LIDAR Data Retrieval and Processing for the Purpose of Obstacle Detection

Philip Schlesinger
Theresia Olson
Bayan Towfiq
Jordan Levy

Professor Ian Harris

Donald Bren School of Information and Computer Sciences

Fall 2004 Quarter
ICS 180 – Special Topics in Computer Science
UCI 2005 DARPA Grand Challenge Team
Sensor Fusion and Obstacle Detection Group

Table of Contents

Section 1: Introduction 3
Current Day 3
Section 2: Overall View Of An Autonomous Vehicle 5
Section 3: Granting the Autonomous Vehicle Sight 7
3.1 Sensor Evaluation 7
3.2 LIDAR Sub-Group Software Architecture 8
3.3 Choosing a LIDAR to Evaluate 9
3.4 Other LIDAR Options 11
Section 4: Difficulties To Address 13
4.1: LIDAR Position And Orientation On The Robot 13
4.1.1 Knowledge Gained 13
4.2: Position And Orientation of Robot With Respect To The World 14
4.3: Blind Spots 15
4.3.1: Possible Solutions 15
4.4: False Readings 16
4.4.1 False Positives 16
4.4.2 False Negatives 17
4.4.3 Possible Solutions 17
Section 5: Implementation 19
5.1 LIDAR Data Stream Capture Program 19
5.2 Convert to Cartesian Program 21
5.3 Obstacle Detection Program 22
Section 6: Gathering Data Sets 24
6.1 Physical Setup Of The Tests 24
6.2 Results Of The Tests 26
6.3 Analysis Of The Results 34
Section 7: Task Schedule 36
Appendix A: Glossary 37
Appendix B: Relevant Documentation 38

Section 1: Introduction

If the visions of the science fiction writers of the early 20th century were accurate, the world would be farther along in technological advancement than what we see today:

1) We would be able to travel point-to-point on the Earth in short periods of time,

2) We would be living on the moon and the other planets of the solar system,

3) Electronics would be assisting us in our daily lives, and

4) We would all own cars that can drive themselves

The Reality Check (respectively)

1) Progress has been made: A few weeks ago, a team completed the two private spaceflights required for the X-Prize competition, which will open up quick intra-planetary transport.

2) Progress has been made: We have been to the moon and back several times. We have explored the other planets and their moons (most recent of note: the Cassini spacecraft sent back pictures of Titan).

3) Progress has been made: The ubiquitousness of PDAs, cell phones, and digital cameras speak for themselves. The Roomba can vacuum floors without help. Robots have even been trained to drive around the house and fetch objects. C3PO & R2D2 are not far off.

4) However, cars cannot drive themselves. The closest we have gotten is a GPS barking orders at the driver of a car: “turn left!”, “exit approaching in 0.5 miles!”, and so on.

Current Day

The Defense Advanced Research Projects Agency (DARPA), the research and development arm of the United States military, has been researching autonomous vehicles, for two reasons:

1) To remove the need for a human to drive a supply shipment through dangerous territory (and thus remove the opportunity for an enemy to kidnap or kill one of our own people), and

2) To give a wounded soldier a quick and safe ride home in the event he/she is too injured to drive.

Unfortunately, they too ran into the same problems. So, instead of relying on the standard military subcontractors, and to spur innovation, they sought and received congressional backing for opening this difficult engineering problem to the private sector. Thus, the “DARPA Grand Challenge” – building an autonomous robot that can cover a 175-mile stretch of the Mojave Desert in 10 hours or less while driving safely.

Section 2: Overall View Of An Autonomous Vehicle

Autonomous vehicles, in general, can be described as having three different areas of focus in a pipe-and-filter system architecture:

Perception → Decision → Action[1]

The Perception module tries to sense the physical environment around the vehicle. The Decision module takes those sensations and cogitates the next move. The Action module interfaces with the physical environment and moves the vehicle.

Our team of 30 students and 3 professors modified this concept slightly:

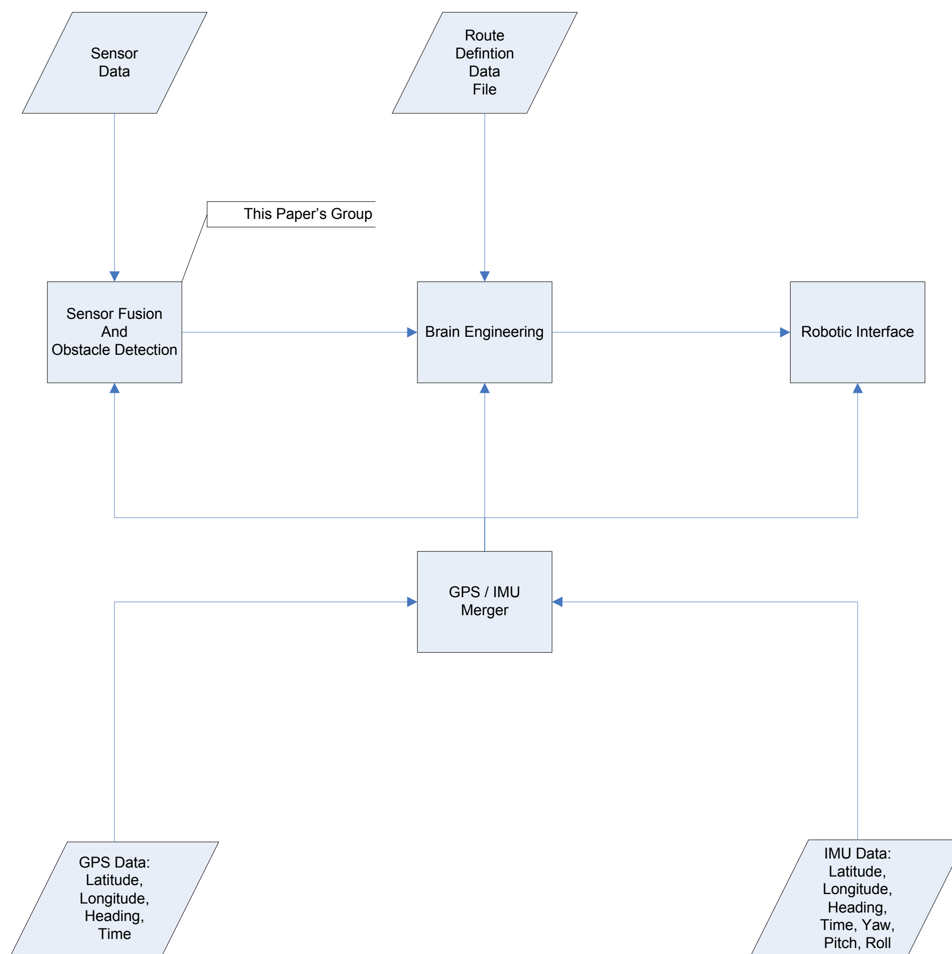


Figure 2-1: High-Level Architecture

[1] Paraphrase of: http://www.frc.ri.cmu.edu/~alonzo/course/course.html (Carnegie Mellon University's Intro to Mobile Robotics course)

To summarize Figure 2-1:

- The Sensor Fusion And Obstacle Detection group handles Perception, sensing the environment and identifying obstacles. Professor Ian Harris advised this group.
- The Brain Engineering group handles Decision-making, handling navigation and guidance. Professor Crista Lopes advised this group.
- The Robotics group would then Act. In addition, the Robotics group also took on the role of GPS & IMU research. Professor Tony Givargis advised this group.

Section 3: Granting the Autonomous Vehicle Sight

Elaborating on the above description for Figure 2-1, the goal of the Sensor Fusion and Obstacle Detection group was to:

1) Evaluate sensors that could potentially give the vehicle "sight".
2) Choose the minimum necessary sensors to give the vehicle enough of a view of the world to detect obstacles large enough to cause problems for the vehicle within the necessary period of time for avoidance at speeds up to 30 mph.
3) Tap the data stream from the sensor or sensors.
4) If using multiple sensors, merge the sensor data streams together to create a unified image.
5) From that unified image, detect obstacles and rate their level of difficulty to surmount.
6) Finally, send a continually updating list of known obstacles and difficulty levels to the Brain Engineering group.

3.1 Sensor Evaluation

The teams that entered the March 2004 race used at least one of the following sensor types for robotic "vision", and for each chosen sensor type, at least one sensor:

- Computer-connected video camera
- LIDAR
- SONAR / Ultrasound
- RADAR
- Infrared camera

Our sub-group was put in charge of LIDAR evaluation. The basic method behind a LIDAR is as follows:

- A mirror rotating at high speeds sends out pulses of light in various directions at small, equal increments (incrementation adjusted by the frequency and timing of the pulsing of a LASER).
- Those pulses either reflect off of a substance, or disappear into the ether.
- A LASER detector then looks for the return of the pulse, and based on the return time, calculates an approximate distance from the LIDAR to the surface of reflection.

LIDAR is one of the easier sensors to use for environmental perception. It scans an arc and reports back distances at specific points, incremented at a specific degree value or fraction of a degree value. Unlike vision, the LIDAR reports whether it sees something, or not. Due to its inherent design, it is very hard to trick it into giving false positives or negatives.

As the LIDAR sub-group, our goals for the Fall 2004 Quarter were as follows:

1) Arrange for a LIDAR evaluation unit.
2) Gain access to the data stream of the LIDAR through a self-developed or adapted existing program and record the raw sensor data to a file.

3) Develop an algorithm to read the recorded data file and pick out the objects.
4) Mount the LIDAR on a car and record online multiple sample data streams in a controlled movement scenario
5) Process offline those recorded sample data streams through the developed obstacle detection into a list of obstacles and determined level of difficulty for each obstacle.

3.2 LIDAR Sub-Group Software Architecture

Our general architecture for converting the LIDAR data stream to a list of obstacles:

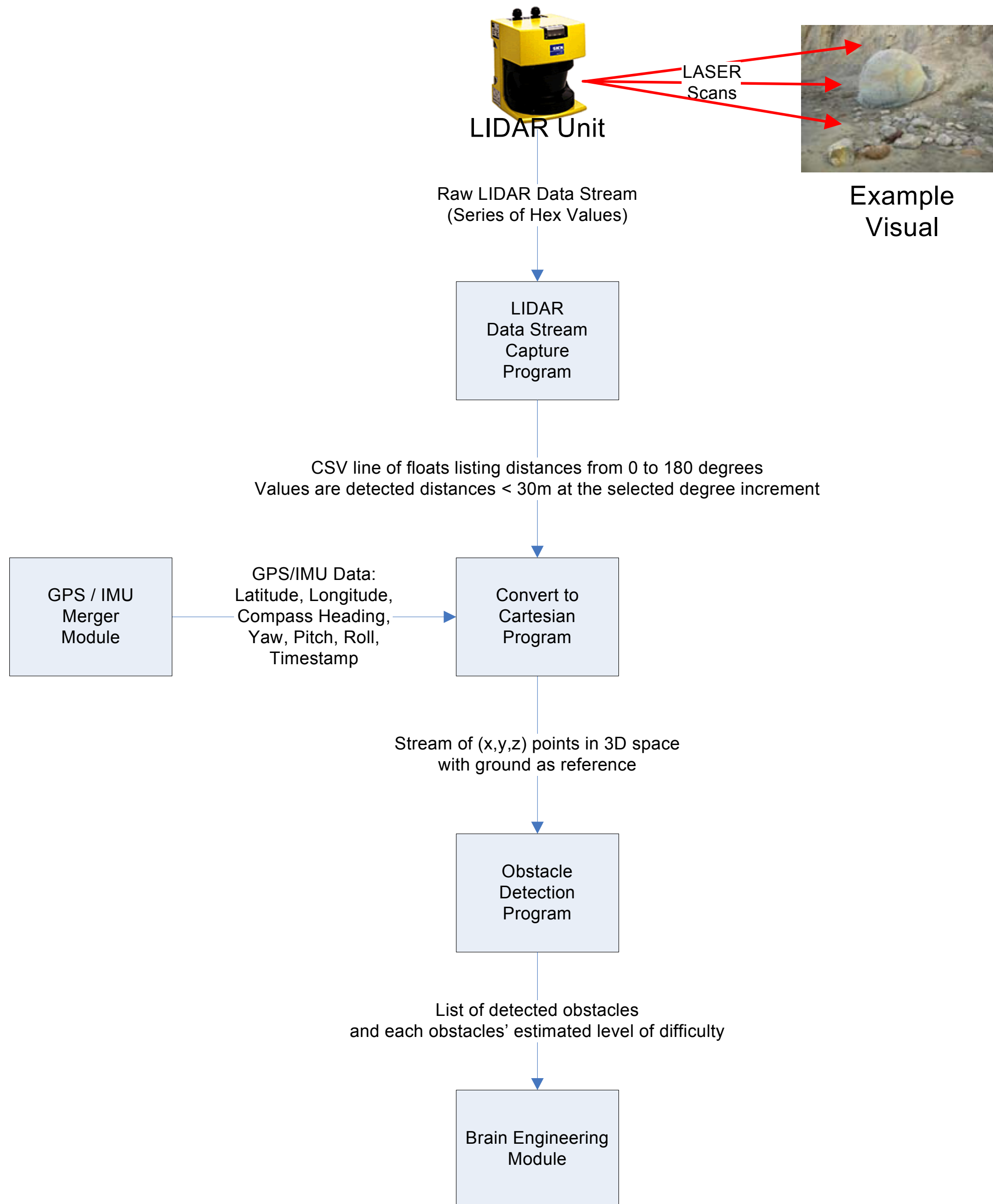


Figure 3-1: LIDAR & Obstacle Detection Sub-Architecture

To summarize Figure 3-1:

1) The LIDAR Data Stream Capture program receives a line of data from the LIDAR, which shows the detected distances from the LIDAR lens seen in a scanned line, and convert it to CSV.
2) Next, the Convert to Cartesian program combines the CSV and the GPS/IMU data (showing current position, heading, and orientation relative to the Earth) to calculate Cartesian points in 3D space (x,y,z) for the positive signals reported by the LIDAR. This is done through standard trigonometric calculations and some linear algebra.
3) We assume that new data is more accurate than old data, so if a point in space changes status from positive to negative (or vice versa), the old perspective on that point is thrown out.
4) The Obstacle Detection Program then receives this list of points in 3D space and looks for patterns of persistent positive signal returns, creating a list of detected obstacles and the expected difficulty level for each obstacle.
5) That list of detected obstacles is sent to the Brain Engineering group.

3.3 Choosing a LIDAR to Evaluate

Most of the teams last year used a brand of LIDAR made by the company German company SICK. SICK has two model lines for LIDAR imaging have fit our needs in this initial research (shown in Figure 3-2):

- PLS (Proximity Laser Sensor) – to be used indoors, and
- LMS (Laser Measurement Sensor) – to be used indoors or outdoors

Figure 3-2: PLS on left[2], LMS on right[3]

SICK's LIDARs (both PLS & LMS) relevant technical details are as follows:

- Power requirements: 24V DC
- Serial communications:
    - Interface:
        - RS-232
        - RS-422
        - RS-485

[2] http://www.sickusa.com/Publish/docroot/Product%20Image/pls101Pic.gif
[3] http://www.sickusa.com/Publish/docroot/Product%20Image/lms291Pic.gif

- Data speeds:
  - Partial set of data:
    - 9,600 bps
    - 19,200 bps
    - 38,400 bps
    - 56,000 bps
  - Full set of data:
    - 500,000 bps

- Laser viewing limitations:
  - Scans a line at a time in the near-infrared range (invisible to naked eye, even with chalk dust)
  - Max distance in perfect conditions: 50 meters
  - Realistic distance in fog: 30 meters
  - Scan view:
    - 100 degrees
    - 180 degrees
  - Increments:
    - 1.0 degrees
    - 0.5 degrees
    - 0.25 degrees
  - Time per scan:
    - 180 degree scan view, 1.0 degree increment: ~13 milliseconds
    - 180 degree scan view, 0.25 degree increment: ~53 milliseconds
- Software:
  - SICK-issued and supported Windows software:
    - For the PLS, SICK has issued one software program specifically to set up the warning and emergency proximity areas, which while key to the intention of the PLS, are not necessary to our efforts for data stream capture. This PLS software does not support data stream exporting.
    - For the LMS, SICK has issued two software programs. One of those programs, the MST (Measurement Software Tool), allows a computer to directly access the data stream.
  - Various SICK-unsupported Open Source software is available (most come from Carnegie Mellon University)
- Security:
  - In order to interact with the LIDAR, one must login to the LIDAR as an authorized client. Passwords:
    - PLS: "SICK_PLS"
    - LMS: "SICK_LMS".

We were informed by Gary Clevenger of J & J Electronics[4] (the local SICK distributor) that in order to get sub-1.0 degree incrementation, the LIDAR scans in the following fashion (if 0.25-degree incrementation overall):

[4] http://www.jandjelectronics.com/

- First, it scans 0.0 to 180.0 degrees with 1.0-degree incrementation.
- Then, it offsets itself by 0.25 degrees, and scans 0.25 to 179.25 degrees with 1.0-degree incrementation.
- Next, it offsets itself by another 0.25 degrees, and scans 0.5 to 179.5 degrees with 1.0-degree incrementation.
- Finally, it offsets itself by another 0.25 degrees, and scans 0.75 to 179.75 degrees with 1.0-degree incrementation.

Our experimentation with Carnegie Mellon University's CARMEN Carnegie Mellon Robot Navigation Toolkit[5] - a C-based package of various libraries and functions - determined that when the LIDAR scans in at 0.5-degree or 0.25 degree incrementation, it appears to buffer, respectively, both of the two 0.5-degree increment scans or all four 0.25 degree increment scan lines, and then transmits a full set of distance reports. To give an example, if using 0.25-degree incrementation, the LIDAR outputs the distance values at degree 0, 0.25, 0.75, 1.0, 1.25, 1.5, 1.75, and so on.

As the LMS was only available via a 30-day evaluation agreement, we opted to do the majority of our research with a PLS that Mr. Clevenger graciously permitted us to use during our testing.

3.4 Other LIDAR Options

We also investigated Riegl's[6] 2D LIDAR products, which were used by CMU for long-distance scanning (75 meter with a 60 degree horizontal view)[7]. The Riegl LIDAR costs $27,000 after educational discount and the Riegl representative stated that demo units are not offered.

We also attempted to find out about LaserOptronix[8], which were used by a team in the March qualifying round. However, they are based in Sweden, so without the appropriate level of funding, contacting them by phone is out of the question. We attempted to email them, but the email bounced back. Hence, with the SICK LIDAR readily available, we are holding off on LaserOptronix until we gain the necessary funds to purchase an appropriate amount of international phone time.

---

[5] http://www-2.cs.cmu.edu/~carmen/
[6] http://www.riegl.com/
[7] Video examples of Riegl's capabilities are located at http://redteamracing.org/media/Video/Quicktime/Point_Cloud.mov and http://redteamracing.org/media/Video/wmv/laser_point_cloud.WMV
[8] http://www.laseroptronix.se/

Section 4: Difficulties To Address

Our software project required us to consider and adjust for four different difficulties:

1) Appropriate position and orientation of the LIDAR on the robot
2) Accurate position and orientation of the robots in respect to the world
3) Blind spots
4) False positives and negatives

4.1: LIDAR Position And Orientation On The Robot

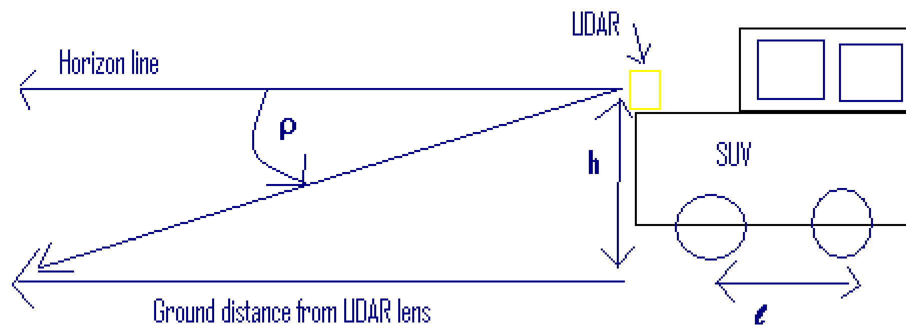


Figure 4-1: Variables for LIDAR interpolation for maximum ground distance

Figure 4-1 shows the variables necessary to accurately calculate the maximum forward ground distance validly reportable on a flat surface if the LIDAR was mounted on the edge of the front of the hood of a car. We calculated that the maximum ground distance would be:

$$\text{Maximum ground distance} = \cot(\rho)$$

With this in mind, given the following:

- The lens of the LIDAR is
  - 1 meter vertical distance above the ground ($h = 1$)
  - Pointed down at an angle of 6 degrees with respect to the horizon ($\rho = 6$)
- The car has a 3 meter wheel base ($\ell = 3$)

The farthest flat ground distance it could theoretically detect with this formula is 9.5 meters. Any reports greater than 9.5 degrees would mean the LIDAR scanned a depression or hole in the ground; any reports less than 9.5 degrees would mean the LIDAR found an object that is closer than 9.5 degrees. Take this calculation and one could create a 3D image (with minimum heights) of what the LIDAR reported.

If we altered this situation by decreasing ρ to 5, then the readings extend almost 2 meters, giving us a maximum ground distance of 11.4 meters.

4.1.1 Knowledge Gained

To see farther, the LIDAR must be mounted higher. However, the 2005 Rules of the DARPA Grand Challenge[9] state that the vehicle must be able to go through an underpass of minimum 9 feet, so we need to watch how high we mount the LIDAR.

One possible way to reduce our chances of scraping a roof-mounted SICK LIDAR may be to turn it upside down. Going back to Figure 3-2, the lens is on the bottom half of the LIDAR. In the CARMEN toolkit has a flag where 0 means right side up and a 1 means upside-down. If we flip the LIDAR and sink the upper housing into the roof (sealing the hole from elements, of course), we might be able to get the LIDAR lens higher while marginally increasing our chances of a scrape.

4.2: Position And Orientation of Robot With Respect To The World

With the information gained in Section 4.1, we then calculated what would happen if the robot traveled to a point where its front wheels were resting on top of a 2" curb (see Figure 4-2).

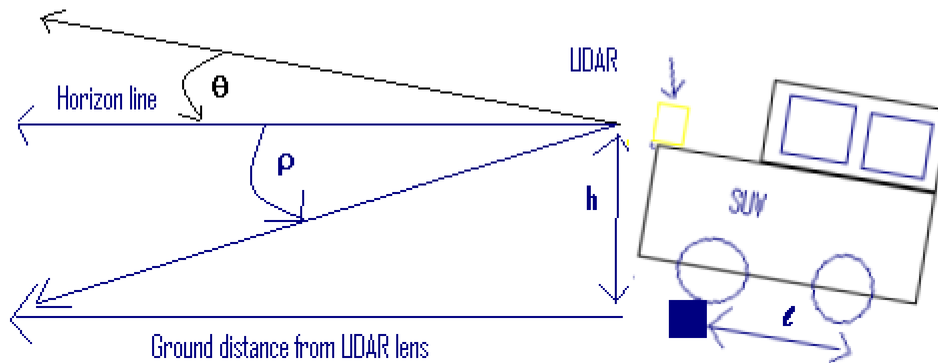


Figure 4-2: A Hypothetical Bump

Applying Figure 4-2, our new calculations concluded:

$$\text{Maximum ground distance} = \cot(\rho-\theta) * [\sin\theta * [\ell - h*\tan\theta] + h*\sec\theta]$$

Using this equation, the 2" curb from our example at the beginning of this section would raise the LIDAR 0.05 meters and decrease the angle of the lens with respect to the horizon from 6 degrees to 5. These very minor changes result in an increase of maximum horizontal distance of approximately 2.5 meters – a major change.

If changing the pitch of the vehicle would change the view of the LIDAR, then roll and yaw would also change the view of the LIDAR. Thus, we needed to integrate GPS/IMU data into our calculations in order to properly adjust the LIDAR data values to the correct positions in 3D space relative to the Earth. In order to be useful, the GPS/IMU data

[9] http://www.darpa.mil/grandchallenge/Rules_8oct04.pdf

would need to be trustworthy, highly accurate to several decimal places, and most important of all, extremely fast so as to be able to appropriately compensate for every bump and crevice in the road while building our list of points in 3D space.

4.3: Blind Spots

The third problem that would need to deal with was blind spots. Once the LIDAR encounters a surface that reflects the LASER light, it just reports the distance to that object. However, if there are any obstacles (or as in Figure 4-3, a cliff) behind that surface, the LIDAR will not see those obstacles or cliffs until after they come into view – similar to running the risk of out-driving a car's headlights by driving too fast during a torrential downpour.

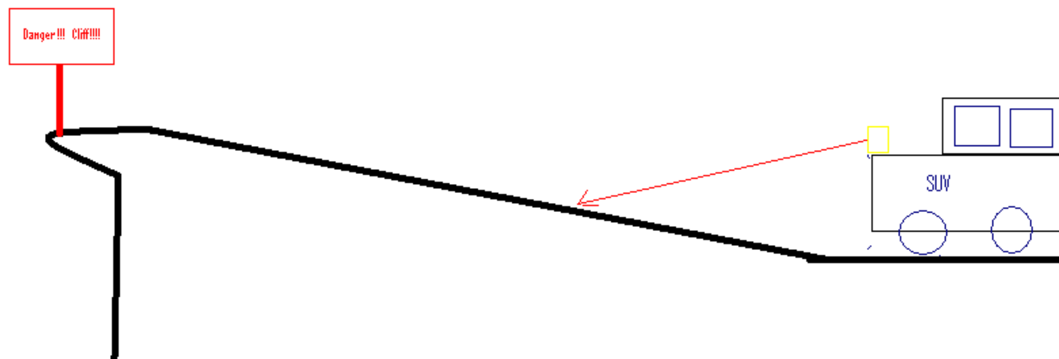


Figure 4-3: A blind cliff

In Figure 4-3, there is a smooth hill that looks to be easily surmountable, and is in the direction the robot should be driving. The robot heads up the hill with speed, but because of the shape of the hill, the robot does not see that just beyond the hill is a significant cliff dropping off into a chasm. One would hope that as the robot nears the summit of the hill, it would realize a chasm exists and slam on the brakes, but is there any way to avoid putting the robot in danger?

4.3.1: Possible Solutions

One solution to this problem is to put additional LIDARs on the vehicle so multiple areas are being scanned. This would give a greater view of the surrounding areas to the Obstacle Detection algorithm, but would also necessitate the design of a Sensor Fusion module. Not difficult, but it would add to the complexity, as well as necessitate dedicating the manpower to development, testing, and integrating such a module. The only financial drawback would be the additional funding to purchase the additional sensors and possibly the computers to process the additional sensor data.

Another solution would be to employ a manufactured off-the-shelf 2D LIDAR, such as those made by Riegl. However, given the cost of a 2D LIDAR at ~$27,000, this might not be an option

Yet another solution would be to build our own rotating LIDAR. However, without the proper mechanical engineering specialists and facilities on board, this too might not be an option.

In all cases, highly accurate timestamps are necessary for determining blind spots. Timestamps help separate new data from old, give us a record of what has been seen when in the past, and permit computers to calculate velocity and acceleration. In Linux, the computer can usually semi-accurately report milliseconds from the "epoch" – January 1, 1970. GPS units, because they synchronize with the atomic clocks aboard the GPS satellites in orbit, also report timestamps, but the lower-quality consumer-level models we have tested are not more accurate than a single second.

4.4: False Readings

The fourth and final problem regards the actual readings from the LIDARs. Two types of false readings could occur: false positives, and false negatives.

4.4.1 False Positives

While we have not confirmed this, supposedly one weakness of LIDARs is their inability to see through a dense cloud (such as dust or fog), or directly into the sun (see Figure 4-3 below for an example of a dust cloud on a road).

Figure 4-3: Dust Cloud On Road

Supposedly, a cloud might show up as a wall, or as a sprinkling of sparse points. Such weaknesses would demonstrate themselves in the LIDAR data stream as objects that are not actually there. The robot might get stymied and not be able to pick a direction to go.

Another problematic false positive is when the LIDAR scans a moving object. The object would theoretically appear as smear in the data. DARPA tested this problem by using a sliding gate at the March qualifying round: just before the finish line, a gate would slide left to right completely blocking the direct path of the robot; after 10 seconds

of being shut, the gate would open again so the robot could proceed to the finish area. If the robot saw the gate in its closed position, but did not update appropriately its data as time went on, the gate would open, but the robot would not know to move to the finish area.

4.4.2 False Negatives

If objects are not detected, the robot may drive into objects (several vehicles became entangled in barbed wire during the first race in March 2004).

Using the example of the opening and closing gate discussed above in Section 4.4.1, if the robot only sees the gate when it was open but did not update its information to reflect the gate as it closes, the robot would crash into the gate because its memory showed an open gate (as was the case with multiple robots during the March 2004 qualifying round).

A rumored false negative problem with LIDARs is barbed wire and sparse bushes (see Figure 4-4).

Figure 4-4: Barbed Wire and Sparse Bushes

Supposedly, barbed wire and sparse bushes will either not be detected by LIDAR, or if they are detected, they will show up as a sprinkling of sparse points. Barbed wire will especially be difficult because its twisting nature means a reflective surface with the appropriate angle to reflect will occur an a sporadic basis. Without something significant, our Obstacle Detection algorithm might incorrectly ignore these sparse point readings.

A third false negative scenario: While humans can only process approximately 30 frames per second, they make up for it with significant feedback mechanisms built into their bodies, including adapted eye and neck muscles. Robots equipped with LIDARs are not as lucky – if the robot is traveling fast enough, a bump may simply cause a LIDAR to not see an object that might put the robot in danger.

4.4.3 Possible Solutions

If these limitations are accurate, one way to solve the problem would be to supplement the LIDAR with non-LIDAR sensors to confirm the LIDAR data. This additional level of complexity is one of the most difficult problems of the DARPA Grand Challenge,

because it involves estimating confidence levels in various sensors and then giving the robot some way to ignore some sensors over others. Not an easy task.

In addition, the problem might be resolved by reporting a low confidence in our obstacle list, which could then potentially trigger the robot to back up, turn left or right, and redraw the surroundings at a different angle, assuming new data is supplants old data. The vehicle SciAutonicsII used this type of backup and redraw system in its successful run of the qualifying round last year in order to properly avoid at least one obstacle.

Section 5: Implementation

5.1 LIDAR Data Stream Capture Program

As mentioned in Section 3, we experimented with and finally chose to adapt CMU's CARMEN toolkit, rather than build our own from scratch. This was for five reasons:

1) We were using a PLS for our testing. The SICK-issued PLS software does not have a C, C++, or Java API – only if we had been using a LMS, would we have been able to use the API portion of SICK's MST software. The CARMEN toolkit was built to handle both SICK LMS & PLS, and by simply switching one value in the carmen/carmen.ini initialization file, we would have been able to easily switch between LMS & PLS.
2) The LMS & PLS both use a non-standard CRC calculation for each command, so we would have to either:
    a. Hard encode every command after running it through the SICK-provided CRC calculator, or
    b. Convert SICK's published pseudo-code CRC algorithm into C, C++, or Java, and then test our code to make sure it's calculating correctly.
3) Due to the limited time left until the March 2005 deadline for making a video of a partial prototype, we decided that we would not have the necessary time to build and thoroughly test a piece of software.
4) Unlike almost all of the other LMS/PLS compatible Open Source software on the Internet, we were able to get the CARMEN software to compile in Linux. With some work, we expected we would also be able to compile CARMEN in Windows, and thus have the possibility for cross-platform development.
5) The CARMEN software demonstrated that it could process the data from the LIDAR faster than the SICK-issued Windows software.

The CARMEN code's interaction with SICK LIDARs works as follows:

- First, a program called "central" must be started as a TSR. This central process coordinates all CARMEN inter-process communications.
- Next, the program "parameter-server" must be started as a TSR with a command line argument specifying a robot name reads that robot's specific arguments from the carmen/carmen.ini. In our review of the carmen.ini file that comes with the downloadable package from CMU, we chose the Pearl robot because it seemed the simplest robot listed in the file.
- Finally, the program carmen/src/laser/laser must be started as a TSR to begin a data stream dump from the LIDAR.
- If one wishes to view a graphical interpretation of the LIDAR data stream, the program carmen/src/laser/laserview produces a picture similar to Figure 5-1:

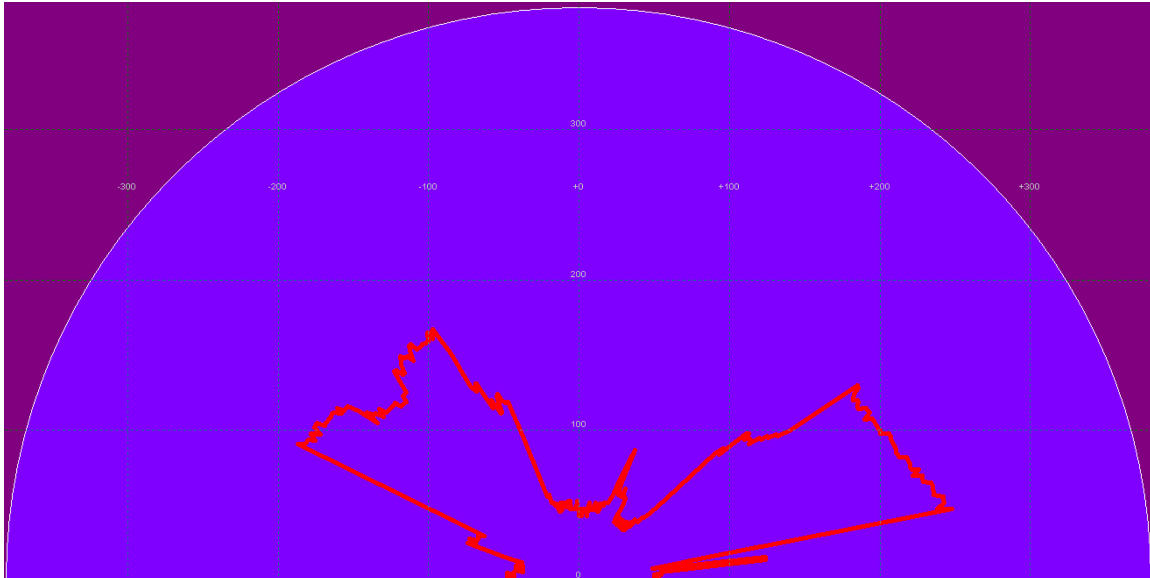


Figure 5-1: An example graphical view of LIDAR data

Figure 5-1 shows the LIDAR detecting objects closest at approximately degrees 0, 90, and 180, and objects slightly farther away at approximately degrees 45 and 135. Note the jagged edge from the image – this was due to the fact that the LIDAR was not able to send a high precision data stream due to the fact that we were limited to RS-232 connectivity, and thus only could communicate at speeds up to 56 kbps, and thus were only getting a partial set of data.

By tracing the CARMEN code, we found the carmen/src/laser/laser_ipc.c file had the most direct connection to the LIDAR itself. Specifically, at lines 50 & 51 of the file:

```
for(i = 0; i < laser->numvalues - 1; i++)
        msg.range[i] = laser->range[i] / 100.0;
```

The struct named laser had many sub-variables, including the total number of values to be returned with each line scan, as well as an array of the ranges being reported back.

By inserting a few extra lines at this point to write the laser->range[i] values to the file with each iteration of the counter variable “i”, we were able to complete the goal of tapping the LIDAR data stream and writing the range data from each LIDAR-scanned line to a CSV file.

Our data structure for the file fields for the CSV was as follows:

<u>Computer Section</u>
1: computer's time stamp (microseconds or milliseconds since epoch (Jan 1, 1970))

<u>LIDAR Section</u>
2: # range entries reported (call this "n")
3: range entry at 0 degrees

.
.
.
2+n: range entry at n-1 degrees

Geo-Location Section
2+n+1: latitude
2+n+2: longitude
2+n+3: compass heading
2+n+4: yaw
2+n+5: pitch
2+n+6: roll

5.2 Convert to Cartesian Program

The next step was to take this data in and, based on the GPS/IMU data, calculate points in 3D space in relation to the Earth. The source code is attached in Appendix B.

The key GPS/IMU integration code was at this point in the program (lines 69 to 88):

First we tokenize a line of the CSV:

```
StringTokenizer st = new StringTokenizer(currentLine, ",");
```

We next get the timestamp:

```
currentTimestamp = Double.parseDouble(st.nextToken());
```

Then, we track the change in time since the last time stamp and figure out how much we've moved since the last line:

```
if(lastTimestamp != 0) {
	double timeDifference = currentTimestamp - lastTimestamp;

	xShift += velocity[0]*timeDifference;

	yShift += velocity[1]*timeDifference;

	zShift += velocity[2]*timeDifference;
}
```

We then figure out how many readings we've recorded:

```
int numReadings = Integer.parseInt(st.nextToken());

double currentReading;
```

Then for each point, we adjust it based on the distance we've moved and adapt it based on the downward angle of the LIDAR:

```
for(int i=0; i < numReadings; ++i) {
	currentReading = Double.parseDouble(st.nextToken());

	if(currentReading <= BLINDCUTOFFDISTANCE) {
		double theta = i*Math.PI/numReadings;

		out.print((xShift + currentReading*cosAlpha*Math.sin(theta)));

		out.print("," + (yShift + currentReading*Math.cos(theta)));

		out.println("," + (zShift + -currentReading*sinAlpha*Math.sin(theta)));
	}
}
```

Note the BLINDCUTOFFDISTANCE constant. We discovered during the tests (see Section 6) that the LIDAR often reported distances much farther than the manufacturer stated (well in excess of 45m). To cut down on these false readings, we instituted this check of the current reading so we could discard the erroneous values.

5.3 Obstacle Detection Program

Professor Harris vetted several papers on obstacle detection over the summer and gave us what he viewed were the two best papers. These papers were:

Obstacle Detection and Terrain Classification for Autonomous Off-road Navigation: R. Manduchi, A. Castano, A.Talukder, L. Matthies; JPL; April 2004[10]

Obstacle Detection and Mapping System: Tsai-Hong Hong, Steven Legowik, Marilyn Nashman; NIST; 1998[11]

A summary of the Manduchi article:

- For navigation indoors or in structured environments (such as roads), obstacles are simply defined as surface elements that are higher then the ground plane by some amount.
- A surface ramp is considered part of an obstacle if its slope is larger than a certain value θ and if it spans a vertical interval larger than some threshold H.
- Obstacle detection (OD) algorithms normally rely on the "flat world" assumption.
- For good accuracy, must have a high-resolution elevation map.
- The algorithm uses two techniques:

[10] http://citeseer.lcs.mit.edu/648676.html
[11] http://citeseer.lcs.mit.edu/hong98obstacle.html

  - 5-NN (5 nearest neighbor) voting to attempt to avoid false positives
  - Second derivative change in distance

A summary of the Hong article:

- In similar fashion, their algorithm used 5-NN voting, but it looked for a change in confidence value above some threshold:
  - A discontinuity in elevation exceeding some value
  - The surface slope exceeding some value

We adapted those systems to our obstacle detection program (attached in Appendix B). The program reads in all of the x,y,z points from the data stream capture program and outputs into a new file the points one-by-one followed by the highest slope of that point calculated respective to its five nearest neighbors.

We have not as of yet completed the difficulty level calculation.

Section 6: Gathering Data Sets

Our test run for real data gathering was the morning of November 10, 2004 at the parking lot of the local Boomers[12].

Our test vehicle was team member Theresia Olson's Jeep Wrangler. We powered the LIDAR and team member Jordan Levy's laptop (which was gathering the data)

6.1 Physical Setup Of The Tests

Figures 6-1, 6-2, and 6-3 show team member Bayan Towfiq mounting the LIDAR on the top of the hood. This was the highest location from where we could safely mount the LIDAR on this vehicle, and as we had calculated, the higher the LIDAR, the farther it can see.

Figure 6-1: Mounting the LIDAR

Figure 6-2: Front view

[12] 3405 Michelson Dr, Irvine, CA 92612

Figure 6-3: Side view

Figure 6-4: View of the CARMEN laserview program while the car is at a standstill

In Figure 6-3, the greenery that bordered the parking lot can be seen in the background. Figure 6-4 demonstrates the view of the LIDAR with that greenery – the right half is completely blue, and the left half is a mix of white and blue (the blue section on the left hand side is a tree in the middle of the parking lot).

Figure 6-5: View of the course

As Figure 6-5 clearly shows, we chose this specific parking lot because it is usually empty mid-week mornings, and it has some long level straight-aways. We made two runs with no added obstacles (Sets 1 & 2), and then two runs with added obstacles (Sets 3 & 4), which were various sized boxes plus team member Philip Schlesinger– see Figures 6-6 & Figure 6-7 for their arrangement in the parking lot.

Figure 6-6: Front view of the added obstacles

Figure 6-7: Rear view of the added obstacles

One problem we ran into multiple times was the LIDAR "stalling", to use the CARMEN term. This happens when the laser TSR program does not receive LIDAR data for some unknown reason. Due to this recurrent problem, we had to re-run many of the tests. Our goal is to pull out of CARMEN the minimum code snippets required to make the CARMEN program retrieve the LIDAR data, yet cut down on the complexity of the programming and the resultant overhead.

6.2 Results Of The Tests

After processing the recorded LIDAR data offline through the ConvertToCartesian program, we ran it through team member Navid Azimi's point mapping program. The results are below.

For orientation purposes, in each of the diagrams below there are three lines, all of which intersect at a point (the origin):

- The red line is the positive x-axis, which symbolizes forward movement on the ground plane away from the origin.
- The orange line is the positive y-axis, which symbolizes leftward movement on the ground plane away from the origin.
- The green line is the positive z-axis, which symbolizes height above the ground plane.

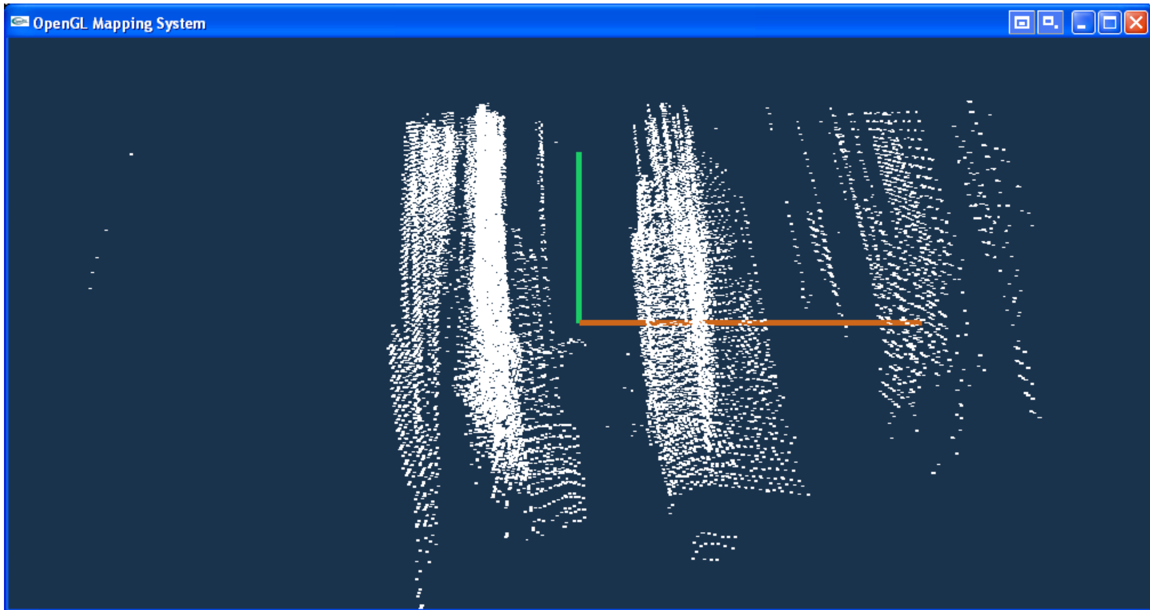


Figure 6-8: Set 1 (no obstacles) viewing backward

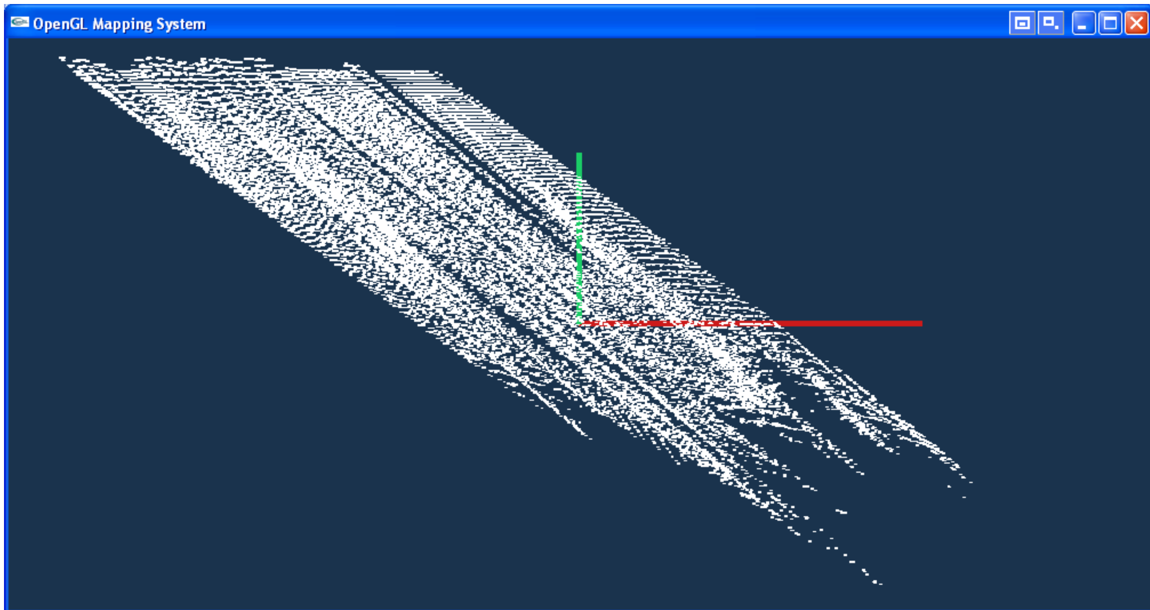


Figure 6-9: Set 1 (no obstacles) viewing leftward

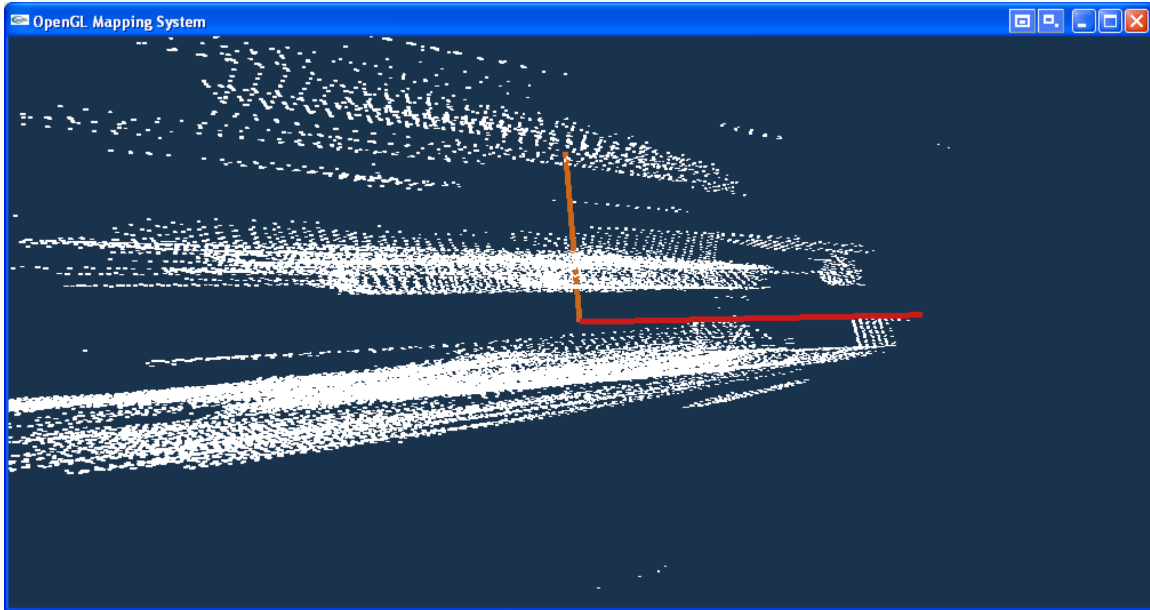


Figure 6-10: Set 1 (no obstacles) viewing downward

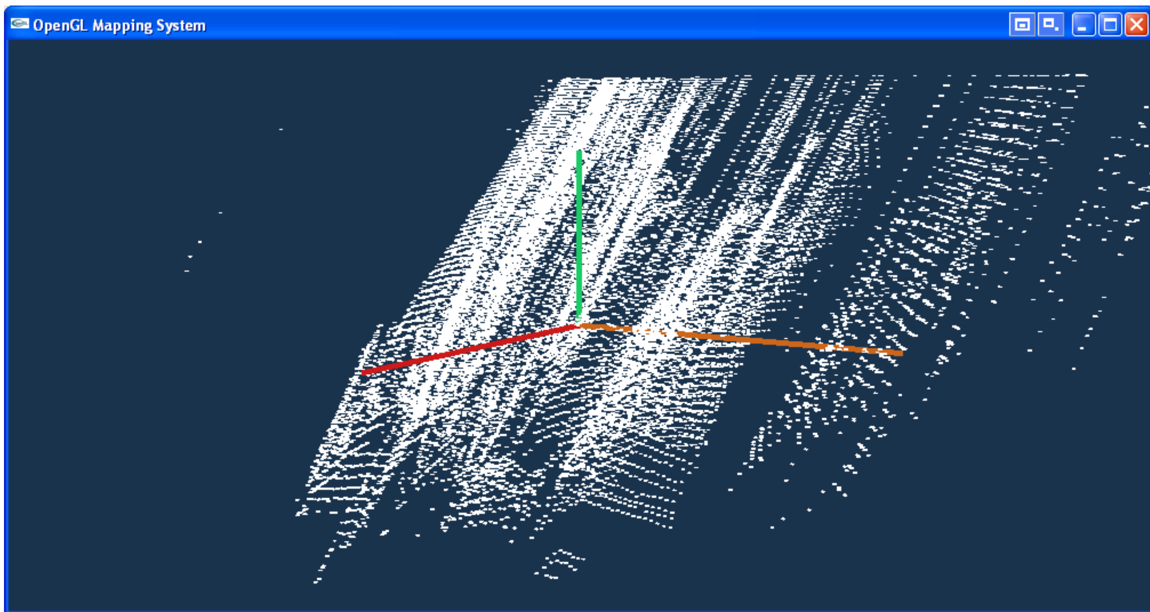


Figure 6-11: Set 1 (no obstacles) viewing all three axes

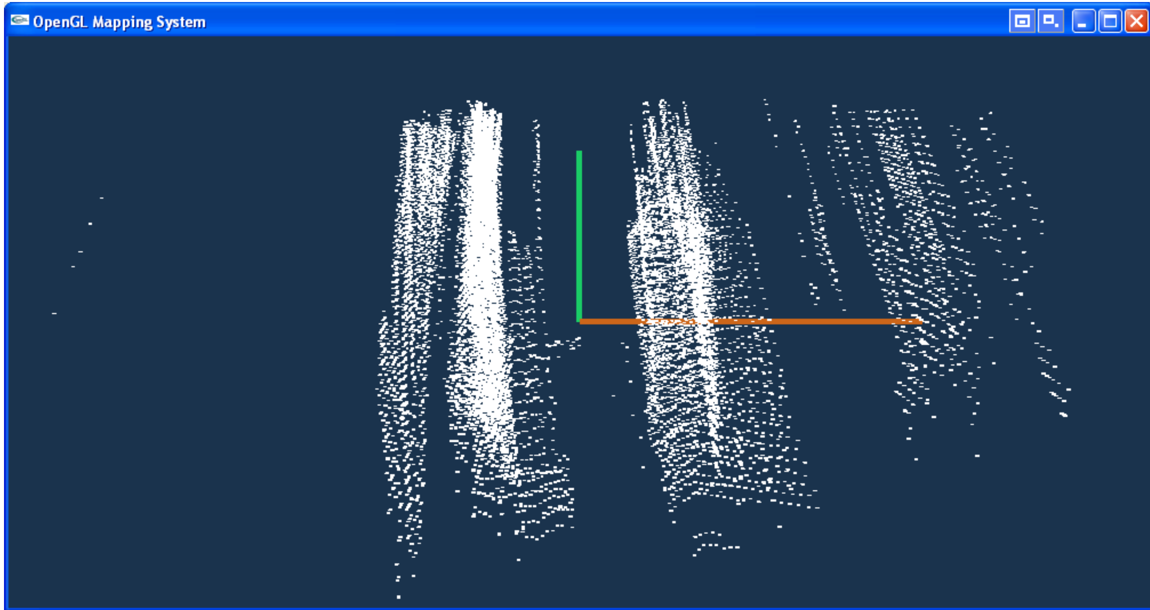


Figure 6-12: Set 2 (no obstacles) viewing backward

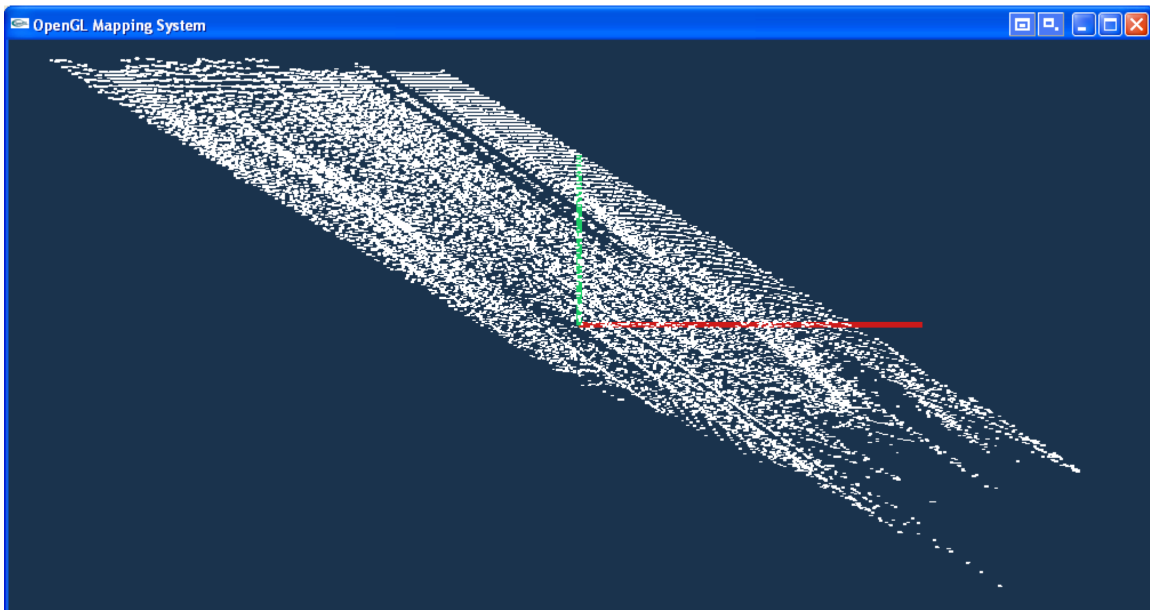


Figure 6-13: Set 2 (no obstacles) viewing leftward

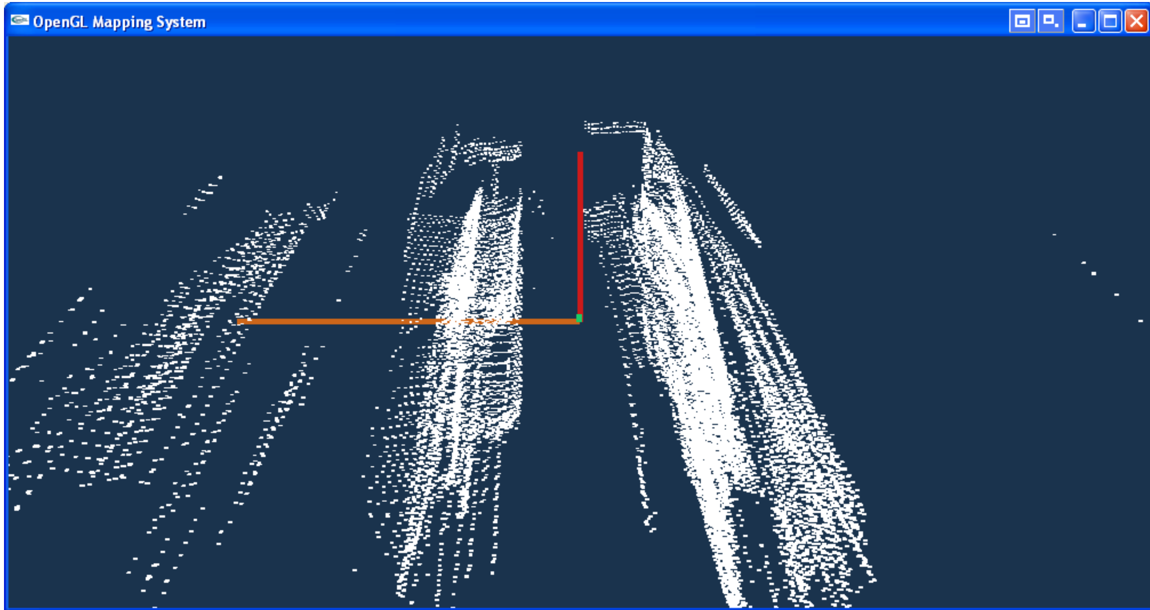


Figure 6-14: Set 2 (no obstacles) viewing downward

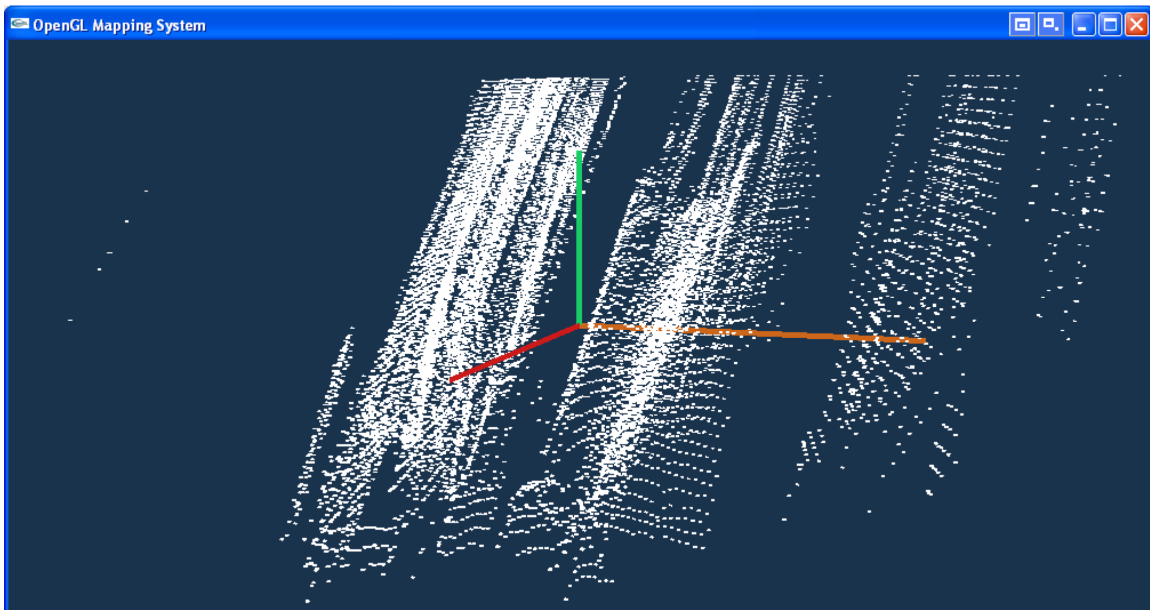


Figure 6-15: Set 2 (no obstacles) viewing all three axes

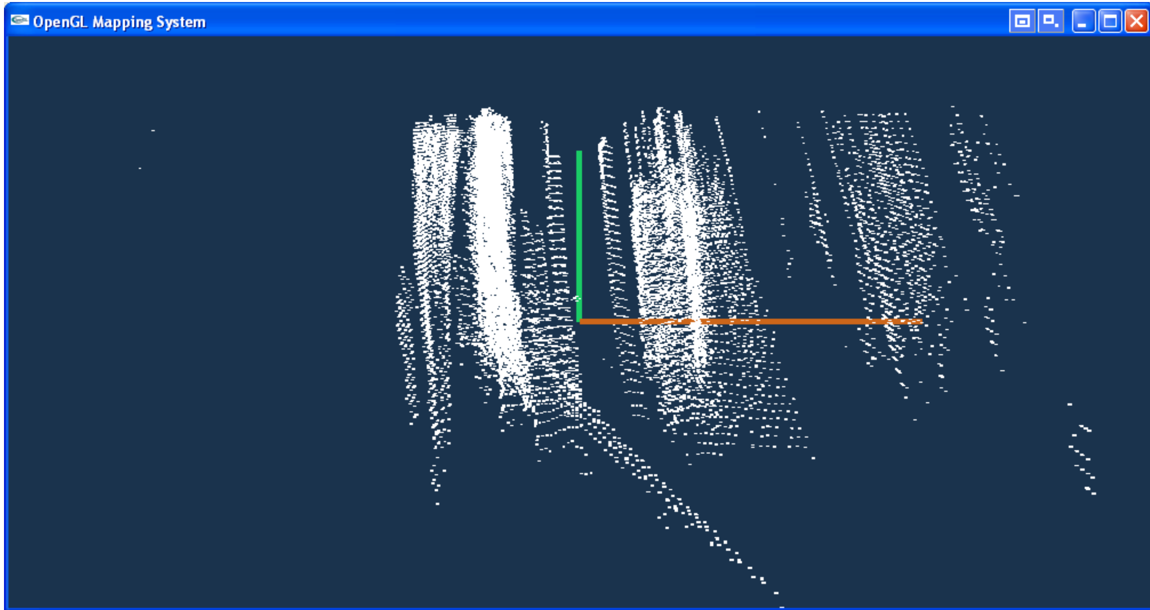


Figure 6-16: Set 3 (with obstacles) looking backward

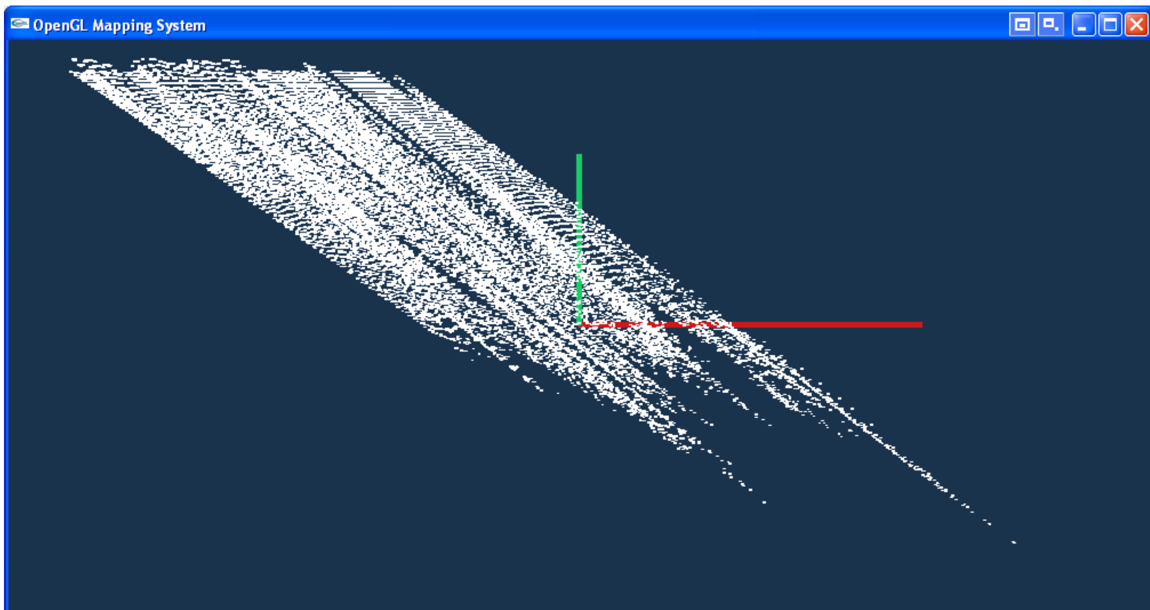


Figure 6-17: Set 3 (with obstacles) looking leftward

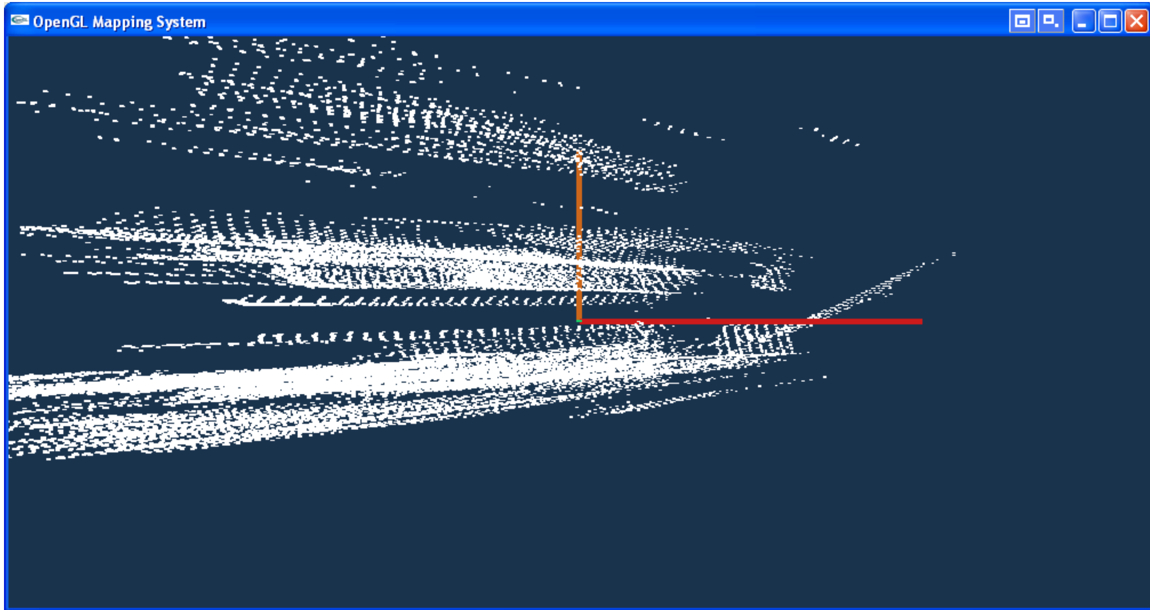


Figure 6-18: Set 3 (with obstacles) looking downward

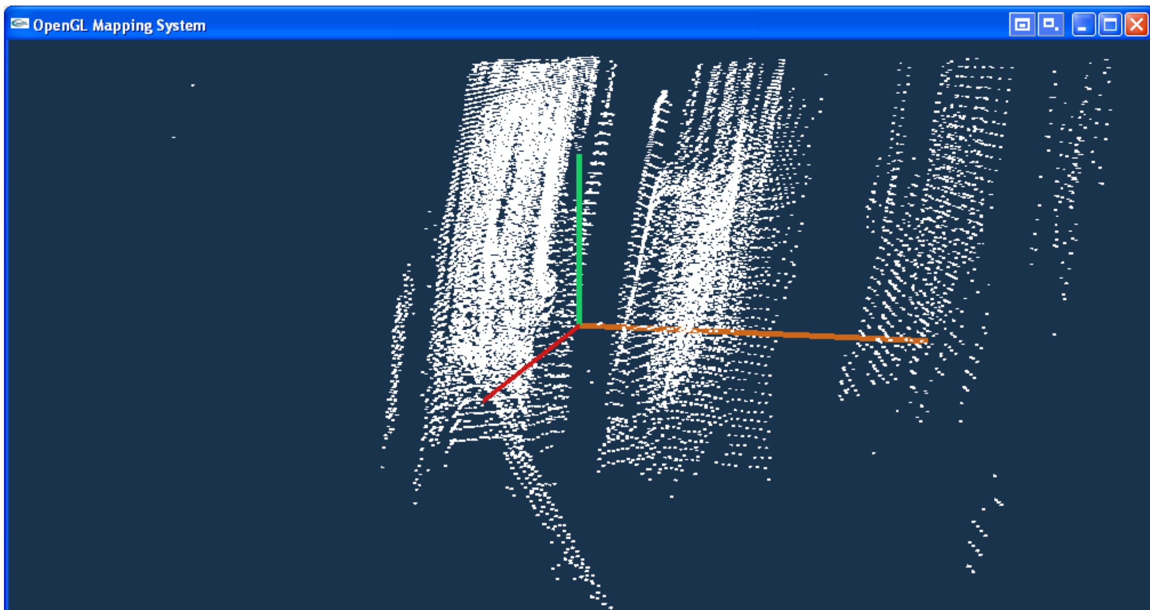


Figure 6-19: Set 3 (with obstacles) looking at all three axes

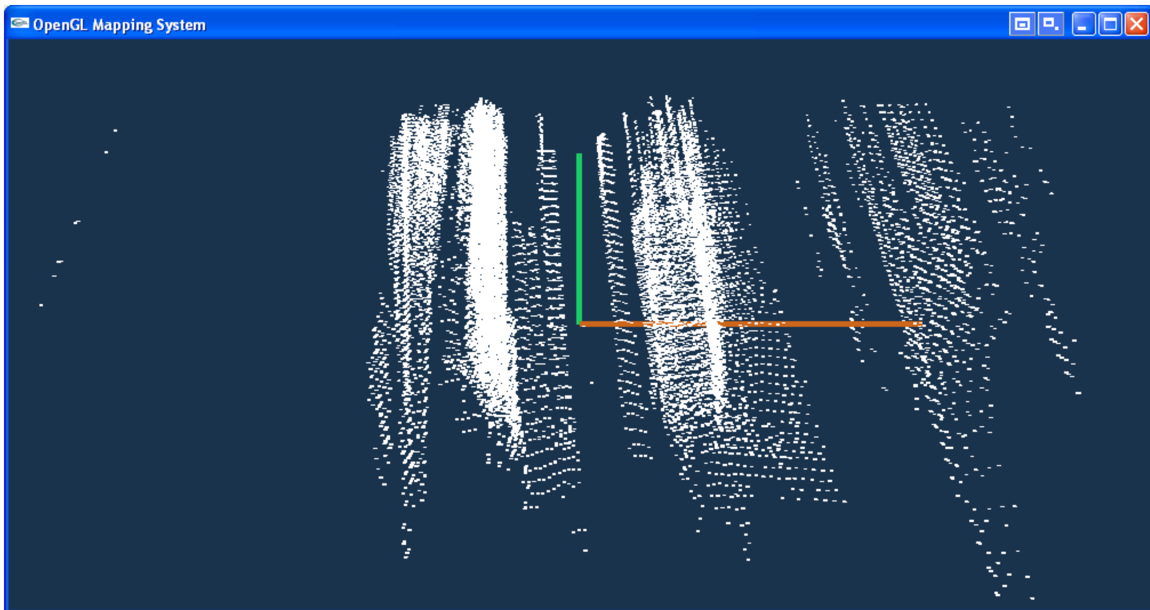


Figure 6-20: Set 4 (with obstacles) looking backward

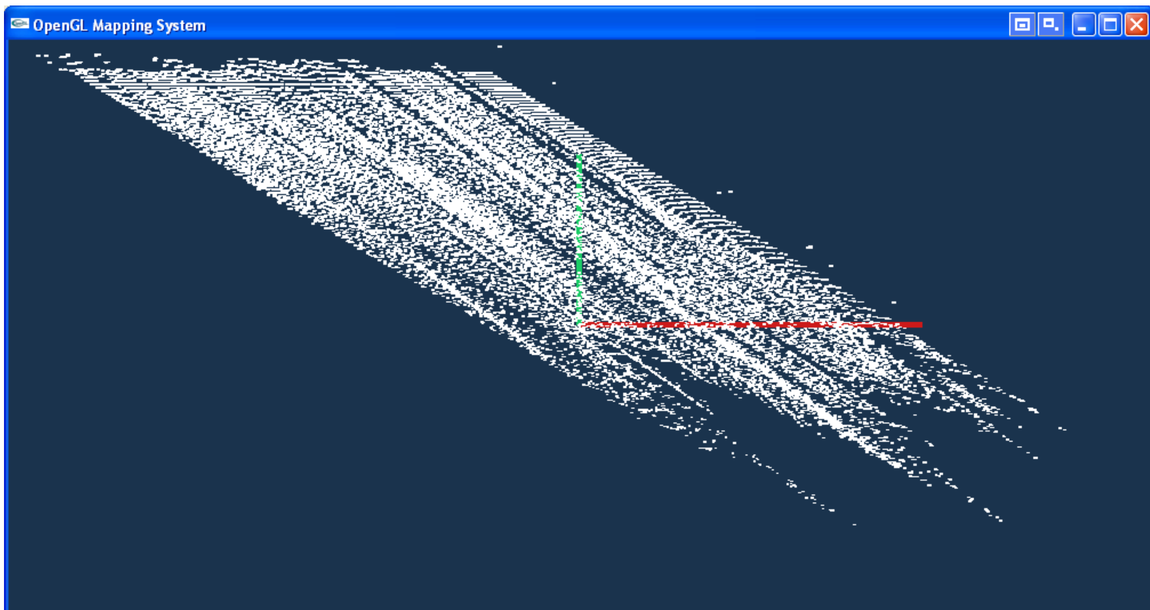


Figure 6-21: Set 4 (with obstacles) looking leftward

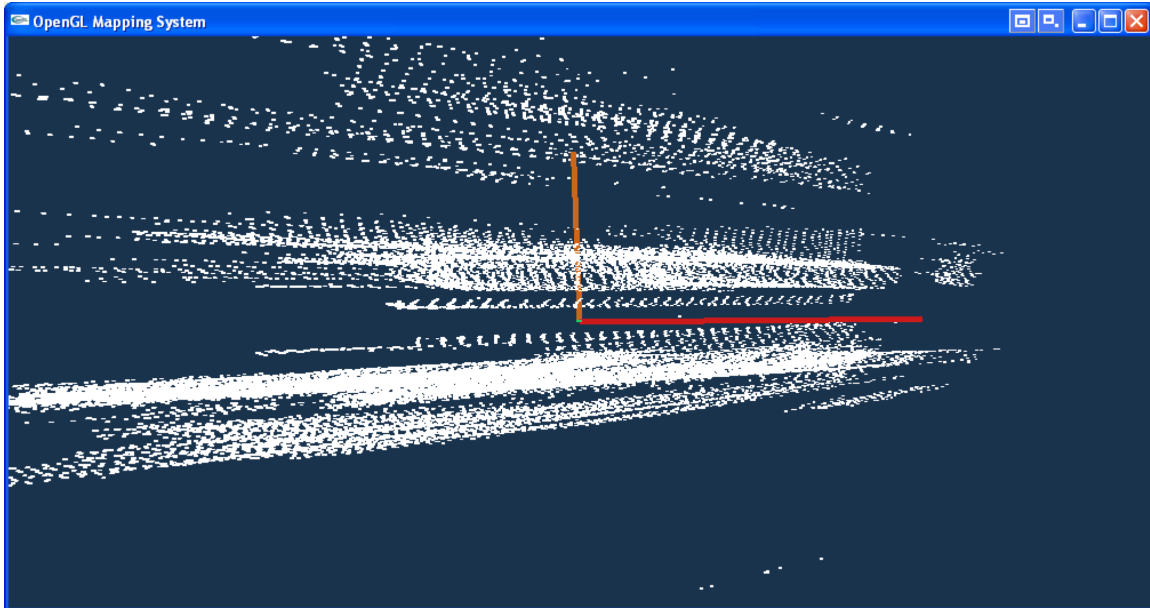


Figure 6-22: Set 4 (with obstacles) looking downward

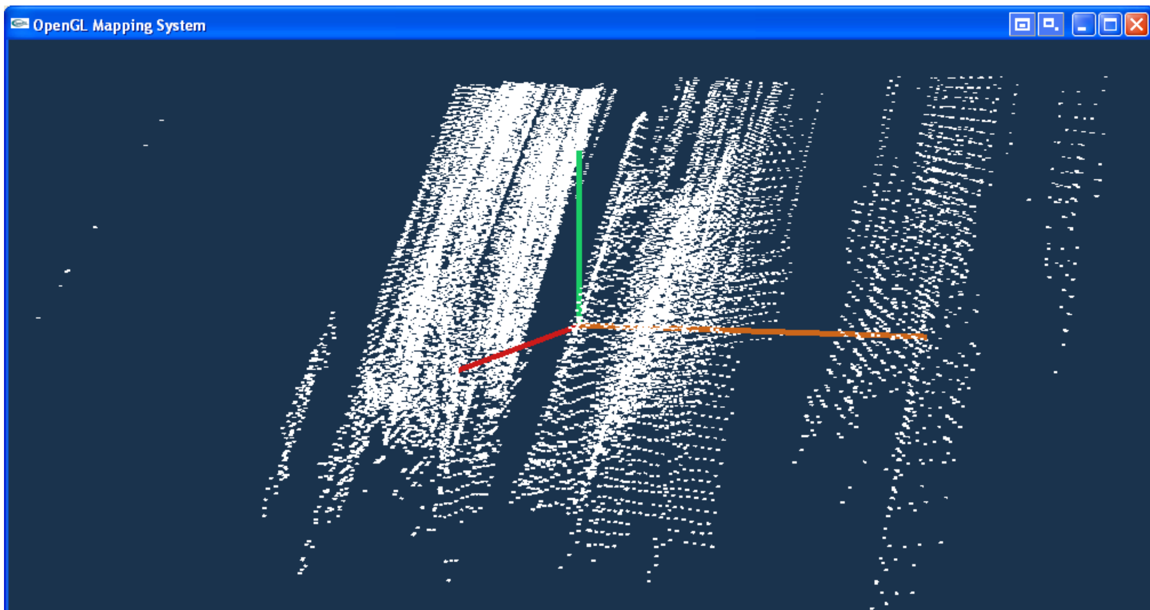


Figure 6-23: Set 4 (with obstacles) looking at all three axes

6.3 Analysis Of The Results

Unfortunately, these diagrams of the converted Cartesian points are not recognizable as compared to the surroundings. The flaw is in at least one of the following areas (sorted in descending order as to level of expectation as being the cause of the diagram problem):

1) ConvertToCartesian.java may not be converting the LIDAR data properly.
2) Navid's mapping program may not be properly pre-processing and graphing the points.

3) Our concept of calculating based on a fixed forward velocity value might not be applied properly.

-

Section 7: Task Schedule

Already completed:

- Software:
    - Reading data out of LIDAR and formatting it
    - Building in future functionality for GPS/IMU data (latitude, longitude, compass heading, yaw, pitch, roll)
    - Calculated 3D space Cartesian coordinates
- Tests:
    - Data gathering from driving in straight line at constant speed with no GPS/IMU integration:
        - With no obstacles (2 sets at ~10 mph)
        - With simple obstacles (boxes) (2 sets at ~10 mph)

Future work:

- Software:
    - Integration with simple GPS/IMU (via Robotics/GPS/IMU group)
    - Finish coding of Obstacle Detection code
    - Integrate inter-module communication code currently being tested by Brain Engineering group.
- Tests:
    - Gather 3D space data with integrated GPS/IMU data
    - Test Obstacle Detection code on gathered 3D space data offline
    - Test Obstacle Detection code on gathered 3D space data online
    - Test hand-off of obstacle list to Brain Engineering group.

Appendix A: Glossary[13]

CMU: Carnegie Mellon University

CSV: Comma Separated Value

GPS: Global Positioning System

IMU: Inertial Measurement Unit

LASER: Light Amplification by Stimulated Emission of Radiation

LIDAR: LASER Identification Detection And Ranging

Pitch: Rotation of the object around the y (left/right) axis

RADAR: Radio Detection And Ranging

Roll: Rotation of the object around the x (forward/backward) axis

SONAR: Sound Navigation And Ranging

TSR: Terminate and Stay Resident

Yaw: Rotation of the object around the z (sky/ground) axis

[13] Acronyms & initializations source: http://www.acronymfinder.com

Appendix B: Relevant Documentation

SICK PLS Datasheet[14]
SICK LMS Datasheet[15]
SICK MST Datasheet[16]
ConvertToCartesian.java source code
Obstacles.h source code
Obstacles.c source code
Modified laser_ipc.c source code

---

[14] http://www.sickusa.com/live/master/datasheet.asp?PN=1016066&FAM=SafeScan
[15] http://www.sickusa.com/live/master/datasheet.asp?PN=1018028&FAM=Measurement
[16] http://www.sickusa.com/Publish/docroot/Technical%20Information%20Sheet(s)/MST%20200%20Software%20Tech%20Info.pdf

```
1   #include <carmen/carmen.h>
2   #include "laser.h"
3   #include "laser_messages.h"
4   #include "sick.h"
5   #include <stdio.h>
6
7
8   int allocsize[4] = {0, 0, 0, 0};
9   float *range_buffer[4] = {NULL, NULL, NULL, NULL};
10  static FILE *stream;
11
12  void publish_laser_alive(int front_stalled, int rear_stalled,
13               int laser3_stalled, int laser4_stalled)
14  {
15    IPC_RETURN_TYPE err;
16    carmen_laser_alive_message msg;
17
18    msg.frontlaser_stalled = front_stalled;
19    msg.rearlaser_stalled = rear_stalled;
20    msg.laser3_stalled = laser3_stalled;
21    msg.laser4_stalled = laser4_stalled;
22
23    err = IPC_publishData(CARMEN_LASER_ALIVE_NAME, &msg);
24    carmen_test_ipc_exit(err, "Could not publish",
      CARMEN_LASER_ALIVE_NAME);
25  }
26
27  void publish_laser_message(sick_laser_p laser)
28  {
29    static char *host = NULL;
30    static carmen_laser_laser_message msg;
31    IPC_RETURN_TYPE err;
32    int i;
33
34    if(host == NULL) {
35      host = carmen_get_tenchar_host_name();
36      strcpy(msg.host, host);
37    }
38    msg.num_readings = laser->numvalues - 1;
39    msg.timestamp = laser->timestamp;
40
41    if(msg.num_readings != allocsize[laser->settings.laser_num]) {
42      range_buffer[laser->settings.laser_num] =
43        realloc(range_buffer[laser->settings.laser_num],
44            msg.num_readings * sizeof(float));
45      carmen_test_alloc(range_buffer[laser->settings.laser_num]);
46      allocsize[laser->settings.laser_num] = msg.num_readings;
47    }
48
49    msg.range = range_buffer[laser->settings.laser_num];
50
51    stream = fopen("laser_ipc_data.csv", "a+");
52    fprintf(stream, "%f,", msg.timestamp*100);        // timestamp rounded
      to nearest millisecond
53    fprintf(stream, "%i", laser->numvalues - 1);
54    if( laser->settings.laser_flipped == 0)
55      {
56        for(i = 0; i < laser->numvalues - 1; i++) {
57      msg.range[i] = laser->range[i] / 100.0;
58      fprintf(stream, ",%f", laser->range[i] / 100.0);
59        }
60      }
61    else
62      {
63        for(i = 0; i < laser->numvalues - 1; i++)
64      msg.range[i] = laser->range[laser->numvalues-2-i];
65      }
```

```
 66     fprintf(stream, "\n");
 67     fclose(stream);
 68 
 69     switch(laser->settings.laser_num) {
 70     case FRONT_LASER_NUM:
 71       err = IPC_publishData(CARMEN_LASER_FRONTLASER_NAME, &msg);
 72       carmen_test_ipc_exit(err, "Could not publish",
 73                            CARMEN_LASER_FRONTLASER_NAME);
 74       break;
 75     case REAR_LASER_NUM:
 76       err = IPC_publishData(CARMEN_LASER_REARLASER_NAME, &msg);
 77       carmen_test_ipc_exit(err, "Could not publish",
 78                            CARMEN_LASER_REARLASER_NAME);
 79       break;
 80     case LASER3_NUM:
 81       err = IPC_publishData(CARMEN_LASER_LASER3_NAME, &msg);
 82       carmen_test_ipc_exit(err, "Could not publish",
 83                            CARMEN_LASER_LASER3_NAME);
 84       break;
 85     case LASER4_NUM:
 86       err = IPC_publishData(CARMEN_LASER_LASER4_NAME, &msg);
 87       carmen_test_ipc_exit(err, "Could not publish",
 88                            CARMEN_LASER_LASER4_NAME);
 89       break;
 90     }
 91   }
 92 
 93   void ipc_initialize_messages(void)
 94   {
 95     IPC_RETURN_TYPE err;
 96 
 97     err = IPC_defineMsg(CARMEN_LASER_FRONTLASER_NAME, IPC_VARIABLE_LENGTH,
 98                         CARMEN_LASER_FRONTLASER_FMT);
 99     carmen_test_ipc_exit(err, "Could not define",
        CARMEN_LASER_FRONTLASER_NAME);
100 
101     err = IPC_defineMsg(CARMEN_LASER_REARLASER_NAME, IPC_VARIABLE_LENGTH,
102                                 CARMEN_LASER_REARLASER_FMT);
103     carmen_test_ipc_exit(err, "Could not define",
        CARMEN_LASER_REARLASER_NAME);
104 
105     err = IPC_defineMsg(CARMEN_LASER_LASER3_NAME, IPC_VARIABLE_LENGTH,
106                         CARMEN_LASER_LASER3_FMT);
107     carmen_test_ipc_exit(err, "Could not define",
        CARMEN_LASER_FRONTLASER_NAME);
108 
109     err = IPC_defineMsg(CARMEN_LASER_LASER4_NAME, IPC_VARIABLE_LENGTH,
110                                 CARMEN_LASER_LASER4_FMT);
111     carmen_test_ipc_exit(err, "Could not define",
        CARMEN_LASER_REARLASER_NAME);
112 
113     err = IPC_defineMsg(CARMEN_LASER_ALIVE_NAME, IPC_VARIABLE_LENGTH,
114                                 CARMEN_LASER_ALIVE_FMT);
115     carmen_test_ipc_exit(err, "Could not define", CARMEN_LASER_ALIVE_NAME);
116   }
117 
```

```
 1  #include "obstacles.h"
 2  #include <stdio.h>
 3  #include <stdlib.h>
 4  #include <math.h>
 5
 6  //global array of [interesting area][180deg scan line][3 coor-X, Y, and
    Z]
 7
 8  float buffer[AREA][SCAN][COOR];
 9  float slope[AREA][SCAN];
10
11  void run()
12  {
13
14       //reads in x,y,z\n formatted arguments
15       //outputs filtered set of data indicating obstacles
16
17      FILE *stream_in;
18      FILE *stream_out;
19      char in[100];
20      int s;
21      float x, y, z;
22      int a=0;
23
24      /* Open files to read line from: */
25      if( (stream_in = fopen( file_in, "r" )) == NULL )
26         exit( 0 );
27
28      if( (stream_out = fopen( file_out, "w")) == NULL )
29         exit( 0 );
30
31      /* Read in data */
32      fgets(in,100,stream_in);
33      sscanf(in,"%f, %f, %f", &x, &y, &z);
34
35
36      //load first scan without testing
37      if(feof(stream_in) == 0)
38      {//
39          for( s=0; s<180; s++)
40          {//
41              slope[a][s] = 0;
42               buffer[a][s][X] = x;
43               buffer[a][s][Y] = y;
44               buffer[a][s][Z] = z;
45               fgets(in,100,stream_in);
46               sscanf(in,"%f, %f, %f", &x, &y, &z);
47
48          }//end for(s
49
50          //load area until end of file, wrapping around at end of array
51          for( a=1; ( feof( stream_in ) == 0 ); a++ )
52           {//
53                //load one 180 degree scan
54             for( s=0; s<180; s++)
55             {//
56                  //load on set of coordinates
57                  slope[a][s] = 0;
58                  buffer[a][s][X] = x;
59                  buffer[a][s][Y] = y;
60                  buffer[a][s][Z] = z;
61                  fgets(in,100,stream_in);
62                  sscanf(in,"%f, %f, %f", &x, &y, &z);
63
64                  //test locals ... one row back, one degree before
65                  check_slope(a-1, s-1);
66                  fprintf(stream_out, "%f, %f, %f, %f\n",
```

```
 66                 buffer[a-1][s-1][X], buffer[a-1][s-1][Y],
 67                              buffer[a-1][s-1][Z], slope[a-1][s-1]);
 68             }//for(s
 69             if(a == AREA-1)
 70                 a=-1;
 71          }//end for a=0
 72      }//end if eof
 73
 74      fclose( stream_in );
 75      fclose( stream_out );
 76  }//end run()
 77
 78
 79
 80  int check_slope(int a, int s)
 81  {
 82      //determine the highest slope for the point at
         buffer[areaInt][scanInt]
 83      //by checking five of its neighbors - one on each side and three in
         front
 84      //check pattern:  1 2 3
 85      //                4 x 5
 86      int check_A;
 87      int check_S;
 88      int rightedge = 0;//false
 89      int topedge = 0;//false
 90      int leftedge = 0;//false
 91      float thisAlpha;
 92      int steep = 0;
 93      int loop_pt;
 94
 95      if(s==0)
 96          leftedge = 1;
 97      else if(s== -1)
 98      {
 99          rightedge = 1;
100          s=SCAN-1;
101      }
102      if(a== -1)
103      {
104          topedge=1;
105          a=AREA-1;
106      }
107
108
109      for(loop_pt=1; loop_pt<6; loop_pt++)
110      {
111          switch (loop_pt)
112          {
113          case 1://test left lower
114              if(leftedge==1)
115              {
116                  check_A=-1;
117                  check_S=-1;
118              }
119              else
120              {
121                  check_A = a;
122                  check_S = s-1;
123              }
124              break;
125          case 2://test left upper
126              if(leftedge==1)
127              {
128                  check_A=-1;
129                  check_S=-1;
130              }
```

```
131                 else
132                 {
133                     check_S=s-1;
134                     if(topedge==1)
135                         check_A=0;
136                     else
137                         check_A=a+1;
138                 }
139                 break;
140             case 3://test upper middle
141                 if(topedge==1)
142                     check_A=0;
143                 else
144                     check_A=a+1;
145                 check_S=s;
146                 break;
147             case 4://test right upper
148                 if(rightedge==1)
149                 {
150                     check_A=-1;
151                     check_S=-1;
152                 }
153                 else
154                 {
155                     check_S=s+1;
156                     if(topedge==1)
157                         check_A=0;
158                     else
159                         check_A=a+1;
160                 }
161                 break;
162             case 5://test lower right
163                 if(rightedge==1)
164                 {
165                     check_A=-1;
166                     check_S=-1;
167                 }
168                 else
169                 {
170                     check_A=a;
171                     check_S=s+1;
172                 }
173                 break;
174             }//end switch
175             //test (a,s) with (check_A, check_S)
176             if( (check_A>=0) && (check_S>=0)) {
177                     int denom =
                        pow((buffer[a][s][X]-buffer[check_A][check_S][X]),2) +
178                         pow((buffer[a][s][Y]-buffer[check_A][check_S][Y]),2)
                            +
179                         pow((buffer[a][s][Z]-buffer[check_A][check_S][Z]),2);
180                     if (denom!=0)
181                     {
182                         thisAlpha = (pow((buffer[a][s][Z]-buffer[check_A][
                            check_S][Z]),2))/denom;
183
184                         if (thisAlpha >= slope[a][s])
185                                 slope[a][s] = thisAlpha;
186                         if (thisAlpha >= ALPHA)
187                             steep = 1;
188                     }
189                 }//end if(check_A...
190
191         }//end for loop
192         return steep;
193     }
194
```

```
 1  #ifdef __cplusplus
 2  extern "C" {
 3  #endif
 4
 5
 6  #define file_in      "data.txt"
 7  #define file_out     "output.txt"
 8  #define AREA         128
 9  #define SCAN         180
10  #define COOR         3
11  #define HMAX         10
12  #define X            0
13  #define Y            1
14  #define Z            2
15  #define ALPHA        1 //actually represents alpha squared
16
17
18  void run();
19  int check_slope(short areaInt, short scanInt);
20
21  #ifdef __cplusplus
22  }
23  #endif
24
```

```
1   import java.io.*;
2   import java.util.*;
3   import java.lang.Math;
4   public class ConvertToCartesian {
5       private static BufferedReader in;
6       private static PrintWriter out;
7       private static double lastTimestamp, currentTimestamp;
8       private static double cosAlpha, sinAlpha;
9       private static double[] velocity;
10      private static double pitch, roll, yaw;
11      private static double xShift, yShift, zShift;
12      private static final double BLINDCUTOFFDISTANCE = 45.0;     // set
        this to something larger than 60 to disable cutoff
13
14
15      public static void initialize(String inFile, double xVelocity, double
        yVelocity, double zVelocity, double alphaAngle) {
16          openInFile(inFile);
17          openOutFile(inFile + ".3d");
18          lastTimestamp = 0;
19          currentTimestamp  = 0;
20          xShift = 0.0;
21          yShift = 0.0;
22          zShift = 0.0;
23          velocity = new double[3];
24          // Temporary vehicle dynamics parameters
25          velocity[0] = xVelocity/1000.0; // x-compnent velocity (convert
            to meters/millisecond
26          velocity[1] = yVelocity/1000.0; // y-component velocity
27          velocity[2] = zVelocity/1000.0; // z-component velocity
28          pitch = 0.0;
29          roll = 0.0;
30          yaw = 0.0;
31          cosAlpha = Math.cos(alphaAngle);
32          sinAlpha = Math.sin(alphaAngle);
33      }
34
35
36      public static void main(String[] args) {
37          if(args.length != 5) {
38              System.out.println();
39              System.out.println("Format: java ConvertToCartesian <input
                file> <x-velocity> <y-velocity> <z-velocity> <angle>");
40              System.out.println();
41              System.out.println("<input file> = the name of the data input
                file");
42              System.out.println("<x-velocity> = x-component velocity
                (meters/sec)");
43              System.out.println("<y-velocity> = y-component velocity
                (meters/sec)");
44              System.out.println("<z-velocity> = z-component velocity
                (meters/sec)");
45              System.out.println("<angle> = angle of lidar mount with
                respect to the horizontal (in degrees)");
46              System.out.println();
47              System.exit(0);
48          }
49          initialize(args[0], Double.parseDouble(args[1]), Double.
            parseDouble(args[2]), Double.parseDouble(args[3]),
            Math.PI*Double.parseDouble(args[4])/180);
50          try {
51              String s = in.readLine();
52              processLine(s, velocity, pitch, roll, yaw);
53              while(s != null) {
54                  processLine(s, velocity, pitch, roll, yaw);
55                  s = in.readLine();
56              }
```

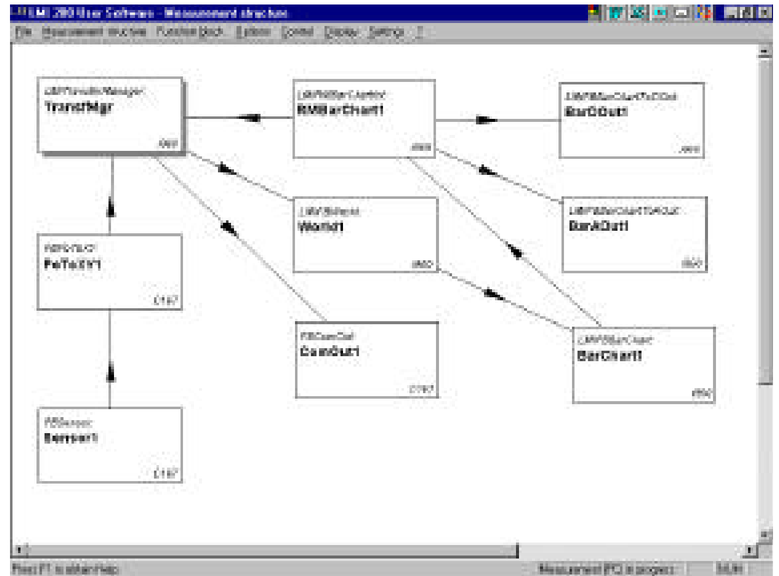
```
 57             in.close();
 58             out.close();
 59         }
 60         catch(IOException e) {
 61             System.out.println("Error while attempting to read input
                file... exiting");
 62             System.exit(0);
 63         }
 64     }
 65 
 66 
 67     public static void processLine(String currentLine, double[] velocity,
        double pitch, double roll, double yaw) {
 68         try {
 69             StringTokenizer st = new StringTokenizer(currentLine, ",");
 70             currentTimestamp = Double.parseDouble(st.nextToken());
 71             if(lastTimestamp != 0) {
 72                 double timeDifference = currentTimestamp - lastTimestamp;
 73                 xShift += velocity[0]*timeDifference;
 74                 yShift += velocity[1]*timeDifference;
 75                 zShift += velocity[2]*timeDifference;
 76             }
 77 
 78             int numReadings = Integer.parseInt(st.nextToken());
 79             double currentReading;
 80             for(int i=0; i < numReadings; ++i) {
 81                 currentReading = Double.parseDouble(st.nextToken());
 82                 if(currentReading <= BLINDCUTOFFDISTANCE) {
 83                     double theta = i*Math.PI/numReadings;
 84                     out.print((xShift + currentReading*cosAlpha*Math.sin(
                        theta)));
 85                     out.print("," + (yShift + currentReading*Math.cos(
                        theta)));
 86                     out.println("," + (zShift + -currentReading*sinAlpha*
                        Math.sin(theta)));
 87                 }
 88             }
 89         }
 90         catch(NumberFormatException e) {
 91             System.out.println("Invalid or corrupt data file...
                exiting");
 92             System.exit(0);
 93         }
 94         lastTimestamp = currentTimestamp;
 95     }
 96 
 97 
 98     public static void openInFile(String filename) {
 99         try {
100             in = new BufferedReader(new FileReader(filename));
101         }
102         catch(IOException e) {
103             System.out.println("Error while trying to open input file: "
                + filename);
104             System.exit(0);
105         }
106     }
107 
108 
109     public static void openOutFile(String filename) {
110         try {
111             out = new PrintWriter(new FileWriter(filename));
112         }
113         catch(IOException e) {
114             System.out.println("Error while trying to open output file: "
                + filename);
115             System.exit(0);
```

```
116         }
117     }
118 }
119
```

TECHNICAL INFORMATION

Features

- Performs measurement functions for LMS laser scanners
- Two pre-installed drivers for real-time communication with up to two laser scanners
- Functions library includes filter functions such as cutting out irrelevant measurement zones
- Tiered structure allows for integration of new function blocks or software drivers for later applications
- Basic routines necessary for preparing measurement data are already installed



The MST 200 software carries out measurement functions for LMS laser scanners. This software handles customer-specific measurement functions quickly, efficiently and cost-effectively.

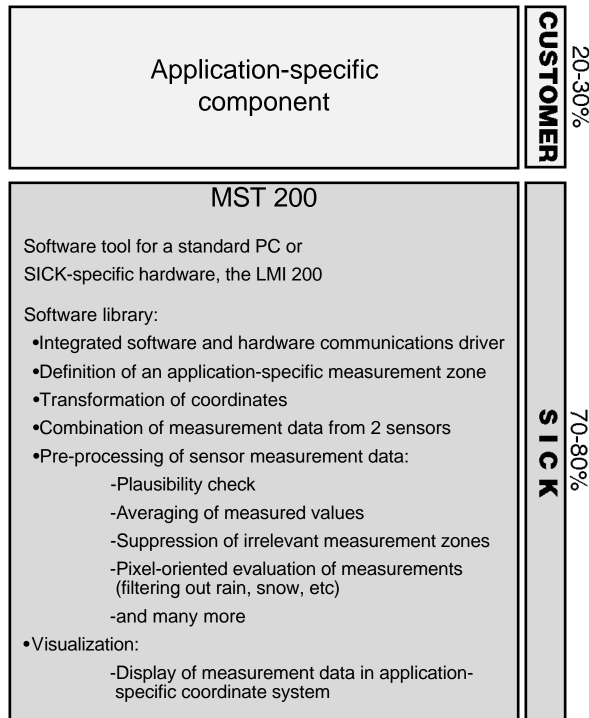
MST 200 software has two pre-installed drivers for real-time communication with up to two laser scanners. Handling of the application can begin directly after transforming coordinates and defining an application-specific measurement framework. The functions library already includes important filter functions such as cutting out irrelevant measurement zones. The MST 200's tiered structure also allows for later applications.

SICK has provided 70-80% of the programming necessary to use the MST 200 software. The user only needs to deal with the application-specific portion of the application. Various basic routines necessary for preparing measurement data are already installed.

NOTE: Users of MST 200 will need to have knowledge of MS Visual C++ programming.

MST Software Specifications & Requirements

Computer	Pentium 133 MHz
Hard Disk Space	4 MB of available disk space
Disk Drive	CD drive
Memory Requirements	Recommended 16 MB RAM
System Software	Windows™ 95, 98 or NT
Mouse	Optional but recommended
LMI Interface / Compatibility	LMI 200



SICK
SICK, Inc.
6900 West 110th Street
Minneapolis, MN 55438 USA
Phone: (952) 941-6780 Fax: (952) 941-9287

Back to Search

LMS 291-S05 (1018028)

- Printer Friendly
- Physical Dimension Drawings
- Related Products
- Documentation

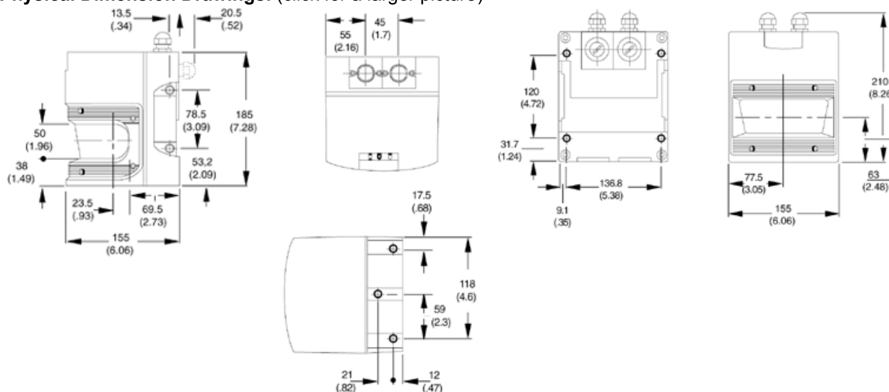
Product Description: Indoor, data interface: RS 232 / 422, 180º, range: 1...45 m, fog correction, switching outputs: 3, IP 65, gray case

Product Specifications

Range (Maximum / 10%)	Maximum 80 m (262.5 ft) / 30 m (98.4 ft)
Angular Resolution	0.25° / 0 .50° / 1.00° (selectable)
Response Time	53 ms / 26 ms / 13 ms
Measurement Resolution	10 mm (.39 in)
System Error (Environmental Conditions: Good Visibility, Ta=	Typical ± 60 mm (mm-mode), range 1…4 m; typical ± 35 cm (cm-mode), range 4…20m
Statistical Error Standard Deviation (1sigma)	Typical ± 10 mm (at range 1…20 m / ³ 10% reflectivity / £ 5 kLux)
Data Interface	RS 232 / RS 422 (configurable)
Transfer Rate	9.6 / 19.2 / 38.4 / 500 kBd
Switching Outputs	3 x PNP, typical 24 V DC OUT A, OUT B maximum 250 mA, OUT C maximum 100 mA
Supply Voltage (Scanner-electronics)	24 V DC ± 15% (maximum 500 mV ripple); current requirements maximum 1.8 A (includes output load)
Supply Voltage (heating, LMS 211/221 only)	24 V DC (maximum 6 V ripple); current requirement maximum 6 A (cyclic)
Current Consumption	Approx. 20 W (without output load)
Electrical Protection Class	Safety Insulated, protection class 2
Laser Protection Class	1 (eye-safe)
Interference Resistance	IEC 801, part 2-4; EN 50081-1/50081-1/50082-2
Ambient Operating Temperature	0...50° C (32...122° F)
Storage Temperature	-30...70° C (-22...158° F)
Enclosure Rating	IP 65 / NEMA 4
Weight	approx. 4.5 kg (9.9 lb)
Dimensions	155 x 156 x 210 mm (6.1 x 6.14 x 8.27 in)
Environment	Indoor

Back to Top

Physical Dimension Drawings: (click for a larger picture)



Back to Top

Related Products:
Accessory: 7026897 LMS PCMCIA Card
Accessory: 6022515 Interface Card
Accessory: 6020756 Scanfinder LS70B
Accessory: 6011807 ISA PC Interface Card LMS
Accessory: 6011156 Main Adapter 24 V DC/10 A
Accessory: 2020926 3rd Axis for Fine Adjustment
Accessory: 2020925 Fine Adjustment Mounting Set
Accessory: 2018965 Connection Set 3 LMS
Accessory: 2018964 Connection Set 2 LMS
Accessory: 2018963 Connection Set 1 LMS
Accessory: 1016761 LMI 200

Back to Top

Back to Search

PLS 101-312 (1016066)

Printer Friendly
Physical Dimension Drawings
Related Products
Documentation

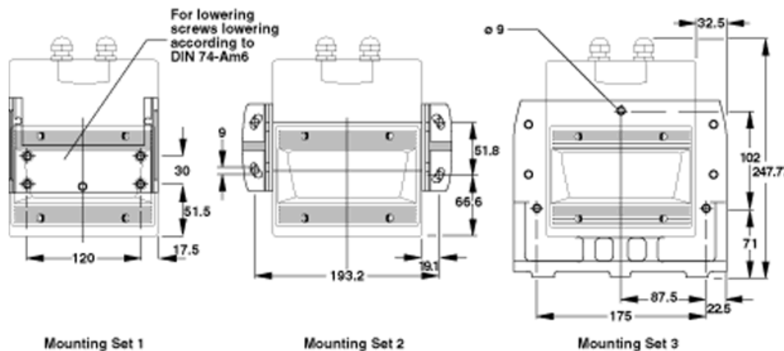
Product Description: PLS with RS 232/422 computer interface. 180° scanning field, safety certified Type 3.

Product Specifications

Scan Area	180 °
Safety Zone Radius	4 m (13 ft)
Warning Zone Radius	50 m (164 ft)
Maximum Number of Protective Field Sets	8 (with LSI); 1 (without LSI)
Maximum Number of Monitoring Cases	15 (with LSI); 1 (without LSI)
Angular Resolution	0.50°
Response Time	80 ms (minimum)
Safety Category	Type 3
Enclosure Rating	IP 65 / NEMA 4
Supply Voltage	24 V DC
Output Type	2 X PNP Semiconductor, 24 V DC, 250 mA
Laser Class	1

Back to Top

Physical Dimension Drawings: (click for a larger picture)



Back to Top

Related Products:
Alternate Part: 1023546 S30A-6011BA
Alternate Part: 1023547 S30A-6011CA
Alternate Part: 1019600 S30A-6011DA
Alternate Part: 1023548 S30A-6011EA
Mounting Brackets: 7027025 PLS-MB123
Mounting Brackets: 2015623 PLS-MB1
Mounting Brackets: 2015624 PLS-MB2
Mounting Brackets: 2015625 PLS-MB3
Relays & Interfaces: 6024916 UE48-2OS3D2
Power Supplies: 6010361 PLS-PS-25
Power Supplies: 6010362 PLS-PS-40
Power Supplies: 7022755 PS-20 24 V DC
Replacement Part: 2022271 PLS/LMS Window
Accessory: 7024047 PLS Keylock Assembly
Accessory: 1016063 LSI 101-112
Accessory: 1016065 LSI 101-114
Accessory: 2016184 PLS-CS1
Accessory: 2016185 PLS-CS2
Accessory: 2016186 PLS-CS3

Accessory: 2016187 PLS-CS4
Accessory: 2016188 PLS-CS5
Accessory: 2016189 PLS-CS6
Accessory: 2016190 PLS-CS7

Back to Top