External Edge (set to positive in this case).

The scan data is logged to a '.rxp' file given in the second command line argument. The first command line argument gives the URI (Uniform Resource Identifier) of the scanner in the form of <host_name_or_ip_address>:<service_name_or_port_number>.

Examples -

"192.168.0.234", "192.168.0.234:20002",

"s9991234" or "s9991234:20002"

Before the scan starts internal storage is enabled and a connection between the scanner and the system is established. Once the pre-set data limit is reached , the scan is terminated by the program and the connection is closed. The data is logged onto a binary file and finally stored as a '.rxp' file.

6.2.2.1. C++ Code

```
/* START OF CODE */

#include <C:\Users\Sagnik\Desktop\PS1\rivlib-2_3_0-x86_64-windows-mgw49\include\riegl\ctrllib.hpp>
#include <C:\Users\Sagnik\Desktop\PS1\rivlib-2_3_0-x86_64-windows-mgw49\include\riegl\scanlib.hpp>
#include <iostream>
#include <fstream>
#include <exception>
#include <stdexcept>
```

```
#include <cmath>
#include <algorithm>
#include <memory>

using namespace ctrllib;
using namespace scanlib;
using namespace std;
int main(int argc,char* argv[]) //3rd argument , argv[2] -> output filename ;argv[0] -> tls s no / ip addrs ; argv[1] -> port no of external gps
{

        if ((argc!=3)||(string(argv[1])=="--help"))
        {
           cerr<<"Usage:"<<argv[0]<<" address[:service] outputfile"<<endl;
           return 1;
        }
ctrl_client_session session;
try
  {
          // now connect to the instrument specified as first argument
          // the connection string is expected in format
          //   "<host_name_or_ip_address>[:<service_name_or_port_number>]"
          // Examples:
          //   "192.168.0.234", "192.168.0.234:20002",
          //   "s9991234" or "s9991234:20002"
          // if port number is not specified, 20002 is assumed as default

   session.open(argv[1]);
   cout<<"Connected to "<<argv[1]<<endl;
   string result;
   vector<string> comment;
   string status;
   session.execute_command("MEAS_ABORT","",&result,&status,&comment);// telling the TLS to abort the current session
   cout<<"MEAS_ABORT() returned "<<result<<endl;
  for(int i=0;i<comment.size();i++) //for(int i=0 ; i<comment.size();i++)
      cout<<comment[i]<<endl;
 cout<<"Status is "<<status<<endl;
session.execute_command("MEAS_BUSY","1",&result,&status,&comment);
```

```
cout<<"MEAS_BUSY(1) returned "<<result<<endl;//waiting for the TLS to terminate the previous session
for(int i=0;i<comment.size();i++)//for(size_t i=0; i<comment.size(); ++i)
    cout<<comment[i]<<endl;
cout<<"Status is "<<status<<endl;
string value;
session.get_property("INST_IDENT",value);
cout<<"Instrument is "<<value<<endl;

                //
if (std::string(value).compare(1,6,"VZ-400")!=0)
  {
   throw runtime_error("Error:VZ-400 instrument required");
   }

if (std::string(value).compare(1,6,"VZ-400")==0)//if (std::string(value).compare(1, 6, "VQ-250") == 0)
  {
   session.set_property("GPS_MODE","2");
   cout<<"GPS_MODE set to 2 (external GPS via RS232)"<<endl;
   session.set_property("GPS_EXT_COM_BAUDRATE","6");
   cout<<"GPS_EXT_COM_BAUDRATE set to 115200"<<endl;
   session.set_property("GPS_EXT_EDGE","0");
   cout<<"GPS_EXT_EDGE set to (0=positive,1=negative)"<<endl;

   session.set_property("GPS_EXT_FORMAT", "\"GPZDA\"");
   cout<<"GPS_EXT_FORMAT set to GPZDA"<<endl;
   session.set_property("GPS_EXT_SEQUENCE", "\"PPS_FIRST\"");
   cout<<"GPS_EXT_SEQUENCE set to PPS_FIRST"<<endl;
  }
  session.get_property("MEAS_PROG",value); //current measurement program
  cout<<"MEAS_PROG is "<<value<<endl;
  session.execute_command("MEAS_SET_PROG","1",&result,&status,&comment); // set measurement program
  cout<<"MEAS_SET_PROG(1) returned "<<result<<endl;
```

```
    for(int i=0;i<comment.size();i++)
        cout<<comment[i]<<endl;
    cout<<"Status is "<<status<<endl;
    session.execute_command("SCN_SET_SCAN","30, 130, 0.9875, 180",&result,&status,&comment);
    cout<<"SCN_SET_SCAN(30, 130, 10) returned "<<result<<endl;
    for(int i=0;i<comment.size();i++)
         cout<<comment[i]<<endl;
    cout<<"Status is "<<status<<endl;
    session.execute_command("MEAS_START","",&result,&status,&comment); //line scan details : starting theta,
theta_end,increment=0.09875 , phi(alignment)=180
    cout<<"MEAS_START() returned "<<result<<endl;
    for(int i=0;i<comment.size();i++)
        cout<<comment[i]<<endl;
    cout<<"Status is "<<status<<endl;

                                    /*acquirirng data*/
    cout<<"Output:"<<argv[2]<<endl;
    ofstream outfile(argv[2],ios::binary|ios::out);//binary file created in output mode

                    /* opening input connection to scanner: */

    bool shutdown_requested=false;
    string connstr(argv[1]);//string connstr(argv[1]);
    connstr="rdtp://"+connstr+"/current";
    cout<<"Input: "<<connstr<<endl;
    std::shared_ptr<scanlib::basic_rconnection> rc;
    rc = basic_rconnection::create(connstr);
    rc->open();
    basic_rconnection::size_type limit=50*1024000;  //basic_rconnection::size_type limit = 50*1000000;
     basic_rconnection::size_type count;
     basic_rconnection::size_type total=0;
     char buffer[8192];  //1 kB of char
     int last_percent=0;
     int data_percent;
```

```
cout<<"Data limit:"<<limit<<" bytes"<<endl;

cout<<"Percent:"<<endl;

cout<<"1..10...20...30...40...50...60...70...80...90..100"<<endl;


            /* loop runs as long as the instrument delivers data */

while(0!=(count=rc->readsome(buffer,8192)))

        {
          outfile.write(buffer,count);
          total+=count;
          if((total>limit)&&(!shutdown_requested))
             {
                  /*requesting shutdown of data delivery*/
              rc->request_shutdown();
              shutdown_requested=true;
             }
          data_percent= min(100,int(ceil(total*100.0/limit)));

                /*updating progress bar*/
          while(last_percent<data_percent)
                 {
                  cout<<"*";
                  last_percent += 2; // 1 '*' = 2 percent
                 }
        }

cout<<endl;


outfile.flush(); // flushing the '.rxp' output file

outfile.close(); // closing the '.rxp' output file


cout<<total<<" bytes received"<<endl;


                    /*stopping measurement:*/

session.execute_command("MEAS_STOP","",&result,&status,&comment);

cout<<"MEAS_STOP() returned "<<result<<endl;

for(int i=0;i<comment.size();i++)

    cout<<comment[i]<<endl;

    cout<<"Status is "<<status<<endl;


            /* the connection is closed if the instrument is no longer needed */
```

```
        session.close();  // closing the connection
        cout << "Connection closed" << endl;
        cout<<"Demo program successfully finished"<<endl;
}
catch(exception& e)
{
        cerr<<e.what()<<endl;
        return 1;
}
catch(...)
{
        cerr<<"unknown exception"<<endl;
        return 1;
}
return 0;
}
```

/* END OF CODE */

6.2.2.2. Building the Code on Ubuntu

To build the code on Ubuntu from the command line terminal, make the following changes and installations –

No significant changes are required , if any modifications are needed for the particular system on which the code is being built , it is advisable that they are done in accordance with official Ubuntu documentation.

After that, execute the following commands from the terminal window-

g++ -g -std=c++11 gps_time_sync_2.cpp -o gps_time_sync_2 -I/home/user/rivlib-2_3_0-x86_64-linux-gcc44/include -L/home/user/rivlib-2_3_0-x86_64-linux-gcc44/lib -lscanifc-mt -lscanlib-mt -lctrllib-mt -lpthread

6.2.2.3. Running the Executable on Ubuntu

To run the executable on Ubuntu from the command line terminal, make the following changes and installations -

No significant changes are required, if any modifications are needed for the particular system on which the code is being built, it is advisable that they are done in accordance with official Ubuntu documentation.

After that, execute the following commands from the terminal window-

To run the code, mention the name of the executable followed by the URI (Uniform Resource Identifier) of the scanner having the form

<host_name_or_ip_address>:<service_name_or_port_number>

and the name of '.rxp' file where the data is to be logged

./gps_time_sync_2 <host_name_or_ip_address>:<service_name_or_port_number> <filename>

Examples-

<host_name_or_ip_address>:<service_name_or_port_number>:
"192.168.0.234" OR "192.168.0.234:20002",

"s9991234" OR "s9991234:20002"

<filename>:"scan1.rxp"

6.2.2.4. Output of Code

The scan data is logged on to the '.rxp' file the name of which has been given as a command line argument.

6.2.3. Code #3 – Combined Code to Synchronise with GPS, Initiate and Continue Scan until the Data Limit Is Reached and Extract the Scan Attributes From The '.rxp' File or Live Data Stream

This combined code was written so that it singlehandedly performs the twin tasks of the previously mentioned C and C++ code. It first synchronizes the GPS with the TLS, terminates a previously running scan, initiates a new scan and completes the scan, finally logging the data onto an '.rxp' file. After that, the later part of the code extracts the attributes from the previously formed '.rxp' file.The program is wriiten in the form of C file aimed at extracting the scan attribute data from a '.rxp' file which uses a C++

code in the form of a header file which actually synchronises with the GPS, initiates a scan, completes it and logs a data to '.rxp' file to be used by the C code using the header.

6.2.3.1. C Code and C++ Header

```
/* Start Of Code */


/* This example uses the RiVLib as a DLL, so do not forget to copy
the scanifc.dll to a place where it can be found ( usually at the
same location as the executable when on windows). */

#include <riegl/scanifc.h>
#include "C:\Users\Sagnik\Desktop\PS1\c++\gps_sync_header.hpp"
#include <stdio.h>
#include <string>
#include <iostream>

using namespace std;

void print_last_error()
{
	char buffer[512];
	scanifc_uint32_t bs;
	scanifc_get_last_error(buffer,512,&bs); // actual size of error -> *bs , 512 size of buffer .. error truncated to 512 if longer than 512
	printf("%s\n",buffer);
}

#define N_TARG 1000000 // no of points per call
scanifc_xyz32       rg_xyz32[N_TARG];
scanifc_attributes  rg_attributes[N_TARG];
```

```
scanifc_time_ns     rg_time[N_TARG];



int main(int argc,char* argv[]) //argv[1] uri of the rxp stream
{
point3dstream_handle h3ds=0; //handle for data stream
int sync_to_pps=0; // since reading data with pps timestamp
scanifc_uint32_t count;   // type uint32_t
scanifc_uint32_t n;
int end_of_frame;
unsigned long int numbr_of_points=0;
int flag=1;
              //scanifc_xyz32       rg_xyz32[N_TARG];     //scanifc_xyz32 -> structure having x,y,z coordinates
               //scanifc_attributes  rg_attributes[N_TARG]; //scanifc_attributes-> structure having attributes like amolitude,deviation,reflectance
               //scanifc_time_ns     rg_time[N_TARG];      // timestamp havin exrernal gps since sync_to_pps=1


while(flag==1)
      {

      flag=gps_sync(argc,argv);
      if(flag==0)
         break;
      }
if (argc==4)
{
          if (scanifc_point3dstream_open_with_logging(argv[1],sync_to_pps,"log.rxp",&h3ds))
              {  // !=0 -> failure...
                    fprintf(stderr,"Cannot open: %s\n",argv[1]);
                    print_last_error();
                    return 1;
              }
           else
```

```
{
        if (scanifc_point3dstream_add_demultiplexer(h3ds,"hk.txt",0,"status protocol")) //no package names, class names - 'status','protocol'
            {
                print_last_error();
                return 1;
            }
        string p(argv[1]);
        int i = p.find_last_of(".");
        p = p.substr(0,i);
        p = p + ".txt";
        FILE *f=fopen(p.c_str(),"w");
        do
         {
            if (scanifc_point3dstream_read(h3ds,N_TARG,rg_xyz32,rg_attributes,rg_time,&count,&end_of_frame))
                {
                   fprintf(stderr,"Error reading %s\n",argv[1]);      //count -> no of points actually read returned in count
                   return 1;
                }
            for (n=0;n<count;++n)
                {
                  fprintf(f,"%f,%f,%f,%u,%lu,%d\n",rg_xyz32[n].x,rg_xyz32[n].y,rg_xyz32[n].z,(rg_attributes[n].flags & 0x40)>>6,rg_time[n],end_of_frame);
                }
            numbr_of_points+=n;
         } while(count >0); // reads 'count' no of points
        fprintf(f,"The number of points obtained from rxp :%lu",numbr_of_points); //writing the number of points to the file

        if (scanifc_point3dstream_close(h3ds)) //closing the file handle
          {
            fprintf(stderr,"Error,closing %s\n",argv[1]);
            return 1;
```

```
            }
        fclose(f);
        return 0;
        }
    }

    fprintf(stderr,"Usage:%s <input-uri>\n",argv[0]);
    return 1;
    }
```

/* End Of Code */

/* Start Of Header */

```
#ifndef GPS_SYNC_HEADER_H
#define GPS_SYNC_HEADER_H

#include <C:\Users\Sagnik\Desktop\PS1\rivlib-2_3_0-x86_64-windows-mgw49\include\riegl\ctrllib.hpp>
#include <C:\Users\Sagnik\Desktop\PS1\rivlib-2_3_0-x86_64-windows-mgw49\include\riegl\scanlib.hpp>
#include <iostream>
#include <fstream>
#include <exception>
#include <stdexcept>
#include <cmath>
#include <algorithm>
#include <memory>

using namespace ctrllib;
using namespace scanlib;
using namespace std;
```

```
int main(int argc,char* argv[]) //3rd argument , argv[2] -> output filename ;argv[0] -> tls s no / ip addrs ; argv[1] -> port no of external gps
{

        if ((argc!=4)||(string(argv[2])=="--help"))
        {
           cerr<<"Usage:"<<argv[0]<<" address[:service] outputfile"<<endl;
           return 1;
        }
ctrl_client_session session;
try
  {
          // now connect to the instrument specified as first argument
          // the connection string is expected in format
          //   "<host_name_or_ip_address>[:<service_name_or_port_number>]"
          // Examples:
          //   "192.168.0.234", "192.168.0.234:20002",
          //   "s9991234" or "s9991234:20002"
          // if port number is not specified, 20002 is assumed as default

   session.open(argv[2]);
   cout<<"Connected to "<<argv[2]<<endl;
   string result;
   vector<string> comment;
   string status;
   session.execute_command("MEAS_ABORT","",&result,&status,&comment);// telling the TLS to abort the current session
   cout<<"MEAS_ABORT() returned "<<result<<endl;
  for(int i=0;i<comment.size();i++) //for(int i=0 ; i<comment.size();i++)
       cout<<comment[i]<<endl;
 cout<<"Status is "<<status<<endl;
 session.execute_command("MEAS_BUSY","1",&result,&status,&comment);
 cout<<"MEAS_BUSY(1) returned "<<result<<endl;//waiting for the TLS to terminate the previous session
 for(int i=0;i<comment.size();i++)//for(size_t i=0; i<comment.size(); ++i)
      cout<<comment[i]<<endl;
 cout<<"Status is "<<status<<endl;
 string value;
 session.get_property("INST_IDENT",value);
 cout<<"Instrument is "<<value<<endl;
```

```
                //
 if (std::string(value).compare(1,6,"VZ-400")!=0)
    {
     throw runtime_error("Error:VZ-400 instrument required");
     }

if (std::string(value).compare(1,6,"VZ-400")==0)//if (std::string(value).compare(1, 6, "VQ-250") == 0)
   {
     session.set_property("GPS_MODE","2");
     cout<<"GPS_MODE set to 2 (external GPS via RS232)"<<endl;
     session.set_property("GPS_EXT_COM_BAUDRATE","6");
     cout<<"GPS_EXT_COM_BAUDRATE set to 115200"<<endl;
     session.set_property("GPS_EXT_EDGE","0");
     cout<<"GPS_EXT_EDGE set to (0=positive,1=negative)"<<endl;

     session.set_property("GPS_EXT_FORMAT", "\"GPZDA\"");
     cout<<"GPS_EXT_FORMAT set to GPZDA"<<endl;
     session.set_property("GPS_EXT_SEQUENCE", "\"PPS_FIRST\"");
     cout<<"GPS_EXT_SEQUENCE set to PPS_FIRST"<<endl;
    }
    session.get_property("MEAS_PROG",value); //current measurement program
    cout<<"MEAS_PROG is "<<value<<endl;
    session.execute_command("MEAS_SET_PROG","1",&result,&status,&comment); // set measurement program
    cout<<"MEAS_SET_PROG(1) returned "<<result<<endl;
    for(int i=0;i<comment.size();i++)
        cout<<comment[i]<<endl;
     cout<<"Status is "<<status<<endl;
     session.execute_command("SCN_SET_SCAN","30, 130, 0.9875, 180",&result,&status,&comment);
     cout<<"SCN_SET_SCAN(30, 130, 10) returned "<<result<<endl;
     for(int i=0;i<comment.size();i++)
          cout<<comment[i]<<endl;
```

```
    cout<<"Status is "<<status<<endl;
    session.execute_command("MEAS_START","",&result,&status,&comment); //line scan details : starting theta,
theta_end,increment=0.09875 , phi(alignment)=180
    cout<<"MEAS_START() returned "<<result<<endl;
    for(int i=0;i<comment.size();i++)
        cout<<comment[i]<<endl;
    cout<<"Status is "<<status<<endl;

                                    /*acquirirng data*/
    cout<<"Output:"<<argv[3]<<endl;
    ofstream outfile(argv[3],ios::binary|ios::out);//binary file created in output mode

                    /* opening input connection to scanner: */

    bool shutdown_requested=false;
    string connstr(argv[2]);//string connstr(argv[1]);
    connstr="rdtp://"+connstr+"/current";
    cout<<"Input: "<<connstr<<endl;
    std::shared_ptr<scanlib::basic_rconnection> rc;
    rc = basic_rconnection::create(connstr);
    rc->open();
    basic_rconnection::size_type limit=50*1024000;  //basic_rconnection::size_type limit = 50*1000000;
     basic_rconnection::size_type count;
     basic_rconnection::size_type total=0;
     char buffer[8192];  //1 kB of char
     int last_percent=0;
     int data_percent;
     cout<<"Data limit:"<<limit<<" bytes"<<endl;
     cout<<"Percent:"<<endl;
     cout<<"1..10...20...30...40...50...60...70...80...90..100"<<endl;

                /* loop runs as long as the instrument delivers data */

    while(0!=(count=rc->readsome(buffer,8192)))
            {
```

```
            outfile.write(buffer,count);
            total+=count;
            if((total>limit)&&(!shutdown_requested))
               {
                    /*requesting shutdown of data delivery*/
                rc->request_shutdown();
                shutdown_requested=true;
               }
            data_percent= min(100,int(ceil(total*100.0/limit)));

                 /*updating progress bar*/
            while(last_percent<data_percent)
                  {
                   cout<<"*";
                   last_percent += 2; // 1 '*' = 2 percent
                   }
          }
    cout<<endl;


    outfile.flush(); // flushing the '.rxp' output file

    outfile.close(); // closing the '.rxp' output file


    cout<<total<<" bytes received"<<endl;

                    /*stopping measurement:*/

    session.execute_command("MEAS_STOP","",&result,&status,&comment);
    cout<<"MEAS_STOP() returned "<<result<<endl;
    for(int i=0;i<comment.size();i++)
        cout<<comment[i]<<endl;
        cout<<"Status is "<<status<<endl;

              /* the connection is closed if the instrument is no longer needed */

        session.close();  // closing the connection
        cout << "Connection closed" << endl;
        cout<<"Demo program successfully finished"<<endl;
}
catch(exception& e)
{
```

```
        cerr<<e.what()<<endl;
        return 1;
}
catch(...)
{
        cerr<<"unknown exception"<<endl;
        return 1;
}
return 0;
}
#endif // GPSS_SYNC_HEADER_H ends
```

/* End Of Header */

6.2.3.2. Building the Code on Ubuntu

To build the code on Ubuntu from the command line terminal, make the following changes and installations –

No significant changes are required, if any modifications are needed for the particular system on which the code is being built, it is advisable that they are done in accordance with official Ubuntu documentation.

After that, execute the following commands from the terminal window-

g++ -g -std=c++11 attributes_output_edited_final.c -o attr_output_edited_final -I/home/user/rivlib-2_3_0-x86_64-linux-gcc44/include -L/home/user/rivlib-2_3_0-x86_64-linux-gcc44/lib -lscanifc-mt -lscanlib-mt -lctrllib-mt –lpthread

6.2.3.3. Running the Executable on Ubuntu

To run the executable on Ubuntu from the command line terminal, make the following changes and installations -

No significant changes are required, if any modifications are needed for the particular system on which the code is being built, it is advisable that they are done in accordance with official Ubuntu documentation.

After that, execute the following commands from the terminal window-

./attributes_output_edited_final <filename>
<host_name_or_ip_address>:<service_name_or_port_number> <filename>

Examples-

<host_name_or_ip_address>:<service_name_or_port_number>:
"192.168.0.234" OR "192.168.0.234:20002",

"s9991234" OR "s9991234:20002"

<filename>:“scan1.rxp”

6.2.3.4. Output of the Code

An ‘.rxp file’ is created and the scan attributes are extracted from the same ‘.rxp file’ onto a ‘.txt file’.

6.3. USE OF SERVERS TO TIME SYNCHRONISE TLS AND GPS RECEIVER:

We adopted this approach mainly because we had problems with synchronising by directly connecting the TLS and the Global Positioning System (GPS) receiver. So we thought of indirectly connecting the two instruments with a computer in between them which will act as an interfacing system for the devices. The basic idea was that the computer will take the accurate time information from the GPS receiver and pass it on to the TLS so that the TLS can use the same to time-synchronise its scans with the GPS receiver. For that we planned to use our computer both as server and client so that it can take time information from the receiver and provide it to the scanner. We used the property of transitivity in our approach in the sense that if the GPS receiver (denoted by A) is synchronised with the computer (denoted by B) and the computer (B) is synchronised with the TLS (denoted by C), then the receiver (A) will be synchronised with the TLS (C) .In this case, we use our computer as client when it is communicating with the receiver and as a server

when it is communicating with the TLS. The process is illustrated below using an arrow diagram (figure 15).

We needed to use two types of communication protocol for this –

i) Network Time Protocol (NTP) –

NTP is a TCP/IP protocol for synchronising time over a network .A client requests the current time from a server and uses it to set its own clock. We needed NTP because we wanted to keep our computer time synchronised with the time source, the GPS receiver, in this case. For synchronisation with the time source, system supporting NTP communication uses NTP Daemon Tools (NTPD) which can act both as server and client. Even our personal computers can act as NTP clients using NTPD and connect with the NTP servers available all over the world to maintain an accurate time. The computer basically uses NTPD to take time from a source but in this case NTPD alone was not enough to obtain the time information to obtain the time interfacing. It needed another interfacing tool called GPS Daemon (GPSD) tool for communicating with the receiver

ii) Dynamic Host Configuration Protocol (DHCP)-

The main reason behind using DHCP was that using the Human Machine Interface (HMI) or the control panel provided on the scanner we found that the scanner can be configured both as DHCP client and DHCP server. DHCP is used for communication between the computer and the TLS in which the computer (a DHCP server) will supply the time information to the TLS (a DHCP client). The TLS will use that time information to time-synchronise its scans with the GPS receiver.

The first problem we faced when implementing this approach was that though Windows systems has full support for Dynamic Host Configuration Protocol (DHCP) and Network Time Protocol (NTP) operations has very low support for using GPS Daemon (GPSD) tool . We referred to numerous websites and found the same and decided to use Ubuntu 14.04, a Linux Distribution, to build our servers and to use our computer as a client. Upon switching over to Ubuntu, we had to use and install numerous packages to get all our servers up and running.

For using the server based approach we needed Linux as some of the daemon tools we were going to use were not supported in windows.

The computer is an NTP client and takes the accurate time data from the GPS receiver which is an NTP server, via NTPD which then uses GPSD as an NTPD can’t directly communicate with GPS receiver and needs GPSD for this purpose. The computer then acts like a DHCP client and supplies the time information to the TLS which becomes a DHCP client which finally uses this time information to time synchronise its

scans. In this way, the time synchronisation between the GPS receiver and the scanner was finally achieved. We had to work on it for several days as it tool time to understand the protocols and their functionalities well enough to use the daemon tools and set up and configure the servers.

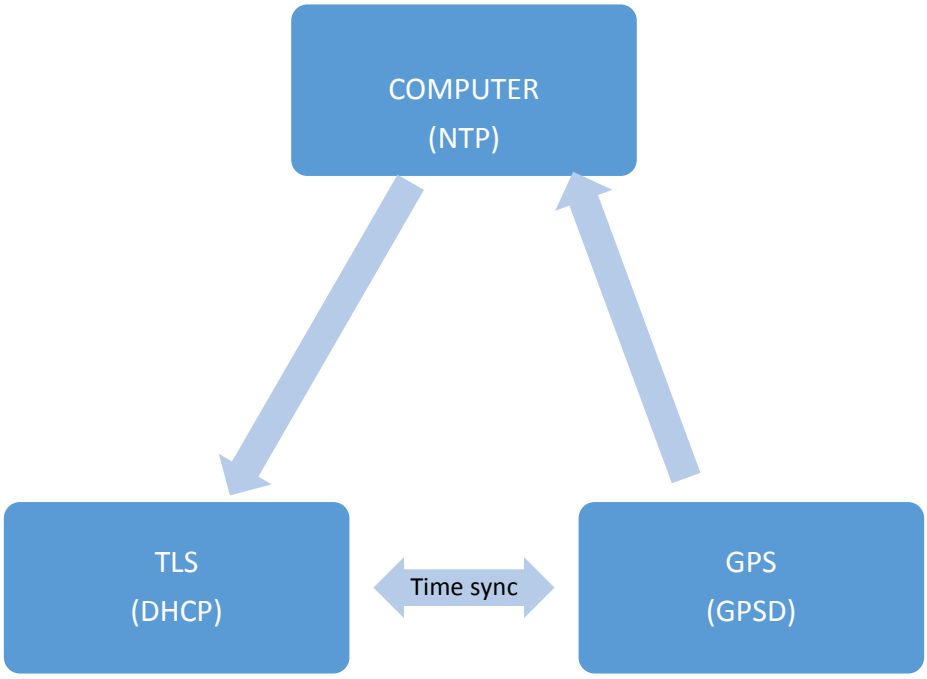


Figure 17 – Representative Diagram of Synchronisation

We have to set three servers and the procedures are as follows:

6.3.1. Using NTPD to set up NTP Communication

NTPD is a daemon tool which helps a system keep its internal clock synchronized with a time source. The time sources are basically NTP servers located all around the world.

NTP uses a hierarchical semi-layered system of time resources. Each level of this hierarchy is called a 'stratum'. A server synchronised to a stratum 'n' server will be running at stratum 'n+1'. It is the distance between the reference clock and the NTP server/client in terms of the number of intermediate servers/clients and it is so implemented to prevent cyclic dependencies. The lower the stratum number of the NTP server /client is, more accurate is the time kept by it. Stratum is not always an indication of quality or reliability; it is common to find stratum 3 time sources that are higher quality than other stratum 2 time sources. Telecommunication systems use a different definition for clock strata. A brief description of strata 0, 1, 2 and 3 is provided below (figure 16)

The NTPD tool understands the Network Time Protocol and can work both as a client and server in an NTP network. Besides that it can synchronize with local time sources. The NTPD cannot interface with GPS devices directly, but it has defined a number of interfaces which can be used by other daemons to interface GPS with NTPD.

The NTPD is either already installed on the system, or it can be downloaded from one of the package repositories. Linux distributions like Debian , Ubuntu and RedHat all have NTPD in their standard repository.

6.3.1.1. Steps for Installing NTPD and Setting up NTP Server/Client on Ubuntu

Note: The commonly available NTPD distribution uses common compilers and tools that come with most Linux distributions. Not all of these tools exist in the standard distribution of modern UNIX versions (compilers are likely to be an add-on product). If this is the case, consider using the GNU tools and 'gcc' compiler included as freeware in some systems. For a successful build, all these tools should be accessible via the current path.

6.3.1.1.1. Installation -

i) We switched to the root directory on your system using the command

sudo –i

ii) We went to the base directory using the command

cd /

iii) We performed an automatic configuration process using the command

. /configure

iv) We compiled and link the distribution using the command

make

v) We changed to the NTP package directory and the executables got installed by default in '/usr/local/bin' using the command (The user may need to use the 'dpkg' command to install NTPD if it comes on the form of '.deb' package)

apt-get install ntp

This completed the installation process. (If any other packages are required, they should be installed side by side.)

The 'ntp.conf' file is shown below. To modify the file according to the requirements of the system or the type of implementatation, we used the command-

nano etc/ntp.conf

#START OF /etc/ntp.conf#

```
# /etc/ntp.conf, configuration for ntpd; see ntp.conf(5) for help

driftfile /var/lib/ntp/ntp.drift


# Enable this if you want statistics to be logged.
#statsdir /var/log/ntpstats/

statistics loopstats peerstats clockstats
filegen loopstats file loopstats type day enable
filegen peerstats file peerstats type day enable
filegen clockstats file clockstats type day enable

# Specify one or more NTP servers.

# Use servers from the NTP Pool Project. Approved by Ubuntu Technical Board
# on 2011-02-08 (LP: #104525). See http://www.pool.ntp.org/join.html for
# more information.
server 0.ubuntu.pool.ntp.org
server 1.ubuntu.pool.ntp.org
server 2.ubuntu.pool.ntp.org
server 3.ubuntu.pool.ntp.org

# Use Ubuntu's ntp server as a fallback.
server ntp.ubuntu.com

# Access control configuration; see /usr/share/doc/ntp-doc/html/accopt.html for
# details.  The web page <http://support.ntp.org/bin/view/Support/AccessRestrictions>
# might also be helpful.
#
# Note that "restrict" applies to both servers and clients, so a configuration
# that might be intended to block requests from certain clients could also end
```

```
# up blocking replies from your own upstream servers.

# By default, exchange time with everybody, but don't allow configuration.
restrict -4 default kod notrap nomodify nopeer noquery
restrict -6 default kod notrap nomodify nopeer noquery

# Local users may interrogate the ntp server more closely.
restrict 127.0.0.1
restrict ::1

# Clients from this (example!) subnet have unlimited access, but only if
# cryptographically authenticated.
#restrict 192.168.123.0 mask 255.255.255.0 notrust


# If you want to provide time to your local subnet, change the next line.
# (Again, the address is an example only.)
#broadcast 192.168.123.255

# If you want to listen to time broadcasts on your local subnet, de-comment the
# next lines.  Please do this only if you trust everybody on the network!
#disable auth
#broadcastclient
```

#END OF /etc/ntp.conf#

Modifications were in terms of the NTP server addresses, access restrictions etc.

6.3.1.1.2. Running NTPD as Server/Client –

i) We started the NTPD tool by using the following command –

service ntp start

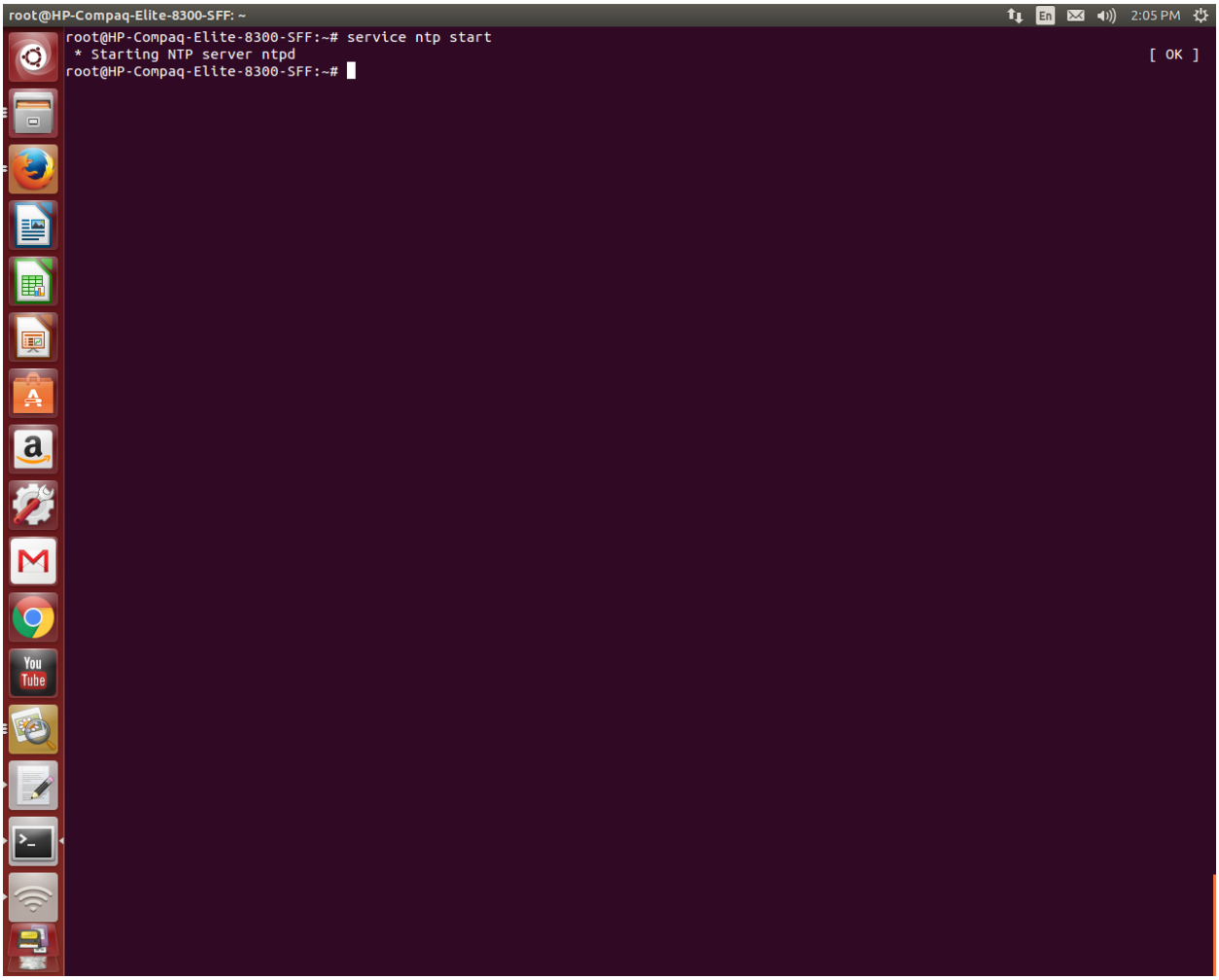


Figure 18 – 'service ntp start' on Command Window

ii)To see the time sources for the current server ,we used the command-

ntpq –p

OR

ntpq –c lpeer

This command gives details like the IP addresses of the time sources for the current server or client . the stratum level of the server etc.

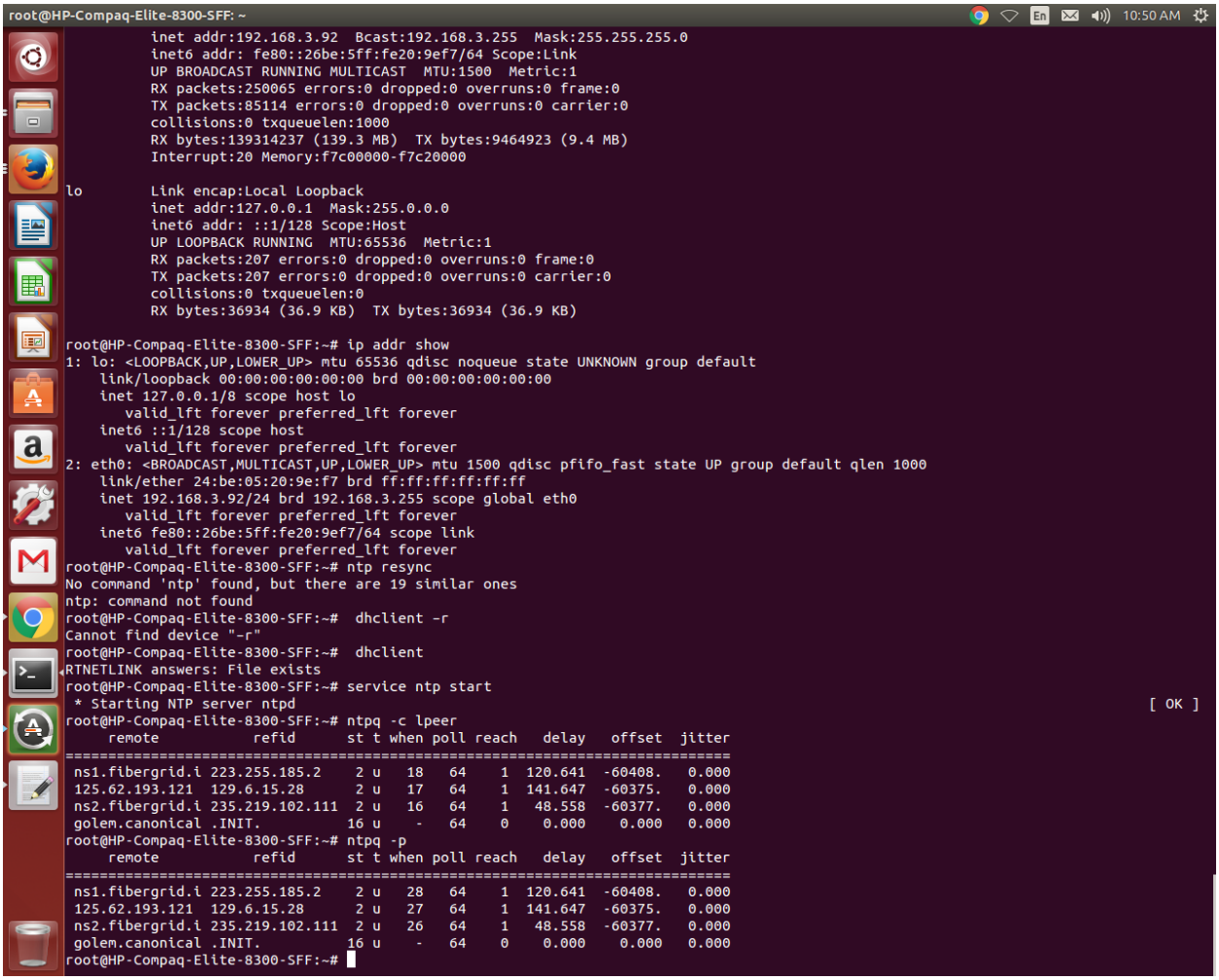


Figure 19– 'ntpq -p' on Command Window

iii)To see the information regarding the current time ,date, timezone etc , we used the command

timedatectl status

OR

timedatectl

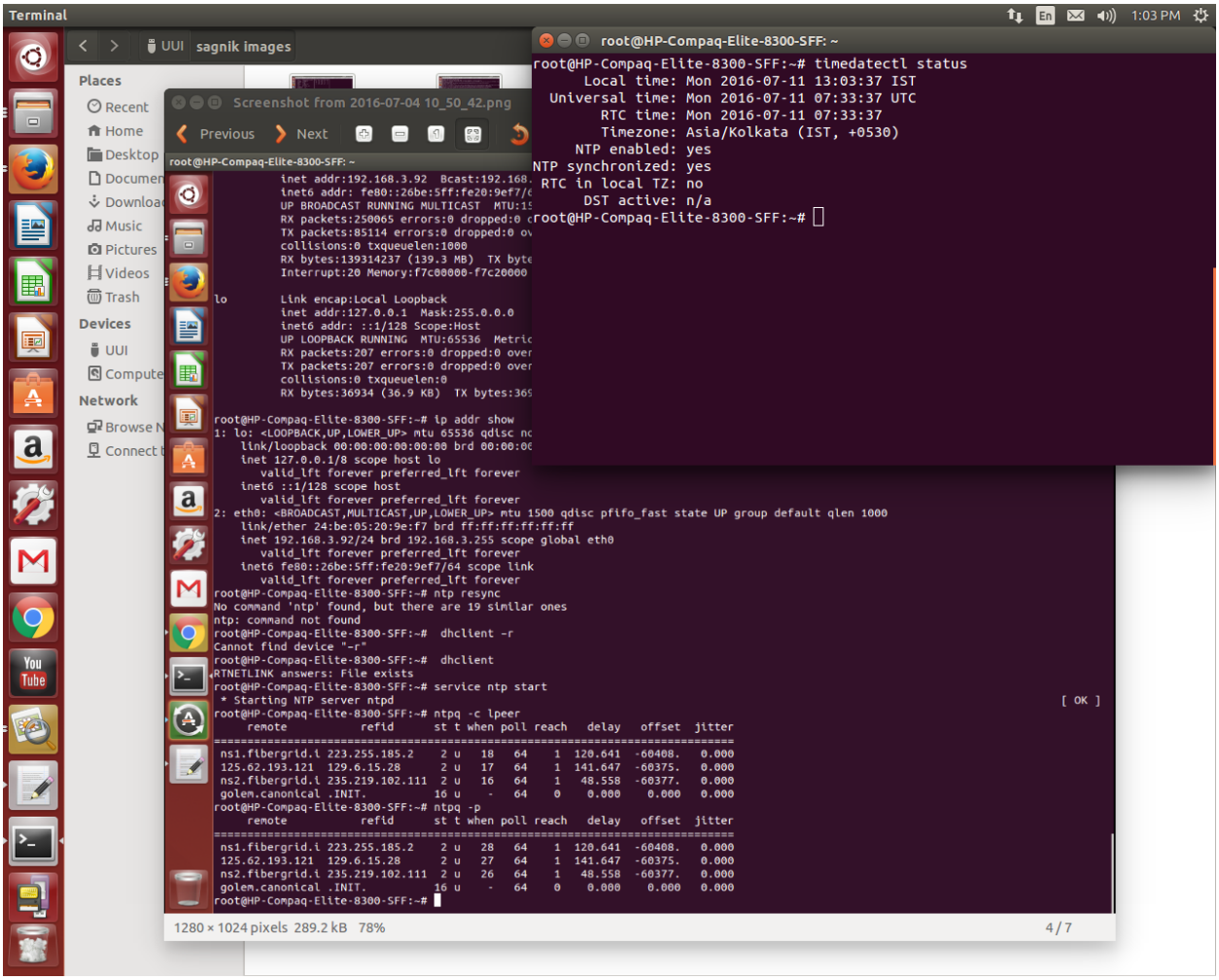


Figure 20 – ‘timedatectl status’ on Command Window

iv)To check the stratum level ,we used the command

ntpq –n

OR

ntpq –c rv

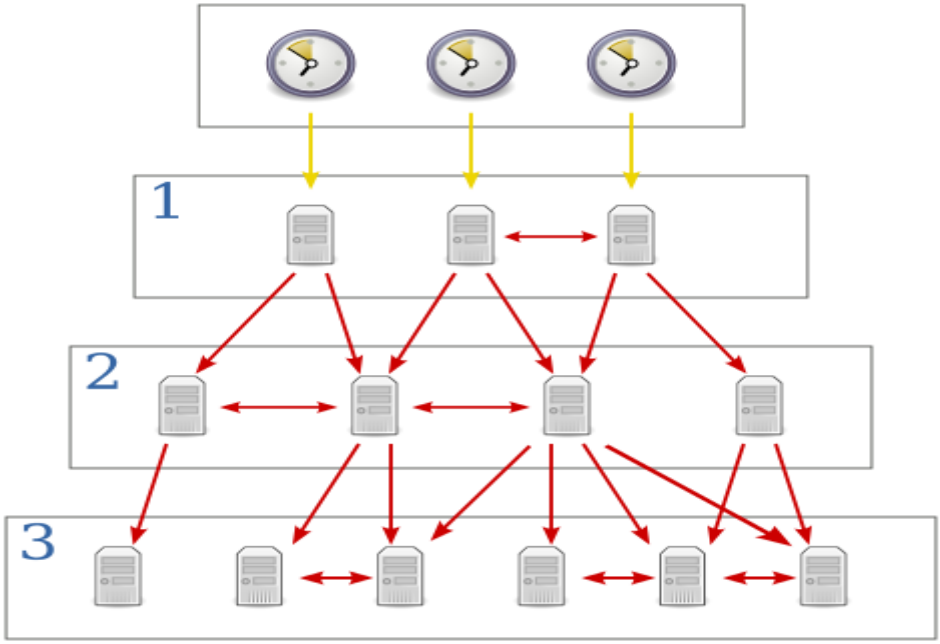


Figure 21 – Representative Diagram of NTP Server Time Synchronisation

6.3.2 Using GPSD to set up Communication between GPS Receiver and NTPD

NTPD can't communicate directly with a GPS device, so a separate daemon tool is needed for the GPS communication. As always in Linux there is more than one solution to this problem, but we have used the GPSD tool for this task. The GPSD tool is the most versatile of all GPS daemons currently available. It does not only interface well with most GPS models available, but it also communicates nicely with NTPD and has a special interface to make it possible to easily integrate in web pages etc. The GPSD tool setup package can be downloaded from 'gpsd.berlios.de'.

6.3.2.1 Steps for Installing GPSD and Running It

6.3.2.1.1 Installation-

i) The 'etc/init.d/gpsd' is a shell command script which can be configured if required. We modified it according to our requirements to suit our purpose best. We also changed '/etc/sysconfig/gpsd' All the build prerequisites were in place and to install GPSD tool we used the command

scons && scons check && sudo scons udev-install

If 'scons' fails, the user should install the 'scons' package using the command

apt-get install scons

For more troubleshooting, the user should refer to the webpage –

'Fossies.org/linux/gpsd/build.txt'

In the default configuration, the GPSD tool is installed in the '/usr/local/sbin' directory. We didn't change the compilation settings to make it easy in the future to download and install a new version of the software. Instead, we linked to this daemon tool from the general '/usr/sbin' directory with the command

ln –s /usr/local/sbin/gpsd /usr/sbin/gpsd

#START OF /etc/rc.d/init.d/gpsd#

```
#!/bin/bash
#
# gpsd          This shell script takes care of starting and stopping
#               gpsd (GPS daemon).
#
# chkconfig: 2345 50 80
# description: gpsd is the GPS daemon. \
# The GPS daemon gets the actual position and time information from \
# an attached GPS device and makes this information available for \
# other daemons and applications.

# Source function library.
. /etc/init.d/functions

# Source networking configuration.
. /etc/sysconfig/network

if [ -f /etc/sysconfig/gpsd ];then
        . /etc/sysconfig/gpsd
fi


RETVAL=0
prog="gpsd"

start() {
```

```
        setserial $DEVICE low_latency
        stty $BAUDRATE -F $DEVICE

        echo -n $"Starting $prog: "
        daemon gpsd $DAEMON_OPTS $DEVICE
        RETVAL=$?
        echo
        [ $RETVAL -eq 0 ] && touch /var/lock/subsys/gpsd
        return $RETVAL
}

stop() {
        echo -n $"Shutting down $prog: "
        killproc gpsd
        RETVAL=$?
        echo
        [ $RETVAL -eq 0 ] && rm -f /var/lock/subsys/gpsd
        return $RETVAL
}

# See how we were called.
case "$1" in
  start)
        start
        ;;
  stop)
        stop
        ;;
  status)
        status gpsd
        RETVAL=$?
        ;;
```

```
  restart|reload)
        stop
        start
        RETVAL=$?
        ;;
  condrestart)
        if [ -f /var/lock/subsys/gpsd ]; then
           stop
           start
           RETVAL=$?
        fi
        ;;
  *)
        echo $"Usage: $0 {start|stop|restart|condrestart|status}"
        RETVAL=3
esac


exit $RETVAL
```

#END OF /etc/rc.d/init.d/gpsd#

#START OF /etc/sysconfig/gpsd#

```
DAEMON_OPTS="-n"
BAUDRATE="115200"
DEVICE="/dev/ttyS0"
```

#END OF /etc/sysconfig/gpsd#

6.3.2.1.2 Running GPSD-

NOTE – Before running the GPSD , we made sure that our NTPD is also running since the computer will use the time information obtained from the GPS receiver via GPSD tool to synchronise its own time . The computer uses the NTPD tool for the purpose.

The NTPD can be started and tested using the commands mentioned above.

i)Since our IRNSS/SBAS/GPS receiver supported NMEA-0183 communication with talker ID GPZDA which contains the relevant time and date information required for time synchronisation at 115200 Baud rate (Bits per sec) only we had to set the speed of communication using the command

stty –F /dev/ttyUSB0 ispeed 115200

We were using serial port USB0 for the serial communication between the TLS and the GPS receiver.The port number can be checked using the commands

dmesg | grep –i usb

OR

dmesg | grep –i tty

OR

dmesg

OR

setserial –g /dev/ttyS[0123456789

OR

cat /dev/ttyUSB0

OR

cat < /dev/ttyUSB0

ii) After testing the GPSD tool according to the instructions which come with the daemon source, we started and tested the GPSD tool using the command

/usr/local/sbin/gpsd -n –N –D5/dev/ttyUSB0

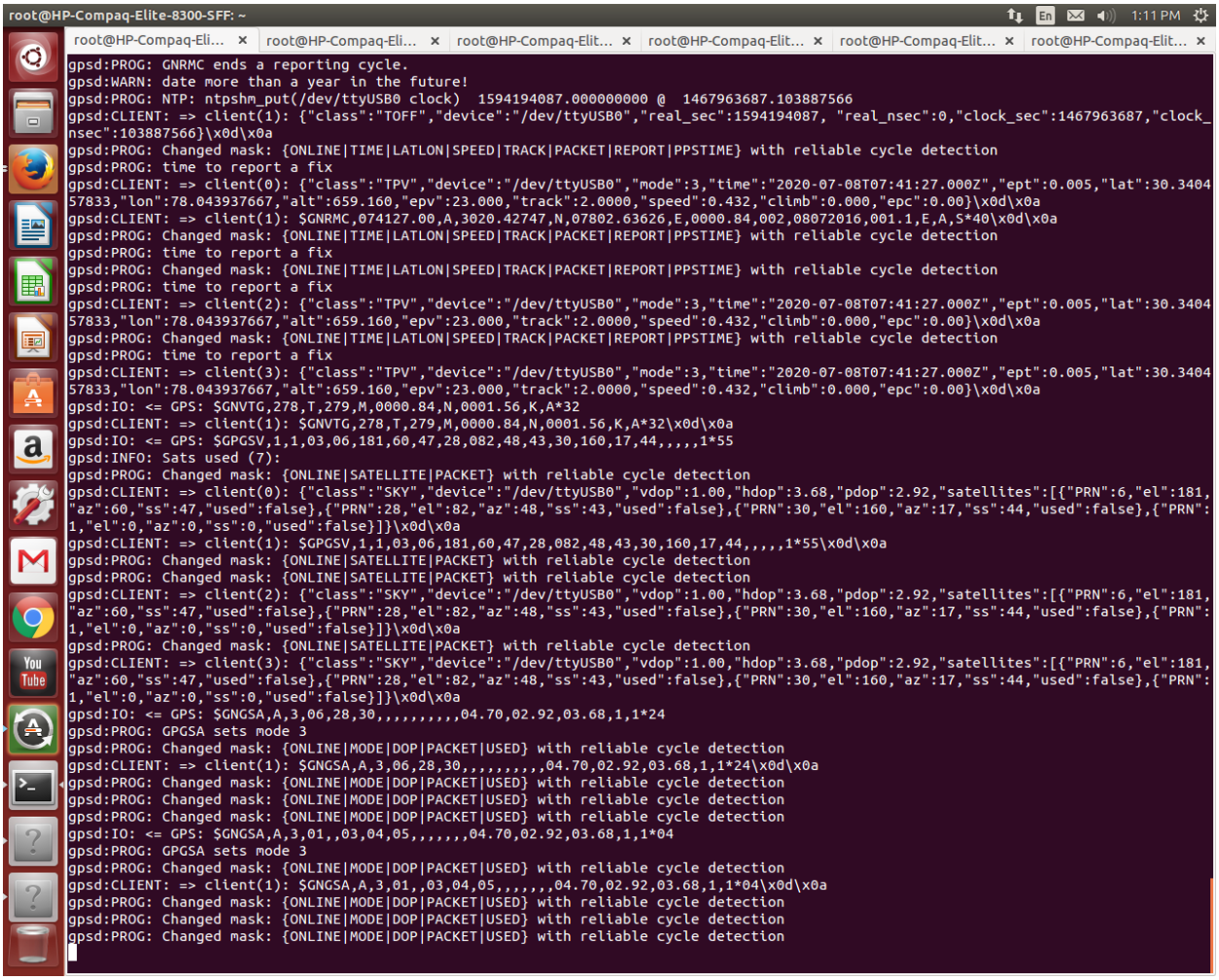


Figure 22 – '/usr/local/sbin/gpsd -n –N –D5/dev/ttyUSB0' on Command Window

The 'n' stands for the instruction to the GPSD that it won't wait for a client to connect before polling to whatever GPS is associated with it.

The 'N' stands for the instruction that the GPSD shouldn't daemonize and should run in the foreground.

The 'D' stands for the debug level and 'D5' is the maximum level of debugging.

iv) We also installed the GPSD clients package using the commands

apt-get update

apt-get install gpsd-clients

The 'gpsd-clients' have the 'xgps' ,'xgpsspeed' ,'cgps' ,'gpsmon' etc which can be used for live tracking and monitoring of GPS data

v) After running the GPSD tool , we tested its working and credibility using the commands

xgps # this command was executed on a new window #

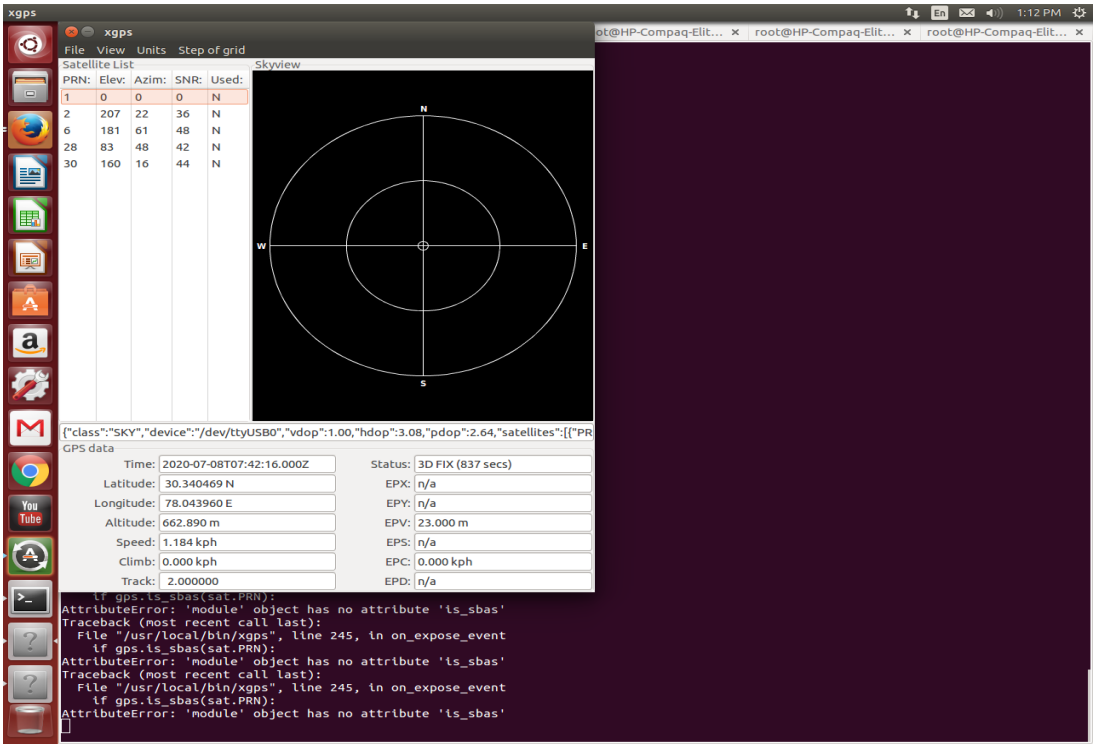


Figure 23 – 'xgps' on command window

xgpspeed # this command was executed on a new window #

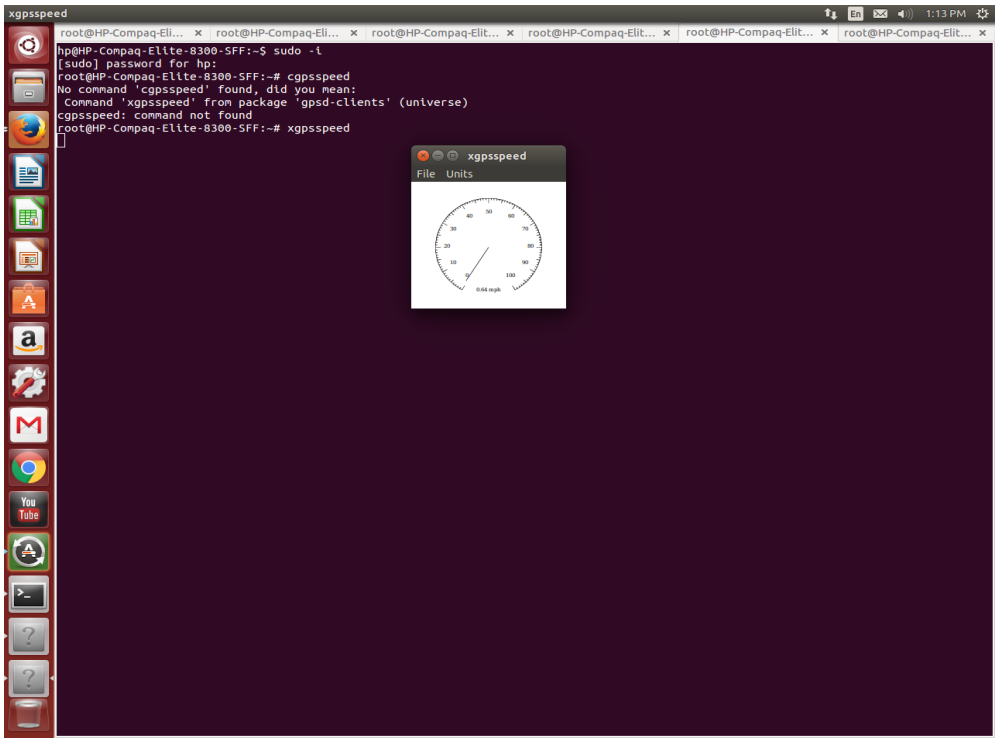


Figure 24 – 'xgpsspeed' on command window

cgps # this command was executed on a new window #

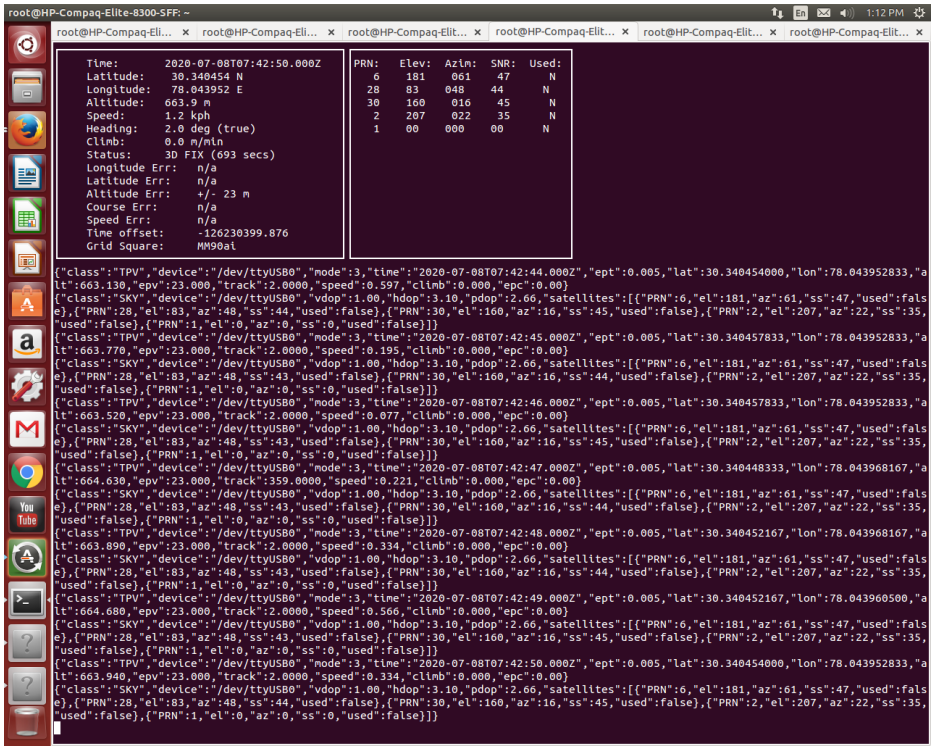


Figure 25 – 'cgps' on command window

gpsmon # this command was executed on a new window #

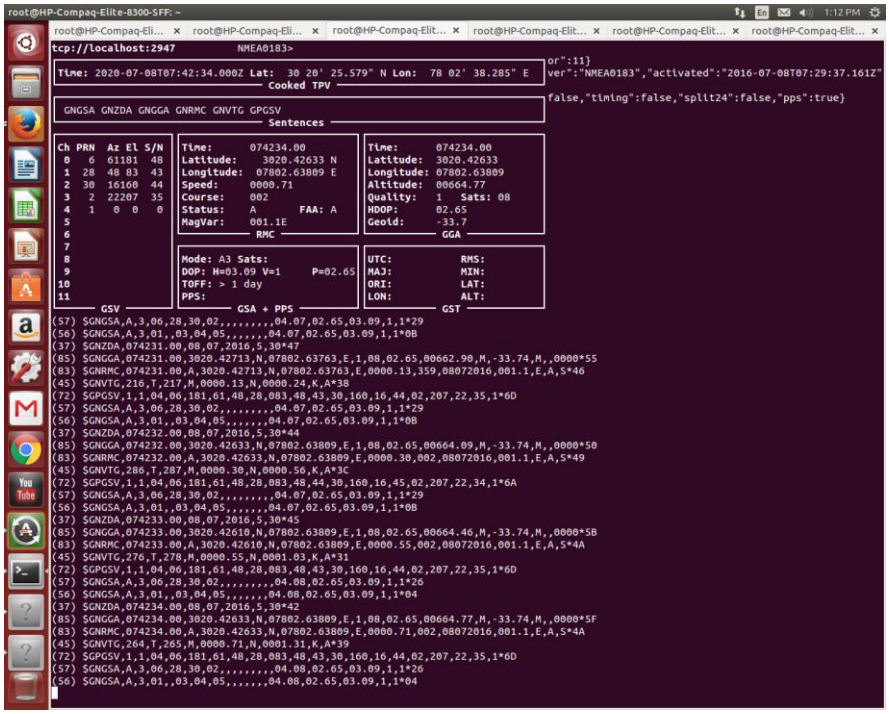


Figure 26 – 'cgpsmon' on command window

All these commands show live data logging on to the computer . The ‘xgps’ , ‘xgpspeed’ commands show GUI windows showing the same data.

vi)The GPSD tool can be stopped by using the command **(^C)** or by pressing **‘CTRL+C’**.

By following the above written procedure, we could finally got our GPSD up and running after working hard on it for several days.

6.3.3. Using DHCPD to set up DHCP Communication

During the initial days of our project , we found out that the TLS can act both as a DHCP server and as a DHCP client. So, we figured out that if we can somehow set up a DHCP server on our computer we can send the time information obtained by the computer from the GPS receiver to the TLS. This will definitely help us synchronise the TLS with the GPS receiver.

DHCP stands for Dynamic Host Configuration Protocol.It is a standardized network protocol used in Internet Protocol(IP) networks for dynamically distributing network configuration parameters such as IP addresses and networking parameters like Subnet Mask, default Gateway automatically from a DHCP server , reducing the need for a network administrator or a user to configure these settings manually. DHCP also helps to send the NTP server information from the DHCP server to the DHCP client.

6.3.3.1. Steps for Installing GPSD and Running It

6.3.3.1.1. Installation-

i) We updated the Linux repository using the command

apt-get update

ii) The server was installed using the following command

apt-get install isc-dhcp-server –y

iii) After installation, the ‘etc/default/isc-dhcp-server’ file was opened and an interface ‘eth0’ was assigned by using the command

nano /etc/default/isc-dhcp-server

INTERFACES=”eth0”

iv)Then the ‘dhcpd.conf’ file was modified using the command

nano /etc/dhcp/dhcpd.conf

A piece of code was added to the ‘dhcpd.conf’ file and both the actual version and the modified version of the file are shown below.

Even the ‘/etc/network/interfaces’ and ‘/etc/NetworkManager/NetworkManager.conf’ were modified as per requirements.

Both the original and changed version of the files are shown below.

The ‘etc/dhcp/dhcpd.conf’ file gives the subnet address, subnet mask,optional routers and optional broadcast address of the server . It also gives the range of IP addresses that can be leased by the dhcp server to the client.The range in this case is ‘192.168.3.100’ to ‘192.168.3.200’ .The default lease time of 600 seconds and the maximum lease time of 7200 seconds are also specified.

The subnet address is basically calculated by bitwise anding between the IP address and the Subnet Mask of the server. The IP address in this case is ‘192.168.3.xx’ and the Subnet Mask is ‘255.255.255.0’ .So the Subnet Address as calculated is ‘192.168.3.0’.

6.3.3.1.2. Running the DHCP Server-

i)Once the server configuration has been done as mentioned above , the server is ready to be started. The server can be started using the command

service isc-dhcp-server start

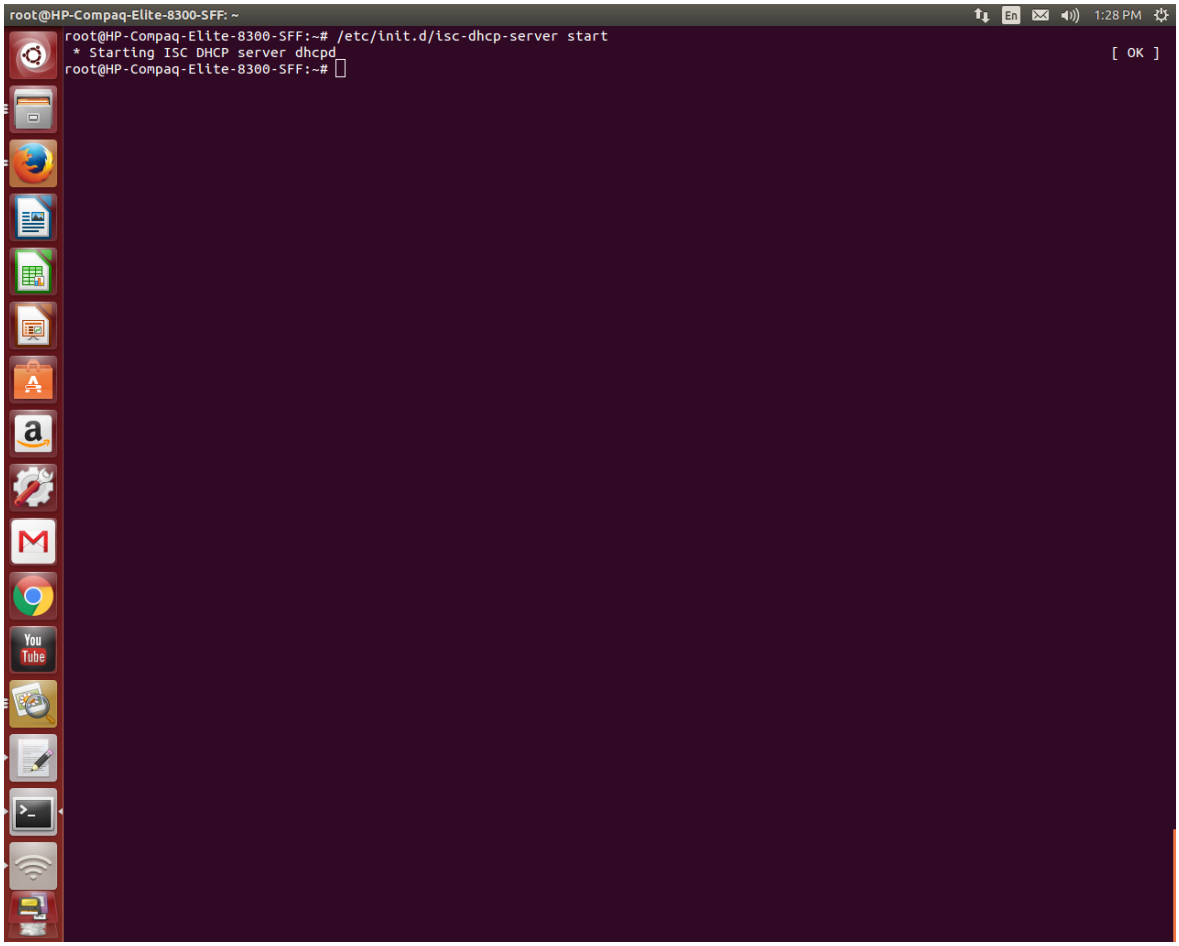


Figure 27– ‘service isc-dhcp-server start’ on command window

ii)The server can be terminated using the command

service isc-dhcp-server stop

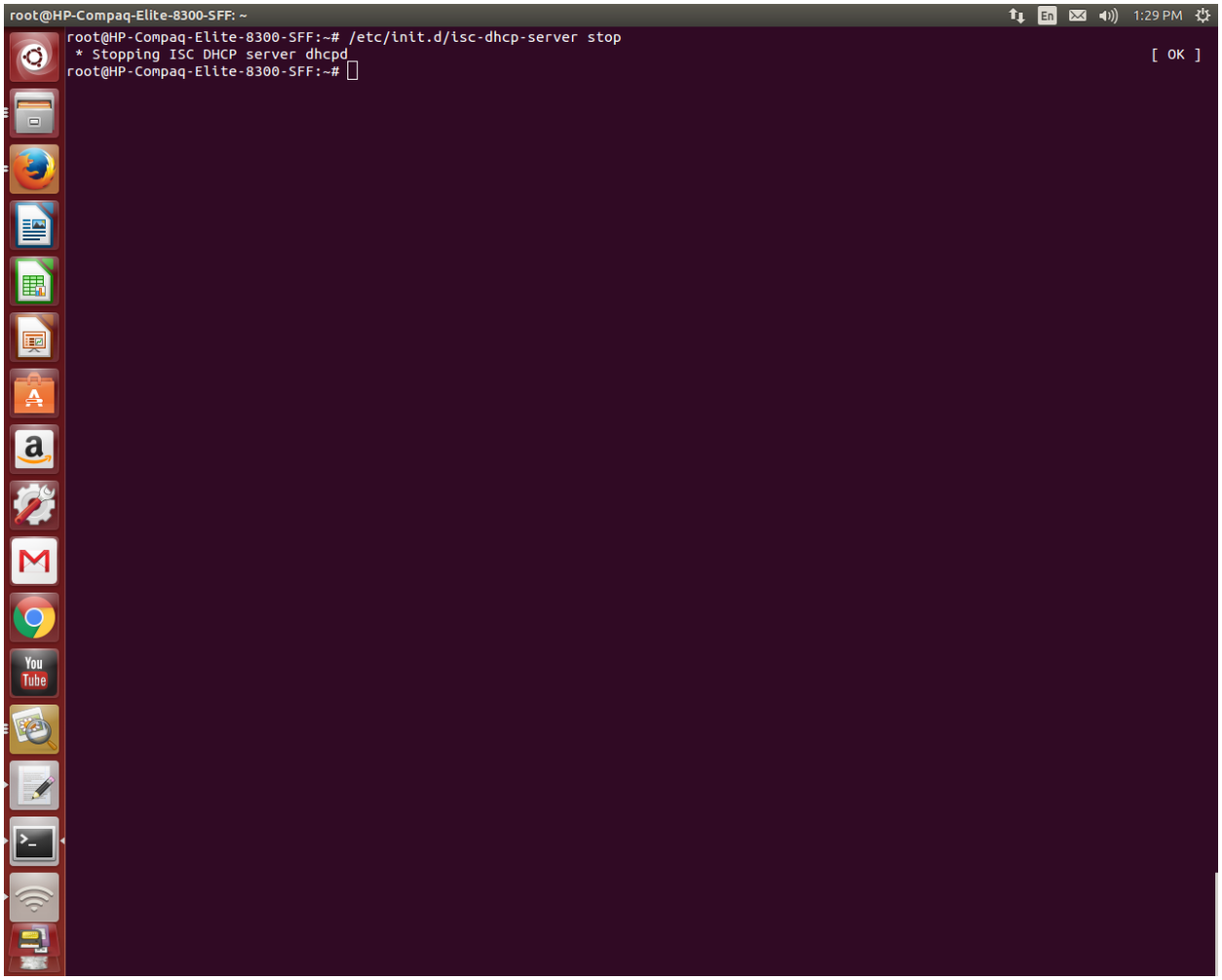


Figure 28 – 'service isc-dhcp-server stop' on command window

iii)The IP address and other network parameters of the client connected to the DHCP server on the computer can be checked using the commands

ifconfig –a

OR

ip addr show

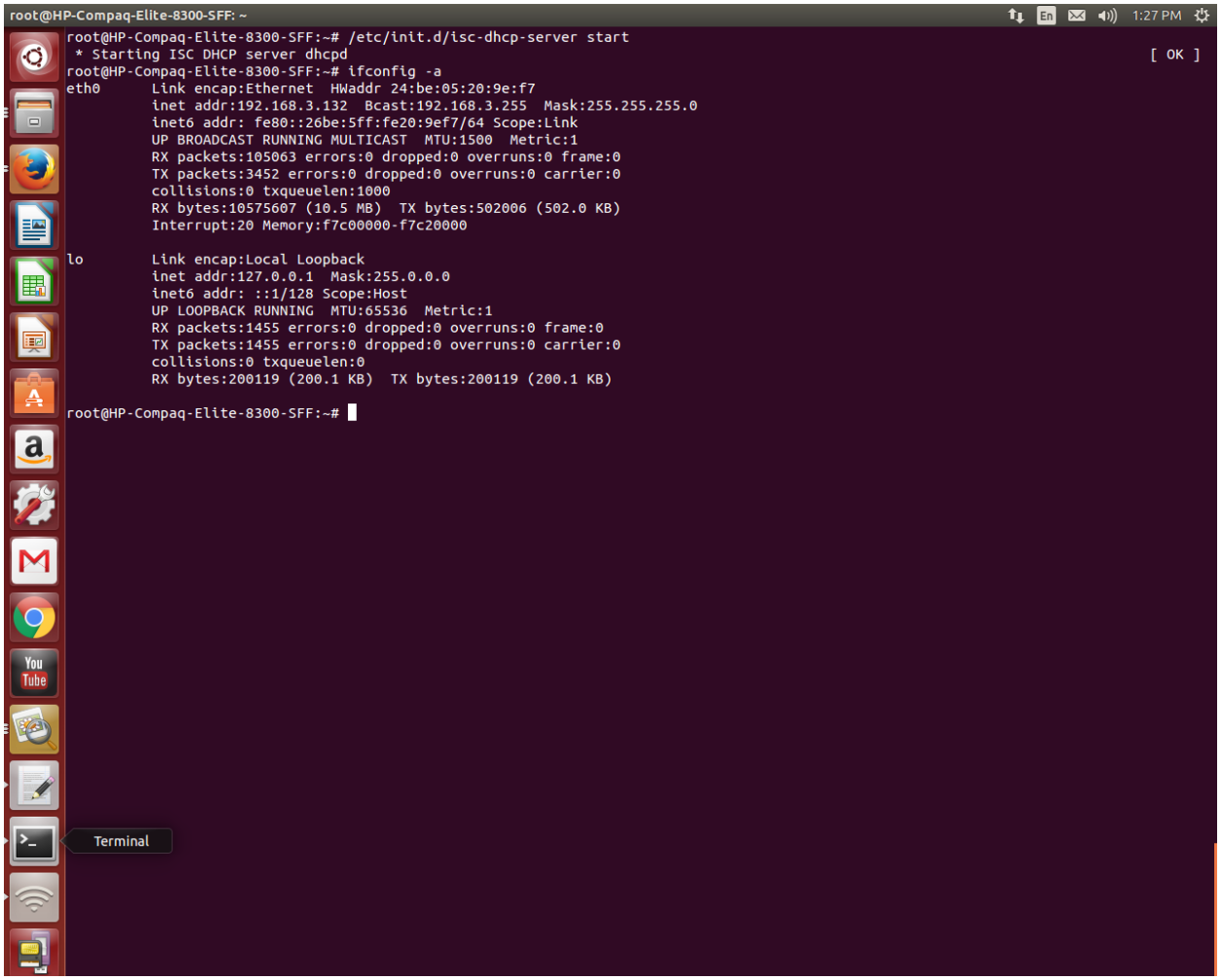


Figure 29 – ‘ifconfig -a’ on command window

#START OF ORIGINAL /etc/dhcp/dhcpd.conf#

#

Sample configuration file for ISC dhcpd for Debian

#

Attention: If /etc/ltsp/dhcpd.conf exists, that will be used as

configuration file instead of this file.

#

#

```
# The ddns-updates-style parameter controls whether or not the server will
# attempt to do a DNS update when a lease is confirmed. We default to the
# behavior of the version 2 packages ('none', since DHCP v2 didn't
# have support for DDNS.)
ddns-update-style none;

# option definitions common to all supported networks...
option domain-name "example.org";
option domain-name-servers ns1.example.org, ns2.example.org;

default-lease-time 600;
max-lease-time 7200;

# If this DHCP server is the official DHCP server for the local
# network, the authoritative directive should be uncommented.
#authoritative;

# Use this to send dhcp log messages to a different log file (you also
# have to hack syslog.conf to complete the redirection).
log-facility local7;

# No service will be given on this subnet, but declaring it helps the
# DHCP server to understand the network topology.

#subnet 10.152.187.0 netmask 255.255.255.0 {
#}
```

```
# This is a very basic subnet declaration.

#subnet 10.254.239.0 netmask 255.255.255.224 {
#  range 10.254.239.10 10.254.239.20;
#  option routers rtr-239-0-1.example.org, rtr-239-0-2.example.org;
#}

# This declaration allows BOOTP clients to get dynamic addresses,
# which we don't really recommend.

#subnet 10.254.239.32 netmask 255.255.255.224 {
#  range dynamic-bootp 10.254.239.40 10.254.239.60;
#  option broadcast-address 10.254.239.31;
#  option routers rtr-239-32-1.example.org;
#}

# A slightly different configuration for an internal subnet.
#subnet 10.5.5.0 netmask 255.255.255.224 {
#  range 10.5.5.26 10.5.5.30;
#  option domain-name-servers ns1.internal.example.org;
#  option domain-name "internal.example.org";
#  option routers 10.5.5.1;
#  option broadcast-address 10.5.5.31;
#  default-lease-time 600;
#  max-lease-time 7200;
#}

# Hosts which require special configuration options can be listed in
```

```
# host statements.   If no address is specified, the address will be
# allocated dynamically (if possible), but the host-specific information
# will still come from the host declaration.

#host passacaglia {
#  hardware ethernet 0:0:c0:5d:bd:95;
#  filename "vmunix.passacaglia";
#  server-name "toccata.fugue.com";
#}

# Fixed IP addresses can also be specified for hosts.   These addresses
# should not also be listed as being available for dynamic assignment.
# Hosts for which fixed IP addresses have been specified can boot using
# BOOTP or DHCP.   Hosts for which no fixed address is specified can only
# be booted with DHCP, unless there is an address range on the subnet
# to which a BOOTP client is connected which has the dynamic-bootp flag
# set.
#host fantasia {
#  hardware ethernet 08:00:07:26:c0:a5;
#  fixed-address fantasia.fugue.com;
#}

# You can declare a class of clients and then do address allocation
# based on that.   The example below shows a case where all clients
# in a certain class get addresses on the 10.17.224/24 subnet, and all
# other clients get addresses on the 10.0.29/24 subnet.

#class "foo" {
```

```
#  match if substring (option vendor-class-identifier, 0, 4) = "SUNW";
#}

#shared-network 224-29 {
#  subnet 10.17.224.0 netmask 255.255.255.0 {
#    option routers rtr-224.example.org;
#  }
#  subnet 10.0.29.0 netmask 255.255.255.0 {
#    option routers rtr-29.example.org;
#  }
#  pool {
#    allow members of "foo";
#    range 10.17.224.10 10.17.224.250;
#  }
#  pool {
#    deny members of "foo";
#    range 10.0.29.10 10.0.29.230;
#  }
#}
```

#END OF ORIGINAL /etc/dhcp/dhcpd.conf#

#START OF MODIFIED /etc/dhcp/dhcpd.conf#

```
#
# Sample configuration file for ISC dhcpd for Debian
#
# Attention: If /etc/ltsp/dhcpd.conf exists, that will be used as
# configuration file instead of this file.
#
```

```
#

# The ddns-updates-style parameter controls whether or not the server will
# attempt to do a DNS update when a lease is confirmed. We default to the
# behavior of the version 2 packages ('none', since DHCP v2 didn't
# have support for DDNS.)
ddns-update-style none;

# option definitions common to all supported networks...
option domain-name "example.org";
option domain-name-servers ns1.example.org, ns2.example.org;

default-lease-time 600;
max-lease-time 7200;

# If this DHCP server is the official DHCP server for the local
# network, the authoritative directive should be uncommented.
#authoritative;

# Use this to send dhcp log messages to a different log file (you also
# have to hack syslog.conf to complete the redirection).
log-facility local7;

# No service will be given on this subnet, but declaring it helps the
# DHCP server to understand the network topology.

#subnet 10.152.187.0 netmask 255.255.255.0 {
#}
```

```
# This is a very basic subnet declaration.

#subnet 10.254.239.0 netmask 255.255.255.224 {
#  range 10.254.239.10 10.254.239.20;
#  option routers rtr-239-0-1.example.org, rtr-239-0-2.example.org;
#}

# This declaration allows BOOTP clients to get dynamic addresses,
# which we don't really recommend.

#subnet 10.254.239.32 netmask 255.255.255.224 {
#  range dynamic-bootp 10.254.239.40 10.254.239.60;
#  option broadcast-address 10.254.239.31;
#  option routers rtr-239-32-1.example.org;
#}

# A slightly different configuration for an internal subnet.
#subnet 10.5.5.0 netmask 255.255.255.224 {
#  range 10.5.5.100 10.5.5.200;
#  option domain-name-servers ns1.internal.example.org;
#  option domain-name "internal.example.org";
#  option routers 10.5.5.1;
#  option broadcast-address 10.5.5.31;
#  default-lease-time 600;
#  max-lease-time 7200;
#}
subnet 192.168.3.0 netmask 255.255.255.0 {
```

```
  interface eth0;
  range 192.168.3.100 192.168.3.200;
  option domain-name-servers ns1.internal.example.org;
  option domain-name "internal.example.org";
  option routers 192.168.3.1;
  option broadcast-address 192.168.3.255;
  default-lease-time 600;
  max-lease-time 7200;
}
# Hosts which require special configuration options can be listed in
# host statements.   If no address is specified, the address will be
# allocated dynamically (if possible), but the host-specific information
# will still come from the host declaration.

#host passacaglia {
#  hardware ethernet 0:0:c0:5d:bd:95;
#  filename "vmunix.passacaglia";
#  server-name "toccata.fugue.com";
#}

# Fixed IP addresses can also be specified for hosts.   These addresses
# should not also be listed as being available for dynamic assignment.
# Hosts for which fixed IP addresses have been specified can boot using
# BOOTP or DHCP.   Hosts for which no fixed address is specified can only
# be booted with DHCP, unless there is an address range on the subnet
# to which a BOOTP client is connected which has the dynamic-bootp flag
# set.
#host fantasia {
```

```
#  hardware ethernet 08:00:07:26:c0:a5;
#  fixed-address fantasia.fugue.com;
#}

# You can declare a class of clients and then do address allocation
# based on that.   The example below shows a case where all clients
# in a certain class get addresses on the 10.17.224/24 subnet, and all
# other clients get addresses on the 10.0.29/24 subnet.

#class "foo" {
#  match if substring (option vendor-class-identifier, 0, 4) = "SUNW";
#}

#shared-network 224-29 {
#  subnet 10.17.224.0 netmask 255.255.255.0 {
#    option routers rtr-224.example.org;
#  }
#  subnet 10.0.29.0 netmask 255.255.255.0 {
#    option routers rtr-29.example.org;
#  }
#  pool {
#    allow members of "foo";
#    range 10.17.224.10 10.17.224.250;
#  }
#  pool {
#    deny members of "foo";
#    range 10.0.29.10 10.0.29.230;
#  }
```

#}

#START OF MODIFIED /etc/dhcp/dhcpd.conf#

#START OF ORIGINAL /etc/network/interfaces#

```
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback
```

#END OF ORIGINAL /etc/network/interfaces#

#START OF MODIFIED /etc/network/interfaces#

```
# interfaces(5) file used by ifup(8) and ifdown(8)
auto lo
iface lo inet loopback
auto eth0
iface eth0 inet dhcp
```

#END OF MODIFIED /etc/network/interfaces#

#START OF ORIGINAL /etc/NetworkManager/NetworkManager.conf#

```
[main]
plugins=ifupdown,keyfile,ofono
dns=dnsmasq

[ifupdown]
managed=false
```

#END OF ORIGINAL /etc/NetworkManager/NetworkManager.conf#

#START OF ORIGINAL /etc/NetworkManager/NetworkManager.conf#

```
[main]
plugins=ifupdown,keyfile,ofono
dns=dnsmasq
```

[ifupdown]
managed=true

#END OF ORIGINAL /etc/NetworkManager/NetworkManager.conf#

In this way, the DHCP server was set up. Once both GPSD and NTPD are running and the DHCP communication has been established, the DHCP server on the computer sends the NTP server information on the computer to the DHCP client.

So, the computer receives time data from the GPS receiver via NTPD tool which indirectly communicates with the GPS using GPSD tool. The time information is then sent by the DHCP server on the computer to the TLS which is a DHCP client. Hence, the GPS receiver and the TLS are synchronised.

Chapter – 7:
WORK TO BE DONE FOR FUTURE DEVELOPMENT

The time synchronisation of the TLS and the GPS receiver has been achieved. But since they are being synchronised indirectly without direct hard-connection, the HMI of the TLS doesn't show the 'Synchronisation: OK' message. So further study can be done on how to configure the servers and the TLS so the above mentioned message gets displayed upon synchronisation.

The previously mentioned executable files can extract different scan parameters like the spatial coordinates and the angular coordinates. These data from the text file created after running the executable files can be used to reconstruct 3D point clouds on any cloud processing software like Cloud Compare and the generated point clouds can be used in many fields and applications like disaster management, fault management, analysis of constituent materials of structures and buildings, and so on.

We are able to transfer only the time information from the GPS receiver to the TLS whereas for proper geo-referencing it is also required to transfer the fix data from the GPS receiver to the TLS. Though the fix data of the GPS is getting transmitted to the computer, it still remains to be seen whether the same data can also be transferred to the TLS for geo-referencing.

Chapter – 8:
CONCLUSION

The time synchronization of the Terrestrial Laser Scanner and the Global Positioning System has thus been achieved using the software based approach as discussed above. We were also successful in our attempt to extract the time stamps and other necessary scan attributes from the scans and we could also automate the initiation of scanning by the TLS with pre-set parameters by using indigenously built executable files. By following the above mentioned approach, a mobile mapping system consisting of TLS, GPS (in our case it is IRNSS receiver) and an IMU can be set up. Along with that, a mobile platform which can either be a trolley for lower order application or car for higher order application will be required. A laptop with internet connection for setting up the servers is also required. Thus, we can synchronise the devices and get a complete geo referenced scan and the time stamp information along with other scan attributes generated with the help of the RivLib library can be reconstructed to generate a 3D image in any point cloud processing software like Cloud Compare. The data, obtained in this manner, can be used to detect cracks and faults or other important features of the site which can be used for other applications.

Chapter – 9:
REFERENCES

9.1. Research papers

[1]. Mobile mapping by Joshua I. France.

[2]. 6DOF Semi-Rigid SLAM for Mobile Scanning,Jan Elseberg, Dorit Borrmann, and Andreas N¨uchter

[3]. Registration of Long-Strip Terrestrial Laser Scanning Point Clouds Using RANSAC and Closed Constraint Adjustment by Li Zheng , Manzhu Yu , Mengxiao Song , Anthony Stefanidis , Zheng Ji and Chaowei Yang

[4].Michael Bosse and Robert Zlot. Continuous 3D scan-matching with a spinning 2D laser scanner.

[5]. In IEEE ICRA ’09, pages 4312 –4319, May 2009. B. J. Brown and S. Rusinkiewicz. Global Non-Rigid Alignment of3-D Scans. ACM Transactions on Graphics, 26(3):21, 2007.H. Chui and A. Rangarajan

[6]. A New Point Matching Algorithm forNon-Rigid Registration. CVIU, 89(2-3):114–141, 2003.B. Pitzer and C. Stiller.

[7]. Probabilistic mapping for mobile robots using spatial correlation models. In IEEE ICRA, pages 5402–5409, May2010.Todor Stoyanov and Achim J. Lilienthal. Maximum likelihood pointcloud acquisition from a mobile platform. In Proc. of the IEEE ICAR,

[8]. June 22–26 2009. J. A. Williams, M. Bennamoun, and S. Latham. Multiple View 3D Registration: A Review and a New Technique .In Proc. of the Int.Conf. On Systems, Man, and Cybernetics, 1999.

[9]. 6-DOF Localization for a Mobile Robot using Outdoor 3D Voxel Maps by Taro Suzuki, Mitsunori Kitamura, Yoshiharu Amano and Takumi Hashizume.

[10]. Processing the point cloud with Riscan Pro or Riprofile Cyber Mapping Lab UT-Dallas

[11]. A MAN-PORTABLE, IMU-FREE MOBILE MAPPING SYSTEM Andreas N¨uchter∗, Dorit Borrmann∗, Philipp Koch+, Markus K¨uhn+, Stefan May Informatics VII – Robotics and Telematics Julius-Maximilians University Wurzburg, Germany Faculty of Electrical Engineering, Precision Engineering, Information Technology
Nuremberg Institute of Technology Georg Simon Ohm, Germany

[12]. Overcoming the Level Bubble: Terrestrial Laser Scanning Reference

9.2. Website Links

http://www.lammertbies.nl/comm/info/GPS-time.html
http://www.ntp.org/
http://www.ntp.org/downloads.html
https://www.eecis.udel.edu/~mills/ntp/html/index.html
https://www.eecis.udel.edu/~mills/ntp/html/build.html
http://askubuntu.com/questions/25347/what-command-do-i-need-to-unzip-extract-a-tar-gz-file
http://askubuntu.com/questions/262068/how-to-extract-a-tar-gz-file
http://askubuntu.com/questions/499807/how-to-unzip-tgz-file-using-the-terminal
http://www.hostingadvice.com/how-to/untar-file-linuxubuntu/
http://www.cyberciti.biz/faq/linux-unix-bsd-extract-targz-file/
http://askubuntu.com/questions/974/how-can-i-install-software-or-packages-without-internet-offline
https://askubuntu.com/questions/86358/how-to-obtain-installed-package-files/86413#86413
http://askubuntu.com/questions/339/how-can-i-install-a-package-without-root-access
https://help.ubuntu.com/community/AptGet/Howto
https://help.ubuntu.com/community/InstallingSoftware
http://www.catb.org/gpsd/
http://www.catb.org/gpsd/installation.html
http://fossies.org/linux/gpsd/build.txt
http://www.catb.org/gpsd/troubleshooting.html
https://www.howtoinstall.co/en/ubuntu/trusty/gpsd-clients
http://askubuntu.com/questions/86822/how-can-i-copy-the-contents-of-a-folder-to-another-folder-in-a-different-directo
http://askubuntu.com/questions/80065/i-want-to-copy-a-directory-from-one-place-to-another-via-the-command-line
http://www.cyberciti.biz/faq/copy-folder-linux-command-line/
http://www.krizna.com/ubuntu/setup-dhcp-server-ubuntu-14-04/
https://help.ubuntu.com/community/isc-dhcp-server
https://help.ubuntu.com/community/dhcp3-server
http://askubuntu.com/questions/140126/how-do-i-install-and-configure-a-dhcp-server
https://www.chegg.com/tutors/opportunities/
http://askubuntu.com/questions/430853/how-do-i-find-my-internal-ip-address
http://www.howtogeek.com/howto/17012/how-to-find-your-ip-address-in-ubuntu/
http://askubuntu.com/questions/197628/how-do-i-find-my-network-ip-address-netmask-and-gateway-info
http://askubuntu.com/questions/278756/dhcp-server-not-starting
http://askubuntu.com/questions/398442/problem-configuring-dhcp-server-job-failed-to-start
http://www.webopedia.com/TERM/S/subnet_mask.html
http://www.linuxfromscratch.org/blfs/view/7.8/basicnet/dhcp.html
https://wiki.debian.org/DHCP_Client
http://www.computerhope.com/unix/dhclient.htm

https://linuxconfig.org/what-is-dhcp-and-how-to-configure-dhcp-server-in-linux
https://help.ubuntu.com/lts/serverguide/dhcp.html#dhcp-configuration
http://askubuntu.com/questions/449876/dhcp-server-client?rq=1
https://www.freebsd.org/doc/handbook/network-dhcp.html
http://askubuntu.com/users/logout
http://askubuntu.com/questions/112885/dhcp-server-not-routing-to-connect-to-internet-for-clients?rq=1
http://stackexchange.com/
https://help.ubuntu.com/lts/serverguide/dhcp.html
https://calomel.org/dhclient.html
http://serverfault.com/questions/329596/how-to-override-the-ntp-information-sent-by-dhcp-in-debian
http://doc.ntp.org/3-5.93e/ntpq.html
https://rbgeek.wordpress.com/2012/04/30/time-synchronization-on-ubuntu-12-04lts-using-ntp/
https://www.eecis.udel.edu/~mills/ntp/html/ntpq.html
https://help.ubuntu.com/lts/serverguide/network-configuration.html
http://manpages.ubuntu.com/manpages/wily/man5/dhclient.conf.5.html
http://manpages.ubuntu.com/manpages/wily/man5/dhcp-options.5.html
http://askubuntu.com/questions/30569/how-to-use-gps-receiver-bu-353
http://www.catb.org/gpsd/gpsd-time-service-howto.html
http://ubuntuforums.org/showthread.php?t=966569
http://stackoverflow.com/questions/17374120/error-trying-to-compile-v8-on-osx-lion-scons-no-sconstruct-file-found
http://stackoverflow.com/questions/17182799/scons-no-sconstruct-file-found
http://www.catb.org/gpsd/gpsd-time-service-howto.html#RFC-2783
https://tools.ietf.org/html/rfc2783
https://www.kernel.org/doc/Documentation/pps/pps.txt
https://lists.nongnu.org/archive/html/gpsd-users/2015-10/msg00018.html
http://unix.stackexchange.com/questions/120992/why-does-cat-ttyusb0-not-produce-output
http://unix.stackexchange.com/questions/200302/reading-dev-ttyusb0
http://www.catb.org/gpsd/gpsctl.html
http://askubuntu.com/questions/453072/what-is-nss-myhostname-and-why-is-it-not-installable