The Pennsylvania State University

The Graduate School

Department of Mechanical and Nuclear Engineering

DESIGN AND TESTING OF A TERRAIN MAPPING SYSTEM FOR MEDIAN SLOPE MEASUREMENT

A Thesis in

Mechanical Engineering

by

Pramod Vemulapalli

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science

Dec 2008

The thesis of Pramod Vemulapalli was reviewed and approved* by the following:

Sean N. Brennan
Assistant Professor of Mechanical Engineering
Thesis Advisor

Henry J. Sommer III
Professor of Mechanical Engineering

Karen A. Thole
Professor of Mechanical Engineering
Head of the Department of Mechanical and Nuclear Engineering

*Signatures are on file in the Graduate School

ABSTRACT

This thesis presents the details of a terrain mapping system that has been used to measure median profiles. Details are presented of the system construction, design, and algorithms to process the data. The capability of the system is gauged by performing tests to compensate for vehicle orientation, tests for repeatability, and tests comparing the system with manual measurement. Results demonstrate the feasibility of using the system for median slope measurement. The advantages of using such a system as compared to traditional manual measurements are also demonstrated in practice as this system has been used to scan over 5000 miles of highways for measuring the slopes of divided rural medians. While the research focuses on median slope measurement, the thesis can be viewed as providing a general direction for the design and testing of terrain mapping systems.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

Firstly, I would like to thank and acknowledge all the community activists, organizers and social workers who have made this world a better place and the fruits of whose sacrifice and action I am currently enjoying.

I would like to thank the Department of Mechanical Engineering and the Thomas D. Larson Pennsylvania Transportation Institute for providing support for this work. I would like to express my gratitude for my advisor, Dr.Brennan, whose patience and enthusiasm was invaluable. I would also like to thank him for being kind and understanding during all the variety of situations that I faced during the last two years. I admire his commitment to teaching and I am amazed by the time and energy he channels into this passion.

I would also like to thank Dr.Sommer for taking time out of his busy schedule to be my reader. I would like to thank Vishi for all his help, from my first semester at Penn State. Vishi's candid and thoughtful analysis of any situation has been incredibly helpful all throughout. Adam and Sittikorn are great guys and have been awesome to work with.

I would like to thank my family for their support and encouragement in all my endeavors. I would like to thank my parents who have whole heartedly dedicated the better part of their lives to making sure that I had better opportunities than they ever did. I would like to thank my maternal grandmother for her unconditional love and support from the day I was born. I would like to thank my paternal grandmother whose selfless work for the community and the family has always inspired me.

Finally, I would like to thank all the good friends that I have made at Penn State and my old friends from IITM for all their time and company.

Chapter 1

Introduction

Median geometry is critical in determining the nature of crashes that occur in a particular median, and selection of geometry requires tradeoffs: “flattened” medians may result in more crossover median crashes, while medians which are steep might cause the vehicle to rollover. The makeup in the vehicle fleet has also affected aspects of this tradeoff, particularly in regard to rollover. In the past decades, consumers have purchased a significant number of Sport Utility Vehicles (SUVs) which have a greater propensity to overturn than some of the smaller cars. The increased travel speeds and traffic volumes also call to attention the need to make possible changes to the American Association of State Highway and Transportation Officials (AASHTO) policy on geometric design of highways and streets, a policy which has remained largely unchanged over the last many years (1). Median geometry also plays a pivotal role in the selection and evaluation of in-median corrective factors such as median barriers. In particular, the relative positioning of a median barrier inside a median directly affects the nature and number of crashes that happen on that median.

While there are a number of research projects that seek to find analytical or descriptive models relating median design to crash incidents, a key shortcoming is the lack of knowledge of median geometries on existing roadways. For example, a descriptive method would be to check for statistical correlation between the median geometry on an existing segment and the roll over and cross over crashes that have occurred therein. However, details of the median geometry within that particular segment are crucial, but often missing. Or one might use an analytical approach where one simulates the behavior of different vehicles on an idealized slope using

vehicle dynamics software packages. However, the selection of representative “idealized” slopes must be motivated by existing median geometries.

Careful study of median geometric design clearly requires collection of significant amounts of median data, a process which in turn requires an easy and reliable method to obtain the median geometry. This thesis describes the design of an automated Light Detecting and Ranging (LIDAR) based terrain mapping system which has been used to collect the median profile data for the study. This automated system satisfies a number of design constraints including:

- use of only off-the-shelf items to stay within budget and time constraints,
- keeping the entire system portable so that it could be shipped to any location for rapid mapping,
- allowing the design to be fitted to any existing large sized rental SUV without causing any permanent changes or making any custom modifications to the vehicle,
- developing a simple and intuitive GUI to control the equipment when on the road ,
- designing software that can process the huge amount of data collected by the system and extract the parameters of median geometry (the adjacent slope and the opposing slope),
- testing the reliability of the system through repeated deployments in a wide range of roadway conditions,
- testing the repeatability of the system to evaluate expected variability of the data, and
- calibrating and conducting error analysis to understand the most common sources of error and accuracy versus existing methods of median measurement.

This thesis presents a system developed to meet these needs, and in particular details the error and repeatability analysis aspects of the system design and evaluation process.

Chapter 2

Literature Survey

The most common modern practice to survey wide swaths of geometry is the use of LIDAR, and LIDAR-based terrain mapping technology has been demonstrated across a variety of applications. While aerial LIDAR technology has been present for some time and has been extremely useful in survey applications (2), there are some limitations with regard to the level of detail that could be obtained by this means. For example, a recent study by Souleyrette (3) revealed that it was infeasible to extract shoulder slope data from aerial LIDAR because of the narrowness of the shoulder. Road vehicle-based LIDAR mapping technology is another option which typically involves collecting terrain information from LIDARs and Global Positioning System – Inertial Measurement Unit (GPS-IMU) units mounted on a vehicle. This approach has the advantage of providing the required amount of detail to accurately measure parameters like the road cross-slope, median geometry etc., but has the disadvantage of complexity in calibration and removing vehicle motion, issues discussed herein. Road cross-slope measurement with LIDARs was initially patented by Mekemson (4). It must be noted that this patent delves into measuring the cross slope of the pavement and is different from the work in this thesis which looks into the measuring the cross slope of off-road features like medians. In this patent, a ring-laser gyroscope is used to specify the roll of the vehicle with respect to a level datum and the LIDARS, which are placed on either side of a platform extending from the rear of the vehicle, are used to measure the roll of the vehicle with respect to the pavement. By subtracting out the roll of the vehicle with respect to the pavement from the roll measured from the ring laser gyro, this method is able to measure the cross slope of the pavement very accurately. In our experiments we have assumed that the roll of the vehicle with respect to the pavement is negligible, this

assumption was based on the work done by Mraz (5) who has shown that accurate measurement of the cross slope of the road is possible by simply using a well calibrated GPS-IMU system mounted on a vehicle. Some of the other applications for which LIDAR based mapping has been used are in road profilometry, for measuring pavement and ride quality (6)(7) (8) (9) (10) (11). These applications typically use high frequency and high accuracy LIDAR systems that usually measure the pavement surface from close proximity (6-10 inches off the ground) and determine the smoothness of the pavement and thereby access whether it meets ride quality standards. Active research is also being conducted in terrain mapping with the aid of LIDARs for its applications in robotics (12) (13) (14). This area is the closest in terms of the sensors and systems used in our design and while the applications in robotics are typically satisfied with detecting and extracting the obstacles in the path of the robot by processing the LIDAR data, we were more concerned about how accurately the LIDAR was measuring the terrain profile.

Another method of terrain mapping is to utilize additional sensors like cameras on the vehicles. This provides interesting opportunities to fuse the information obtained from multiple sensors (15)(16), and vision sensors can also be used to extract interesting features such as road conditions (17), lane markers etc. An example combining vision and LIDAR sensors to obtain road and off-road features for road safety has been presented by the ARRB group (18) (19).

While one may see parallels in some of the above work and the work presented in this thesis, it is important to note that this is the first time a detailed study has been performed to study the feasibility of utilizing a terrain based LIDAR scanning system for the purpose of measuring off-road features such as median slopes.

Chapter 3

Data Acquisition System

The sensors present in a typical terrain mapping vehicle are a LIDAR, a GPS and an IMU. While the LIDAR scans the environment and provides the terrain information relative to the vehicle, the global position of the vehicle is measured at relatively long intervals by the GPS. The orientation and fine motion of the vehicle is measured by the IMU. One can observe that a particular challenge to operate mobile mapping systems is to obtain very accurate position and orientation information at very high bandwidth by fusion of GPS and IMU data, a process which requires high accuracy and high-bandwidth IMU's. Only within the past decade have such units been available at reasonable cost, outside of defense applications.

To measure information from these three sensors simultaneously, an integrated data acquisition system is needed. Such a system was developed (Figure 3-1, Figure 3-2), and the resulting unit is a portable instrument frame that has been designed to acquire data from multiple sensors simultaneously. The main aspects of the system including power electronics, sensor systems, data routing architecture and data acquisition software are described below:

Power Electronics

A key problem with mobile data acquisition, particularly with a moderate power LIDAR scanner, is the issue of power quality. To solve this, a complete stand-alone power system was developed alongside the data-collection system. A two-level design of the instrument frame was adopted to separate the power electronics systems from the sensor and the computer systems; the power circuitry is located on the bottom level. The details of the architecture for the power

circuitry are specified in Appendix A; this system basically combines the power input from the on-board battery packs and the vehicle's alternator to provide well-regulated output independent of the highly variable vehicle power system.

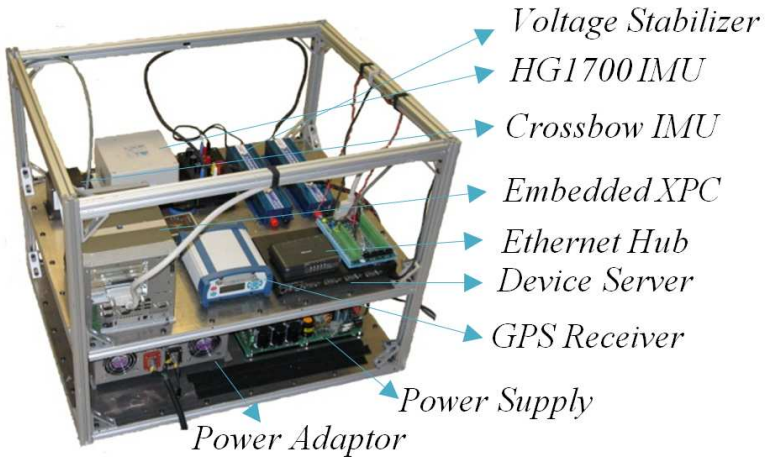

Figure **3-1**: Data Acquisition Platform

Figure **3-2**: The Entire Data Acquisition System When Mounted on a Vehicle

GPS – IMU Unit

To obtain integrated GPS-IMU data at the rate of 100Hz, the NovAtel's Synchronized Position Attitude Navigation (SPAN) system was used, based on an OEM4 DL4-PLUS GPS receiver and the HONEYWELL HG1700 military tactical IMU. This is a defense-grade system whose position errors in the latitude and longitude data, with full satellite visibility, are about 2 meters (one sigma) and the errors in the orientation angles are 0.017, 0.02 and 0.042 degrees (one sigma) for the roll, pitch and the yaw angles respectively (20) (21). While the GPS location errors are large, the high-accuracy IMU filters the errors such that the data exhibits a very slowly drifting bias, not a measurement-to-measurement random change typical of most GPS systems. The orientation accuracies are critical in determining the repeatability and the accuracy of the terrain data mapped by using this system.

LIDAR Unit

The LIDAR sensor used on the system is the SICK LMS 291. With a range of up to 30 meters and accuracy of +/- 35mm, it is able to view most traversable medians, e.g. medians without obstructions to vehicle motion such as barriers, trees, etc. It has a scan rate of 37.5 Hz, and each scan includes 361 LIDAR data points subtending an angle of 180 degrees at 0.5 degree increments. The data rate of the LIDAR corresponds to having a combined LIDAR-GPS-IMU data packet, and hence one complete lateral scan, once every 0.8 meters of road when travelling at highway speeds (30 meters/sec).

Data Routing Architecture

The schematic in Figure 3-3 illustrates the data routing architecture of the instrumentation setup. The setup consists of an Ethernet hub which routes data between the sensors and the data acquisition laptops. A network of sensors approach was used because it facilitates distributed processing of the data and complex command and control structures through different laptops.

Software

To facilitate debugging, the data acquisition interface was coded in Simulink within a Windows environment. The LIDAR acquisition code was written in the PLAYER environment (21). The field data is then post-processed to obtain the adjacent and the opposing median slopes using a code written in MATLAB.

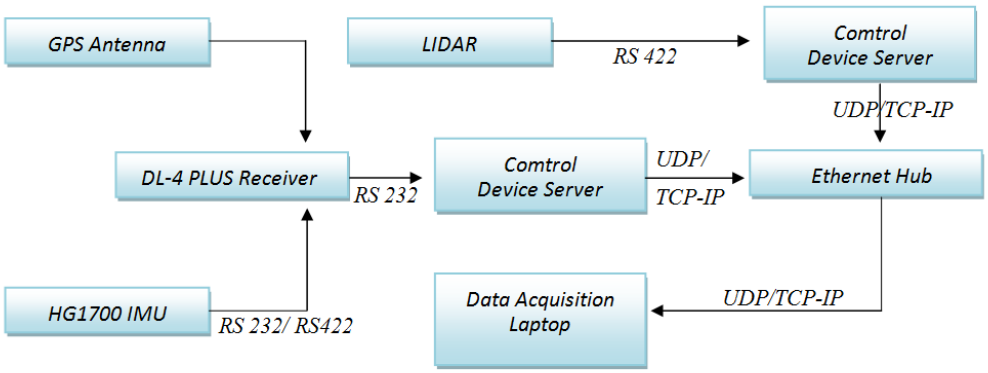

Figure **3-3**: Data Routing Architecture

Chapter 4

Data Processing Algorithm

Each scan contains within it information that provides estimates of the adjacent and the opposing slope, but extraction of this information is not trivial. The task of the data processing algorithm can be broken down into four main parts:

Step 1) Correct the LIDAR scan for vehicle orientation.

The LIDAR is positioned on the vehicle to look down perpendicularly to the road, orthogonal to the direction of travel, and any deviation from the perpendicularity of the LIDAR with respect to the road must be corrected. Static offsets are initially identified through an offline calibration routine. Dynamic offsets are caused mainly by vehicle roll angle changes while on the road, pitch and yaw effects are both found to be minor. To correct for the roll, a single LIDAR data point (r, θ) where r is the distance of the LIDAR hit at an angle θ. The equations to transform the LIDAR data into a Cartesian coordinate system while compensating for just the vehicle roll angle (α) and the initial calibration angle (φ) are as follows:

$$x = r\ \cos(\theta - \alpha - \varphi)$$

$$y = r\ \sin(\theta - \alpha - \varphi)$$

Figure 4-1 shows the effect of this transformation on a LIDAR scan. Once the coordinate data is obtained the data is re-sampled so that the final data is at regular intervals of the x coordinate. This re-sampled data is used in all subsequent analysis.

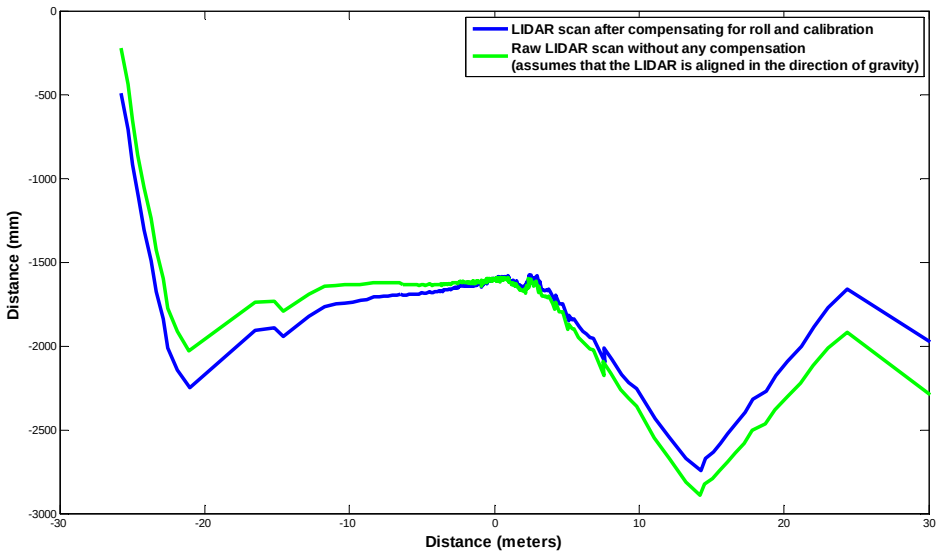

Figure **4-1**: Compensating for Roll and Calibration in the LIDAR Scan

Step 2) Identify the road and road edge

As the LIDAR is setup perpendicular to the road, the LIDAR data points obtained from immediately underneath the LIDAR are assumed to be from the road. These form a very smooth line up to the point of the road edge, and by applying regression, a road line (RL) is identified.

Once the road line is identified (Figure 4-2), the edge of the road must be inferred and because this determination is based on LIDAR data, it might not be the true road edge. A definition to arrive at the edge of the road from the LIDAR data points, given the equation for the line of the road, has been formulated by incorporating multiple thresholds as a single threshold would be very noise sensitive. The definition uses the metric $perp(\alpha, \beta) = \left|\frac{a\alpha+b\beta+c}{\sqrt{(a^2+b^2)}}\right|$ which is the perpendicular distance of a point (α, β) from a line $ax + by + c = 0$. The definition presented here has been used in computing the edge of the adjacent and the opposing slopes as well. We also define the

Point of Significant Departure(PSD): Given a set of n points (x_i, y_i) where $i \in C$ and $C = \{1,2,3 \ldots . n\}$. Suppose there exists a line $ax + by + c = 0$ and there exists $j \in C$, such that j is the smallest value that satisfies

$$perp(x_p, y_p) > thre1 \quad for\ all\ j < p < j + thre2$$

Then the PSD is defined as the point (x_m, y_m) where $m \in C$ and m is the smallest value that satisfies

$$perp(x_q, y_q) > thre3 \quad for\ all\ m < q < j$$

where $thre1, thre2$ and $thre3$ are empirically determined thresholds and $thre3$ is set lower than $thre1$. The road edge is defined as the PSD of the road line.

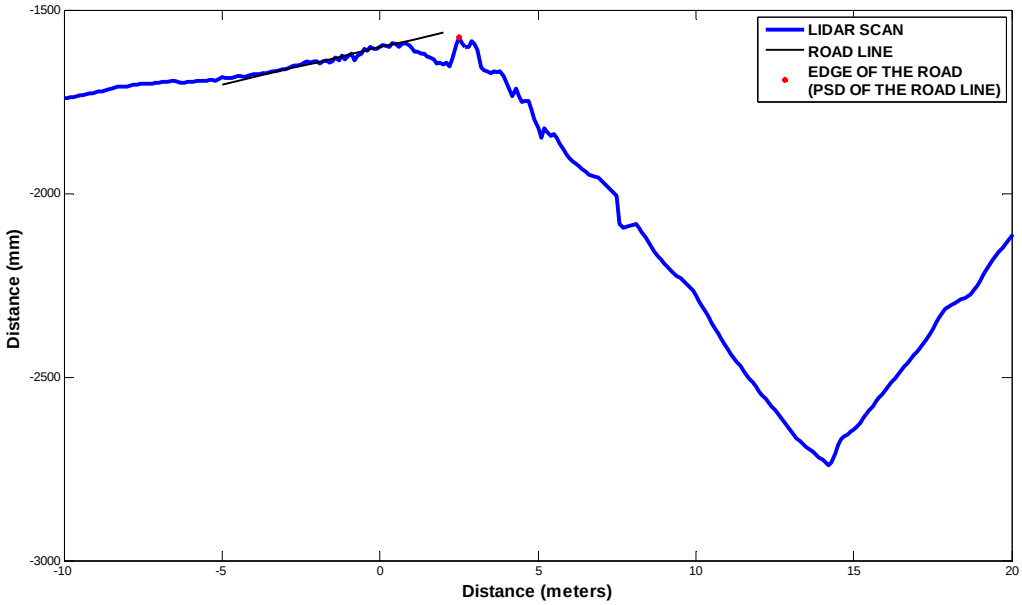

Figure **4-2**: The Figure Shows the Identification of a Road Line and the Associated Edge of the Road by Using that Line.

Step 3) Given the road edge, identify the adjacent slope from the scan and approximate it to a line and then identify the edge of the adjacent slope.

The search algorithm starts from the road edge and considers a small length of the median each time and finds the point in that stretch whose height is the statistical median of all the heights within that stretch. The slope of the tangent to the profile at this point is calculated. The slope of this tangent is varied and a few lines from the resulting set of lines are chosen based on a set of criterion, presented below. Each line that satisfies the criterion is characterized by the means of an optimization function which calculates the number of points whose perpendicular distance from the line is less than a threshold (25mm). This process is repeated along the whole median until no lines satisfying the above criterions are found. Figure 4.3 shows the lines which have been tried, by the search algorithm, in blue and those which match the set of criterion in black. We can see that as we reach closer to the end of the adjacent slope the number of black lines decreases until there are not any lines that satisfy the criterion and the search is terminated. Once the search is terminated and the line having the optimal value, e.g. the most points fit by that line, is selected as the adjacent slope (red line), the PSD of the line is selected as the edge of the adjacent slope.

The set of criterion used to select a line are as follows:

- The PSD for the line is identified and the line is checked to confirm that the length of the data segment fitting this line is within reasonable limits (between 1.5m and 10m).
- If the slope of the line is not within a reasonable limit (between 4 degrees and 18 degrees), the line is eliminated.
- If the perpendicular distance of any point between the road edge and the PSD for the line is beyond a certain threshold (250mm) away from the line, the line is considered to have an obstruction and is eliminated.

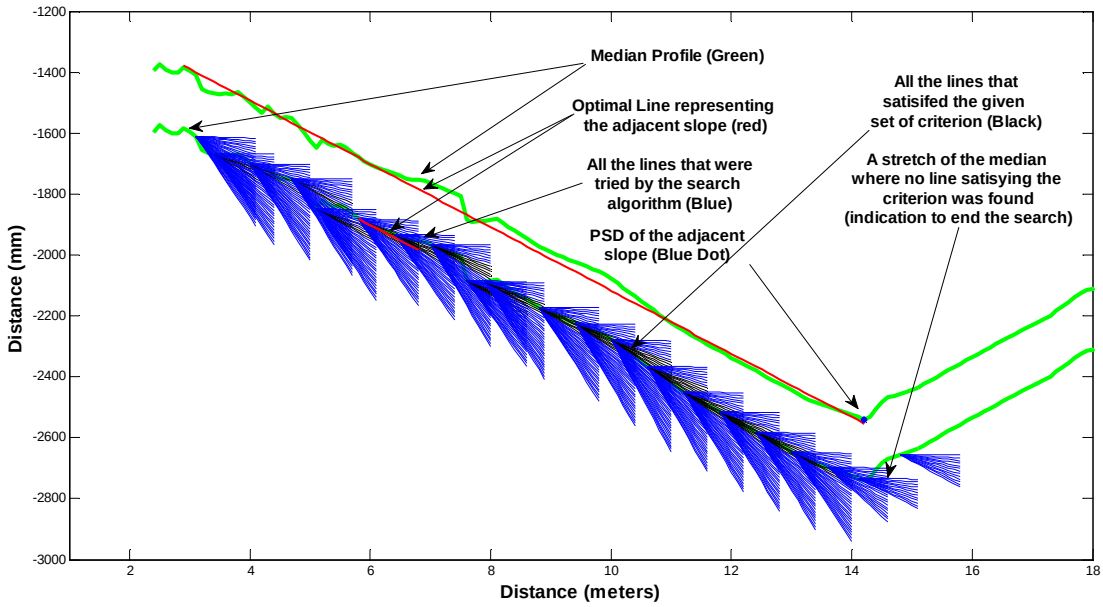

Figure **4-3**: The Figure Illustrates the Processing Done by the Search Algorithm. The Final Result of the Search Algorithm has been Lifted Up for Visibility.

Step 4) Given the edge of the adjacent slope, identify the opposing slope from the scan and approximate it to a line and then identify the edge of the opposing slope.

Step4 is identical to Step3 and it finds the opposing slope. The PSD for the line representing the opposing slope is identified as the edge of the opposing slope. As an example of these steps, Figure 4-4 illustrates a raw LIDAR data scan and the data points and lines obtained after processing the scan. Appendix B contains the MATLAB code for the data processing algorithm presented in this chapter.

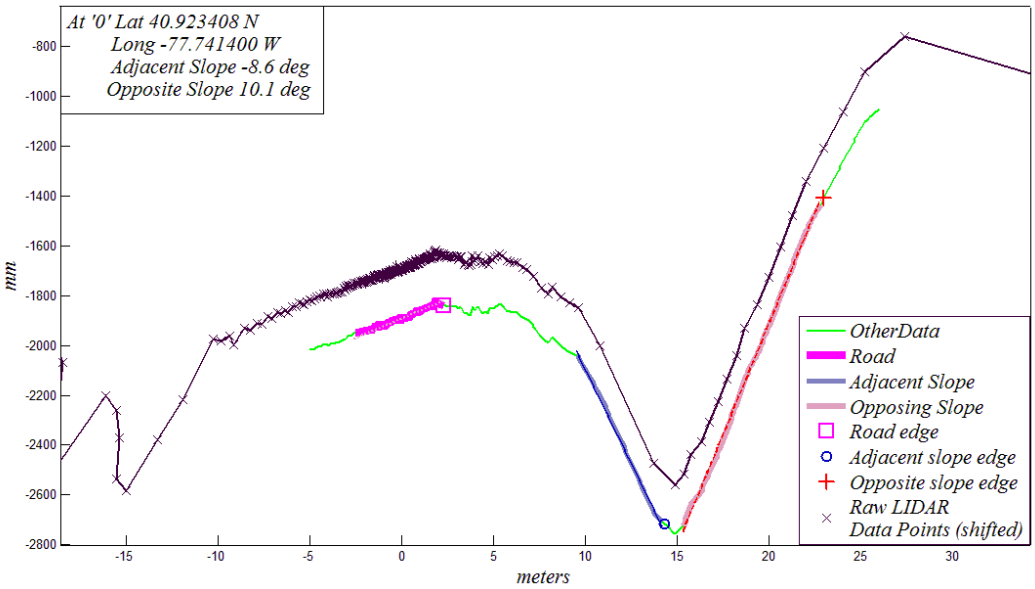

Figure **4-4**: The Processed Scan Data for a Typical Road Cross Section Showing the Road, the Adjacent Slope and the Opposing Slope as Identified by the Algorithm. (The Raw LIDAR Scan Data has Also Been Shown and has been Shifted Up for Visibility)

Chapter 5

Tests and Analysis

One of the important design considerations for the system was to choose the location of the IMU. Ideally one would want the IMU to be placed directly on top of the LIDAR so that the dynamics of the LIDAR system during the scanning process could be directly measured. The IMU unit has been placed inside the vehicle for safety purposes as shown in Figure 5-1, but this placement assumes that in-vehicle IMU measurements are equivalent to measurements made with the IMU rigidly mounted to the sensor. To confirm this assumption, an experiment was conducted with the IMU mounted in both locations while the vehicle is driven as close as possible to the same path. The results (Figure 5-2) indicate a very small average difference of 0.22 degrees and a maximum error of 0.66 degrees. This agreement is mainly because the frame holding the LIDAR is very rigid. During operation, the only observable motion of the sensor relative to the vehicle is a small oscillation in the vertical direction; this would only affect the pitch angle of the LIDAR but not the roll. Figures 5-2, 5-3 and 5-4 detail the plots of the roll, pitch and yaw measurements under the two different conditions. While the pitch and yaw values do not affect the median slope calculation, they are of importance in any three dimensional mapping application. Mean errors of 0.13 degrees and 0.35 degrees, and maximum errors of 1.08 degrees and 0.96 degrees were observed between the pitch readings and the yaw readings under the two different conditions. It must be noted that a significant portion of the error in these tests could be because of the small differences in the path of the vehicle between the two times the experiments were performed.

Figure **5-1**: The Figure Shows the System Configuration in Which the Roll of the Vehicle is Measured by Placing the IMU on the LIDAR.

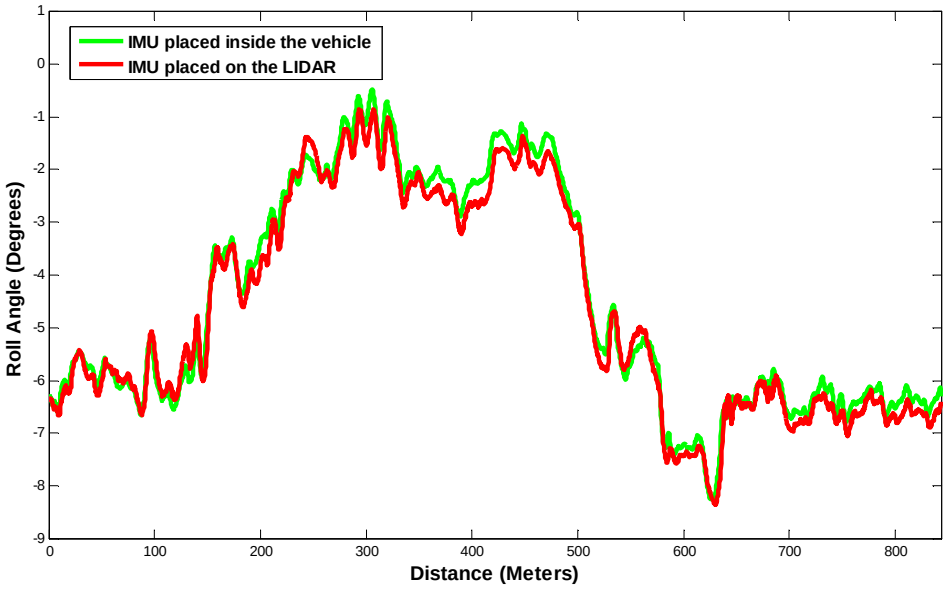

Figure **5-2**: The Figure Shows the Roll, Experienced by the IMU, Measured Under Two Different Conditions in Which the IMU is Placed Inside the Vehicle (Figure **3-2**) and on the LIDAR (Figure **5-1**).

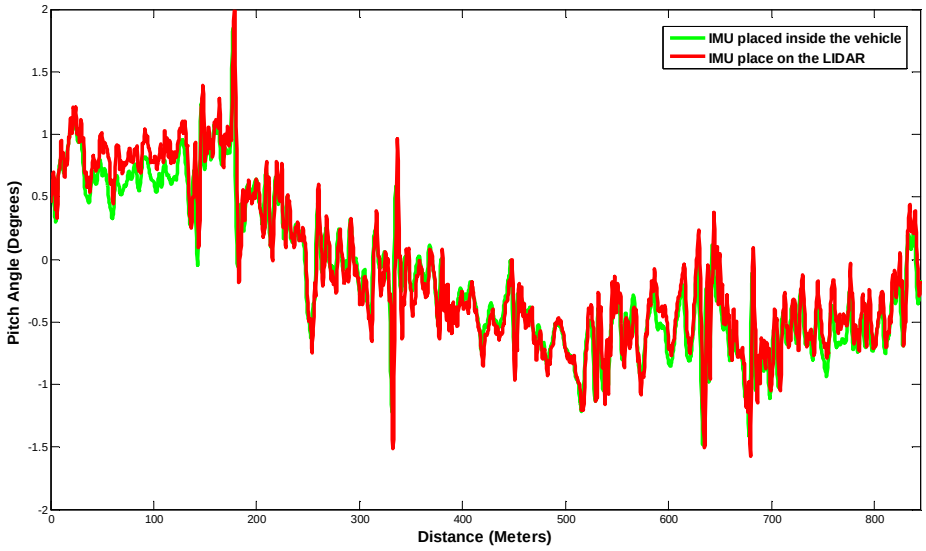

Figure **5-3**: The Figure Shows the Pitch, Experienced by the IMU, Measured Under Two Different Conditions in Which the IMU is Placed Inside the Vehicle (Figure **3-2**) and On the LIDAR (Figure **5-1**).

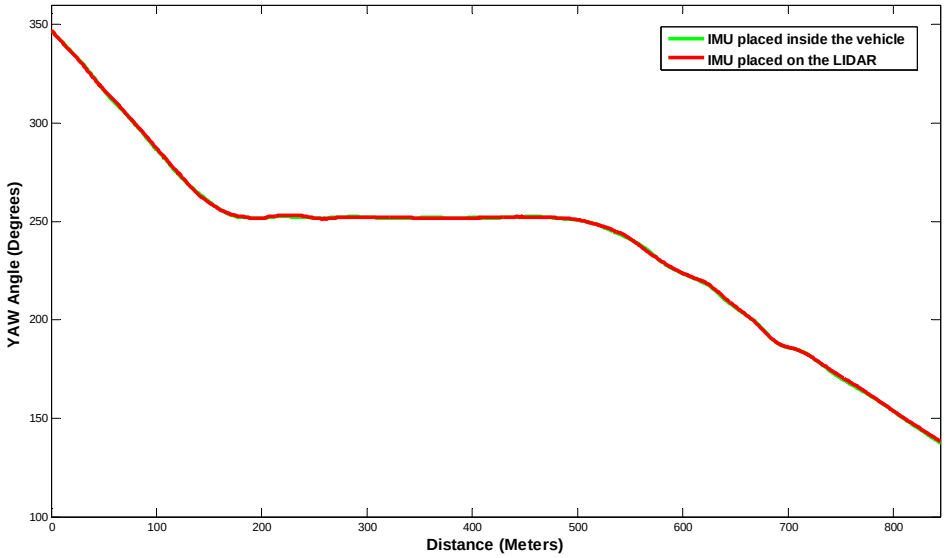

Figure **5-4**: The Figure Shows the Yaw, Experienced by the IMU, Measured Under Two Different Conditions in Which the IMU is Placed Inside the Vehicle (Figure 3-2) and on the LIDAR (Figure **5-1**).

To illustrate the ability of the IMU data to dynamically compensate for vehicle roll in the LIDAR scan data, the data collection vehicle was rocked violently in roll while it was parked on a horizontal surface and simultaneously scanning. This is a worst case scenario because the roll amplitude is significantly higher than what would be expected by the vehicle under ordinary road conditions. The rocking motion moves the LIDAR and consequently the horizontal surface underneath it as seen from the perspective of the LIDAR. While the roll of the vehicle can be obtained from the IMU onboard, the change in the roll of the vehicle can also be computed by observing the change in the slope of the ground as observed by the LIDAR, since the same surface is being repeatedly measured.

Figure 5-5 illustrates the data collected by the sensors in this test, and the small diagram to the bottom-right corner of the figure illustrates the back-view of the motion of the vehicle as this test is performed. The close match between the blue line, indicating the change in roll as measured by the IMU, and the green line, which is the change in the slope of the horizontal surface as seen through the LIDAR, indicates the effectiveness of using the IMU for roll compensation. There is a small phase difference between the IMU data and the LIDAR slope measurement due to buffers and other time delays in the electronic equipment associated with the 37 Hz scan rate. This constant phase delay is easily corrected in post-processing, and with this correction, the difference between the slopes is indicated by the black line in Figure 5-5, and the difference amounts to a standard deviation of the error of about 0.03 degrees.

The effect of the compensation on the LIDAR scan data acquired when rocking the vehicle is clearly illustrated in Figure 5-6. In this figure, the raw LIDAR data of the horizontal surface underneath the vehicle is plotted in red, the LIDAR data with roll compensation is plotted in blue, and the LIDAR data with roll compensation and phase correction is plotted in green. Higher accuracy might be obtained if each point within a scan were delay-corrected individually rather than the entire scan as implemented here; however, this level of accuracy was found

unnecessary for the mapping task. After scan-level compensation, the maximum error in this test was about 0.2 degrees. The vertical band of the green data at 0 meters in Figure 5-6 is due to a combination of the vertical motion of the frame holding the LIDAR when the vehicle was rocked and the error in the LIDAR measurements (+/- 3.5mm).

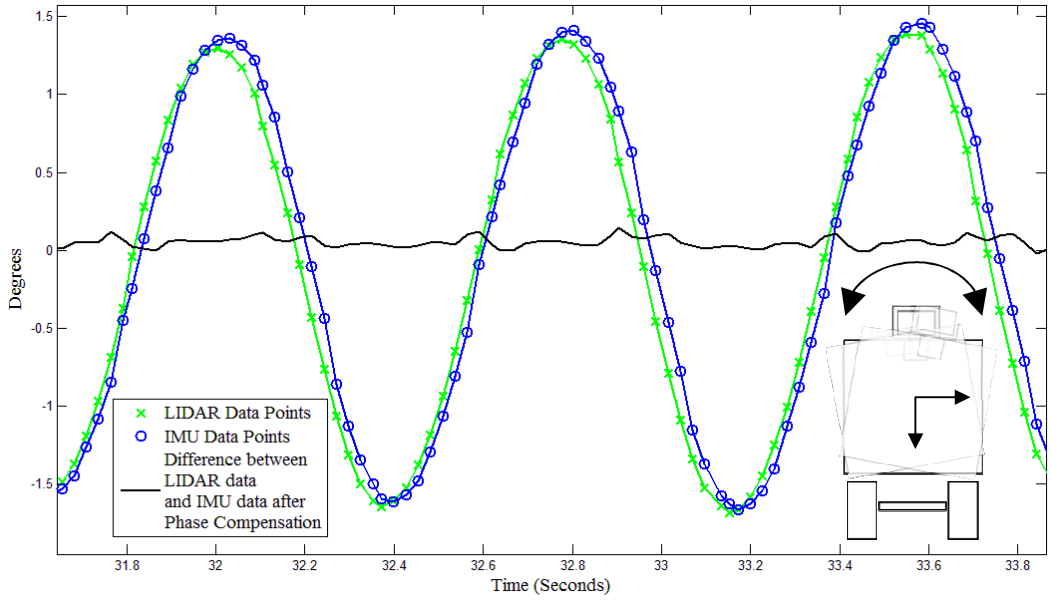

Figure **5-5**: The Figure Illustrates the Ability of the IMU Data to Compensate for the Roll Experienced by the Vehicle.

In order to test the ability of the system to measure slopes in a controlled environment, two boards (40in x 32 in) were placed as shown in Figure 5-7 to represent a controlled, constant-slope surface to serve as a reference for comparison of manual and LIDAR measurements. The slopes of the boards were measured manually by using a digital inclinometer (PRO SMART LEVEL) which has a resolution of 0.1 degrees, and length of 4 feet. This manual measurement process is a low-order survey method commonly used for the measurement of median slope. While higher-order survey methods would facilitate point-to-point correspondence checking, point correspondence is not trivial to obtain from LIDAR data. Fortunately, such validation is not necessary if the LIDAR data is only used for automated median slope measurement.

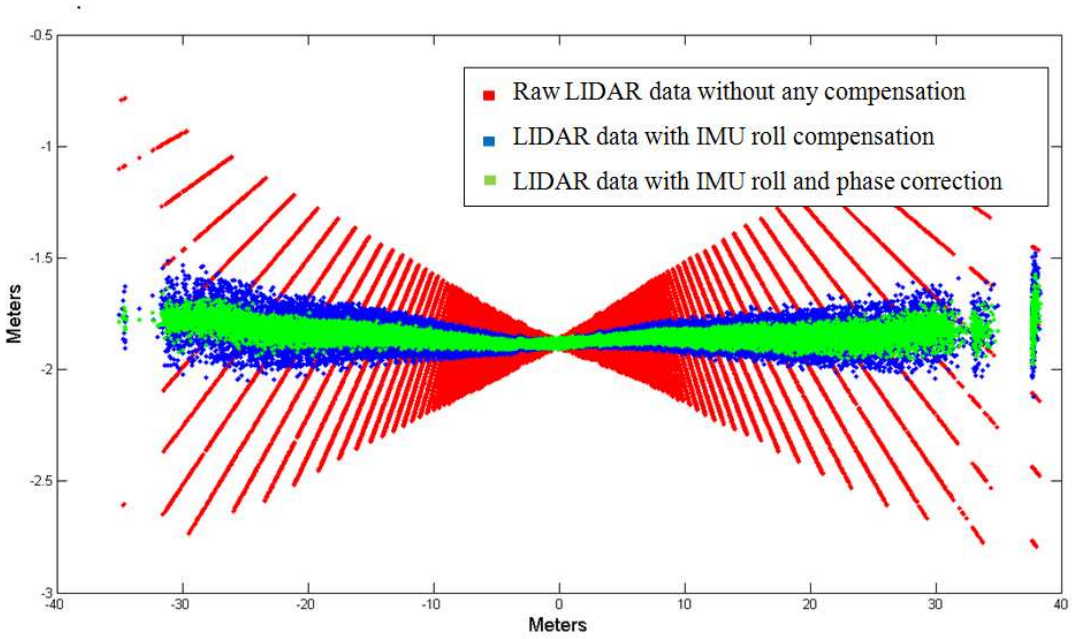

Figure **5-6**: The Figure Shows the Horizontal Surface Under the LIDAR, Under Different Levels of Correction for the Roll.

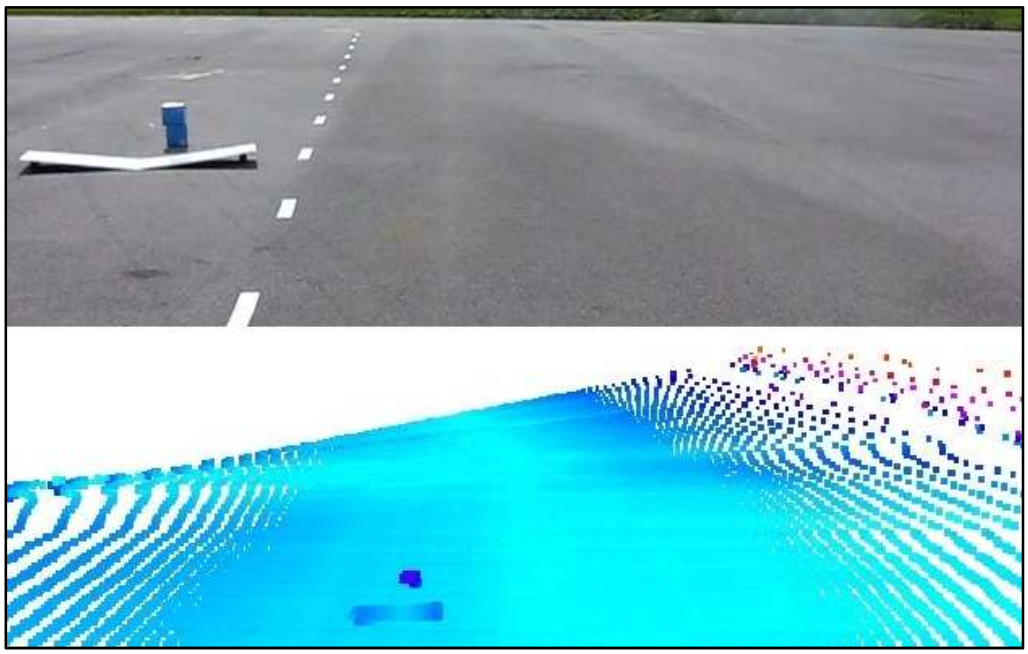

Figure **5-7**: A Photograph and the LIDAR Visualization of the Controlled Sample Surface that has been Used to Compare the LIDAR Slope Measurements with the Manual Slope Measurements.

To test the median scanning system, the LIDAR data was collected while driving past the boards at a spacing of 2-3 meters between the vehicle and the boards. The LIDAR slope data was then compared to manual measurements, as shown in Figure 5-8. Figure 5-8 also compares manual and LIDAR slope measurements for an actual median, a test performed by scanning a stretch of road with the system and manually measuring the median slope at different mile marker locations with the same digital inclinometer. The controlled surface measurements showed an average error of 0.36 degrees, and an average variation of 0.62 degrees was observed in actual medians. Possible reasons for the larger variation observed in actual medians are due to the LIDAR scans hitting vegetation (grass) on the median, while the same grass is impressed by the inclinometer when the slope is measured manually. The errors have been calculated on V-style medians whose center was an average distance of 10 – 20 meters away from the vehicle, and this larger distance versus that of the controlled surfaces can also explain some of the added inaccuracy.

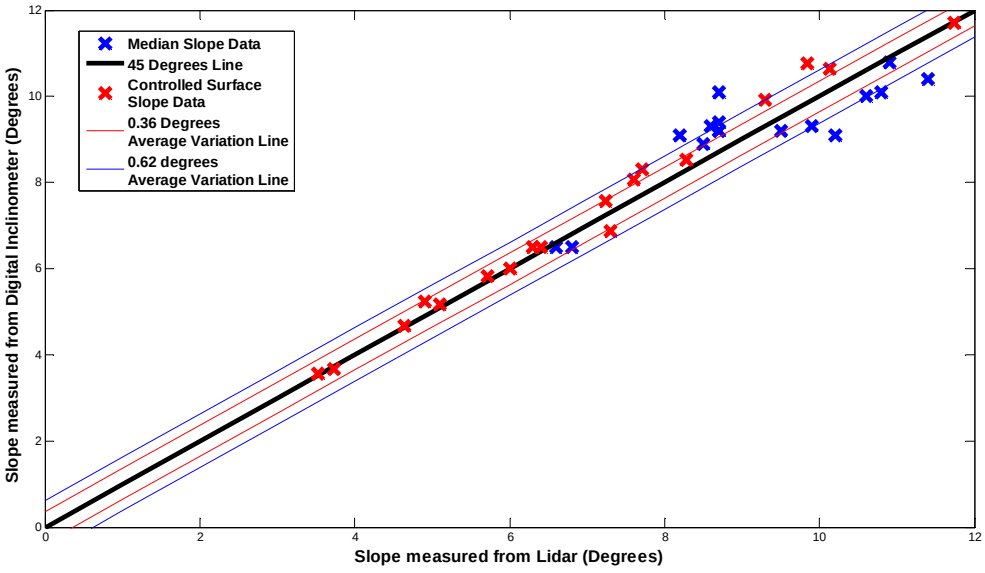

Figure **5-8**: Comparison of Digital Slope Measurements to Manual Slope Measurements.

Figure 5-9 shows results where the repeatability of the system was examined. To test the repeatability of slope measurements, the vehicle was driven at highway speeds on the same section of the road for three different times, and the slopes measured by the system in two of the trials are plotted against the first trial. An average variation of 0.42 degrees in the slope data was observed across all trials. A possible cause for the variation in the data could be that the LIDAR scans are not obtained from exactly the same location in each of the trials, simply due to the motion of the vehicle and uncertainty in position. The system is obviously constrained in its ability to locate a particular scan by the accuracy of the GPS system and the resolution of the scans. It is interesting to note the variation of the slope in the opposing slope is 0.29 degrees, while that on the adjacent slope is 0.56 degrees. The reason for the better opposing slope measurement (despite this slope being farther away) is that the opposing slope is angled towards the LIDAR system and thus has greater number of LIDAR hits when compared to the adjacent slope. Figure 5-10 illustrates these concepts in a clear way as one can observe the shift of the blue scan away from the red and the green scans because of positional inaccuracies of the GPS system. One can also observe that the LIDAR points on the adjacent slope are sparser than the opposing slope.

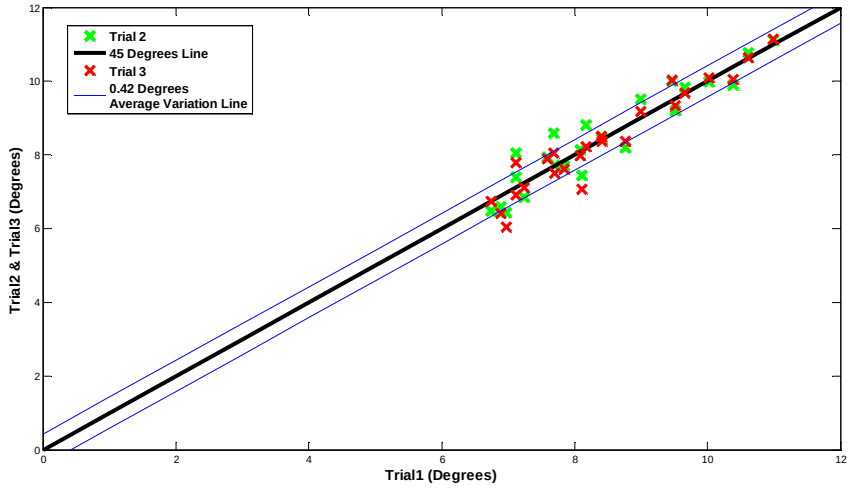

Figure **5-9**: Repeatability Test for LIDAR Slope Measurement at Different Known GPS locations.

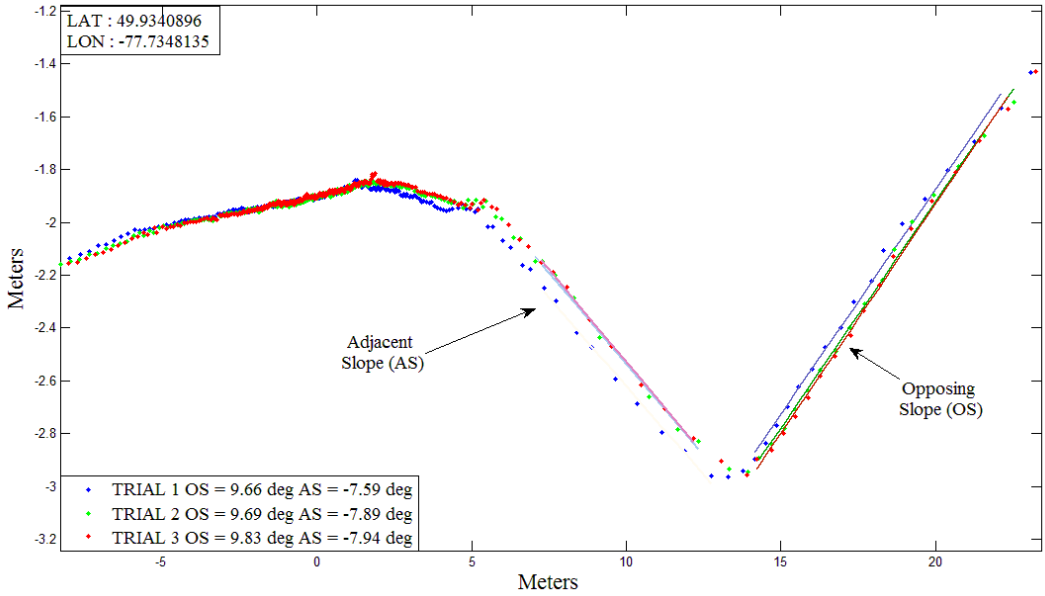

Figure **5-10**: LIDAR Scans Obtained in Three Different Runs At a Known GPS Location.

Variation in median slope

An interesting observation that was made during these measurements was that significant variations of the slope could occur within a single median cross section and between several scans separated by a relatively short distance. This was unexpected given that build plans generally specify a constant slope within the median sections that were scanned. Figure 5-11 illustrates the manual and the LIDAR based slope measurements over small sections of a single median. The manual locations were not surveyed but instead measured using a tape guide from the edge of the road, and hence have lateral position error visible in the graph. Even so, the graph clearly shows that a significant variation in slope is possible depending on where one placed the manual slope meter. Hence, the manual measurement system typically used for measuring the median slope might not be very repeatable. The source of this problem is that a hand

measurement usually records slope data at only a few points and with a relatively short span (1 meter) of the entire slope. This large source of potential error is not evident until one compares to the LIDAR based measurement that uses the entire slope to characterize the median.

The LIDAR scans for road sections have also revealed that there could be a significant variation of the median slope even across a small stretch of a road. This is illustrated in Figure 5-12. By providing a vastly larger amount of data, one can conclude that the LIDAR system might present a clearer picture of the median than conventional manual measurement. Further, the scanning can be done at highway speeds without any obstruction to the on-going traffic.

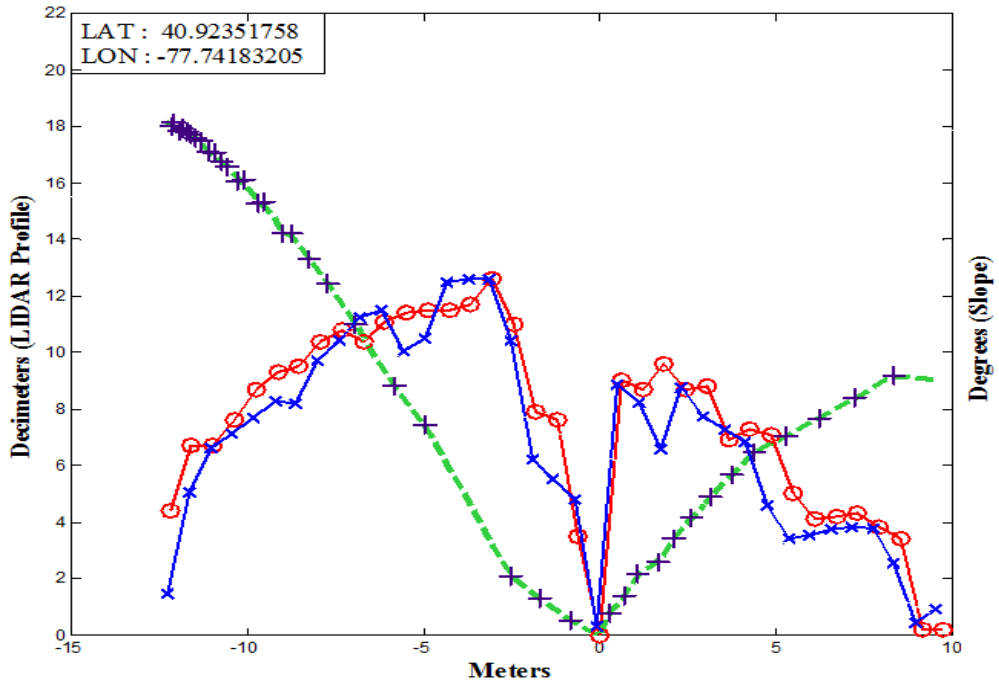

Figure **5-11**: The Variation of a Slope Measured Across a Median Cross Section Measured Manually and with LIDAR.

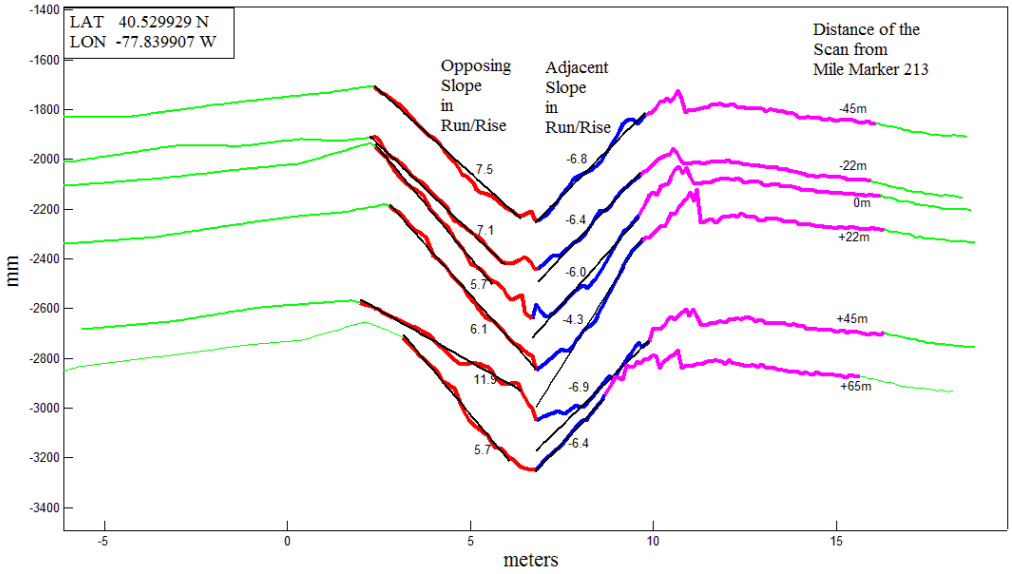

Figure **5-12**: The Plot Illustrates the Variation in the Median Profile at a Distance of +/- 50 Meters Around Milemarker 213 on Route 220S, in Centre County, PA.

Chapter 6

Conclusions and Future Work

This paper presents a LIDAR-based scanning technology for measuring the median slope and compares it to manual measurement techniques. The results indicate accuracies on the order of 0.36 degrees in controlled tests with fixed surfaces, and 0.62 degrees for tests on actual medians. Repeatability was found to be approximately 0.42 degrees for actual median scans. The compensation of vehicle motion was found to quite good, and independent of whether the IMU was mounted on the LIDAR sensor or mounted within the vehicle. Using tests where the vehicle was aggressively rocked back and forth, the error due to compensation for vehicle motion was found to be 0.03 degrees

The system is very cost effective with an approximate expense of $1/mile (2250 data points/mile/minute) to take measurements as compared to manual measurement which has been estimated to cost $100/mile (5 data points/mile/100 minutes), an estimate from past work by researchers at Penn State. This 100 times cost savings agrees with estimates from other researchers. (22)

Thus far, this particular system has been used to scan more than 5000 miles of road (Figure 6-1) in order to extract slopes from divided rural median highways. A number of design modifications have been made in the system over the past year and the final version presented in this paper is a product of this extensive field testing.

A simple way to avoid the inaccuracies in measuring the adjacent slope as compared to the opposing slope would be to measure the slope of the road from either side. Currently work is being done to fuse such data in 3D. Additionally, this also enables one to visualize the terrain

information in a 3D environment, and Figures 6-2 and 6-3 show examples in this regard for road segments with interesting features.

Figure **6-1**: All the Routes that were Covered as a Part of Median Slope Measurement Effort for the NCHRP 22-21 Median Design Project.

The above terrain mapping system can be briefly described as an integrated DAQ system for a Sick LIDAR scanner and a GPS-IMU system that can be used for terrain mapping. An important feature of this system is that the timing issues involved in the data acquisition process have been closely examined and understood and this enables that system to be readily used for other research applications. The system can be used in road surveying applications, which include the measurement of a variety of on-road and off-road parameters. The system can also be used in surveillance applications wherein the difference in the maps of the terrain can be observed over time. It is important to note that the current system contains a fairly common configuration of sensors that have been implemented in other robotic systems (14), (23). Most of these robotics systems have been used to study a variety of robotic research areas such as Simultaneous Localization and Mapping (SLAM), mapping, map based localization, lost robot problems etc in

the indoor environments; the system developed as a part of this research project has been specially designed for the outdoor vehicular environment and will enable progress in solving the above research problems for outdoor applications. A camera can be added to the current system and a registered camera LIDAR data can also lead to research in interesting areas such as building facade models, and localization based on facade models etc.

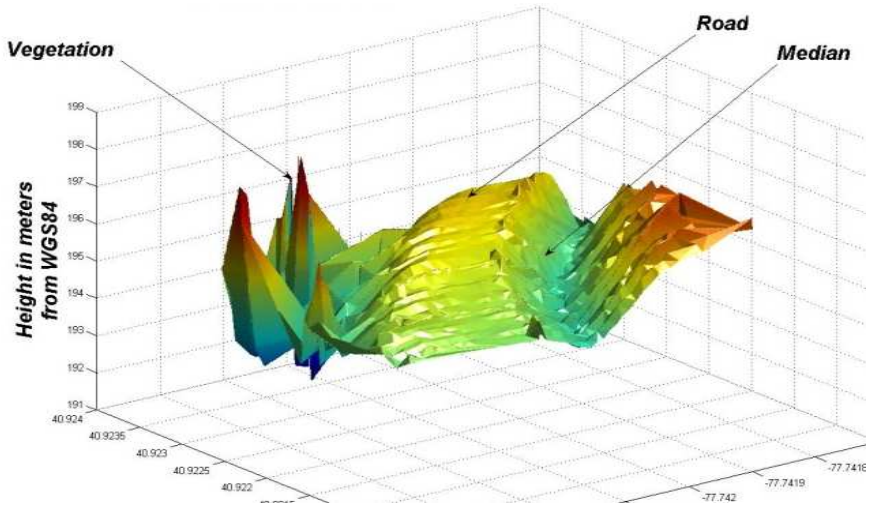

Figure **6-2**: 3-D Visualization of the Road Profile on US 220 S Center County, PA.

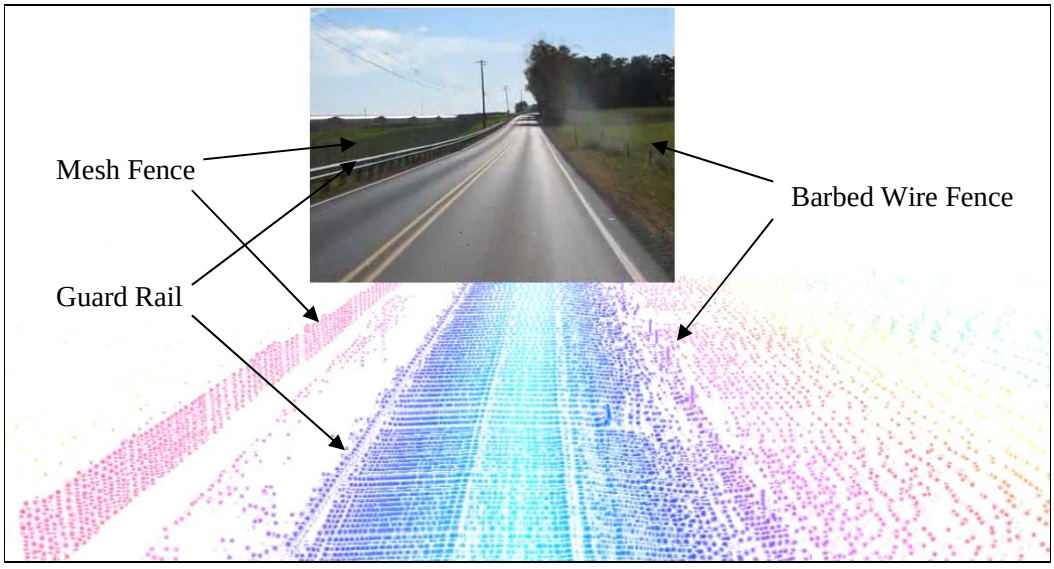

Figure **6-3**: 3-D Visualization of LIDAR Point Cloud, Taken at Foxhill Road West Bound.

BIBLIOGRAPHY

1. **Graham, Jerry.** Median Cross-Section Design for Rural Divided Highways. [Online] Transportation Research Board . http://www.trb.org/TRBNet/ProjectDisplay.asp?ProjectID=694..

2. *Light detection and ranging (LIDAR): An emerging tool for multiple resource inventory.* **Reutebuch SE, Andersen HE, McGaughey RJ.** 2005, Journal of Forestry, pp. 286-292.

3. **Souleyrette R., S. Hallmark, S. Pattnaik, M. O'brien, and D. Veneziano.** *Grade and Cross Slope Estimation From LIDAR-Based Surface Models.* Ames, Iowa : Midwest Transportation Consortium, 2003.

4. **Mekemson James R., and Nicolas Gagarin.** *Method and Apparatus for Pavement Cross Slope Measurement. US 7,142,952 B2* United States, 11 08, 2006.

5. *Innovative Techniques for Automated Analysis of Cross-Slope Data Using Multipurpose Survey Vehicle.* **Mraz Alexander, and Abdenour Nazef.** 2008. Transportation Research Board 87th Annual Meeting.

6. **Walker, Roger S.** *Using Profile Measurements to Locate and Measure Grind and Fill Areas to Improve Ride.* s.l. : Texas Department of Transportation, 2006.

7. **Phares Brent, Halil Ceylan, and Reginald S.** Development of a Device for Analysis of Portland Cement Concrete and Composite Pavements: Phase 1 Feasibility Study. [Online] 10 31, 2007. [Cited: 3 4, 2008.] http://www.ctre.iastate.edu/research/detail.cfm?projectID=1173987080.

8. **Fernando, Emmanuel G., Cindy Estakhri, Gerry Harrison, Eric Becker, and Roger Walker.** *Evaluation of Methods to Measure Ride and Cross Profile on New Base Course and Pavements.* s.l. : Texas Department of Transportation, 2006.

9. **Schmitt Measurement Systems, Inc.** Laser Displacement Sensor for Road Profilometry. [Online] 2003. [Cited: 3 4, 2008.] http://www.acuitylaser.com/pdf/ar600-road-profiling.pdf.

10. *Application of Profile Data to Detect Localized Roughness.* **Fernando E., and C. Bertrand.** 2002, Journal of the Transportation Research Board, pp. 55-61.

11. *Wavelength-Related Ride Equation.* **Walker, R. S., E. Fernando, and C. Bertrand.** 2004, Journal of the Transportation Research Board, pp. 136-144.

12. *Fast Digitization of Large-Scale Hazardous Facilities.* **B. Grinstead, A. Koschan, and M. Abidi.** Gainesville,FL : s.n., 2004. Proc. of 10th Int. Conf. on Robotics & Remote Systems for Hazardous Environments. pp. 521-525.

13. *Developing detailed a priori models of large environments to aid in robotic navigation tasks.* **B. Grinstead, A. Koschan, and M. Abidi.** Orlando,FL : s.n., 2004. Proc. of the SPIE Defense & Security Symposium: Unmanned Ground Vehicle Technology. pp. 561-568 .

14. *Stanley: The robot that won the DARPA grand challenge.* **S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel J. van Nieke.** 9, s.l. : Journal of Field Robotics, 2006, Vol. 23.

15. *Calibration of Inertial and Vision Systems as a Prelude to Multi Sensor Fusion.* **D. Randeniya, Gunaratne Manjriker, Sarkar S., and Nazef Abdenour.** s.l. : Transportation Research Part C, 2006, pp. 255-274.

16. *Hybrid Self Localization for a Mobile Robotic Platform in Indoor and Outdoor Environments.* **B. Grinstead, A. Koschan, and M. Abidi.** San Deigo,CA : s.n., 2005. In Transactions of the American Nuclear Society. pp. 52-53.

17. *Experimental Evaluation of a Pavement Imaging System: Florida Department of Transportation's Multipurpose Survey Vehicle.* **Marz Alexander, Manjriker Gunaratne, Abdenour Nazef, and Bouzid Choubane.** 2006, Journal of Transportation Research Board, pp. 97-106.

18. *Road infrastructure safety assessment.* **Appleton, I. Hannah, J. Noone, Wilkie S.** Canberra,Australia : s.n., 2006. 22nd ARRB conference: Research into Practice.

19. **Turner Blair, and Lee Comport.** *Automatic Collection of Safety Related Road and Roadside Data.* Melbourne, Australia : Arrb Group.

20. *The Player Stage Project: Tools for Multi-Robot and Distributed Sensor Systems.* **Brian P. Gerkey, Richard T. Vaughan, and Andrew Howard.** Coimbra, Portugal : s.n., 2003. In Proc. Of the Intl. Con. on Advanced Robotics. pp. 317-323.

21. **NovAtel.Inc.** *SPAN Technology System Characteristics and Performance.* Alberta, Canada : s.n., 2005.

22. *FDOT Multi-Purpose Survey Vehicle - An Enhancement to Inertial Roadway Profilers. Road Profiler's User Group Meeting.* **Mraz, Alexander and Nazef, Abdenour.** Columbus : http://www.rpug.org/2005/RPUGPresentationsPDF/Alex%20Mraz%20Nour%20Nazef%20FLDOT%20MPV.pdf.

23. **Inc, MobileRobots.** Mobile Robots Research & University Robots, Software & Accessories. *Mobile Robots Inc.* [Online] http://www.activrobots.com/.

Appendix A

Architecture of the Power Circuitry

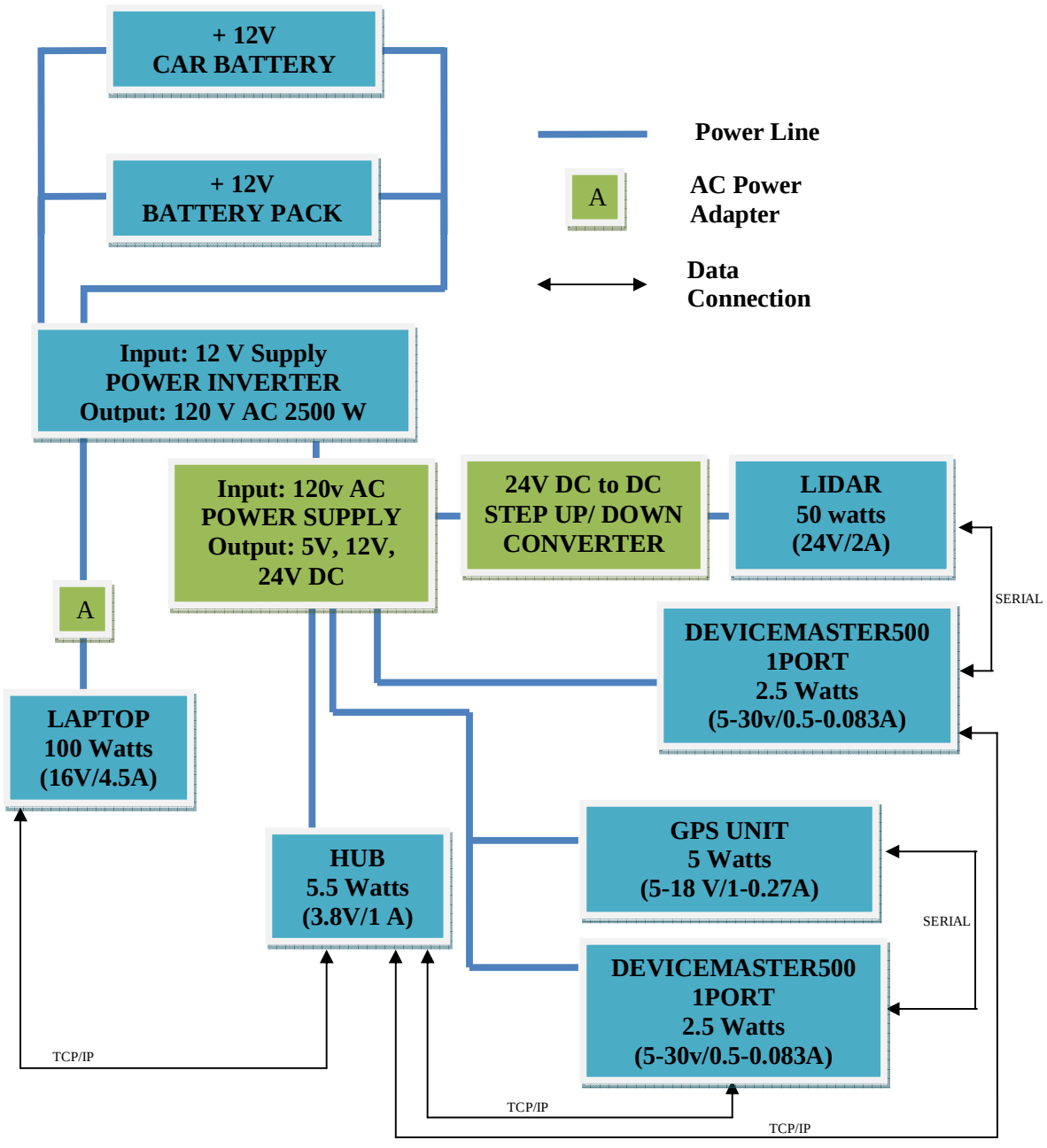

Figure **App A-1**: A Schematic Illustrating the Architecture of the Power System.

A stable and consistent power supply is very important for all the sensors and the computers on board the system. Unstable power supplies can jeopardize the expensive equipment and induce unpredictable behavior in the sensor systems. Figure App A-1 has the schematic illustrating the power architecture used for the scanning system. The key to having a stable power supply is to have a backup battery system which would be able to support the power requirements of the equipment when the vehicle is turned off or in case of a low vehicle battery. Typically, three 12v-20 Amp Hour batteries were linked parallel to the car battery and the setup would usually suffice for 6 hours of scanning. An AIMS 2500 watt modified power inverter was chosen to convert the 12v DC power to AC power. Some of the features of this inverter that were very useful for this application were:

1) Low Battery Voltage Alarm which would set off when the battery voltage reached 10.5V (+/-0.5V) indicated the need to change to a new set of batteries.
2) Low Voltage Automatic Shutdown which would turn off the output power supply when the input dc voltage was at 10V (+/-0.5V). This implies that the power would "blackout" instead of "browning out". This is an important safety feature to protect the sensor equipment.
3) It has high input voltage protection
4) AC output short circuit protection

The AC output of the inverter is used to power the laptop and the power supply which converts the AC voltage back to DC. A Sorenson model power supply which outputs 5V, 10 V and 24 V DC voltages has been chosen for this purpose. A DC to DC step up/Down down converter made by Neuron Technology Limited has been used to ensure a very consistent 24V supply to the LIDAR under all conditions. It must be noted that all the device servers and the sensors have a common ground supplied by the Sorensen Model Power Supply; this reduces the possibility of

equipment damage as all the operating equipment have a common ground. An alternative to this design would be to use a separate AC adaptor for each piece of electronic equipment and isolate the serial interface connections between the equipment by using serial isolators.

Appendix B

Matlab Code for the Data processing Algorithm

The pseudo code for the data processing algorithm is presented below and the Matlab code for the functions is presented thereafter. The Matlab code can be run by running the script "script_iterate_through_data.m".

Pseudo Code

---

script_iterate_through_data.m

```
% The script iterates through mile marker data stored in data files and feeds the data into
% fun_get_avg_slope_4_goodroads to get the average slope, of the adjacent and the opposing
% sides of the median, for all the mile markers.
For (All Mile Marker Data) {
        [Average Slope Data] = fun_get_avg_slope_4_goodroads (Mile Marker Data);
        Store Average Slope Data;
}
```

```
[Average Slope Data] = fun_get_avg_slope_4_goodroads (Mile Marker Data) {
```

```
% The function iterates through the LIDAR scans for a particular mile marker and calculates the
% slope data for each of the scans by calling testa3 and returns the average slope for the mile
% marker.
For (LIDAR scans within a mile marker segment) {
        Slope data = Testa3(LIDAR scan);
}
Return Average Slope Data;
}
```

```
Slope Data = testa3 (LIDAR Scan) {
% Calculates the adjacent slope and the opposing slope for a particular LIDAR scan by calling
% the following functions
Road End Point = tetsa3_fcn_get_adjacent_slope_start_point (LIDAR Scan);
[Adjacent Slope, Adjacent Slope End Point] = tetsa3_fcn_get_optimalslope (LIDAR Scan, Road
End Point);
[Opposing Slope, Opposing Slope End Point] = tetsa3_fcn_get_optimalslope (LIDAR Scan,
Adjacent Slope End Point);
Slope Data = [Adjacent Slope, Opposing Slope];
Return Slope Data;
}
```

```
Road End Point = tetsa3_fcn_get_adjacent_slope_start_point (LIDAR Scan) {
```

```
% The function observes the points that have been obtained from immediately underneath the
% laser to identify the road and subsequently identifies the road edge.
Return Road End Point;
}
```

```
[Slope, Slope End Point] = tetsa3_fcn_get_optimalslope (LIDAR Scan, End Point) {
% The function searches for the optimal slope to represent the profile. The search is performed at
% different points on the median profile. At any given point on the profile the search is performed
% by calling tetsa3_fcn_get_optimalslope_at_a_point_by_changing_slope

For (Perform Search at different points on the profile) {
[Optimal Slope for a Point] = tetsa3_fcn_get_optimalslope_at_a_point_by_changing_slope
(LIDAR Scan, Point On The Scan);
}
Return Optimal Slope Along The Whole Profile;
}
```

```
[Optimal Slope for a Point] = tetsa3_fcn_get_optimalslope_at_a_point_by_changing_slope
(LIDAR Scan, Point On The Scan){
% The function iterates through a reasonable set of lines with different slopes and which pass
% through the given point and finds the line with the optimal fit to the profile
For (Lines With Different Slopes) {
[ Value to characterize the fit of the Line ] = tetsa3_analyze_fit_of_a_line ( LIDAR scan, Line )
```

}

Return Optimal Slope For The Given Point;

}

[Value to characterize the fit of the Line] = tetsa3_analyze_fit_of_a_line (LIDAR scan, Line) {

% The function checks to see if the line satisfies a fixed set of criterion and then computes a value

% that characterizes the goodness of fit of the line with the LIDAR scan.

Return Value to characterize the fit of the Line;

}

Matlab Code

script_iterate_through_data.m

```
% The script processes the entire data collected over the
% span of a day. The data is stored as a set of data segments ,
% where each data segment contains data for a continuous set of
% mile markers on a particular road. The script basically loads
% each data segment and then feeds the data for each mile marker
% in to 'fun_get_avg_slope_4_goodroads' which computes the mean
% and standard deviation for the adjacent and the opposing slopes
% for that particular mile marker. The resulting processed data
% is stored in a variable named 'ans_mat' which is eventually
% stored in a file. The script results in the creation of as many
% processed files as the number of data segments that were
% processed by the script.

num_data_segments=1;

for count00=1:num_data_segments

    %% The static calibration offset
    comp_angle=-1;

    %% create the filename for the data segment that is to be loaded

filename_datasegment=cat(2,'day2_batch_carolina',num2str(count00),'.mat
');

    %% Loads the appropriate data segment
    %% which has the variables 'file_str' and 'indi'
    load (filename_datasegment);

    %% The number of rows in variable indi is stored in
    %% variable size_data. indi is a variable with 2 columns.
    %% the first column contains the mile marker locations and
    %% the second column contains the index of the scan that was
    %% collected at those mile markers
    size_data=size(indi,1);

    %% creates a variable 'ans_mat 'to store the processed
    %% median slope data
    ans_mat=zeros((size_data-1),5);

    %% The below code iterates through mile marker data for a
    %% particular data segment and the resulting processed data
    %% is stored in 'ans_mat'
    for count1=1:size_data-1
```

```
        %% the first column of 'ans_mat' stores the mile marker
        %% number
        ans_mat(count1,1)=indi(count1,1);

        %% the remaining four columns of 'ans_mat' contain the
        %% mean and standard deviation of the adjacent slope and
        %% the mean and the standard deviation of the opposing slope.
        %% file_str is a variable holding the filename of the
        %% file that contains the LIDAR and the GPS data for
        %% this particular data segment.
        [ans_mat(count1,2),ans_mat(count1,3),...
            ans_mat(count1,4),ans_mat(count1,5)]...
            = fun_get_avg_slope_4_goodroads...
            (file_str,indi(count1,2),indi(count1+1,2),...
            indi(count1,1),comp_angle,indi(count1,3));

        %% displays the processed slope data on the screen
        ans_mat(count1,1:5)

    end

    %% store the processed data in a file whose filename is stored in
    %% the variable filename_processed_data
    filename_processed_data=cat(2,'processed',filename_datasegment);
    save(filename_processed_data);

    %% clear all the variables before loading the next data segment
    clear indi size_data ans_mat filename_datasegment ...
        count1 file_str filename_processed_data;

end


clear;
```

fun_get_avg_slope_4_goodroads.m

```
function [adjacent_mean,adjacent_std,opposing_mean,opposing_std]=...
    fun_get_avg_slope_4_goodroads(str,start_index,end_index,...
                                  milemarker,comp_angle,...
                                  cable_barrier_flag)

% the function calculates the mean and the standard deviation
% of the adjacent and the opposing slope for a particular mile
% marker location. The input variables and the output variables
% of the function are described below:


%%%%%%%%%%%%%%%% INPUT VARIABLES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% str - contains the file name of the file holding the LIDAR and
% the GPS data for this mile marker segment

% start_index - contains the starting index for the mile marker
% data in the LIDAR and GPS data

% end_index - contains the ending index for the mile marker data
% in the LIDAR and GPS data

% milemarker - contains the milemarker whose data is being
% processed

% comp_angle - contains the static calibration compensation angle

% cable_barrier_flag - is a flag to indicate if a cable median
% barrier is present in the median for this mile marker

%%%%%%%%%%%%%%%% OUTPUT VARIABLES %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% adjacent_mean - mean of the adjacent slope data

% adjacent_std - standard deviation of the adjacent slope data

% opposing_mean - mean of the opposing slope

% opposing_std - standard deviation of the opposing slope

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% load the LIDAR and GPS data
load(str);

% flag_graphics is a flag that when set to different values
% allows visualization of the data in a Matlab figure
```

```
% flag_grahics =1; Plots data shown in figure 4-4
% flag_graphics =2; Plots data shown in figure 4-1
% flag_graphics =0; is used to avoid plotting the above plots
flag_graphics=0;

% create variables to store the processed adjacent and opposing
% slope data
final_data_adjacent_slope=zeros(1,1000);
final_data_opposing_slope=zeros(1,1000);


% The following code computes the exact index in the
% GPS and LIDAR data at which this mile marker starts
count1=1;
while (final_gps_data(7,count1)<start_index)

    count1=count1+1;

end

% Incase flag_graphics is set to 1,
% The following code creates the handles for
% all the elements in the figure which will be used
% later on.

if (flag_graphics==1)

    h_fig=figure(1);

    MileMarker=milemarker;

    set(gca,'Title',text('String',sprintf...
        ('Profile Of The Median At MileMarker %6.3f Route 40 W NC', ...
        MileMarker),...
        'FontSize',18,'FontWeight','bold','FontAngle','italic'));
    set(gca,'XLabel',text('String','meters','FontSize',...
        16,'FontWeight','normal','FontAngle','italic'));
    set(gca,'YLabel',text('String','mm','FontSize',16, ...
        'FontWeight','normal','FontAngle','italic'));
    set(gca,'XLim',[-15,75],'YLim',[-4000,0]);

    x_c=0;
    y_c=0;
    hold on;
    h_otherdata=plot(x_c,y_c,'g','LineWidth',3);
    h_road=plot(x_c,y_c,'m','LineWidth',4);
    h_adjacent_slope=plot(x_c,y_c,'b','LineWidth',4);
    h_opposing_slope=plot(x_c,y_c,'r','LineWidth',4);
    h_adjacentline=plot(x_c,y_c,'k','LineWidth',2);
    h_opposingline=plot(x_c,y_c,'k','LineWidth',2);
    h_redline=plot(x_c,y_c,'r');
```

```
    set(h_otherdata,'XData',x_c,'YData',y_c);
    set(h_otherdata,'YData',y_c);

    h_leg=legend([h_otherdata;h_road;...
        h_adjacent_slope;h_opposing_slope],...
        'OtherData','Road','adjacent Slope','opposing_slope');

set(h_leg,'FontSize',16,'FontWeight','normal','FontAngle','italic');

    cons=0;
    h_annotate=annotation(h_fig,'textbox',...
        'String',{sprintf('At '' 0'' Lat %8.6f N',cons),...
        sprintf('         Long %8.6f W',cons),...
        sprintf('         adjacent Slope %3.1f deg',cons),...
        sprintf('        opposing Slope %3.1f deg',cons),...
        },...
        'FontSize',16,...
        'FontName','Arial',...
        'FontAngle','italic',...
        'FitHeightToText','off',...
        'Position',[0.1389 0.7468 0.2197 0.1672]);

end

%  increments by 1 when an adjacent slope is identified
count_adjacent_slope=0;

% increments by 1 when an opposing slope is identified
count_opposing_slope=0;

%  increments the distance corresponding to each LIDAR scan
total_dist=0;

% increments the distance for which a slope has been
% identified
dist_with_slopes=0;

% counts the total number of scans for the milemarker
count_total_scans=0;

old_index=count1;
% code to iterate through all the scans for the mile marker
while (final_gps_data(7,count1)<end_index)


    [count1-old_index,end_index-old_index]
    gps_data_count1=final_gps_data(1:6,count1);
    z=final_laser_data(:,count1);
    theta = ((0:0.5:180)*pi/180-0.5*pi/180)...
        - ((gps_data_count1(4)+comp_angle)*pi/180);
    cost=cos(theta');
    sint=sin(theta');
```

```
% calculate the distance between the current scan and the
% previous scan
[distance_btw] = fcn_convert_latlongelev_to_xyz...
    (final_gps_data(1:3,count1),...
    final_gps_data(1:3,count1-1));
dist_value= sqrt((distance_btw(1)*distance_btw(1))+...
    (distance_btw(2)*distance_btw(2))+...
    (distance_btw(3)*distance_btw(3)));

% convert the laser range data into coordinate data
range_y_data=-z.*sint;
range_x_data=z.*cost;


% incase flag_graphics is set to 2, plot the appropriate
% graphics
if (flag_graphics==2)
    plot(range_x_data/1000,range_y_data,'b','Linewidth',3)
    hold on
    theta1=((0:0.5:180)*pi/180-0.5*pi/180);
    cost1=cos(theta1');
    sint1=sin(theta1');
    range_y_data_1=-z.*sint1;
    range_x_data_1=z.*cost1;
    plot(range_x_data_1/1000,range_y_data_1,'g','Linewidth',3)
    axis([-30 30 -3000 0])
    xlabel({'Distance (meters)'},...
        'FontWeight','bold','FontSize',14,...
        'FontName','Arial');
    ylabel({'Distance (mm)'},...
        'FontWeight','bold','FontSize',14,...
        'FontName','Arial');
    legend1 = legend(gca,'show');
    set(legend1,'FontWeight','bold',...
        'FontSize',12,'FontName','Arial');
    hold off;
end

% Eliminate all the range data points which have a range greater
% than 80 meters
z_80=find(z>80000);
range_y_data(z_80)=[];
range_x_data(z_80)=[];
range_y_data(1)=[];
range_x_data(1)=[];

minx=min(range_x_data);
maxx=max(range_x_data);

% Interpolate Y data at regular intervals of X data
x_new=-10*1000:100:maxx;
y_new = interp1(range_x_data,range_y_data,x_new,'linear');
```

```
    % Median filtering eliminates the cable median barrier from the
    % scan
    if (cable_barrier_flag ==1)
        y_new = medfilt2(y_new, [1 15]);
    end

    % Call the testa3 function which computes the median slopes for
    % this particular scan
    [opposingslopeangle,adjacentslopeangle,opposing_slope_coords,...
        adjacent_slope_coords,slope_exists,reply]=tetsa3(x_new,y_new);
    % note the reply output variable can take any of the following
values:
    % 'm' -- when both adjacent and opposing slopes exist
    % 'q' -- when only the adjacent slope exists
    % 'e' -- when only the opposing slope exists
    % 'z' -- when neither exist


    %  The following code plots the data obtained from processing the
scan
    if (flag_graphics==1)

        if (slope_exists==1 && adjacentslopeangle~=0 )

            greenline=[-5000;24000];
            pinkline=[-2500;adjacent_slope_coords(1,1)];
            blueline=[adjacent_slope_coords(1,1);...
                adjacent_slope_coords(end,1)];
            redline=[adjacent_slope_coords(end,1);...
                opposing_slope_coords(end,1)];

            %       h_fig=figure;
            x2=greenline(1):100:greenline(2);
            linex1=pinkline(1):100:pinkline(2);
            linex2=blueline(1):100:blueline(2);
            linex3=redline(1):100:redline(2);
            y2 = interp1(range_x_data,range_y_data,x2,'linear');
            set(h_otherdata,'XData',x2/1000,'YData',y2);
            liney1 =
interp1(range_x_data,range_y_data,linex1,'linear');
            set(h_road,'XData',linex1/1000,'YData',liney1);
            liney2 =
interp1(range_x_data,range_y_data,linex2,'linear');
            set(h_adjacent_slope,'XData',linex2/1000,'YData',liney2);
            liney3 =
interp1(range_x_data,range_y_data,linex3,'linear');
            set(h_opposing_slope,'XData',linex3/1000,'YData',liney3);
            if (adjacentslopeangle~=0)
                set(h_adjacentline,'XData',...
                    adjacent_slope_coords(:,1)/1000,...
                    'YData',adjacent_slope_coords(:,2));
```

```
    end
    if (opposingslopeangle~=0)
        set(h_opposingline,...
            'XData',opposing_slope_coords(:,1)/1000,...
            'YData',opposing_slope_coords(:,2));
    end
    set(h_annotate,'String',...
        {sprintf('At '' 0'' Lat %8.6f N',...
        gps_data_count1(1)),...
        sprintf('        Long %8.6f W',...
        gps_data_count1(2)),...
        sprintf('        adjacent Slope %3.1f deg',...
        adjacentslopeangle),...
        sprintf('        opposing Slope %3.1f deg',...
        opposingslopeangle)    });
    set(h_redline,'XData',0,'YData',0);


else

    set(h_redline,'XData',x_new/1000,'YData',y_new);
    set(h_otherdata,'XData',0,'YData',0);
    set(h_road,'XData',0,'YData',0);
    set(h_adjacent_slope,'XData',0,'YData',0);
    set(h_opposing_slope,'XData',0,'YData',0);

    set(h_annotate,'String',...
        {sprintf('At '' 0'' Lat %8.6f N',...
        gps_data_count1(1)),...
        sprintf('         Long %8.6f W',...
        gps_data_count1(2)),...
        sprintf('         adjacent Slope %3.1f deg',...
        adjacentslopeangle),...
        sprintf('        opposing Slope %3.1f deg',...
        opposingslopeangle)    });

    if (adjacentslopeangle~=0)
        set(h_adjacentline,'XData',...
            adjacent_slope_coords(:,1)/1000,'YData',...
            adjacent_slope_coords(:,2));
    else
        set(h_adjacentline,'XData',0,'YData',0);
    end
    if (opposingslopeangle~=0)
        set(h_opposingline,'XData',...
            opposing_slope_coords(:,1)/1000,'YData',...
            opposing_slope_coords(:,2));
    else
        set(h_opposingline,'XData',0,'YData',0);
    end

end
```

```
end

%% if graphics are ON then pause for user input
 if (flag_graphics~=0)
     user_entry = input('Press enter to continue');
 end


% the total distance for the mile marker is incremented
total_dist=total_dist+dist_value;

% the total scans for the mile marker are incremented
count_total_scans=count_total_scans+1;


switch reply

    % if reply == m ; both the adjacent and the opposing slope
    % are recorded
    case 'm'
        if (adjacentslopeangle~=0 && opposingslopeangle~=0)
            dist_with_slopes=dist_with_slopes+dist_value;
            count_adjacent_slope=count_adjacent_slope+1;
            count_opposing_slope=count_opposing_slope+1;
            final_data_adjacent_slope(1,count_adjacent_slope)...
                =adjacentslopeangle;
            final_data_opposing_slope(1,count_opposing_slope)...
                =opposingslopeangle;
        end

    % if reply == q ; only the adjacent slope is recorded

    case 'q'

        if (adjacentslopeangle~=0)
            dist_with_slopes=dist_with_slopes+dist_value;
            count_adjacent_slope=count_adjacent_slope+1;
            final_data_adjacent_slope(1,count_adjacent_slope)...
                =adjacentslopeangle;
        end

    % if reply == e; only the opposing slope is recorded
    case 'e'

        if (opposingslopeangle~=0)
            dist_with_slopes=dist_with_slopes+dist_value;
            count_opposing_slope=count_opposing_slope+1;
            final_data_opposing_slope(1,count_opposing_slope)...
                =opposingslopeangle;
        end
```

```
    end

    % the count is incremented to process the next scan
    count1=count1+1;



end



final_data_adjacent_slope=final_data_adjacent_slope...
    (1,1:count_adjacent_slope);
final_data_opposing_slope=final_data_opposing_slope...
    (1,1:count_opposing_slope);

% median filtering the slope data smoothes out any sudden spikes in
% the data
final_data_adjacent_slope=medfilt1(final_data_adjacent_slope,3);
final_data_opposing_slope=medfilt1(final_data_opposing_slope,3);


min_qty_thre=30;
% if the number of scans for which a slope has been observed is less
% than min_qty_thre then a slope is not reported for this mile marker
% , else a mean and standard deviation of the slope value is
% reported to the program calling the function

if (size(final_data_adjacent_slope,2)>min_qty_thre)
    adjacent_mean=mean(final_data_adjacent_slope);
    adjacent_std=std(final_data_adjacent_slope);
else
    adjacent_mean=0;
    adjacent_std=0;
end

if (size(final_data_opposing_slope,2)>min_qty_thre)
    opposing_mean=mean(final_data_opposing_slope);
    opposing_std=std(final_data_opposing_slope);
else
    opposing_mean=0;
    opposing_std=0;
end
```

testa3.m

```
function  [opposing_slopeangle,adjacent_slopeangle,...
    opposing__slope_coords,adjacent__slope_coords,...
    flag_continue_processing,reply]...
    =tetsa3(x_new,y_new)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% INPUT VARIABLES

% x_new contains the x coordinates of the median profile
% y_new contains the y coordinates of the median profile

%%%% OUTPUT VARIABLES

% opposing_slopeangle contains the opposing slope's angle
% calculated for the input median profile
% adjacent_slopeangle contains the adjacent slope's angle
% calculated for the input median profile
% opposing__slope_coords contains the x,y coordinates of the line that
% represents the opposing slope
% adjacent__slope_coords contains the x,y coordinates of the line that
% represents the adjacent slope
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% plot_val is used for debugging purposes in order to visualize
% the data
plot_val=0;

% flag_continue_processing is a flag that turns to 0 when the start
% point for the next segment of the median is not identified
flag_continue_processing=1;

% reply is set to default
reply='z';

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Choose the relevant portion of x_newdata by
% restricting the search for the median to
% (651-100)*100 mm= 55 meters
% (refer to line 196 of fcn_get_avg_slope_4_goodroads.m)
index_of_maxwidth_in_x=min(651,size(x_new,2));

% Check to see if the scan has the required number of data points
if (size(x_new,2)>150)

    % the points 95:105 are data points that are expected to lie
    % immediately underneath the laser and are from the road
```

```
    road_x_points=x_new(95:105);
    road_y_points=y_new(95:105);

    % calculating the road line
    coeff_roadline=polyfit(road_x_points,road_y_points,1);

    % The road edge and median data points are computed
    % by calling the tetsa3_fcn_get_adjacent_slope_start_point
    % function
    [road_edge_point,median_x_points,...
        median_x_points_indices ,flag_continue_processing]=...
        tetsa3_fcn_get_adjacent_slope_start_point...
        (coeff_roadline,x_new,y_new,100);
else
    road_edge_point=0;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


if(road_edge_point~=0)

    if plot_val;

        plot(x_new/1000,y_new,'b','Linewidth',3);
        hold on;

        plot(-5:1:2,...
            polyval(coeff_roadline,(-5:1:2)*1000),...
            'k','Linewidth',2);

        plot(median_x_points(road_edge_point)/1000,...
            y_new(median_x_points_indices ...
            (road_edge_point)+100-1),...
            'rX','Linewidth',3);

        axis([-10 20 -3000 -1500]);

        xlabel({'Distance (meters)'},...
            'FontWeight','bold','FontSize',14,...
            'FontName','Arial');

        ylabel({'Distance (mm)'},...
            'FontWeight','bold','FontSize',14,...
            'FontName','Arial');

        legend1 = legend(gca,'show');

        set(legend1,'FontWeight','bold',...
            'FontSize',12,'FontName','Arial');
```

```
    end;

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Given the edge of the road and the coordinates of the median
% the following code feeds this data into the
% tetsa3_fcn_get_optimalslope function in order to
% search for the adjacent slope. Once this slope is identified
% the coordinates of the adjacent slope line are calculated.
adjacent_slopeangle=0;
adjacent__slope_coords=0;
if (road_edge_point~=0 && flag_continue_processing==1)
    slopes_region_x=x_new...
        ((median_x_points...
        (road_edge_point)/100)+100:index_of_maxwidth_in_x);
    slopes_region_y=y_new...
        ((median_x_points...
        (road_edge_point)/100)+100:index_of_maxwidth_in_x);

    [start_val,adjacent_slopeangle,...
        adjacent_slope_end,flag_continue_processing]=...
        tetsa3_fcn_get_optimalslope...
        (slopes_region_x,slopes_region_y,0,25);

    if plot_val ;
        if adjacent_slopeangle(1)~=0; ...

plot(slopes_region_x(start_val:adjacent_slope_end)/1000,...
                polyval(adjacent_slopeangle,...
                slopes_region_x(start_val:adjacent_slope_end))...
                ,'m');
        end ;
    end

    if start_val~=0;
        if adjacent_slope_end~=0 ;
            adjacent__slope_coords=...
                [slopes_region_x(start_val:adjacent_slope_end);...
                polyval(adjacent_slopeangle,...
                slopes_region_x(start_val:adjacent_slope_end))]'...
                ;adjacent_slopeangle=adjacent_slopeangle(1);
        end;
    end;

    if (adjacent_slopeangle(1)<0)
        adjacent_slopeangle=atand(adjacent_slopeangle(1));
        reply='q';
    else
        adjacent_slopeangle=0;
    end
```

```
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Given the edge of the adjacent slope and the coordinates of the
% median pertaining to the opposing slope
% the following code feeds this data into the
% tetsa3_fcn_get_optimalslope function in order to
% search for the opposing slope. Once this slope is identified
% the coordinates of the opposing slope line are calculated.
opposing_slopeangle=0;
opposing__slope_coords=0;
if (road_edge_point~=0 && ...
        adjacent_slopeangle(1)~=0 && ...
        flag_continue_processing==1)

    slopes_region_opposing_x=slopes_region_x(adjacent_slope_end:end);
    slopes_region_opposing_y=slopes_region_y(adjacent_slope_end:end);
    [start_val,opposing_slopeangle,...
        opposing_slope_end,flag_continue_processing]=...
        tetsa3_fcn_get_optimalslope...
        (slopes_region_opposing_x,slopes_region_opposing_y,1,25);

    if plot_val;
        if opposing_slopeangle(1)~=0;
            plot(slopes_region_opposing_x...
                (start_val:opposing_slope_end)/1000 ...
                ,polyval(opposing_slopeangle,...
                slopes_region_opposing_x...
                (start_val:opposing_slope_end)),'m');
        end;
    end;

    if start_val~=0 ;
        if opposing_slope_end~=0
            opposing__slope_coords=...
                [slopes_region_opposing_x...
                (start_val:opposing_slope_end);...
                polyval(opposing_slopeangle,...
                slopes_region_opposing_x...
                (start_val:opposing_slope_end))]';
            opposing_slopeangle=opposing_slopeangle(1);
        end;
    end;

    if (opposing_slopeangle(1)>0 && reply=='q')
        opposing_slopeangle=atand(opposing_slopeangle(1));
        reply='m';
    else
        if (opposing_slopeangle(1)>0)
            reply='e';
        else
            opposing_slopeangle=0;
        end
```

```
    end
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

tetsa3_fcn_get_adjacent_slope_start_point.m

```
function [roadedge_point,median_x_points, ...
            median_x_points_indices ,flag_continue_processing]...
            =tetsa3_fcn_get_adjacent_slope_start_point...
            (coeff_roadline,x_new,y_new,start_pos)

%%%%%%%%%%%%%%% INPUT VARIABLES
% coeff_roadline contains the coefficients for the line
% representing the road.
% x_new and y_new contain the x and y coordinate data
% for the LIDAR scan.
% start_pos contains the point in the coordinate data
% from which the road starts.

%%%%%%%%%%%%%%% OUTPUT VARIABLES
% roadedge_point contains the index of the point in the coordinate
% data where the road ends.
% median_x_points_indices  contains the indices of all the points in
% the coordinate data which are part of the median (and are hence not
% part of the road)
% median_x_points contains the x coordinate data of all the points in
% the coordinate data which are part of the median (and are hence not
% part of the road)
% flag_continue_processing is a flag that indicates if a road edge
% has been found by the below algorithm
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


debug=0;

% thre1 is the threshold that sets the height(mm), from the roadline,
% for the point where the road ends and where the median adjacent slope
% begins .It is the same as thre1 defined in chapter 4
thre1=75;


% if there are thre2 continuous points(thre2*100 mm of distance worth
% of points) above thre1 after a particular point
% then that point is considered to be the start of the median
% downslope. It is the same as thre2 defined in chapter 4
thre2=30;


% thre3 is the second layer of threshold included to prevent any
% effects of noise that may occur if only thre1 was used.
% It is the same as thre3 defined in chapter 4
thre3=25;

%  thre_roadedge_maxdistance sets the threshold for the case when a
road
% edge cannot be identified by the above thresholding system within a
```

```
% reasonable distance. The 'reasonable' distance is set by
% thre_roadedge_maxdistance = 14 meters
thre_roadedge_maxdistance=14000;


% Choose the relevant portion of x_newdata by
% restricting the search, for the road edge,
% to (351-100)*100 mm= 25 meters
% (refer to line 196 of fcn_get_avg_slope_4_goodroads.m)
thre_25meters_xnew=min(351,size(x_new,2));


% threshold to check for obstruction objects.
% A scan is considered to have an obstruction
% if it has a point whose height is
% above 250mm from the road line
thre_obstruction=-250;


% no_roadedge_flag is set to 1 if any of the above thresholds
% has been crossed. Once the no_median_flag is set, the program
% skips over the remaining steps and reports to the program
% calling the function that a road edge could not be found
no_roadedge_flag=0;


% Calculates the distance of the points on the scan
% from the road line
hits_above_line=(polyval(coeff_roadline,...
    x_new(start_pos:thre_25meters_xnew))...
    -y_new(start_pos:thre_25meters_xnew));
% ax+by+c>0 if the point(x,y) is to the that side
% of the line which does not contain the origin


% implementing thre1
% median_x_points_indices holds all the points which cross thre1
median_x_points_indices =find(hits_above_line>thre1);
median_x_points=x_new(median_x_points_indices +start_pos-1);
size_median_x_points_indices =size(median_x_points_indices ,2);



% implementing thre2 & thre3
% as described in chapter 4
roadedge_point=0;
found_downslope_flag=0;
if (no_roadedge_flag==0)

    roadedge_point=0;
    while(found_downslope_flag==0&&...
            roadedge_point+thre2<size_median_x_points_indices  )
```

```
        roadedge_point=roadedge_point+1;
        if isequal(median_x_points_indices ...
                (roadedge_point+1:roadedge_point+thre2),...
                median_x_points_indices (roadedge_point)...
                +1:median_x_points_indices (roadedge_point)+thre2)
            found_downslope_flag=1;
            if debug==1;
                fprintf(1,'continuous set of thre2 points found !!\n');
            end;
        end
    end
    if (roadedge_point>0)
        no_roadedge_flag=0;
        while (hits_above_line...
                (median_x_points_indices (roadedge_point))>thre3)
            median_x_points_indices (roadedge_point)...
                =median_x_points_indices (roadedge_point)-1;
            median_x_points(roadedge_point)=...
                x_new(median_x_points_indices ...
                (roadedge_point)+start_pos-1);
        end
    else
        no_roadedge_flag=1;
        if debug==1;
            fprintf(1,'no continuous set of thre2 points found !!!\n');
        end;
    end

end



% Step 2 : Implements thre_roadedge_maxdistance
if (no_roadedge_flag==0)
    if (   ...
       (x_new(median_x_points_indices (roadedge_point)...
       +start_pos-1)-x_new(start_pos)) ...
        > thre_roadedge_maxdistance ...
        )
        no_roadedge_flag=1;
        if debug==1;
            fprintf(1,'The Segment length has been exceeded \n');
        end;
    end

end



% Step 3: Checks for any obstructions and implements the
% thre_obstruction
```

```
if (no_roadedge_flag==0)

    obstruction_points=find(hits_above_line...
        (1:(median_x_points_indices
(roadedge_point)))<thre_obstruction);
    x_obstruction_points=x_new(obstruction_points+100-1);

    if (~isempty(obstruction_points))
        no_roadedge_flag=1;
        if debug==1;
            fprintf(1,'There is thre_obstruction obstruction \n');
        end;
    end

end


% notify the calling program that a road edge was not detected
if (no_roadedge_flag==0)

     flag_continue_processing=1;

else

    flag_continue_processing=0;

end
```

tetsa3_fcn_get_optimalslope.m

```
function [final_slope_start,final_slopeangle,...
    final_slope_end,final_slope_exists]=...
    tetsa3_fcn_get_optimalslope...
    (slopes_region_x1,slopes_region_y1,direc,no_of_search_points)


%%%%%%%%%%%%%INPUT VARIABLES

% slopes_region_x1 and slopes_region_y1 contains the x and y coordinate
% data of the adjacent or the opposing slope.
% direc indicates if the search for the slope data pertains to an
% adjacent slope or the opposing slope.
% no_of_search_points sets the maximum number of discrete
% search points that will be processed by the below algo.
no_of_search_points=25;

%%%%%%%%%%%%% OUTPUT VARIABLES
% final_slope_start contains the index of the start point of the
% adjacent or the opposing slope.
% final_slope_end contains the index of the end point of the
% adjacent or the opposing slope.
% final_slope_exists is a flag which indicates if the
% algorithm was able to identify a slope.
% final_slopeangle contains the angle of the line that
% best represents the adjacent or the opposing slope of this
% median.


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% The below set of 'store_' variables store the results obtained
% from searching for the optimal slope, by calling
% tetsa3_fcn_get_optimalslope_at_a_point_by_changing_slope'
% at each of the search points

% store_slopestart stores the index of the start point of the
% optimal slope line at all search points.
store_slopestart=zeros(no_of_search_points,1);

% store_search_points stores the index of all the search points.
store_search_points=zeros(no_of_search_points,1);

% store_slopeangle stores the slope of the optimal slope line
% at all search points.
store_slopeangle=zeros(no_of_search_points,1,2);

% store_slope_end stores the index of the end point of the
% optimal slope line at all search points.
store_slope_end=zeros(no_of_search_points,1);
```

```
% store_optval stores the value that characterizes the optimal
% fit at all the search points.
store_optval=-25*ones(no_of_search_points,1);

% store_slope_exists stores a flag that indicates if a slope
% has been found at each of the search points
store_slope_exists=zeros(no_of_search_points,1);
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% plot(slopes_region_x1/1000,slopes_region_y1+200,'g','Linewidth',3);
% hold on;
% plot(slopes_region_x1/1000,slopes_region_y1,'g','Linewidth',3);


count1=0;
flag_endcount=0;
count_noslope=0;
while(flag_endcount==0)

    % The below code sets the width of the stretch of median profile
    % that will be used to identify a search point
    count1=count1+1;
    if (direc==0)
        width=6;
    else
        width=12;
    end


    % The statistical median of all the heights of the points within
    % that region is chosen and is fed as the search point to
    % tetsa3_fcn_get_optimalslope_at_a_point_by_changing_slope
    sort_y=slopes_region_y1((width*(count1-1)+1) ...
        :min((width*(count1)+1),size(slopes_region_y1,2)));
    [sort_hold,IX]=sort(sort_y);
    if (round(size(IX,2)/2)>0)
        pos1=IX(round(size(IX,2)/2));
    end


    % the following code feeds the search point into the function
    % tetsa3_fcn_get_optimalslope_at_a_point_by_changing_slope
    % and keeps a tab on them to see if lines representing the slope
    % are being found by the search function, in order to
    % stop the search process if no lines are being found.
    if (size(slopes_region_y1,2)>(width*(count1-1)+1)+pos1+10)

        [slopestart,slopeangle,slope_end,opt_val,slope_exists]...

=tetsa3_fcn_get_optimalslope_at_a_point_by_changing_slope...
            (slopes_region_x1,...
            slopes_region_y1,...
            (width*(count1-1)+1)+pos1,...
```

```
            direc);
        % The iteration stops if three consecutive search points
        % donot find any line that fit the profile and pass through
them
        if (slope_exists==0)
            count_noslope=count_noslope+1;
            if (count_noslope==3)
                flag_endcount=1;
            end
        else
            count_noslope=0;
        end

    else
        slopestart=0;
        slopeangle=0;
        slope_end=0;
        opt_val=0;
        pos1=0;
        slope_exists=0;
    end

    if (count1>no_of_search_points)
        flag_endcount=1;
    end

    % store the optimal parameters at each point
    store_slopestart(count1,:)=slopestart;
    store_slopeangle(count1,:,:)=slopeangle;
    store_slope_end(count1,:)=slope_end;
    store_optval(count1,:)=opt_val;
    store_slope_exists(count1,1)=slope_exists;
    store_search_points(count1,1)= (width*(count1-1)+1)+pos1;

end

% the following code calculates the optimal line representing
% the profile of the median and stores the parameters in
% the set of 'final_' variables which transfer this data
% back to the calling program
[max_val,max_pos]=max(store_optval);

% plot(slopes_region_x1(store_search_points...
%     (max_pos):store_search_points(max_pos)+10)/1000,...
%     polyval(squeeze(store_slopeangle(max_pos,:,:))...
%     ,slopes_region_x1(store_search_points(max_pos):...
%     store_search_points(max_pos)+10)),...
%     'r','Linewidth',2);
% plot(slopes_region_x1(store_slopestart(max_pos,:)...
%     :store_slope_end(max_pos,:))/1000,...
%     polyval(squeeze(store_slopeangle(max_pos,:,:))...
%     ,slopes_region_x1(store_slopestart(max_pos,:)...
%     :store_slope_end(max_pos,:)))+200,...
%     'r','Linewidth',2)
```

```
% plot(slopes_region_x1(store_slope_end(max_pos,:))...
%     /1000,slopes_region_y1(store_slope_end(max_pos,:))+200 ...
%     ,'bx','Linewidth',3);
% axis([1 18 -3000 -1200])
% xlabel({'Distance (meters)'},'FontWeight','bold','FontSize',14,...
%     'FontName','Arial');
% ylabel({'Distance (mm)'},'FontWeight','bold','FontSize',14,...
%     'FontName','Arial');


final_slope_start=store_slopestart(max_pos,:);
final_slopeangle=squeeze(store_slopeangle(max_pos,:,:));
final_slope_end=store_slope_end(max_pos);
final_slope_exists=store_slope_exists(max_pos);
```

tetsa3_fcn_get_optimalslope_at_a_point_by_changing_slope.m

```
function [slope_start,slope_angle,...
            slope_end,opt_val,flag_slope_exists]=...
            tetsa3_fcn_get_optimalslope_at_a_point_by_changing_slope...
            (slopes_region_x,slopes_region_y,start_pos,direc)


%%%%%%%%%%%%%% INPUT VARIABLES
% slopes_region_x, slopes_region_y contain the x and y coordinate
% data of the median slope
% start_pos contains the point about which the slope has to be changed
% direc has a value of 1 for opposing slope and 0 for adjacent slope

%%%%%%%%%%%%%% OUTPUT VARIABLES
% slope_start contains the index, in the coordinate data, where
% the slope starts.
% slope_angle contains the angle of the slope
% slope_end  contains the index, in the coordinate data, where
% the slope ends.
% opt_val contains the value that characterizes the goodness of
% the slope line fit to the median profile.
% flag_slope_exists is a flag that indicates to the calling program
% whether a viable line has been found by the search algorithm.
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


flag_slope_exists=1;



% The following check is to make sure that a median of sufficient
% length (2m) exists beyond the start_pos to perform the below tests
if (size(slopes_region_x,2)-start_pos>20)


    % Calculate the slope at start_pos by considering various segments
    % of the median, that include the start_pos, and choosing the
    % statistical median of all the slopes that have been calculated
    slopes(1:5)=0;
    j=0;
    for i=5:3:19
        j=j+1;
        p1=polyfit(slopes_region_x(start_pos:start_pos+i)...
            ,slopes_region_y(start_pos:start_pos+i),1);
        slopes(j)=p1(1);
    end
    slopes=sort(slopes);
    start_slope=atand(slopes(3));
```

```
% Once the slope of the line is calculated at the start_pos by
% the above code. This slope is varied to achieve an optimal
% fit by the following code
%
opt_val=0;
opt_slope=0;
opt_end=0;
opt_start_val=0;
slope_ok=0;
for i=start_slope-12:0.5:start_slope+12

    if ( (direc==1&&i>0) || (direc==0&&i<0) )

        cons= slopes_region_y(start_pos)...
            -(slopes_region_x(start_pos)*tand(i));
        % y-mx=c calculate constant c

        [curr_val,start_val,end_val,slope_ok]=...
            tetsa3_analyze_fit_of_a_line([tand(i),cons],...
            slopes_region_x,slopes_region_y,direc,start_pos);
        % calculate the goodness of fit for line of each slope

        if (curr_val>opt_val && slope_ok==1 )
        % store the characteristics of the line which has the
        % most optimal fit

            opt_slope=[tand(i),cons];
            opt_end=end_val;
            opt_val=curr_val;
            opt_start_val=start_val;

        end
    end

end



% finally optimal fit parameters for a line passing through
% start_pos and fitting the median profile are transmitted back
% to the calling program
if (opt_end>1 && opt_start_val<opt_end)
    slope_start=opt_start_val;
    slope_angle=opt_slope;
    slope_end=opt_end;
else
    flag_slope_exists=0;
    slope_end=0;
    slope_angle=0;
    slope_start=0;
```

```
    end


else

    flag_slope_exists=0;
    slope_start=0;
    slope_end=0;
    slope_angle=0;
    opt_val=0;

end
```

tetsa3_analyze_fit_of_a_line.m

```
function [value,start_val,end_val,slope_ok]=...
             tetsa3_analyze_fit_of_a_line...
            (coeff_slopeline,x_new,y_new,direc,start_pos)

%%%%%%%%%% INPUT VARIABLES

% coeff_slopeline -- contains the coefficients of the line
% representing the slope.
% x_new contains the x coordinates of the median profile.
% y_new contains the y coordinates of the median profile.
% direc is a flag which indicates whether the current data
% pertains to the adjacent slope or the opposing slope.
% start_pos contains the index of the starting coordinate
% in x_new and y_new after which we expect the adjacent or
% the opposing slope to be present
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%% OUTPUT VARIABLES

% value contains the goodness of fit parameter that characterizes
% the fit of the line to the median profile.
% start_val contains the index of the starting point of the line
% ,representing the adjacent or the opposing slope, in the coordinate
% data.
% end_val contains the index of the ending point of the line
% ,representing the adjacent or the opposing slope, in the coordinate
% data .
% slope_ok is a flag that indicates if a line representing the slope
% has been identified
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% Use debug to indicate the results of each processing step
debug=0;


% thre1 is the threshold that sets the height(mm) ,from the adjacent
% line or the opposing line, to find the PSD (Point of Significant
% Departure) for the corresponding line
% It is the same as thre1 defined in chapter 4
thre1=100;


% if there are thre2 continuous points(thre2*100 mm of distance worth
% of points) above thre1 after a particular point
% then that point is considered to be the PSD of the corresponding
% adjacent line or the opposing line
% It is the same as thre2 defined in chapter 4
thre2=40;
```

```
% Step 2 Segment length
% The thresholds check to see if the length of the line representing
% the slope is within reasonable limits. (1.5 meters to 20 meters)
thre3_max_segmentlength=20000;
thre3_min_segmentlength=1500;

% Choose the relevant portion of x_newdata by
% restricting the search, for the PSD of the opposing or
% adjacent slope,
% to (351-100)*100 mm= 25 meters
% (refer to line 196 of fcn_get_avg_slope_4_goodroads.m)
thre_25meters_xnew=min(351,size(x_new,2));

% threshold to check for obstructions.
% A scan is considered to have an obstruction
% if it has a point whose height is
% above 250mm from the slope line.
thre_obstruction=250;

% Step 4 angle limitation
thre_min_slopeangle=4;
thre_max_slopeangle=18;


if (direc==0)
    slope_str='AdjSlope';
    pow_startpos=0;
else
    slope_str='OppSlope';
    pow_startpos=0.15;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


% no_median_flag is set to 1 if any of the above thresholds
% have been violated. Once the no_median_flag is set, the program
% skips over the remaining steps and reports to the program
% calling the function that a road edge could not be found.
no_median_flag=0;
hits_above_line=...
    abs(polyval(coeff_slopeline,x_new(start_pos:thre_25meters_xnew))...
    -y_new(start_pos:thre_25meters_xnew));
% ax+by+c>0 if the point(x,y) is to the that
% side of the line which doesnot contain the origin

% implementing thre1
% good_thre1_points holds all the points which cross thre1
good_thre1_points=find(hits_above_line>thre1);
```

```
x_good_thre1_points=x_new(good_thre1_points+start_pos-1);
size_good_thre1_points=size(good_thre1_points,2);

% implementing thre2
c_i=0;
found_downslope_flag=0;
if (no_median_flag==0)

    c_i=0;
    while(found_downslope_flag==0&& c_i+thre2<size_good_thre1_points )
        c_i=c_i+1;
        if isequal(good_thre1_points(c_i+1:c_i+thre2),...
                good_thre1_points(c_i)+1:good_thre1_points(c_i)+thre2)
            found_downslope_flag=1;
            if debug==1;
                fprintf(1,...
             '%s continuous set of thre2 points found !\n',slope_str);
            end;
        end
    end
    if (c_i>0)
        no_median_flag=0;
    else
        no_median_flag=1;
        if debug==1;
            fprintf(1,...
            '%s no continuous set of thre2 points found
!!!\n',slope_str);
        end;
    end

end



%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 2 : check for segment length
if (no_median_flag==0)

    xdist=(x_new(good_thre1_points(c_i)+start_pos-1)-x_new(start_pos));
    ydist=(y_new(good_thre1_points(c_i)+start_pos-1)-y_new(start_pos));

    xydist=sqrt((xdist*xdist)+(ydist*ydist));
    if ( xydist > thre3_max_segmentlength )
        no_median_flag=1;
        if debug==1;
            fprintf(1,'%s The Segment length has been exceeded \n'...
                ,slope_str);
        end;
    end

end
```

```
if (no_median_flag==0)
    if (   xydist < thre3_min_segmentlength )
        no_median_flag=1;
        if debug==1;
            fprintf(1,...
                '%s The Segment length is too short \n'...
                ,slope_str);
        end;
    end

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 3: Check for Obstruction
if (no_median_flag==0)

    bad_th5_points=find(hits_above_line...
        (1:(good_thre1_points(c_i)))>thre_obstruction);

    if (~isempty(bad_th5_points))
        no_median_flag=1;
        if debug==1;
            fprintf(1,...
            '%s There is thre_obstruction obstruction \n',slope_str);
        end;
    end

end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 4: Check for Angle constraints

if (no_median_flag==0)
    if (direc==0 && atand(coeff_slopeline(1))>-thre_min_slopeangle )
        no_median_flag=1;
        if debug==1;
            fprintf(1,'%s adjslopeangle too small\n',slope_str);
        end;
    end
end
if (no_median_flag==0)
    if (direc==0 && atand(coeff_slopeline(1))<-thre_max_slopeangle )
        no_median_flag=1;
        if debug==1;
            fprintf(1,'%s adjslopeangle too large\n',slope_str);
        end;
    end
end
if (no_median_flag==0)
    if (direc==1 && atand(coeff_slopeline(1))<thre_min_slopeangle )
        no_median_flag=1;
```

```
        if debug==1;
            fprintf(1,'%s Oppslopeangle too small\n',slope_str);
        end;
    end
end
if (no_median_flag==0)
    if (direc==1 && atand(coeff_slopeline(1))>thre_max_slopeangle )
        no_median_flag=1;
        if debug==1;
            fprintf(1,'%s Oppslopeangle too large\n',slope_str);
        end;
    end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 5 : In case the current check is being conducted on the adj
% slope, then a check needs to be done to see if an opp slope exists

if (no_median_flag==0)
    if (direc==1 && start_pos>35 )
        no_median_flag=1;
        if debug==1;
            fprintf(1,'%s Oppslopeangle has gone too far
\n',slope_str);
        end;
    end
end

if (direc==1)
    direc=direc;
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Step 6 : Calculation of the value that characterizes the fit of the
% line to the adjacent or the opposing slope

value=0;
end_val=0;
start_val=0;

if (no_median_flag==0)

    debug1=0;
    slope_ok=1;

    a=x_new(1:(good_thre1_points(c_i)+start_pos-1));
    b=y_new(1:(good_thre1_points(c_i)+start_pos-1));
    z=abs(polyval(coeff_slopeline,a)-b);

    % all the points on the median that lie within a distance of
    % 25 mm from the line are used to estimate the goodness of fit
```

```
    % for the line
    numbelow_thre=find(z<25);
    if debug1;

plot(a(numbelow_thre2)/1000,polyval(p1,a(numbelow_thre2)),'kx');
    end
    if debug1;plot(a/1000,polyval(p1,a),'k');end

    if (~isempty(numbelow_thre) )

        % The start value estimates the starting point from where the
        % the line represents the median based on the above criterion
        start_val=min(numbelow_thre);

        % The value characterizes the goodness of the fit of the line
        % to the median profile
        value=size(numbelow_thre,2)/(start_val^pow_startpos);

        % The end value estimates the end point till where the
        % the line represents the median based on the above criterion
        end_val=max(numbelow_thre);

        % This check is performed to make sure that the start and end
        % points are away from each other by a distance of more than
2.5 meters
        if ((end_val-start_val)<25)
            no_median_flag=1;
            slope_ok=0;
        end
    else
        slope_ok=0;
    end

else

    slope_ok=0;

end
```

fcn_convert_latlongelev_to_xyz.m

```
function [location,factor_conv] =
fcn_convert_latlongelev_to_xyz(latlongelev, origin)

% Adapted from script_lla2lp_track.m by Ryan Martini

% fcn_convert_latlongelev_to_xyz
%   - converts latitude and longitude to local x, y, z
% Input 1: a 3x1 vector of [lat long elev],
%   representing the measurement to convert
% Input 2: a 3x1 vector of [lat long elev],
%   representing the origin
% Output 1: a 3x1 vector of consisting of [x, y, z],
%   representing converted position in meters

format long


origin(3)=latlongelev(3);

% latitude of measurement (degrees)
lat = latlongelev(1);
% longitude of measurement (degrees)
long = latlongelev(2);
% height of measurement (meters)
% above reference ellipsoid (WGS84)
elev = latlongelev(3);
% latitude of origin (degrees)
lat0 = origin(1);
% longitude of origin (degrees)
long0 = origin(2);
% height of origin (meters) above
% reference ellipsoid (WGS84)
elev0 = origin(3);

% intermediate calculation
% (prime vertical radius of curvature)

aa = 6378137;
ee = 8.1819190842622e-2;

% intermediate calculation
% (prime vertical radius of curvature)
% arc_distance = 6372795.477598;  % units are meters
arc_distance = aa ./ sqrt(1 - ee^2 .* sin((lat).*pi/180).^2);

arc_distance_long = arc_distance .* cos(lat*pi/180);


factor_conv(1)=(arc_distance+elev)*pi/180;
```

```
factor_conv(2)=(arc_distance_long+elev)*pi/180;
factor_conv(3)=1;

x = (lat-lat0)*factor_conv(1);
y = (long-long0)*factor_conv(2);
z = -(elev-elev0)*factor_conv(3);
location = [x y z];
```