DEPTH SENSING PLANAR STRUCTURES: DETECTION OF OFFICE FURNITURE CONFIGURATIONS

By

Brendan O'Shaughnessy

B.S. Plymouth State University, 2006

A THESIS

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Master of Science

(in Spatial Information Science and Engineering)

The Graduate School

The University of Maine

May, 2012

Advisory Committee:

Reinhard Moratz, Associate Professor of Spatial Information Science and Engineering, Advisor

Kate Beard, Professor of Spatial Information Science and Engineering

Nicholas Giudice, Assistant Professor of Spatial Information Science and Engineering

LIBRARY RIGHTS STATEMENT

In presenting this thesis in partial fulfillment of the requirements for an advanced degree at The University of Maine, I agree that the Library shall make it freely available for inspection. I further agree that permission for "fair use" copying of this thesis for scholarly purposes may be granted by the Librarian. It is understood that any copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Signature: _____ Date: _____

Brendan O'Shaughnessy

DEPTH SENSING PLANAR STRUCTURES: DETECTION OF OFFICE FURNITURE CONFIGURATIONS

By Brendan O'Shaughnessy

Thesis Advisor: Dr. Reinhard Moratz

An Abstract of the Thesis Presented
in Partial Fulfillment of the Requirements for the
Degree of Master of Science
(in Spatial Information Science and Engineering)
May, 2012

Handheld devices with depth sensors have the potential to aid low-vision users in performing tasks that are difficult with traditional modes of assistance. Heuristic studies have revealed that tables have a key functional role in indoor scene descriptions. The research question addressed in this thesis is: how can we robustly and efficiently detect tables in indoor office environments? This thesis presents a solution that utilizes a functional approach to robustly detect rectangular tables in depth images generated from a Kinect sensor.

Perhaps the most significant function of a table is to provide its users with a supporting plane. This demands that the table's surface is orthogonal to the scene's gravity vector. In order to fully take advantage of this functional property in the detection process, the scene must be properly oriented. A planar model fitting procedure is used to detect the scene's floor, which is utilized to properly orient the scene.

The scene is then sliced at average table height, using a small buffer. The height component is removed from the 3-dimensional slice by projecting it into a two-dimensional plane. Next, an iterative labeling procedure is used to separate the image into independent blobs, allowing for 2-dimensional shape detection.

Sufficiently large blobs are then subjected to a cleaning process in order to remove any extraneous features. Several features of the cleaned blobs are calculated and used in a

supervised classification process. The coordinates of blobs that are classified as tables are translated back to 3-dimensions, allowing for the segmentation of all detected tables in the scene.

THESIS ACCEPTANCE STATEMENT

On behalf of the Graduate Committee for Brendan O'Shaughnessy, I affirm that this manuscript is the final and accepted thesis. Signatures of all committee members are on file with the Graduate School at the University of Maine, 42 Stodder Hall, Orono, Maine.

Submitted for graduation in May, 2012.

Signature: _____ Date: _____

Dr. Reinhard Moratz
Associate Professor, Spatial Information Science and Engineering

ACKNOWLEDGMENTS

I would first like to thank my advisor Reinhard Moratz for his support in the development of this thesis. I would also like to thank the professors of Spatial Information Science and Engineering for providing such an excellent learning environment. Special thanks to Chris Dorr, Matt Dube, Shreyans Jain, and all of my fellow students who have assisted me throughout this process. Finally, I would like to thank my parents for their constant encouragement and support.

I would also like to gratefully acknowledge that this work has been supported by the NSF Research Grant # CDI-1028895.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Human perception of indoor environments is highly dependent on obtaining visual information and processing this information using learned mental models. Visually acquiring and decoding spatial information is inherent to humans and is accomplished effortlessly. However, while newer technologies allow computers to efficiently collect spatial information, the methods for achieving a comparable sense of spatial awareness remain largely undeveloped. Providing computers with this ability would open the door for many useful applications.

Current technology is capable of supplying several modes of assistance for indoor navigation but little has been accomplished in providing users with a meaningful description of indoor environments. This capability would be particularly useful for low-vision or blind persons. For example, suppose a person with low-vision enters a room for a meeting. While current assistance techniques may allow for safe movement around the room, the task of locating an open seat can prove to be difficult, especially in new and unknown environments. This task would require the person to perform a thorough exploration of the room to first locate the table, and then to determine which chairs are unoccupied. This process can prove to be cumbersome and potentially disruptive. Ideally, a system could be constructed that provides a user with a handheld device that collects and processes 3-dimensional data upon entering an unknown room and conveys useful spatial information to them (e.g. the location of an open seat).

New depth sensing technologies show promise in supplying the data necessary for this type of application. The mass production of the Kinect sensor provides low-cost, compact hardware that can collect high quality 3-dimensional data. Another contributing factor is the steady increase in the computing power of handheld devices. These devices could provide the means for quickly processing data in order to provide real-time information to the user. This thesis provides an algorithm that allows for the detection of

tables [1] in office environments, one of the initial steps in creating methods that can utilize new hardware in order to provide useful information about indoor scenes.

1.1 Motivation

One of the first steps in developing a depth sensing device that can provide key spatial information to a user is to accurately detect the relevant objects. The question therefore is: what objects are most important to the user of such a device? The experiments discussed in Chapter 3 suggest that tables play a significant role in the description of office environments, presenting the research question: how can we efficiently and robustly detect tables in office environment?

1.2 Technical Approach

The algorithms described in this thesis were implemented using the Java programming language. The Java 3-D API was used to build an interactive 3-D point cloud viewer, as well as perform many of the rotations and transformations. Indoor scene data was collected using the Kinect sensor in conjunction with the Robot Operating System software. This process is described in detail in Chapter 4.

The simpler and less computationally expensive mathematical operations in this program were implemented in Java. However, in order to improve efficiency and minimize the size of the program, the classifiers were written using the R language. R is an open source statistical package that provides a very efficient statistical computing environment. In order to fully integrate the classifiers with the Java program, the Rserve engine was used.

Rserve is a TCP/IP server that provides an interface for Java and other programming languages to use all of the facilities of R without requiring it to be initialized by the user. Commands are written in the R language within a Java class and sent to the R server using

[1] Horizontal surfaces that provide the function of tables.

methods provided in the Rserve library for Java. The resulting output is then returned to the Java program for further analysis.

1.3 Thesis Organization

This thesis is organized as follows: Chapter 2 examines and reviews the relevant background research. The chapter begins by introducing generic depth sensing methods and then delves into a more detailed description of the hardware that was used. An overview of segmentation algorithms that are designed to detect planar surfaces in range images is also provided.

Chapter 3 references two human subject experiments, conducted by cooperating researchers, that relate to indoor scene description. Each experiment provides insight on the important role tables play in humans' perception of indoor scenes. After presenting the experimental results, potential applications of the research are discussed.

Chapter 4 discusses, in detail, the implementation of the algorithms presented in this thesis. Potential approaches for developing floor and table detection algorithms are first reviewed, after which descriptions of the selected methods are provided. This chapter also covers the classification of tables through the use of Bayesian and Support Vector Machine classifiers.

The thesis concludes with Chapter 5 by presenting a summary of the research and a discussion of future work. Appendix A includes pseudocode of the algorithms developed for floor and table detection.

Chapter 2

TECHNOLOGICAL BACKGROUND

2.1 Depth Sensing

The act of collecting 2½-dimensional data using a range finding device can be described as depth sensing. Typically, range images and point clouds are referred to as 2½-dimensional due to the single perspective utilized during data collection. Collecting data from only one perspective results in occlusions in the data. True 3-dimensional images have no occlusions and fully represent an object or scene. Depth sensing methods can generally be classified into two categories: active or passive. Passive techniques have been widely used for nearly a century, while active methods have been in development since the 1970's [5]. Due to recent advances in hardware development, active methods are currently the predominant technique for the collection of depth information.

2.1.1 Passive Depth Sensing Methods

Passive depth sensing techniques generally involve stereo camera systems. The camera system uses stereo triangulation to generate depth information. The process requires the calibration of two or more cameras that are used in unison to capture 2-D images of a scene. An algorithm, often normalized correlation or sum of squared differences, is then used to locate corresponding areas in the images. This is known as the correspondence problem and generally requires intensive calculations. The stereo camera approach has several disadvantages. The disparity problem arises when one image captures a feature that is not contained in the other image. This leads to the occlusion of objects in the scene [1]. Another common issue is that uniform areas with few objects, or areas containing many objects, can greatly reduce accuracy [14]. In the past, this process was considered to be too computationally expensive for most realistic applications. However,

present levels of technology provide reduced computation time, making the method more tractable.

2.1.2 Active Depth Sensing Methods

An active depth sensing method can be defined as a method that provides and controls its own illumination [14]. Active sensors generally employ one camera and a structured light emitter [12]. Structured light is projected into an environment where it reflects off objects in the scene. The camera used in this process is built to capture the reflections of the type of emitted light. This data is then processed to generate a range image. There are many techniques that active depth sensors utilize to accurately collect data. One of the oldest active depth sensing methods is the time of flight method.

Time of flight methods have been used in radar and LIDAR applications for decades [12], and recent technological advancements have allowed for an increasing number of useful applications. Time of flight depth sensors collect data using an active sensing process. The process begins by the transmission of a signal, generally a light wave, to the local environment. After the signal interacts with the environment, it is reflected back to the sensor. The time of flight and the speed of the signal are then used to calculate an accurate depth measurement.

Achieving accurate results using time of flight methods in an indoor environment will often require the hardware to have the ability to distinguish between femto seconds (10^{-15}) [22]. The amount of precision required to produce accurate results is inversely related to the distance traveled [22], making time of flight methods more useful for outdoor depth sensing [5]. However, there are several advantages to time of flight depth sensing methods; they generally produce very dense range images and require little or no image processing [16]. A major problem that can occur when using time of flight methods indoors stems from excessive noise in the data. In order to reduce noise while using these methods, several recorded values can be averaged providing a smoothing effect [5]. While time of

flight methods can use pulsed light or continuously modulated light sources, continuously modulated light methods produce a higher resolution range image [22].

The first implementation of structured light as a depth sensing technique occurred in the early 1980's. One of the first techniques used was Gray (or binary) coding [7]. A Gray coded pattern projects light encoded with pixel values that are either black or white, a binary pattern that is either on or off [29]. The value encoded in each pixel is known as a codeword. These patterns provide very accurate results because the pattern resolutions are exponentially increasing using coarse to fine light projections [7].

Coded structured light depth sensing involves the projection of a light pattern in which each point in the pattern is encoded with information that identifies its coordinates [27]. Once each point of light is reflected and captured by a sensor, its coordinates make it possible to calculate depth using triangulation. This method is very similar in concept to passive stereo triangulation, however the correspondence problem is solved by the encoding of the projected light. Pattern projection techniques used in coded structured light depth sensing can be distinguished by the encoding method used. While there are many specific techniques, [27] suggests classifying them into three main categories: spatial neighborhood, time multi-plexing (temporal), and direct coding.

Spatial neighborhood methods involve coding each point with information about its surrounding points. This technique uses the information from surrounding points to uniquely identify each individual point [27]. The decoding of a point's location using spatial neighborhood techniques can prove to be difficult because when values are occluded or shadowed, they are not returned. Therefore, the location of a point may not be able to be determined with certainty. Due to this uncertainty, spatial neighborhood coding techniques work generally best when measuring dynamic environments [27].

The first spatial neighborhood methods to be developed did not utilize any formal coding techniques. This resulted in methods that were not robust; mainly because non-formal codification methods can not guarantee that each point is coded uniquely [27].

This problem was solved by using De Bruijn sequences for encoding the light. A De Bruijn sequence is a cyclic sequence that uses a given alphabet and a specified length to generate every possible unique sequence. These unique sequences are used to encode the projected light, removing any possibility of repeated values. M-arrays can also be used to generate unique values. These methods are very similar to the De Bruijn methods, but function through the use of two dimensional arrays. When bi-dimensional coding is used, each point is allocated two codewords [27]. This allows for more flexible system configuration and windowed image processing.

Temporal coding is the most frequently implemented active depth sensing technique [27]. This method involves sequentially projecting several encoded signals into the environment. In most applications, the codeword for a pixel consists of the unique sequence of light values projected for each pixel. Projecting multiple signals in a set sequence improves accuracy by reducing the number of colors or shades of gray needed, therefore minimizing detection errors. Employing a coarse to fine model for projecting this type of signal can also be beneficial because the precision of the position of a pixel is increased by the time the pattern is projected [27]. In addition to excellent accuracy rates, temporal coding methods can provide high resolution range images. The main limitation to temporal coding methods is that they are only suitable for static environments [7].

Direct codification involves encoding every pixel with unique information that allows its location to be identified based on only the information from that point. In order for every pixel to be assigned a unique code, there are two primary options: encode signals with a very large range of colors or using periodicity in the pattern [27]. While direct codification theoretically performs well, methods using these techniques are error prone; they are very sensitive to noise and certain colors can also present issues. Direct codification works best for measuring static environments [27].

2.1.3 Kinect Sensor

While it was originally developed for video gaming, the Kinect sensor has provided consumers with an affordable device that is capable of collecting depth and color images. Shortly after the device was released, an open source driver was developed allowing users to connect the sensor to a computer. Due to the increasing enthusiasm for alternative uses of the device, PrimeSense, the developer of its reference design, released its OpenNI driver and NITE Middleware for public use. This allowed developers to produce applications that are capable of taking full advantage of the sensor's features.

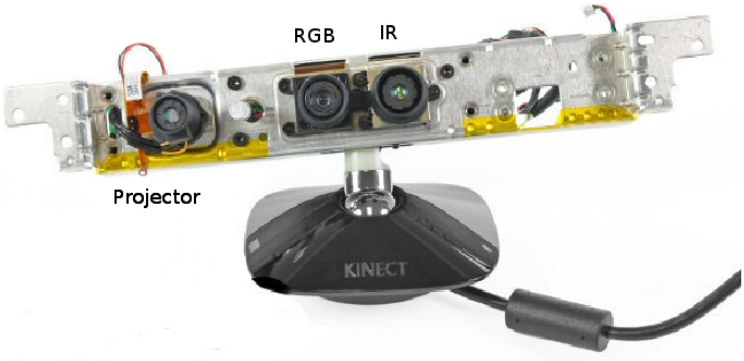

Figure 2.1. Inside the Kinect sensor [32]

The Kinect sensor consists of three main components: an RGB camera, an infrared laser emitter, and an infrared (IR) sensor. The IR camera has a fixed resolution of 640x480 pixels. In order to acquire depth information, the diode of the infrared laser emitter projects a single beam of IR light. Through the process of diffraction grating, the beam is split into pattern and projected across the scene. After the infrared light reflects off objects in the scene, the resulting pattern is captured by the sensor's infrared camera.

First, a microprocessor in the sensor decodes the pattern. Triangulation with its stored reference pattern then produces a depth image. In addition to collecting depth

Figure 2.2. The infrared pattern projected by the Kinect sensor

information, the sensor also simultaneously captures an RGB image of the scene. Through the process of registration, each 2½-dimensional point is assigned a color value.

The sensor is capable of capturing depth and color images simultaneously at approximately 30 frames per second [19]. Once the depth and color images have been combined through registration, a three-dimensional point cloud that consists of 307,200 points is produced. The release of the OpenNI drivers permit the sensor's capabilities to be fully harnessed, therefore allowing the geometric quality of the data to be assessed. By examining the accuracy and density of points, [19] has provided a thorough analysis of the sensor's performance.

Even in the most controlled environments, errors will inevitably occur when using depth sensing equipment. The analysis performed in [19] found that when collecting depth information with the Kinect, errors stem from three main sources: the measurement setup, the properties of the surfaces in the scene, and sensor itself. Errors occur during

the processing on the sensor for two main reasons: improper calibration or an inaccurate correlation of the projected pattern. The release of the OpenNI drivers has greatly reduced the amount of errors due to calibration, while the correlation errors will most often be random and are not likely to greatly affect the overall quality of the data.

Errors caused by lighting conditions, as well as the distance and orientation of objects in the scene, can be considered measurement setup errors. Intense light can greatly reduce the contrast of the projected infrared light, which will often produce outliers or gaps in a point cloud. The orientation of an object can also lead to situations where infrared light is projected on a side of an object that is occluded from the infrared camera. The opposite can also occur; the infrared camera is able to see a side of an object that the light cannot be projected onto. Both scenarios will produce gaps and missing data in the resulting point cloud.

The characteristics of an object's surface can also greatly affect the amount of errors that occur in data collection. Surfaces that are especially smooth or shiny can negatively impact the measurement of depth, producing gaps in the data [19].

Figure 2.3 illustrates a situation where the combination of an object's position and surface properties result in gaps in the point cloud. The negative effect caused by shiny surfaces is quite apparent in the middle-left section of the table, where the area that is illuminated by bright sun has a clear gap. The largest gaps in Figure 2.3 appear in the quarter of the table that is furthest from the sensor. The back wall in the scene is clearly further away from the sensor than the table, but the wall has no noticeable gaps. The angle between the sensor and objects can also play a factor; when the angle between the sensor and the table's surface is greater, less gaps appear. However, increasing the angle also limits the sensor's field of view.

While it is essential to acknowledge that errors can and will occur during data collection, the sensor's limitations must also be taken into account. One key limitation is that as the distance between an object and the sensor increases, the point density of that

Figure 2.3. Gaps in a point cloud due to the properties of a table's surface

object decreases. This is due to the fixed resolution of the infrared camera; the number of points in an area is proportional to the square distance from the sensor [19]. The Kinect sensor collects depth measurements using 11-bit integers, of which 10-bits are allocated for recording these measurements. This produces depth images that contain 1024 levels of depth. However, one level of depth at a distance of 2m from the sensor is much less than one level at a 5m distance. This is due to the inverse relationship between depth resolution and levels of depth; depth resolution will decrease as the distance from the sensor increases.

2.2 Detection of Planar Surfaces

As discussed in Section 2.1, a range image can be defined as an image that contains 2½-dimensional data about a scene from a single perspective. Segmenting a range image

involves splitting the range data into multiple meaningful regions. Each region contains a complete set of points that belong to a surface or object. The aggregate of all regions must equal the entire range image. The majority of range image segmentation methods can be categorized as edge based, region based, or a hybrid of the two.

The process of extracting flat surfaces from range information is known as planar segmentation. Surface normals are often utilized in planar segmentation. A surface normal is a vector that is perpendicular to a surface's orientation. Knowledge of normal vectors can be utilized to group points into homogeneous regions [6].

While edge based methods will be briefly covered, this research focuses mainly on region based algorithms. Region based segmentation methods can be classified as either region growing or parametric model based [15], both of which are discussed below.

2.2.1 Region Growing Segmentation

Region growing segmentation algorithms generally begin by creating preliminary regions or seed points of a range image. The seed points can be created through random selection or user defined criteria such as an evenly spaced grid or based on a certain pixel value. The process of seed creation should be done carefully; the chosen method can greatly affect the success of the segmentation.

Once the seed points have been selected, the image is split into its initial regions and a region growing strategy is implemented. Seeded region growing (SRG) strategies generally have allocated pixels, which are labeled as belonging to a region, and unallocated pixels which are not yet labeled. Each unallocated pixel is entered into an equation that analyzes its location and/or color. Based on the results and the values of its neighbors, it is then assigned to a region. The centroid of the modified region is then recalculated based on the pixel's addition. This process continues until all pixels are assigned to a region. A drawback to this type of region growing algorithms is that the resulting boundaries are often distorted because segmentation is done at a regional level, not universally.

The adaptive distance dynamic clusters algorithm (ADDC) is a region growing segmentation algorithm that can be used to locate planar regions in 3-dimensional data. The algorithm attempts to discover clusters that are located within regions of the feature space through the use of affine manifolds or kernels as symbolic representations of the clusters [20]. While the ADDC algorithm performs very well when clusters are separated, clusters that are close together or cross each other can prove to be problematic. Although there are several variations of the algorithm that attempt to reduce this type of error, it is always of concern. [20].

A planar segmentation method presented in [17] involves using scan lines. This method can also be classified as region growing and involves using straight line segments as seeds instead of the traditional method of using pixels. Using line segments instead of pixels considerably reduces the amount of data processing necessary for the growing process. The first step in the algorithm involves filtering the range data to reduce the amount of noise in the data. Filtering range data is often the first step in segmentation algorithms, however it is important to ensure that the edges are not lost in the process through over-smoothing. The algorithm presented in [17] uses the Median filter on a 3x3 neighborhood to reduce the noise in the range data. This planar segmentation algorithm attempts to group points based on the planar surface they belong to. In order to accomplish this, the algorithm must use a threshold to determine if a point fits a planar surface. The threshold should be determined directly from the data to allow the algorithm to be capable of handling range data with varying levels of noise. In order to calculate a proper threshold, an estimation of noise variance is computed. This is done by fitting a least squares plane $z = Ax + By + C$ to a 3x3 neighborhood of each pixel.

In cases where the pixel is located in the interior portion of the planar surface, the error is most likely because of noise in the data. In cases where the pixel is not on the interior surface, it is generally near a jump boundary and should be discarded to ensure that it is not used in the calculation of the threshold. The root-mean-square error

of all qualifying pixels is used to produce an optimal threshold value for the segmentation process.

The final step before segmentation can occur involves dividing each scan line into straight line segments. The authors of [17] use a slightly modified version of Duda and Hart's split and merge algorithm for this process. This algorithm keeps the start and end points of the line, while recursively dividing the line until a threshold is reached. This essentially performs a smoothing process on each line. Line segments are stored as (x_1, x_2, y_0) where y_0 is the y-coordinate of the scan line, x_1 is the location of the leftmost pixel of the line segment, and x_2 is the rightmost pixel in the line segment.

Determining seed regions is done by grouping three neighboring line segments that meet a set length criteria. All qualifying line segments are grouped and then the best candidates are determined. While the least square plane fitting method can be used for this process, it is not ideal because of its computational intensity. Therefore, the authors decide to test if each line segment has the same slope but a different intersect, making them parallel. However, this presents a problem because a_i and b_i are not appropriate metrics for this type of comparison. In this case, "the absolute difference in space is a nonlinear function of the relative difference of such kind of parameters" [17]. In order to get a usable comparison, the normals of line segments are used. Normals m_i and m_j are used $(m_i * m_j)/(|m_i| * |m_j|)$ which is equal to the cosine of the angle between the two normals. This calculation is completed for each line segment (s) and one is chosen as the best candidate to begin region growing.

For the seed region R_k, the least square plane P_k is found using by selecting the plane with the least squared error. Once the optimal plane is calculated, all line segments that neighbor s are tested using θ as the threshold. The right and left pixels of the line segment are used to test the fitness of line because the maximum amount of error will occur at these pixels' locations. This reduces the computation time by making it unnecessary to test every pixel in the line segment. When a line segment is determined to be part of the

plane, it is added to a linked list (L) that stores all of the region's line segments. The region growing process is iterative; once all unlabeled neighbors are tested, the process begins again using the updated linked list.

This type of segmentation algorithm often does not provide perfectly clean edges between regions. However, post-processing techniques can help remove some of the noise from the edges. The method presented by Jiang and Bunke in [17] involves iteratively testing the rightmost and leftmost pixel of each line segment to its neighbor. If it is a better fit, the pixel is added to the neighboring line segment.

2.2.2 Edge Based Segmentation

Range images are often excellent candidates for edge based segmentation due to the availability of depth information. The color of a pixel can be used for edge detection, however it often does not produce ideal results. Using depth information, edges can be detected using jump boundaries. Jump boundaries are basically discontinuities in the range image. The detected edges are analyzed and grouped through common properties. The groups of edges are then used with an edge based segmentation algorithm to segment the image. While edge based segmentation generally produces regions that have minimal noise and are very well defined, these methods will often generate gaps between boundaries [17]. Another drawback to edge based methods is the potential of poor performance when segmenting curved surfaces; the discontinuities are smooth and difficult to locate, frequently resulting in under segmentation.

2.2.3 Parametric Model Segmentation

Parametric model based segmentation is also considered to be a region based segmentation method. These methods involve grouping data points which all belong to a surface or object that fits the specified model. Model based segmentation algorithms can be classified as either general or dedicated. General methods involve segmenting the data

with only general knowledge about the surfaces involved, while dedicated methods attempt to locate a specific structure in the data, such as a plane.

2.2.3.1 The Hough Transform

The Hough transform is a feature extraction technique that was patented by Paul Hough in 1962 and was popularized by Duda and Hart in 1972 [10]. The technique involves transforming data to a parameter space that allows object candidates to be detected through grouping techniques. While the traditional Hough transform uses a two dimensional feature space, adding a third dimension is necessary for detecting planes.

Okada et al. [26] present a planar segmentation method that involves the use of a three dimensional Hough transform. This transform requires additional parameters to account for the third dimension, as shown in Figure 2.4. In a three dimensional region space, ρ is the distance between the plane and the origin, ϕ is the angle against the x-axis, θ is angle against the y-axis, and (x_1, y_1, z_1) represents the point on the plane.

This equation is used to extract plane segment candidates. All three dimensional points are transformed into Hough space where peak points or clusters can be detected. These clusters are equivalent to planar segmentation candidates in three dimensional space. The detected clusters are then transformed back to 3D space as a planar object. For each planar segmentation candidate, a virtual plane must be created. This virtual plane is then used in conjunction with the original point cloud or depth map to calculate the distance between one another at each point or pixel. Using a threshold, all points are tested based on their proximity to the plane and are labeled accordingly. The authors also discuss several variations to this method that improve accuracy and reduce computation time. A more complete description of the Hough transform variations is described in [3]. Probabilistic, Adaptive Probabilistic, Progressive Probabilistic, and Randomized Hough Transform are a few of the variations mentioned.

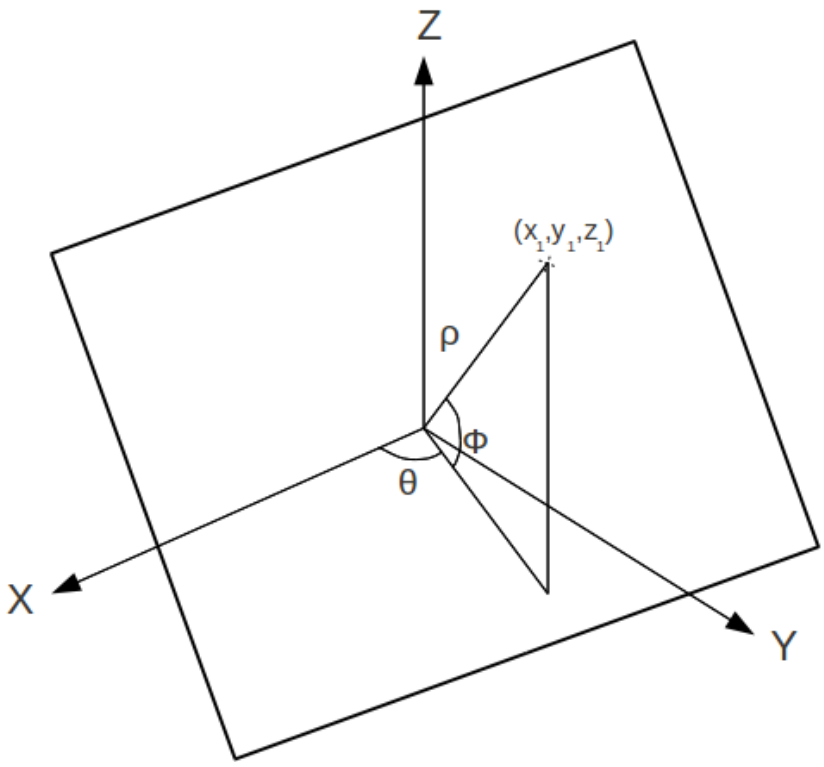

Figure 2.4. Representation of 3-dimensional Hough space

2.2.3.2 Random Sample Consensus

Random Sample Consensus (RANSAC) is an iterative algorithm that can be utilized in planar segmentation. RANSAC is a powerful algorithm because of its simplicity, its ability to ignore noise in the data, and its flexibility. When using the algorithm, it is assumed there are only two types of data: inliers that can be modeled by a set of parameters, and outliers that do not fit the model.

RANSAC is a robust algorithm that is able to effectively work with data sets where over half of the data are considered to be outliers. Many other segmentation methods attempt to use as much of the data as possible and then remove data that does not fit. The RANSAC approach essentially does the opposite; it begins with the smallest possible data set and attempts to find data that is consistent with the initial model in order to enlarge the data set [11].

When using RANSAC to detect planar regions in a point cloud, the initial sample (also known as the consensus set) should consist of three points (S_1). Three points is the minimum number of points (n) that can be used to define a unique plane. In this case the instantiated model is represented as M_1. Once a plane is established, each point in the point cloud is then tested to determine if it fits on the plane. A predetermined error tolerance (e) is used as a threshold, and if the point falls within e of the plane it is added to S_1. If the number of points that fall within e of the plane is greater than the threshold (t), the points can be removed from the point cloud and classified as a plane. The process is continued by using S_1, which has been updated, to compute a new M_1 and then find all points that fall within the threshold of M_1. This process continues until there are no remaining points within the threshold of M_1.

The next step is selecting a second consensus set (S_2) and repeating the process. This process must not be repeated indefinitely; at some point the algorithm must stop selecting new consensus sets. The number of trials (consensus sets selected and tested) to be run before stopping is determined by calculating the number of expected trials $E(k)$ it takes to find a given number of good data points. The probability that a given point is located on a planar surface is represented as w.

$$w = \frac{numberofinliers}{totalnumberofpoints} \tag{2.1}$$

The value of w is often an unknown and must be estimated to calculate the number of expected trials:$E(k) = w^{-n}$. In order to ensure that a planar structure is not overlooked, it is important to exceed the number of expected trials by at least one standard deviation [11]. The standard deviation is calculated using

$$SD(k) = \frac{\sqrt{1 - w^n}}{w^n} \tag{2.2}$$

The error tolerance (e) is the threshold used for determining if a point is part of a plane. This value must be calculated before the algorithm is run. In an ideal situation, the model is a simple function of the data points and the error tolerance can be determined analytically. In cases where the relationship between the model and data is not a simple function, often times the best method involves trial and error. One approach is to use several sets of sample data, compute the model, and calculate the average measured error. The error tolerance can then be set to one or two standard deviations above the average error measurement [11]. When using RANSAC for planar segmentation, the largest planes will generally be detected first; smaller planes may only be detected after the larger ones have been removed [34].

While RANSAC is related to the Hough transform, it is a much less computationally expensive. It also has the advantage of providing a flexible measure of consensus; this ensures that the amount of planes that are found will be based upon the amount in the image and not on chance alone [34]. A potential downfall of RANSAC is that it can be difficult to determine the proper number of iterations that should be run. In cases where the upper bound is set too small, the best fitting model may not be optimal and could potentially not even represent a real structure. Detecting false structures in the data is a major disadvantage to the RANSAC algorithm. Certain man made structures can repeatedly cause non-existent planes to be segmented from the data. When using RANSAC to detect planar surfaces, this type of error can occur very easily on structures such as a set of stairs as shown in Figure 2.5, where enough points fall within the threshold to segment only one planar surface from an entire set of stairs.

In [2] the method (Seq-NV-RANSAC) is presented for planar segmentation by modifying the RANSAC algorithm in an attempt to avoid classifying spurious objects as planes. The algorithm first obtains neighbors for every point. The points are then clustered using a clustering method; KNN, fixed distance neighbors, and TIN are mentioned. The

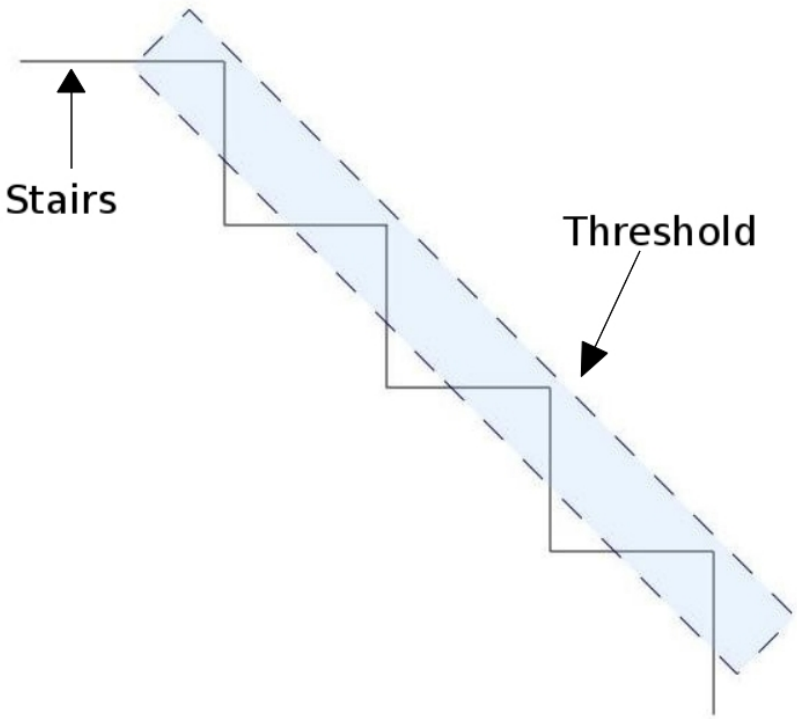

Figure 2.5. Falsely detected planar structure using the RANSAC algorithm

least squares method is then used to calculate the best fitting plane for each group along with the normal vector of that plane.

The normal vector of a group is then used with each point in the group to calculate the perpendicular distance between the two. Points that have a perpendicular distance that is less than the threshold value are then added to the group. This method will often include noisy data in the groups, but the following step that utilizes the RANSAC method easily handles noisy data. The segmentation is then performed using the standard RANSAC methodology with an additional constraint; each point's normal vector must be within the threshold of the consensus set's normal vector for it to qualify. This process is repeated until there are no more detectable planar structures in the data. This method is best used in cases of extremely noisy data due to the increased cost of computation.

Another interesting normal vector based RANSAC approach was presented in [4] is known as the Normal Driven RANSAC (ND-RANSAC) algorithm. This method only

uses the normal vectors for selecting the three points that make up the sample set; all three point's normal vectors must have an orientation that is within a threshold. The use of ND-RANSAC is generally limited to situations where the data contains excessive amounts of outliers.

This section discusses several approaches to detecting planar structures in 2½-dimensional data. While these sections focus mainly on the generic advantages and disadvantages of each method, the analysis in Section 4.1.1 concludes that using the traditional RANSAC algorithm is the optimal approach for plane detection in this research.

Chapter 3

INDOOR SCENE DESCRIPTION

Sighted humans are capable of detecting the configuration of indoor environments through visual input. However, in instances where this resource is unavailable, becoming familiar with the layout of an unknown space can be challenging. Consider a situation that requires learning an indoor environment's configuration without any visual input; in place of visual information, a natural language description of the scene's configuration is used. What type of information should that description contain? Are there certain object types that should be given preference due to their saliency? This chapter addresses these questions by referencing experiments conducted by researchers in the Virtual Environments and Multimodal Interaction Lab (VEMI), a research group directed by Dr. Nicholas Giudice.

3.1 Experimental Results

While it may seem intuitive that tables play a key role in describing indoor scene configurations, it is important to further explore this supposition. In order to accomplish this, an analysis of human subject studies was performed. This section focuses on two human subject experiments conducted by [18] that involved the subjects' description of indoor scenes. These experiments were performed with the purpose of further investigating methods for generating spatial natural language descriptions from visual scenes. The experiments described in this section, as well as the algorithms presented in this thesis, were developed for the National Science Foundation funded project, "Perception of Indoor Scene Layout by Machines and Visually Impaired Users"[1].

The first experiment involved having the subjects rank objects based on their importance to the functionality of the indoor scene they were asked to observe. The subject were all presented with a typical office scene that they had not previously encountered.

[1]NSF Grant #CDI-1028895

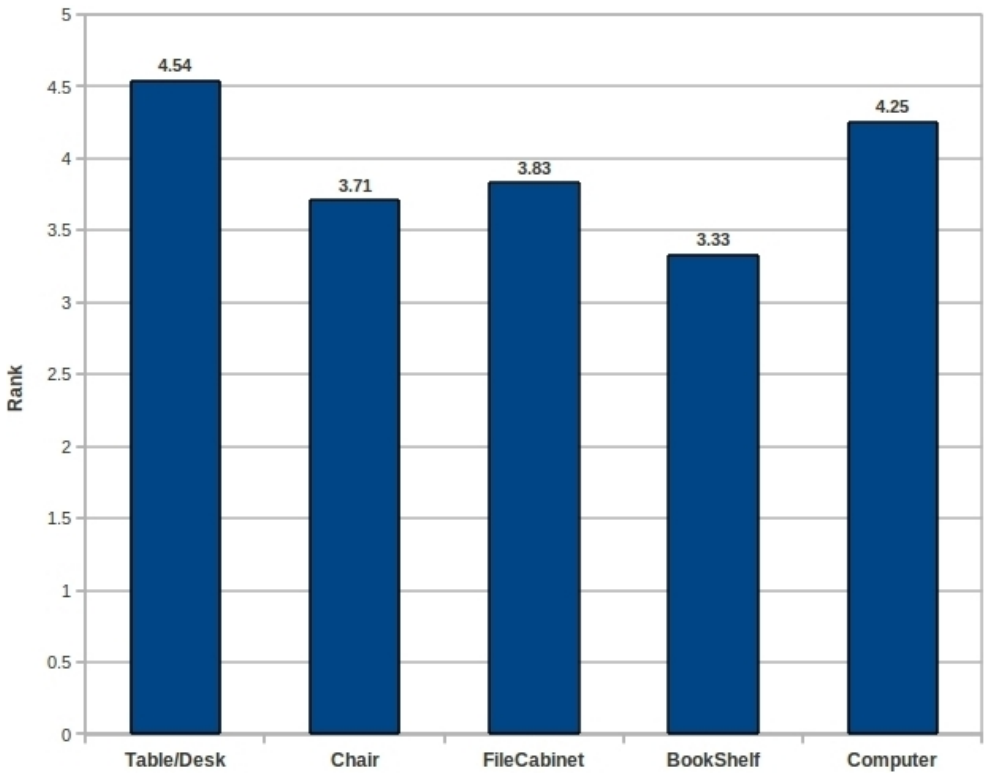

Figure 3.1. Average rankings of common objects in indoor scenes

They were then asked to provide a ranking for each of the observed objects. The ranking was based upon an object's perceived importance and was scaled 1-5, 5 being the most salient to the room's layout and functionality. Although the scene consisted of many types of objects, five were distinguished as most important: table/desks, chairs, file cabinets, bookshelves, and computers. As seen in Figure 3.1, the subjects ranked the table/desk object as the most important, receiving a mean score of 4.54 out of 5.

The second experiment involved collecting subjects' verbal descriptions of an office environment. Each subject was first introduced to a previously unseen office-like environment. They were then asked to describe the scene in a way that would best relay its configuration to another person. The verbal descriptions were then analyzed in order to determine the frequency that each object was mentioned.

However, simply recording the number of times each type of object was mentioned would produce biased results; several object types had multiple instances in the scene. For

Object	Total Count	Mean Per Person
File Cabinet 1	63	2.63
File Cabinet 2	61	2.54
Trash Can	13	0.54
Door	49	2.04
Bookshelf 1	54	2.25
Bookshelf 2	51	2.13
Table 1	63	2.63
Table 2	84	3.50
Table 3	64	2.67
Chair 1	25	1.04
Chair 2	18	0.75
Computer	28	1.17

Table 3.1. Object frequency in verbal descriptions

example, there were three tables and two chairs, but only one computer. Therefore, the total count of the number of times each object was mentioned would be skewed toward objects that had several instances. Fortunately, the data was collected in a way that allowed each instance of the objects to be distinguished from one another as shown in Table 3.1

Table 2 was the object most frequently mentioned by the subjects with an average of 3.5 mentions per person, while the trash can was the lowest, receiving only 0.54 mentions per person. In order to provide a mean value for each object type, the objects with multiple instances were combined and averaged as shown in Figure 3.2. This figure indicates that tables were the most referred to type of object in this experiment overall.

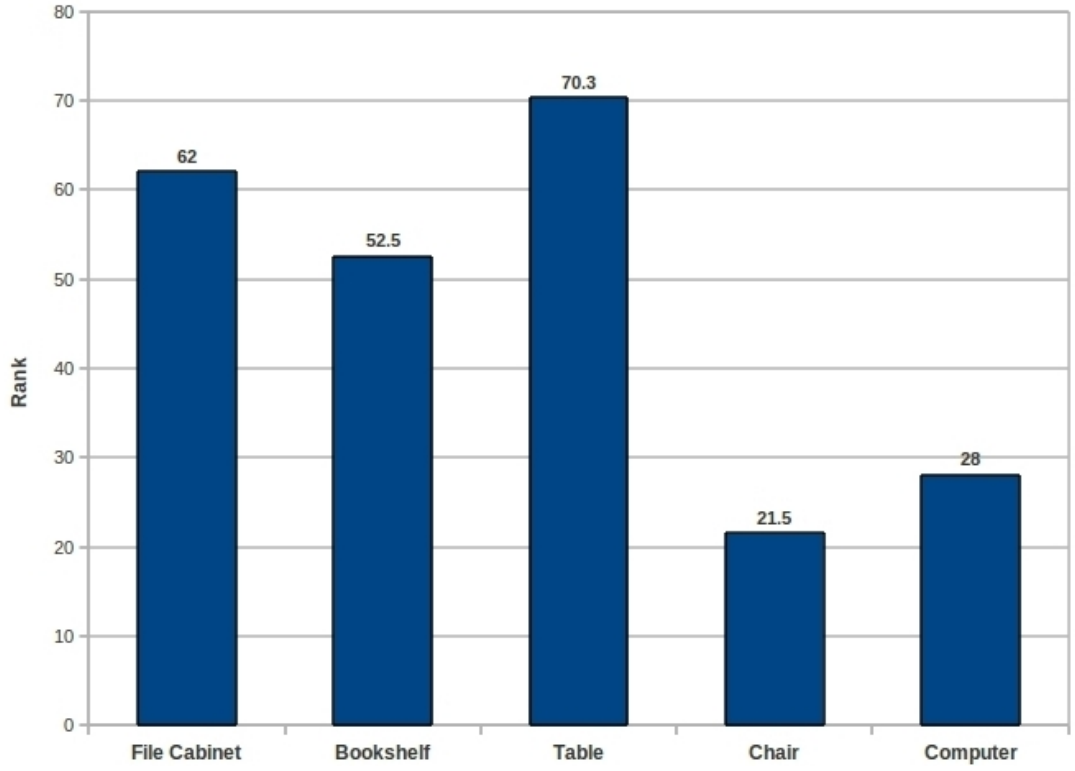

Figure 3.2. Mean number of occurrences per object type

The data presented above indicate that tables play a key role in the description of office-like indoor scenes. For example, picture yourself entering a conference room. Conference rooms generally have a large table that is centrally located. The locations of other objects are most likely affected by the position of the conference table; chairs are positioned around the table, while other objects may be placed on the table. Office or classroom scene configurations also frequently follow the trend of having objects positioned around a table's surface.

The fact that tables were the most frequently cited object type in the verbal scene description allows us to make several inferences. First, information regarding tables' location is instrumental in describing office-like indoor scenes. Also, tables were repeatedly used to reference other types of objects in the scene. Therefore, the location of a table can provide a reference point allowing the location of other objects to be extrapolated.

In [33] the authors develop a computational model that is used to describe several objects' spatial properties through the use of a spatial reference system. The authors also discuss the three types of spatial reference systems:

> "In localising reference objects in space, humans have - broadly speaking - three kinds of reference systems at their disposal, which (in Levinson's [21] terminology) may be called *intrinsic*, *relative*, and *absolute*. In intrinsic reference systems, objects are located by referring to the intrinsic properties of another entity, such as the speakers front in *The ball is in front of me*. Relative reference systems depend on the presence of a further entity (the so-called *relatum*), as in *The ball is in front of the table*. Absolute reference systems depend on the earths cardinal directions, such as north or south." [33] (p. 1-2)

The principles presented in [33] could be applied to this research through the construction of a relative spatial reference system that is based around the location and attributes of tables in the scene.

3.2 Applications

The previously described experiments indicate that tables play a key role in humans' perception and description of office-like indoor environments. This leaves us with the question: what applications can take advantage of this knowledge? The use of functional reasoning would provide an opportunity for other objects in the scene to be recognized in a much more efficient manner. For example, a chair will most often be located within a small distance of a table or desk's surface. Therefore, once a table's location is known, other objects in its vicinity can be analyzed differently than an object with no spatial reference information. In this case, the knowledge that there is a high probability that the object is a chair could allow a classification method to be developed that minimizes classification error and computation time.

The relationships between objects could then be further exploited through additional relationships between multiple objects. Once a chair has been detected in close proximity to a table, the chair's properties could be used to locate other chairs around the table. Knowledge of a single chair's properties could be utilized to build upon the initial detection method. In typical office environment interior design patterns, the shape and color of all of the chairs around a table are consistent. Therefore, these attributes could be collected and analyzed in order to develop methods that provide fast and accurate chair detection.

When providing navigational information to a low vision or blind user, it is important to present them with only the most useful information. The importance of filtering out non-crucial information stems from the amount of information that can be successfully transmitted through non-visual channels. In this case, we can utilize the knowledge that tables are key to describing the layout of the room by presenting a low-vision user with spatial information describing a table and its surrounding objects before a less important object such as a bookcase or file cabinet.

Chapter 4

DETECTION ALGORITHMS

The floor and table detection methods presented in this chapter are function-based object recognition algorithms. This type of object recognition system uses knowledge of an object's functional properties in the detection process. First, a set of functional primitives are defined for an object type. For example, [30] provides the functional primitives for a chair. The functional primitives in their model are: provide sittable surface, provide stable support, and provide back support. The ground plane is used to determine if the sittable surface is the proper height and if it is stable. Objects that fulfill these functional primitives can then be classified as a chair.

An alternative to this approach is model-based object recognition. This type of recognition system requires a set of models that the algorithm attempts to locate in a scene. For the purposes of this research, a model-based system is not ideal since tables greatly vary in size and shape. However, all tables perform the same function, even if they are not physically identical. Two key functional primitives for tables are that their surface must be located at a comfortable height for a sitting person and they must provide stable support. The algorithms discussed below use these primitives in the table detection process.

4.1 Floor Detection

Using the OpenNI driver provided by PrimeSense, the Kinect sensor is capable of producing 2½-dimensional point clouds. A point cloud can be formally defined as a set of three dimensional points that all belong to one coordinate system.

$$p_i = \{x_i, y_i, z_i\} \in P = \{p_1, p_2, \ldots, p_i, \ldots, p_n\} \tag{4.1}$$

In addition to spatial information, points can also be augmented with non-spatial information, such as a color or segmentation information.

Software provided by Robot Operating System (ROS) allows point clouds to be stored as a PCD file in one of two formats: binary or ASCII. For the purposes of this research, the ASCII format was chosen. The standard PCD file created using the ROS software begins with a header that indicates the type, size, and viewpoint of the data. Below the header, each point is represented using four floating point numbers: x-value, y-value, z-value, and RGB value. While all color values are consistently recorded using the color camera, depth values are occasionally missing. As discussed in the sensor's accuracy analysis, the entire pattern of IR light projected by the device does not always return to the IR sensor.

When RGB values are present and no depth values are available, a *nan* value is used to represent the x, y, z values. The floor detection program first reads the data from the PCD file, eliminating the header and all *nan* values. The *nan* values are removed because they are not relevant to the floor detection due to the absence of locational information, while each valid data point is stored in the program. The program then converts the cleaned file to binary form to increase the performance in future steps.

For the data collection process, we selected a set configuration for the sensor. The sensor was mounted approximately 2 meters above the floor and at an angle of roughly 30°. This configuration was chosen for several reasons. First, it captures the scene in a way that attempts to minimize errors occurring due to the scene's geometric features, while concurrently ensuring that a sufficient amount of the scene is included. This configuration is also meant to simulate situations where the depth sensing device is contained in a handheld device or mounted on the user's head. Configurations where the angle of the sensor, as discussed in Section 2.1.3, is less than 30°, frequently produce missing data values. Conversely, increasing the sensor's angle reduces the amount of the scene that can be captured. This can lead to an improper orientation, as well as the partial or complete exclusion of relevant objects in the scene.

In order to compensate for the tilted sensor, the entire point cloud is rotated around the x-axis by 30°. This rotation is not designed to precisely orient the scene and it will not consistently achieve this goal. However, we assume that it will result in a coarse orientation because of the consistent configuration used in the data collection process. In order to achieve the precise orientation that is required for robust feature detection, the planar surface that represents the floor must be found using a planar segmentation algorithm. This initial rotation provides us with sufficient knowledge of the scene's orientation to consistently detect the floor, as described in 4.1.2

Detecting the floor is not only essential to properly orient the scene; it is also instrumental in the table detection process. In [25] the author discusses the concept of the supporting plane and how it can be utilized in object recognition. An object part whose main function is to support another object, must have a surface parallel to the ground. Therefore, the surface must also be orthogonal to the gravity vector. This knowledge can be taken advantage of once the plane representing the ground has been detected.

4.1.1 Method Feasibility

For the purposes of this research, the seeded region growing (SRG) strategies discussed in Section 2.2.1 are not ideal. The computational cost of determining each point's neighbors is exceedingly large. In addition, the high probability that a plane could be erroneously split into two segments is unacceptable. This type of error is more likely to occur in a seed generation process. It is not feasible for the user to select seeds manually, therefore an automated seed generation process must be implemented. While this process will find a plane, it may not be the best fitting plane (one that includes all points belonging to the floor). The table detection process is very dependent on finding only one plane that accurately represents the entire floor.

The Hough Transform is a more robust segmentation method than SRG; it can easily handle noise and occlusions. Although there are variations of the algorithm that attempt

to reduce calculation time, it is still very computationally expensive in three dimensions [9]. It is also important to note that when using the Hough Transform in three dimensions, the closer a plane's direction is to vertical, the greater the risk of poor results. This is due to the fact that when larger values are transformed to Hough space, such as those on vertical plane, the noise in the data is magnified, making clusters more difficult to detect and producing infinite slopes. Although [3] suggests that these risks could be minimized by using Hesse normal form to define planar structures, the cost of constructing and storing an accumulator for a 3-dimensional Hough Transform is extremely large.

Therefore, a RANSAC based algorithm is our choice for the floor detection step. When considering the computational cost of building an adjacency matrix used by region growing methods or the number of calculations required by the Hough Transform, a RANSAC based approach is clearly the most efficient. While a RANSAC based algorithm offers efficiency and robustness, it also presents the possibility of detecting spurious planar structures. However, there are measures that can be taken to greatly reduce this possibility. As previously discussed, utilizing knowledge of a scene's configuration in combination with position and angle of the sensor is key to minimizing the risk of detecting a planar object other than the floor.

4.1.2 Methodology

The first step in this process involves rotating the entire point cloud by 30° around the x-axis. This initial rotation attempts to account for the angle at which the data was collected. However, this rotation will most likely not result in an orientation precise enough for the robust detection of tables; a more precise orientation is required. After the initial rotation is performed, the z-range or height of the rotated point cloud is calculated. Every point cloud is populated with points that are measured in meters and are represented as floating point decimals. The z-values generally range from -2.0m to 2.0m. After the z-range of the point cloud is calculated, all points belonging to the lowest 15% are added to a new

point cloud. This new sliced point cloud will be used to determine the equation of the plane that represents the floor. It is not ideal to simply detect the largest plane in the entire point cloud for several reasons. This method would require unnecessary computation and depending on the scene's configuration, the largest plane will not always be the floor.

Due to the data collection requirements and the initial rotation of the data, the largest plane in the sliced point cloud typically would be the floor. Therefore, determining the equation of the largest plane in the sliced point cloud and applying this equation to the complete point cloud will return the entire visible floor.

In order to find the largest plane in the sliced point cloud, the RANSAC algorithm is used. Three points from the sliced point cloud are chosen at random and are used to generate an equation describing a plane. The distance from the plane of every remaining point in the sliced point cloud is then calculated. The sum of these distances and the equation of the plane is then stored. Three random points are chosen again from the sliced point cloud and the same process is run. If the sum of distances from the plane is less than that of the previous plane, the new plane's equation and sum of distances are stored as the best model.

Another method to determine the best fitting plane involves the same general process, but measures the number of points that fall within a threshold of the plane instead of the sum of distances. This process selects the plane with the greatest number of points located within a set threshold of it. While both methods have individual benefits and drawbacks, the current implementation uses the threshold method due to its superior performance in the testing phase.

The number of required iterations can be estimated by using the probability of each point belonging to the floor. As discussed in section 2.2.3.2 the equation

$$w = \frac{numberofinliers}{totalnumberofpoints} \tag{4.2}$$

is used to determine the probability that a single point belongs to the floor. Through the use of sample data, we have determined that the average floor slice in our data set contains over 50% inliers. The slice contains a large percentage of inliers because of the improved orientation from the initial rotation, as well as the sparseness of non-floor points in the lower regions of the point clouds. By recording the number of inliers in the sliced point clouds for several scene configurations, it was determined that the average value for w is 0.584.

The probability that all points in any consensus set belong to the floor can be calculated using w^n where n is the number of points needed to construct the model; in this case $n = 3$. Therefore, the probability that three inliers are selected is 0.584^3 or 0.19917. This probability can then be used to calculate the number of iterations required to produce a model that contains all inliers (points belonging to the floor), with a high degree of confidence. In [11], the seminal work done by Fischler and Bolles, they present a method to perform RANSAC using a set confidence level. The equation they use is

$$k = \frac{log(1-p)}{log(1-w^n)} \tag{4.3}$$

For the purposes of floor detection at a 99.9% confidence, the equation is

$$k = \frac{log(1-.999)}{log(1-0.584^3)} \tag{4.4}$$

Therefore, $k = 31.099$. While k number of trials should be sufficient, [11] recommends exceeding k in order to ensure the number of trials is large enough to produce optimal results.

They accomplish this through the use of the standard deviation for k.

$$SD(k) = \frac{\sqrt{1-w^n}}{w^n} \tag{4.5}$$

In this case the values are

$$SD(k) = \frac{\sqrt{10.19917}}{0.19917} \tag{4.6}$$

and one standard deviation = 4.493. Two standard deviations are then added to the number of trials k to gain additional confidence, bringing the final number of required iterations to 41. After the k, or in this case, 41 iterations have been run, the planar equation that best describes the floor is returned.

This equation and the entire point cloud are now used to find all of the points that belong to the floor. In order to accomplish this, every point's distance to the best fitting plane is calculated. Points within the threshold of the plane are labeled as belonging to the floor, while points that are not within the threshold are labeled accordingly. Determining the optimal threshold value was accomplished through the manual process of visual inspection. Several scene configurations were inspected at various thresholds in order to determine the ideal threshold value.

The initial threshold value of 0.02 was determined to be too low because, as seen in Figure 4.1, many points that belong to the floor were not included in the segmentation. Using the value of 0.1 was deemed too high because many non-floor points were mislabeled as belonging to the floor. After evaluating several values, it was determined that 0.04 is the optimal value; values greater than 0.04 tend to include points belonging to other objects than the floor, and values less than 0.04 generally do not include all of the points belonging to the floor. For the purpose of this research, it is acceptable to have the algorithm fail to detect a very small number of points belonging to the floor, but not to include non-floor points; It is essential that the normal vector to the floor is calculated using three points that actually belong to the floor.

Detecting the floor provides enough information to easily calculate the gravity vector of the scene. The gravity vector can be described as the normal vector to the plane representing floor and is used to precisely orient the scene. In order to calculate the gravity vector, an equation of a plane derived from three points is required. The best

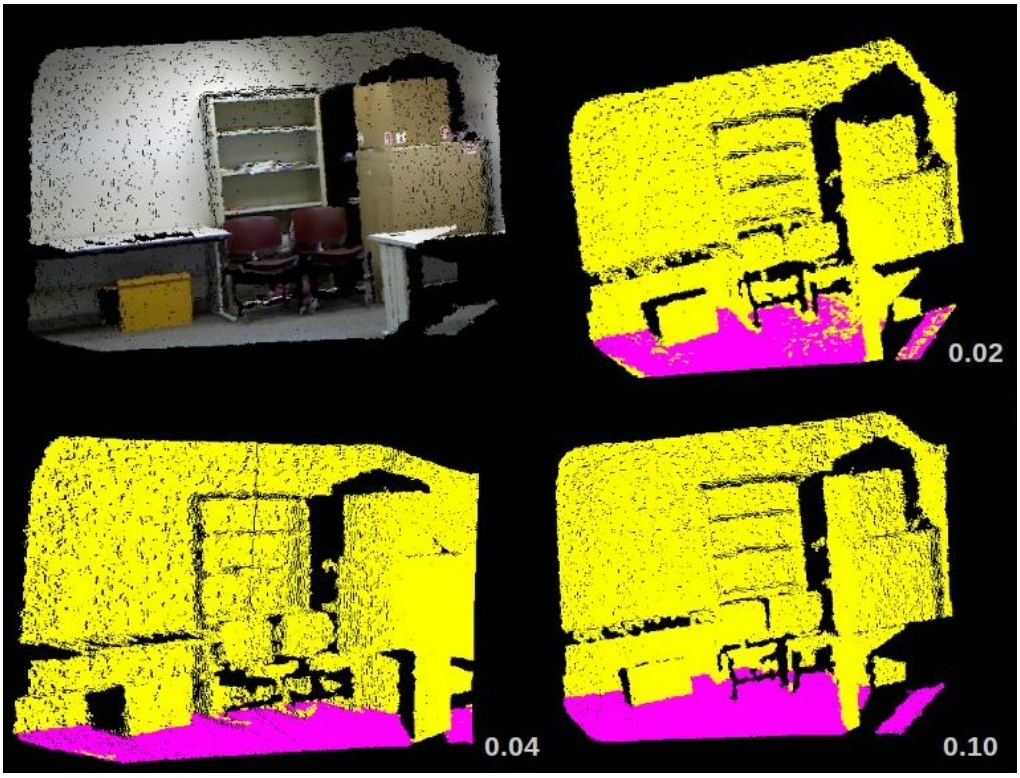

Figure 4.1. Various thresholds used for floor detection

fitting plane from the previous step could be used; it was proven to be a good fit and should be representative of the entire floor. Another method is to use three random points that are labeled as belonging to the floor to calculate the normal. This option could be improved by performing this operation several times and calculating the sum of every point's distance from the plane. The normal of the plane with the lowest sum of distances would then be used for the final rotation.

Although the above methods are all valid options, an approach that produces a more accurate result was implemented. In general, the greater the distance between the three points that are used to generate the plane equation, the better fitting the plane will be. Therefore, for twenty iterations the sum of the distance between the selected points is calculated, and the set of points with the greatest sum is used to calculate the gravity vector. The scene can now be properly oriented using the gravity vector. In this case, a properly oriented scene is one where the gravity vector is parallel to the z-axis.

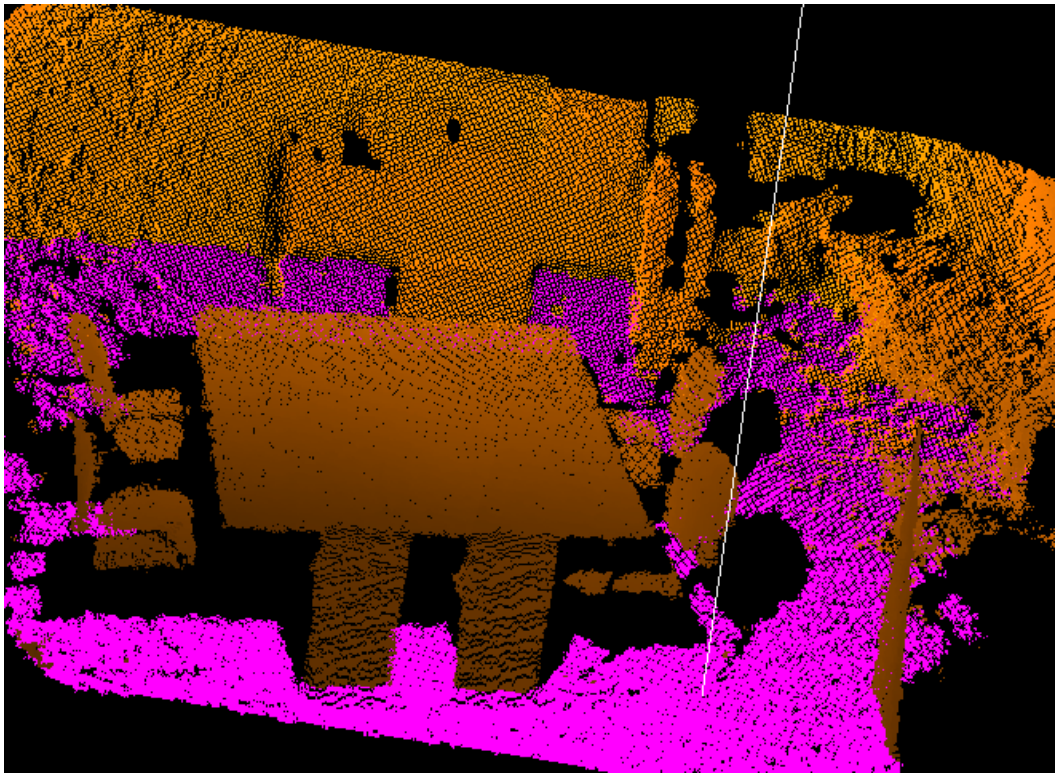

Figure 4.2. A scene's segmented floor and gravity vector

In order to achieve this rotation, two calculations are required. First, the gravity vector's y-value is set to zero and the angle between it and the z-axis is calculated, providing the required angle of rotation around the y-axis. Similarly, the following step involves setting the gravity vector's x-value to zero on order to determine the rotation angle needed for the x-axis. The resulting angles are used to precisely rotate the entire point cloud, resulting in a precisely oriented scene.

4.2 Table Detection

Once the floor has been identified and has been used to properly orient the point cloud, the knowledge of the scene's orientation can be taken advantage of in order to detect table's surfaces in the scene. The first assumption that can be made is that the surface of a standard table will always be parallel to the surface of the floor.

This assumption about a standard table's surface is based upon a concept presented in [25]. Here the author discusses the properties of supporting planes:

> "When the function of an object part is to support a potential other object, this part has to be parallel to the ground. A full three-dimensional segmentation based approach is not necessary when additional clues like object arrangement information is given by the user." [25] (p. 1484)

Based on the above definition, a table's surface can clearly be defined as a supporting plane. This allows us to follow the segmentation approach presented in [25] that projects a slice of a three-dimensional point cloud into two dimensions before segmentation. The method used to slice the point cloud in [25] accounts for the object's height. This leaves us with the question of what height to slice the point cloud.

In [31], the authors discuss the concept of function-based object recognition. The basis for their approach is that an object's structure is directly related to the function it is meant to fulfill. The main utility of a table is to offer functions to a human sitting in a chair and to support any object which they place on it.

The standard height of a table's surface is directly proportional to the average height of a sitting human. Table height can be calculated by first determining the average shoulder height of a sitting person. This can be calculated by first measuring the knee to the hip and the shoulder to the top of head. The sum of these measurements is subtracted from a person's height resulting in shoulder height. Proper table height can be derived directly from shoulder height by subtracting the length of the upper part of the arm. This essentially provides a height at which the elbows will rest while a person is seated. This height is also optimal for for grasping and interacting with objects supported by the table. Therefore, as seen in Figure 4.3, the average table height is 70 – 80 cm above the floor plane; a height which is comfortable for most sitting humans.

As discussed in Section 4, object recognition approaches that focus solely on matching a finite set of geometric models are constrained by the number of models in

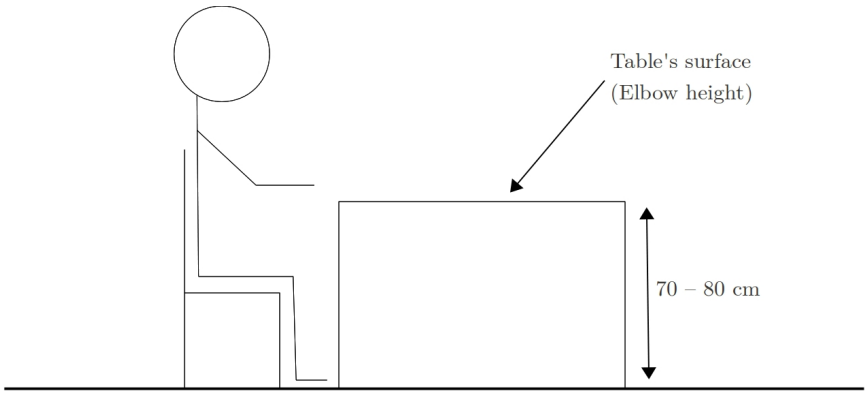

Figure 4.3. Function based recognition requires knowledge of an object's function

the set. This type of method generally utilizes a collection of CAD-like models that are specific in shape, while function-based approaches are able to detect general structures [31]. Although the data collected for this research only contained rectangular tables, a functional-based approach allows for the detection of tables of various shapes and sizes.

4.2.1 Methodology

The first step in detecting the surface of a table is to eliminate non-essential data. The average point cloud generated in this study consists of almost 300,000 points, the majority of which do not belong to a table's surface. In order to minimize the number of necessary calculations, knowledge of the scene's orientation is used to slice the point cloud using a range based on the average height of a table.

Due to the rotation that properly oriented the point cloud, every point belonging to the floor now has z-values at or very close to zero. This allows the point cloud to be sliced using each point's z-value. As previously mentioned, the average height of a table's surface is 70 – 80 cm. In order to ensure that the entire surface is included in the analysis, a buffer of 20 cm is used. Therefore, every point in the point cloud whose z-value falls within the region of interest (60 – 90 cm) is added to a new point cloud. This smaller point cloud, represented by the red points in Figure 4.4, now contains all points in the scene that belong to a table's surface and will allow for more efficient feature detection.

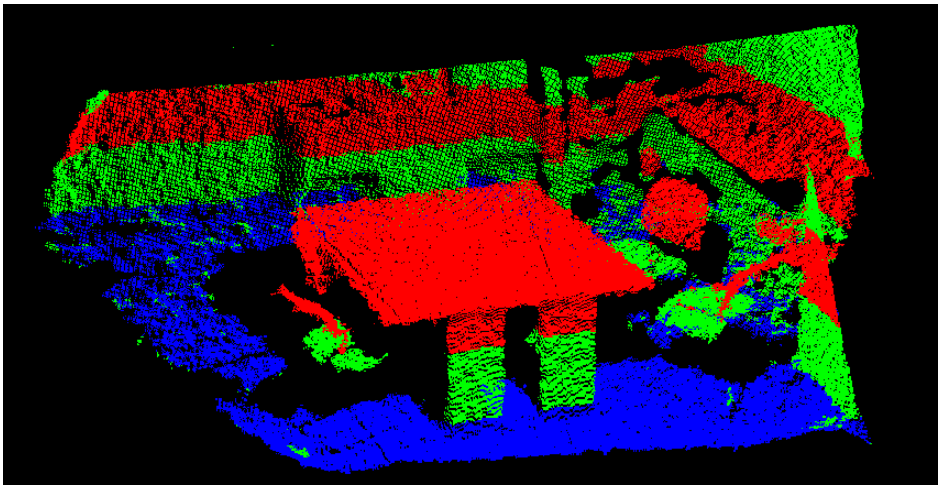

Figure 4.4. Point cloud sliced at table height

While working with 3-dimensional data may not *always* be extremely computationally intensive, it can be useful to reduce the dimensionality of data. This applies to most cases where one dimension is constant or irrelevant. Following the work done in [35], the sliced point cloud is projected into a two-dimensional plane. The two remaining values, x and y, now represent the portion of the scene where a table's surface would be located.

As seen in Figure 4.5, the 2-dimensional slice has distinct areas of clustered points that represent objects in the scene. These clustered points must be grouped together in order to perform a further analysis of each group's characteristics. Using a simple grouping algorithm, all connected pixels are assigned to the same blob.

The first step in the grouping algorithm[1] is to create a binary image, as seen in Figure 4.5. Pixels that have a corresponding point in the sliced point cloud are assigned a value of 1, while pixels that do not are assigned a 0. For every pixel with a 1 value, the grouping algorithm searches the eight surrounding pixels for 1 values. This process continues until no more 1 values are found. All of the connected pixels with 1 values are then given the same unique blob ID and are removed from the set of pixels being searched. Once all pixels with 1 values have been assigned a unique blob ID, the process terminates.

[1]The algorithm used in this process is based on the work of Andrew Greensted [13]. His algorithm can be found here.

Figure 4.5. Binary image created from the flattened point cloud

The result of this grouping algorithm is a set of blobs whose features can be analyzed in order to determine if they correspond to a table's surface.

In Figure 4.6, there are blobs of various shapes and sizes. Since we are interested only in blobs that could potentially represent tables, the first step is to filter out the blobs whose size is insufficient. A threshold of 500 pixels was determined based upon test data in which the size of the table is known. Eliminating blobs smaller than the threshold is important for two reasons: to reduce the number of unnecessary computations and to decrease the probability of false positives.

Before collecting a blob's feature vectors for classification, a cleaning process must be applied. The reason for this is illustrated in Figure 4.6; several objects, including the table are all part of one blob. In many of the scene configurations, other objects, such as chairs and people sitting around the table, are sometimes included as part of the table blob. This can lead to inconsistent values of the feature vectors used for classification. Therefore,

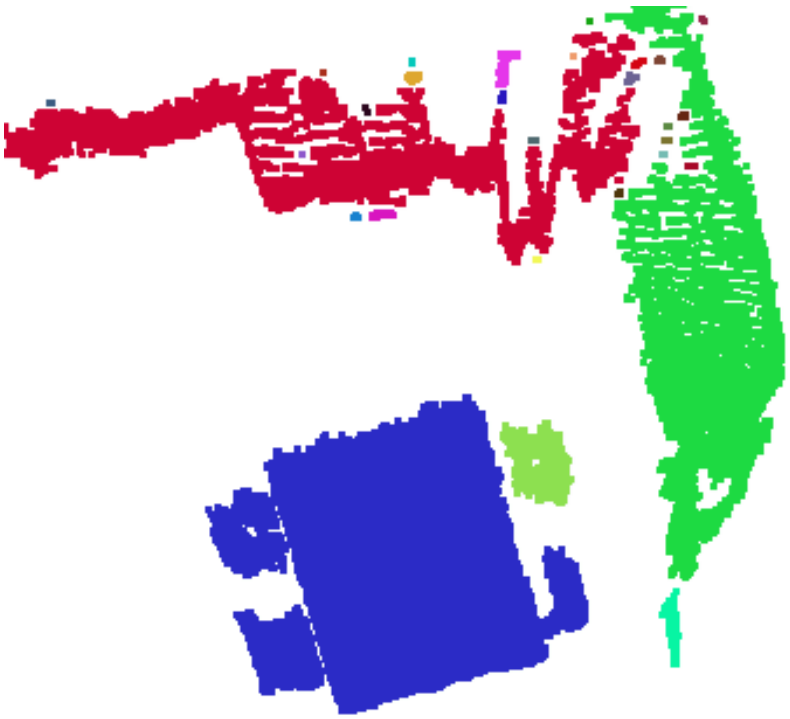

Figure 4.6. 2-D image of separated blobs

these features should be removed from the blob, leaving only the data that belongs to the table's surface. In order to achieve this goal, each blob is converted back to three dimensions using the blob's coordinates and the point cloud consisting of points at table height. Using the RANSAC algorithm, the largest plane in each three dimensional blob is found. All points belonging to the plane are then converted back to a two dimensional image. As shown in Figure 4.7, several blobs are most likely present in the resulting image. The blob detection algorithm is run on the image to isolate a potential table from any extraneous non-connected features.

The smallest blobs in the new image are filtered out and the remaining blob's features are used in the classification process.

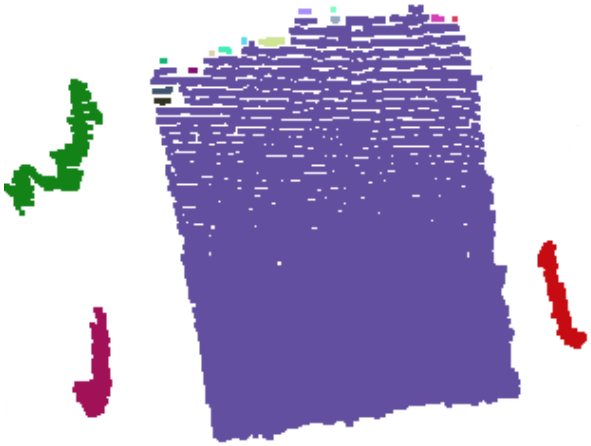

Figure 4.7. 2-D image of the table blob after cleaning

4.2.2 Classification

In order to reliably detect blobs that correspond to a table, a classifier must be constructed. Classification can be described as an algorithmic procedure that determines what class an observation belongs to. There are two types of classification: supervised and unsupervised. Unsupervised classification involves finding existing patterns in unlabeled observations with no prior knowledge of the observation's classes. In supervised classification, the classes are pre-defined by a training set. A training set is comprised of multiple observations whose feature vectors are known and are labeled as belonging to one of the pre-defined classes. A learning algorithm is then used with the training set to produce a function, the classifier, that can assign unlabeled observations to their proper class.

4.2.2.1 Classification Features

For the purposes of recognizing rectangular table shapes in two dimensional blobs, three numerical features are used to train the classifier: highest ratio of bounding box:area, uniformity of length, and uniformity of height.

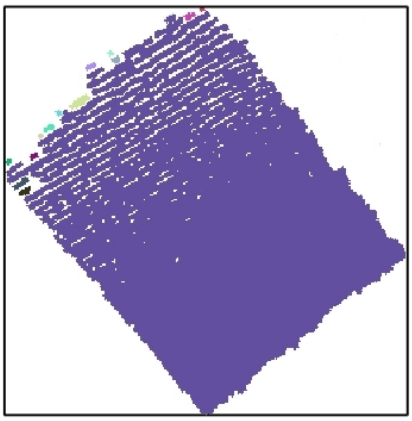

Figure 4.8. Bounding box of an unrotated blob

The initial bounding box of each blob is calculated using its highest and lowest x and y coordinates. As previously mentioned, blobs smaller than 500 pixels were automatically discarded since they are too small to be considered a table.

When the point cloud data is collected, there is a high probability that the table's surface is not square with the depth sensor. As seen in Figure 4.8, a table blob will most often require a rotation in order to determine its smallest bounding box. To efficiently accomplish this rotation, the coordinate system is transformed from a Cartesian to a Polar coordinate system.

The location of a point in a polar coordinate system is determined using two parameters: the distance from the origin r and an angle θ. Converting from the Cartesian coordinate system to the polar coordinate system involves two equations. First, $r = \sqrt{x^2 + y^2}$ which uses the Pythagorean theorem to determine the radial coordinate, while $\theta = arctan2(y, x)$ is used to calculate the angular coordinate.

Polar coordinates allow the blob to be rotated around the origin by simply increasing θ. Each blob is rotated in one degree increments for 90 iterations. The rotated shape is translated back to a Cartesian coordinate system and its bounding box:area ratio is calculated. The angle that produces the highest bounding box:area ratio is stored for later

use. The ratio also provides a metric for the overall rectangularity of a blob and is the first feature used by the classifier.

Figure 4.9. Bounding box of a rotated blob

Each blob is then rotated using the angle that produced the smallest bounding box in the previous step. This rotation ensures that if the blob is rectangular, it will be oriented so that two sides are parallel to the x-axis and two sides are orthogonal to the x-axis. This is essential for calculating the uniformity of the blob's length and width. While the relationship between the length and width of a blob is not an extremely useful metric, the straightness of each individual feature is. Although there may be some noise present in the data, a blob representing a rectangular table should have relatively straight sides.

Now that the blob is properly oriented, the first row consists of the all pixels with the lowest y-value. As seen in Figure 4.10 the distance between the highest and lowest x-values in this row is then calculated. This process continues until the distance of all rows in the blob have been recorded. In cases where the blob is perfectly rectangular, all values should be equivalent. However, due to the sensor's limitations, the values of a rectangular table will have some variation. In order to account for the sensor's limitations and occlusions due to scene configurations, the median value of the row distances is derived. Each row's width is then compared to the median value of the all rows. The percent of values that fall

within 10% of the median value is then calculated; this metric is used as the second feature in the classifier.

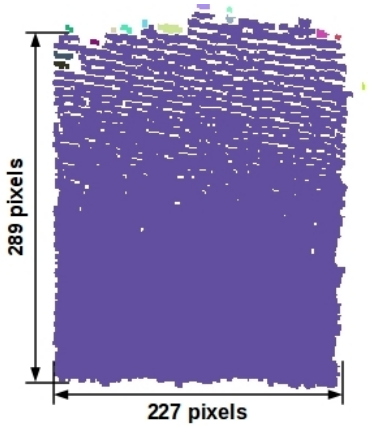

Figure 4.10. The length of every row and column are measured and recorded

This process is then repeated for every column in the blob; first using the row with the lowest x-value to measure the distance between the row's lowest and highest y-values. The output from this process is used as the third feature in the classifier.

4.2.2.2 Training Data Collection

The training set for the classifier was collected in uncluttered rooms containing large rectangular tables. The Kinect sensor was used to collect 2½-dimensional point clouds from several view points and object configurations. All tables were free of clutter and were surrounded by multiple chairs. The first configuration had all chairs pushed in and arranged uniformly around the table. This configuration, shown in Figure 4.11, is meant to simulate a situation where a user enters an empty room. Several point clouds were captured from various angles using this schema.

The second configuration involves arranging the chairs in a non-uniform manner. This layout mimics a realistic indoor space that people have interacted with. Several point clouds were also captured at various angles for this arrangement. The last schema consists of people occupying one to many chairs around the table.

Figure 4.11. Table configuration of an unused room

Figure 4.12. Table configuration after use

This layout is meant to depict a situation where a room is currently in use.

The position of the sensor used in this data collection process was restricted to vantage points that ensured a sitting person did not occlude a large portion of the table (e.g.

Figure 4.13. Table configuration while in use

being between the sensor and the table). Twelve samples were collected for each of the three types of chair configurations, producing a training set of 36 point clouds.

4.2.2.3 Bayesian Classification

A Bayes classifier is a probability model based on the Bayes theorem, a formula used for calculating conditional probabilities.

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)} \tag{4.7}$$

Since the theorem's popularization in the mid 20th century, it has been used in many fields including: medical diagnosis, genetics, sports modeling, artificial intelligence, and machine learning.

A Bayes classifier is considered a supervised learning algorithm because it requires a set of training data in which the feature vectors of a class and the class types are known. A key assumption of the naive Bayes classifier is that the presence of all features describing a

class are statistically independent of one another. However, the assumption that all features are independent of each other is often invalid [28].

The minimum size of a Bayes classifier's training set is directly related to the number of dimensions of the feature vector that the classifier uses. One of the advantages to using a naive Bayes classifier is the relatively small amount of training data required. For example, a non-naive Gaussian Bayes classifier that does not assume independence between features, minimally requires that the training set is $2^{(2d-1)}$, where d is the number of dimensions of the feature vector, while a naive Gaussian Bayes classifier's training set requires only $2d$ [24]. The Bayes classifier used in this thesis has a d value of 3; requiring a minimum of 32 training observations for a non-naive Gaussian Bayes and 6 observations for a naive Gaussian Bayes. Therefore, in cases where the amount of training data is limited, the naive Bayes classifier is often the better choice. However, in this case an abundance of training data is available; the 32 required training data observations were collected and used to build a Gaussian Bayes classifier.

The first step in constructing a Gaussian Bayes classifier is to identify the two sample sets that will serve as training data. S_1 is comprised of 16 observations that were derived from table blobs. Each observation consists of the three numerical features that were previously discussed. S_2 also contains 16 observations, however this set contains non-table blobs that were large enough to not be filtered out before classification. The numerical features required by the classifier are also calculated for the observations in S_2.

The prior probability $P(\omega_k)$ for each class must now be calculated. This can be determined by calculating the relative frequency of the number of observations in each class. Since the training set was constructed in a controlled fashion, the cardinality of both sets is equal, making the prior probability for both classes 0.5.

The next step in constructing the classifier involves a building a conditional density function for each class. The equation used in this process is:

$$p(\mathrm{x}|\omega_k) = \frac{1}{\sqrt{(2\pi)^d|\Sigma_k|}}\, e^{-(\mathrm{x}-\mu_k)^T\Sigma_k^{-1}(\mathrm{x}-\mu_k)/2} \tag{4.8}$$

In the above formula, d represents the number of dimensions and is, in this case, equal to 3, while x represents the feature vector that is to be classified. There are a number of processes that must be completed in order to solve for $p(\mathrm{x}|\omega_k)$. First, the mean vectors are calculated for each feature vector in both classes. Next, a covariance matrix is constructed from each class. Each covariance matrix is then used to find its inverse as well as its determinant. The resulting conditional density functions for ω_k will both be used to determine the posteriori probability through the Bayes Rule:

$$P(\omega_k|\mathrm{x}) = p(\mathrm{x}|\omega_k)\frac{P(\omega_k)}{\Sigma_{j=1}^{k_{max}} p(\mathrm{x}|\omega_j)P(\omega_j)} \tag{4.9}$$

Assuming equal costs for both types of error, the two resulting posteriori probabilities are compared and the highest value is used to determine if a blob is classified as a table.

4.2.2.4 SVM Classification

Support Vector Machines (SVMs) are supervised learning methods that have been recently introduced in [8]. SVMs currently have many applications including: biomedical research, financial applications, computer vision, and pattern recognition. While they can be used for both classification and regression, this section will focus solely on using SVMs for classification.

Classification is generally thought of as a function that can separate observations into two or more classes. For example, the Bayes classifier discussed in section 4.2.2.3 has two distinct classes: table and other object. The training data used to build the Bayes classifier consists of observations from table blobs as well as non-table blobs. A SVM

classifier is able to use training data from only the table blobs in order to determine which observations are outliers or non-tables. This is known as one or single class classification.

SVM classifiers rely on the the concept of decision planes to determine an observation's class. In certain situations, a simple linear classifier will sufficiently separate classes. However, in many classification tasks, the complexity of the data requires a non-linear function to optimally separate classes. A SVM must avoid the use of a non-linear decision boundary because of its reliance on hyperplanes. In order to generate an optimal hyperplane, the SVM transforms the data into a higher dimensional class using a kernel function. This is known as the kernel trick and allows for linear separation.

4.2.2.5 Building and Using the Classifiers

The Gaussian Bayes classifier used in this research was implemented in R statistical language using the standard packages. The SVM classifier was also implemented in the R language using the e1071 package. After testing several kernels using the 16 observations belonging to the table training set, it was determined that the classifier should use a radial basis function. The radial basis function is a non-linear SVM classifier based on a Gaussian distribution and is expressed as: $K(\vec{x}, \vec{z}) = e^{-(\vec{x}-\vec{z})^2/(2\sigma^2)}$ [23]. The final step in implementing the SVM classifier involved tuning its parameters. This was accomplished through a combination of cross validation and a tuning function provided in the e1071 library. The final SVM classifier is comprised of 5 support vectors that were generated from the 16 observations classified as tables in the training data set. The initial training set consisted of only three numerical features; the other two were added by the SVM to allow for linear classification.

While the Bayes and SVM classifiers were both implemented by the table detection program, we determined that the Bayes classifier was the optimal choice. Although the SVM was tuned specifically for the training data, it was very rigid with its classification;

when testing its performance it failed to properly classify approximately 12% of the tables, while the Bayes classifier correctly identified all tables.

Chapter 5

CONCLUSIONS AND FUTURE WORK

This chapter presents an overview of the research presented in this thesis. Section 5.1 summarizes our findings and provides an overview of the table detection algorithms that were developed. There are several opportunities to expand on and make use of the table detection algorithms. These potential additions and future uses of this research are discussed in Section 5.2.

5.1 Summary

This thesis presents a set of algorithms that were developed for the robust detection of tables in office-like indoor environments. Chapter 2 introduces the concepts of active and passive depth sensing. Several active depth sensing methods are discussed, including coded structured light, the method employed in the data collection process. A more detailed description the Kinect sensor's functionality and accuracy is provided in Section 2.1.3.

The algorithms used in the table detection process are highly reliant on the detection of planar surfaces. Section 2.2 discusses various approaches to this task. After comparing several plane detection algorithms, we determine that RANSAC is the best candidate due to its efficiency and robustness.

The two experiments presented in Chapter 3 indicate that tables play a central role in the description of indoor environments. One experiment involved asking the subjects to rank objects based on their importance to the function of an office-like room. The other experiment recorded the verbal descriptions provided by the subjects of a different office environment. The number of mentions for each type of object were then calculated. Tables were ranked as the most important objects in both experiments, demonstrating their importance in indoor scene descriptions.

Chapter 4 describes the development and functionality of the floor and table detection process. The algorithms discussed in this chapter are function-based, meaning the functional properties of an object are the basis for its classification. The first step in this process is to properly orient the point cloud. This is accomplished through the detection of the scene's floor. Utilizing the planar surface that represents the floor, the scene's gravity vector is calculated and is used to rotate the scene to its proper orientation.

The properly orientated scene is then sliced based on the standard table height. The sliced area is projected to a two dimensional plane by removing the height component. The resulting data is used to create a two dimensional image which is used as input for an iterative labeling procedure that separates connected regions into independent blobs. Each blob's size is calculated and values below the size threshold are filtered out. Sufficiently large blobs' attributes are then calculated for use in the classification process.

Point clouds of several indoor scenes were collected as the training set for the supervised classification process. Bayes and Support Vector Machine classifiers were then constructed using the R statistical language. During the testing phase, the Bayes classifier consistently outperformed the SVM. A cost function was also taken into account; it is much costlier to fail to recognize a table than to falsely detect one. Therefore, we determined that the Bayes classifier is the optimal classifier for this process.

5.2 Future Work

One of the initial steps in utilizing depth sensors to produce meaningful spatial information about the configuration of indoor scenes is the detection of salient objects. These objects, such as tables in office-like environments, can be referenced in order to efficiently extrapolate spatial information about other objects in the scene. The table detection algorithms developed in this thesis could be used in conjunction with a spatial reference system that stores relationships between objects.

For example, chairs and tables have a very distinct relationship; a chair's primary function to support a sitting human, and a table is designed to be functional for a human sitting in a chair. As discussed in Section 3.2, this relationship can be incorporated into an efficient and robust algorithm for chair detection.

The floor detection algorithm developed in this thesis provides a function for properly orienting a point cloud of an indoor scene. This presents the opportunity to build an occupancy grid of the scene. An occupancy grid essentially determines which areas of the scene contain objects and which areas do not. The grid can be constructed by rotating previously oriented scene by 90°; this achieves a precise bird's eye view of the scene. Areas of the floor with no objects can be considered navigable and unoccupied, while occupied areas can be further analyzed by object detection algorithms.

REFERENCES

[1] M.D. Adams. Coaxial range measurement - current trends for mobile robotic applications. *Sensors Journal, IEEE*, 2(1):2 –13, 2002.

[2] Tarek M. Awwad, Qing Zhu, Zhiqiang Du, and Yeting Zhang. An improved segmentation approach for planar surfaces from unstructured 3d point clouds. *The Photogrammetric Record*, 25(129):5–23, 2010.

[3] Dorit Borrmann, Jan Elseberg, Kai Lingemann, and Andreas Nchter. The 3d hough transform for plane detection in point clouds: A review and a new accumulator design. *3D Research*, 2:1–13, 2011.

[4] F. Bretar and M. Roux. Extraction of 3d planar primitives from raw airborne laser data: a normal driven ransac approach. In *Proceedings of the IAPR Conference on Machine Vision Applications*, 2005.

[5] Council Canada and Blais F. Review of 20 years of range sensor development. 2004.

[6] P. Checchin, L. Trassoudaine, and J. Alizon. Segmentation of range images into planar regions. In *3-D Digital Imaging and Modeling, 1997. Proceedings., International Conference on Recent Advances in*, pages 156 –163, 1997.

[7] S.Y. Chen, Y.F. Li, and Jianwei Zhang. Vision processing for realtime 3-d data acquisition based on coded structured light. *Image Processing, IEEE Transactions on*, 17(2):167 –176, 2008.

[8] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

[9] Yihong Ding, Xijian Ping, Min Hu, and Dan Wang. Range image segmentation using randomized hough transform. In *Information, Communications and Signal Processing, 2003 and the Fourth Pacific Rim Conference on Multimedia. Proceedings of the 2003 Joint Conference of the Fourth International Conference on*, volume 2, pages 807 – 811 vol.2, 2003.

[10] Richard O. Duda and Peter E. Hart. Use of the hough transformation to detect lines and curves in pictures. *Commun. ACM*, 15:11–15, January 1972.

[11] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24:381–395, June 1981.

[12] S.B. Gokturk, H. Yalcin, and C. Bamji. A time-of-flight depth sensor - system description, issues and solutions. In *Conference on Computer Vision and Pattern Recognition Workshop, 2004.*, page 35, 2004.

[13] Andrew Greensted. Blob detection - the lab book pages. `http://www.labbookpages.co.uk/software/imgProc/blobDetection.html`. Accessed: 03/13/2012.

[14] M. Hebert. Active and passive range sensing for robotics. In *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on*, volume 1, pages 102 –110, 2000.

[15] D. Holz, S. Holzer, R.B. Rusu, and S. Behnke. Real-time plane segmentation using rgb-d cameras. 2011.

[16] R. C. Jain and A. K. Jain. Report on range image understanding workshop. In *Machine Vision and Applications*, 1988.

[17] X.Y. Jiang and H. Bunke. Fast segmentation of range images into planar regions by scan line grouping. In *Machine Vision and Applications*, pages 115–122, 1994.

[18] Saranya Kesavan and Nicholas Giudice. Automated natural language description of indoor spaces. In *Doctoral Colloquium of the 10th international conference on Spatial information theory*, COSIT, 2011.

[19] K. Khoshelham. Accuracy analysis of kinect depth data. In *ISPRS workshop laser scanning 2011*, page 6. International Society for Photogrammetry and Remote Sensing, 2011.

[20] R. Krishnapuram and C.-P. Freg. Fuzzy algorithms to find linear and planar clusters and their applications. In *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR '91., IEEE Computer Society Conference on*, pages 426 –431, 1991.

[21] Stephen C. Levinson. Frames of reference and molyneux's question: Crosslinguistic evidence. 1996.

[22] V Gui M Otesteanu. 3d image sensors, an overview. *WSEAS Transactions on Electronics*, Vol 5:53–56, 2008.

[23] C.D. Manning, P. Raghavan, and H. Schutze. *Introduction to information retrieval*, volume 1. Cambridge University Press Cambridge, 2008.

[24] Tom M. Mitchell. Machine learning. McGraw Hill series in computer science, 1997.

[25] Reinhard Moratz. Intuitive linguistic joint object reference in human-robot interaction: human spatial reference systems and function-based categorisation for symbol grounding. In *proceedings of the 21st national conference on Artificial intelligence - Volume 2*, pages 1483–1488. AAAI Press, 2006.

[26] K. Okada, S. Kagami, M. Inaba, and H. Inoue. Plane segment finder: algorithm, implementation and applications. In *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, volume 2, pages 2120 – 2125 vol.2, 2001.

[27] J. Pages, J. Salvi, R. Garcia, and C. Matabosch. Overview of coded light projection techniques for automatic 3d profiling. In *Robotics and Automation, IEEE International Conference on*, volume 1, pages 133 – 138, 2003.

[28] Irina Rish. An empirical study of the naive bayes classifier. In *IJCAI-01 workshop on "Empirical Methods in AI"*, 2001.

[29] D. Scharstein and R. Szeliski. High-accuracy stereo depth maps using structured light. In *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, volume 1, pages 195–202, 2003.

[30] L. Stark and K. Bowyer. Achieving generalized object recognition through reasoning about association of function to structure. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 13(10):1097–1104, 1991.

[31] Louise Stark and Kevin W. Bowyer. Function-based object recognition. In *Modelling and Planning for Sensor Based Intelligent Robot Systems*, pages 272–288, 1994.

[32] Eckehard Steinbach, Matthias Kranz, Werner Meier, Florian Schweiger, Nicolas Alt, Anas Al-Nuaimi, and Clemens Schuwerk, editors. *The Kinect Sensor Platform*, volume 2 of *Advances in Media Technology*, July 2011.

[33] Thora Tenbrink and Reinhard Moratz. Group-based spatial reference in linguistic human-robot interaction. *Spatial Cognition and Computation*, 6:63–106, 2003.

[34] Roland Wahl, Michael Guthe, and Reinhard Klein. Identifying planes in point-clouds for efficient hybrid rendering. In *In The 13th Pacific Conference on Computer Graphics and Applications*, pages 1–8, 2005.

[35] Michael Wünstel and Reinhard Moratz. Automatic object recognition within an office environment. In *CRV*, pages 104–109, 2004.

APPENDIX

ALGORITHMS

The algorithms used for floor and table detection are presented below.

DEFINITIONS

P - collection of three dimensional points

p - single three dimensional point

M - planar model

n - the minimum number of data required to fit the model (3)

k - the number of iterations performed by the algorithm

t - threshold value for determining when a point belongs to a plane

h - height of point cloud

N - normal vector

r - range

R_ω - three dimensional rotation matrix about the ω axis

I - two dimensional image

Algorithm 1 Floor Detection

for all p in P_{total} **do**
  Rotate p around x-axis by 30°
end for
for all p in P_{total} **do**
  if $p_i < .15h$ **then**
    add p_i to P_{slice}
  end if
end for
while count $< k$ **do**
  Select three random points (S_i) from P_{slice}
  Generate plane equation Pl using S_i
  for all p in P_{slice} **do**
    Calculate d_{min} of p_i to M
    if $d <$ threshold **then**
      add p_i to consensusSet
    end if
  end for
  if newCounter $>$ highestCounter **then**
    highestCounter = newCounter
    M_{best} = M_i
  end if
  Increment count
end while
return M_{best}

Algorithm 2 Table Detection

calculate N of M_{best}
calculate R_x and R_z to make N parallel to $(0, 1, 0)$
for all p in P_{total} **do**
    use R_x and R_z to rotate p_i
end for
for all p in P **do**
    if p within $r_{tableheight}$ **then**
        add p to P_{table}
    end if
end for
for all p in P_{table} **do**
    remove z value
end for
use P_{table} to create I
detect and separate blobs in I
for all blobs **do**
    calculate size
    if size $< t_{size}$ **then**
        discard blob
    end if
end for
for all blobs **do**
    calculate bounding box:size ratio
    for $i = 0; i < 90i++$ **do**
        rotate blob by i°
        calculate bounding box:size ratio
        if $ratio_{new} > ratio_{old}$ **then**
            $angle_{best} = i$
        end if
    end for
    rotate blob by i degrees
    calculate horizontal straightness
    calculate vertical straightness
    classify blob using Bayes classifier
end for

BIOGRAPHY OF THE AUTHOR

Brendan O'Shaughnessy was born in Pittsfield, MA on November 16th, 1981. He graduated from Hollis/Brookline High School in 2000. In 2006, Brendan graduated from Plymouth State University with a B.S. in Environmental Planning. After completing an internship at the Nashua Regional Planning Commission, he was offered a full time position as a GIS Planner. Brendan worked at the Nashua Regional Planning Commission for three years before deciding to continue his education at the University of Maine. He entered the Spatial Information Science and Engineering program in 2010.

Brendan O'Shaughnessy is a candidate for a Master of Science degree in Spatial Information Science and Engineering for May, 2012.