COMPARING THE PERFORMANCE OF STRUCTURED LIGHT DEPTH SENSORS AND TRADITIONAL TIME-OF-FLIGHT DEPTH SENSORS FOR USE IN A LUNAR MINING ENVIRONMENT

by

CHRISTOPHER HALL

KENNETH RICKS, COMMITTEE CHAIR
JEFF JACKSON
MONICA ANDERSON

A THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science
in the Department of Electrical and Computer Engineering
in the Graduate School of
The University of Alabama

TUSCALOOSA, ALABAMA

2014

ABSTRACT

Autonomous robots are seen as a necessary component for long term manned missions to the Moon. The robots are necessary to excavate lunar soil for processing for in situ resource utilization. The lunar environment poses several challenges to autonomous robotic navigation and the choice of sensor technologies is more restricted than on Earth. Without GPS and ultrasonic technologies, localization and obstacle detection are often performed using data from a laser-based scanner. Laser scanners have been used in robotics on Earth for many years to provide the distances to surrounding objects. Newer sensors, based upon the use of structured light, can provide range data faster and at a lower cost than traditional laser scanners. The purpose of this project is to evaluate a structured light depth sensor, the Microsoft Kinect for Xbox 360, and a traditional multi-echo laser scanner, the Hokuyo UTM-30LX-EW, to determine if they are suitable for autonomous robotic navigation tasks in a lunar mining application. Experimental results are presented that indicate that IR saturation will prevent the Kinect from producing usable distance data. While IR does not affect the lidar, suspended dust in the environment adversely affect both sensors, differently. In dusty environments, the Kinect performs better at shorter distances while the lidar performs better at longer distances to target. The results indicate that a hybrid system utilizing a Kinect for short range obstacle detection and avoidance combined with a lidar for long range landmark identification and localization could serve as a solution in dusty lunar mining environments protected from excessive IR saturation.

DEDICATION

To my parents, who have always believed that I could do anything I put my mind to; and whose unconditional support and encouragement inspires me to believe as well.

LIST OF ABBREVIATIONS AND SYMBOLS

2D	Two-Dimensional
3D	Three-Dimensional
COM	Communications
FOV	Field Of View
GUI	Graphical User Interface
IR	Infrared
LED	Light Emitting Diode
LRF	Laser Range Finder
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
NASA	National Aeronautics & Space Administration
PCL	Point Cloud Library
RANSAC	Random Sample Consensus
ToF	Time-of-Flight
USB	Universal Serial Bus
VDC	Volts Direct Current

ACKNOWLEDGMENTS

I would like to take this opportunity to acknowledge some of the many people who have helped me through my graduate studies.

I am deeply grateful for the leadership and direction of my advisor Dr. Kenneth Ricks. His expertise has helped guide me from scheduling the initial semester of classes to finalizing the last chapter of the thesis. He always found time to meet with me about problems or questions I had from the beginning, and always worked so hard to get the thesis revisions back when time was growing short at the end. The jokes and laughter that inevitably found their way into most meetings made the whole graduate experience much more enjoyable. Thank you, Dr. Ricks, for being my mentor.

I would also like to say thank you to Dr. Jeff Jackson and Dr. Monica Anderson for agreeing to be part of my thesis committee. I've learned a great deal from both of you over the years through classes, labs, and office hours; and I appreciate you being part of my last big test.

I'm grateful for the help of Andrew Faulkner and David Sandel, who introduced me to the point cloud software and provided all the raw lidar data used in this research. You battled faulty power equipment and temperamental routers to get that data, and I thank you. And thanks to Caleb Leslie for building a great new testbed after the lab was relocated. I couldn't have done this without you guys.

I would like to express my gratitude to Dr. Susan Burkett, Dr. Sushma Kotru, and all the other professors with whom I worked teaching labs and classes during grad school. This teaching

experience has helped me become more confident in speaking and leadership, and I appreciate the opportunity you afforded me to grow these skills.

I'd like to thank Maureen Beard, Leslie Jones, and Debbie Woods from the ECE office who have been so helpful all the way back to the beginning of my time at the university. Everything from class flowcharts to ordering parts for a lab, you were always willing to lend an ear and find an answer.

I'm grateful for Drew Taylor, Priya Bangal, and Scott Corley for sharing an office with me and for being great friends. Through all the classes, meals, and long hours grading papers you made me laugh and enriched my time at Alabama – thank you.

I'm so thankful for my family and all my friends who have been there to support and encourage me. I have the best parents who have always made sure I was taken care of, and along with my friends have always been there to do everything from talking to moving everything I own. Thank you for being such an integral part of my life before, during, and after my time in school.

I have met so many great people at the University of Alabama that it is impossible to name them all. And yet when I look back, I don't see professors, staff, fellow grad students, or undergrad students from my labs. I see friends who have all had an impact on my education and my life. Thank you – and Roll Tide!

CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Robots are ubiquitous in today's modern technological society. Some robots are used in industry where they repeatedly follow a preprogrammed set of movements from a stationary base for tasks such as welding and painting on an automobile assembly line. There is another class of robots which are built on mobile platforms. Mobile robots move around in their environment, which is often highly unstructured, dynamic, and unpredictable [1].

As autonomous, mobile robots begin to traverse their surroundings, it becomes important for them to know their location within the environment as well as the locations of any obstacles [2]. One way to accomplish this is by engineering the environment [1]. This is done by placing artificial landmarks within the operating space (e.g. buried wires to guide autonomous lawnmowers) or by embedding sensor systems within the environment which observe the robot and communicate navigation instructions back to it wirelessly [3]. However, this is not feasible in cases where the environment often changes or is unexplored. A second method, usually seen with more complex robots, is to have the sensors reside on the robot itself where they can observe the environment and make onboard navigation decisions.

Some sensors commonly mounted on smaller robots include relative position sensors such as inertial and odometry sensors for navigation [4] as well as ultrasonic and infrared (IR) proximity sensors for obstacle detection. Odometry sensors measure wheel rotation while inertial sensors can be gyroscopes or accelerometers. These provide position estimation but cannot be

solely relied upon for navigation due to unbounded error introduced by wheel slippage and the integration of gyroscope or accelerometer output [2][5][6]. Ultrasonic and IR sensors can detect nearby obstacles, however their range and information detail are very limited thereby rendering them suitable for only rudimentary obstacle avoidance [7][8]. More complex sensors that have greater range and scanning resolution are needed for not only the absolute positioning required for better navigation, but also the ability to provide obstacle negotiation as opposed to simple obstacle avoidance.

Some common exteroceptive sensors better suited to the task are radar, stereo vision, and lidar. While radar can have a longer range compared to ultrasonic and IR, the system is expensive and relatively large. Therefore it is usually reserved for larger and more expensive vehicles such as cars, airplanes, and ships. Stereo vision systems are compact but require a great deal of back end computation to transform the images into usable depth maps before they can be used by the navigation and obstacle negotiation algorithms. Lidar is a laser based Time-of-Flight (ToF) system which is commonly found in two varieties: two-dimensional (2D) and three-dimensional (3D) scanning. Two-dimensional laser scanners are popular and widely used within the mobile robotics community [2][9] for tasks such as: object following and obstacle avoidance [9][10], feature extraction [11], localization [12], and map building [13][14]. These 2D scanners are good for navigating corridors or other spaces with tall walls, but they only scan a single horizontal plane and are therefore incapable of detecting obstacles that do not intersect the scanning plane. A common method to improve upon this is to mount the 2D laser scanner to a pivoting platform that allows it to tilt vertically as it scans horizontally, thereby creating a sensor capable of rendering a full 3D image. This type of 3D scanning lidar, utilizing the Hokuyo UTM30LXEW lidar system, is one of the two sensors used in this research.

Another type of ranging sensor that has come into popularity recently is the structured light depth sensor. The success of Microsoft's Kinect has resulted in the proliferation of inexpensive structured light sensors. These devices are desirable in the field of robotics for several reasons: they are less expensive than traditional laser scanners, they can capture a 3D image faster than a 3D lidar system, and they are less computationally intensive than stereo image setups. However, the major drawback to structured light sensors is the noise inherent to the depth measurements. Even under ideal circumstances, stationary objects in the field of view (FOV) can appear to have points with fluctuating distances. This is especially true near boundary edges. In order to be a useful alternative to other distance and ranging systems, one must understand whether or not structured light sensors can provide equally valuable data in the same environment. Other researchers have studied this question in environments found here on Earth, but more environments should be considered as mankind pushes past the boundaries of our planet.

The National Aeronautics & Space Administration (NASA) and most of the scientific community would like to one day see humans return to the Moon and build a colony there. It would be impossible to transport from Earth all the materials needed for a colony due to payload size restrictions and the astronomically high cost per pound associated with launching a payload beyond the atmosphere. Therefore some necessities could be transported while other resources which would be in high demand, such as oxygen, would need to be gathered on site. It is known that the lunar soil, or regolith, contains oxygen and could be mined and processed to provide in situ resource utilization [15][16]. This would allow astronauts to remain on the Moon for longer durations. There have also been discussions on other uses for the different components and properties of the regolith [17][18]. In order to be used for any purpose, the soil would first need

to be collected by excavators. These machines will need to run continuously, making it impractical to use human operators. The solution is to design mobile, autonomous excavators using sensors and navigation software. These excavators will stir up considerable amounts of dust while operating. This dust, in addition to what is already expelled from the surface through an electrostatic phenomenon known as dust fountains [19][20], could interfere with the sensors needed to scan the environment for autonomous navigation and obstacle avoidance.

It should be noted that sensor choices are more limited when considering lunar applications. GPS is not available for accurate positioning, and ultrasonic technologies are unusable. Therefore, distance sensors are well suited to lunar applications for both navigation and obstacle avoidance. The purpose of this research is to compare two technologies for their use in an autonomous robotic lunar mining application. Specifically, the Kinect is compared with a more traditional laser-based distance sensor in a harsh, lunar-like environment. Tests are conducted to evaluate the impact of lunar-like soil on the sensors' ability to collect accurate data suitable for use in autonomous robotic navigation applications.

The remainder of this thesis is organized as follows: Chapter 2 discusses other research related to this work. Chapter 3 introduces the hardware used in the experimental setup and provides an insight into the operation of the individual components. Chapter 4 outlines not only the procedures followed during the data acquisition and analysis phases, but also the software programs which facilitated both of these activities. Chapter 5 presents the data gathered during testing as well as a comparison of the results. Chapter 6 summarizes the knowledge gleaned from the data and reveals future work to expand upon the research presented here.

CHAPTER 2

RELATED WORK

Projects directly related to the performance of navigation sensors in dusty environments are rare. Those specific to sensor performance in lunar applications are even more difficult to find. However, there are articles indirectly related to this research that help to put this project into context.

One group of related work deals with the dust encountered on the Moon and how to handle it. The major process for weathering and erosion on the Moon is micro-meteorite impact. This produces the finely textured outer blanket of the Moon known as lunar regolith which consists of numerous particles of various sizes. Lunar dust (defined as particles < 20 um) makes up about 20 wt% of the typical lunar soil [18]. After samples of the regolith were returned to Earth, scientists realized that the individual particles are very jagged with a reentrant surface structure that enables them to physically "hook on" like Velcro, making them hard to remove [21]. This is different from the dust and sand particles on Earth which have been smoothed from erosion by wind and water, and rarely hook to objects and surfaces. These regolith particles pose a health hazard to astronauts if tracked back inside the habitat and inhaled. On later missions the crew had a vacuum system to help remove some of the dust from their suits. However, it quickly failed due to the abrasive nature of the lunar soil [22]. Therefore researchers are investigating self cleaning coatings that can prevent dust from adhering to the fabric of spacesuits [18][23].

One approach is superhydrophobic coatings that mimic the material properties and surface structure of the lotus leaf. These leaves are considered self cleaning because contaminates cannot attach and are easily removed by wind, water, or shaking of the plant. Similar approaches include superhydrophillic and catalytic coatings which are able to not only shed debris but also neutralize unintentionally captured microorganisms or harmful chemicals. This is important both on Earth during assembly and on the Moon to prevent inadvertent and possibly catastrophic cross contamination.

Devices using optical elements such as video cameras and solar panels can also suffer in dusty environments. Accumulated lunar dust is estimated to reduce solar power system efficiencies by as much as 50% [23]. Some researchers are developing systems to clean these devices by taking advantage of the aforementioned electrostatic properties of lunar dust and using electrostatic traveling-waves [24]. The device functions via a conveyor consisting of parallel transparent electrodes printed in a vortical pattern on a glass substrate. A power supply provides a four-phase rectangular high voltage signal to the four electrodes in such a way that a traveling-wave is set up across the glass. This allows the charged dust particles to be transported to the edge of the glass by the sweeping movement of the wave. Similar systems described by the same author also allow the technology to clean the suits of astronauts by means of a handheld device [25] or conductive fibers woven into the fabric itself [26]. Still others are looking at harnessing the electrostatic properties of the dust by designing electromagnetic field generating systems that could be built into the decompression chamber [27]. This would allow all of the astronauts' suits to be cleaned during the depressurization process.

The dust found on the Moon's surface is a serious hazard to astronauts and their equipment, giving validity to the research dedicated to controlling it. However, while coatings

and active electronics can provide for the mitigation of lunar dust on sensors and their lenses, they do nothing to address the effect of the dust on the *data* provided by the sensors.

Other work relevant to this project concerns lidar data performance and how it compares with other distance and ranging equipment in different Earth environments. In [28] two scanning laser range finders, the Sick LMS291-S05 and the Riegl LMSQ120, are compared with the 2D HSS, a 95 GHz scanning millimeter-wave radar, to evaluate their performance in adverse environmental conditions. A rectangular test chamber is setup with the sensors on one end and various targets at the other end. Controlled rain, mist, and dust conditions are replicated within this volume. A sprinkler system installed along the roof simulates rain and mist conditions, and suspended dust conditions are created by a fan inducing airflow across a vibration table onto which dust of various particle size distributions and material is delivered. During the dust tests, there is no negligible effect observed in the radar output. In contrast, the laser sensors demonstrate that target acquisition failure is abrupt – probability of target acquisition varies from 100% to complete failure within a very small change in transmittance (the fraction of laser light that passes through the dust). This would suggest that at least some types of suspended dust could have a dramatic impact on lidar performance.

In [29] there is a comparison of 10 range sensors for underground void modeling; however, no adverse environmental conditions such as dust are present. Five of these sensors are scanning lidar devices, including the Hokuyo UTM-30LX which is similar to the one used in this research. An interesting note is that the authors adapted these to perform 3D scans by mounting them to a 360° rotary actuator instead of tilting the scanner. Other sensors used are a flash lidar, phase shift lidar, stereo camera system, and two structured light systems including the Kinect. Tests are conducted by scanning an unstructured corridor, an unstructured intersection, and a

structured corridor. The testing results show the phase shift lidar to have the best performance, followed by the group of ToF scanning lidar sensors. However, the authors praise the Kinect stating that it greatly outperforms its pricepoint. They also conclude that for the motivating tunnel application, the Hokuyo UTM30-LX is the preferred sensor due to it having "the right balance of performance, mass, features, and cost."

The authors of [30] have titled their work "Are laser scanners replaceable by Kinect sensors in robotic applications?" and seek to answer the question by comparing the Kinect with a Hokuyo URG-04LX-UG01 and a Sick LMS200. However, instead of modifying the lidar sensors to capture 3D scans, they modify the Kinect by isolating a single horizontal line from its output. This effectively transforms the Kinect into a 2D sensor thereby eliminating the advantages associated with 3D capture. Three typical robotic applications are considered in this work: obstacle avoidance, map building, and localization. It is determined that the Kinect is better at obstacle detection but not map building or localization. The authors ultimately answer the title question with "No," stating the most important disadvantage of the Kinect is its small horizontal FOV.

Two other papers compare the Kinect with a Hokuyo UBG-04LX-F01 [8], and a Hokuyo URG-04LX and Asus Xtion Pro Live [31]. The authors do not conduct any tests in adverse conditions, but do present similar observations about the sensors. Both state the lidar is superior in terms of horizontal FOV and minimum measureable range, but due to the amount of time needed to collect a 3D scan (due to physically tilting the sensor) it could not be used for real time 3D applications. Concerning accuracy, [31] ranks the lidar as superior while [8] gives mixed results which are dependent on the target color.

One paper, [32], only tests lidar sensors. However, it does so in risky and adverse environments such as those found in firefighting applications – including smoky environments. Two scanning laser range finders, Sick LMS200 and Hokuyo URG-04LX, and two industrial laser range finders, IFM Efector O1D100 and Sick DT60, are evaluated. Two reduced visibility tests are conducted – one using water vapor from an ultrasonic air humidifier and another using induced smoke from a smoke machine. In both tests the sensors are placed two meters away from a white medium density fiberboard surface. All of the sensors are affected by the water vapor and also perform poorly in the smoky conditions with the first two sensors performing worse than the industrial ones. Again, this would suggest that lidar systems are susceptible to low visibility conditions.

Kise et al. [33] compares a ToF camera, two scanning laser sensors, and a stereo vision system (no information is given on specific models) in dusty conditions encountered with agricultural vehicle automation applications. Three tests are conducted in a closed chamber with a white cylinder placed at 5 m, 10 m, and 15 m as a target. Peat moss is used as artificial airborne obscurant and is manually agitated using a leaf blower. The conclusions show that scanning lasers have superior performance in terms of measurement range but "did not exhibit superior performance in an airborne-obscurant rich environment." While this reinforces the conclusions from other work, the authors went on to state that the two lidar sensors "exhibited very different performance" and that "similar specifications do not necessarily imply similar performance." So it is likely that all laser scanning systems may be impacted by dust, but some could be affected more than others.

Perhaps the work done in [34] would be considered the most closely related to this research. Here the authors compare the Hokuyo UTM-30LX, Sick LMS-111, and Sick LMS-291

laser range finders with the Microsoft Kinect and a MESA SR-3100 ToF camera - also in smoky conditions like those found in firefighting. The sensors are set up one at a time in the same spot in an empty room on tripods (with the planar scanners rotating on a rotary unit) and configured to deliver 3D point clouds. All tests are first performed with no smoke to obtain a base reading, and then smoke from a fog machine is injected into the side of the room opposite the sensor. Two parameters are analyzed: the change in number of points returned with valid distances, and the change in number of points belonging to a particular extracted plane. The target plane in all tests is the floor and is extracted using a random sample consensus (RANSAC) method provided by the Point Cloud Library (PCL). Analysis reveals all lidar sensors are negatively affected by the smoke with the Hokuyo having a higher sensitivity than the Sick LMS-111. The Kinect performs well compared to the ToF camera, and surprises the authors with the amount of valid data returned. These results coincide with all those already reported from similar projects.

The results from these projects are related to this research by means of comparing the performance of range sensing devices, some even in adverse conditions. However, none directly investigate the impact that dust would have on the ability of a lidar and Kinect to provide autonomous navigation data in a dusty lunar environment.

CHAPTER 3

EXPERIMENTAL SETUP

There are several pieces of hardware needed to evaluate and compare the sensors' performance. The first, and largest, is the regolith testbed. This is the location in which all of the experiments are carried out, and the origin of the dust used to test the resilience of the equipment. It is desirable to have the ability to quantify how much dust is in the air during a test, so a sensor system is needed to measure suspended dust levels. And, the depth sensors themselves are needed to capture depth data using various targets.

3.1 Regolith Testbed

The regolith testbed used in these tests is a room measuring 6.70 x 2.26 x 5.08 m (LxWxH) and is part of the University of Alabama Lunabotics Lab. This is a facility dedicated to the development and testing of various robotic lunar excavators. This room has therefore been specifically designed to simulate the regolith and mining conditions found on the Moon. A diagram showing the layout and dimensions of the regolith testbed is given in Figure 3.1.

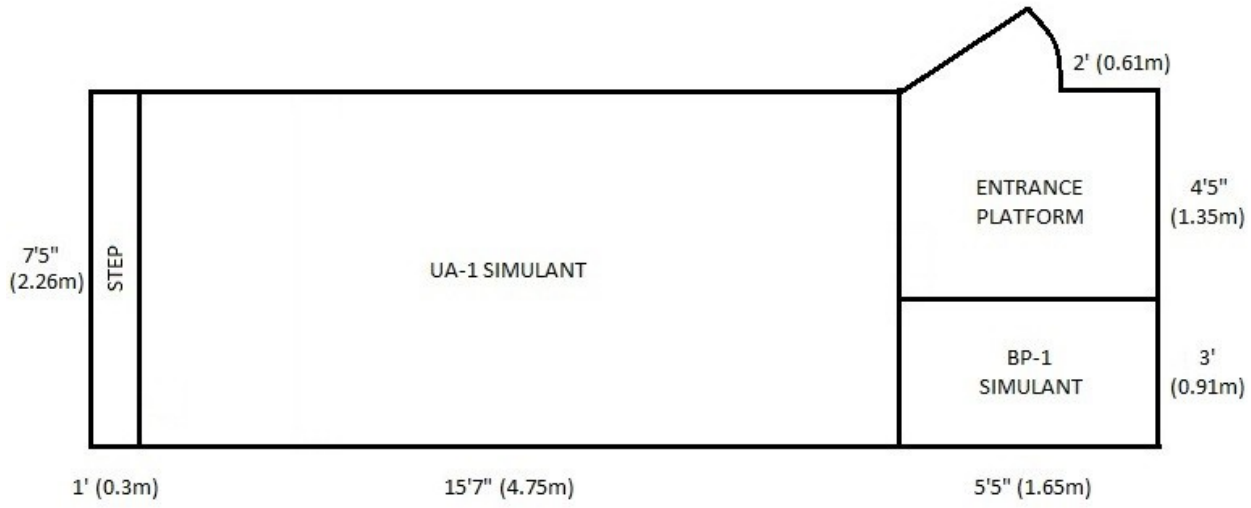


Figure 3.1. Regolith Testbed Diagram

The regolith testbed is divided into three areas. After entering the room and stepping up onto the entrance platform area, one has immediate access to both the BP-1 and UA-1 regolith areas. The BP-1 area is one meter deep to support digging at depths and houses the BP-1 lunar simulant. This is a simulant currently used by NASA and is made from the Black Point basalt flow, San Francisco Volcanic Field, in northern Arizona [35]. BP-1 is difficult to obtain and rather expensive. Therefore, a custom mixture called UA-1 and comprised of a 3:1 ratio of mortar mix to dry industrial sand was created at the University of Alabama to mimic the BP-1. This mixture is more easily obtained and less expensive than BP-1 while maintaining similar particle sizes and suspension and compaction characteristics. Its availability allows for a much larger area for testing. The simulant in the UA-1 area is 5-12 cm deep, and this area is where the majority of the experiments are conducted. Figure 3.2 is comprised of two images of the testbed: one from the entrance platform (left) and one from the opposite end of the UA-1 area (right).

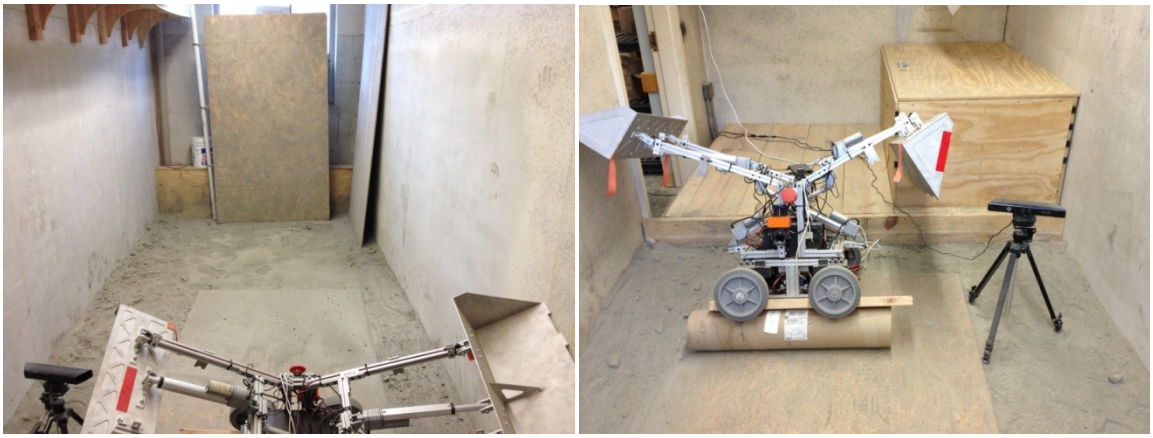


Figure 3.2. Regolith Testbed

3.2 Dust Sensor System

A mechanism is needed to monitor how much dust is suspended in the air to quantify at what dust level the navigation sensors are affected. Initially, several commercial devices were considered from companies such as Lighthouse [36] and TSI [37]. Both sell devices that are handheld and can detect particle sizes down to 0.01 µm. For example, one device from Lighthouse [38] has six particle size channels, can perform cumulative or differential measurements, and also gives mass concentration readings. This means it is capable of dividing and counting the dust particles based on size and presenting the data either cumulatively (all particles larger than channel size) or differentially (all particles between channel sizes). The only part that is relevant to this research is the mass concentration which estimates how much dust is suspended and gives a readout as amount of material per cubic area, usually mg/m^3. There are two downsides to all these products. First, they only have one sampling point. A system with multiple sensors which can be distributed will give a more comprehensive estimation of suspended dust levels within the test area. Second, they are prohibitively expensive. After

carefully considering the options, it was determined that a custom, distributed system was the best choice for this project.

The final system design consists of four sensors mounted vertically on a pole at 0.61 m, 1.22 m, 1.83 m, and 2.44 m above the ground. This allows one to see how the dust rises, moves, and settles with more detail than a single sampling point can provide. Each sensor box is connected to the central controller box via a universal serial bus (USB) cable. They do not use the USB communication protocol. However, two power and two data wires are needed, and the ubiquitous USB cable interface was a natural choice. The main control box powers the sensor array, gathers the analog output from the sensors, and relays the data wirelessly to a computer outside the enclosed regolith testbed.

The entire system is designed around the Sharp GP2Y1010AU0F optical dust sensor [39]. This uses an infrared light emitting diode (LED) and a phototransistor arranged diagonally in a small housing to allow it to detect the light reflected from dust in the air that passes through a small hole. A picture is shown in Figure 3.3 along with a schematic showing the function of each of the six connector pins.

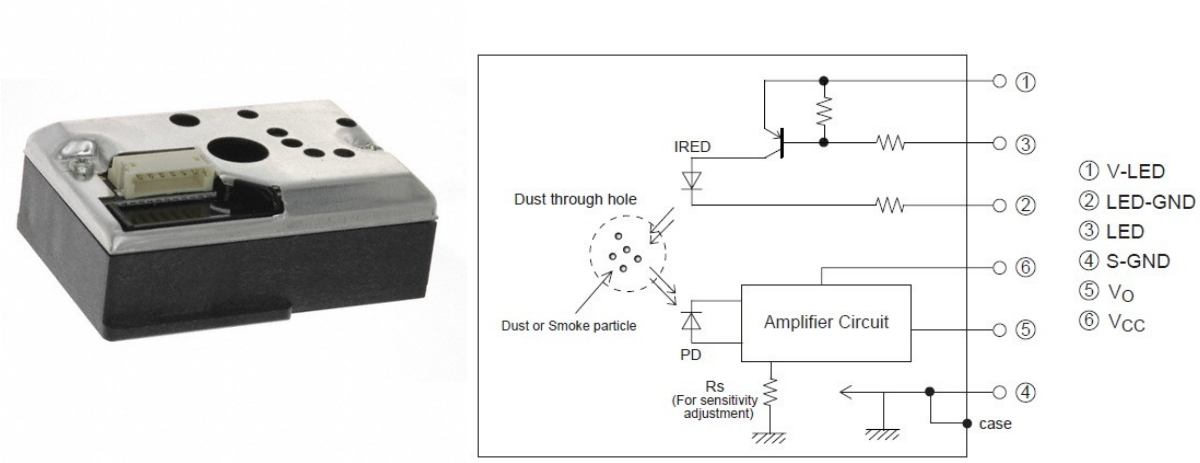


Figure 3.3. Sharp Dust Sensor and Schematic [39][40]

Each sensor is powered by 5 volts direct current (VDC) with 11 mA typical current consumption. The output of the sensor is an analog voltage proportional to the dust density with a sensitivity of 0.5 V / 0.1 mg/m^3. The detection range is approximately 0 - 0.7 mg/m^3. Each sensor requires a few external components to function properly so a section of prototyping board is used to build the necessary circuit. This circuit is given in Figure 3.4 along with a diagram of the USB B-type connector that shows how the four USB wires are used for sensor input and output. The red numbers in Figure 3.4 designate the connection points of the corresponding pins in the dust sensor schematic from Figure 3.3. An N-channel MOSFET is used to control the dust sensor sampling LED and a small external case LED that gives a visual indication when it is in operation. One RC branch powers this external LED while the other RC branch powers the sensor. All of these components are housed in a small plastic enclosure that protects them from the dust. Images of the final circuit in the enclosure are shown in Figure 3.5.

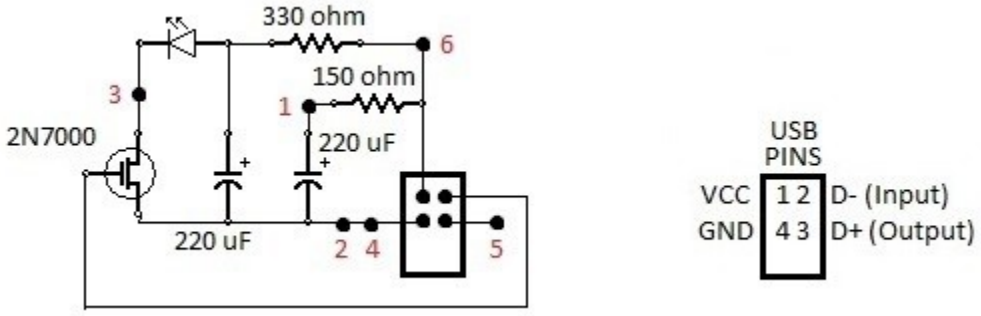


Figure 3.4. Individual Dust Sensor Power Board Schematic

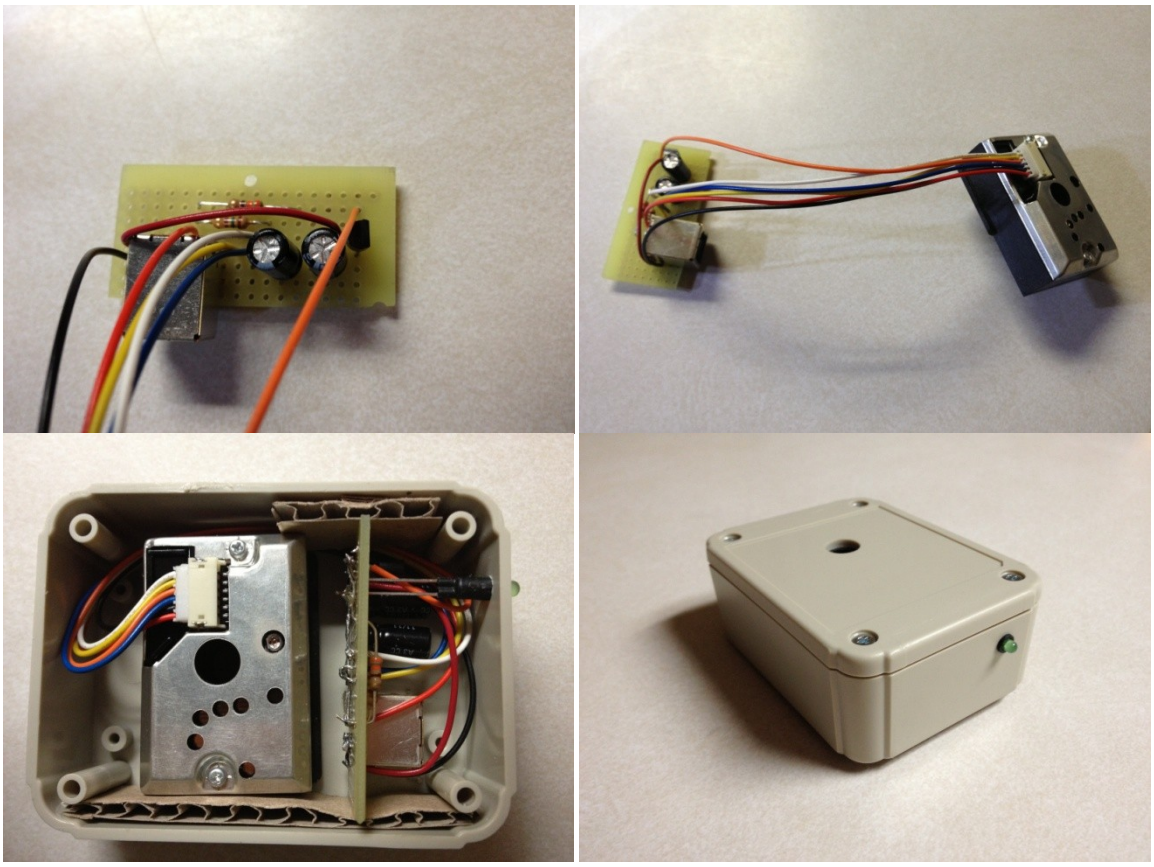


Figure 3.5. Dust Sensor and Power Board with Enclosure

The central controller unit consists of a main power board and a microcontroller. The main power board is custom designed to take 7 - 9 VDC from either an external wall adapter or an internal battery and regulate it to 5 VDC for the system. The power is then divided into two branches with each fused at 500 mA to protect against short circuits. The first branch is dedicated to the onboard internal USB plug powering the microcontroller. The other branch feeds an onboard LED (to verify the system is active) and the power bus connected to each of the external USB plugs for the sensors. A schematic of this board is given in Figure 3.6.

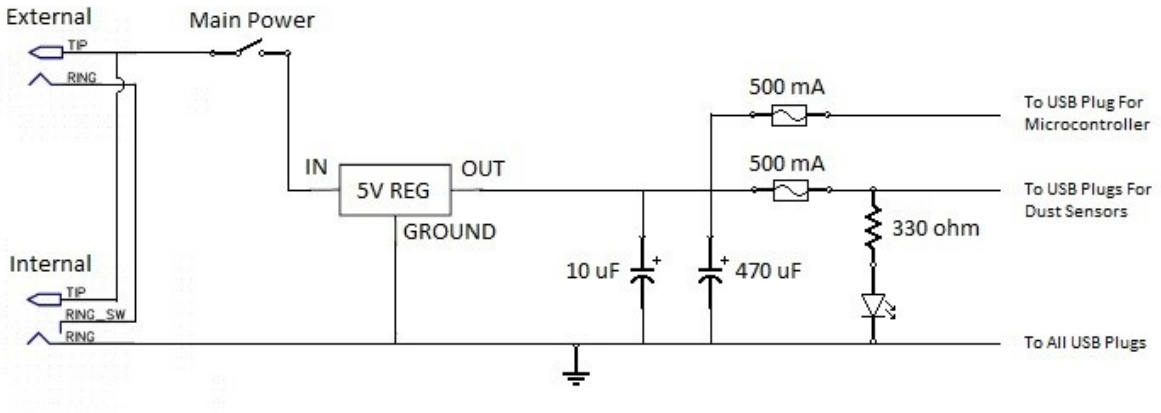


Figure 3.6. Main Power Board Schematic

The microcontroller used for this project is an Arduino Mega 2560 which is coupled with an XBee shield to provide wireless communications [41]. Figure 3.7 gives an image of the Arduino. Each dust sensor node requires one digital output and one analog input on the microcontroller to function. The Arduino has 54 digital input/output pins and 16 analog input pins, allowing it to administer up to 16 dust sensors. The data pins from each of the external USB sensor connections are wired to the corresponding input/output pins on the controller board. The digital output pin must be driven with a specific waveform to meet the operational requirements of the dust sensor, and the analog pin must sample the sensor output at a precise point during the waveform in order to obtain valid readings. These waveforms and timing requirements are provided in Figure 3.8 with sensor input on the right and sensor output on the left. Custom software was written and loaded on the microcontroller enabling it to properly interface with the sensors. This program performs a group of 50 air quality samples in 0.5 seconds. It then stops sampling, averages the data for each of the four sensors, and transmits the four averages wirelessly. These events are repeated in a loop that provides new data once per second. The number of dust sensors in the system, number of samples taken in a group, and wait time

Figure 3.7. Arduino Mega 2560 Microcontroller [41]

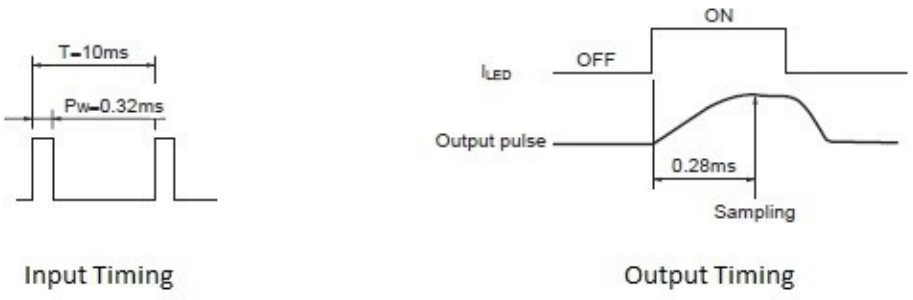


Figure 3.8. Timing Diagrams of Sensor Input and Output Signals [39]

between sample groups are all customizable by simply changing the appropriate values in a header file. The complete code is provided in Appendix A.

All components of the central controller unit and their wiring are housed inside a sealed enclosure to protect them from the dust. Additionally, there is an external normally open pushbutton switch connected to the reset pin on the microcontroller so in the event of a problem the board can be reset without opening the enclosure. Images of the completed central controller unit are shown in Figure 3.9.

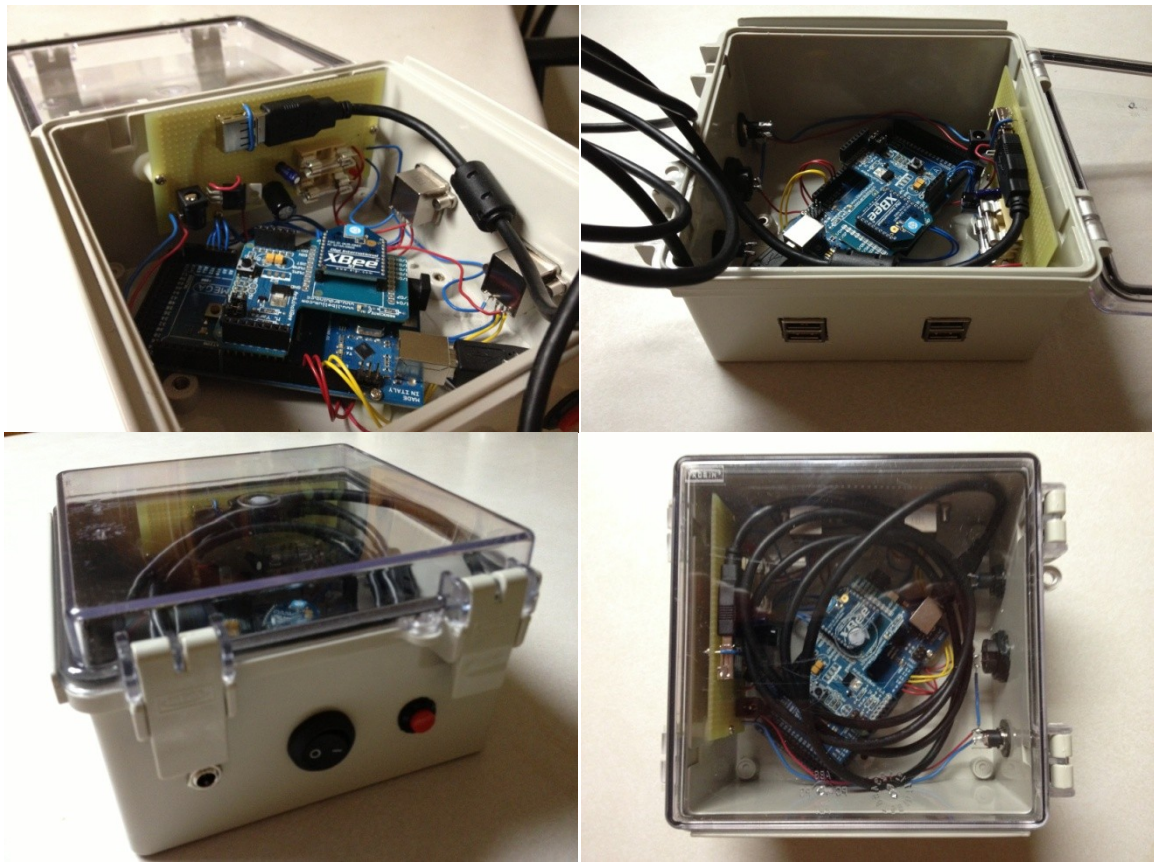


Figure 3.9. Central Controller Unit for Dust Sensor System

3.3 Depth Sensors

This research focuses on two types of distance or depth measuring devices. The first is a structured light sensor and the other a ToF sensor.

3.3.1 Structured Light

The chosen structured light sensor is a Microsoft Kinect for Xbox 360 [42]. This device communicates via USB and contains three major components: a depth system consisting of an IR emitter and IR depth sensor, a color sensor, and a microphone array. Figure 3.10 is an image of

the Kinect with these components highlighted. The RGB camera (color sensor) is capable of returning color images of up to 1280x1024 resolution at 30 Hz. The microphone array consists of four microphones along the bottom of the sensor, and a 24-bit ADC along with Kinect-resident signal processing allows for functions such as noise suppression, acoustic echo cancellation, and source localization [43]. Onboard processing also allows the Kinect to directly generate skeleton data for player tracking and positioning. These features have importance in other applications; however, this research is interested in the raw depth data.

The raw depth data is provided in frames where each pixel represents a disparity value instead of a color or intensity value. The maximum resolution available is 640x480 pixels at 30 Hz. Frames are captured using the IR laser and camera (IR emitter and depth sensor) via a process known as triangulation. The laser emits a single beam which is split into multiple beams by a diffraction grating to create a constant pattern of dots projected onto the scene. Examples of this pattern are shown in Figure 3.11. This pattern is captured by the infrared camera and is compared to a reference pattern. The reference pattern is obtained by capturing a plane at a known distance from the sensor, and is stored in the memory of the sensor

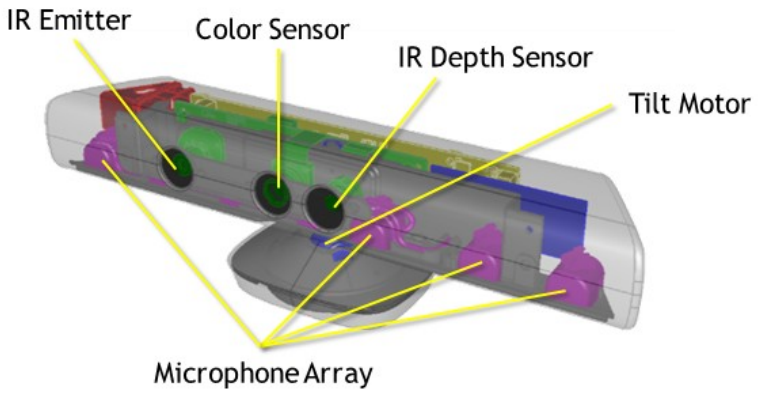


Figure 3.10. Microsoft Kinect Sensor [43]

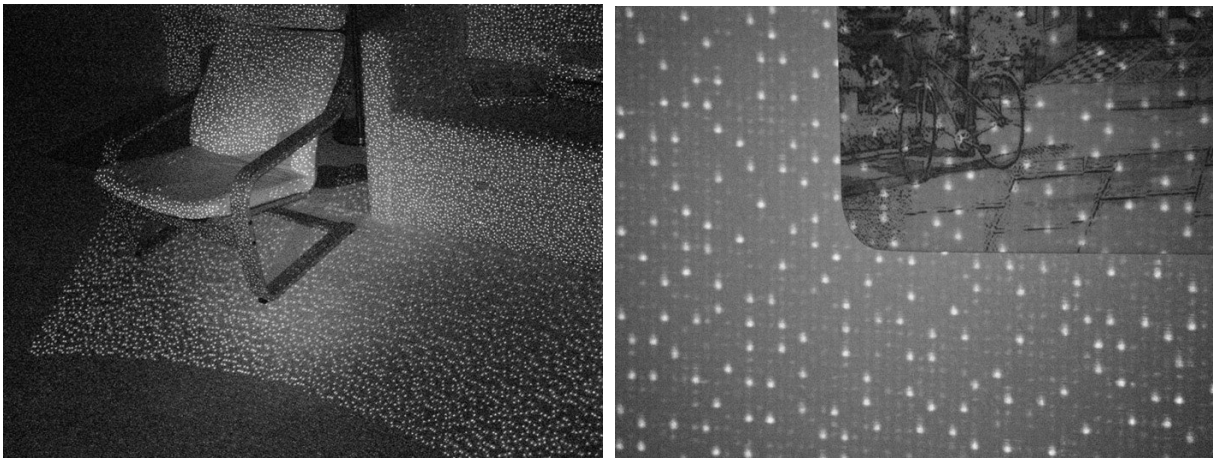


Figure 3.11. Examples of IR Dot Pattern Projected by Kinect [44]

during the manufacturing process. When an object enters the sensor's FOV, it distorts this pattern and shifts the affected dots from their original position. These shifts are measured for all dots by an image correlation procedure, which yields a disparity image. The disparity values are normalized to a value between zero and 2047 (11 bit integer) before being returned by the Kinect where the user's software can then retrieve the corresponding distance value for each pixel. It is important to note that the distance value this provides is not the distance from the object to the camera, but rather the distance from the object to the camera plane as demonstrated in Figure 3.12. More technical details about the Kinect, in addition to the operational information given here, can be found in the exhaustive sensor characterizations referenced in [45] and [46].

There are some inherent disadvantages with the Kinect sensor. The first is the decrease in depth resolution and increase in quantization error as the distance to target increases. Quantization error is the distance between successive depth values, i.e. depth resolution. It has been stated that the Kinect returns an 11 bit output that, in principle, represents 2048 levels of disparity. However, it mostly returns disparity values between 400 and 1050, restricting the

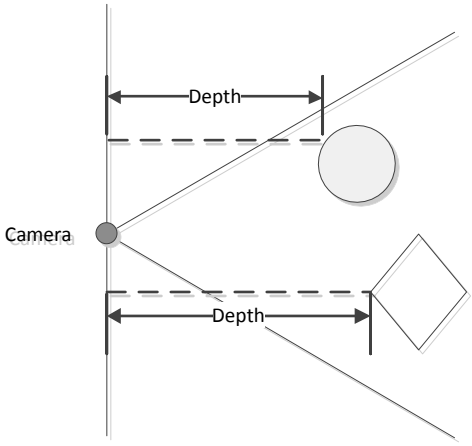


Figure 3.12. Object Distances to Camera Plane [47]

distance output between 0.5 m and 8.5 m. This suggests a maximum range of 8.5 m; however, the furthest operational range is considered to be 3-5 m because the depth resolution decreases non-linearly as the distance to the object increases. For example, the Kinect returns 350 disparity levels between 0.5 m and 1 m, while only 23 disparity levels are returned between 2.5 m and 3 m. This decrease in resolution causes the quantization error to increase exponentially. The error ranges from 7 mm to 25.8 mm within the operating region of the Kinect (0.5 – 3 m). This quadratic relationship between distance and resolution/error is due to the nature in which the projected light pattern spreads as it moves outward. A square object of size 40 x 40 cm is mapped by approximately 149,769 points at a distance of 60 cm from the sensor, while that same object is mapped by only 6084 points when located at 3 m.

Two other shortcomings are the sensor's susceptibility to infrared noise and its limited horizontal FOV. It is known that the Kinect performs poorly in IR saturated conditions such as outdoor environments [30][8]. In these conditions, the projected structured light pattern is overcome by the ambient infrared light and is no longer discernible by the camera. This was confirmed when the sensor returned blank frames with no valid pixels during an outdoor test in

sunlight. This places it at a disadvantage to the laser scanner, which can operate in sunlight. In contrast, both sensors would have an advantage over stereo image systems at night since any form of external illumination is not needed. Some researchers cite the 57° horizontal FOV of the Kinect as a drawback because it is narrower than the laser scanner. However, this can be remedied to a certain extent by using multiple sensors.

The 43° vertical FOV gives the Kinect an advantage over lidar, allowing it to capture a 3D image without moving. This provides an incredible increase in speed of image acquisition; up to 30 frames per second compared with one frame per minute for the laser scanner setup used in this research. This leads to another advantage – no moving parts. Eliminating the need to tilt or rotate the sensor reduces the probability of mechanical system failure.

Perhaps the most significant advantage is cost. At the time of writing, the Kinect retails for $100 – a price that is well below the cost of lidar systems. The Kinect's specifications are summarized in Table 3.1 at the end of the chapter.

3.3.2 Time-of-Flight

The ToF sensor used in this research is a Hokuyo UTM-30LX-EW scanning laser range finder (LRF) [48]. An image of this sensor is shown in Figure 3.13. This device operates on a 12 VDC power supply, communicates via an Ethernet interface, and is capable of multi-echo operation. Lidar sensors use IR lasers for ranging, but use a different technique than structured light. The LRF consists of a laser source, a laser detector, and a motorized rotating mirror. A single pulse of light is emitted from the laser, reflects off the mirror, and travels to an object in the scanning range. When the pulse hits a target, it is reflected back along the original path and

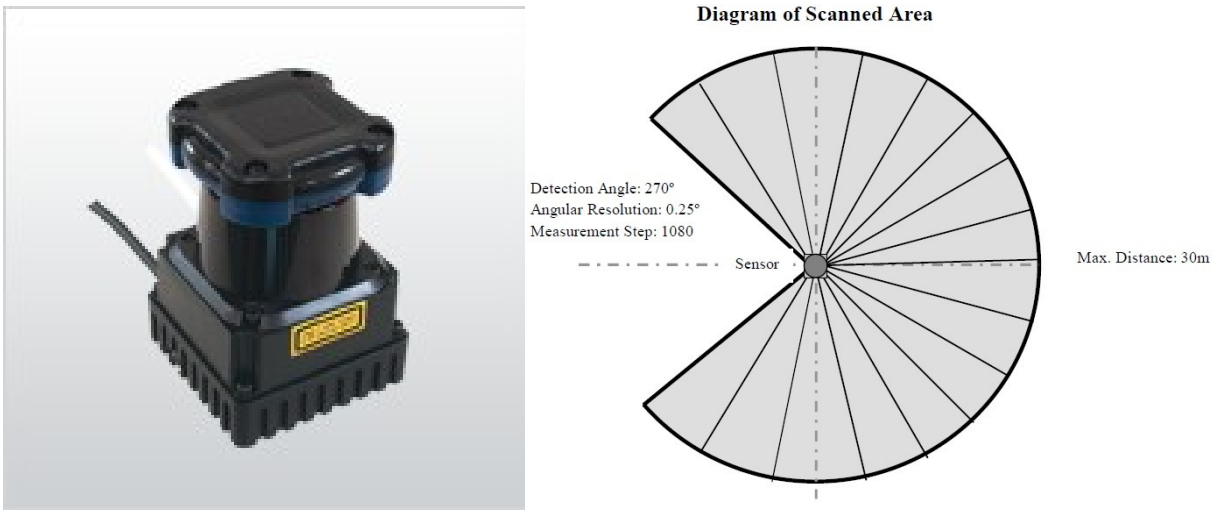


Figure 3.13. Hokuyo UTM-30LX-EW Sensor and Scanning Area [48][49]

is captured by the detector. A diagram is given in Figure 3.14 illustrating the laser and mirror configuration. Once the return pulse is detected, the distance to the object can be calculated using precision timing since the speed of light is a constant (300,000 m/s). The motor spins the mirror at 2400 rpm allowing the sensor to fan out the samples in a circular scan pattern, completing one full scan in 25 ms. This model lidar provides 1081 samples per scan over a 270° FOV, giving an angular resolution of 0.25° per sample. The guaranteed range for distance measurements is

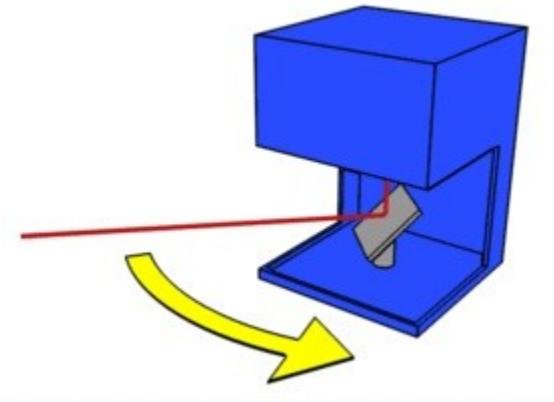


Figure 3.14. Scanning LRF Diagram [50]

0.1 ~ 30 m with a resolution of 1 mm and an accuracy of +/- 30 mm at a distance of 0.1 - 10 m and +/- 50 mm between 10 – 30 m. This range exceeds that of the Kinect, in regards to both minimum and maximum values. Lidar clearly does not suffer an exponential degradation in accuracy or resolution over distance like the Kinect. However, the smallest object size that can be seen by the lidar does increase with distance because of the divergence of the sampling beams. At 10 m an object must have a minimum width of 130 mm to guarantee detection by the sensor. Sensor characterizations for the functionally similar UTM-30LX can be found in [51] and [52].

The Hokuyo UTM-30LX-EW lidar was chosen for a specific reason: the multi-echo functionality. A single sample beam can produce multiple returns, or echoes, when it encounters obstacles such as: transparent objects; an object's boundary; or small particles such as rain drops, mist, dust, and fog. In these circumstances, the sensor may see one depth return for the object and another return for the background. Most lidar sensors will either report the first return and ignore all subsequent echoes, or combine them to produce an average value for the sample point. On the other hand, a multi-echo sensor reports each echo as a separate value, allowing the user to determine how they are to be handled. The UTM-30LX-EW is capable of reporting up to three echoes for every point. This is important in the motivating application because dust is one cause of multiple echoes. Two sets of lidar data were analyzed from each test in order to determine if the multi-echo functionality offers improved obstacle detection performance in dusty situations. One dataset contains only the first echo to simulate the performance of a single-echo lidar sensor. The other dataset contains the last echo for every point (second and third echoes are not always produced) to test the benefits of multi-echo functionality. The last echo was chosen due to the

fact the laser cannot penetrate a solid target. Therefore the target, if seen, will always be the last echo.

Despite all these features, the lidar is still a 2D sensor that provides only a single horizontal line with every scan. As mentioned before, the lidar must be tilted vertically to provide a complete 3D image. This is accomplished by mounting it to a servo-driven platform which enables a 70° vertical FOV. The platform tilts 0.5° between scans creating a total of 141 lines which are combined into the final 3D image. This physical movement of the sensor introduces time delays between each line of the image, causing a full 3D image to require one minute for capture.

Some advantages of the lidar sensor such as range and horizontal FOV have already been discussed. The mechanical reliability issue of requiring a separate pivoting or rotating mount to provide 3D image capture has also been reviewed. Another disadvantage is the cost of the sensor. At the time of writing, the Hokuyo UTM-30LX-EW retails for approximately $6000. This is 60x the cost of the Kinect. The lidar's specifications are summarized and compared to those of the Kinect in Table 3.1.

Table 3.1. Kinect and Lidar Specification Comparison

	Kinect	Lidar	
		2D Config	3D Config
Horizontal FOV	57°	270°	
Vertical FOV	43°	0°	70°
Image Resolution	640x480	1081x1	1081x141
Typical Depth Range	0.5 - 3 m	0.1 - 30 m	
Depth Resolution	7 - 25.8 mm *	1 mm	
Images/Scans per min	1800	2400	1
Cost	$100	$6,000	$6,000 +

* Depends on distance from sensor

CHAPTER 4

TESTING AND DATA ANALYSIS PROCEDURES

Chapter 4 consists of five sections detailing the testing and data analysis procedures. Section 4.1 describes the setup of the dust sensor system and the software used to collect the suspended dust level data. Section 4.2 explains the setup of the depth sensors and the software used to record depth images. Section 4.3 outlines the target positioning for each of the five testing configurations. Section 4.4 presents the process that was followed for collecting depth data at three dust levels in the five target configurations. Section 4.5 gives a detailed examination of the software used in the post processing of the depth data.

4.1 Dust Sensor Setup and Data Collection

The first task in running all tests is to initialize the dust sensor system. In section 3.2 it is shown that the hardware transmits air quality values wirelessly at a rate of 1 Hz. To enable this, one must simply connect the dust system to a power source and turn on the main power switch located on the side of the central controller unit. No other configuration is needed. The wireless data transmitted by the system is received by another Xbee radio shield which is connected to a computer via USB interface and emulates a standard serial communications (COM) port. A custom graphical user interface (GUI) executing on the computer receives the serial data and is able to simultaneously graph and save the readings. The GUI is written in Matlab using GUIDE

and is compiled with the Matlab compiler for a 64-bit system. A screenshot is shown in Figure 4.1 and the complete main code, including subroutines, is given in Appendix B. After opening the program, the user selects the refresh button under Dust Sensor System COM Port. This scans the computer and populates the drop down box with the names of available COM ports. After selecting the port corresponding to the Xbee wireless receiver, the user then selects the connect button. The GUI then connects to the COM port and begins monitoring it for data. The microcontroller of the dust system transmits the data in plain text using the following format: sensor 1 voltage (tab) sensor 2 voltage (tab) sensor 3 voltage (tab) sensor 4 voltage (newline).

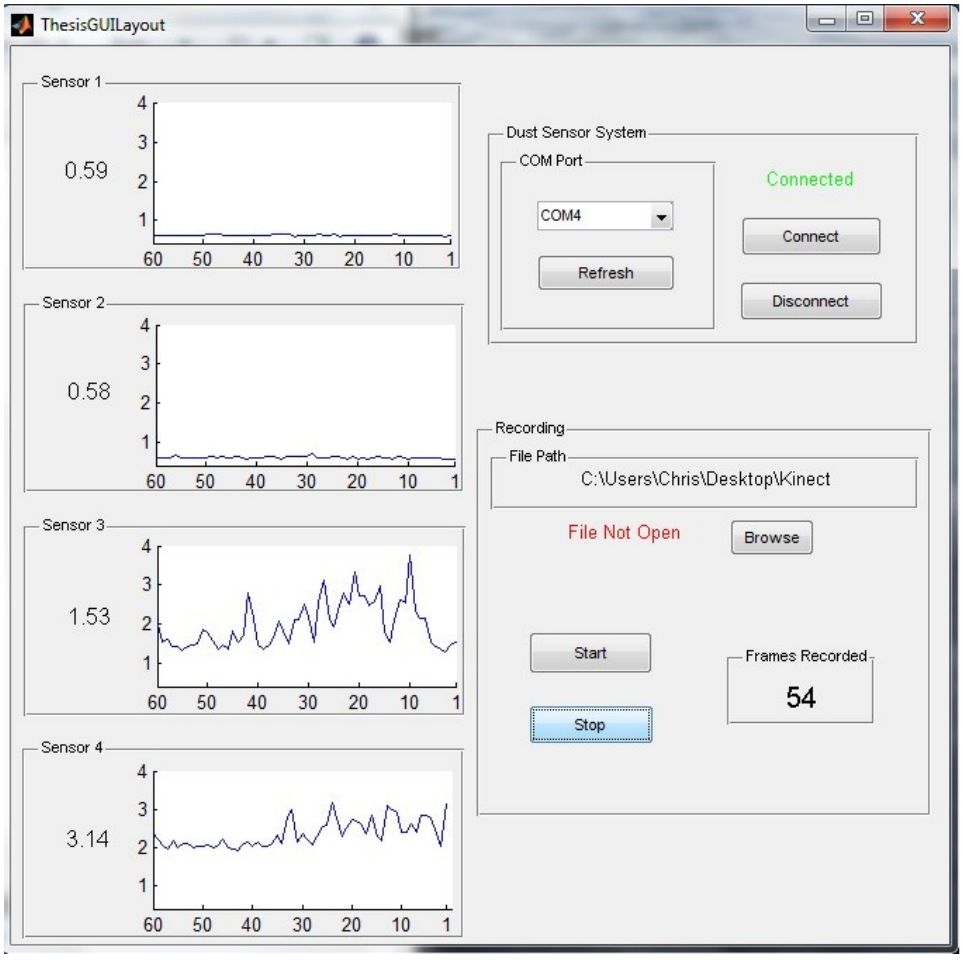


Figure 4.1. Dust Sensor System GUI Screenshot

When the newline is received, it triggers a subroutine that extracts the integer voltage values and displays them in the appropriate sensor boxes. It also plots the last 60 values received thus providing the ability to identify trends over the last minute. To record this data during a test, the user selects a folder using the browse button under the recording section and selects the start button. A text file is created in the chosen folder with the name "DataValues.txt" and new data values are recorded in plain text along with a timestamp and the frame number. The number of frames is also displayed by the GUI so the user can identify how many frames have been recorded. The stop button is used to stop recording and close the file. The disconnect button releases the COM port allowing the user to close the program.

4.2 Depth Sensor Setup and Data Collection

The Kinect and lidar sensors also have to be configured. The Kinect is mounted on a tripod during most of the tests to allow it to be adjusted to match the lidar's vertical positioning. It also uses the USB interface and requires a separate power connection. The Point Cloud Library (PCL) is used to handle all Kinect depth data [53]. A custom program called PCD_Writer was created to record depth data from the Kinect in the compressed binary point cloud data format (.pcd). This is a command line program which is called using the following argument format:

>PCD_Writer.exe [location to store files] [number of images to record].

Once started, the program begins to save images at a rate of 1 Hz using the naming convention DepthImage_[image number].pcd while also displaying every 10^{th} image in a preview window enabling the user to monitor the acquisition process. A screenshot of the computer desktop during recording is given in Figure 4.2 showing the command line window (bottom left), Kinect Depth Image preview pane (top left), and dust system GUI (right) executing simultaneously.

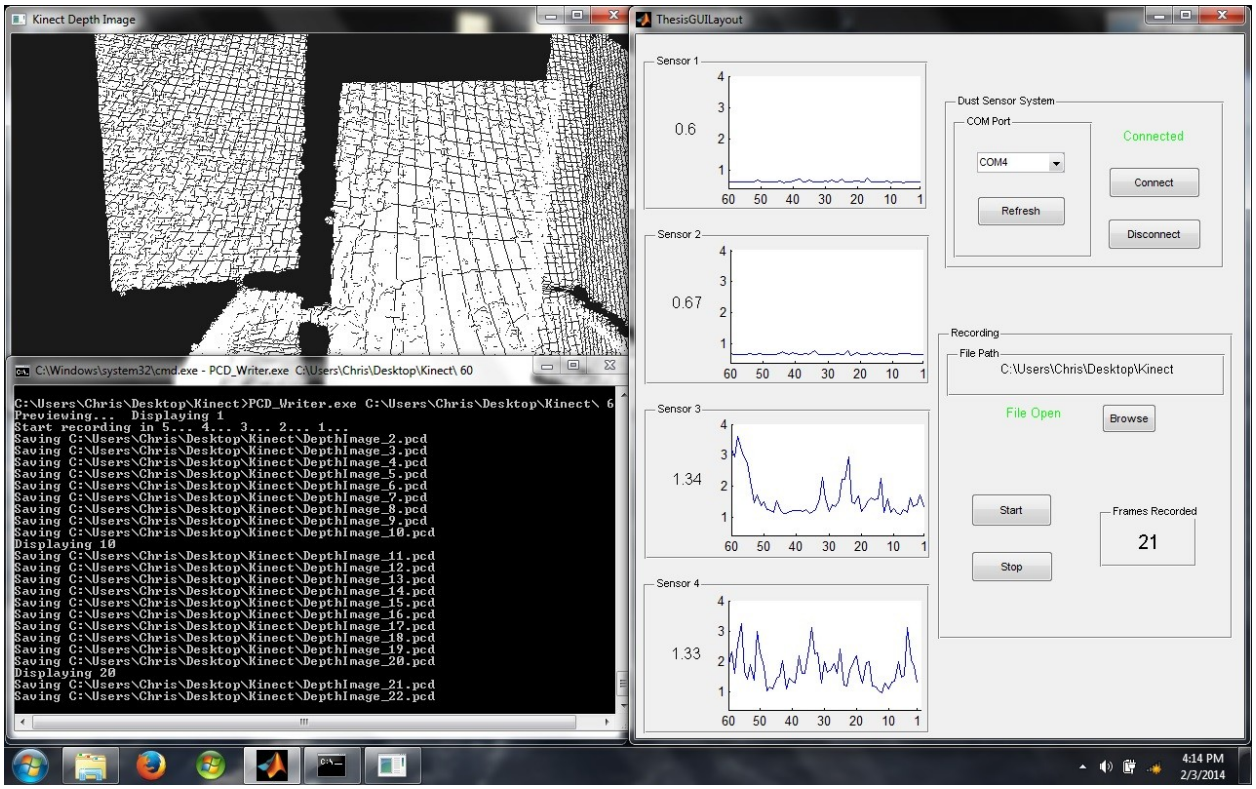


Figure 4.2. Desktop Screenshot During Testing

Once the PCD_Writer program has finished recording the desired number of images from the Kinect, the user is given the option to either enter a new number to record the corresponding additional images or enter zero to exit. The complete code is given in Appendix C. Another custom program named PCD_Reader simply allows the user to open a point cloud file after the recording process to assess the image and determine whether it is satisfactory. The code for this program is given in Appendix D.

The Hokuyo lidar is positioned near the Kinect with an emphasis on aligning the two sensors to have the same distance from the target. It is powered by a battery and 12 V DC-to-DC converter and communicates data to a host computer via an Ethernet interface. The servo platform, which allows the sensor to pitch thus creating a 3D view, is controlled via a USB cable.

The lidar begins the test with the scanning plane parallel to the floor and tilts downward through the 70° vertical FOV. The images are captured in raw data format which is converted into PCD format.

4.3 Target Configurations

Once the dust system, depth sensors, and software are ready for data capture, the targets are placed in their appropriate positions for testing. Five target scenarios were constructed in total. In tests 1 - 3, a sheet of plywood measuring 1.22 x 2.13 m is placed in the UA-1 area parallel to the sensor plane to simulate a large, ideal target such as a wall. The distance from the sensor plane to the target is 3.81 m for test 1, 2.59 m for test 2, and 1.07 m for test 3. A diagram is given in Figure 4.3 illustrating the positions of the Kinect in green, the lidar in blue, the dust sensor array in yellow, and the target in red at 3.81 m. Figure 4.4 is an image of the depth sensors and large target at a distance of 1.07 m.

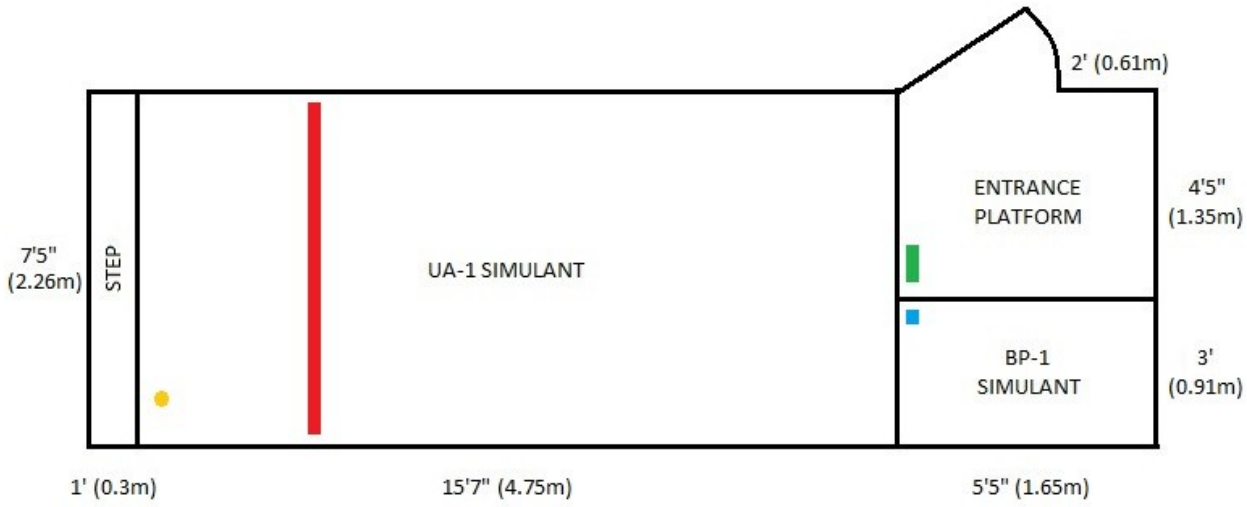


Figure 4.3. Test 1 Testbed Diagram with Kinect (Green), Lidar (Blue), Dust System (Yellow), & Target (Red)

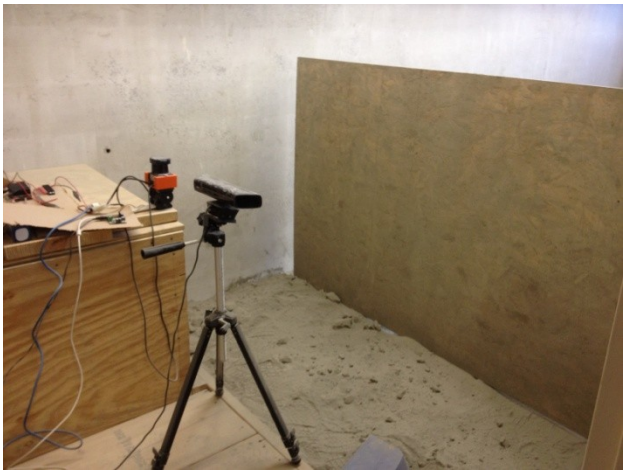


Figure 4.4. Test 3 Configuration

For test 4, two 0.28 x 0.56 m and two 0.28 x 0.81 m pieces of lumber were placed in the UA-1 simulant simultaneously at distances of 0.91 m, 1.52 m, 2.13 m, and 2.74 m to represent smaller targets such as obstacles. Figure 4.5 is a diagram illustrating this configuration. Pictures of the test 4 configuration during saturated, intermediate, and reference dust conditions are shown in Figure 4.6.

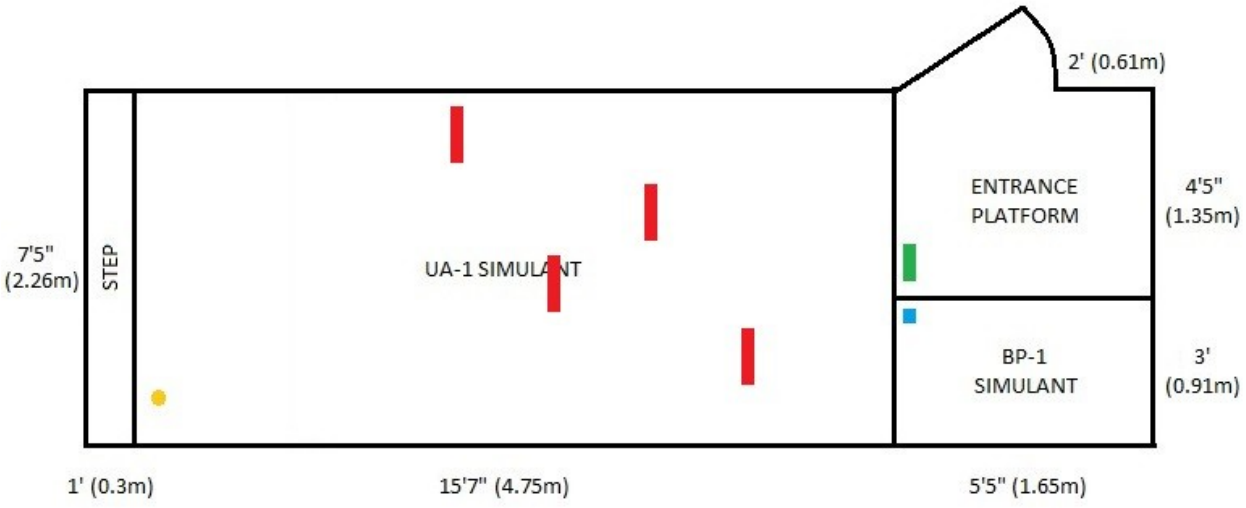


Figure 4.5. Test 4 Testbed Diagram with Kinect (Green), Lidar (Blue), Dust System (Yellow), & Targets (Red)

(a) (b) (c)

Figure 4.6. Test 4 Configuration: Saturated (a), Intermediate (b), and Reference (c) Dust Levels

Test 5 was conducted in the BP-1 simulant to determine if the depth sensors will have a similar response in this material as in the UA-1. A single 0.41 x 0.91 m target is placed at a distance of 1.07 m from the depth sensors. A diagram of this configuration is given in Figure 4.7, and Figure 4.8 is a picture of the test 5 setup.

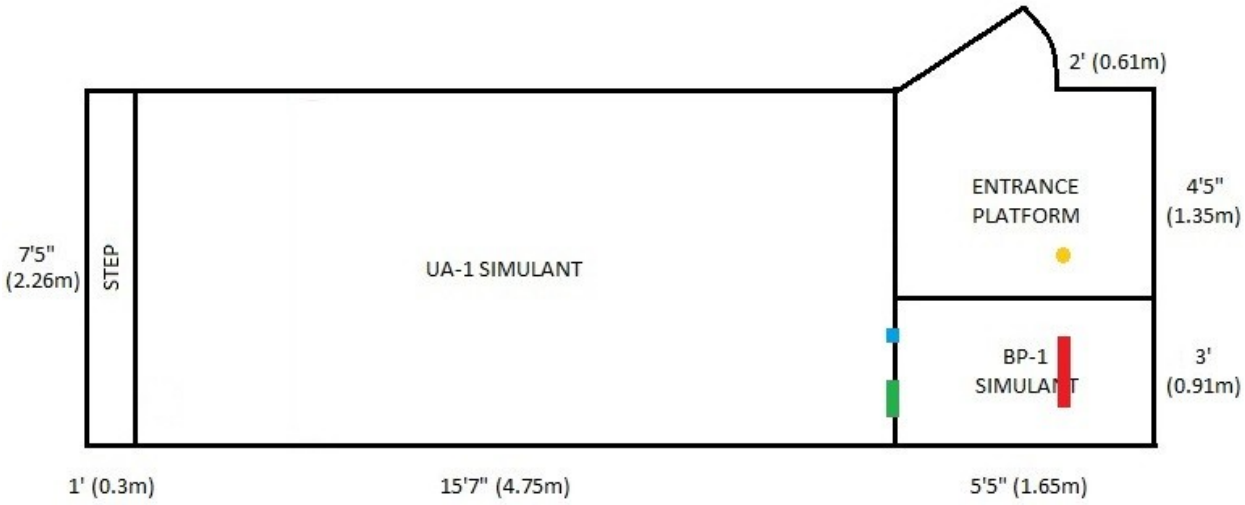


Figure 4.7. Test 5 Testbed Diagram with Kinect (Green), Lidar (Blue), Dust System (Yellow), & Target (Red)

Figure 4.8. Test 5 Configuration

4.4 Depth Data Collection Process

For all five tests, images were captured with both depth sensors with no suspended dust to serve as a reference case or baseline. The range of suspended dust for the baseline case was 0 – 0.22 mg/m^3 as measured with the dust sensor system. Once the baseline data was captured, the desired simulant was manually agitated until the testbed was completely saturated with suspended dust. Saturation was defined by a suspended dust reading of greater than 0.58 mg/m^3 from the dust sensor system. For visual reference, saturation is shown in Figure 4.6a. Another set of images was then captured by the distance sensors representing the saturated dust level. The dust sensor system continued to monitor the dust levels in the testbed. As the dust levels drop into the 0.22 – 0.58 mg/m^3 range, another set of images is obtained representing an intermediate dust level. For all tests, suspended dust levels are averaged for dust sensors 3 and 4 only. Dust sensors 1 and 2 are 6 ft off the ground or higher and do not measure the dust levels at the same elevation where the depth sensors and targets are positioned. Also, it is very difficult to saturate the testbed at heights of greater than 6 feet off the floor. Therefore, only the two lower dust sensors were used to measure suspended dust levels.

During all tests, depth data was collected by both the lidar and Kinect simultaneously so that dust conditions were as similar as possible for both sensors. To insure the two depth sensors were not interfering with the other's data capture, additional sets of images were captured with only one depth sensor operational at a time. These images were compared to the reference images taken by both sensors simultaneously and no significant differences were found in the data. Therefore, no interference is believed to be caused by operating the depth sensors simultaneously.

4.5 Post Processing Analysis

After all test data is gathered, it must be analyzed to extract the target plane distance values from the images. Before planes can be extracted, the lidar data must undergo one additional step in order to be processed in the same manner as the Kinect data. Although the lidar data is in the point cloud data format, it is scaled differently and has a different axis orientation than the Kinect data. A custom program called PCD_Converter was used to resolve this. The program is called with the argument format:

>PCD_Converter.exe [name of lidar pcd file to convert] [minimum distance in mm] [maximum distance in mm].

The lidar has a tendency under dusty conditions to report some pixel distances that are outside the possible range. The converter program uses the maximum distance limit to eliminate these extreme outliers in all three axes. The minimum distance limit is used similarly but only to eliminate points in the Z axis (in front of the sensor) with distances less than the specified value. This limit is often set to 0 m to remove all points behind the lidar or 0.5 m to filter out points that form a specific phenomenon dubbed "the dust bowl" which is caused by the dust and discussed

in Chapter 5. Occasionally in high dust, this minimum limit is set to a higher value in order to remove more of the dust interference between the lidar and the target. In all three cases, this pre-filtering does not alter the data of the target plane but simply allows the plane extraction algorithm to isolate the target in a timely fashion. Pre-filtering the data to increase the speed of target acquisition should be considered when designing an autonomous robot to perform navigation maneuvers in real-time in dusty environments. Once a pixel is determined to be within these limits, its values are reoriented about the origin. Table 4.1 shows the relationship of the two sets of axes used by the lidar and the Kinect. For example, a lidar pixel's Y value is negated and set as the X value for the Kinect orientation which corresponds to the axis running from sensor left to right in the image. In addition to orientation adjustments, PCD_Converter must also scale the lidar distance values. The Kinect outputs distances in meters while the lidar gives distances in millimeters. This causes the viewer program to interpret 1000 from the lidar as 1 km instead of 1 m. The lidar distance value is therefore divided by 1000 before finally being written to the output file, called Modified.pcd. The complete code for PCD_Converter is provided in Appendix E.

All depth images from the lidar and Kinect were analyzed using the same software: PCD_Analyzer. This is another custom program which uses the PCL to handle the point cloud depth data. It takes a single command line argument, which is the filename of the image to be

Table 4.1. Relation of Lidar Axes to Kinect Axes

Kinect	Lidar	Positive Direction
X	- Y	Right
Y	- Z	Down
Z	X	Forward

processed, and uses the RANSAC algorithm to extract planar objects. Once the file is loaded, the user is asked to input three parameters: the axis to which the desired plane is perpendicular, the distance threshold value, and the plane variance angle. A higher distance threshold will cause points from farther in front of the plane and behind it to be included in the returned data, allowing for a thicker target plane. The plane variance angle determines the number of degrees the plane angle is allowed to be offset from perpendicular to the axis. The algorithm attempts to find a plane matching the provided characteristics and then offers the user a menu with various command options. A screenshot of the command line window showing the menu options is given in Figure 4.9. The program also opens a cloud viewer window which is divided into three panes. This allows the user to see the original image in the left pane, the planar object that the algorithm has extracted in the center pane, and the remainder of the image without the extracted objects in the right pane (to ensure all object points were identified). An example of the cloud viewer is shown in Figure 4.10. In this example, the distance data shows the 1.22 x 2.13 m target at 2.59 m from the depth sensor (test 2), the floor (covered with UA-1 simulant), and the two walls of the testbed. The walls have already been analyzed and removed from the image in the right pane, and the floor plane is the object currently under analysis. The floor is isolated in the

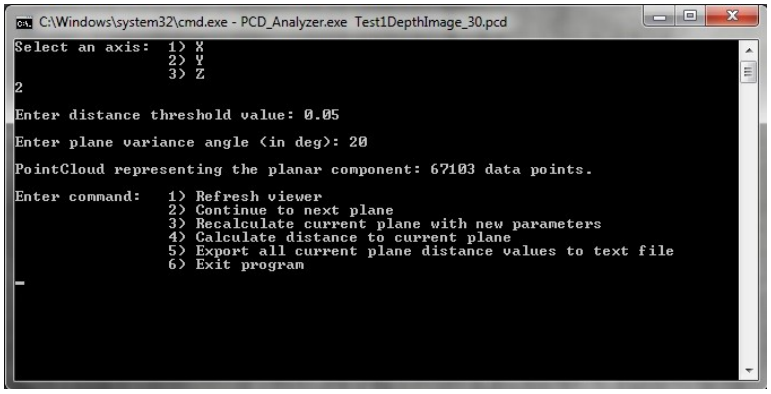


Figure 4.9. PCD_Analyzer Command Window

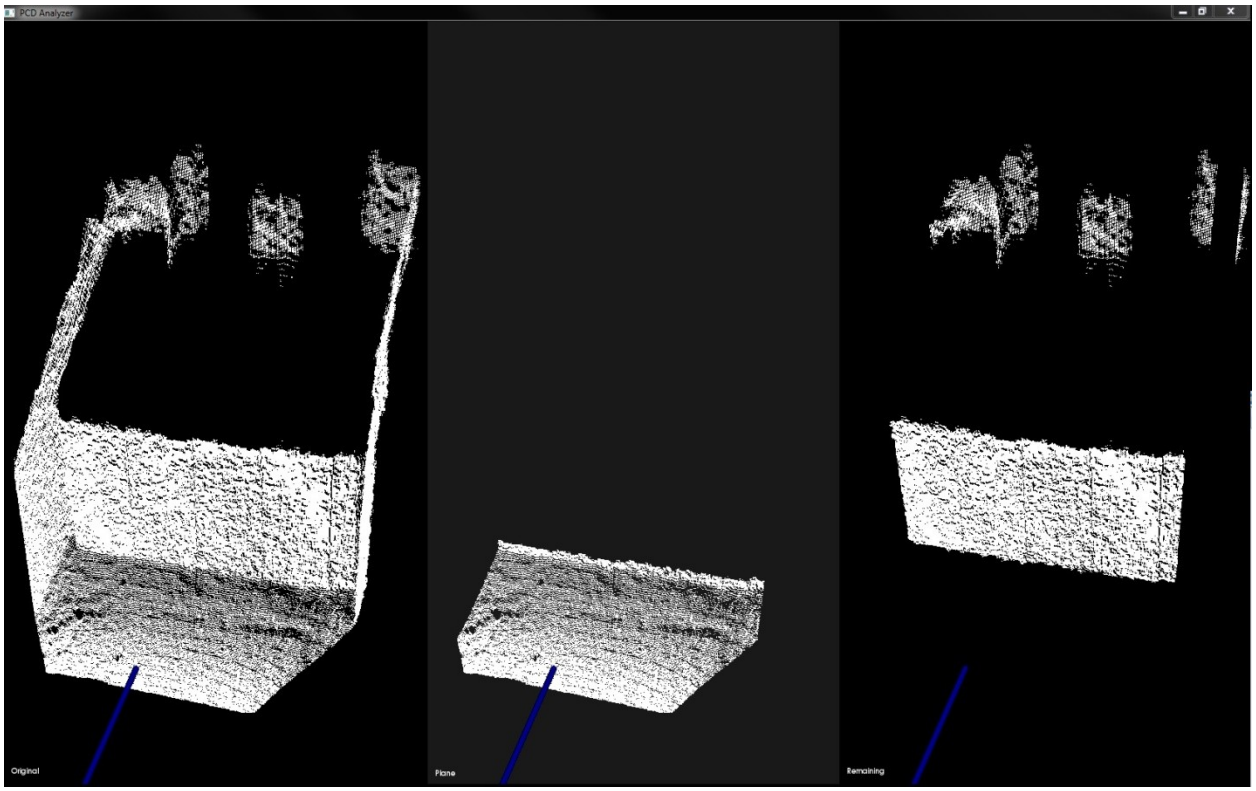


Figure 4.10. PCD_Analyzer Cloud Viewer Window

center pane and also removed from the dataset in the pane on the right. The walls and floor were routinely removed from the image first in order to assist the RANSAC algorithm in isolating the desired target plane. A similar multi-pass approach would need to be performed by a robot if using the RANSAC algorithm. Once the user has extracted the target, the distance value to each point in the plane can be exported to a text file to be averaged and graphed. The code for PCD_Analyzer is provided in Appendix F.

It takes 60 seconds for the lidar to complete its angular sweep of the test area to create its 3D view. During this same time, the Kinect captures 60 images. For each case, one image from the Kinect was chosen to compare with the lidar image. Kinect image number 30 is analyzed where possible due to it being the median image in the 60 second data capture window. When

image 30 produces no target plane, two other images are randomly chosen which contain at least a partial plane. In these cases, the data for both images is presented separately and the alternate image numbers used in the results are indicated in the data tables.

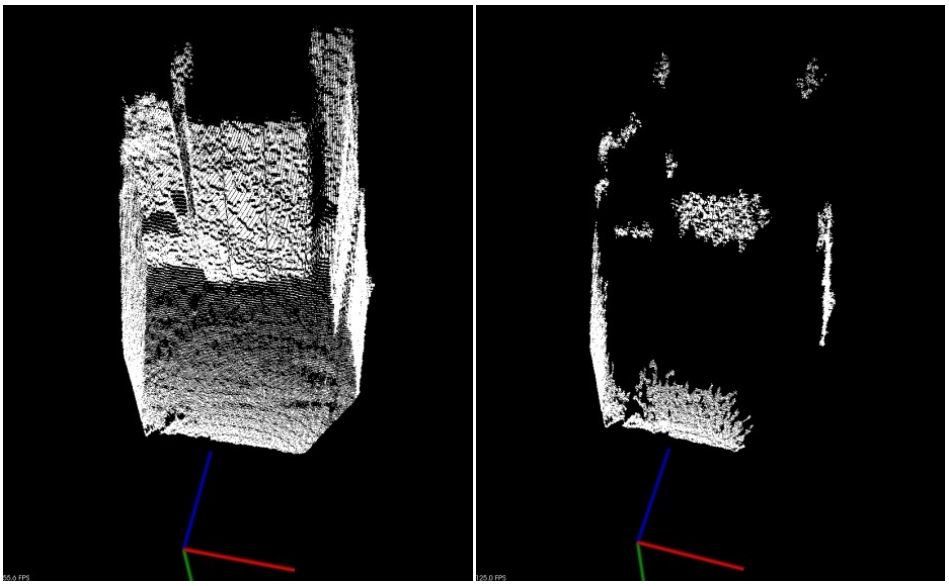
CHAPTER 5

ANALYSIS RESULTS

The results of the five testing scenarios are presented here along with observations about the performance of the sensors. In each case, the results are related to potential impacts for autonomous robotic navigation applications to keep the research in the proper context.

5.1 Test 1 Results

For test 1, a single large target is placed in the UA-1 area at a distance of 3.81 m from the sensors. Four images are presented: a reference image, an intermediate image, and two saturated images. Figure 5.1 provides the Kinect reference and saturation images for comparison. The axes markers are scaled to a length of 1 m, and Table 5.1 provides a legend for identifying which marker corresponds to each axis. A visual inspection reveals the dust does have an impact on the Kinect's ability to see the entire target at this distance. The saturated image at this distance is an example where image 30 did not contain the target plane; therefore images 5 and 10 were used instead.



(a) (b)

Figure 5.1. Kinect Images – Test 1:
Reference (a) and Saturated (b – Image 5)

Table 5.1. Axes Identification

Axis	Color	Positive Direction
X	Red	Right
Y	Green	Down
Z	Blue	Forward

Table 5.2 presents the data collected from each test image. The table format consists of ten columns which provide information on different parameters of the data. The first column is simply the name, which allows one to see which image corresponds to each dust level. The second column gives the number of valid points, or pixels, returned by the plane extraction algorithm as belonging to the target plane. The % of Ref column compares the number of valid

Table 5.2. Kinect Data – Test 1

Dust Level	Valid	% of Ref	Min (m)	Max (m)	Avg (m)	Error (mm)	Std Dev	Unique	Δ Unique (mm)
Reference	44236	N/A	3.621	4.296	3.918	N/A	0.127	16	45.00
Intermediate	43879	99.19	3.621	4.350	3.918	0.36	0.128	17	45.56
Saturated, Image 5	8442	19.08	3.698	4.142	3.898	19.95	0.073	11	44.40
Saturated, Image 10	5633	12.73	3.738	4.093	3.900	17.39	0.069	9	44.38

points from each image to the number of valid points in the reference image and presents the results in the form of a percentage. This indicates what percentage of the reference target each subsequent image can see. The Min, Max, and Avg columns provide the minimum, maximum, and average distance values returned by the valid points, respectively. The Error column is a calculation of the difference between the average distance of an image and the average distance of the reference image. The Std Dev column provides the standard deviation of all the distance values. A smaller deviation indicates the values are grouped more closely together forming a thinner plane. This is also used later in conjunction with the average value to plot the distribution of points. The Unique column provides the number of unique distance values returned in the planar points. A larger number of unique distance values could mean a larger gap between minimum and maximum planar limits, or it could indicate the sensor has a better resolution in the same range. The Δ Unique column provides the average difference between consecutive unique distance values. For the Kinect, this gives an indication of the average resolution between the given minimum and maximum distances. The lidar resolution is always 1 mm; therefore, a larger Δ Unique value here indicates larger gaps in consecutive distance values.

The data in Table 5.2 reveals that the saturated dust levels reduce the identified target area to only 13 – 19% of the target area captured in the reference case. Accurately identifying less than 20% of a landmark or obstacle would significantly limit the ability of an autonomous robot to correctly interpret its environment. However, saturated dust levels only skew the

average distance measurement by at most 19.95 mm, which is less than the Kinect's resolution of 45 mm at that range.

While tables can provide a great deal of information, some data is more easily interpreted with a graphical representation. Figure 5.2 provides a chart of the number of points, or pixels in the extracted plane, at each distance. This allows one to see how the points in the extracted plane are distributed, as well as the 4-5 cm gaps in consecutive distance values. However, this is not practical for a large number of unique values such as those encountered with the lidar. Also, the information for images with a small number of valid points (such as the saturated images in this figure) becomes hard to read among the larger values from the other images. Another method is needed to more efficiently convey and compare this information. Therefore, the average and standard deviation values are used to calculate the normal distribution curve for each image. In a normal distribution, the peak of each curve is centered at the average distance value, and all

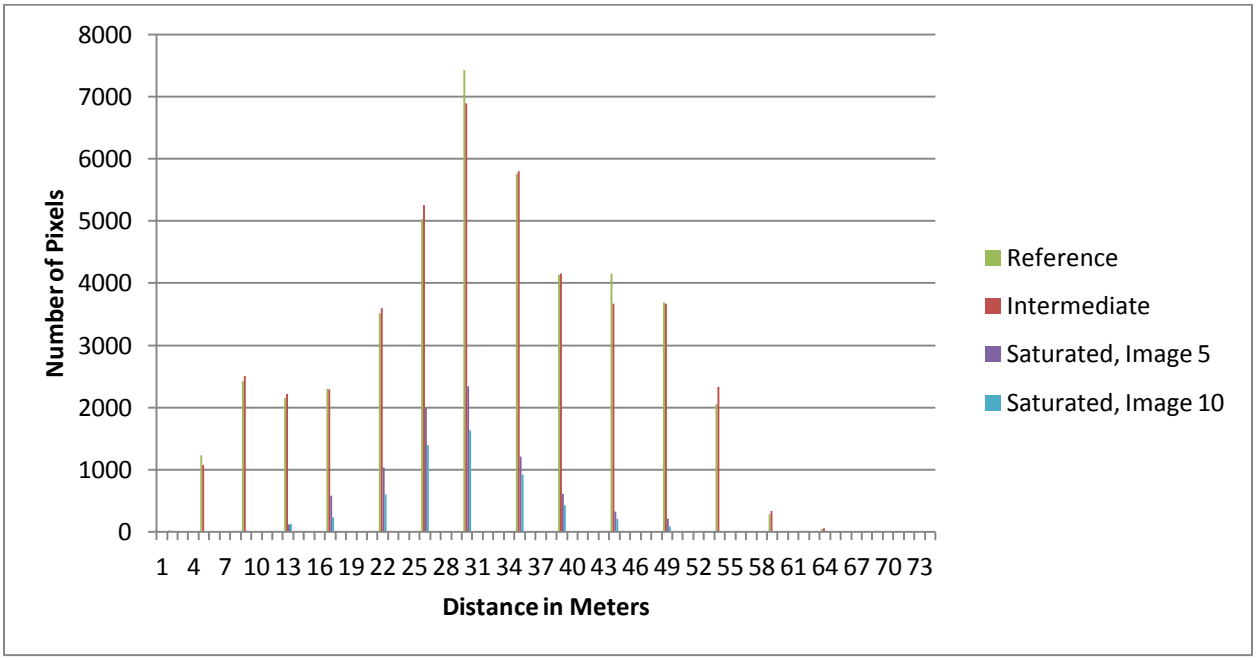


Figure 5.2. Kinect Number of Pixels vs. Distance – Test 1

distance values are plotted against their distribution coefficients. For example, a distance value with a distribution coefficient of three has approximately three times as many points at that distance than a distance with a coefficient of one. Figure 5.3 is the graph of these distributions for the Kinect data for test 1.

Figure 5.3 confirms that the normal distribution closely follows the distribution of points seen in Figure 5.2. However, it is clearly more evident that the saturated images have a similar average as the others, having a difference of only approximately 2 cm. This helps to visually reinforce the data from Table 5.2. The saturated images only returned 13 – 19% of the valid points returned by the reference image; hence the taller, thinner curve indicating the data is concentrated among a smaller range of values. In all tests, the image curves with dust should match the reference curve as closely as possible. Any deviation from the reference curve indicates data that has been affected by the dust, and it is easy to see in Figure 5.3 how closely the data from the intermediate image coincides with the reference image. This is the format in

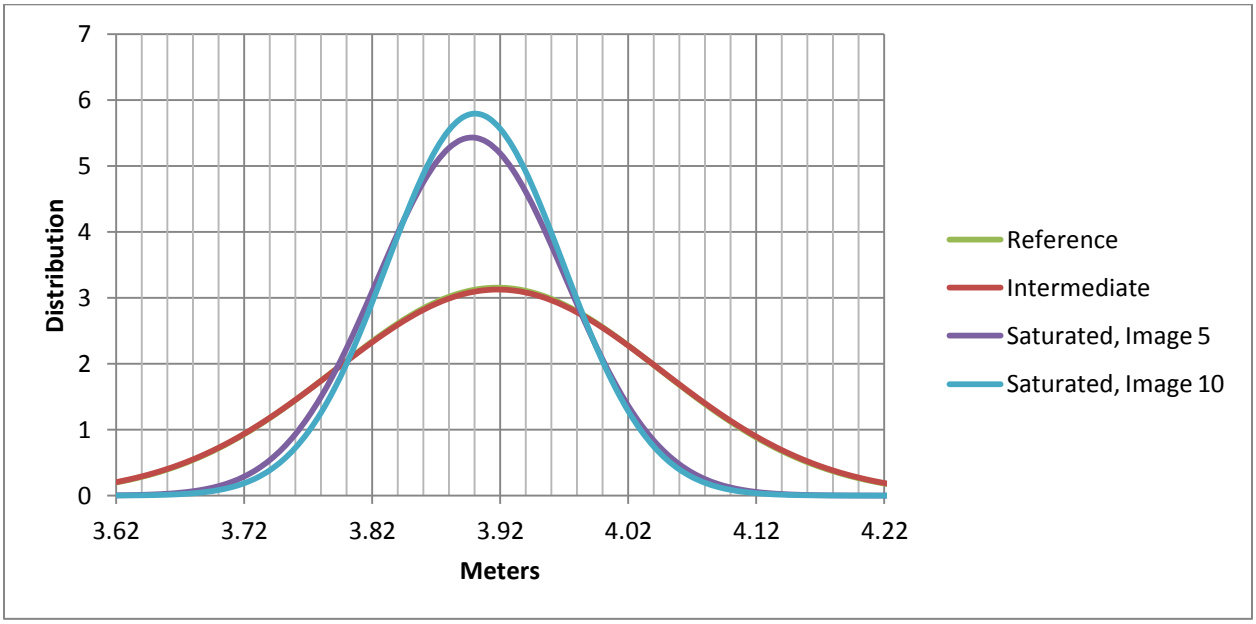
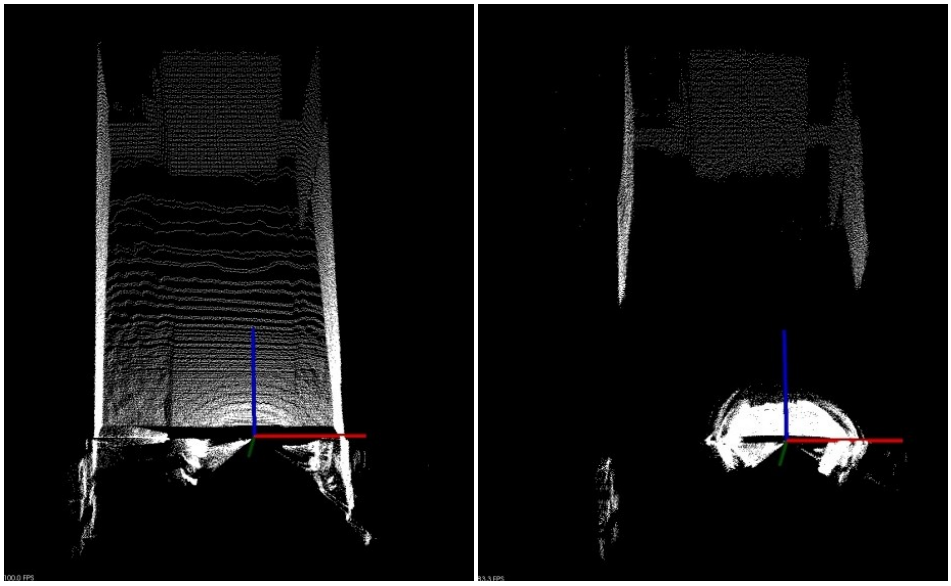


Figure 5.3. Kinect Distance Value Distributions – Test 1

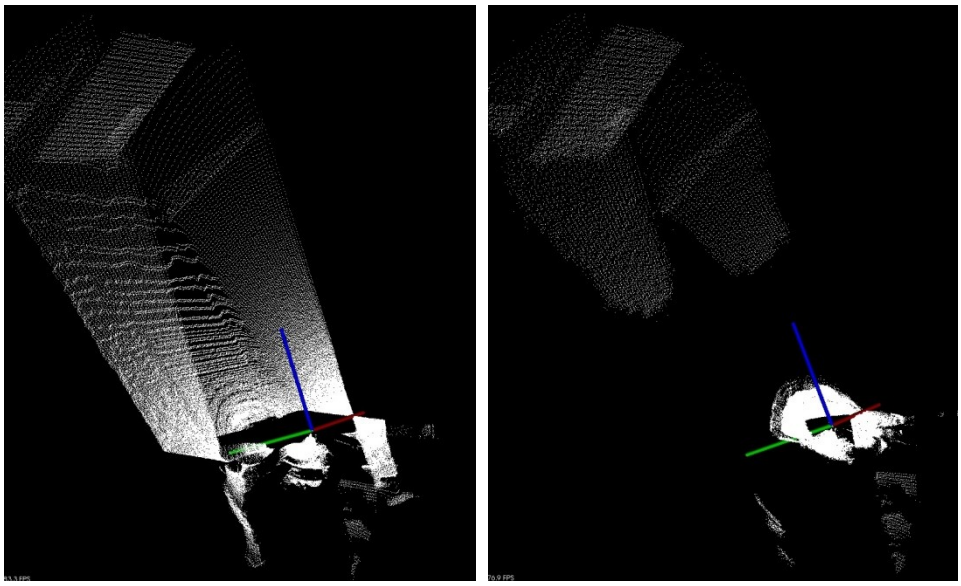
which all other test data are graphically presented.

The images for last echo lidar data from the reference and saturated images of test 1 are presented with views from above the lidar in Figure 5.4 and from the side of the lidar in Figure 5.5. The side view images reveal how the scanning planes converge at the X axis due to the rotation around that axis needed to obtain a 3D scan. One can also see the artifacts caused by the dust surrounding the lidar even in the reference images, and the dust bowl effect caused by the dust in the saturated images. This dust bowl appears to be a solid spherical object surrounding the lidar at a distance of approximately 0.5 m. These effects on the data cause interference with the RANSAC algorithm's ability to extract planar objects from the image in a timely manner. In order to be analyzed, these artifacts should be filtered out using the minimum distance input to the conversion program. Figure 5.6 shows the same images as Figure 5.4 after removing all points with distances less than 0.5 m from the lidar.



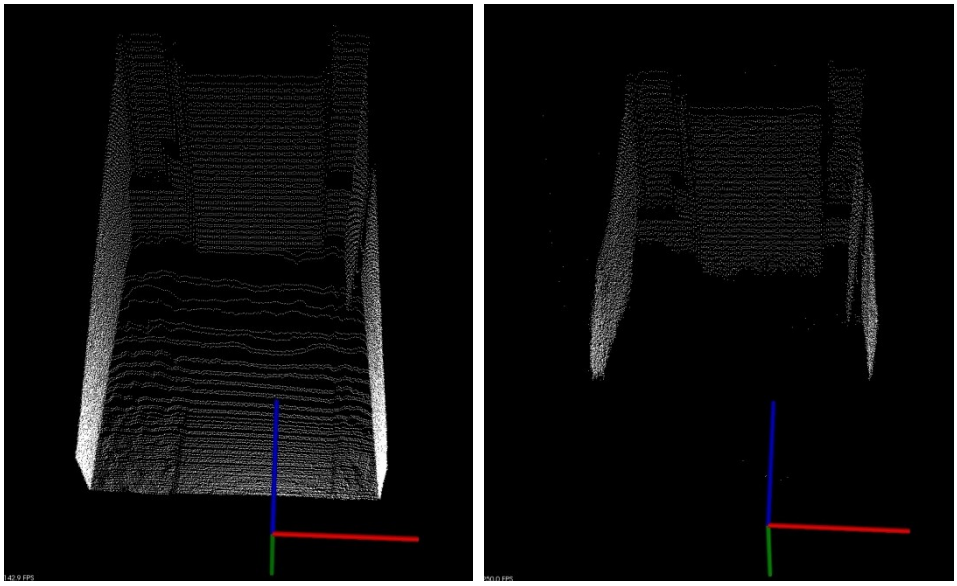
(a) (b)

Figure 5.4. Last Echo Lidar Full Images – Test 1:
Reference (a) and Saturated (b)



(a) (b)

Figure 5.5. Last Echo Lidar Full Images – Test 1 From Side:
Reference (a) and Saturated (b)



(a) (b)

Figure 5.6. Last Echo Lidar Filtered Images – Test 1:
Reference (a) and Saturated (b)

It is reasonable to believe that these artifacts would have a similar impact on obstacle avoidance and navigation algorithms as well. A close up view of the dust bowl effect from above the sensor is shown in Figure 5.7. This shows how an obstacle avoidance algorithm could mistake the dust for an obstacle that encompasses the sensor's entire 270° FOV. In a robotic navigation application, an obstacle avoidance algorithm would have to stop or redirect the robot based upon this data. However, the last echo of the multi-echo lidar is capable of seeing through the dust cloud as shown in Figures 5.4-5.6 where data points are returned well beyond the dust bowl. Once the interference is removed from each test image, the target plane is extracted and the results are given in Table 5.3.

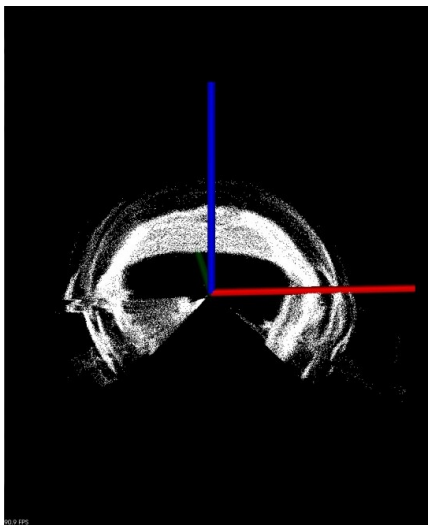


Figure 5.7. Dust Bowl From Above Lidar – Test 1

Table 5.3. Last Echo Lidar Data – Test 1

Dust Level	Valid	% of Ref	Min (m)	Max (m)	Avg (m)	Error (mm)	Std Dev	Unique	Δ Unique (mm)
Reference	3792	N/A	3.366	4.180	3.709	N/A	0.201	721	1.13
Intermediate	3836	101.16	3.272	4.180	3.705	3.97	0.204	752	1.21
Saturated	3710	97.84	3.337	4.096	3.719	10.16	0.193	716	1.06

The data in Table 5.3 indicates that the last echo of the lidar is not as susceptible to dust interference as the Kinect at this distance. The lidar is still able to see 98% of the target at the saturated dust level. While the 10 mm error is greater than the stated resolution of the lidar, it is still half that of the Kinect for the same distance. Figure 5.8 graphs the normal distribution for each test image. This shows that the data from both dust images corresponds closely with the reference image, with the saturated image slightly outside the curve shared by the other images.

The first echo of the lidar could not see through the dust nearly as well. Figure 5.9 gives images of the first echo data for the reference and saturated tests. The reference image was unable to see the target even in the negligible dust of the reference level. This implies that a single-echo lidar would be unable to see a target at this distance even in small amounts of dust. The saturated image reveals nothing outside the dust bowl interference.

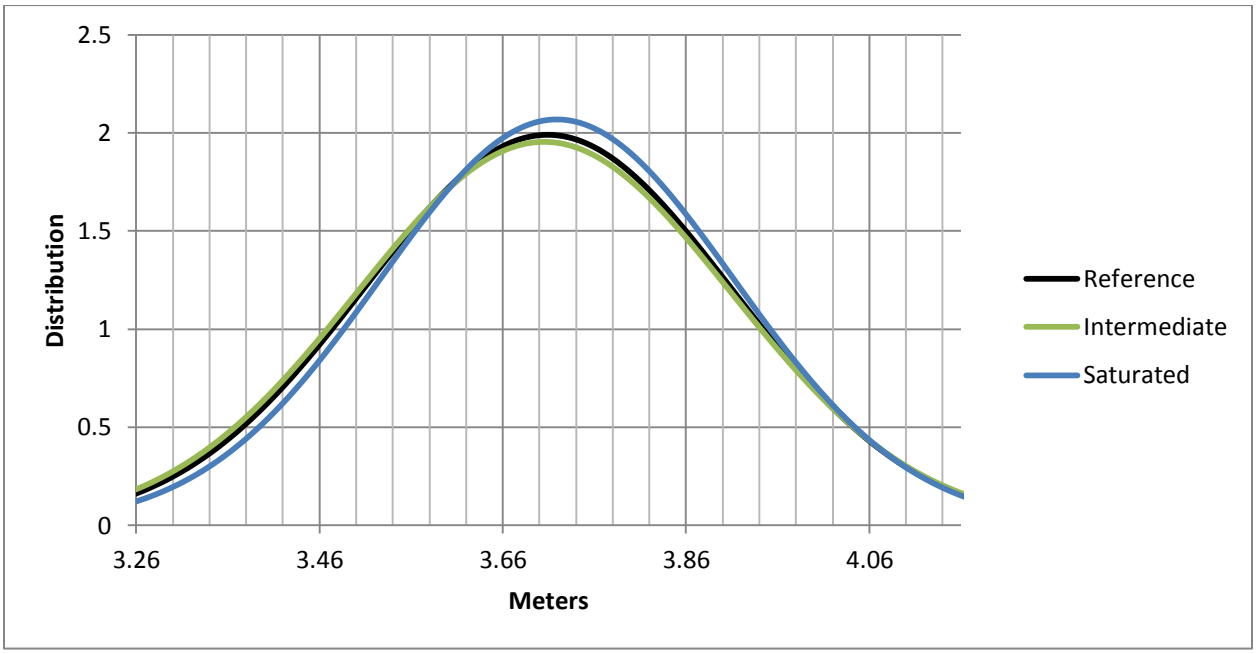
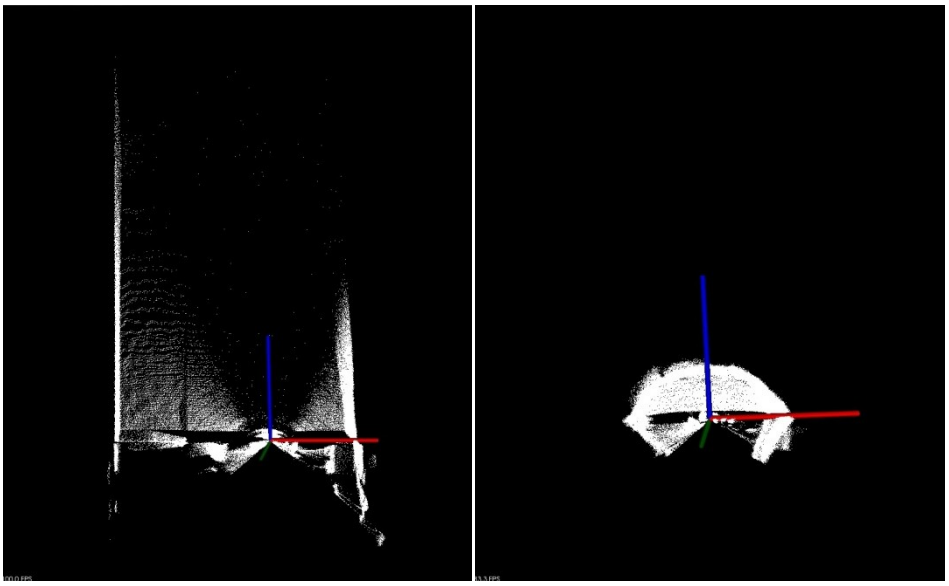


Figure 5.8. Last Echo Lidar Distance Value Distributions – Test 1



(a) (b)

Figure 5.9. First Echo Lidar Full Images – Test 1:
Reference (a) and Saturated (b)

The data from test 1 suggests that at a distance of 3.81 m the dust has an effect on both sensors, with a greater impact on the Kinect. The lidar was able to detect 98% of the target even in the saturated image indicating that it would be much more valuable for robotic navigation under these conditions than the Kinect. Also, a multi-echo lidar is needed to see a target if any small amount of dust is present.

5.2 Test 2 Results

In test 2, a single large target is placed at a distance of 2.59 m from the sensors. Three tests are again presented: a reference image, and images with saturated and intermediate dust levels. The Kinect's image number 30 of the saturated test has no target, so images 50 and 60 are used. The results from all images are presented in Table 5.4. Test 2 represents another example of the Kinect returning only a small portion of the target, at most 35.23% in saturated conditions.

Table 5.4. Kinect Data – Test 2

Dust Level	Valid	% of Ref	Min (m)	Max (m)	Avg (m)	Error (mm)	Std Dev	Unique	Δ Unique (mm)
Reference	100342	N/A	2.496	2.649	2.569	N/A	0.022	9	19.13
Intermediate	96924	96.59	2.478	2.629	2.567	1.78	0.016	9	18.88
Saturated, Image 50	13517	13.47	2.514	2.609	2.557	11.65	0.019	6	19.00
Saturated, Image 60	35346	35.23	2.460	2.649	2.559	10.23	0.021	11	18.90

However, the error here has been cut in half compared to that of test 1. The Δ unique distance values have also been reduced by half due to the non-linearity of the Kinect's resolution mentioned earlier. Therefore, the error is again approximately half the resolution at this distance. The graphical data is given in Figure 5.10. This may not look as cohesive as the Kinect distribution for test 1 shown in Figure 5.3, but the curves are much taller and the major gridlines occur every two centimeters as opposed to every ten centimeters. While test 2 images share a similar number of unique distance measurements, the valid point values are not distributed in the

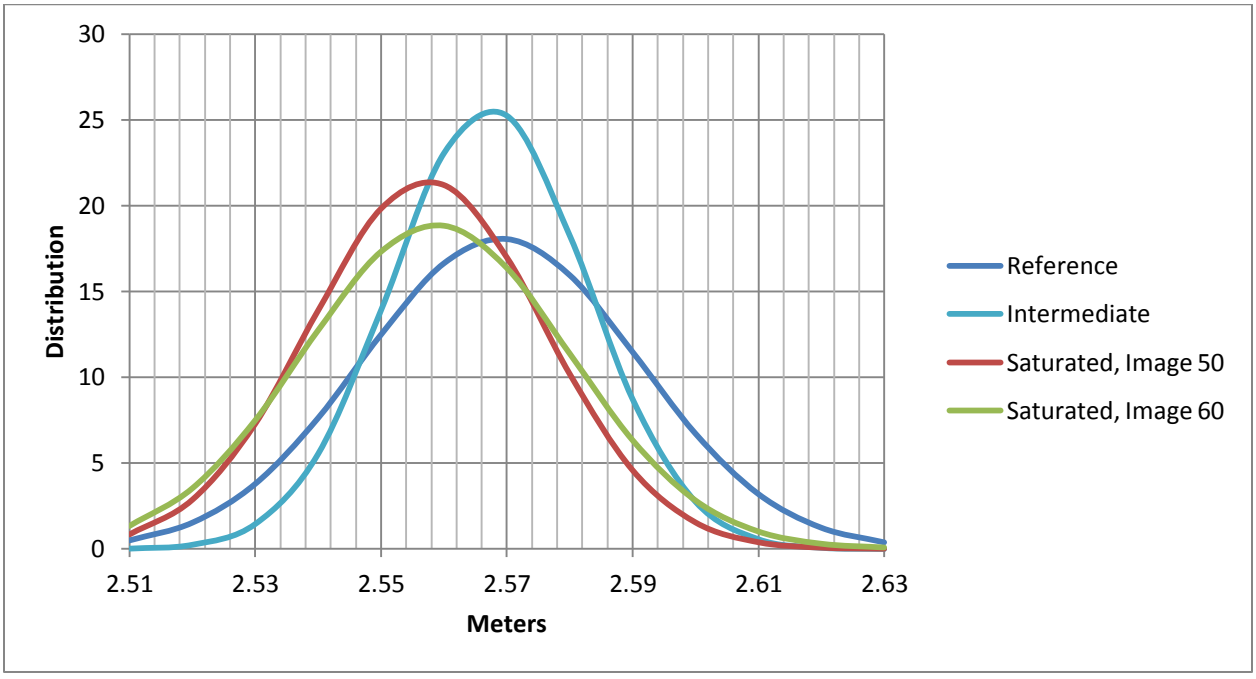


Figure 5.10. Kinect Distance Value Distributions – Test 2

same manner. This is what causes the disparity in the curves. All the data suggests that the Kinect is again affected by the dust in that it obscures part of the target and causes a small error. The error rate at this distance is actually better because the resolution of the Kinect is improved with the shorter distance to target.

The data for the last echo lidar images is given in Table 5.5. This time the saturated image shows a drop in target percentage similar to the Kinect, but with an extremely high error, approximately five times higher than the stated accuracy of the lidar. Even at intermediate dust levels, the lidar error is greater than the Kinect error at saturated dust levels for this distance. This is confirmed graphically in Figure 5.11. One can see how the average jumps significantly for the saturated dust image and moves back toward the reference value in the intermediate image. This indicates that although the lidar can produce a partial target at this distance during saturated dust levels, the returned distance values should not be trusted due to the very high error rate. In an autonomous robotics navigation application, this uncertainty will create difficulty for feature extraction, obstacle avoidance, and short range path planning.

Table 5.5. Last Echo Lidar Data – Test 2

Dust Level	Valid	% of Ref	Min (m)	Max (m)	Avg (m)	Error (mm)	Std Dev	Unique	Δ Unique (mm)
Reference	8892	N/A	2.237	2.787	2.470	N/A	0.113	461	1.20
Intermediate	9370	105.38	2.220	2.813	2.488	18.38	0.116	501	1.19
Saturated	3420	38.46	2.416	2.910	2.626	156.96	0.068	329	1.51

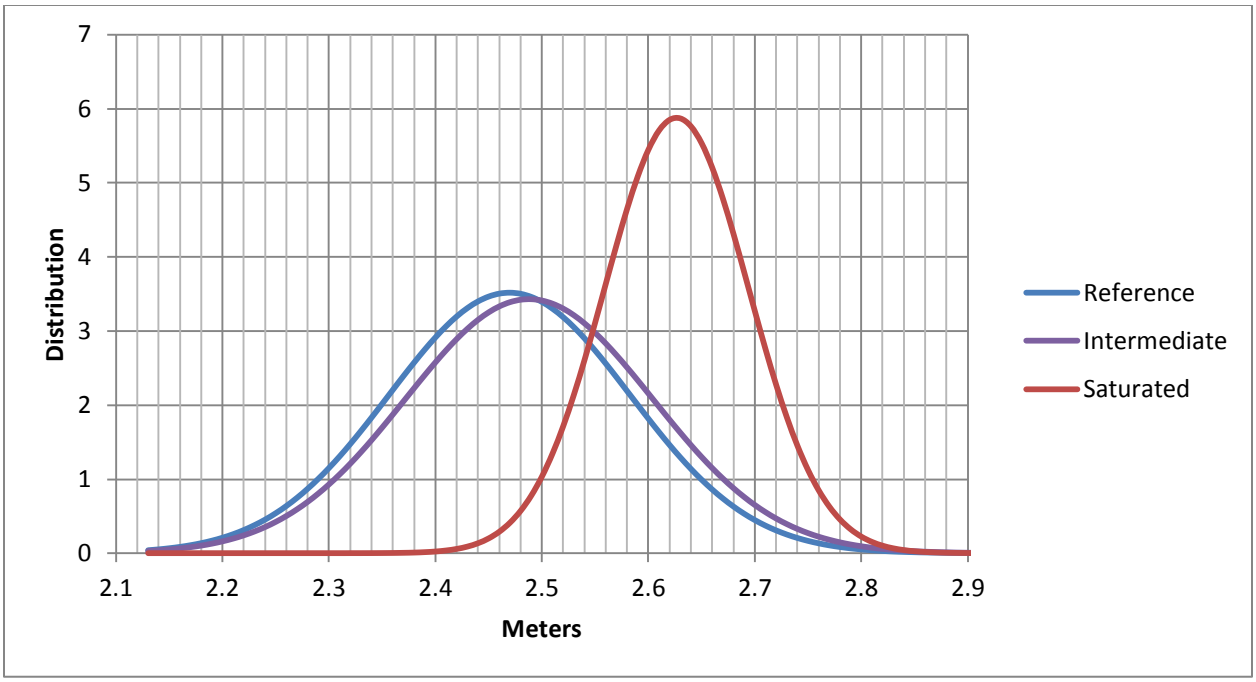


Figure 5.11. Last Echo Lidar Distance Value Distributions – Test 2

For test 2, the first echo lidar data was not very useful. First echo return was able to capture a partial target only during the reference case. The data for this is included in Table 5.6. There are few valid points with an average distance that is 29 mm less than the last echo data in the same test.

Results from test 2 indicate that although the lidar still identifies more of the target at 2.59 m than the Kinect, the results from the Kinect are more reliable in dust than those of the lidar. The first echo lidar data was not useful and would not aid autonomous navigation in any way in a dusty scenario.

Table 5.6. First Echo Lidar Data – Test 2

Dust Level	Valid	% of Ref	Min (m)	Max (m)	Avg (m)	Error (mm)	Std Dev	Unique	Δ Unique (mm)
Reference	204	N/A	2.227	2.689	2.441	N/A	0.121	166	2.80
Intermediate	None								
Saturated	None								

5.3 Test 3 Results

This test uses the same large target in the UA-1 area at a distance of 1.07 m from the sensors. The resulting Kinect image data is given in Table 5.7 and Figure 5.12. One immediately notices the uniformity of the data. There is no loss of target as in previous scenarios, and the min, max, average, and standard deviation values are almost identical. The error for all images is less than 1 mm, and the difference in consecutive distance values has decreased by almost a factor of six. The distribution curves in Figure 5.12 confirm that these datasets are closely aligned. The

Table 5.7. Kinect Data – Test 3

Dust Level	Valid	% of Ref	Min (m)	Max (m)	Avg (m)	Error (mm)	Std Dev	Unique	Δ Unique (mm)
Reference	205651	N/A	1.064	1.105	1.079	N/A	0.005	13	3.417
Intermediate	205079	99.72	1.064	1.102	1.079	0.50	0.005	12	3.455
Saturated	205836	100.09	1.061	1.102	1.079	0.57	0.005	13	3.417

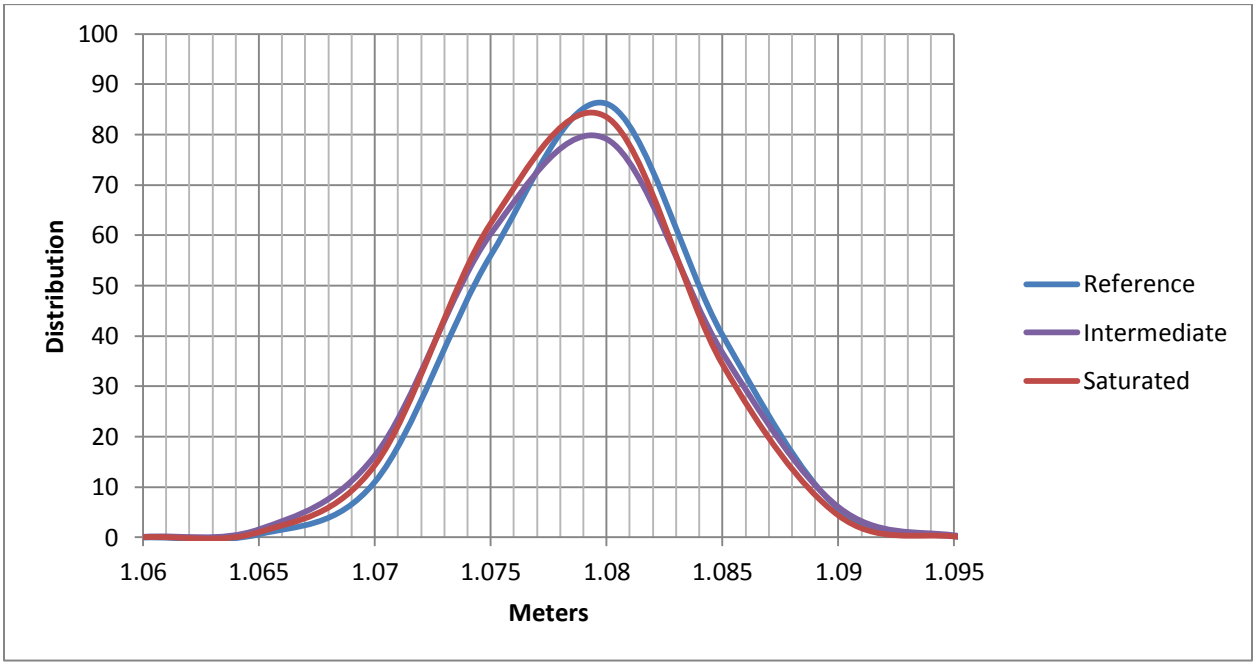


Figure 5.12. Kinect Distance Value Distributions – Test 3

curves also have very tall peaks which indicate a small deviation and a concurrence within the distance values. All this indicates that the Kinect data is very reliable in a range of suspended dust levels at a distance of 1.07 m.

The last echo results of the lidar are given in Table 5.8 and Figure 5.13. The lidar returns no values for the target in the saturated case. About 25% of the target is returned for the intermediate case, but this comes with another large error value. For the intermediate dust case, the target appears to be approximately 94 mm closer than it actually is, which is nearly twice the

Table 5.8. Last Echo Lidar Data – Test 3

Name	Valid	% of Ref	Min (m)	Max (m)	Avg (m)	Error (mm)	Std Dev	Unique	Δ Unique (mm)
Reference	28831	N/A	0.944	1.158	1.055	N/A	0.027	212	1.01
Intermediate	7234	25.09	0.773	1.069	0.960	94.76	0.039	228	1.30
Saturated	None								

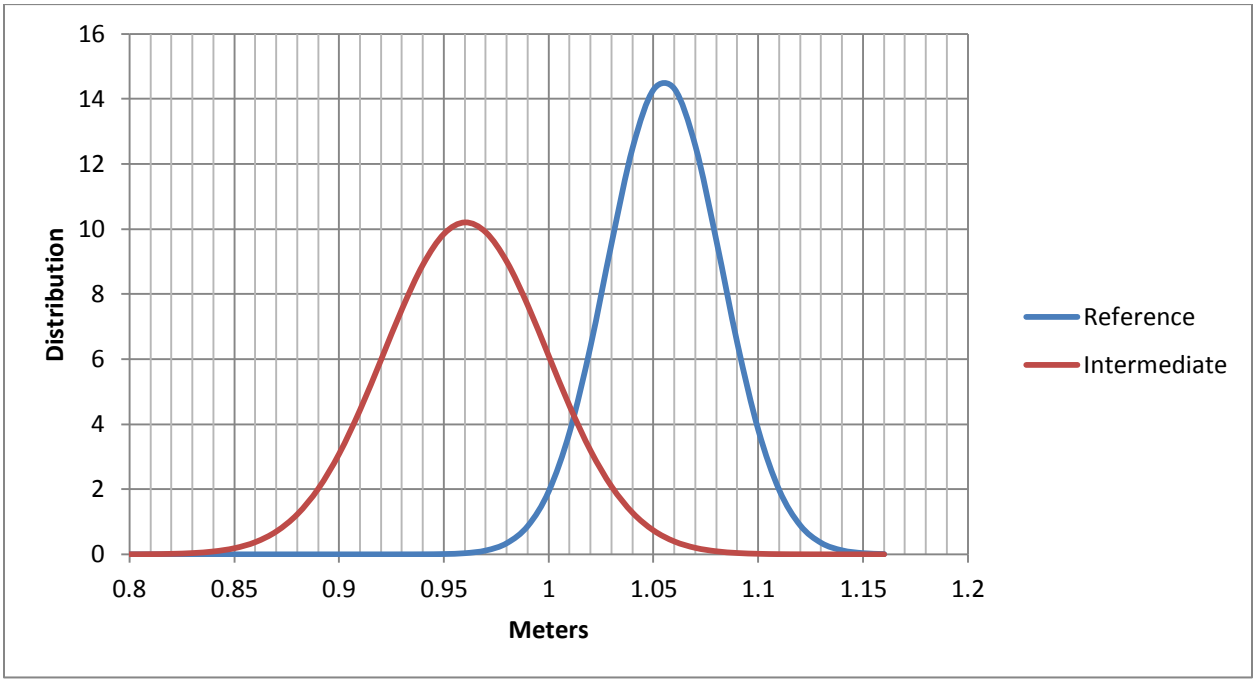


Figure 5.13. Last Echo Lidar Distance Value Distributions – Test 3

stated accuracy of the lidar. This data would be difficult to use in an autonomous navigation application due to the small percentage of the target recognized and the high degree of uncertainty with the data. (It should be noted that in Figure 5.11, the lidar showed the target further away than it really was. But, in Figure 5.13, the target appears closer than it should be. Therefore, the error introduced by the dust in the lidar data is non-linear.)

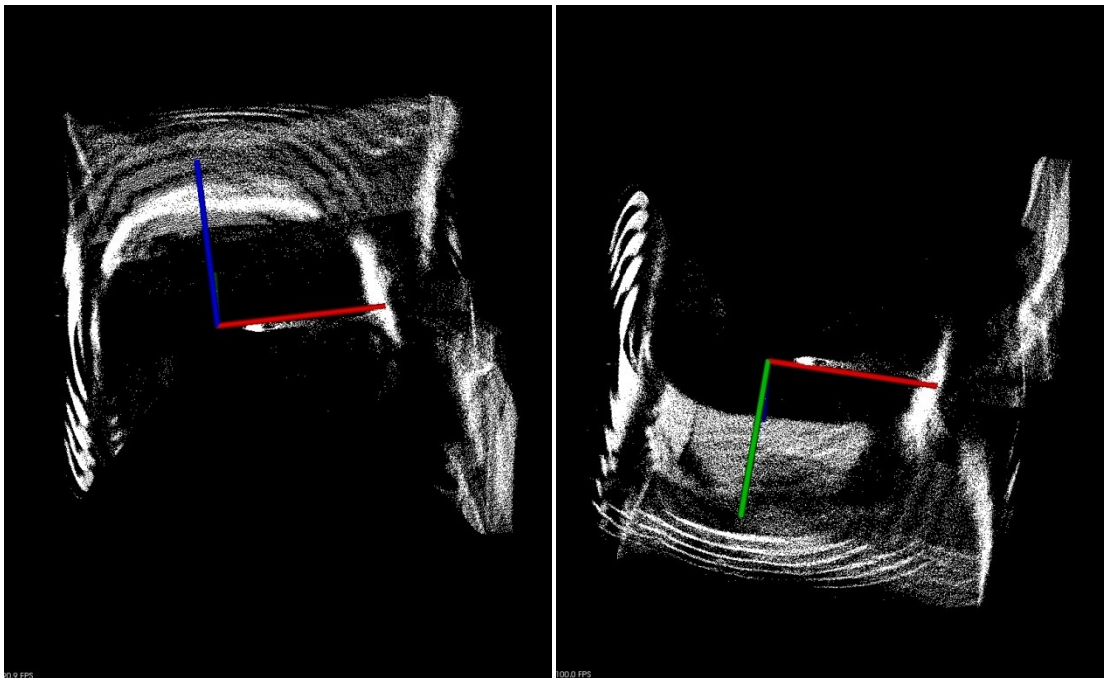
The first echo of the lidar only returns valid points for the reference dust level. These results are given in Table 5.9. This data provides no more information than the last echo; although the min, max, and average distance values are similar to the last echo. This could simply indicate that for many of the valid points the first echo was the only echo returned, and thus the same data as presented by the last echo results.

The results from test 3 show that the Kinect data is stable in dust at 1.07 m, and the lidar data is less reliable at higher levels of suspended dust. Combined with tests 1 and 2, this would indicate a trend where the Kinect seems to perform better as the distance to the target decreases, and the lidar seems to perform worse as the distance decreases. There is no doubt, however, that at these shorter distances to target, the lidar would not provide sufficient data to support autonomous navigation or obstacle avoidance.

Table 5.9. First Echo Lidar Data – Test 3

Dust Level	Valid	% of Ref	Min (m)	Max (m)	Avg (m)	Error (mm)	Std Dev	Unique	Δ Unique (mm)
Reference	26138	N/A	0.946	1.154	1.055	N/A	0.028	208	1.01
Intermediate	None								
Saturated	None								

Figure 5.14 illustrates another interesting lidar artifact caused by the dust. The image is taken from the last echo data of the intermediate dust image. When the dust settles and the dust bowl effect begins to spread back out into the image of the surrounding room, it creates a halo effect in the image. This causes all surfaces around the sensor to appear comprised of concentric rings or spheres centered at the lidar. These rings form a stair step pattern in the surfaces which distorts them and their distance measurements. One can see the target in the top of Figure 5.14a and how the center of it appears bowed in toward the lidar and part of it appears to have this halo effect. This could be a major contributor to the error present in the target measurements at these closer distances. The testbed floor and walls also exhibit the halo effect in Figure 5.14b. The surfaces at farther distances do not present this artifact since they are not affected by the dust bowl effect.

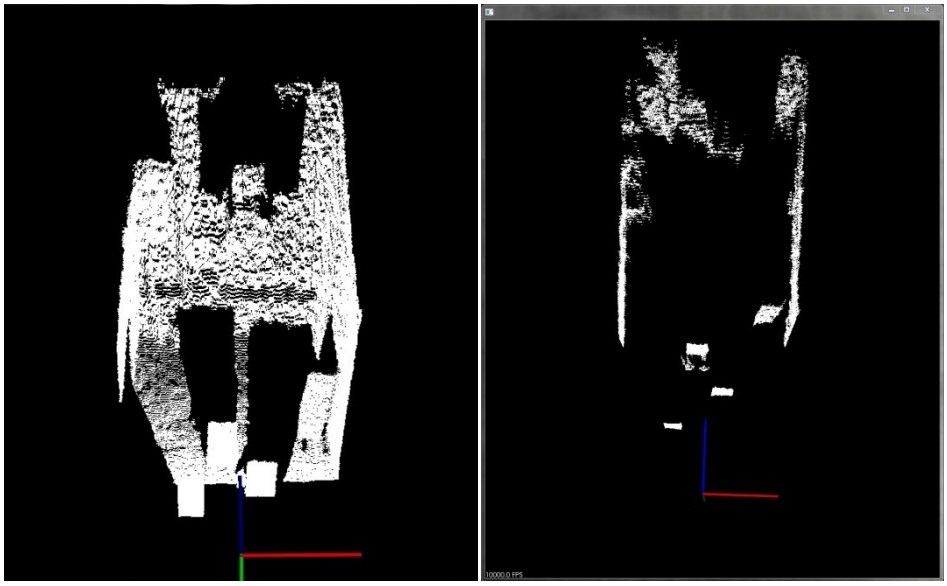


(a) (b)

Figure 5.14. Halo Effect From Above (a) And Below (b) the Lidar

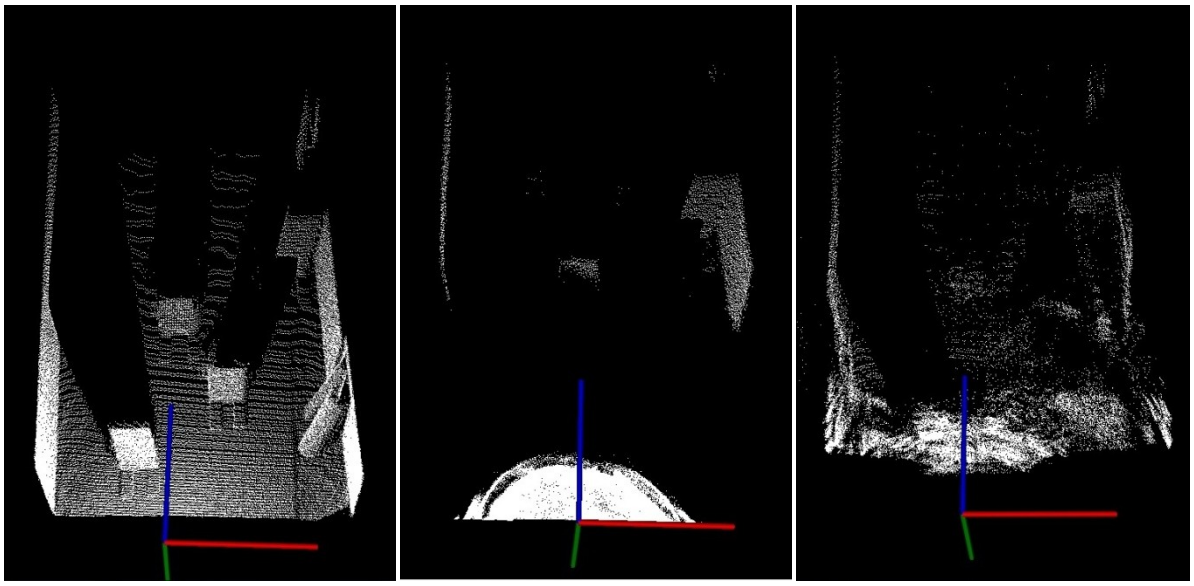
5.4 Test 4 Results

Test 4 consists of four smaller targets in the UA-1 area which are designed to mimic small obstacles. Figure 5.15 provides the reference and saturated images from the Kinect. The reference image is taken from behind the Kinect, and one can see the shadows of the targets on the floor. The saturated image is taken from above the Kinect. Even though the floor is missing, the four targets are still clearly visible. Figure 5.16 gives images of the last echo lidar data for three different dust levels. Figure 5.16b shows that the dust bowl caused by the saturated dust levels completely blocks the sensor's ability to see the first two targets and only allows for a partial third target. The warped walls in Figure 5.16c illustrate how much interference is still present in the image when the dust bowl effect begins to expand into the halo effect. This concurs with earlier tests that revealed missing or distorted targets in the lidar images at close range.



(a) (b)

Figure 5.15. Kinect Images – Test 4:
Reference (a) and Saturated (b)



(a) (b) (c)

Figure 5.16. Last Echo Lidar Images – Test 4:
Reference (a), Saturated (b), and Intermediate (c)

The results for each dataset from the first (closest) target, located at a distance of 0.91 m, are given in Tables 5.10-5.12. This data compares favorably with that collected in previous tests using the larger target. Again, the Kinect shows stable performance at this distance across various dust levels while the lidar only produces results for the two lowest dust levels. The lidar also generates larger errors than the Kinect, which reinforces the theory that the Kinect performs better at short distances. The distribution graphs for the three datasets are shown in Figures 5.17-5.19.

Table 5.10. Kinect Data – Test 4 – 0.91 m Target

Dust Level	Valid	% of Ref	Min (m)	Max (m)	Avg (m)	Error (mm)	Std Dev	Unique	Δ Unique (mm)
Reference	36721	N/A	0.917	0.966	0.927	N/A	0.006	20	2.58
Intermediate	36416	99.17	0.910	0.958	0.924	3.41	0.007	20	2.53
Saturated	36383	99.08	0.903	0.966	0.925	2.58	0.006	25	2.63

Table 5.11. Last Echo Lidar Data – Test 4 – 0.91 m Target

Dust Level	Valid	% of Ref	Min (m)	Max (m)	Avg (m)	Error (mm)	Std Dev	Unique	Δ Unique (mm)
Reference	5949	N/A	0.803	1.111	0.894	N/A	0.080	213	1.45
Intermediate	6057	101.82	0.791	1.145	0.886	7.77	0.095	224	1.59
Saturated	None								

Table 5.12. First Echo Lidar Data – Test 4 – 0.91 m Target

Dust Level	Valid	% of Ref	Min (m)	Max (m)	Avg (m)	Error (mm)	Std Dev	Unique	Δ Unique (mm)
Reference	4987	N/A	0.810	0.912	0.858	N/A	0.012	93	1.11
Intermediate	4724	94.73	0.792	0.909	0.842	16.28	0.014	106	1.11
Saturated	None								

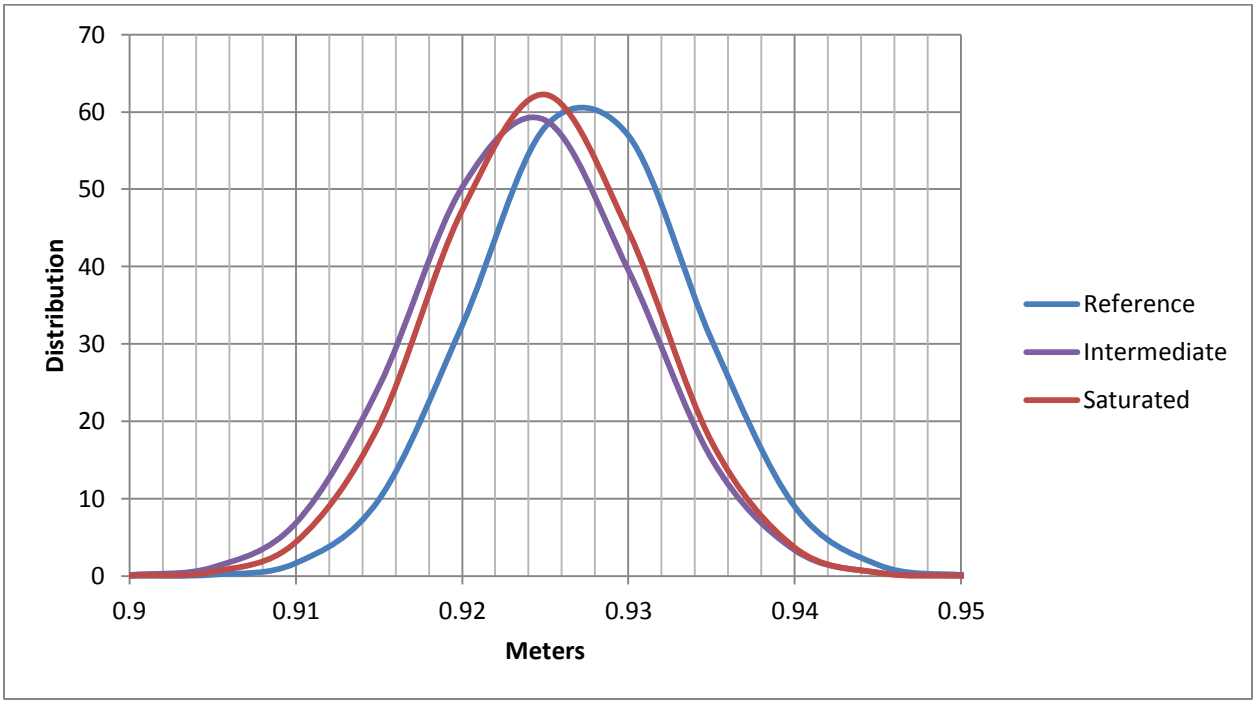


Figure 5.17. Kinect Distance Value Distributions – Test 4 – 0.91 m Target

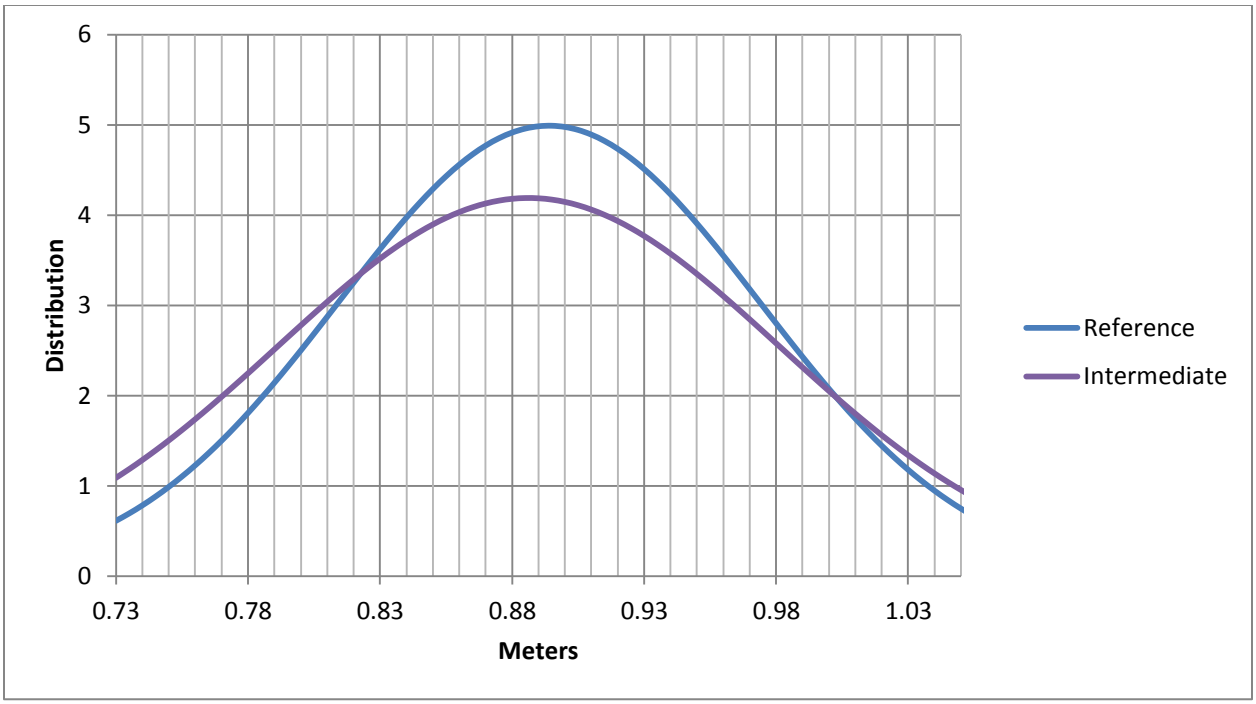


Figure 5.18. Last Echo Lidar Distance Value Distributions – Test 4 – 0.91 m Target

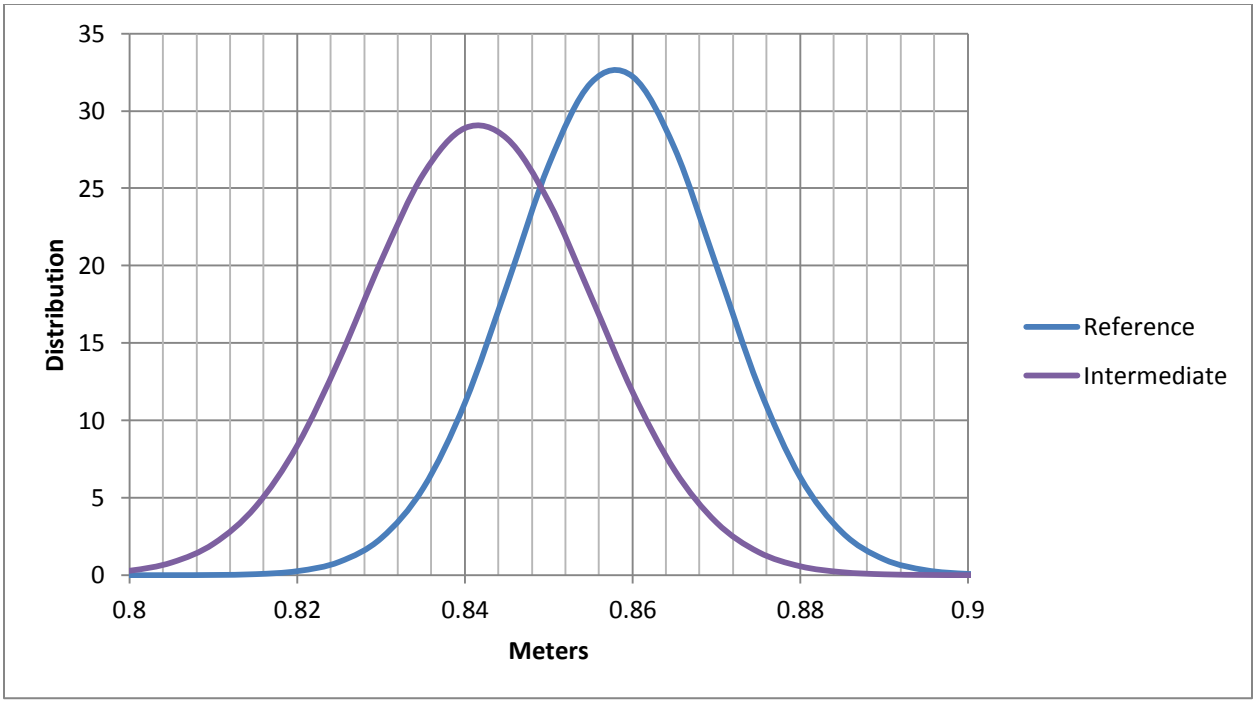


Figure 5.19. First Echo Lidar Distance Value Distributions – Test 4 – 0.91 m Target

The results from the second target, located at 1.52 m, are given in Tables 5.13-5.15. Again, the Kinect has larger Δ unique values but good performance with minimal error while the lidar is unable to see the target in the saturated case. The distributions for the Kinect data are presented in Figure 5.20, and show the slight change in readings only during the saturated case. The last echo and first echo lidar distributions are given in Figure 5.21 and Figure 5.22.

Table 5.13. Kinect Data – Test 4 – 1.52 m Target

Dust Level	Valid	% of Ref	Min (m)	Max (m)	Avg (m)	Error (mm)	Std Dev	Unique	Δ Unique (mm)
Reference	15563	N/A	1.437	1.519	1.473	N/A	0.006	14	6.31
Intermediate	15497	99.58	1.437	1.525	1.474	0.17	0.006	15	6.29
Saturated	15440	99.21	1.443	1.512	1.472	1.47	0.006	12	6.27

Table 5.14. Last Echo Lidar Data – Test 4 – 1.52 m Target

Dust Level	Valid	% of Ref	Min (m)	Max (m)	Avg (m)	Error (mm)	Std Dev	Unique	Δ Unique (mm)
Reference	1821	N/A	1.312	1.560	1.443	N/A	0.042	198	1.26
Intermediate	1822	100.05	1.312	1.651	1.450	6.93	0.042	179	1.90
Saturated	None								

Table 5.15. First Echo Lidar Data – Test 4 – 1.52 m Target

Dust Level	Valid	% of Ref	Min (m)	Max (m)	Avg (m)	Error (mm)	Std Dev	Unique	Δ Unique (mm)
Reference	1352	N/A	1.400	1.521	1.462	N/A	0.019	117	1.04
Intermediate	1185	87.65	1.363	1.533	1.450	11.64	0.037	151	1.13
Saturated	None								

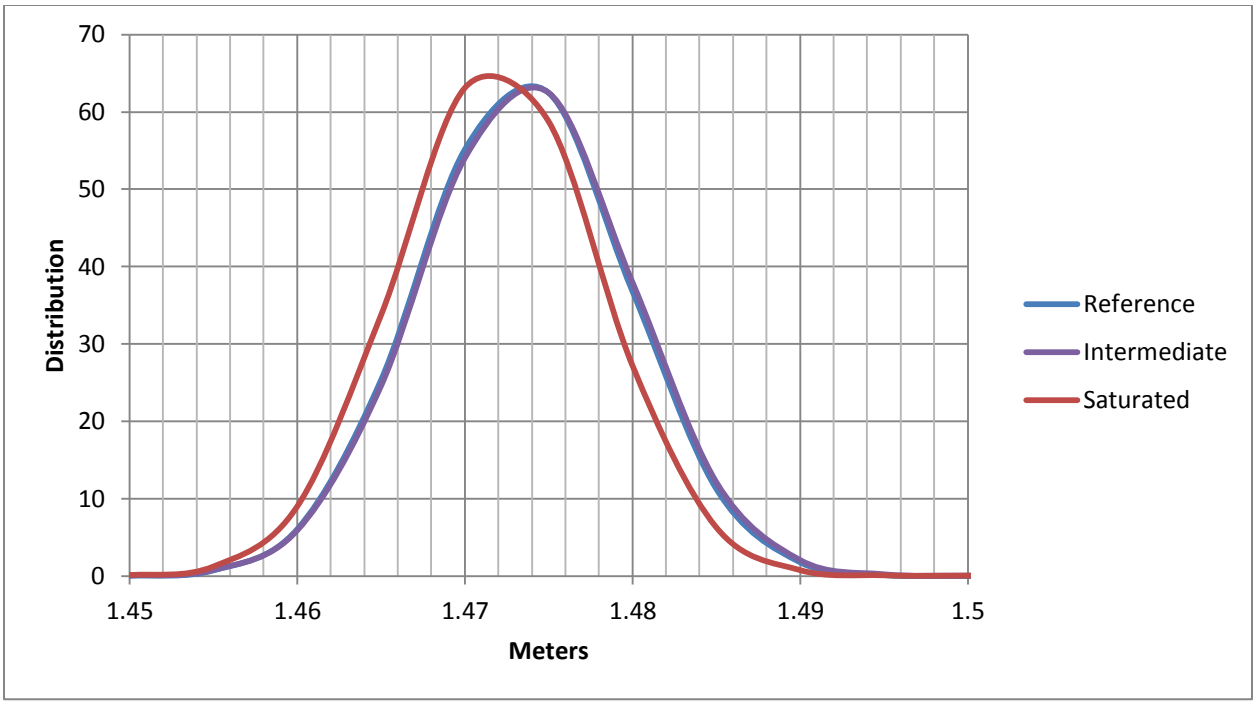


Figure 5.20. Kinect Distance Value Distributions – Test 4 – 1.52 m Target

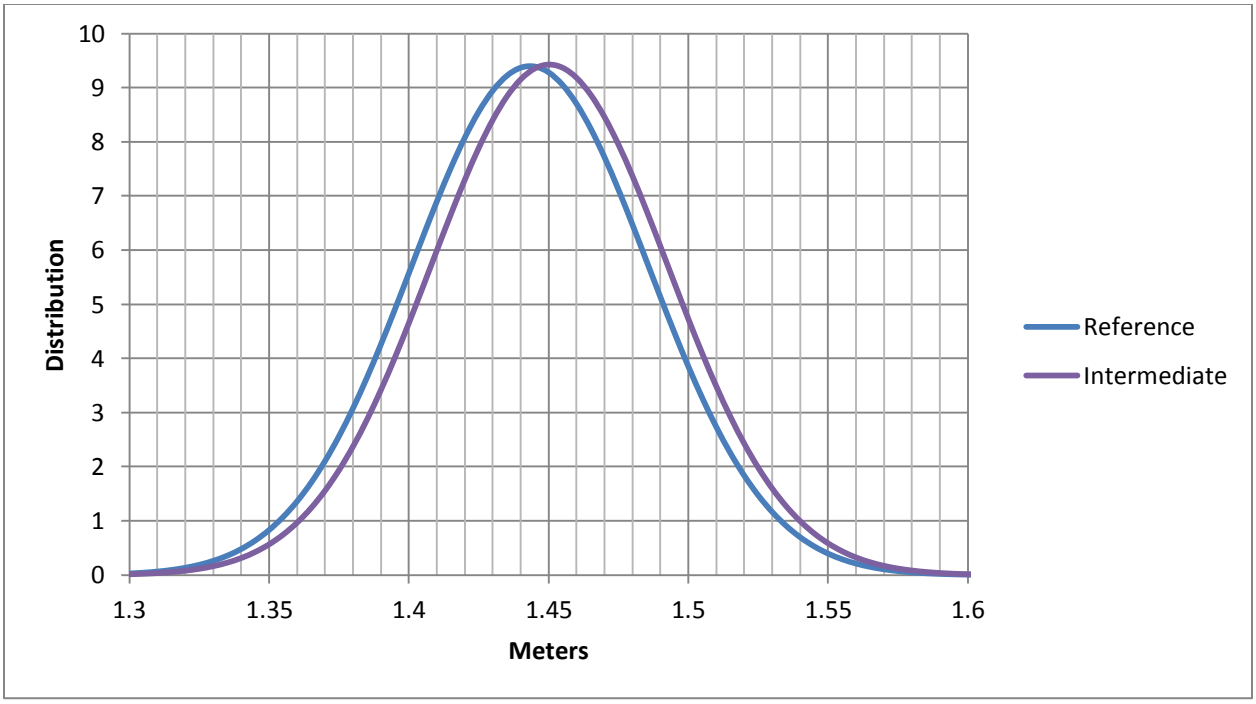


Figure 5.21. Last Echo Lidar Distance Value Distributions – Test 4 – 1.52 m Target

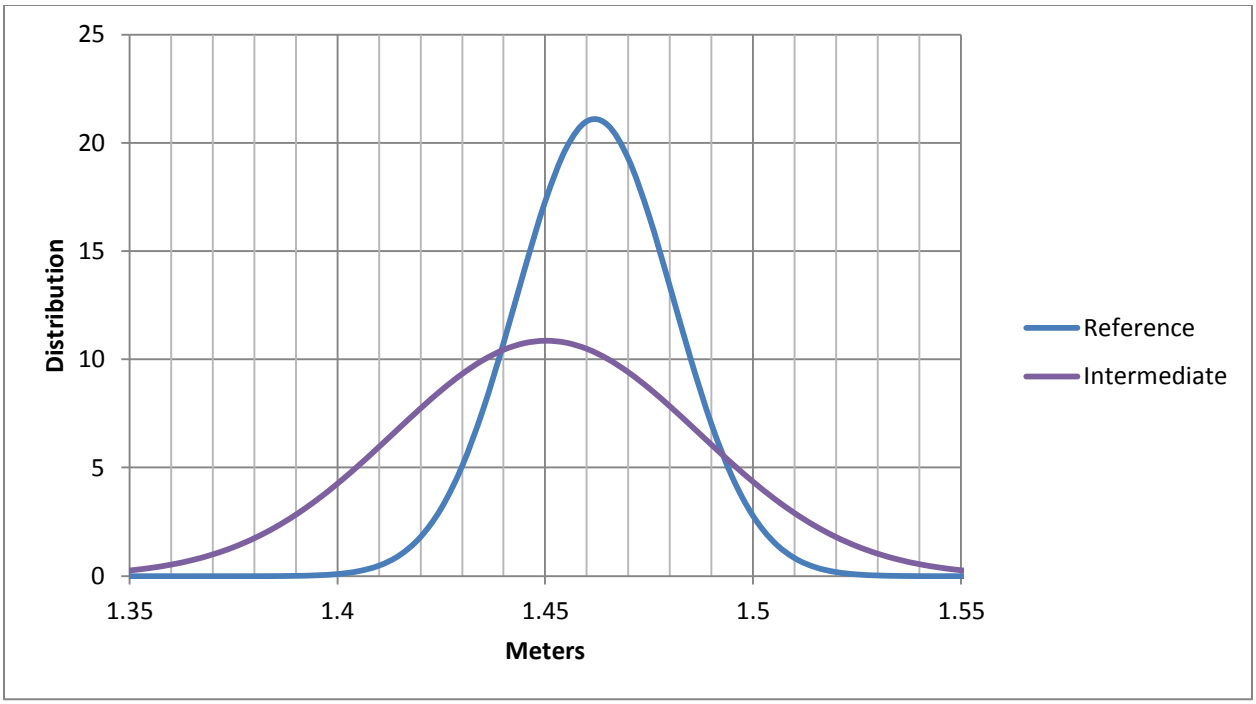


Figure 5.22. First Echo Lidar Distance Value Distributions – Test 4 – 1.52 m Target

Tables 5.16-5.18 present the results for the third target which is located at 2.13 m. The Kinect data reveals the number of valid points has decreased compared to the target at 1.52 m, and the Δ unique values have increased. This is expected as the target moves away from the sensor. Table 5.16 also shows there is no loss of target percentage identified and no large error in the tests. The graph for the Kinect data is given in Figure 5.23. One can see the dust has a slight effect on the distribution of points but not the average distance. Table 5.17 indicates that the lidar is able to return a partial target in saturated dust levels, but there is again significant error due to interference. This reinforces the results presented in Section 5.2 and the visual images in Figure 5.16. The plots in Figure 5.24 illustrate how the average distance jumps sharply and then moves back toward the reference value for the intermediate case last echo lidar data. Again, the dust makes the target appear to be approximately 85 mm further from the lidar than it actually is, while the stated accuracy of the lidar at this distance is +/- 30 mm. The first echo lidar data is

once again unable to see the target in the saturated dust case. The graphs for the reference and intermediate cases are given in Figure 5.25.

Table 5.16. Kinect Data – Test 4 – 2.13 m Target

Dust Level	Valid	% of Ref	Min (m)	Max (m)	Avg (m)	Error (mm)	Std Dev	Unique	Δ Unique (mm)
Reference	10406	N/A	2.076	2.140	2.098	N/A	0.009	6	12.80
Intermediate	10382	99.77	2.076	2.140	2.099	1.29	0.009	6	12.80
Saturated	10973	105.45	2.051	2.140	2.098	0.11	0.011	8	12.71

Table 5.17. Last Echo Lidar Data – Test 4 – 2.13 m Target

Dust Level	Valid	% of Ref	Min (m)	Max (m)	Avg (m)	Error (mm)	Std Dev	Unique	Δ Unique (mm)
Reference	862	N/A	2.027	2.128	2.067	N/A	0.012	72	1.42
Intermediate	918	106.50	2.019	2.177	2.075	8.31	0.015	93	1.72
Saturated	200	23.20	2.124	2.200	2.153	85.51	0.011	45	1.73

Table 5.18. First Echo Lidar Data – Test 4 – 2.13 m Target

Dust Level	Valid	% of Ref	Min (m)	Max (m)	Avg (m)	Error (mm)	Std Dev	Unique	Δ Unique (mm)
Reference	803	N/A	2.026	2.121	2.068	N/A	0.013	78	1.23
Intermediate	567	70.61	2.019	2.130	2.075	7.12	0.014	74	1.52
Saturated	None								

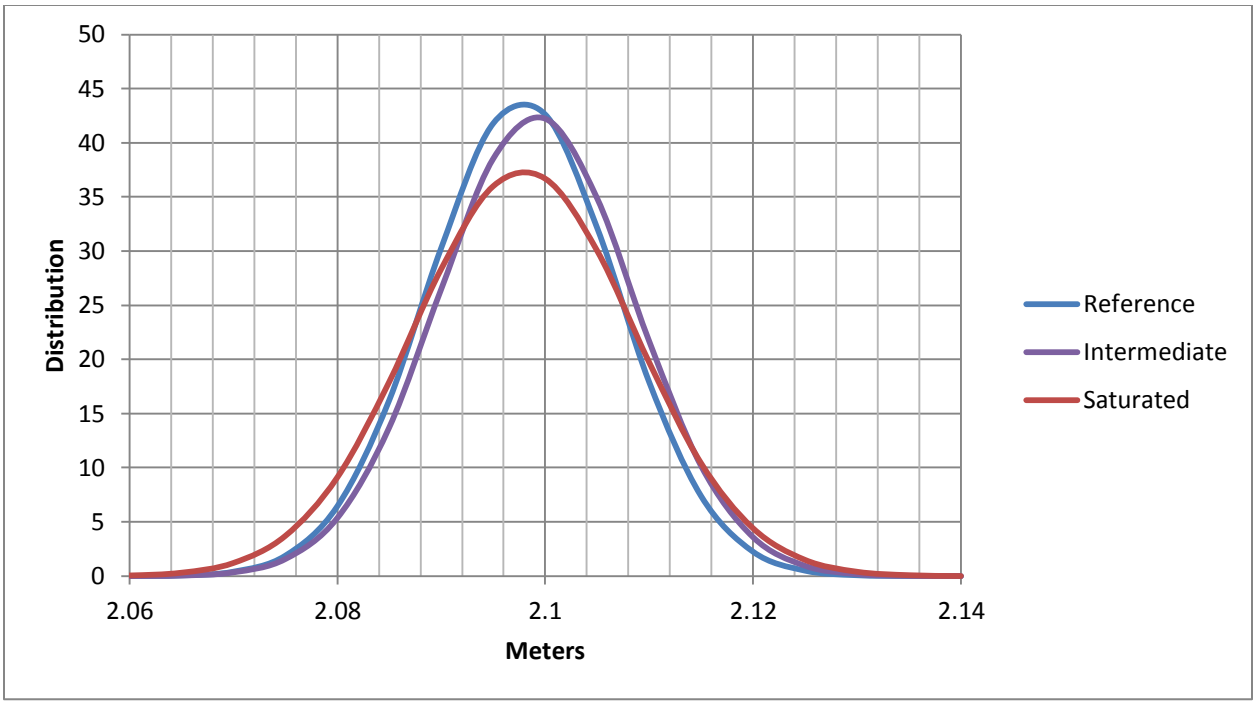


Figure 5.23. Kinect Distance Value Distributions – Test 4 – 2.13 m Target

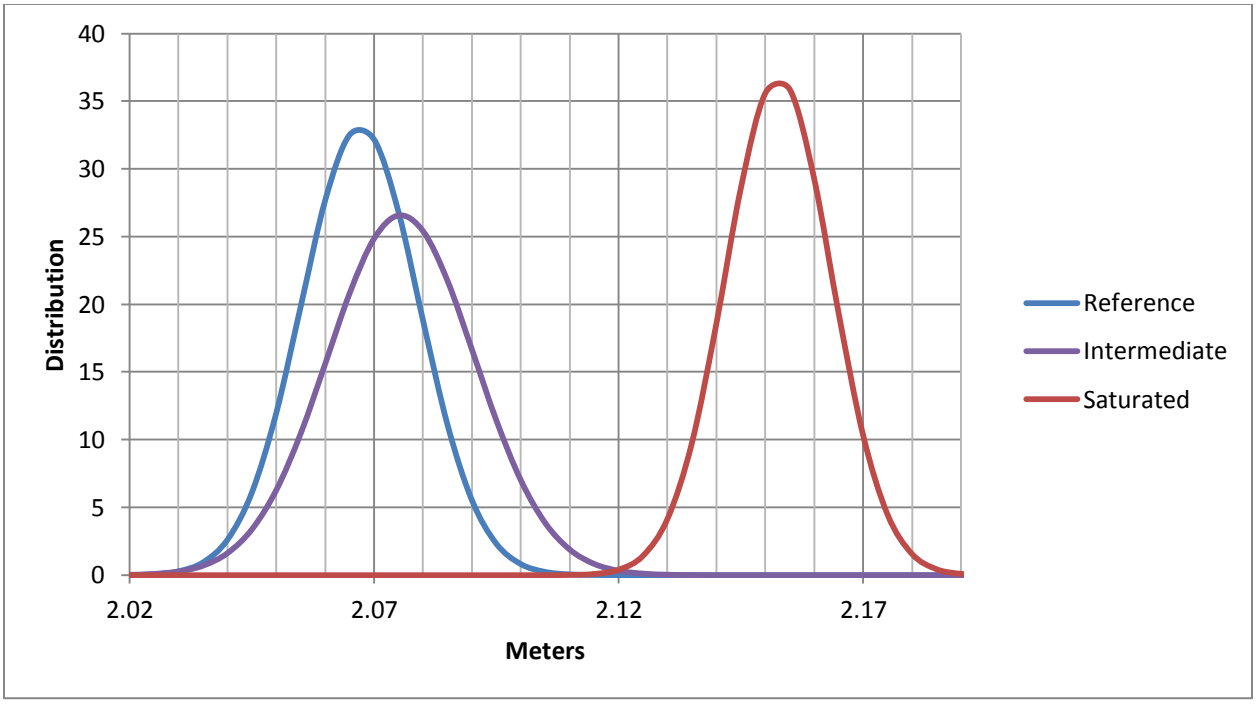


Figure 5.24. Last Echo Lidar Distance Value Distributions – Test 4 – 2.13 m Target

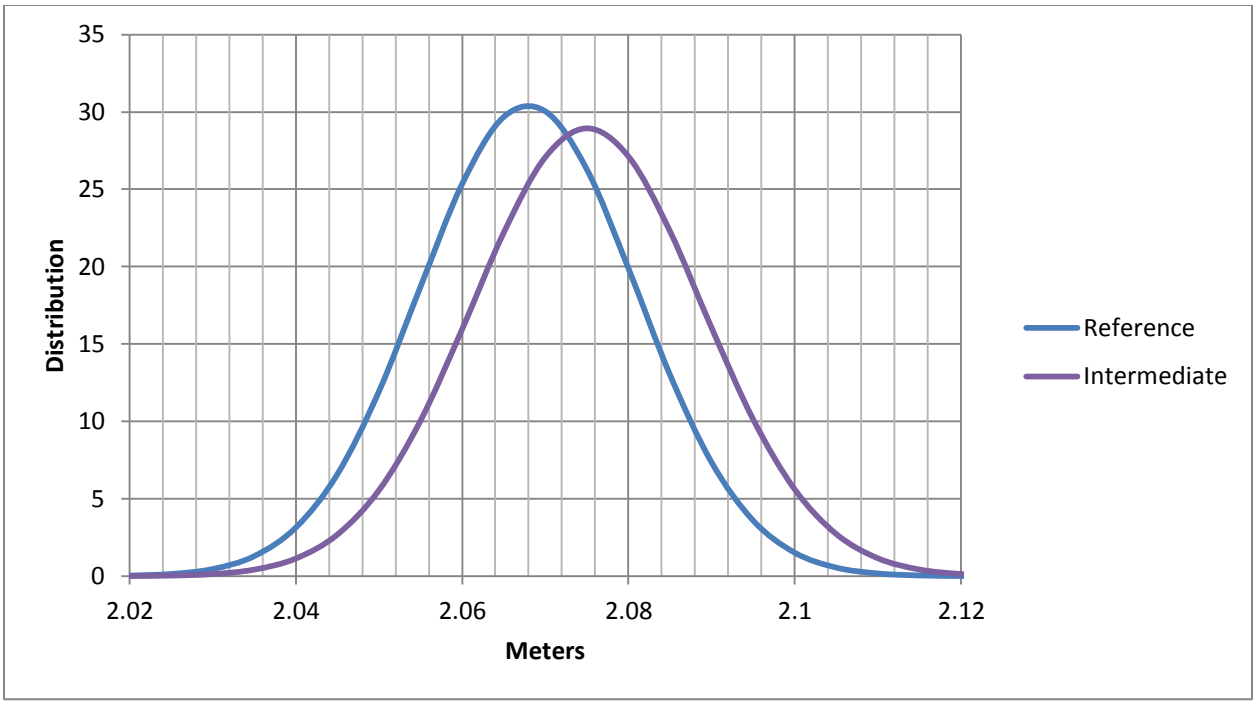


Figure 5.25. First Echo Lidar Distance Value Distributions – Test 4 – 2.13 m Target

The final target for test 4 is located at a distance of 2.74 m from the sensors. Tables 5.19-5.21 give the results. Table 5.19 indicates the Kinect is starting to lose part of the target and incurs a larger error during dusty conditions at this distance. This supports the results from earlier tests with larger targets at longer distances. However, it is still less error than that reported by the lidar. It is unclear what causes the anomalous results for the intermediate dust image. Figure 5.26 shows how the data for the intermediate dust image does not align with the others for the same target. The lidar again has problems with the error rate during dusty conditions. The chart in Figure 5.27 looks almost identical to the one from the previous target given in Figure 5.24. This shows that at 2.74 m the last echo still has trouble with interference and is still reporting an error of almost 90 mm in the saturated case. The first echo again produced no results in saturated dust. The first echo lidar data is graphed in Figure 5.28.

Table 5.19. Kinect Data – Test 4 – 2.74 m Target

Dust Level	Valid	% of Ref	Min (m)	Max (m)	Avg (m)	Error (mm)	Std Dev	Unique	Δ Unique (mm)
Reference	9217	N/A	2.551	2.669	2.602	N/A	0.021	7	19.67
Intermediate	9236	100.21	2.532	2.942	2.610	8.01	0.045	13	34.17
Saturated	7120	77.25	2.551	2.649	2.608	6.19	0.020	6	19.60

Table 5.20. Last Echo Lidar Data – Test 4 – 2.74 m Target

Dust Level	Valid	% of Ref	Min (m)	Max (m)	Avg (m)	Error (mm)	Std Dev	Unique	Δ Unique (mm)
Reference	557	N/A	2.587	2.701	2.649	N/A	0.012	67	1.73
Intermediate	558	100.18	2.506	2.703	2.657	8.49	0.015	72	2.78
Saturated	550	98.74	2.671	2.774	2.737	87.62	0.013	64	1.64

Table 5.21. First Echo Lidar Data – Test 4 – 2.74 m Target

Dust Level	Valid	% of Ref	Min (m)	Max (m)	Avg (m)	Error (mm)	Std Dev	Unique	Δ Unique (mm)
Reference	533	N/A	2.616	2.706	2.650	N/A	0.014	72	1.27
Intermediate	430	80.68	2.600	2.706	2.659	9.12	0.014	70	1.54
Saturated	None								

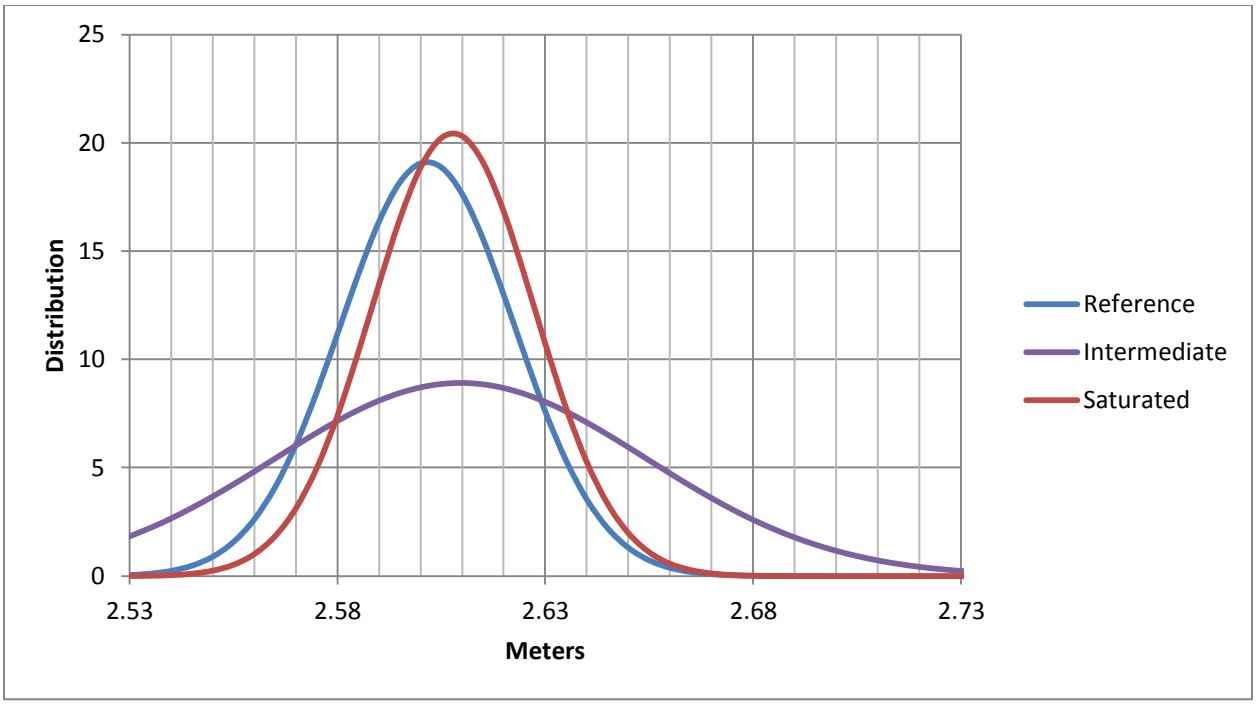


Figure 5.26. Kinect Distance Value Distributions – Test 4 – 2.74 m Target

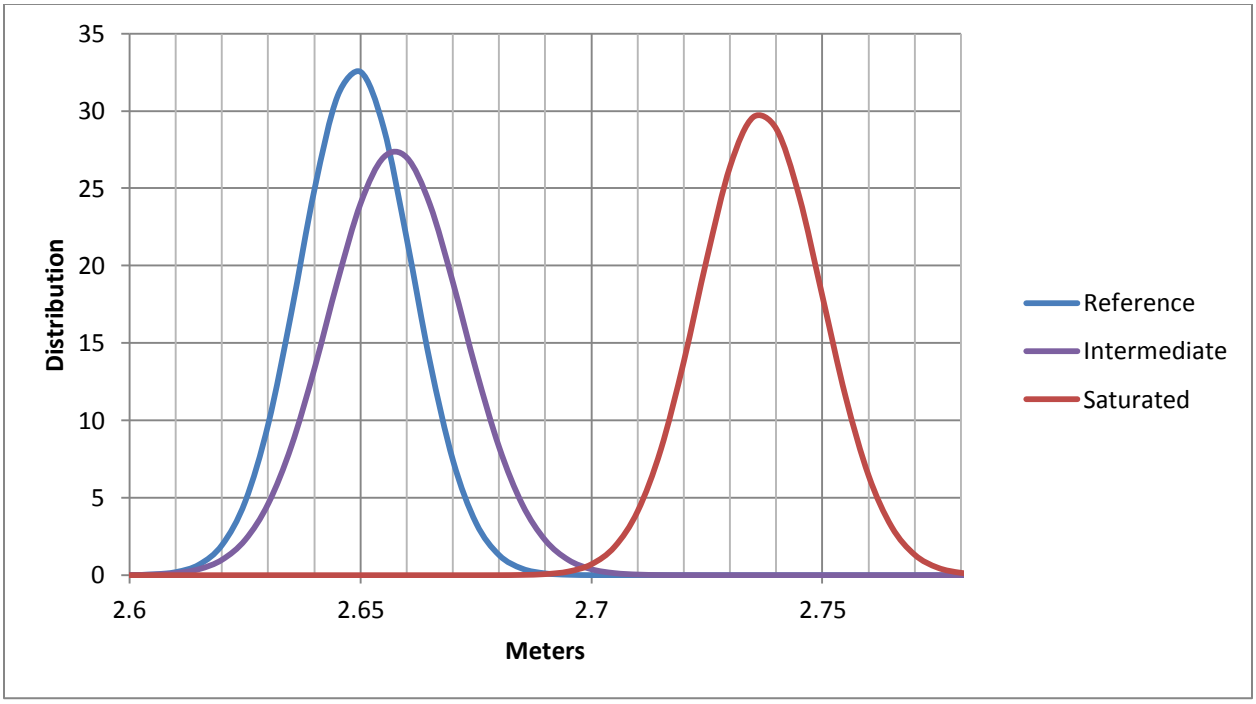


Figure 5.27. Last Echo Lidar Distance Value Distributions – Test 4 – 2.74 m Target

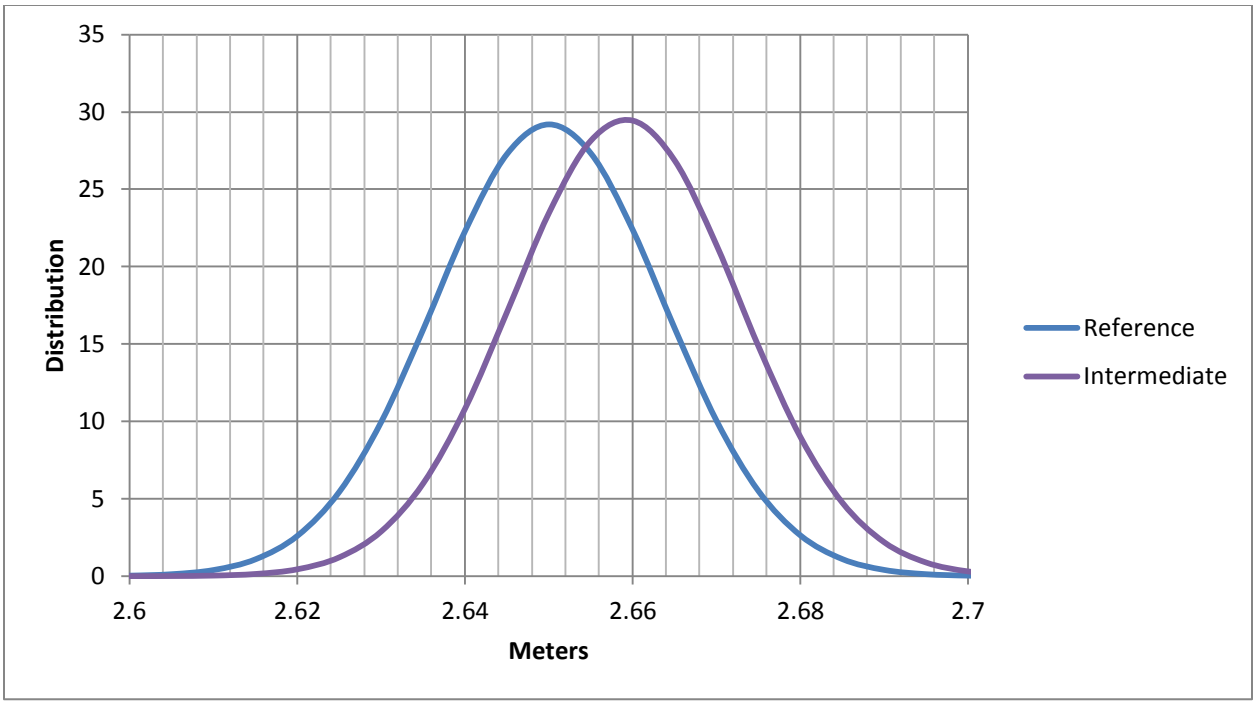


Figure 5.28. First Echo Lidar Distance Value Distributions – Test 4 – 2.74 m Target

The results from this test 4 reveal that the data using the smaller targets follows the same pattern set by the larger targets in tests 1 - 3. That is, the performance of the Kinect improves as the targets are moved closer to the sensors, and the performance of the lidar improves as the targets are moved away from the sensors. There is an area within about 2 m of the lidar where the sensor returns nothing during high dust levels. Distances from approximately 2 m to 3 m constitutes a zone where the lidar returns target points but the distance values are unreliable making their use with autonomous navigation and obstacle avoidance a challenge.

5.5 Test 5 Results

All of the testing so far has been conducted in the UA-1 simulant. However, there is another simulant available called BP-1, and it is desirable to know whether this material affects the sensors in the same manner as the UA-1. Due to the restricted size of the BP-1 area, only one

test is conducted with a single target at a distance of 1.07 m. This is the same distance as in test 3, but a smaller target is used here. Tests were conducted for all three suspended dust levels, saturated, intermediate, and reference. It should be noted that the distance data reported for the intermediate range could be skewed by the fact that the suspended dust level dropped below the intermediate range during the test. The results are given in Tables 5.22-5.24. The Kinect data in Table 5.22 shows little decrease in target percentage identified and a very small error across all dust levels. This corresponds to the data in test 3. Figure 5.29 illustrates the uniformity of this

Table 5.22. Kinect Data – Test 5

Dust Level	Valid	% of Ref	Min (m)	Max (m)	Avg (m)	Error (mm)	Std Dev	Unique	Δ Unique (mm)
Reference	62261	N/A	1.017	1.116	1.064	N/A	0.005	31	3.30
Intermediate	62174	99.86	1.030	1.127	1.065	0.88	0.006	30	3.35
Saturated	61805	99.27	1.033	1.134	1.065	0.68	0.006	31	3.37

Table 5.23. Last Echo Lidar Data – Test 5

Dust Level	Valid	% of Ref	Min (m)	Max (m)	Avg (m)	Error (mm)	Std Dev	Unique	Δ Unique (mm)
Reference	3340	N/A	0.996	1.139	1.057	N/A	0.017	127	1.14
Intermediate	3354	100.42	0.988	1.130	1.046	10.48	0.018	116	1.24
Saturated	None								

Table 5.24. First Echo Lidar Data – Test 5

Dust Level	Valid	% of Ref	Min (m)	Max (m)	Avg (m)	Error (mm)	Std Dev	Unique	Δ Unique (mm)
Reference	2479	N/A	1.001	1.125	1.054	N/A	0.026	123	1.02
Intermediate	1920	77.45	1.001	1.156	1.069	14.16	0.041	155	1.01
Saturated	None								

data. The last echo lidar data in Table 5.23 shows improvement over UA-1 tests in that it is able to return a target for the intermediate dust level. However, this could be the result of the dust level falling just below the threshold defined for the intermediate range as the test was conducted. There is still no return for the saturated dust case, and the error for the intermediate case is higher than with the Kinect. The first echo data in Table 5.24 also gives a return for the intermediate dust level. Unfortunately, the first echo intermediate case has a higher error and a lower target percentage than the last echo results for the same image. Figures 5.30 and 5.31 show the distributions for the last echo and first echo results, respectively. One can see in both figures how the average value, or curve peak, shifts during the intermediate dust case. This is similar to the effects observed previously.

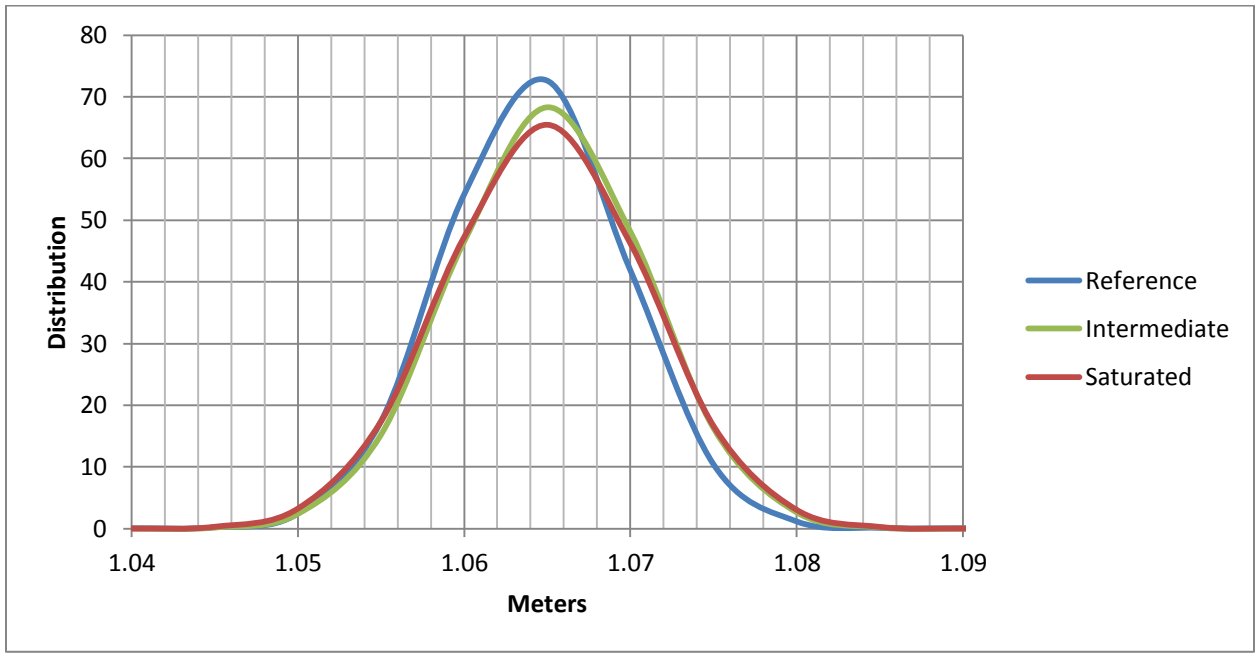


Figure 5.29. Kinect Distance Value Distributions – Test 5

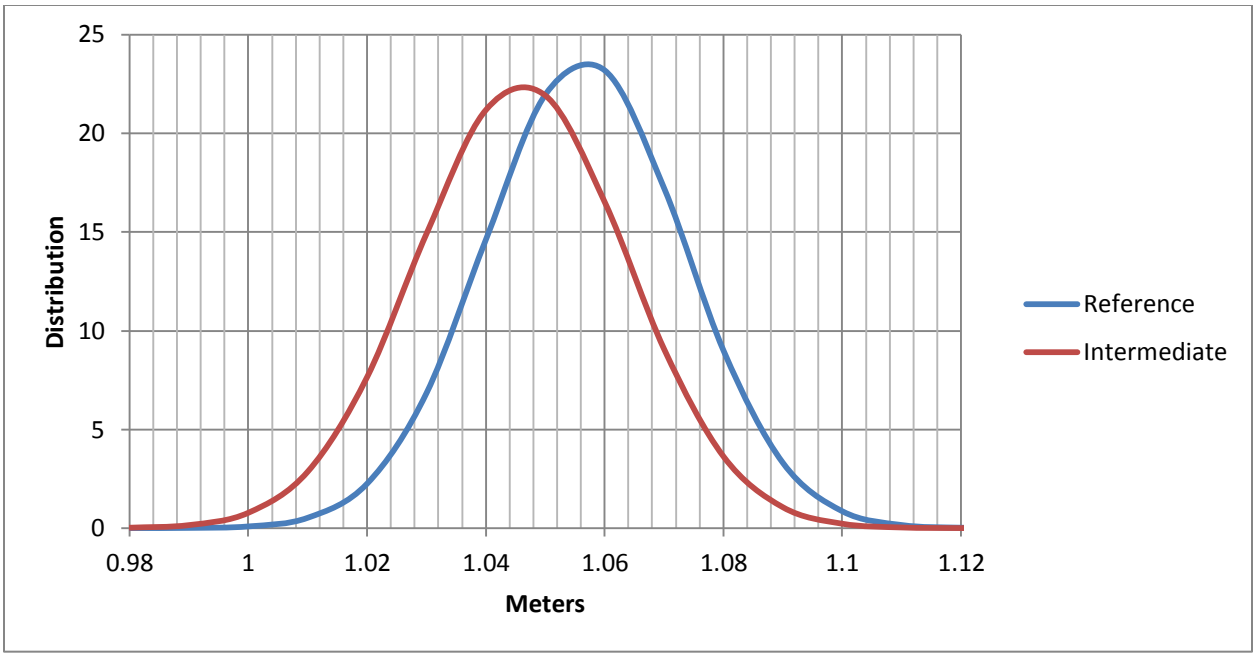


Figure 5.30. Last Echo Lidar Distance Value Distributions – Test 5

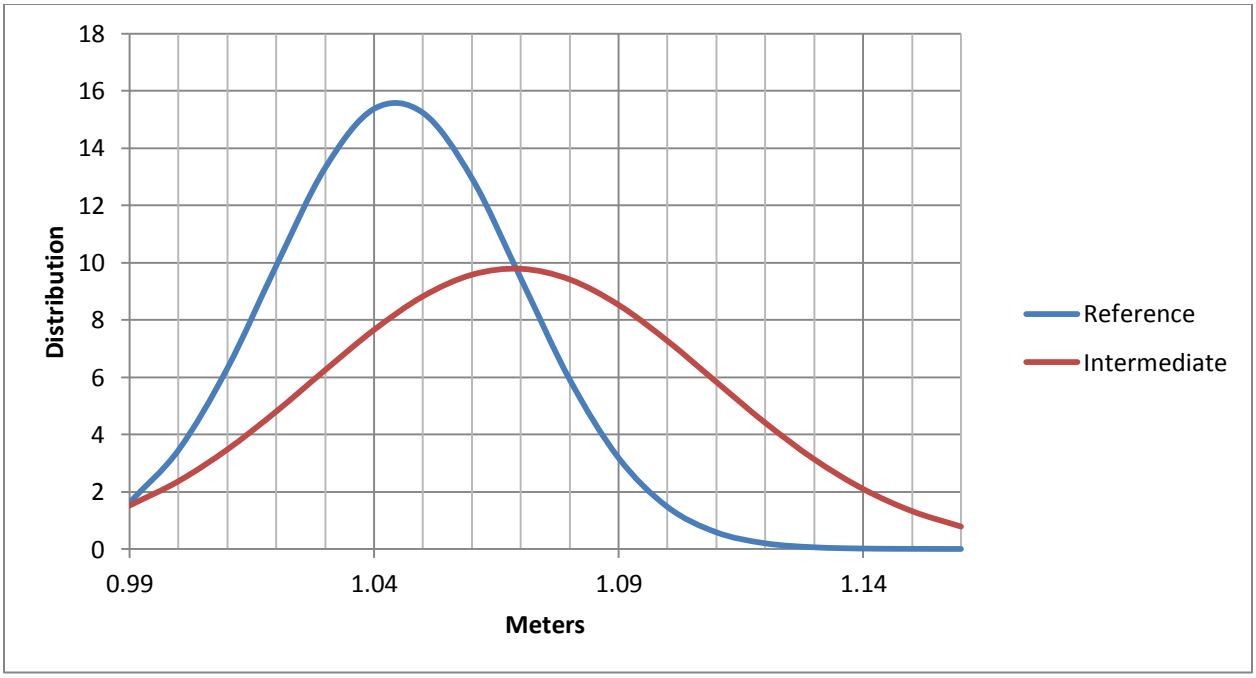


Figure 5.31. First Echo Lidar Distance Value Distributions – Test 5

The improved performance of the lidar sensor in this scenario could indicate that the BP-1 simulant does not interfere with the operation of the sensor as much as the UA-1 simulant. Another plausible explanation is that due to the smaller BP-1 test area, the testbed was not as thoroughly saturated with dust. When testing in the intermediate range, the dust settled faster than during UA-1 testing leading to the threshold of the suspended dust level dropping below that of the intermediate range before completing data collection. However, one must keep in mind the smaller size of the BP-1 area and the possibility that more simulant is needed to reproduce the effects seen in the larger UA-1 testing area. In any case, it is clear that the BP-1 causes the most interference with the first echo lidar data, and less interference on the last echo data. The Kinect sensor data is the least affected of the three datasets, and this confirms the results seen in the UA-1 simulant at close distances.

5.6 Data Comparison

To help put all of this data into perspective, two metrics for the Kinect and last echo lidar data are pulled from the tables above and presented comparatively. These are the percentage of the target returned by the sensor (% of Ref), and the error incurred in the average distance measurements (Error). Both are taken from the saturated dust test for each target in the UA-1 testing area. This allows a direct examination of the dust's impact on sensor image quality at seven distances. For the targets at 2.59 m and 3.81 m, the previous data tables present two images for the Kinect in the saturated tests. In these cases, the values are averaged. Table 5.25 shows the data aggregated from the previous tests.

Table 5.25. Kinect and Lidar Image Quality Comparison

Distance (m)	Kinect		Lidar	
	Target %	Error (mm)	Target %	Error (mm)
0.91	99.08	2.58	0	
1.07	100.09	0.57	0	
1.52	99.21	1.47	0	
2.13	105.45	0.11	23.20	85.51
2.59	24.35	10.94	38.46	156.96
2.74	77.25	6.19	98.74	87.62
3.81	15.91	18.67	97.84	10.16

It is immediately obvious that the Lidar has trouble with any target closer than ~ 2 m, while the Kinect excels at this range. A graphical representation of the target percentage vs. distance is given in Figure 5.32. Except for the data point at 2.59 m, the Kinect curve largely follows an exponentially declining path. This is in accordance with the non-linear nature of the sensor as discussed earlier. The lidar begins by not returning any target, and then surpasses the Kinect sensor at ~ 2.5 m.

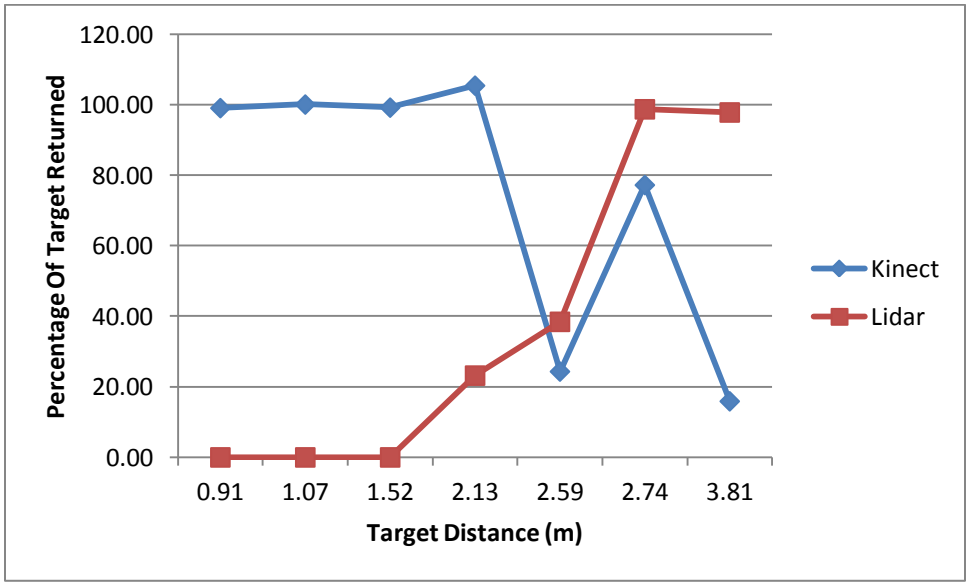


Figure 5.32. Target Percentage Comparison

A plot of the average distance error vs. distance is given in Figure 5.33. The Kinect data once again follows an exponential curve. However, it is dwarfed by the significantly higher error present in the lidar data. Even as the target percentage of the lidar surpasses the Kinect at ~ 2.5 m, the lidar error does not drop below that of the Kinect until ~ 3.5 m. This indicates the Kinect is capable of returning more accurate depth data in dusty conditions, even when it sees less of the target than the lidar. Only when the target begins to move out of the effective range of the Kinect and into the range of the lidar unimpeded by the dust bowl effect does the error rate shift in favor of the lidar.

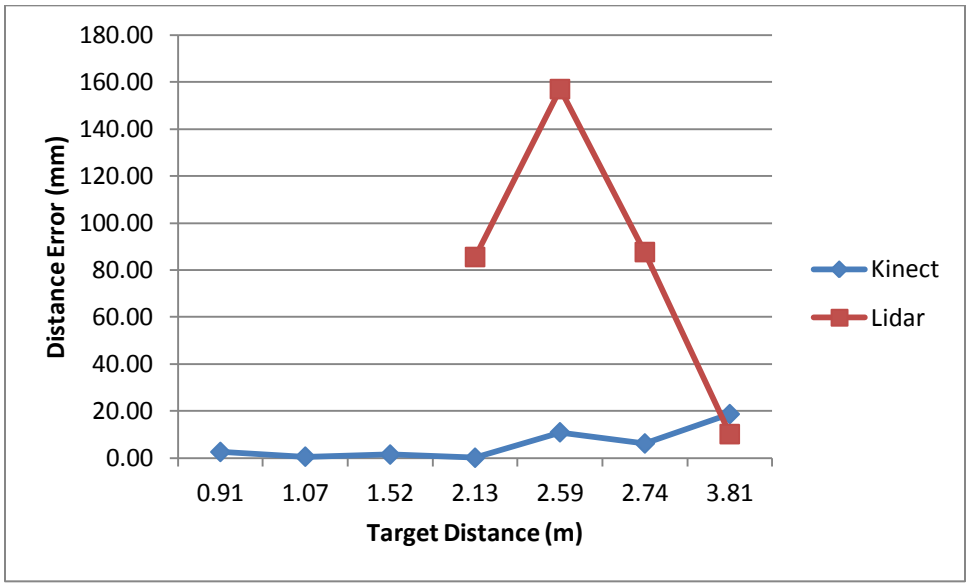


Figure 5.33. Distance Error Comparison

CHAPTER 6

CONCLUSIONS

6.1 Summary

The preceding chapters provide an experimental analysis of the effects of lunar dust simulant on the ability of two different sensors to provide data for autonomous robotic navigation and obstacle avoidance applications on the Moon. The two sensors investigated are the Microsoft Kinect for Xbox 360, a structured light depth sensor; and a Hokuyo UTM-30LX-EW, a scanning ToF depth sensor.

The scanning lidar data is broken into two sets: first echo only data to simulate a single-echo sensor, and last echo data to observe the effectiveness of the multi-echo functionality. Five test scenarios were set up to accommodate various target sizes and distances. Testing was conducted at multiple dust levels to compare with a reference case, i.e. no dust, for each scenario. A total of eight target planes are extracted for each of these dust levels from the three different sensor datasets and analyzed for errors. Accuracy is determined by the percentage of a target which is visible and the average distance to the target plane as compared to the reference case.

The results of the testing show the single-echo data to be unusable in any amount of dust. Therefore, a lidar sensor with multi-echo functionality is needed in lunar or terrestrial mining conditions. The last echo of the lidar dataset displays unique characteristics in that it returns

targets at greater distances but cannot see targets at shorter distances in dusty conditions. In saturated dust levels, the dust reflects the laser and forms a dust bowl artifact around the sensor. This appears to be a solid sphere surrounding the sensor at a distance of approximately 0.5 m. This not only blinds the sensor to any targets within 2 m, but also interferes with the planar extraction algorithm's ability to identify any planes in the image. The dust bowl would theoretically have a similar impact on obstacle avoidance algorithms as it would be interpreted as an obstacle surrounding the robot. One possible explanation as to why this would affect targets out to 2 m is found in the lidar's datasheet. There is a statement that reads: "Two closely positioned objects or low reflectance objects may not produce multiple echoes, so that they are not detectable as separate ones [49]." It could be that the return timing between the dust and these close targets is not sufficient to distinguish between the two. There is another boundary point at a distance of approximately 3.5 m in the lidar data. For points further past this boundary, the dust seems to have little effect on the distance measurements. In the zone from 2-3.5 m, the sensor returns targets, but they can be significantly distorted, and there is a large error in the average distance values.

The Kinect sensor has proven to be a solid performer in adverse dust conditions. While it struggles to provide full targets at distances over 3-3.5 m, the average distance values returned for these partial planes have a relatively low error. This error rate drops and the performance improves exponentially as the targets move closer to the sensor. The best feature of the Kinect is that it does not return false positive target points due to reflection from the dust. Only physical objects large enough to reflect part of the projected structured light pattern will produce a return in the data. This means there is no dust bowl effect which would impede the operation of obstacle avoidance or navigation algorithms.

These findings suggest that the Kinect sensor is a more capable short range alternative to lidar for autonomous robotic navigation applications in lunar and other related environments where dust is a major concern and IR saturation is not a concern. The output data format of the Kinect is similar to that provided by the lidar sensor. This allows the Kinect to serve as a lower cost drop-in replacement solution which requires little modification to an existing algorithm. If the wide FOV provided by the lidar is a necessity, one can use an array of four Kinect sensors to provide a similar coverage area and still have a 1500% reduction in cost. If a longer range is needed, a hybrid system employing both sensors is proposed. In this case, the multi-echo lidar would be used for long range localization and navigation. All the lidar data from 3.5 m and closer would be discarded and replaced with Kinect data for short range obstacle detection and avoidance.

6.2 Future Work

There are several research paths which could expand on the knowledge gathered through this experimentation. The results presented here could have an impact for Earth mining applications as well. Different types of terrestrial dust, such as that found in coal mines, could be analyzed to determine if these sensors would provide similar performance characteristics in those environments.

The dust sensor system used in these tests provides a rudimentary measurement of the suspended dust level. Tests performed using more expensive and sophisticated dust sensors could allow one to determine the exact dust levels which cause distortion in the lidar sensor images. This would allow robotic designers to determine on an individual basis if the environment for which their robot is intended would warrant a hybrid depth sensor system.

The targets used in this research are all flat wooden planes. Obstacles in the real world are not always so uniform and precise; and the color, composition, and reflectivity of an object are known to have an effect on some depth sensors [45]. Therefore, testing could be performed on targets made of various materials to see if they show similar performance in dust. The targets could also be made with different shapes and colors. Another option would be to test moving targets to see if the sensors are capable of providing enough data to allow an algorithm to keep a lock on the targets and track them in the dust. Also, the targets in this research are analyzed using a planar extraction algorithm. Some robotic applications use a line extraction algorithm to analyze the obstacles and surfaces in their surroundings. One could study how the performances of these two types of algorithms compare in dusty environments.

The Kinect provides depth images at a rate of 30 frames per second. This is video speed and one could possibly take advantage of video noise filtering algorithms to further reduce the noise caused by the dust at greater distances. Filtering algorithms such as those mentioned in [54] can look not only spatially within a frame but also temporally between multiple frames. This has the possibility of making the Kinect data even more valuable in dusty environments.

REFERENCES

[1] P. Althaus, “Indoor Navigation for Mobile Robots: Control and Representations,” Ph.D. dissertation, Royal Inst. Tech., Stockholm, Sweden, 2003. Available: http://www.diva-portal.org/smash/get/diva2:9477/FULLTEXT01.pdf

[2] V. Nguyen et al., “A comparison of line extraction algorithms using 2D laser rangefinder for indoor mobile robotics," in *Int. Conf. Intelligent Robots and Systems*, 2005, pp. 1929-1934. doi: 10.1109/IROS.2005.1545234

[3] M. A. Batalin et al., “Mobile Robot Navigation using a Sensor Network,” in *Int. Conf. Robotics and Automation*, New Orleans, LA, 2004, pp. 636-642. doi: 10.1109/ROBOT.2004.1307220

[4] J. Borenstein et al., “Where am I? Sensors and Methods for Mobile Robot Positioning,” University of Michigan, April 1996. Available: http://www.iau.dtu.dk/ancona/385partwhereami.pdf

[5] F. Zhang et al., “Cumulative Error Estimation from Noisy Relative Measurements,” in *Proc. 16th Int. IEEE Conf. on Intelligent Transportation Systems*, The Hague, The Netherlands, 2013, pp. 1422-1429. doi: 10.1109/ITSC.2013.6728430

[6] G. Dai et al., “Systematic Error Analysis and Modeling of MEMS close-loop capacitive accelerometer,” in *Proc. 5th IEEE Int. Conf. Nano/Micro Engineered and Molecular Systems*, Xiamen, China, 2010, pp. 88-92. doi: 10.1109/NEMS.2010.5592157

[7] A. Tar et al., “3D Geometry Reconstruction using Large Infrared Proximity Array for Robotic Applications,” in *Proc. IEEE Int. Conf. Mechatronics*, Malaga, Spain, 2009, pp. 1-6. doi: 10.1109/ICMECH.2009.4957218

[8] C.-S. Park et al., "Comparison of plane extraction performance using laser scanner and Kinect," in *8th Int. Conf. Ubiquitous Robots and Ambient Intelligence*, Incheon, Korea, 2011, pp. 153-155. doi: 10.1109/URAI.2011.6145951

[9] C. Ye and J. Borenstein, “Characterization of a 2-D Laser Scanner for Mobile Robot Obstacle Negotiation,” in *Proc. IEEE Int. Conf. Robotics & Automation*, Washington, DC, 2002, pp. 2512-2518. doi: 10.1109/ROBOT.2002.1013609

[10] J. L. Martinez et al., “Object following and obstacle avoidance using a laser scanner in the outdoor mobile robot Auriga-α," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, Victoria, BC, 1998, pp. 204-209. doi: 10.1109/IROS.1998.724620

[11] S. I. Roumeliotis and G. A. Bekey, "SEGMENTS: a layered, dual-Kalman filter algorithm for indoor feature extraction," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, Takamatsu, 2000, pp. 454-461 vol.1. doi: 10.1109/IROS.2000.894646

[12] L. Zhang and B. K. Ghosh, "Line segment based map building and localization using 2D laser rangefinder," in *IEEE Int. Conf. Robotics and Automation*, San Francisco, CA, 2000, pp. 2538-2543 vol. 3. doi: 10.1109/ROBOT.2000.846410

[13] J. Guivant et al., "High accuracy navigation using laser range sensors in outdoor applications," in *IEEE Int. Conf. Robotics and Automation*, San Francisco, CA, 2000, pp. 3817-3822 vol. 4. doi: 10.1109/ROBOT.2000.845326

[14] A. Scott et al., "Quantitative and qualitative comparison of three laser-range mapping algorithms using two types of laser scanner data," in *IEEE Int. Conf. Systems, Man, and Cybernetics*, Nashville, TN, 2000, pp. 1422-1427 vol. 2. doi: 10.1109/ICSMC.2000.886054

[15] C. C. Allen et al., “Lunar Oxygen Production – A Maturing Technology,” Engineering, Construction, and Operations in Space IV American Society of Civil Engineers, 1994, pp. 1157-1166 [Online]. Available: http://ares.jsc.nasa.gov/humanexplore/exploration/exlibrary/docs/eic048.html

[16] W. Fowler and A. Cohen, "The Texas Space Grant Multi-University Multi-Disciplinary Design Project," in *Proc, Frontiers in Education Conf.*, Salt Lake City, UT, 1996, pp. 489-493 vol. 1. doi: 10.1109/FIE.1996.570017

[17] A. Ignatiev et al., "Solar cell development on the surface of Moon from in-situ lunar resources," in *Proc. IEEE Aerospace Conf.*, 2004, pp. 318 vol. 1. doi: 10.1109/AERO.2004.1367615

[18] M. DiGiuseppe et al., "Lunar regolith control and resource utilization," in *IEEE Long Island Systems, Applications, and Technology Conf.*, Farmingdale, NY, 2009, pp. 1-5. doi: 10.1109/LISAT.2009.5031560

[19] T. J. Stubbs et al., "A Dynamic Fountain Model For Lunar Dust," Lunar and Planetary Science XXXVI, 2005, [Online]. Available: http://www.lpi.usra.edu/meetings/lpsc2005/pdf/1899.pdf

[20] M. K. Mazumder et al., "Electrostatic and Gravitational Transport of Lunar Dust in the Airless Atmosphere of the Moon," in *IEEE Industry Applications Society Annu. Meeting*, Edmonton, Alta, 2008, pp. 1-4. doi: 10.1109/08IAS.2008.124

[21] P. E. Clark et al., "Characterizing Physical and Electrostatic Properties of Lunar Dust as a Basis for Developing Dust Removal Tools," NLSI Lunar Science Conf., 2008, [Online]. Available: http://www.lpi.usra.edu/meetings/nlsc2008/pdf/2077.pdf

[22] L. David. (2006, November 7). *Lunar Explorers Face Moon Dust Dilemma* [Online]. Available: http://www.space.com/3080-lunar-explorers-face-moon-dust-dilemma.html

[23] R. Pirich et al., "Self-Cleaning and Anti-Contamination Coatings for Space Exploration: An Overview," in *Proc. SPIE*, 2008, Vol. 7069. doi: 10.1117/12.793794

[24] H. Kawamoto et al., "Mitigation of lunar dust on solar panels and optical elements utilizing electrostatic traveling-wave," in *Journal of Electrostatics 69*, 2011, pp. 370-379. doi: 10.1016/j.elstat.2011.04.016

[25] H. Kawamoto, "Electrostatic Cleaning Device for Removing Lunar Dust Adhered to Spacesuits," in *American Society of Civil Engineers Journal of Aerospace Engineering*, 2012, pp. 470-473 Vol. 25 No. 3. doi: 10.1061/(ASCE)AS.1943-5525.0000143

[26] H. Kawamoto and N. Hara, "Electrostatic Cleaning System for Removing Lunar Dust Adhering to Space Suits," in *American Society of Civil Engineers Journal of Aerospace Engineering*, 2011, pp. 442-444 Vol. 24 No. 4. doi: 10.1061/(ASCE)AS.1943-5525.0000084

[27] T. Cruz et al., *The Lunar Dust Dilemma* [Online]. Available: http://www.academia.edu/download/31116466/Design_Report.pdf

[28] J. Ryde and N. Hillier, "Performance of laser and radar ranging devices in adverse environmental conditions," in *Journal of Field Robotics*, vol. 26, no. 9, pp. 712-727, Sept. 2009. doi: 10.1002/rob.20310

[29] U. Wong et al., "Comparative evaluation of range sensing technologies for underground void modeling," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, San Francisco, CA, 2011, pp. 3816-3823. doi: 10.1109/IROS.2011.6094938

[30] S. Zug et al., "Are laser scanners replaceable by Kinect sensors in robotic applications?," in *IEEE Int. Symp. Robotic and Sensors Environments*, Magdeburg, Germany, 2012, pp. 144-149. doi: 10.1109/ROSE.2012.6402619

[31] T. Bernhard et al., "A Comparative Study of Structured Light and Laser Range Finding Devices," Correll Lab, Department of Computer Science, University of Colorado at Boulder, May 2012, [Online]. Available: http://correll.cs.colorado.edu/wp-content/uploads/bernhard.pdf

[32] J. Pascoal et al., "Assessment of Laser Range Finders in risky environments," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, Nice, France, 2008, pp. 3533-3538. doi: 10.1109/IROS.2008.4650961

[33] M. Kise et al., "Performance Evaluation on Perception Sensors for Agricultural Vehicle Automation," in *2nd Int. Conf. Machine Control and Guidance*, 2010, [Online]. Available: http://www.mcg.uni-bonn.de/proceedings/30_kise.pdf

[34] V. Tretyakov and T. Linder, "Range sensors evaluation under smoky conditions for robotics applications," in *IEEE Int. Symp. Safety, Security, and Rescue Robotics*, Kyoto, Japan, 2011, pp. 215-220. doi: 10.1109/SSRR.2011.6106776

[35] D.B. Stoeser et al., "Preliminary Geological Findings on the BP-1 Simulant," NASA MSFC, Huntsville, AL, Tech. Memo NASA/TM-2010-216444, Doc. ID: 20100036344, Sept. 2010. Available: www.sti.nasa.gov

[36] Lighthouse Worldwide Solutions [Online]. Available: www.golighthouse.com

[37] TSI [Online]. Available: www.tsi.com

[38] Lighthouse Handheld 3016 IAQ Airborne Particle Counter [Online]. Available: http://www.golighthouse.com/counter/handheld-3016-iaq/

[39] Sharp GP2Y1010AU0F Compact Optical Dust Sensor Datasheet, [Online]. Available: http://www.sharp-world.com/products/device/lineup/data/pdf/datasheet/gp2y1010au_e.pdf

[40] Sharp GP2Y1010AU0F Compact Optical Dust Sensor, Digi-Key, [Online]. Available: http://www.digikey.com/product-detail/en/GP2Y1010AU0F/425-2068-ND/720164

[41] Arduino Mega 2560 Microcontroller, [Online]. Available: http://arduino.cc/en/Main/arduinoBoardMega2560

[42] Microsoft Kinect for Xbox 360, [Online]. Available: http://www.xbox.com/en-US/KINECT

[43] Kinect for Windows Sensor Components and Specifications, Microsoft Developer Network, [Online]. Available: http://msdn.microsoft.com/en-us/library/jj131033.aspx

[44] bbzippo, "Kinect in infrared," Nov, 2010, [Online]. Available: http://bbzippo.wordpress.com/2010/11/28/kinect-in-infrared/

[45] R. Macknojia et al., "Experimental characterization of two generations of Kinect's depth sensors," in *IEEE Int. Symp. Robotic and Sensors Environments*, Magdeburg, Germany, 2012, pp. 150-155. doi: 10.1109/ROSE.2012.6402634

[46] K. Khoshelham and S. O. Elberink, "Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications," *Sensors*, vol. 12, no. 2, pp. 1437-1454, Feb. 2012. doi: 10.3390/s120201437

[47] Coordinate Spaces, Microsoft Developer Network, [Online]. Available: http://msdn.microsoft.com/en-us/library/hh973078.aspx

[48] Hokuyo UTM-30LX-EW, [Online]. Available: http://www.hokuyo-aut.jp/02sensor/07scanner/download/products/utm-30lx-ew/

[49] Hokuyo UTM-30LX-EW datasheet, [Online]. Available: http://www.hokuyo-aut.jp/02sensor/07scanner/download/products/utm-30lx-ew/data/UTM-30LX-EW_spec_en.pdf

[50] File:LIDAR-scanned-SICK-LMS-animation.gif, Wikipedia, [Online]. Available: http://en.wikipedia.org/wiki/File:LIDAR-scanned-SICK-LMS-animation.gif

[51] N. Pouliot et al., "LineScout power line robot: Characterization of a UTM-30LX LIDAR system for obstacle detection," in *IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, Vilamoura, Portugal, 2012, pp. 4327-4334. doi: 10.1109/IROS.2012.6385476

[52] F. Pomerleau et al., "Noise characterization of depth sensors for surface inspections," in *2nd Int. Conf. Applied Robotics for the Power Industry*, Zurich, Switzerland, 2012, pp. 16-21. doi: 10.1109/CARPI.2012.6473358

[53] Point Cloud Library, ver. 1.6, [Online]. Available: pointclouds.org

[54] K. Essmaeel et al., "Temporal Denoising of Kinect Depth Data," in *8th Int. Conf. Signal Image Technology and Internet Based Systems*, Naples, Italy, 2012, pp. 47-52. doi: 10.1109/SITIS.2012.18

APPENDIX A

DUST SENSOR SYSTEM MICROCONTROLLER SAMPLING CODE

```
// Continuous Sampling Program for Arduino Mega 2560 Microcontroller
// Using Sharp Dust Sensor GP2Y1010AU0F
// Christopher Hall

// The following parameters are set by the user:
const int NumOfSensors = 4;                          // Number of sensors connected to the
                                                        microcontroller
const int SensorPin[NumOfSensors] = {9, 10, 11, 12}; // Input/Output pin numbers used by the sensors
const int StatusLED = 13;                            // There is a built-in LED connected to pin 13

const int NumOfSamples = 50;                         // Number of air samples to take before sending
                                                        back an average reading
const int RestTime = 480;                            // Amount of time (in ms) before restarting
                                                        sample loop

// The following variables are used by the program and should not be changed:
int SensorValue = 0;                                 // Storage for sensor value (number of ADC
                                                        increments)
float Sum[NumOfSensors];                             // Running total for each sensor
float Average = 0;                                   // Calculates average value from running total
float Voltage = 0;                                   // Calculates voltage corresponding to average
                                                        value


int i = 0;                                           // Counter for number of samples taken
int x = 0;                                           // Counter for current sensor being sampled
int LoopDelay;                                       // Delay needed to give each sensor a period of
                                                        10ms


void setup() {                                       // Setup procedure
    Serial.begin(9600);                              // Initialize serial communication

    for (x = 0; x < NumOfSensors; x = x + 1){
        pinMode(SensorPin[x], OUTPUT);               // Initialize the digital pins as outputs:
        Sum[x] = 0;                                  // Initialize array to zero
    }

    pinMode(StatusLED, OUTPUT);                      // Initialize status pin to use LED
    LoopDelay = (10000 - (NumOfSensors * 450));      // Calculates LoopDelay value
}
```

```
void loop() {                                   // Main loop that runs continuously
   digitalWrite(StatusLED, HIGH);               // Turn on status LED to signal sampling has
                                                        started


   for (i = 0; i < NumOfSamples; i = i + 1){    // Loop to run desired number of samples
      for (x = 0; x < NumOfSensors; x = x + 1){ // Loop to run each sensor during sample

         digitalWrite(SensorPin[x], HIGH);      // Turn on sensor's IR LED
         delayMicroseconds(280);                // Delay before sampling (specified by dust
                                                        sensor datasheet)
         SensorValue = analogRead(SensorPin[x]); // Analog return is connected to same pin # as
                                                        digital send
         delayMicroseconds(40);                 // Delay from datasheet (also gives time for
                                                        board to finish sampling)
         digitalWrite(SensorPin[x], LOW);       // Turn off sensor's IR LED
         Sum[x] = Sum[x] + SensorValue;         // Add current reading to running total

      }
      delayMicroseconds(LoopDelay);             // Delay needed for sensors (specified in
                                                        datasheet)
   }

   digitalWrite(StatusLED, LOW);                // Turn off status LED to signal sampling has
                                                        ended


   for (x = 0; x < NumOfSensors; x = x + 1){    // Loop to analyze/print sensor readings

      Average = Sum[x] / i;                     // First find average from running total
      Voltage = Average * 0.00489;              // Find corresponding voltage [10 bit ADC
                                                        measuring 0-5V (5.0 / 1023.0)]
      Serial.print(Voltage);                    // Prints voltage to serial line
      Serial.print("\t");                       // Prints tab to serial line to separate readings
      Sum[x] = 0;                               // Resets running total to zero

   }
   Serial.println();                            // Prints new line to signal end of sensor
                                                        readings
   delay(RestTime);                             // Delay before starting next batch of samples

}
```

APPENDIX B

DUST SENSOR SYSTEM MATLAB GUI CODE

MAIN CODE

```
function varargout = ThesisGUILayout(varargin)
% THESISGUILAYOUT M-file for ThesisGUILayout.fig
%      THESISGUILAYOUT, by itself, creates a new THESISGUILAYOUT or raises
the existing
%      singleton*.
%
%      H = THESISGUILAYOUT returns the handle to a new THESISGUILAYOUT or the
handle to
%      the existing singleton*.
%
%      THESISGUILAYOUT('CALLBACK',hObject,eventData,handles,...) calls the
local
%      function named CALLBACK in THESISGUILAYOUT.M with the given input
arguments.
%
%      THESISGUILAYOUT('Property','Value',...) creates a new THESISGUILAYOUT
or raises the
%      existing singleton*.  Starting from the left, property value pairs are
%      applied to the GUI before ThesisGUILayout_OpeningFcn gets called.  An
%      unrecognized property name or invalid value makes property application
%      stop.  All inputs are passed to ThesisGUILayout_OpeningFcn via
varargin.
%
%      *See GUI Options on GUIDE's Tools menu.  Choose "GUI allows only one
%      instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help ThesisGUILayout

% Last Modified by GUIDE v2.5 24-Jun-2013 04:50:33

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @ThesisGUILayout_OpeningFcn, ...
                   'gui_OutputFcn',  @ThesisGUILayout_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
```

```
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT


% --- Executes just before ThesisGUILayout is made visible.
function ThesisGUILayout_OpeningFcn(hObject, eventdata, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
% varargin   command line arguments to ThesisGUILayout (see VARARGIN)



% Choose default command line output for ThesisGUILayout
handles.output = hObject;

% Update handles structure
guidata(hObject, handles);

% UIWAIT makes ThesisGUILayout wait for user response (see UIRESUME)
% uiwait(handles.figure1);


% --- Outputs from this function are returned to the command line.
function varargout = ThesisGUILayout_OutputFcn(hObject, eventdata, handles)
% varargout  cell array for returning output args (see VARARGOUT);
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure
varargout{1} = handles.output;



% --- Executes on button press in pushbutton1.
function pushbutton1_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


handles.y1values = nan(1,60);
handles.y2values = nan(1,60);
handles.y3values = nan(1,60);
handles.y4values = nan(1,60);
```

```
handles.Record = 0;

cp = char(handles.CurrPort);
%disp(cp);
handles.SerialObj = serial(cp);
handles.SerialObj.BytesAvailableFcnMode = 'terminator';
handles.SerialObj.BytesAvailableFcn = {@SerialDataCallback, hObject};
fopen(handles.SerialObj);
set(handles.ComConnectTextObject,'String','Connected');
set(handles.ComConnectTextObject,'ForegroundColor','green');

%start(handles.t);

% Update handles structure
guidata(hObject, handles);


% --- Executes on button press in pushbutton2.
function pushbutton2_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton2 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

%stop(handles.t);
set(handles.ComConnectTextObject,'String','Disconnected');
set(handles.ComConnectTextObject,'ForegroundColor','red');


fclose(handles.SerialObj);
delete(handles.SerialObj);


% --- Executes on selection change in popupmenu1.
function popupmenu1_Callback(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: contents = get(hObject,'String') returns popupmenu1 contents as cell
array
%        contents{get(hObject,'Value')} returns selected item from popupmenu1

contents = get(hObject,'String');
handles.CurrPort = contents(get(hObject,'Value'));
%disp(handles.CurrPort);


% Update handles structure
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
```

```
function popupmenu1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to popupmenu1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: popupmenu controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
handles.popupmenu1Object = hObject;
guidata(hObject, handles);


% --- Executes on button press in pushbutton4.
function pushbutton4_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton4 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

handles.AvailPorts = scanports;
%disp(handles.AvailPorts);
if (~(strcmp('',handles.AvailPorts)))
    set(handles.popupmenu1Object,'String',handles.AvailPorts);
    set(handles.popupmenu1Object,'Value',1);
    handles.CurrPort = handles.AvailPorts(1);
end

% Update handles structure
guidata(hObject, handles);


% --- Executes when user attempts to close figure1.
function figure1_CloseRequestFcn(hObject, eventdata, handles)
% hObject    handle to figure1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hint: delete(hObject) closes the figure
delete(hObject);


% --- Executes during object creation, after setting all properties.
function text5_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

handles.Sensor1TextObject = hObject;
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function text6_CreateFcn(hObject, eventdata, handles)
```

```
% hObject    handle to text5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

handles.Sensor2TextObject = hObject;
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function text7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

handles.Sensor3TextObject = hObject;
guidata(hObject, handles);

% --- Executes during object creation, after setting all properties.
function text8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

handles.Sensor4TextObject = hObject;
guidata(hObject, handles);


% --- Executes during object deletion, before destroying properties.
function text5_DeleteFcn(hObject, eventdata, handles)
% hObject    handle to text5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)


% --- Executes during object creation, after setting all properties.
function text9_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text9 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

handles.ComConnectTextObject = hObject;
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function axes1_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes1 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes1

handles.Sensor1PlotObject = hObject;
guidata(hObject, handles);
```

```
% --- Executes during object creation, after setting all properties.
function axes6_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes6

handles.Sensor2PlotObject = hObject;
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function axes7_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes7 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes7

handles.Sensor3PlotObject = hObject;
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function axes8_CreateFcn(hObject, eventdata, handles)
% hObject    handle to axes8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: place code in OpeningFcn to populate axes8

handles.Sensor4PlotObject = hObject;
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function text10_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text10 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

handles.FilePathTextObject = hObject;
guidata(hObject, handles);


% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton5 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)
```

```
FolderName = uigetdir;
handles.SavePath = FolderName;
set(handles.FilePathTextObject,'String',handles.SavePath);
guidata(hObject, handles);


% --- Executes on button press in pushbutton6.
function pushbutton6_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton6 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

handles.FramesRec = 0;
File = strcat(handles.SavePath,'\DataValues.txt');
handles.FileID = fopen(File,'a');
if handles.FileID ~= -1
    handles.Record = 1;
    set(handles.FileOpenTextObject,'String','File Open');
    set(handles.FileOpenTextObject,'ForegroundColor','green');
end
guidata(hObject, handles);


% --- Executes on button press in pushbutton8.
function pushbutton8_Callback(hObject, eventdata, handles)
% hObject    handle to pushbutton8 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

fclose(handles.FileID);
handles.Record = 0;
set(handles.FileOpenTextObject,'String','File Not Open');
set(handles.FileOpenTextObject,'ForegroundColor','red');
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function text11_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text11 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

handles.FileOpenTextObject = hObject;
guidata(hObject, handles);


% --- Executes during object creation, after setting all properties.
function text12_CreateFcn(hObject, eventdata, handles)
% hObject    handle to text12 (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

handles.FramesRecTextObject = hObject;
guidata(hObject, handles);
```

SERIALDATACALLBACK FUNCTION

```
function SerialDataCallback(obj,event,hObject)
%Handles the terminator char callback
handles = guidata(hObject);

t = event.Data.AbsTime;
handles.timestamp = datestr(t);
handles.values = fscanf(obj);

%out = [handles.timestamp, sprintf('\t'), handles.values];
%disp(out);
A = sscanf(handles.values, '%f');

set(handles.Sensor1TextObject,'String',num2str(A(1)));
set(handles.Sensor2TextObject,'String',num2str(A(2)));
set(handles.Sensor3TextObject,'String',num2str(A(3)));
set(handles.Sensor4TextObject,'String',num2str(A(4)));

handles.y1values = [A(1) handles.y1values(1:59)];
handles.y2values = [A(2) handles.y2values(1:59)];
handles.y3values = [A(3) handles.y3values(1:59)];
handles.y4values = [A(4) handles.y4values(1:59)];

plot(handles.Sensor1PlotObject,handles.y1values);
plot(handles.Sensor2PlotObject,handles.y2values);
plot(handles.Sensor3PlotObject,handles.y3values);
plot(handles.Sensor4PlotObject,handles.y4values);

if (handles.Record == 1)
    handles.FramesRec = handles.FramesRec + 1;
    SFrame = num2str(handles.FramesRec);
    SFrameR = strcat('Frame number:  ',SFrame);

fprintf(handles.FileID,'%s\r\n',SFrameR,handles.timestamp,handles.values);
    set(handles.FramesRecTextObject,'String',SFrame);
    handles.trigger = 1;
end

% Update handles structure
guidata(hObject,handles);
```

SCANPORTS FUNCTION

```
function portlist = scanports(portlimit)
% Function that scans for available serial ports.
% Returns cell array of string identifiers for available ports.
% Scans from COM1 - COM15 unless portlimit is specified,
% then scans from COM1 - COM[portlimit].
% version 1 by Robert Slazas, October 2011

% check for existing serial connections and close them
if ~isempty(instrfind)
    fclose(instrfind);
    delete(instrfind);
end

% set portlimit if not specified
if nargin < 1
    portlimit = 15;
end
portlist = cell(0);

h = waitbar(0,'Scanning Serial Ports...','Name','Scanning Serial Ports...');
for i = 1:portlimit
    eval(['s = serial(''COM',num2str(i),''');']);
    try
        fopen(s);
        fclose(s);
        delete(s);
        portlist{end+1,1}=['COM',num2str(i)];
        waitbar(i/portlimit,h,['Found ',num2str(numel(portlist)),' COM
Ports...']);
    catch
        delete(s);
        waitbar(i/portlimit,h);
    end
end
close(h);
```

APPENDIX C

PCD_WRITER – DEPTH IMAGE RECORDING CODE

```
#include <pcl/io/openni_grabber.h>
#include <pcl/visualization/cloud_viewer.h>
#include <pcl/io/pcd_io.h>
#include <string.h>
#include <boost/lexical_cast.hpp>
#include <stdio.h>

//#include <pcl/io/openni_camera/openni_image.h>


class SimpleOpenNIViewer
{
public:
	SimpleOpenNIViewer () : viewer ("Kinect Depth Image")
	{
		interface = new pcl::OpenNIGrabber();

		boost::function<void (const pcl::PointCloud<pcl::PointXYZ>::ConstPtr&)> f =
			boost::bind (&SimpleOpenNIViewer::cloud_cb_, this, _1);

		//		boost::function<void (openni_wrapper::Image&)> g =
		//		    boost::bind (&SimpleOpenNIViewer::RGB_cb_, this, _1);

		//const boost::shared_ptr<openni_wrapper::Image>&
		// const boost::shared_ptr<const pcl::PointCloud<pcl::PointXYZ> >&

		interface->registerCallback (f);
		// interface->registerCallback (g);

		counter = 1;
		record = false;
		RGBrecord = false;
		display = false;
		RGBdisplay = false;
		path = "";
	}


	void start()
	{
		interface->start ();
	}

	void stop()
	{
		interface->stop();
	}
```

```
	int getCounter()
	{
		return counter;
	}

	void setCounter(int var)
	{
		counter = var;
	}

	void setDisplay()
	{
		display = true;
//		RGBdisplay = true;
	}

	void setRecord()
	{
		record = true;
//		RGBrecord = true;
	}

	void setPath(std::string P)
	{
		path = P;
	}


private:
	void cloud_cb_ (const pcl::PointCloud<pcl::PointXYZ>::ConstPtr &cloud)
	{
		if (record)
		{
			filename = path + "DepthImage_" +
boost::lexical_cast<std::string>(counter) + ".pcd";
			pcl::io::savePCDFileBinary(filename, *cloud);
			cout << "Saving " << filename << endl;
			record = false;
		}
		if (display)
		{
			cout << "Displaying " << counter << endl;
			viewer.showCloud (cloud);
			display = false;
		}
	}

	void RGB_cb_ (openni_wrapper::Image::ConstPtr &image)
	{
		if (RGBrecord)
		{
			cout << "RGB record" << endl;
			RGBrecord = false;
		}
		if (RGBdisplay)
		{
			cout << "RGB display" << endl;
```

```
                RGBdisplay = false;
            }
        }

        pcl::visualization::CloudViewer viewer;
        pcl::OpenNIGrabber* interface;
        int counter;
        std::string filename;
        bool display;
        bool RGBdisplay;
        bool record;
        bool RGBrecord;
        std::string path;

};

int main (int argc, char* argv[])
{
        if (argc != 3) {
                cout << "Error: program needs 2 parameters!" << endl;
                cout << "1) Record Path  2) Number of Frames" << endl;
                return 1;
        }

        SimpleOpenNIViewer v;
        v.setPath(argv[1]);
        v.start();

        int temp = boost::lexical_cast<int>(argv[2]);
        int temp2 = 0;
        cout << "Previewing...  ";
        v.setDisplay();
        boost::this_thread::sleep (boost::posix_time::seconds (2));
        cout << "Start recording in ";

        for (int x=5; x>0; x--)
        {
                cout << x << "... ";
                boost::this_thread::sleep (boost::posix_time::seconds (1));
        }
        cout << endl;

        while (temp > 0){
                while (v.getCounter() <= temp)
                {
                        v.setRecord();
                        if ((v.getCounter() % 10) == 0)
                        {
                                v.setDisplay();
                        }
                        boost::this_thread::sleep (boost::posix_time::seconds (1));
                        v.setCounter(v.getCounter()+1);
                }

                cout << "Finished" << endl;

                cout << "Enter 0 to exit or a number of frames to keep recording: ";
                cin >> temp2;
```

```
                if (temp2 == 0)
                {
                        temp = 0;
                }
                else
                {
                        temp = temp + temp2;
                }

        }


        v.stop();

        return 0;
}
```

APPENDIX D

PCD_READER – DEPTH IMAGE VIEWING CODE

```
#include <pcl/io/openni_grabber.h>
#include <pcl/visualization/cloud_viewer.h>
#include <pcl/io/pcd_io.h>
#include <string.h>



 int main (int argc, char *argv[])
 {
        if (argc != 3) {
               cout << "Error: program needs 2 parameters!" << endl << "Target file and
axis scale" << endl;
               return 1;
        }

        pcl::PointCloud<pcl::PointXYZ>::Ptr cloud (new pcl::PointCloud<pcl::PointXYZ>);
        pcl::visualization::PCLVisualizer viewer;

        if (pcl::io::loadPCDFile<pcl::PointXYZ> (argv[1], *cloud) == -1) //* load the
file
        {
               PCL_ERROR ("Couldn't read file %s \n", argv[1]);
               return(-1);
        }

        float CS = atof(argv[2]);

        viewer.addPointCloud<pcl::PointXYZ>(cloud);
        viewer.addCoordinateSystem(CS);

        viewer.spin();


   return 0;
 }
```

APPENDIX E

PCD_CONVERTER – LIDAR TO KINECT FORMAT CONVERSION CODE

```
#include <fstream>
#include <iostream>
#include <string>
#include <sstream>

using namespace std;



int main(int argc, char * argv[])
{
	if (argc != 4)
	{
		cerr << "Expecting <filename> argument, min distance (in mm), & max
distance (in mm)" << endl;
		return 1;
	}

	ofstream output ("Modified.pcd");
	if (!output.is_open())
	{
		cerr << "Unable to open output file" << endl;
		return 1;
	}

	ifstream input(argv[1]);
	if (!input.good())
	{
		cerr << "Unable to open input file " << argv[1] << endl;
		return 1;
	}

	string line;
	float value;
	float IX, IY, IZ;
	float OX, OY, OZ;
	int lineno = 1;
	int stored = 0;
	int MinDist = atoi(argv[2]);
	int MaxDist = atoi(argv[3]);

	while (input.good())
	{
		getline(input, line);
//		istringstream iss(line);
//		iss >> value;
```

```
		if (input.good())
		{
			if (lineno > 10)
			{
				istringstream iss(line);
				iss >> IX;
				iss >> IY;
				iss >> IZ;

				if ((IX > MinDist) && (IX < MaxDist) && (IY < MaxDist) && (IY
> -MaxDist) && (IZ < MaxDist) && (IZ > -MaxDist))
				{

					OX = IX/1000;
					OY = IY/1000;
					OZ = IZ/1000;

//					output << OX << ' ' << OY << ' ' << OZ << endl;
//					output << IX << ' ' << IY << ' ' << IZ << endl;

					output << -(OY) << ' ' << -(OZ) << ' ' << OX << endl;

					stored++;
				}

			}
			else
			{
				output << line << endl;
			}
//			process(value);
		}
		else
		{
			if (input.eof())
			{
				break;
			}
			cerr << "Invalid data on line " << lineno << endl;
			input.clear();
		}
		++lineno;

	}

	output.close();
	cout << "Processed " << (lineno - 11) << " points" << endl;
	cout << "Stored " << stored << " points" << endl;
	cout << "NOTE: You MUST manually change WIDTH and POINTS in Modified.pcd to match
points stored!" << endl;
	return 0;
}
```

APPENDIX F

PCD_ANALYZER – DEPTH IMAGE ANALYSIS CODE

```
#include <pcl/io/openni_grabber.h>
//#include <pcl/visualization/cloud_viewer.h>
#include <pcl/io/pcd_io.h>
#include <string.h>
#include <boost/lexical_cast.hpp>

#include <pcl/visualization/pcl_visualizer.h>

#include <iostream>
#include <pcl/ModelCoefficients.h>
#include <pcl/point_types.h>
#include <pcl/sample_consensus/method_types.h>
#include <pcl/sample_consensus/model_types.h>
#include <pcl/segmentation/sac_segmentation.h>
#include <pcl/filters/extract_indices.h>

#include <sstream>
#include <fstream>

using namespace std;

//#include <pcl/sample_consensus/sac_model_normal_parallel_plane.h>



 int main (int argc, char *argv[])
 {
        system("CLS");

        Eigen::Vector3f axis;
        axis[0] = 1.0;
        axis[1] = 0.0;
        axis[2] = 0.0;

        double DistThresh = 0.1;
        double PlanePointMin = 1000000.0;
        double PlanePointMax = -1000000.0;
        double PlanePointSum = 0.0;
        double PlanePointAvg = 0.0;
        double UnitConv = 0.0;

        pcl::PointCloud<pcl::PointXYZ>::Ptr cloud (new pcl::PointCloud<pcl::PointXYZ>);
        pcl::PointCloud<pcl::PointXYZ>::Ptr cloud2 (new pcl::PointCloud<pcl::PointXYZ>);
        pcl::PointCloud<pcl::PointXYZ>::Ptr cloud3 (new pcl::PointCloud<pcl::PointXYZ>);
```

```
	if (pcl::io::loadPCDFile<pcl::PointXYZ> (argv[1], *cloud) == -1) //* load the
file
	{
		PCL_ERROR ("Couldn't read file %s \n", argv[1]);
		return(-1);
	}


	pcl::ModelCoefficients::Ptr coefficients (new pcl::ModelCoefficients ());
	pcl::PointIndices::Ptr inliers (new pcl::PointIndices ());
	// Create the segmentation object
	pcl::SACSegmentation<pcl::PointXYZ> seg;
//	pcl::SACSegmentationFromNormals<pcl::PointXYZ, pcl::PointXYZ> seg;
	// Optional
//	seg.setOptimizeCoefficients (true);
	// Mandatory
//	seg.setModelType (pcl::SACMODEL_NORMAL_PARALLEL_PLANE);



	seg.setModelType (pcl::SACMODEL_PERPENDICULAR_PLANE);
//	seg.setModelType (pcl::SACMODEL_PLANE);
	seg.setMethodType (pcl::SAC_RANSAC);
	seg.setMaxIterations (1000);
//	seg.setDistanceThreshold (DistThresh);

//	seg.setAxis(axis);
//	seg.setEpsAngle(0.349);


	// Create the filtering object
	pcl::ExtractIndices<pcl::PointXYZ> extract;


	pcl::visualization::PCLVisualizer viewer("PCD Analyzer");

	int v1(0);
	viewer.createViewPort(0.0, 0.0, 0.34, 1.0, v1);
	viewer.setBackgroundColor(0, 0, 0, v1);
	viewer.addText("Original", 10, 10, "v1 text", v1);
	viewer.addPointCloud<pcl::PointXYZ>(cloud, "sample cloud", v1);

	int v2(0);
	viewer.createViewPort(0.34, 0.0, 0.67, 1.0, v2);
	viewer.setBackgroundColor(0.1, 0.1, 0.1, v2);
	viewer.addText("Plane", 10, 10, "v2 text", v2);
	viewer.addPointCloud<pcl::PointXYZ>(cloud2, "second cloud", v2);

	int v3(0);
	viewer.createViewPort(0.67, 0.0, 1.0, 1.0, v3);
	viewer.setBackgroundColor(0, 0, 0, v3);
	viewer.addText("Remaining", 10, 10, "v3 text", v3);
	viewer.addPointCloud<pcl::PointXYZ>(cloud3, "third cloud", v3);

	viewer.resetCameraViewpoint("sample cloud");
	viewer.addCoordinateSystem(1.0);
```

```
        viewer.spinOnce(5000);


        int loop = 0, command = 2, NewAxis = 0;
        string name;
        stringstream ss;
        ofstream output;
        float EpsAngleD = 0.0;
        float EpsAngleR = 0.0;

        while (1)
        {
               switch (command){
                   case 1:
                               cout << "PointCloud representing the planar component: " <<
cloud2->width * cloud2->height << " data points." << endl;
                               cout << "Using a threshold of: " << DistThresh << endl;
                               cout << "And an angle of: " << EpsAngleD << " degrees." <<
endl;
                               viewer.spinOnce(5000);
                               break;

                       case 2:
                               loop++;

                               cout << "Select an axis:\t 1) X" << endl;
                               cout << "\t\t 2) Y" << endl;
                               cout << "\t\t 3) Z" << endl;
                               cin >> NewAxis;
                               cout << endl;

                               switch (NewAxis){
                                       case 1:
                                               axis[0] = 1.0;
                                               axis[1] = 0.0;
                                               axis[2] = 0.0;
                                               seg.setAxis(axis);
                                               break;
                                       case 2:
                                               axis[0] = 0.0;
                                               axis[1] = 1.0;
                                               axis[2] = 0.0;
                                               seg.setAxis(axis);
                                               break;
                                       case 3:
                                               axis[0] = 0.0;
                                               axis[1] = 0.0;
                                               axis[2] = 1.0;
                                               seg.setAxis(axis);
                                               break;
                                       default:
                                               cout << "Invalid choice. Defaulting to Z axis"
<< endl;
                                               axis[0] = 0.0;
                                               axis[1] = 0.0;
                                               axis[2] = 1.0;
                                               seg.setAxis(axis);
                                               break;
```

```
                        }

                    case 3:

                        cout << "Enter distance threshold value: ";
                        cin >> DistThresh;
                        cout << endl;
                        seg.setDistanceThreshold (DistThresh);

                        cout << "Enter plane variance angle (in deg): ";
                        cin >> EpsAngleD;
                        cout << endl;
                        EpsAngleR = (EpsAngleD * 0.01745);
                        seg.setEpsAngle(EpsAngleR);

                        // Segment the largest planar component from the remaining
cloud
                        seg.setInputCloud (cloud);
                        seg.segment (*inliers, *coefficients);

                        if (inliers->indices.size () == 0)
                        {
                            std::cerr << "Could not estimate a planar model for
the given dataset." << std::endl;
                            break;
                        }

                        // Extract the inliers
                        extract.setInputCloud (cloud);
                        extract.setIndices (inliers);
                        extract.setNegative (false);
                        extract.filter (*cloud2);

                        std::cerr << "PointCloud representing the planar component: "
<< cloud2->width * cloud2->height << " data points." << std::endl;
//                      cout << "Using a threshold of: " << DistThresh << endl;
                        //            viewer.addPointCloud<pcl::PointXYZ>(cloud2,
"second cloud", v2);
                        viewer.updatePointCloud(cloud2,"second cloud");

                        // Create the filtering object
                        extract.setNegative (true);
                        extract.filter (*cloud3);
                        viewer.updatePointCloud(cloud3,"third cloud");

                        viewer.spinOnce();

                        break;

                    case 4:

                        std::cerr << "PointCloud representing the planar component: "
<< cloud2->width * cloud2->height << " data points." << std::endl;
                        cout << "Using a threshold of: " << DistThresh << endl;
                        cout << "And an angle of: " << EpsAngleD << " degrees." <<
endl;

                        PlanePointMin = 1000000.0;
```

```
                        PlanePointMax = -1000000.0;
                        PlanePointSum = 0.0;

                        for (size_t i = 0; i < cloud2->points.size(); ++i)
                        {
                                if (cloud2->points[i].z < PlanePointMin)
                                {
                                        PlanePointMin = cloud2->points[i].z;
                                }

                                if (cloud2->points[i].z > PlanePointMax)
                                {
                                        PlanePointMax = cloud2->points[i].z;
                                }

                                PlanePointSum += cloud2->points[i].z;
                        }

                        PlanePointAvg = (PlanePointSum / cloud2->points.size());

                        UnitConv = PlanePointMin / 0.3048;
                        cout << endl << "Minimum distance: " << UnitConv << " ft" <<
endl;


                        UnitConv = PlanePointMax / 0.3048;
                        cout << "Maximum distance: " << UnitConv << " ft" << endl;

                        UnitConv = PlanePointAvg / 0.3048;
                        cout << "Average distance: " << UnitConv << " ft" << endl;

                        break;

                case 5:
                        ss.clear();
                        ss.str("");
                        ss << argv[1] << "Plane" << loop << "DistanceValues.txt";
                        name = ss.str();

                        output.open(name);
                        if (!output.is_open())
                        {
                                cerr << "Unable to open output file" << endl;
                                break;
                        }

                        cout << "Exporting " << cloud2->points.size() << " points
to:" << endl;
                        cout << name << endl;

                        for (size_t i = 0; i < cloud2->points.size(); ++i)
                        {
                                output << cloud2->points[i].z << endl;

                        }

                        output.close();

                        break;
```

```
                        case 6:
                                system("CLS");
                                return 0;
                                break;

                        default:
                                cout << "Not a command. Try again" << endl;
                                break;
                }



                cout << endl << "Enter command:\t 1) Refresh viewer" << endl;
                cout << "\t\t 2) Continue to next plane" << endl;
                cout << "\t\t 3) Recalculate current plane with new parameters" << endl;
                cout << "\t\t 4) Calculate distance to current plane" << endl;
                cout << "\t\t 5) Export all current plane distance values to text file" <<
endl;
                cout << "\t\t 6) Exit program" << endl;
                cin >> command;

                if (command == 2)
                {
                        cloud.swap (cloud3);
                }


                system("CLS");


        }


   return 0;
 }
```