

VOXEL BASED SENSING AND PERCEPTION SYSTEM FOR AN AUTONOMOUS
VEHICLE USING AN ARTICULATED LADAR

By

RYAN CHILTON

A THESIS PRESENTED TO THE GRADUATE SCHOOL
OF THE UNIVERSITY OF FLORIDA IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

UNIVERSITY OF FLORIDA

2010

© 2010 Ryan Chilton

To my Lord and Savior Jesus Christ who gives me the strength to do all things
and to my loving parents

ACKNOWLEDGMENTS

I would especially like to thank Dr. Crane for providing me and all the other students in the Center for Intelligent Machines and Robotics with such an open environment to learn and explore one's interests. I am thankful for the opportunity he presented to me as an undergraduate to help out and learn about the laboratory before making my decision to go to graduate school. I would like to thank Jae for helping me learn CIMAR programming practices and helping me get adjusted to the lab as well as for being a good friend. I would like to thank Vishesh for explaining difficult concepts to me over and over again and always being able to help me understand new topics. I would like to thank Cuk for pushing me to dive into research. I would like to thank Shannon for giving me difficult tasks and for being able to get me whatever hardware I needed by the very next day. I would like to thank Drew and Nick for some great intellectual, sociological, religious, but often absurdly irrelevant conversations. Special thanks to Drew for playing racquetball with me and to Nick for being able to answer every single one of my programming questions. Jonathon, your sense of humor always got me laughing and went a long way to making the lab environment more fun.

TABLE OF CONTENTS

	<u>page</u>
ACKNOWLEDGMENTS.....	4
ABSTRACT.....	11
CHAPTER	
1 INTRODUCTION	13
Problem Statement.....	13
Ladar Sensors	15
Related Work.....	16
2 IMPLEMENTATION.....	21
Vehicle System Architecture.....	21
Data Collection and Merging.....	22
Storage Model Selection.....	23
Voxel and Block Data Structures	27
Coordinate Transformations	30
3D Bresenham Line Function	33
Voxel Occupancy Scheme.....	35
Data Fusion	37
Finding Ground Surface.....	37
Estimating the Height of Above Ground Objects.....	44
Finding Trees.....	45
Saving Objects in the Local Knowledge Store	51
3D Imaging techniques	52
Blur.....	52
Differencing	53
Hough Transform	53
Smart Sensor Grid	55
Software Architecture.....	55
3 RESULTS	57
Static Data Collection.....	57
Dynamic Data Collection.....	59
Ground Plane Identification.....	61
Brush Height Estimation and Tree Detection	62
Hough Plane Extraction	63
4 CONCLUSION.....	65
LIST OF REFERENCES	67

BIOGRAPHICAL SKETCH..... 69

LIST OF FIGURES

<u>Figure</u>		<u>page</u>
1-1	An All Purpose Remote Transport System (ARTS) that has been converted to an autonomous vehicle.....	14
1-2	A graph of a single scan from a Ladar.	15
1-3	A Sick LMS-151 Ladar which employs a rotating mirror within a weather resistant housing.	16
1-3	Ladar beam being used to measure occupied space (red) and free space (green).....	19
2-1	Interaction among JAUS software components.....	22
2-2	A point cloud broken up into smaller manageable subregions.	24
2-3	A voxel grid arranged into blocks and placed around the vehicle's location with a fixed orientation.....	29
2-4	Vehicle orientation convention based on the JAUS standard.	30
2-5	Coordinate systems attached to the two moving pan-tilt bodies and the Ladar.	32
2-6	Bresenham's line function is used to determine all the cells through which a line segment passes in two dimensions.	34
2-7	Ray tracing through voxels performed by finding the intersection points with planes.....	35
2-8	The pyramid under a candidate ground surface voxel.....	38
2-9	The pyramid under a candidate ground voxel which does not pass the ground surface test.....	39
2-10	The original scene before applying the ground surface filter.	41
2-11	The scene after applying the ground surface filter.....	41
2-12	A ground representation technique using tiles of average ground heights.	42
2-13	Two dimensional interpolation performed using a sequence of one dimensional interpolations.	42
2-14	The result of performing interpolation and extrapolation on a set of ground surface voxels.....	43

2-15	The final ground surface representation.	43
2-16	A sample scene with brush and a tree trunk.....	45
2-17	The above ground density plot of a scene with brush and a tree trunk.....	46
2-18	A convolution performed with a 3 by 3 kernel. The left array is the image data, the 3 by 3 array in the center is an example convolution kernel, and the array on the right shows the resultant value of the convolution at this one position.	47
2-19	Convolution kernel used in tree trunk detection.....	48
2-20	Image of above ground densities after the convolution filter has been applied. .	48
2-21	The image of above ground densities after the convolution filter and the threshold filter have been applied.....	49
2-22	Psuedocode of the clustering process.....	50
2-23	A three dimensional blurring effect applied to voxel data.	52
2-24	Hough plane extraction.....	54
2-25	Process diagram. The clock symbolizes a timer which triggers the functions it points to.	56
3-1	A SICK LMS-151 LADAR mounted on a pan-tilt actuator on top of a fixed tripod.	58
3-2	Scene 1: Part of the University of Florida campus with a large tree.	58
3-3	Scene 2: A section of the University of Florida campus with rocks and brush....	58
3-4	Scene 3: The side of a building.	59
3-5	The ratio of occupied voxels to total voxels and free space voxels to total voxels as data is collected over time. The data is taken from one of the blocks in scene 1.	59
3-6	The vehicle used for collecting Ladar data.	60
3-7	A scene with trees observed aboard a moving vehicle.....	60
3-8	A scene with trees and brush (to the right, off camera) observed aboard a moving vehicle.....	61
3-9	Results of ground surface identification among heavy brush.	61

3-10	Results of ground surface identification with sparse data.....	62
3-11	Results of brush height estimation and tree detection on data collected from a static location. Green voxels represent the ground surface, dark green the brush, and purple the tree trunks.....	63
3-12	Results of brush height estimation and tree detection on data collected from a moving vehicle. Green voxels represent the ground surface, dark green the brush, and purple the tree trunks.....	63
3-13	Hough plane extraction used on ground voxels.....	64

LIST OF ABBREVIATIONS

GPOS	Global Position and Orientation System
JAUS	Joint Architecture for Unmanned Systems
LADAR	Laser Detection and Ranging
SVM	Support Vector Machines
TSS	Terrain Smart Sensor

Abstract of Thesis Presented to the Graduate School
of the University of Florida in Partial Fulfillment of the
Requirements for the Degree of Master of Science

VOXEL BASED SENSING AND PERCEPTION SYSTEM FOR AN AUTONOMOUS
VEHICLE USING AN ARTICULATED LADAR

By

Ryan Chilton

December 2010

Chair: Carl Crane
Major: Mechanical Engineering

An autonomous vehicle's performance can only be as good as the information it has about its surroundings. In order to make intelligent decisions while operating in an unknown environment, it is necessary to rely on information sensors can glean from the surroundings. One type of sensor which has become quite commonplace in autonomous vehicles and smaller robots is a Ladar. A Ladar is a laser based range finder which collects distance measurements at discrete angular steps, usually in a two dimensional plane. This type of sensor is especially useful because it has a relatively long range (10-300 meters), a good degree of accuracy, and can scan at rates of 10 to 50 Hz. These features provide a wealth of information that if processed correctly can provide valuable information to a robotic vehicle's planning system.

This thesis deals with the implementation of a Ladar based perception system for an autonomous vehicle operating in an outdoor environment. Instead of rigidly fixing the Ladar sensor to the vehicle frame, the sensor is actuated to rotate about an axis so that the angle of tilt can be controlled. This tilting motion, along with the appropriate coordinate transformations, allows for the collection of three dimensional data of the environment.

The details of the data collection process and data storage model are presented. Also the processing algorithm which attempts to classify objects such as ground surface, brush, and trees is outlined. In addition to the processing algorithm used in this research, other methods from related work are presented. Additionally, the software architecture for the perception system is described and how it fits into the overall system architecture used for testing. The results of the classification algorithm are shown and evaluated based on a human's ability to identify objects in the same scene. Benefits and drawbacks of the design are also discussed.

CHAPTER 1 INTRODUCTION

Problem Statement

The ability for an autonomous vehicle to operate in an unknown environment depends upon the system's ability to gather data from the surroundings using sensors, the ability to interpret this information, and the ability to make decisions based on this knowledge. Since an autonomous system's decisions can only be as good as the information these decisions are based upon, the data collection and interpretation process is critical to both task achievement and overall performance. With the increasing usage of autonomous vehicles operating in off road environments, the ability to sense both obstacles and traversable surface are critical to simple navigational tasks. The 2005 DARPA Grand Challenge is an example of a government sponsored competition in which vehicles were provided with a high level pre-specified route to follow, but the task of navigation could not be accomplished without the ability to sense information about the surrounding terrain and reflexively plan and drive around obstacles.

Much of this work is driven by requirements for an autonomous range clearance project. The vehicle platforms on which the system will be deployed include tracked vehicles such as the one shown in Figure 1-1, and a four wheeled rock crawler type vehicle. They are slower moving vehicles so the speed at which the data is processed is not as critical as is in some other applications; however, mission objectives necessitate a higher level of interpretation of the information than simple obstacle detection.

Besides providing information about where it is safe to travel, it is sometimes desired to understand more about an autonomous vehicle's surroundings. The location of other vehicles, traffic lights, pedestrians, and buildings are examples of items that would be of interest in an urban environment. In a rough terrain or vegetated environment, the location and size of brush, trees, and rocks, could be of interest depending on the mission objectives. In order to interpret the sensor data and provide the autonomy with a greater understanding, the process of classification must be performed in which data is assigned to a specific class or type such as trees or terrain. Classification is an essential step which can provide the planning element of an autonomous system the information it needs to make intelligent decisions.

One of the most common sensors seen on autonomous vehicles today is a type of laser range finder called a Ladar. Ladar sensors are excellent at providing fast and accurate range information in indoor and outdoor settings.



Figure 1-1. An All Purpose Remote Transport System (ARTS) that has been converted to an autonomous vehicle.

Ladar Sensors

The LADAR sensor reports range data by emitting an infrared laser beam and measuring the time required for the reflected light to be sensed. This time of flight measurement process is repeated sequentially at regular angular increments in a sweeping motion, usually by using a rotating lens or mirror. The result is a two dimensional cross section of distance measurements in the plane of the laser beam. With scanning rates of 10-50 Hz, the sensors provide high speed and accurate data with distance errors less than 4 cm. By articulating the LADAR about an axis, and performing the appropriate coordinate transformation, 3D data can be collected. Similarly, if the Ladar is mounted aboard a moving robot, and the robot's global position and orientation are known, the appropriate transformations provide 3D data in the global coordinate system.

The laser beam used by the Ladar is actually a cone, which diverges so as to maintain complete angular coverage instead of leaving angular gaps in between the sweeping beam as pulses are emitted. A sample scan from a Sick LMS-151 Ladar is shown in Figure 1-2 below. The sensor itself is pictured in Figure 1-3.

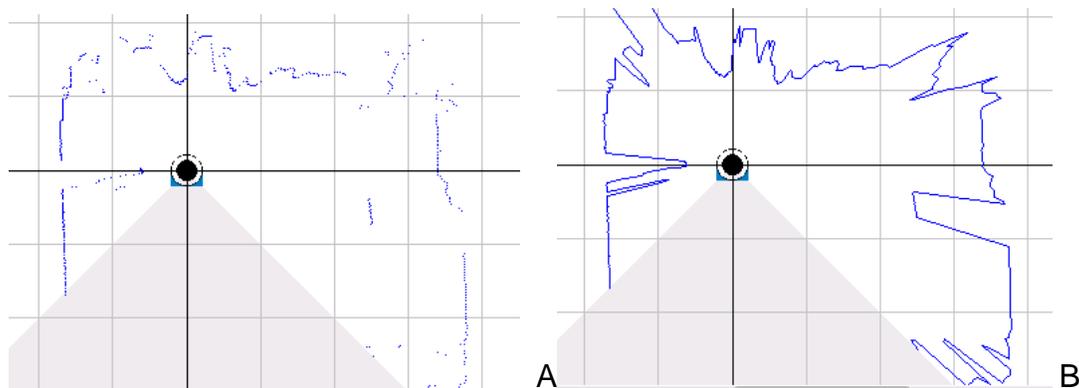


Figure 1-2. A graph of a single scan from a Ladar. A) The scan shown with only the data points drawn. B). The scan shown with lines connecting the data points.



Figure 1-3. A Sick LMS-151 Ladar which employs a rotating mirror within a weather resistant housing.

The use of Ladar range scanners in outdoor environments have proven to be capable of providing usable geometric information for vehicle navigation [1]. One advantage of laser range scanners is the accuracy of the reported range data which is often much more accurate than stereo vision could attain at long distances. Another feature that could be useful for certain tactical applications is that the scanners are fairly independent of ambient light and will even work in the dark. Another feature, the large amount of data which is reported, creates the need for efficient methods of data processing and storage. Approaches for storing and utilizing this vast amount of information are presented in the next section.

Related Work

Much work has been conducted in recent years using Ladars to provide information about a mobile robot's environment. Systems designed for both urban and off road environments have been developed and tested [1], [2]. The use of Ladars have sometimes been paired with other sensors such as cameras to aid in the classification

process, but this presents another problem related to the fusion of data from different sensor types [3]. Most work has been performed using three dimensional data collected with a single articulated Ladar, but researchers have also fused data from multiple sensors with careful synchronization of incoming data [4].

One of the difficulties associated with three dimensional data collected with Ladars is the vast amount of information collected. The sheer volume of data makes the collection and storage process as well as the classification process very intensive. If all the data points are saved, then the classification process must operate on more and more data as scans are collected in the same area. However, if the data is to be merged or reduced, more effort is needed to ensure that valuable information is not lost and that the merging process is performed efficiently. Researchers in [2] describe the difficulties of using Ladar sensors with regard to the acquisition process, occlusions, objects with variable reflectance, and data redundancy. Techniques which optimize efficiency when accessing features of three dimensional data have been presented in [5].

Much work has been done related to classifying sensor data collected in a forested environment [6], [7]. The somewhat cluttered and inconsistent environment makes perception a challenging problem. Several algorithms tested in [8] were reported to take between 15 to 45 seconds for every 100,000 points of data. For context, a typical Ladar sensor can collect this many points in a little over three seconds. Nevertheless, algorithms that only perform classification on a single area once or that have efficiency improvements and simplifications necessary for real time on board processing have been demonstrated [9]. Another problem related to the use of articulated laser

scanners is that the information gathered beyond a certain range has higher error due to uncertainty in the vehicle's orientation and sensor's orientation [7].

There are typically two different storage models used to represent the data collected from 3D sensors. One model is based on the point cloud concept; the other is based on the voxel concept. Each has its own benefits and drawbacks. The point cloud model consists of a set of points in three dimensional space which represent points on the surface of objects. The point cloud does not have any explicit linking or connectivity defined between points. The voxel model, on the other hand, consists of a three dimensional grid of cells called voxels. The name is a shortened version of the term "volumetric pixel". Each voxel can contain information about the space within it. In general, this information could include color, density, temperature or other data, but in the application of sensing with a laser range sensor, we are provided with distances to objects in various directions; hence, an appropriate parameter to store is the voxel's occupancy. The occupancy is the belief that the space within the voxel is occupied with an object or some part of an object. This value can be discrete (occupied vs. unoccupied) or continuous (an occupancy value or probability). Successful implementations using both the point cloud model [3] and the voxel model [7] have been created and tested.

The Ladar reports the distance to the first reflection distance at each angle increment. This data not only provides the location to the nearest object along that line but also implies that the space along the beam's line of travel is unoccupied. Figure 1-3a shows the voxels that can be regarded as vacant (green) and the voxel that can be regarded as occupied (red).

The use of the implicit information given by the beam's line of travel is useful for two reasons. Firstly, it accounts for changes in a non-static environment. Since it is necessary to keep the data persistent over time, an object which has been removed from the scene will persist as an artifact unless the beam information is used to reflect its disappearance as the beams pass through the now unoccupied space. The case of a removed object is shown in Figure 1-3b with the tree having been cut down. As the beams from the LADAR pass through the region of space that once was occupied by the tree, the area becomes marked as free space again. Secondly, using the entire beam can allow for the evaluation of a region's density. A region which contains both points which reflect beams and beams which pass through it can be considered to be semi-occupied as a result of tall grass or light brush similar to the work presented in [10].

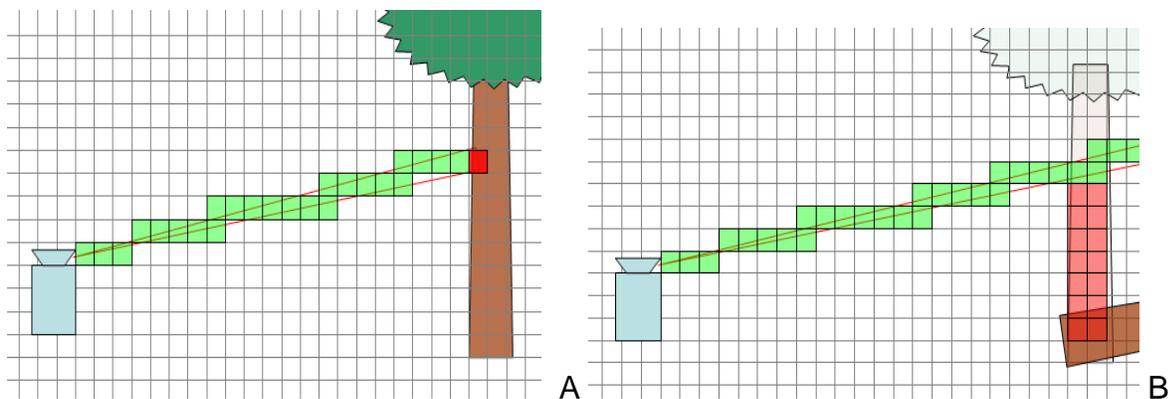


Figure 1-3. Ladar beam being used to measure occupied space (red) and free space (green) when A) a tree is within detection range and B) once the tree is removed.

After data has been collected and stored, classification can be performed, either off line or on board the robot. Since the processed data is needed for navigation and other mission specific tasks, it is necessary to be able to perform the classification on board the robot. There are many classification methods which apply to three

dimensional spatial data including Support Vector Machines (SVM), maximum likelihood estimation (ML), Associative Markov Fields (AMF), and Neural Networks, as well as more deterministic approaches. One classifier used in [11] relies on saliency features which basically quantify the shape of a local neighborhood of data. This is then grouped into one of three classes: linear, surface, and clutter. Segmentation is another important process in classification. A segmentation algorithm functions to group data according to objects. A segmentation approach presented in [12] uses Markov Random Fields to segment point cloud data with better results and more consistent grouping than using an SVM classifier alone. Typically, classification algorithms work on a point-wise approach, which sometimes causes various parts of a single object to be classified differently. Researchers have used a Markov network to ensure consistent classification for contiguous regions, which gives noticeable improvements in classifying at junctions and for small components [13].

Other research has been done to specifically aid vehicle navigation. Analysis of the ground surface and detection of negative obstacles is critical to path planning [14]. Automating heavy equipment has successfully been demonstrated for both pre-planned paths and reactive path planning based on sensor data [15].

CHAPTER 2 IMPLEMENTATION

Vehicle System Architecture

The perception system was developed to operate with other software components which comprise an autonomous vehicle's intelligence system. The system for which it was designed is based on the Joint Architecture for Unmanned Systems, or JAUS, framework. This provides a common specification for software component communication and behavior. The overall responsibility of the Ladar based sensor system is to collect raw data from the sensors and reduce this to usable information which the planning components can use to make decisions.

The interconnections among the various components are illustrated in Figure 2-1. The component responsible for reporting the vehicle's global position and orientation is known as GPOS. The Knowledge Store is a software component which acts as a database for storing and retrieving information. Either a priori information or information gathered at run time can be saved in the Knowledge Store. The Local Reactive Planner is responsible for planning and executing vehicular locomotion. The Reactive Payload Planner is similarly responsible for controlling any manipulator or payload on the robot. The Subsystem Commander is an overarching component which makes high level decisions based on sensor findings or human input and sets the behavior of the system. The Human Machine Interface provides a method for interacting with a human supervisor. The Terrain Smart Sensor, or TSS, is the title given the software component discussed in this thesis.

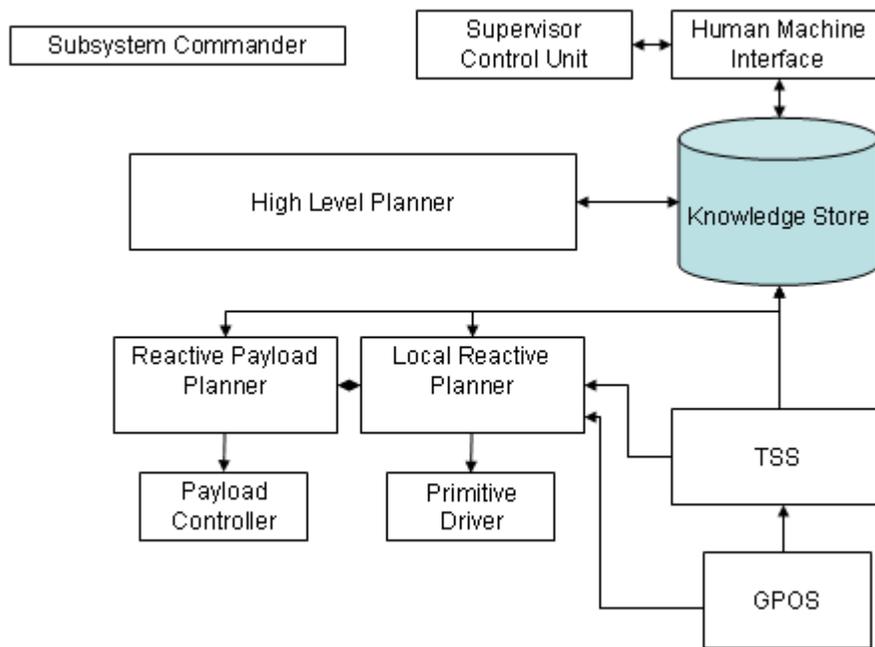


Figure 2-1. Interaction among JAUS software components.

Data Collection and Merging

The process of collecting data from the Ladars involves a substantial amount of information. Since 3D data is constructed using 2D sensors, the data must be kept persistent over time. Data persistence is necessary for building up a well populated scene, using data from multiple scans. A well populated scene can be useful for contextual classification. Contextual classification makes use of an element's location relative to other objects in the scene such as the ground surface, instead of simply computing saliency values using a limited neighborhood as in [10]. Merging new data with old data in a way that saves information from both samples is important for two reasons. Firstly, the incoming data from the Ladar has the ability to grow very large very quickly if the new data is not merged with the old data. Secondly, if the new data is not used to update the old data, there is the tendency for the data to become "blurry"

due to drift in the vehicle's estimated position as multiple range measurements from the same object are recorded at slightly different locations.

Storage Model Selection

As described in Chapter 1, there are two different approaches for storing the collected 3D data, point clouds and voxels. The choice between a point cloud representation and a voxel based representation is usually a trade off between memory requirements and processing speed requirements. On the one hand, the point cloud storage method saves the data in its purest form and usually requires less memory since memory for each point can be allocated dynamically. On the other hand, the voxelized storage method requires a large amount of memory to be allocated before data collection takes place, but the regularity of the data provides quick access to data in specific geometric regions. The speed of accessing subsets of the data is important for classification techniques which compute feature vectors based on data within a local neighborhood.

The data collection process consists of several steps. The first step is the geometric transformation of the beam data to a globally fixed reference frame. The second step involves tracing the ray from the Ladar to the point of reflection in order to update the current data with this known free space information. The final step is to use the location of the beam's reflection point as an indicator of occupancy in the data.

The last two steps will differ in implementation based on the storage model chosen. While there are different ways to implement both a point cloud model and a voxel model, two reasonable methods are chosen for comparison. The first is a voxel based model with the voxels being represented as integer occupancy values stored in an array in contiguous memory. The second is a point cloud in which space is divided

into smaller manageable sub-regions which each contain a linked list of all points within them. This geometric organization of smaller linked lists facilitates faster access time for a specific region of space. Figure 2-2 illustrates the point cloud data organized into sub-regions.

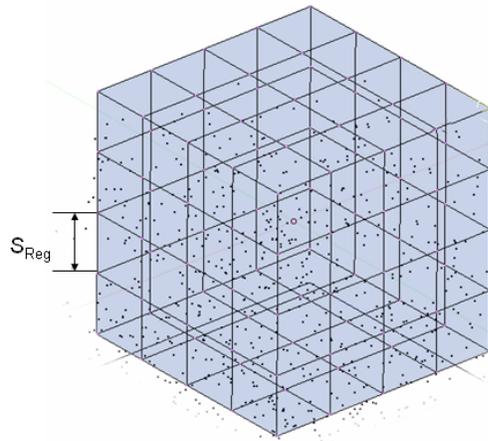


Figure 2-2. A point cloud broken up into smaller manageable subregions. The side length of each subregion is designated as S_{Reg} .

Computationally speaking, there are three important tasks of concern: the time required to insert a new beam to the data structure, the time required to get a neighborhood of points/voxels at a particular location, and the storage space associated with storing the data. In order to compare the two storage methods performance in these areas, the following variables are defined.

S_{vox} : side length of voxel

S_{Reg} : side length of point cloud sub - region

\bar{n} : average number of points in a point cloud region

L : length of a beam segment from the Ladar measurement

S_N : side length of neighborhood used for classification

Because of the free space assumption for the space along a Ladar beam, the process of adding a new beam to the data requires that all voxels/points that are within

some distance of the beam be modified. For the voxel model, the occupancy probability of those voxels should be decreased. For the point cloud model, the points could either be removed or the confidence in the point's existence could be decreased. Regardless of how the beam's free space information is used, the points/voxels in close proximity to this line must be retrieved. Assuming a beam of length L , with an orientation given by the unit vector $a\hat{i} + b\hat{j} + c\hat{k}$, the number of voxels that contain some part of this beam is given by Equation 2-1.

$$n_v = \frac{aL}{S_{vox}} + \frac{bL}{S_{vox}} + \frac{cL}{S_{vox}} \quad (2-1)$$

Using the triangle inequality, we can upper bound the number of accessed voxels with Equation 2-2.

$$n_v \leq \frac{\sqrt{3}L}{S_{vox}} \quad (2-2)$$

For the point cloud case, the number of points visited will be dependent on the size of the point cloud sub-region and the average number of points within each sub-region. The sub-regions through which the beam passes can be identified directly, but since the points within each sub-region are contained within a linked list, every point must be tested to see if it lies within a threshold distance of the beam. (This distance calculation consists of taking the norm of a cross product.) Hence the average number of points visited for this type of storage model, n_p , is given by Equation 2-3. Note that the average number of points in a point cloud sub-region will increase as more data is collected.

$$n_p \leq \bar{n} \times \frac{\sqrt{3}L}{S_{Reg}} \quad (2-3)$$

The process of inserting the reflection point into the data is simple in either model. For voxels, the voxel containing the reflection point is modified to have a higher occupancy probability, whereas for a point cloud, the new point is added to the linked list associated with that sub-region of space.

The last comparison is that of data storage size. The size of the voxel based storage model depends solely on the size of the volume for which data is allocated and the resolution of the 3D grid. The size of the point cloud based storage model is essentially based on the number of points collected. If the voxel occupancy is computed based on the number of beams which pass through the voxel and the number of beams which terminate in the voxel as in [7], the occupancy probability or density can be formulated as

$$p = \frac{n_h}{n_h - n_p} \quad (2-4)$$

where n_h is the number of beam reflections within the voxel and n_p is the number of beams which pass entirely through the voxel. This formulation requires two counters. If 32-bit unsigned integers are used, each voxel would require 8 bytes. However, if we note that only the ratio is of interest, we can use a single byte to hold each counter, and then when one of the counters approaches the maximum value of a byte, the algorithm can scale both of the counters by a fractional number. Assuming that this implementation is used, the amount of memory needed to represent a volume is given by

$$m_{vox} = \frac{volume}{S_{vox}^3} \times 2 \times size(byte) \quad (2-5)$$

Assuming that a point in a point cloud is represented as three floating point numbers and that they are stored in a singly linked list, the amount of memory required to hold the data is dependent on the number of points as shown in Equation 2-6.

$$m_{PC} = n_{\text{points}}(3 \times \text{size}(\text{float}) + \text{size}(\text{pointer})) \quad (2-6)$$

For this implementation, the voxel model was selected for several reasons. Firstly, it makes the merging process faster in the presence of large amounts of data in a small region. Secondly, the voxel model permits the data size stored in memory to be constant, and therefore, expensive dynamic memory allocation such as for a point cloud stored in a linked list or tree structure can be avoided. Thirdly, classification techniques can be simplified because of the geometric regularity of the data and image processing techniques can be extended to the 3D voxel space. Fourthly, data fusion between multiple sensors is aided by the use of voxels because the raster format makes the fusion process much simpler than if the data sets were in vector format. Even data collected by other types of sensors such as cameras can be easily merged once the data is stored in the common format. It is simply a matter of assigning weights to the cells based on the sensor's confidence and then taking the weighted average of the occupancy values (or even color values in the case of vision information).

Voxel and Block Data Structures

The voxels are organized into groups of voxels in the shape of rectangular solids called blocks. Each voxel has a single number assigned to it which represents the occupancy value. A voxel can be assigned a value which indicates that no information is known about its occupancy, or it can be assigned a scalar value which indicates that the region is either occupied or unoccupied with some certainty. While a single number

is used in this implementation, a set of values could just as easily be assigned to each voxel to represent other sensed information such as reflectance or color value. The voxel values are stored within the block structures as an array of contiguous bytes. (The three dimensional data is actually stored in a one dimensional array with appropriate striding to speed up access time.) If a three dimensional array were used, the access time would need to include two additional memory look ups for every access to a voxel value.

The Blocks of voxels are fixed in the global coordinate system. The Block class has members which define the origin in terms of Universal Transverse Mercator coordinates. The Universal Transverse Mercator coordinate system (UTM) divides the earth's surface into 60 different "zones" in which a 2D Cartesian coordinate system can be applied with relatively little distortion. All the data collected from the Ladar and all the voxel locations can be represented in these coordinates. The Block class also has variables which define the resolution of the voxels and the dimensions of the overall block. The advantage of breaking up the space into blocks is that the region around the vehicle can be dynamically buffered with blocks. Since the voxel grid is fixed to the earth instead of fixed to the vehicle, the voxels do not move as the vehicle travels, therefore, new blocks must be created as the vehicle moves into new areas. The decision to fix the voxels in the global frame instead of the vehicle's frame alleviates the need to transform the data through time as the vehicle rotates and translates. This calculation would become more and more expensive as the amount of saved data increased in the scene. Conversely, by keeping the voxels fixed in the global frame, the data need only be transformed from the sensor's frame to the global frame once before

it is added to the voxel data. Figure 2-3 illustrates the case of nine blocks arranged around the vehicle's current location. The process of buffering blocks around the vehicle is simply a matter of deciding the size of the working area desired around the vehicle and then periodically checking to see if blocks need to be added in front of the vehicle as well as checking if blocks are out of range and should either be freed from memory or saved to disk. There are two methods for managing the blocks which were implemented. The first deletes blocks when the vehicle moves out of range and creates fresh blocks when needed. The second saves blocks to the hard drive when the vehicle moves out of range and when new blocks are needed, they will be loaded from disk if they exist or created fresh if not. The first option is currently used because long term drift in the global position can cause conflicts in data collected several minutes or several hours ago resulting in data mismatch.

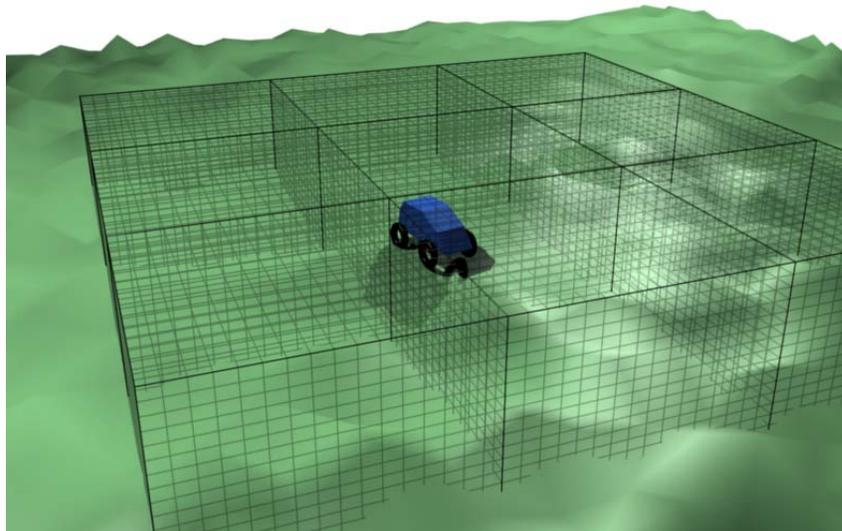


Figure 2-3. A voxel grid arranged into blocks and placed around the vehicle's location with a fixed orientation.

Coordinate Transformations

In order to utilize the data coming from the Ladars, the proper coordinate transformations must be applied so that the data can be represented in the global coordinate system. In order to transform the data into the global coordinate system, both the sensor's position and orientation with respect to the vehicle must be known as well as the vehicle's position and orientation in the global coordinate system. The standard JAUS coordinate system is attached to the vehicle as shown in Figure 2-4 below. The order of rotations is first a rotation of Ψ about the vehicle's z-axis (yaw), followed by a rotation of θ about the vehicle's new y-axis (pitch), followed by a rotation of ϕ about the vehicle's new x-axis (roll).

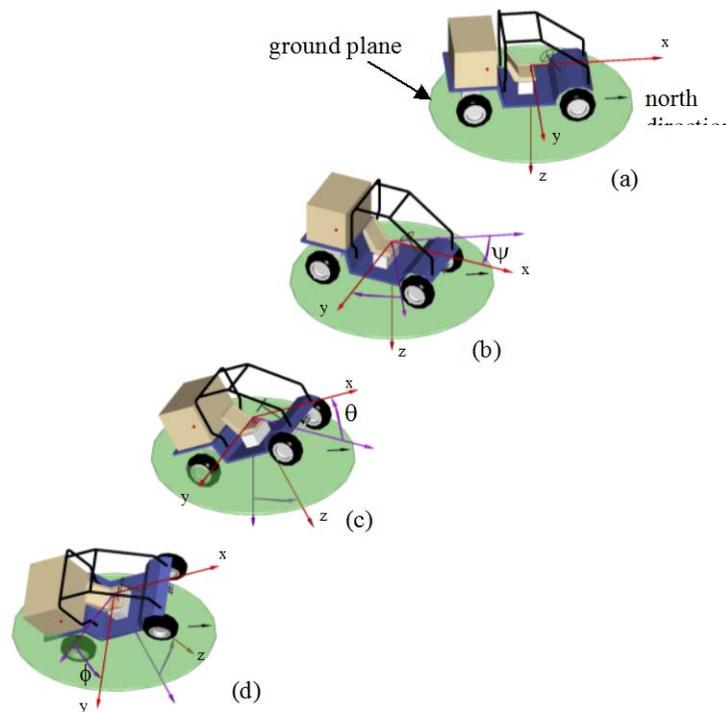


Figure 2-4. Vehicle orientation convention based on the JAUS standard.

The coordinate system chosen for the global reference frame is as follows. The origin is located at the origin of the currently occupied UTM zone. The x-axis is to the east, the y-axis is to the north, and the z-axis is away from the earth's center. Using the UTM system allows Cartesian coordinates to be applied to a localized area of the earth's surface without accounting for the complexity of the earth's curvature, a valid assumption for the scale of ground vehicles.

The coordinate transformation is performed using homogeneous coordinates. The rotation matrix from the vehicle to the global coordinate system is given in Equation 2-7. The first matrix is a result of the orientation of the vehicle's coordinate system with respect to the global coordinate system when the roll, pitch, and yaw variables are all zero. In this configuration, the vehicle's x-axis is to the north, the vehicle's y-axis is to the east, and the vehicle's z-axis is downward. Equation 2-8 defines the transformation matrix from the vehicle to the global coordinate system.

$$R_V^G = \begin{bmatrix} 0 & 1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & -1 \end{bmatrix} \begin{bmatrix} \cos(\Psi) & -\sin(\Psi) & 0 \\ \sin(\Psi) & \cos(\Psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \quad (2-7)$$

The transformation matrix is then as follows.

$$T_V^G = \begin{bmatrix} & \textit{Easting} \\ \begin{bmatrix} R_V^G \end{bmatrix} & \textit{Northing} \\ & \textit{Altitude} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2-8)$$

The rest of the information needed to transform the data comes from the Ladar's orientation with respect to the vehicle. The Ladar is mounted to the vehicle at some arbitrary location and has two additional degrees of freedom provided by a panning and

tilting mechanism with two intersecting perpendicular axes. The figure below shows the coordinate systems assigned to the pan/tilt unit and the Ladar.

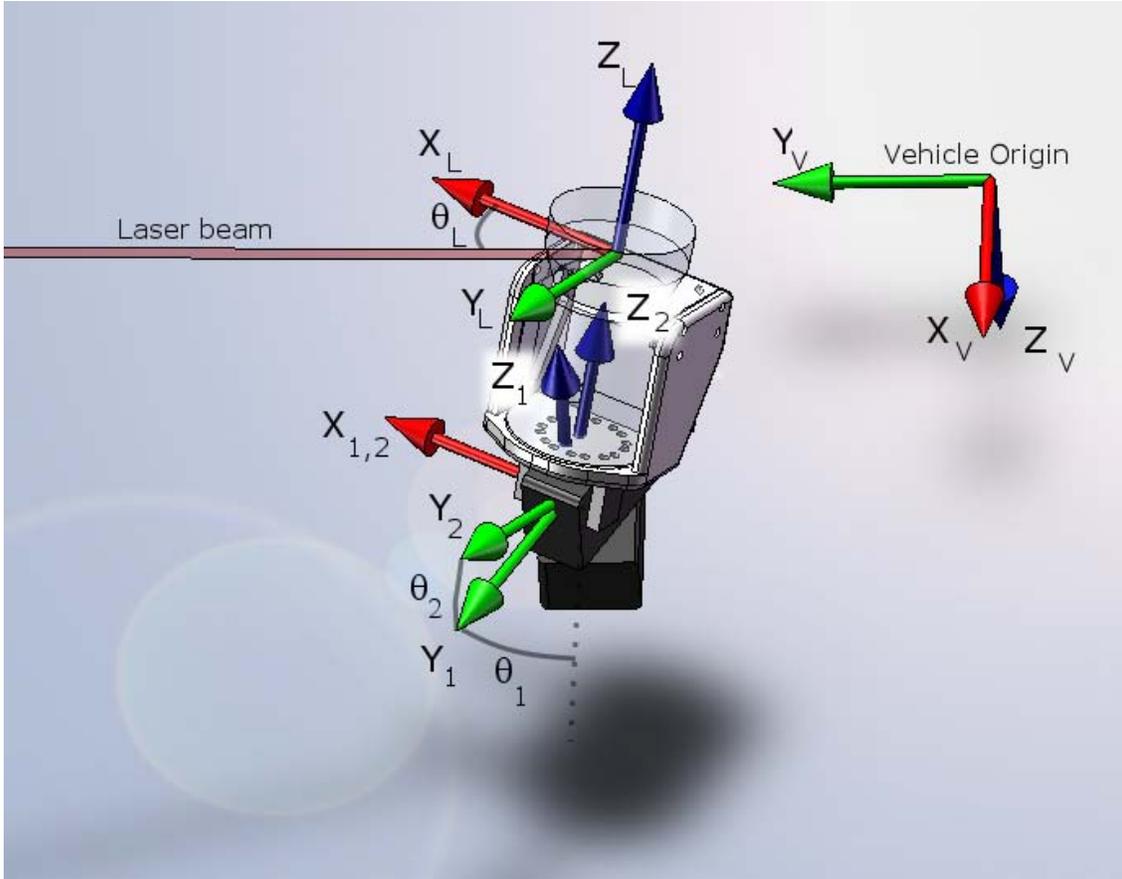


Figure 2-5. Coordinate systems attached to the two moving pan-tilt bodies and the Ladar.

The coordinate transformation between the Ladar sensor and the vehicle is given in Equation 2-9.

$$T_L^V = \begin{bmatrix} 0 & 1 & 0 & \\ 1 & 0 & 0 & \\ 0 & 0 & -1 & \\ 0 & 0 & 0 & 1 \end{bmatrix} P_{1,origin}^V \begin{bmatrix} \cos(\theta_1) & -\sin(\theta_1) & 0 & 0 \\ \sin(\theta_1) & \cos(\theta_1) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta_2) & -\sin(\theta_2) & 0 \\ 0 & \sin(\theta_2) & \cos(\theta_2) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & \\ 0 & 1 & 0 & \\ 0 & 0 & 1 & \\ 0 & 0 & 0 & 1 \end{bmatrix} P_{L,origin}^2$$

(2-9)

The complete transformation from the Ladar's coordinate system to the global coordinate system, given in Equation 2-10, is then a function of the variables roll, pitch, yaw, vehicle easting, northing, altitude, and the pan/tilt angles θ_1 and θ_2 as well as the fixed parameters which consist of the location of the Ladar in the tilting module's coordinate system and the location of the pan/tilt base in the vehicle's coordinate system.

$$T_L^G = T_V^G T_L^V \quad (2-10)$$

3D Bresenham Line Function

Since the voxel structure allows for values which can indicate both occupancy or free space (i.e. zero occupancy), the entire Ladar beam can be used to add information about the environment to the voxel data. At each angular step as the Ladar beam rotates, it measures and reports the distance to the nearest object that is able to reflect the beam. This implies that the voxel in which the beam's reflection point lies is likely occupied. By the nature of the sensor, this also implies that the space along the beam's length is free of obstacles. This implicit beam information can be used to update the voxels which lie along the beam to have an occupancy value that indicates a known free space condition.

In order to perform this free space update, the set of all voxels through which the line segment passes must be determined. This is essentially Bresenham's line function extended to three dimensions with the additional freedom that the end points can be rational coordinates instead of integer coordinates within the grid. The application of Bresenham's line function is illustrated in Figure 2-6 below where the blue dotted line is the input and the darkened cells are the cells through which the line segment passes.

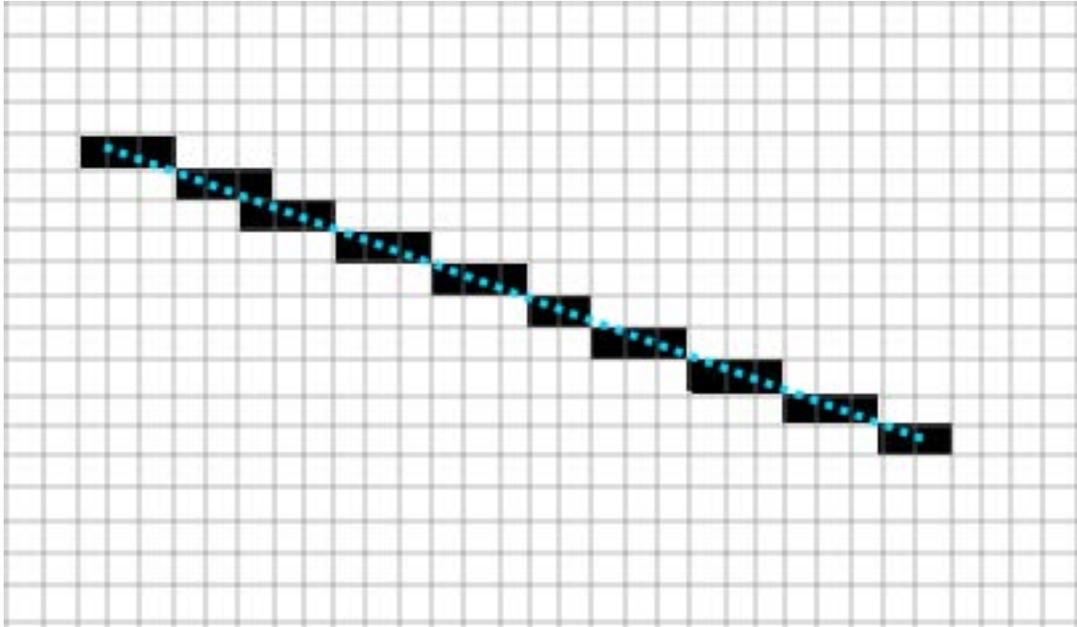


Figure 2-6. Bresenham's line function is used to determine all the cells through which a line segment passes in two dimensions.

The three dimensional extension of this function is required for tracing the Ladar ray through each voxel through which it passes. Also, as stated above, the end points of the beam will not usually be located in the center of a voxel, so the algorithm needs to be general enough to account for this. The voxel grid is essentially formed by layers of mutually perpendicular planes. Realizing that a beam passing through a voxel will always enter through a single face and exit through a single face, the process becomes a matter of simply calculating all the points through which the beam intersects these dividing planes. Figure 2-7 illustrates how the beam's intersection with each of these planes marks the entrance into a voxel along this line segment. By taking all of the voxels found at the intersection points of each of the x, y, and z, planes, we then have the complete set of all voxels through which the beam passes so that the occupancy values of those voxels can be decreased.

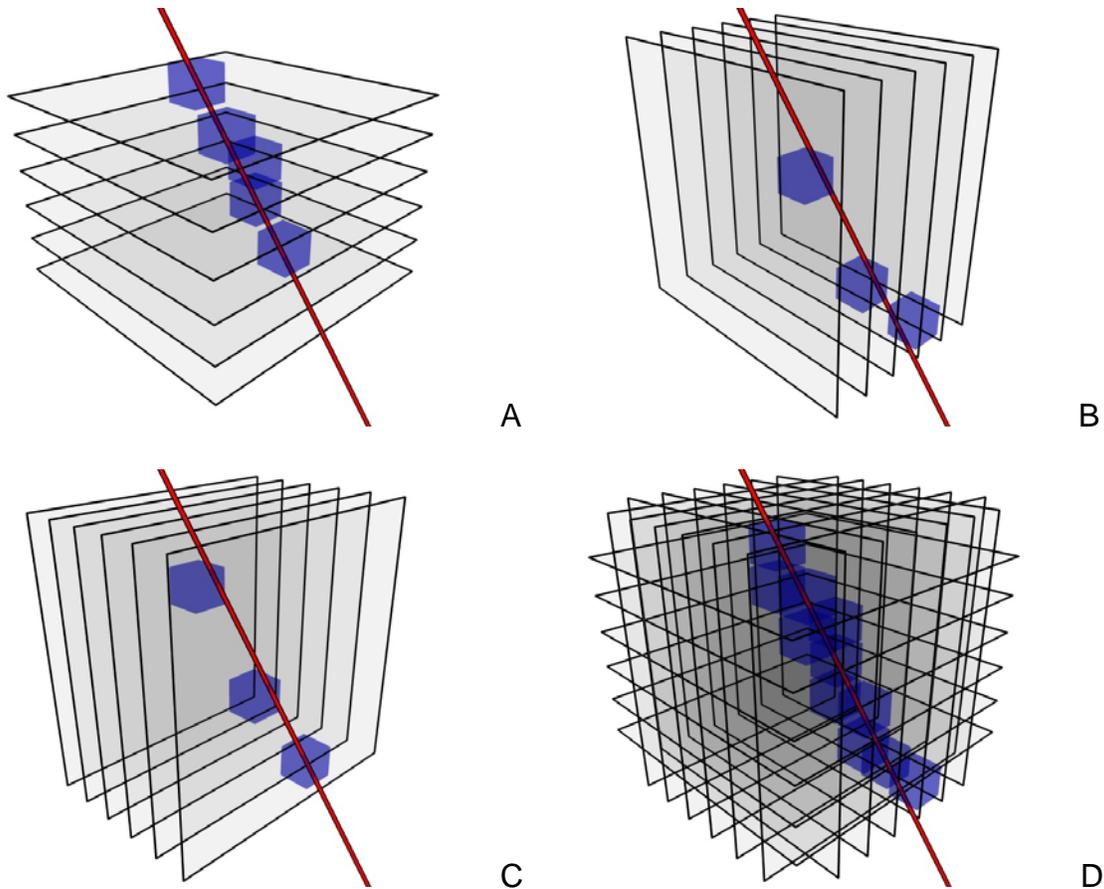


Figure 2-7. Ray tracing through voxels performed by finding the intersection points with planes. A) intersections at each of the z-planes, B) each of the y-planes, C) and each of the x-planes. D) The sum of these voxels form the complete set through which the line segment passes.

Voxel Occupancy Scheme

The voxel occupancy scheme defines how voxel occupancy is determined based on the sensor data and how this information is stored. Several approaches were tried. The first approach involves simply incrementing the occupancy value of the voxel when a Ladar beam terminates in that voxel, and decrementing the occupancy value when a beam passes through that voxel, with a minimum of zero and some maximum value set. This is a simple scheme, but does not take into account a confidence value and the meaning of the occupancy value does not have any probabilistic meaning.

The second approach was taken from [3] and provides a truer probability. Two counters are used for each voxel to keep track of how many beams pass through the voxel, and how many beams terminated in the voxel. The ratio of terminal hits to the total number of terminal hits and pass throughs is used as the occupancy probability. This scheme provides a more meaningful occupancy value, but involves more computational overhead, and the amount of memory required is doubled.

The final approach which was tested, and eventually implemented as the final solution, consists of a counter for the number of times the voxel was modified, similar to a confidence value, plus a counter used in the same manner as described in the first method (beams passing through decrement the value, while beams terminating within increment the value). This provided the most usable results because the confidence value could easily be decayed over time, and the data could more easily be updated to reflect changes in the environment or drift in the vehicle's global positioning and orientation system. The reason that changes are more readily reflected is that when using the second scheme, a purely probabilistic approach, if a voxel that once contained an object had the object removed, then the occupancy probability would only asymptotically decay to zero with more measurements. However, if the incremental counter is used, the same voxel's occupancy value would linearly go to zero and more quickly reflect the change in the environment. And whether the change is real, due to object motion, or just an artifact of a drift in positioning error, the responsiveness of the update is still important. A confidence value which is dependent on the reported range was considered since a shorter range measurement should have a higher confidence value than one made at a long distance, due to the exaggeration of angular uncertainty

in the Ladar's orientation. However, the nature of the sensor and the voxel storage model tends to create this effect naturally. The range measurements recorded at close distances tend to create multiple strikes within a single voxel, increasing the confidence multiple times, whereas measurements coming from objects farther away tend to only strike a single voxel, incrementing the confidence only once.

Data Fusion

Data fusion is essential when data is being collected from multiple sensors or even a single sensor if the data is kept persistent. If data is kept persistent over time, a single sensor can take measurements in the same vicinity multiple times and thus requires the merging of the new data with the previously recorded data. The use of voxels makes this process very simple compared to the same task for vector data. Since voxels are fixed in the global reference frame, data can be added in a point wise fashion independent of sensor location. Also, since the voxel data is allocated before data is collected, the size of the data structure is fixed and does not grow with added data, new data simply updates the voxel values and adds more information. This is especially useful when merging data from different sensor types such as camera data and Ladar data as was done in [3].

Finding Ground Surface

Identifying the ground surface is the first step of the classification process. Without knowing which voxels make up the ground surface, evaluating traversability, identifying trees, or calculating brush height would be nearly impossible. The algorithm used for identifying these ground surface voxels is adapted from [9]. Since there are no measured reflection points below a ground surface voxel, we can assume that there will be no occupied voxels below a ground surface voxel. Also, since no beams can

penetrate below the ground surface, there should be no voxels with a known free space value below a ground surface voxel. The algorithm works by evaluating each candidate voxel one at a time. A pyramidal region of a certain depth and slope is defined below the candidate voxel. While searching through the pyramid, if an occupied voxel or known free space voxel is found within it, the voxel fails the ground surface test; otherwise, the candidate voxel is marked as being part of the ground surface. Figure 2-8 illustrates the pyramidal region drawn under a candidate ground voxel. This voxel passes the test and is marked as part of the ground surface. Figure 2-9 illustrates the pyramidal region drawn under a candidate voxel which does not pass the test as it contains other occupied voxels.

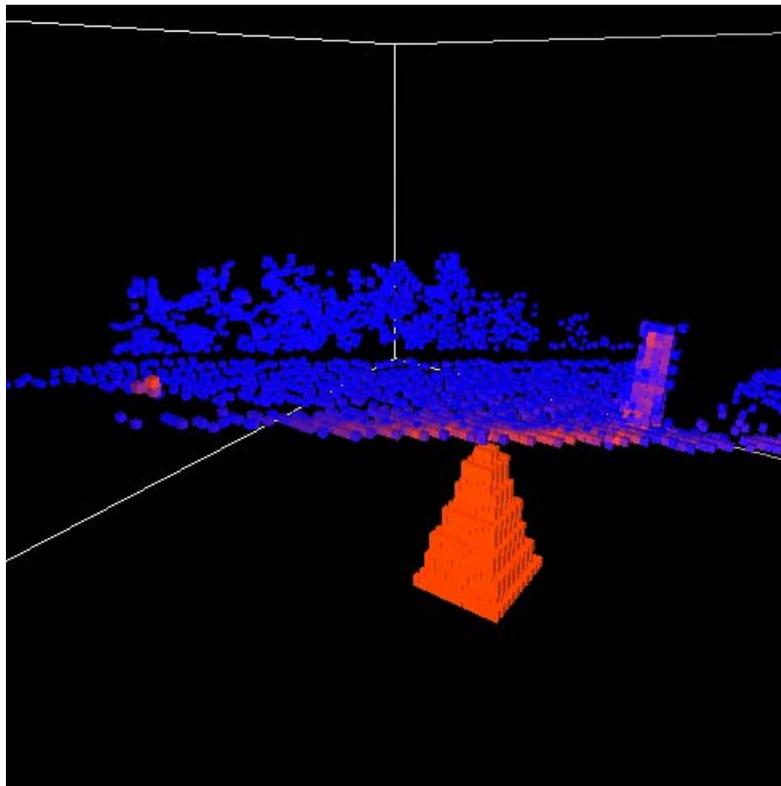


Figure 2-8. The pyramid (drawn in orange) under a candidate ground surface voxel.

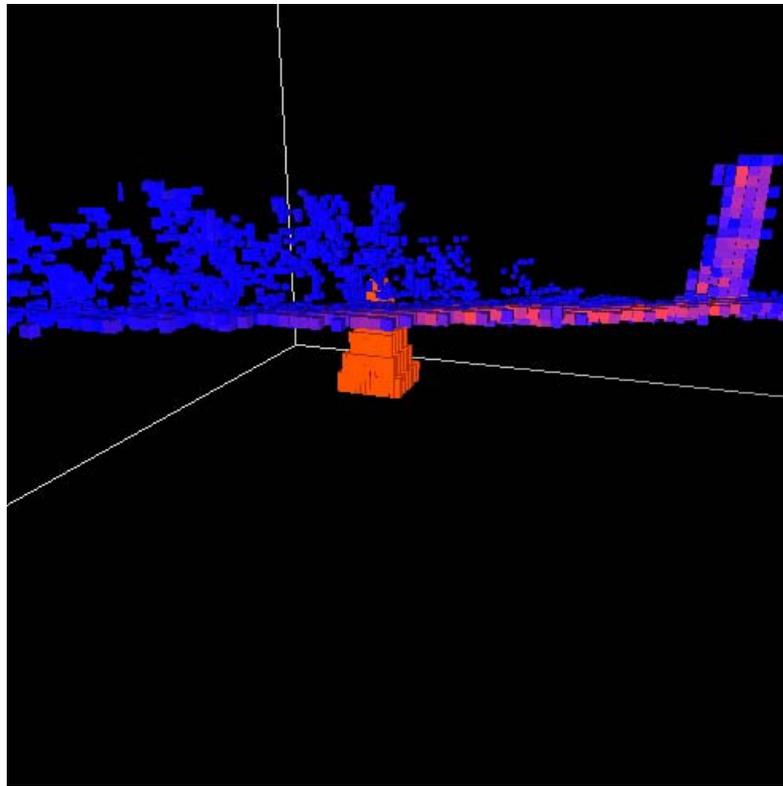


Figure 2-9. The pyramid (drawn in orange) under a candidate ground voxel which does not pass the ground surface test.

The results of applying the aforementioned technique to an entire scene can be seen in the difference between the original image shown in figure 2-10 and the image with only ground voxels shown in Figure 2-11. The resulting set of voxels is usually sparse due to the size of the voxels and the spacing of the beams at longer ranges. This poses a problem when a ground height estimate is needed in a column with no ground voxel. Several approaches were tested to solve the problem.

Breaking the working area into a coarse grid (where grid cell sizes from 1 m² to 25 m² were tested) and then calculating the average ground height proved fairly reliable, especially when filtering out regions with a low data count or high standard deviation. However, the resulting grid, does not provide an accurate ground height estimate if the terrain is sloped instead creating a step effect. Additionally, there is a loss of information

about the ground contour that was contained in the original data. The average ground height process just described is shown in Figure 2-12.

Another approach is to fit an analytical function to the data using a regression technique. The problem then arises of determining the function to best describe the surface without using such a high order function that computations become too expensive or the regression process yields an over fit and erratic solution. But a function whose order is too low will result in the loss of information of certain ground features. Another approach would be to fit a surface spline to the data.

Since the data is already in a raster format, a raster approach to interpolation was implemented. This two dimensional raster interpolation process is performed by sequentially applying two one dimensional interpolations. First, linear interpolation is performed in the x-direction between ground voxels (provided they are within some maximum range of each other). Then, linear interpolation is performed in the y-direction in the same way. The results are stored in a temporary array of ground heights. The results of these steps are shown in Figure 2-13. Then to mitigate the effects of performing the interpolation in this order, the same process is performed but in the opposite order, first the y-direction and then the x-direction. The resulting ground heights from using the y-x order are averaged with the previous ground height array based on the x-y order. Finally, extrapolation is performed. The results of this process are shown in Figure 2-14 and Figure 2-15.

An additional source of ground surface information comes from the vehicle's wheels. Since we assume that the vehicle is resting on the ground, the vehicle roll,

pitch, yaw, and elevation can be used to estimate the geometry of the ground surface immediately around the vehicle.

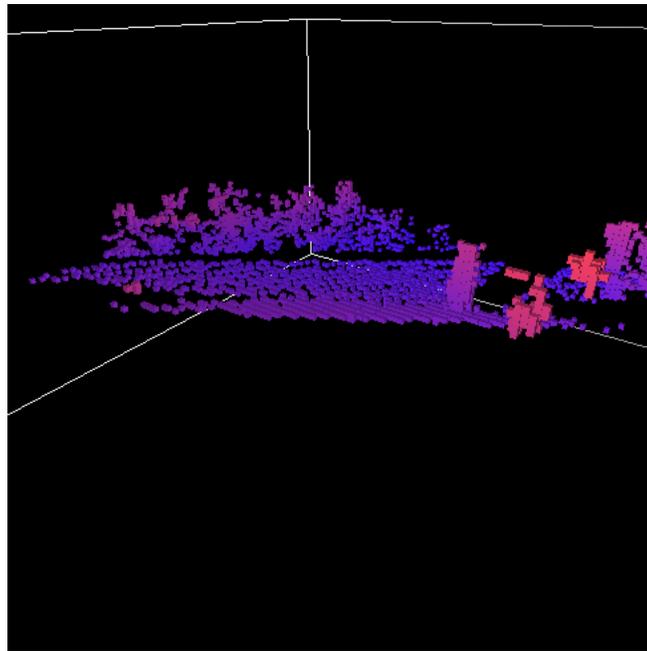


Figure 2-10. The original scene before applying the ground surface filter.

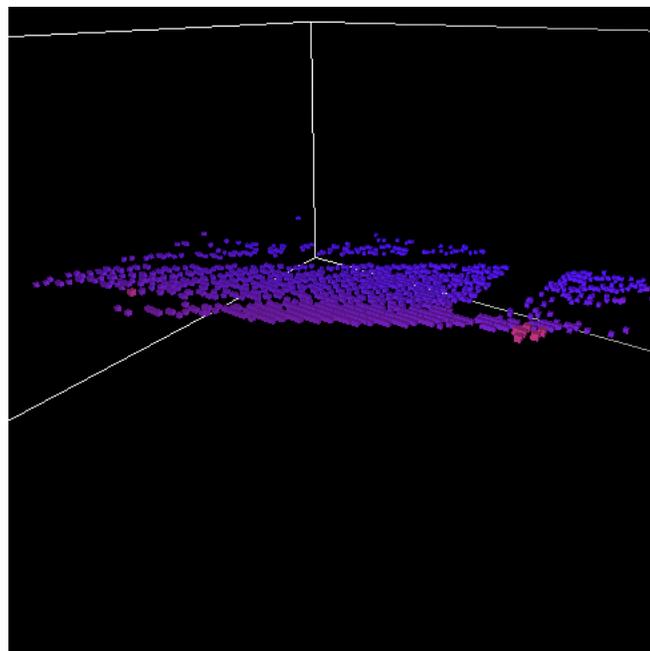


Figure 2-11. The scene after applying the ground surface filter

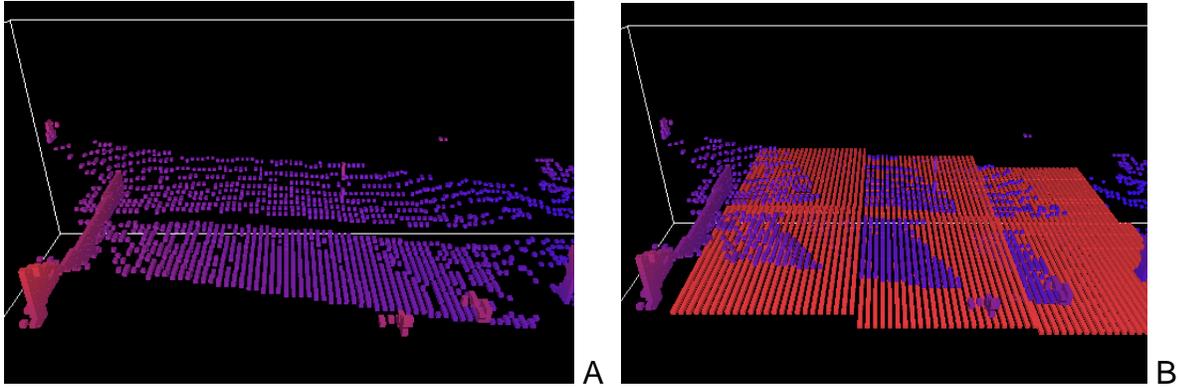


Figure 2-12. A ground representation technique using tiles of average ground heights. A) The original scene and B) the ground surface fitted with six meter wide tiles.

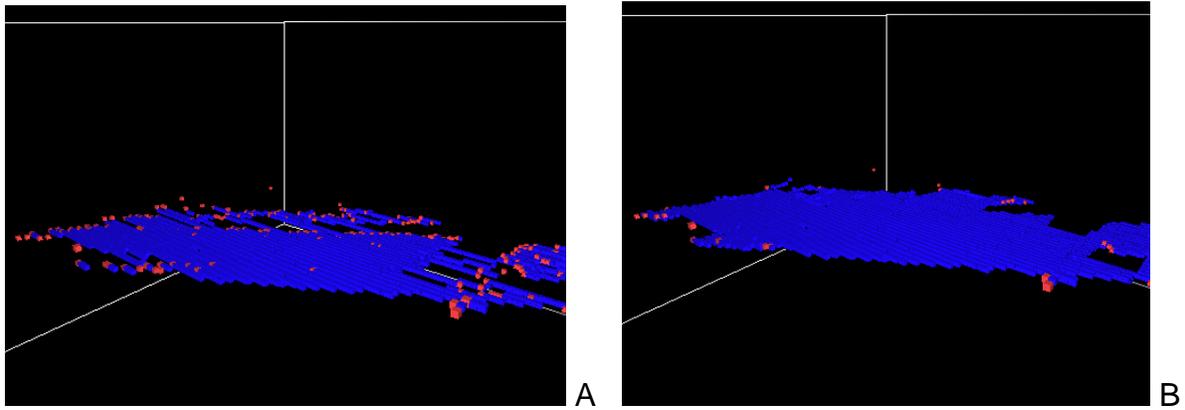


Figure 2-13. Two dimensional interpolation performed using a sequence of one dimensional interpolations. The set of ground surface voxels are A) interpolated in the x-direction and then B) interpolated in the y-direction. (Red voxels are simply the end points.)

This is useful information when no Ladar data is available close to the vehicle. To do this, the ground in close proximity to the vehicle is represented as a plane and Plucker coordinates are used to describe it's position and orientation using Equation 2-11, where \bar{r} is the position vector of all points in the plane and \bar{S} and D_0 are constant parameters to be found.

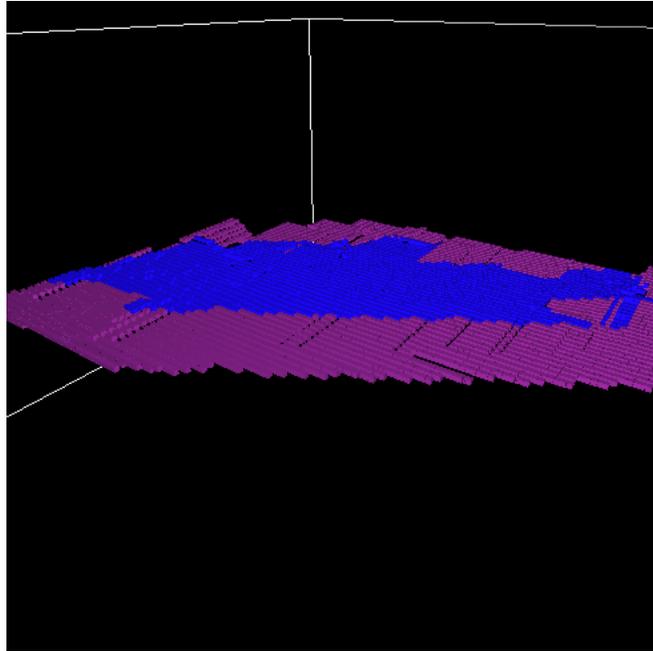


Figure 2-14. The result of performing interpolation (blue) and extrapolation (purple) on a set of ground surface voxels.

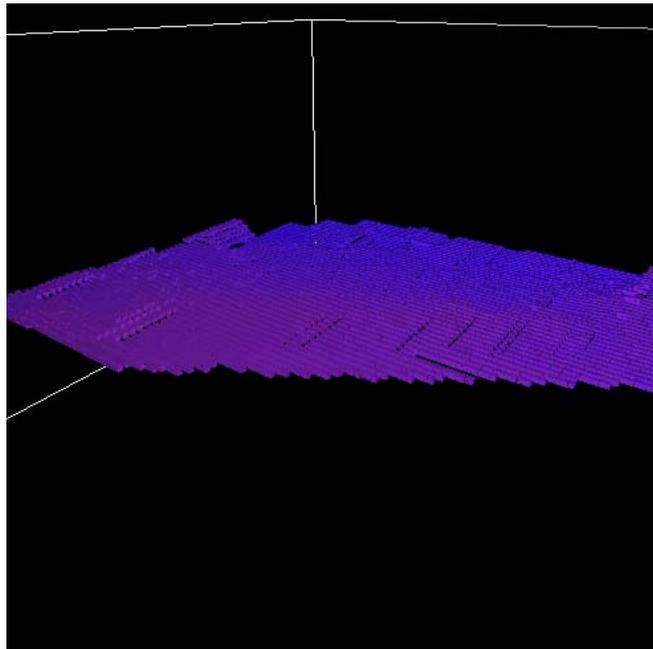


Figure 2-15. The final ground surface representation.

The normal vector \vec{s} is simply the negative z-direction of the vehicle which can be found using Equation 2-12 (recall that the coordinate system of the vehicle defines z as

downward) and represented in the global coordinate system using Equation 2-13. The scalar D_0 is then found using Equation 2-14 where r_1 is the point directly below the vehicle origin.

$$\vec{r} \cdot \vec{S} + D_0 = 0 \quad (2-11)$$

$$\vec{S} = -\vec{z}_v \quad (2-12)$$

$$\{z_v\}_G = T_V^G \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \quad (2-13)$$

$$D_0 = -\vec{r}_1 \cdot \vec{S} \quad (2-14)$$

The estimated elevation of the ground surface at any point around the vehicle can now be found with Equation 2-15. Note that this becomes undefined when the z component of the \vec{S} vector is zero, but this will never occur unless the vehicle is pitched or rolled a complete ninety degrees.

$$z(x, y) = \frac{-(x * S_x + y * S_y + D_0)}{S_z} \quad (2-15)$$

Estimating the Height of Above Ground Objects

After performing the process to extract the ground surface from the voxel data, the height of objects on the ground surface can accurately be estimated. If the height of the ground surface estimate is subtracted from the height of the highest occupied voxel, we have a simple estimate of the height of an object relative to the ground. An additional computation which can be performed is calculating the density of the space above the ground in each voxel column. This is done by starting at the ground level for that particular column and calculating how many voxels in that column are occupied versus free. When performing this calculation, the search is only performed up to a certain

predefined height above the ground so that an object that was that tall would tend to create one hundred percent density value while a column above an open area would tend to yield zero percent density.

Finding Trees

For obstacle avoidance or other tasks such as the autonomous felling of trees, it is important to be able to correctly identify trees in a scene. After the ground surface has been identified, the classification of trees can be performed more easily. The features that identify a tree are 1) its connection to the ground 2) a somewhat cylindrical and upright trunk and 3) the canopy. Starting with the estimated ground surface, the density of each of the columns above the ground is computed with the modification that an added penalty for free space is imposed. This means that occupied space increases the density value, while known free space decreases the density value, hence an overhanging branch would be penalized for not being connected to the ground, while a trunk connected to the ground would not. A two dimensional grid of above-ground density values is thus computed. An example of this result is shown with the original scene in Figure 2-16 and the above-ground density values shown in Figure 2-17.

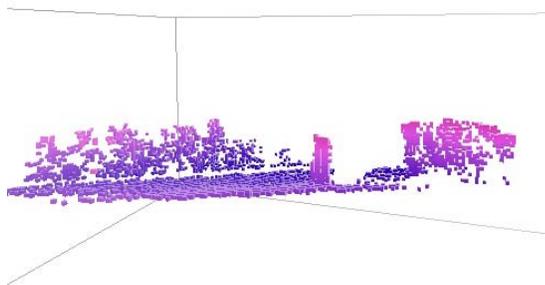


Figure 2-16. A sample scene with brush and a tree trunk.

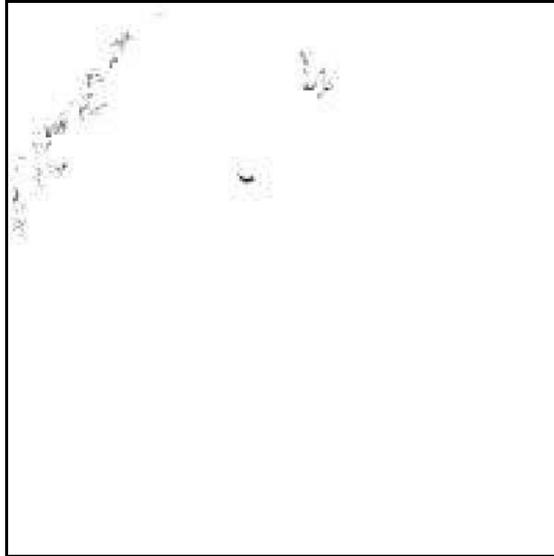
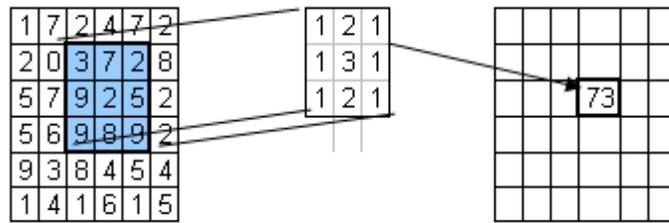


Figure 2-17. The above ground density plot of a scene with brush and a tree trunk.

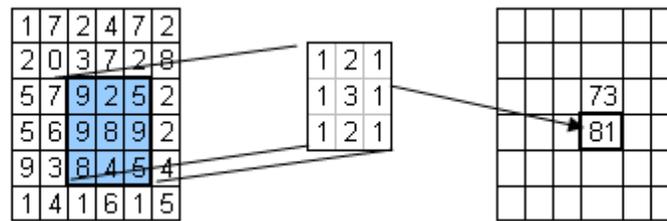
In order to isolate trees from the other brush which might exist, a filter must be applied to this data. A common two dimensional image processing algorithm known as a convolution was selected to isolate point like or circular areas of high density (i.e. trunks). In a continuous domain, a convolution is performed by integrating the product of two functions with a variable offset from one another. The definition of a convolution is described mathematically by Equation 2-16.

$$f(x) * g(x) = \int_{-\infty}^{\infty} f(\tau)g(t - \tau)d\tau \quad (2-16)$$

In the context of image processing, the convolution is performed by selecting a convolution kernel and summing the products of each of the kernel's pixel values with the overlapping image's pixel values. The convolution summation is performed only over the range of the kernel size which is usually much smaller than the image size. This process is demonstrated in Figure 2-18.



(a)



(b)

Figure 2-18. A convolution performed with a 3 by 3 kernel. The left array is the image data, the 3 by 3 array in the center is an example convolution kernel, and the array on the right shows the resultant value of the convolution at this one position.

The overall effect of this filter is that when the kernel is shifted over a portion of the image that closely matches the pixel values, the summation will be high, whereas portions of the image which do not closely match the kernel result in lower values. The convolution kernel chosen for finding trees is an 11 by 11 array with large values in the center, neutral values around the center, and negative values further out. The fall off, or steepness of the gradient is left as a tunable parameter. A sample kernel used for tree detection is shown in Figure 2-19.

When using this as the kernel, the convolution will create higher summations when the kernel is over a tree trunk and lower summations when the kernel is over an area that does not exhibit this point like density distribution.

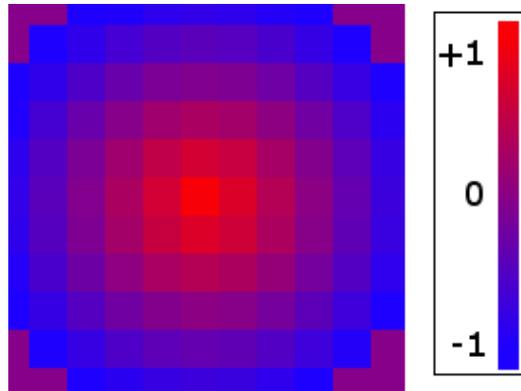


Figure 2-19. Convolution kernel used in tree trunk detection.

The results of applying the convolution to the above ground density data are shown in Figure 2-20 where it is apparent that the tree trunk has created higher values (indicated as darker pixels) because it closely matched the kernel whereas the brush in the scene did create some higher values, but not as high as the full tree trunk.

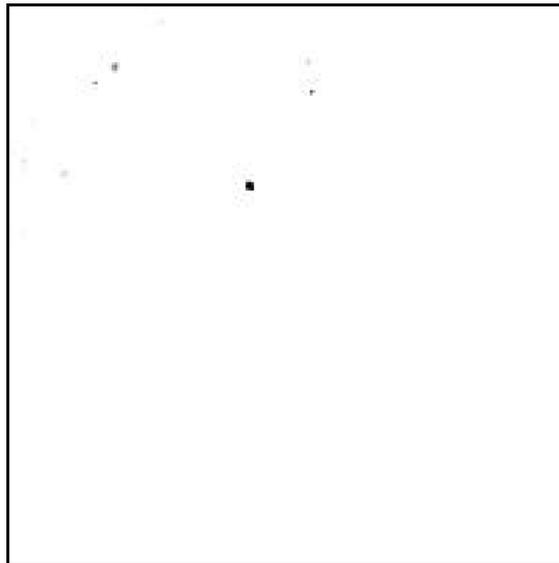


Figure 2-20. Image of above ground densities after the convolution filter has been applied.

The tree can be isolated by applying a threshold filter to the image which sets any pixels less than a predetermined value to zero, while anything equal to or above the threshold will be kept. The threshold value is a hand-tuned parameter which can vary

based on the size of trees, and the amount of brush in an environment. The results of applying the threshold can be seen in Figure 2-21 where only the pixels corresponding to the voxel columns in which the tree lies are marked.

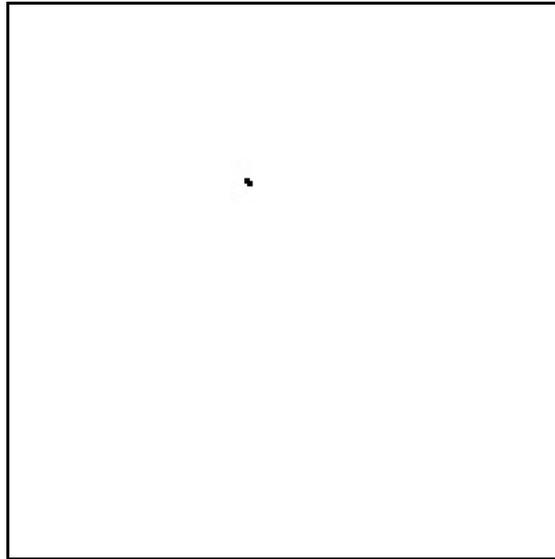


Figure 2-21. The image of above ground densities after the convolution filter and the threshold filter have been applied.

While finding the location of tree trunks in an outdoor environment may seem easy to accomplish with just two dimensional horizontal Ladar scans, there are two reasons why this approach would fail. The first is the presence of brush and other objects like low hanging moss that can appear to be trunk like objects, and the second is the presence of hilly terrain which could intersect with the fixed horizontal scan. The key advantage to using the technique described in this chapter is that using three dimensional data and extracting the ground surface, then finding the above ground densities of each column and performing the convolution correctly isolates trees even amidst brush and hilly surfaces. The process effectively has a normalizing effect on a hilly or sloped ground surface, whereby filters and other statistics can be applied to the space above the ground surface.

The final step in determining the tree locations is clustering. After the technique described above has been performed, the result is a set of pixels in an image corresponding to the voxel columns which are deemed to contain a portion of a tree trunk. In order to create a vector list of tree locations, these regions of pixels must be grouped into distinct trees and the centroids of each of these distinct tree groups must be found. This is done by a simple clustering algorithm which begins by seeding the current set with one of the tree pixels from the image. Then, every other pixel which touches any of the pixels in the current set are removed from the global set and moved to the current set. This process is repeated until no more cells from the global set are added. At this point, the current set contains all the pixels belonging to a single tree and the centroid and radius can be calculated. The process is repeated by clearing and reseeding the current pixel set until there are no more cells in the global cell set.

```

global_tree_cells
current_tree_cells := []
tree_centroids := []

current_tree_cells.add(global_tree_cells[0]);
global_tree_cells.removeAt(0);

added_new_cells := true;
while (global_tree_cells.size() > 0)
  while (added_new_cells == true)
    added_new_cells := false
    for j from 0 to current_tree_cells.size()
      for k from 0 to global_tree_cells.size()
        if (cellsTouch(current_tree_cells[j], global_tree_cells[k]))
          current_tree_cells.add(global_tree_cells[k])
          global_tree_cells.removeAt(k)
          added_new_cells := true
        k--
      end_while
    end_while
    tree_centroids.add( find_centroid(current_tree_cells) )
    clear(current_tree_cells)
  end_while
end_while

```

Figure 2-22. Psuedocode of the clustering process.

Saving Objects in the Local Knowledge Store

In order to facilitate the storing of the reduced information about the environment, namely tree locations, a local knowledge store was developed. The purpose of this piece of code is to keep a copy of the known objects so that the list can be queried and sent to other components that will need this information such as the Local Reactive Planner and the Knowledge Store (the system level knowledge database). The other purpose is to handle the merging of objects already in the knowledge store with a new set of sensed objects. Since the classification function is performed multiple times a second, trees are continually detected and merged with the knowledge store's current data. This acts as another type of filtering in the system. When an object is first added, its initial confidence value is low and its existence will not be reported to other components. As the object continues to be detected by the classification algorithm and is merged into the knowledge store, the confidence increases until it reaches a threshold which allows it to be reported. As an additional filter, the confidence of the objects decays exponentially with a fairly long time constant, so if the object fails to be detected for a while, it is removed.

The merging process is simple. New objects are initialized to a low confidence level. From then on, every time an object is added, the closest object already in the knowledge store is found. If the distance between the two is below a threshold (usually about 0.3 m) then the objects are merged into one object using their respective confidence values as weightings for determining the centroid. The new object is assigned a confidence value equal to the sum of confidences of the two objects.

3D Imaging techniques

There are several two dimensional image processing techniques which can be extended to three dimensions and applied to voxel data. These techniques were experimented with, but not used in the final implementation. They are listed below for informational purposes.

Blur

One such filter is a blur. In a two dimensional image, this has the effect of smoothing rough edges and taking out high frequency components. The effect is the same for voxel data. The blur is accomplished by setting each voxel value in the block to be a weighted average of its neighboring voxel values. The weights assigned to the neighbors and the size of the neighborhood dictate the amount of blurriness of the resulting data. Originally, this was thought to be an option if sensor noise in the Ladar or GPOS system caused too much noise, but the accuracy of the sensors obviated the need for such a filter. Below are two images before and after a blurring effect.

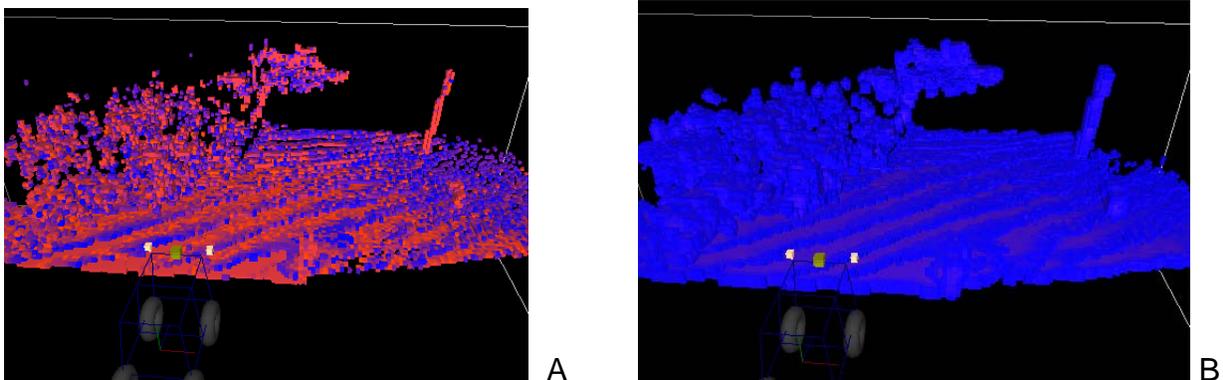


Figure 2-23. A three dimensional blurring effect applied to voxel data. A) Original data. B) The data after blurring which is smoother and has more uniform occupancy.

Differencing

Another very useful technique which can be applied is differencing. If a scene contains both static objects and moving objects, the moving objects can be isolated by taking two snapshots at different times and performing a pixel-wise subtraction from each other. The result is that the difference of all the pixels corresponding the parts of the image that remained unchanged are close to zero, while the absolute difference of parts of the image that did move will usually be much greater than zero. The same is true with three dimensional voxel data. If a snapshot of the data is taken before an object moves and after, the voxel-wise difference of the two scenes will create high values where changes have taken place and almost zero values elsewhere.

Hough Transform

Another useful technique is the Hough transformation. The Hough transformation is used frequently in images to extract lines, but can be used for extracting other shapes as well such as circles or ellipses. For the case of three dimensional data, we can apply a version of the Hough transform to extract planes. The basic premise is that any point has an infinite number of planes passing through it. If the orientation of all possible planes is discretized and limited so that there are a fixed number planes which contain the point, then we can calculate the plane parameters for each of these possibilities. When all points in a set of data (or subset) are analyzed together, the specific parameters which define a plane which passes through a large number of points will be repeated frequently. The parameter sets which are most frequently repeated can therefore considered as the most dominant planes in the data. To perform this transform, planes are described by three variables: the angle between the plane's normal vector and the z-axis (denoted ϕ), the angle between the normal vector's

projection onto the x-y plane and the x-axis (denoted θ), and the shortest distance between the coordinate origin and the plane (denoted ρ). This representation of a plane is pictured in Figure 2-24. To determine the set of parameters describing all the planes passing through a single point, the value of θ was varied over the range of $[0, 2\pi)$ with 40 evenly spaced bins and the value of ϕ was varied over the range of $[0, \pi/2)$ with 20 bins. For each of these angle pairs, the value of ρ is computed and the value of the bin corresponding to those θ , ϕ , and ρ values is incremented. The value of ρ is found using Equation 2-17 where \hat{u} is the unit vector defined by the angles θ and ϕ and \vec{p} is the vector from the origin to the current point or voxel.

$$\rho = \hat{u} \cdot \vec{p} \quad (2-17)$$

Figure 2-24 shows multiple planes going through a single point. The grey plane contains one point, so the bin associated with these coordinates would have a value of one; but the blue plane contains all three points, hence the bin associated with these three coordinates would have a value of three. Once this is done for every point or voxel, the bins are sorted and the ones with the highest values indicate the parameter sets (θ , ϕ , and ρ) of the most dominant planes.

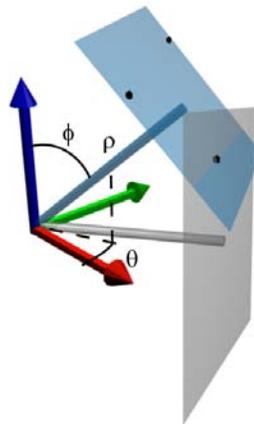


Figure 2-24. Hough plane extraction.

This was originally used to extract the ground surface from voxel data, but because the ground is not well characterized by a plane, the more robust method described in the ground surface section earlier was used.

Smart Sensor Grid

The information about the environment gathered by the sensors and interpreted using the classifier is necessary for the Local Reactive Planner for path planning. In order to provide the most up to date and detailed data, a message which contains a georeferenced square grid of values is used. This two dimensional grid can be populated with vegetation height values, traversability values, and tree location values. The vegetation height is reported using the maximum height of occupied voxels above the ground surface (using the assumption of course that in a vegetated environments, only brush and other plants exist). The traversability can be computed using a test of the local smoothness or slope of the ground surface which has already been extracted. The tree locations can be placed directly into the grid cells. This gives the planner a detailed picture of the surroundings without having to provide it with a heavy three dimensional data set.

Software Architecture

The entire process of collecting data and processing data is based around the block manager. This class is designed to create, save, and keep track of all the blocks of voxels placed around the vehicle and is responsible for passing down the Ladar's beam information to modify the individual blocks. At set time intervals, a snapshot is taken of the current state of all the voxels in the blocks surrounding the vehicle for processing. This allows data to continue to be added to the block manager's blocks while the processing algorithm is working on the copy. This instantaneous snapshot is

also sent to the visualizer server which sends out the data over a TCP connection to a client which displays the voxel data using OpenGL. The process is diagramed in Figure 2-25 below.

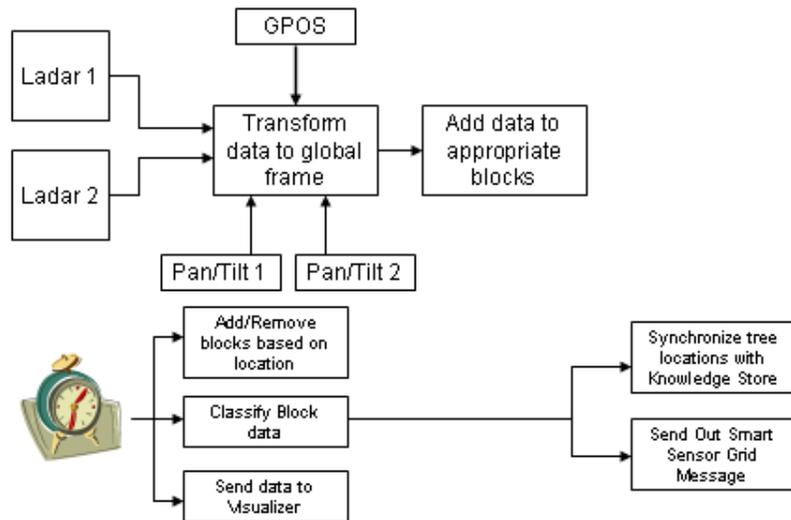


Figure 2-25. Process diagram. The clock symbolizes a timer which triggers the functions it points to.

CHAPTER 3 RESULTS

This chapter presents the results of the entire perception system including data collection, scene classification, and several other processes. The ability to define quantitative metrics to evaluate the performance of the system is difficult because of the unstructured nature of the outdoor environment used for testing. Generally, the best way to evaluate the classification of a scene is to compare it to what a human can perceive. While this is not a well defined metric, usually it is very clear whether the algorithm performed correctly or failed in certain areas.

Static Data Collection

Static data collection was performed by mounting a SICK LMS-151 Ladar on a pan-tilt mechanism and fixing this on a tripod, shown in Figure 3-1. The Ladar was set to tilt up and down at an average rate of 20 degrees / sec as data was collected and the ray tracing operation was performed through the voxels. A voxel side length of 0.15 m was used with each block having base dimensions of 100 voxels by 100 voxels and a height of 40 voxels (15 m by 15 m by 6 m). Enough blocks were added around the location to maintain a working space of 30 m by 30 m with a height of 12 m. At the end of about ten scans, a snapshot of the block's voxel data was taken and saved. The results are shown in Figure 3-2 through Figure 3-4. The ratio of voxels with non-zero occupancy values and the ratio of voxels with known free space are shown in Figure 3-5. The number of occupied voxels is quite small, however, maintaining the slightly occupied blocks does not come at any additional computational cost, only more memory cost. Using these parameters, the size of each block is about 0.8 MB.



Figure 3-1. A SICK LMS-151 LADAR mounted on a pan-tilt actuator on top of a fixed tripod.

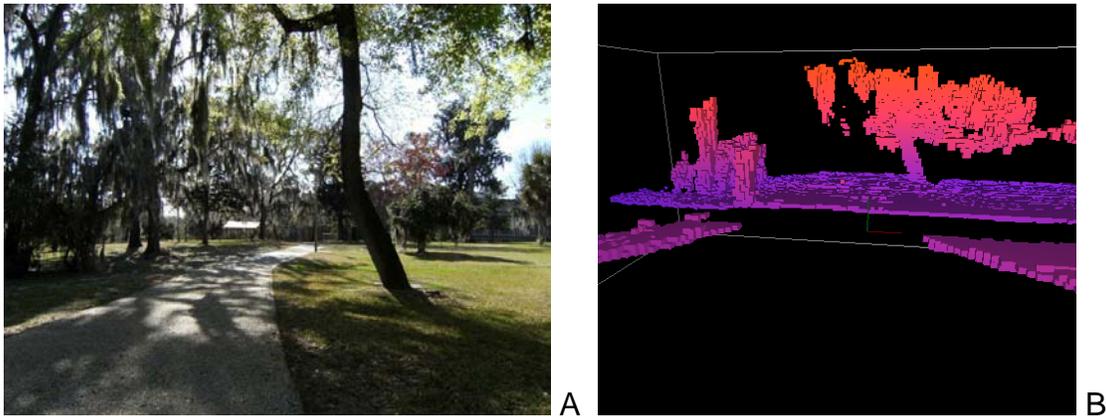


Figure 3-2. Scene 1: Part of the University of Florida campus with a large tree. A) A photograph of the scene and B) the sensed data in the voxel representation colored by height value.

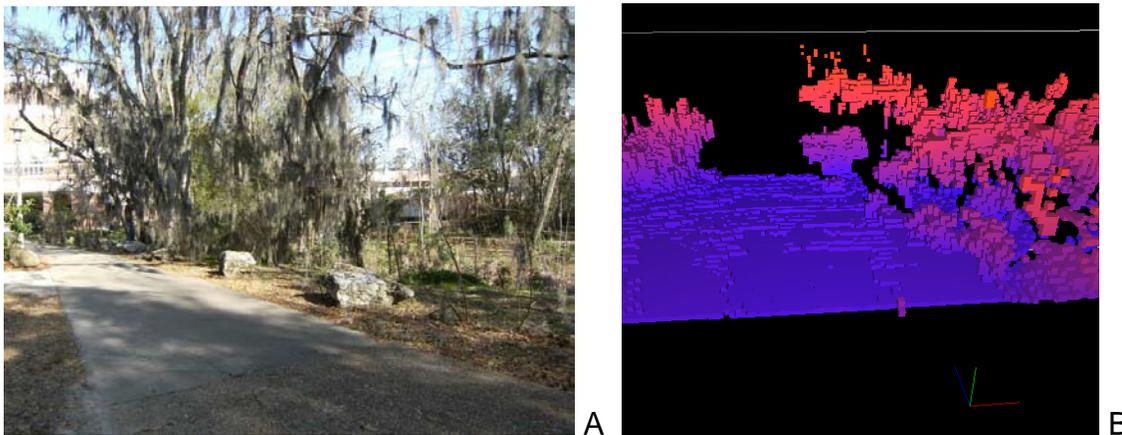


Figure 3-3. Scene 2: A section of the University of Florida campus with rocks and brush. A) A photograph of the scene and B) the sensed data in the voxel representation colored by height value.

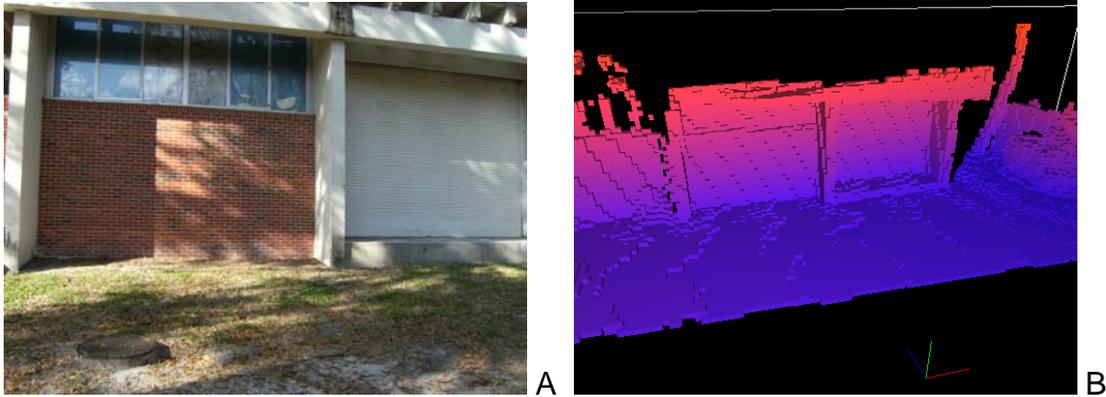


Figure 3-4. Scene 3: The side of a building. A) A photograph of the scene and B) the sensed data in the voxel representation colored by height value.

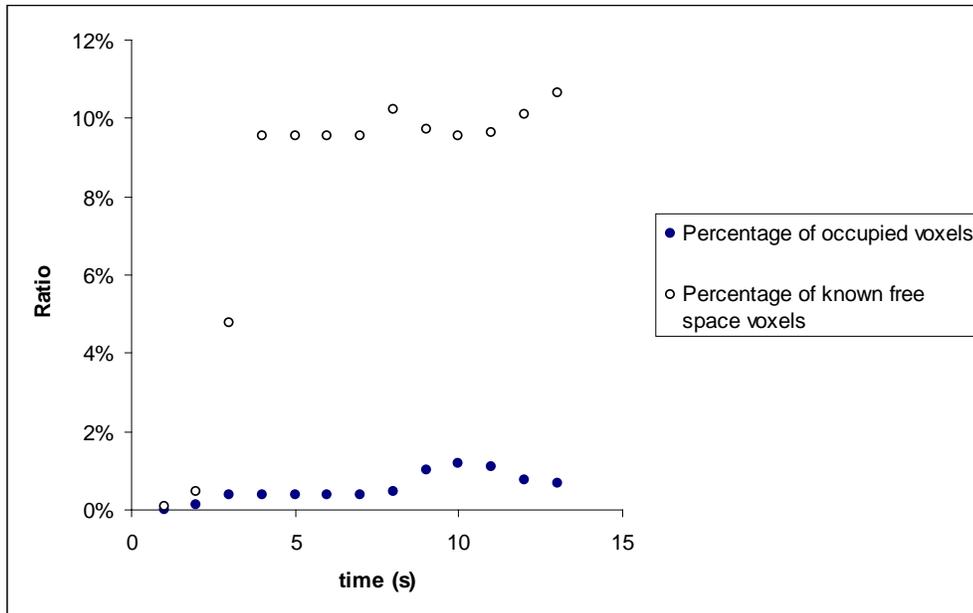


Figure 3-5. The ratio of occupied voxels to total voxels and free space voxels to total voxels as data is collected over time. The data is taken from one of the blocks in scene 1.

Dynamic Data Collection

Data collection on board a moving vehicle presents more problems than static collection, primarily from having to rely on position and orientation data for the vehicle which contain uncertainty, but also due to vehicle vibration. While data collected from a static position need not be decayed quite as fast, data collected aboard a moving

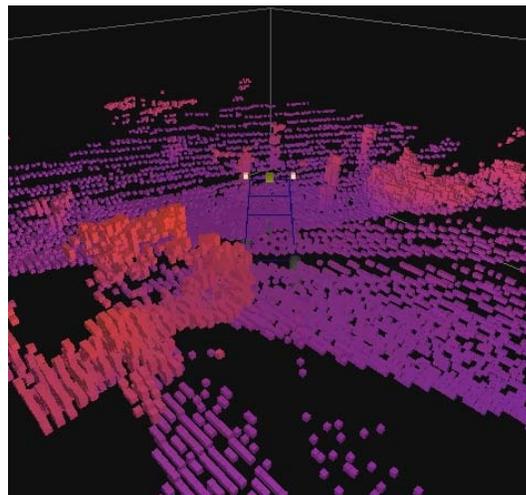
vehicle needs to be decayed more quickly. The decay rate should be on the order of the period of time for which drift in GPOS data can be ignored. Data was collected with a Ladar mounted on a pan-tilt unit and fixed to the frame of an off-road vehicle shown in Figure 3-6. The speed of the vehicle ranged between 2 – 5 mph. Several frames captured from the data collected aboard a moving vehicle are shown in Figure 3-7 and Figure 3-8.



Figure 3-6. The vehicle used for collecting Ladar data.



A



B

Figure 3-7. A scene with trees observed aboard a moving vehicle. A) The still frame from the video and B) a snapshot of the voxel data taken at the same time.

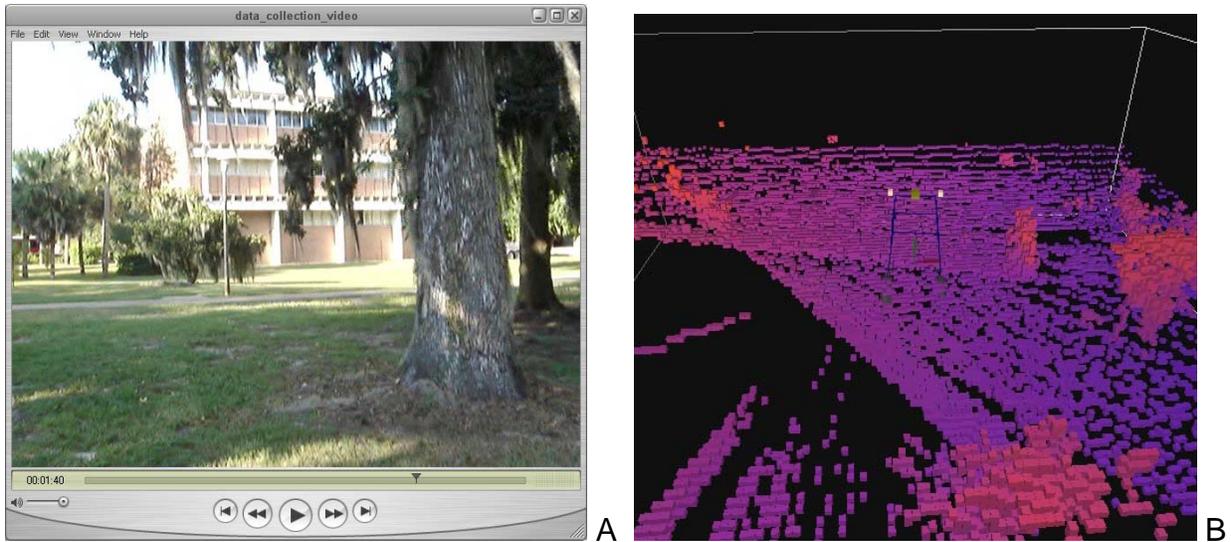


Figure 3-8. A scene with trees and brush (to the right, off camera) observed aboard a moving vehicle. A) The still frame from the video and B) a snapshot of the voxel data taken at the same time.

Ground Plane Identification

The ground plane identification algorithm works quite well. Figure 3-9 shows how the detection algorithm works even amidst heavy brush and is able to correctly find the ground surface and interpolate amidst occlusions from the brush. Figure 3-10 shows how even with sparse data, the algorithm can interpolate and extrapolate effectively.

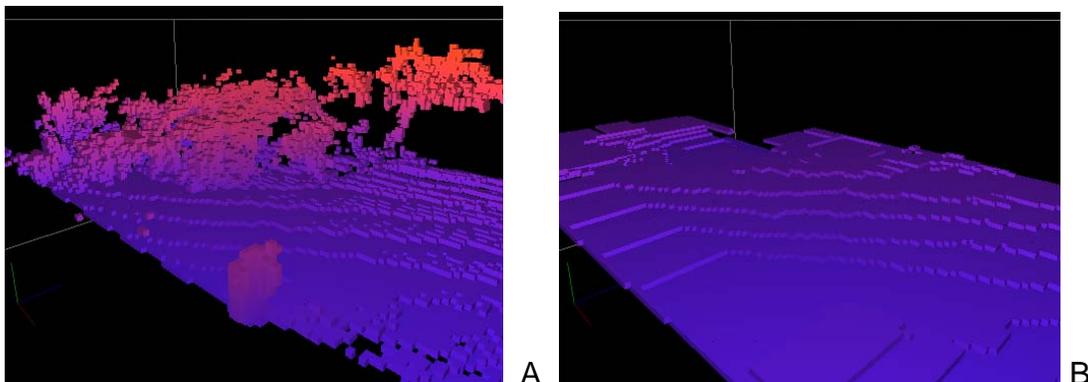


Figure 3-9. Results of ground surface identification among heavy brush.

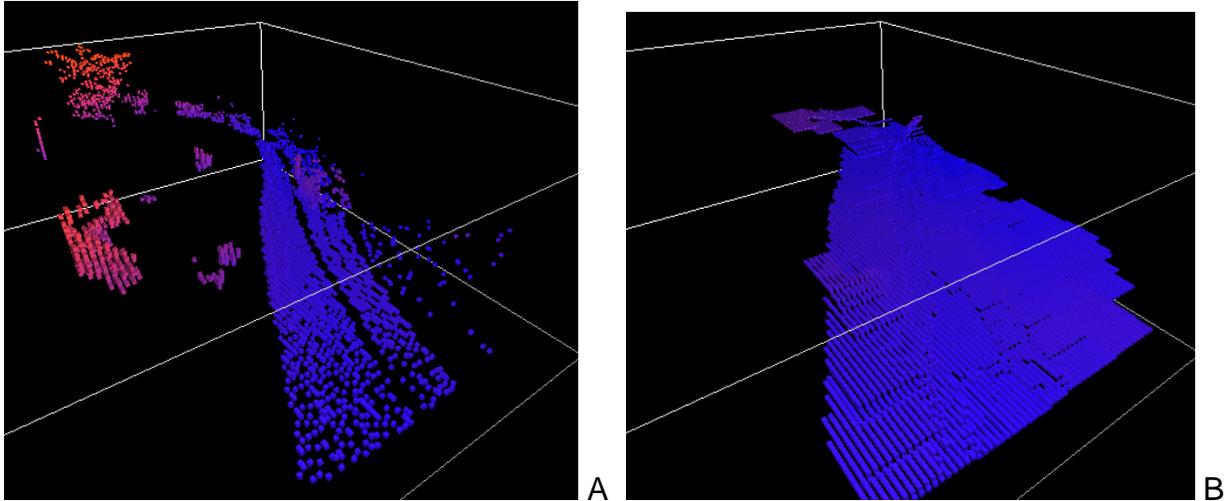


Figure 3-10. Results of ground surface identification with sparse data.

Brush Height Estimation and Tree Detection

The brush height estimation and tree detection algorithm requires some tuning based on the size of trees and the average height of the brush surrounding the trees. Tunable parameters include the size of the tree kernel, the threshold for determining tree locations based on the result of the convolution, and the height above the ground to start computing the above ground densities. The results of the tree detection algorithm is not perfect, and will frequently misclassify brush as tree trunks, but since the Local Knowledge Store operates as an intermediary before outputting the results to the other components, the random misclassifications do not build up a high enough confidence value to be reported since it takes consistent classification to become valid. Figure 3-11 shows the results of the brush height detection and tree detection on data collected with a stationary vehicle. Figure 3-12 shows the results on data collected aboard a moving vehicle. Note that there is a false tree trunk identification indicated in the figure.

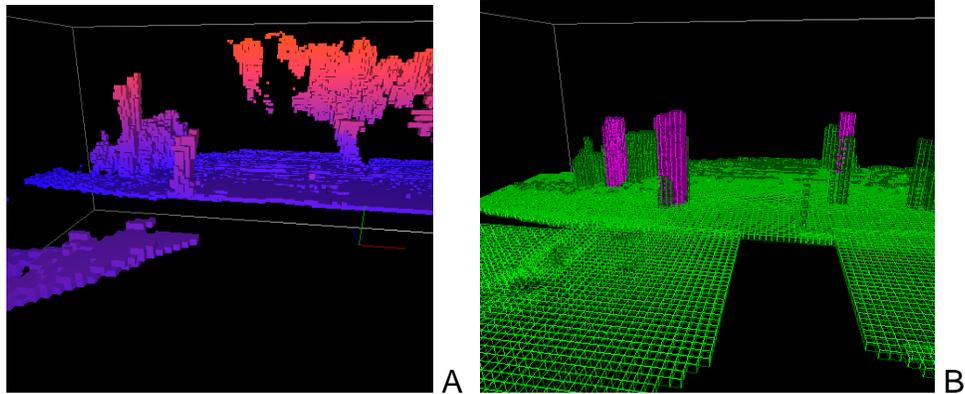


Figure 3-11. Results of brush height estimation and tree detection on data collected from a static location. Green voxels represent the ground surface, dark green the brush, and purple the tree trunks. A) The original data colored by height and B) the output.

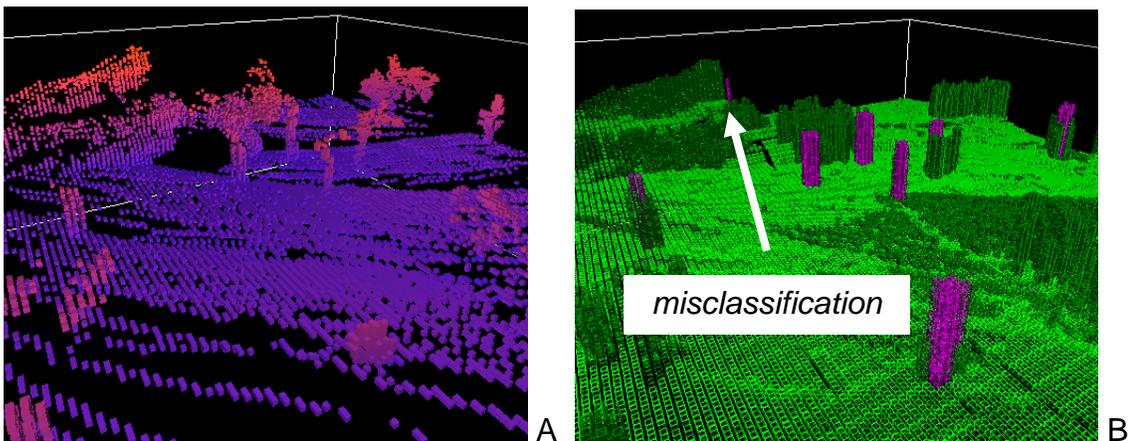


Figure 3-12. Results of brush height estimation and tree detection on data collected from a moving vehicle. Green voxels represent the ground surface, dark green the brush, and purple the tree trunks. A) The original data colored by height and B) the output.

Hough Plane Extraction

The Hough plane detection algorithm described in Chapter 2 was not used in the final version of the ground plane detection; however, a sample result is presented in Figure 3-13 for completeness. Note that the plane fits the ground data over some portion of the scene but towards the right side of the image, there is a growing discrepancy between the plane and the actual ground surface due to the curvature of the ground.

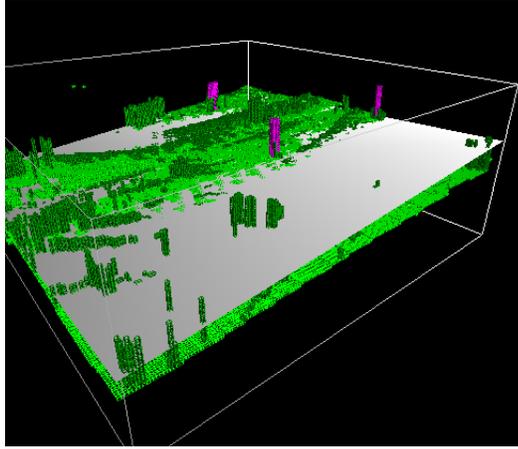


Figure 3-13. Hough plane extraction used on ground voxels.

CHAPTER 4 CONCLUSION

The use of Ladars on board autonomous vehicles can provide valuable information for the system's autonomy. While the amount of raw data presents a significant challenge at both managing and interpreting the data, there exists methods which can effectively utilize the information and to reduce it to a usable format. The two primary ways of representing the data, point clouds and voxels, were presented along with their advantages and disadvantages. The choice between the two is a design decision that depends on the type of classification scheme being used, the available hardware resources, and the ability to efficiently store and access the data.

The ability to use the Ladar's implied free space information along the length of the beam can prove to be very useful. The process of merging new data with old data is very easy in the voxel data representation, but more challenging if a point cloud representation is used. Also, the ability to merge data gathered from various sensors becomes much easier if the two sensor's data sets are stored in voxel format.

The collection of three dimensional data aboard a moving vehicle presents additional challenges because of uncertainty and drift in the estimation of the vehicle's position and orientation. The added degree of uncertainty necessitates measures to decay the data to prevent blurriness between old and new data.

While there have been many successful implementations of probabilistic classification algorithms, more deterministic algorithms like the one implemented in this work have proven to be effective at extracting useful data from a scene. One of the most important steps in the classification algorithm is the determination and representation of the ground surface. Once the ground surface is known, other relevant

properties such as the smoothness and slope of the ground can be calculated in addition to the density and height of objects above the ground. This information is critical to vehicle navigation. The interpolation process provides very good consistent results for the ground plane representation.

The application of imaging techniques such as the convolution filter is especially useful in the presence of noisy data or scenes cluttered with brush. This technique for finding trees is simple, yet effective and can be extended to objects of other shapes such as walls if desired. Other tools, such as the Hough plane transformation and blurring effects may be useful for various other tasks.

The use of performing classification multiple times a second and merging the resulting objects using a Local Knowledge Store proved useful at reducing false positives and filtering the results so that the reported object locations are more accurate. The use of Ladars on autonomous vehicles have been used for many years, and the extension of this data into a full three dimensional representation for classification will prove to be invaluable for the advancement of autonomous vehicle research and robotics in general.

LIST OF REFERENCES

1. R. Manduchi, A. Castano, A. Talukder, and L. Matthies, "Obstacle Detection and Terrain Classification for Autonomous Off-Road Navigation," in *J. Autonomous Robots*, 2004, pp. 81–102.
2. K. Hammoudi, F. Dornaika, and N. Paparoditis, "Extracting Building Footprints from 3D Point Clouds Using Terrestrial Laser Scanning at Street Level," in *Workshop. City Models, Roads and Traffic. International Archives of Photogrammetry and Remote Sensing*, vol. XXXVIII, part 3/W4, 2009, pp. 65–70.
3. C. Wellington, A. Courville, and A. Stentz. "A Generative Model of Terrain for Autonomous Navigation in Vegetation," *The International Journal of Robotics Research*, vol. 25, no.12, pp. 1287–1304, Dec. 2006.
4. C. Wellington and A. Stentz, "Learning Predictions of the Load-Bearing Surface for Autonomous Rough-Terrain Navigation in Vegetation," *Field and Service Robotics*, Berlin: Springer, 2006, pp. 83–92.
5. J. Lalonde, N. Vandapel, M. Hebert. "Data Structures for Efficient Dynamic Processing in 3-D," *Int. J. Robotics Research*, vol. 26, no 8, pp. 777–796, Aug. 2007.
6. M. McDaniel, T. Nishihata, C. Brooks, and K. Iagnemma, "Ground Plane Identification Using LIDAR in Forested Environments," in *Proc. IEEE Int. Conf. Robotics and Automation*, May 2010, pp. 3831–3836.
7. A. Lacaze, K. Murphy, M. DelGiorno. "Autonomous Mobility for the Demo III Experimental Unmanned Vehicles," in *Proc. Association for Unmanned Vehicle Systems International*, 2002.
8. D. Munoz, N. Vandapel, and M. Hebert, "Onboard Contextual Classification of 3-D Point Clouds with Learned High-order Markov Random Fields," in *Proc. Int. Conf. Robotics and Automation*, 2009, pp. 4273–4280.
9. J. Lalonde, N. Vandapel, D. Huber, and M. Hebert, "Natural Terrain Classification using Three-Dimensional Ladar Data for Ground Robot Mobility," *J. Field Robotics*, vol. 23, no. 10, pp. 839–861, Oct. 2006.
10. M. Hebert and N. Vandapel, "Terrain Classification Techniques from Ladar Data for Autonomous Navigation," in *Proc. Collaborative Technology Alliances Conference*, May, 2003.
11. N. Vandapel, D. Huber, A. Kapuria, and M. Hebert, "Natural Terrain Classification using 3-D Ladar Data," in *Proc. Int. Conf. Robotics and Automation*, 2004, pp. 5117–5122.

12. D. Anguelov, B. Taskary, V. Chatalbashev, D. Koller, D. Gupta, G. Heitz, A. NG. "Discriminative Learning of Markov Random Fields for Segmentation of 3D Scan Data," in *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2005, pp. 169–176.
13. D. Munoz, N. Vandapel, M. Hebert, "Directional Associative Markov Network for 3-D Point Cloud Classification," in *Proc. Fourth International Symposium on 3D Data Processing, Visualization and Transmission*, June 2008.
14. N. Heckman, J. Lalonde, N. Vandapel, and M. Hebert, "Potential Negative Obstacle Detection by Occlusion Labeling," in *Proc. IEEE/RSJ Int. Conf. Intelligent Robots and Systems*, Oct. 2007.
15. A. Stentz, C. Dima, C. Wellington, H. Herman, and D. Stager, "A System for Semi-Autonomous Tractor Operations," *Autonomous Robots*, Vol. 13, No. 1, pp. 87–103, July, 2002

BIOGRAPHICAL SKETCH

Ryan Chilton was born in Orange, California in 1986. Ryan received his Bachelor of Science degree in mechanical engineering at the University of Florida in 2008.

Ryan's research interests include sensing for robotics and autonomous vehicles. In his spare time, Ryan enjoys biking, playing racquetball and volleyball, and skiing when possible.