USING THE XBOX KINECT TO DETECT FEATURES OF THE FLOOR SURFACE

By

STEPHANIE COCKRELL

Submitted in partial fulfillment of the requirements

For the degree of Master of Science

Thesis Advisor: Greg Lee

Department of Electrical Engineering and Computer Science

CASE WESTERN RESERVE UNIVERSITY

January, 2013

Committee approval sheet

Table of Contents

List of Tables

List of Figures

Using the Xbox Kinect to detect features of the floor surface

Abstract

By Stephanie Cockrell

This thesis examines the effectiveness of the Xbox Kinect as a sensor to identify features of the navigation surface in front of a smart wheelchair robot. Recent advances in mobile robotics have brought about the development of smart wheelchairs to assist disabled people, allowing them to be more independent. Because these robots have a human occupant, safety is the highest priority, and the robot must be able to detect hazards like holes, stairs, or obstacles. Furthermore, wheelchairs often need to locate and navigate on ramps. The results demonstrate how data from the Kinect can be processed to effectively find these features, which will help to keep the occupant safe and allow for a smooth ride.

Chapter 1: Introduction

This thesis presents an algorithm which uses point cloud data from the Xbox Kinect, a product created by Microsoft Corporation, Redmond, WA, in order to identify features of the floor surface in front of a robot. Specifically, the results of this research can be applied to smart wheelchair robots. As the field of mobile robotics advances, it is becoming increasingly important to have different types of sensors, which provide data to process and help the robot run more safely and efficiently. In particular, the sensors must be able to detect obstacles and areas unsafe for navigation, so the robot can avoid collisions and traveling on unsafe terrain.

Smart wheelchairs are robotic wheelchairs which assist disabled people who could not otherwise operate a wheelchair, enabling them to be more independent and have a high quality of life. Since the safety of the human user is the highest priority, a smart wheelchair must have sensors which can detect many different types of obstacles and surfaces (such as small objects on the ground, ramps, stairs, and uneven ground) and determine whether or not they are safe to drive over.

Previous work in robotics has been focused on finding large obstacles to avoid, but not looking at the ground surface itself. That approach is very useful for autonomous vehicles traveling on streets that are presumed navigable, but it is not adequate for a wheelchair robot. The smart wheelchair must be able to detect ramps and stairs, which are very important in the task of navigating safely from one point to another, and must be aware of any small objects, bumps, and cracks in the floor, so it can ensure a smooth ride for the wheelchair operator. Detecting these irregularities in the floor surface is a huge problem which must be addressed in order to create wheelchairs which ensure the safety of the user.

This thesis explores a possible solution which uses an Xbox Kinect, mounted on the front or back of the robot and pointed down toward the floor. The Kinect is a sensor which uses an infrared depth camera to return an array of three-dimensional points, representing the objects that are in front of the camera. The Kinect itself is a comparatively cheap, ubiquitous sensor which gives a large amount of reliable data whose potential for use in robotics is being realized.

When data from the Kinect is processed, the robot can determine the slope of the floor itself and the heights of any objects that may be on the floor. Some objects (like power cords) may be small enough that running over the object poses no danger, while larger objects must be avoided. The algorithm presented in this thesis automates the decision. Also, the Kinect sensor will enable the robot to detect holes in the floor and determine whether they are large enough to pose a danger to the wheelchair; for example, the robot may use the Kinect data to conclude that it is at the top of a flight of stairs, and the robot can then navigate safely away from the stairs.

The results of this thesis suggest that the Xbox Kinect is an excellent solution to the problem of detecting drivable surfaces. The Kinect provides detailed and accurate data which can reliably detect unpassable terrain. Detecting stairs, ramps, and small objects is a problem that other sensors have not been able to adequately address, but the Kinect performs effectively at this task.

This thesis is divided into several parts. Chapter 2 examines work that other researchers have done in the areas of obstacle detection, smart wheelchairs, drivable surfaces, and the Xbox Kinect. Chapter 3 focuses on characterizing the Kinect and summarizes its strengths and limitations so that it can then be utilized in the best way possible. In chapter 4, an algorithm to calibrate the Kinect is shown. In chapter 5, the Kinect data was analyzed to classify areas according to their drivability. Finally, chapter 6 gives the conclusion and discusses the meaning of these results.

Chapter 2: Literature Review

Mobile robots have thousands of applications, from autonomously mapping out a lawn and cutting grass [1] to urban search and rescue [2] to autonomous ground vehicles [3]. One important application concerns “smart wheelchairs”, which would give wheelchair-bound people (with limited ability to drive) more independence and have safety features. For any autonomous robot, obstacle avoidance is extremely important, and it is especially important when there is a human user, as is the case for a smart wheelchair. Currently, the sensor systems used for obstacle detection have many weaknesses, such as cost and the type of data generated. Recently, Microsoft released the Xbox Kinect, and it proves useful for detecting objects or obstacles in front of a robot.

Smart wheelchairs and obstacle detection

Many smart wheelchair robots use ultrasonic or sonar sensors to gather information about the robot’s surroundings and build a map [4] [5] [6] [7] [8]. These sensors are useful because they are small and cheap. The data they collect serves a specific purpose which does not address the needs of this application. Often they would be placed near the base of the robot, and cannot see tables. Also, this type of sensor would not give information about the floor surface itself because they return only limited depth information. Using the Kinect is an improvement over sonar or ultrasonic sensors because the Kinect can provide three-dimensional depth data to help identify the floor itself.

In other cases with smart wheelchairs, a laser rangefinder was used to detect obstacles in the environment [9] [10]. The laser rangefinder is reliable and gives higher resolution information than a sonar sensor, so it can effectively detect large obstacles. However, these sensors are usually large and very expensive. Also, a typical Sick Lidar unit only gives data for one “slice” of the robot’s environment; it would not be able to see small objects on the floor, or table tops.

In addition, it is common for mobile robots to use cameras and image processing for obstacle detection. Often this is done through image segmentation, edge detection, and other methods. For example, researchers have used structured light to find irregularities corresponding to objects in front of the robot [11]. Light is projected on the ground in front of the robot. Distortions in the pattern of light seen by the camera are attributed to obstacles. The Kinect itself actually uses this method. Another image processing technique is optical flow, which takes two consecutive images and compares them to account for the movement of the camera [12]. It is able to identify objects by looking at the difference between the optical flow and planar flow in the images.

Also, some mobile robots make use of stereo cameras [13] [14]. By comparing images from multiple cameras and using image processing techniques, objects in front of the robot are identified. This method is dependent on camera separation and distance to objects, and it may be unable to identify some critical features.

Using the Kinect for obstacle detection has advantages over using cameras and image processing. The Kinect directly provides 3-dimensional information, requiring no processing to obtain depth data.

Kinect

The Microsoft Xbox 360 Kinect was released in 2010, to be used as a sensor or controller for video games [15]. It is revolutionary because, unlike traditional video game controllers which receive user input through buttons and joysticks, the Kinect is able to identify people in a room and track their movements and gestures [16].

In the two years immediately following the release of the Kinect, robotics researchers have investigated its effectiveness as a sensor. The Kinect's accuracy and limitations were explored [17]. There are already many applications for the Kinect. It can be used to detect and distinguish between different kinds of objects [18]. In another example, the depth information was analyzed to identify the different parts of fingers or hands, in order to interpret gestures from a human operator [19]. In the field of mobile robots specifically, the Kinect was found to be an effective tool for target tracking and navigation [20]. It was even found to be useful for localization and mapping for a robotic wheelchair [21].

Drivable surfaces

One particular concern for smart wheelchairs is an ability to identify not only obstacles but any problems or unusual characteristics with the driving surface. For example, there may be a hole in the floor, or a flight of stairs, and it is very important that the wheelchair can detect this to keep the operator safe. Additionally, the wheelchair robot should be able to identify ramps, which are different from the regular floor surface, but are not like other obstacles; ramps can be driven on (and in many cases this is required). Also, ideally such a system would be able to differentiate between a traversable object and non-traversable or dangerous obstacles, like stairs.

Many autonomous robots use image processing techniques to identify the road surface, which is assumed to be traversable. (Other sensors identify large obstacles, like cars.) A camera is mounted on the robot, and the video feed is analyzed to determine which part of the image represents a drivable surface [22]. In the case of robotic cars, the lines painted on the road are detected; the traffic lanes are located and a "drivable surface" is defined to be a lane. However, a smart wheelchair robot uses a very different definition of “drivable surface.” Wheelchairs operate on surfaces not specifically designed for wheeled vehicles. Drivability cannot be determined by just looking for certain lines on the ground. If a camera is to be used, additional object-detection methods must be employed.

One issue common with a single camera is a lack of depth perception, which makes it hard to identify whether differences in color or texture represent an obstacle or a shadow. Some of the methods involve examining the image histogram, coupled with feature detection [23] or Canny and Hough transforms [24]. Progress has also been made using image segmentation techniques and logical operations [25]. Additionally, there are techniques based on using a stereo camera to get a 3D representation of the road surface, and further processing that information along with the image data [26].

However, these methods are not sufficient for ensuring the safety of the wheelchair operator. They can detect the road surface itself, but are unable to identify large bumps in the road, particularly if the bump is the same color and texture as the rest of the road surface. Also, image processing will not give information about the slope of the road surface. The Kinect addresses this need by returning a three-dimensional point cloud with depth information.

Uneven surfaces and ramps

The main focus of this project is detecting uneven surfaces. Wheelchairs must avoid them or traverse cautiously over this type of terrain. In the past, uneven or sloped surfaces were known to be a problem because as a robot drives over it, the point-of-view of the robot’s sensors is tilted up or down, causing inaccurate readings [27]. Also, in the case of a robot with legs, sensors can be placed on the bottom of the robot's foot to detect the slope of the floor and adjust the foot position accordingly [28]. The wheelchair robot, however, must sense these uneven areas of the floor before driving over them.

In many cases where irregularities in the road surface are a concern, autonomous vehicles use sensors that give information about depth, rather than using a simple camera. For example, time-of-flight cameras [29] and Sick Lidar units are used. Typically the lidar is pointed down toward the road at an angle, allowing it to see the road surface and determine the location of a curb or other obstacles [30]. Furthermore, the height information can be subtracted in order to find the gradients of the surrounding area [31] [32] [33]. This is useful for identifying ramps or sloped surfaces and distinguishing them from actual obstacles; this method of calculating gradients will be expanded upon in this thesis.

There are also several approaches which use point cloud data similar to the data output by the Kinect, to find ramps and irregularities in the ground surface. One example used a time-of-flight camera to assist in seeing curbs and ramps for a car backing up. A variation on the RANSAC algorithm (an approach that is often used to fit planes to data containing outliers) was used to fit planes to the point cloud data and locate the road surface and any other surfaces [34]. However, this approach would not be adequate for the smart wheelchair; there may be some small holes or bumps which present a risk to the wheelchair but are too small or irregularly-shaped to be found using a plane fitting.

Another plane-fitting approach uses a patch of data points around the point being evaluated, and calculates the slope and roughness of the best-fit plane for that patch. Next, the values for slope and roughness are used to determine traversability, and this process is repeated for each data point in the point cloud [35]. However, this algorithm is computationally intensive, and the "roughness" calculation does not quite address the needs of the wheelchair; the wheelchair robot needs to know whether the floor is "rough" because of a large obstacle, small bump, or just a bumpy texture on the floor itself.

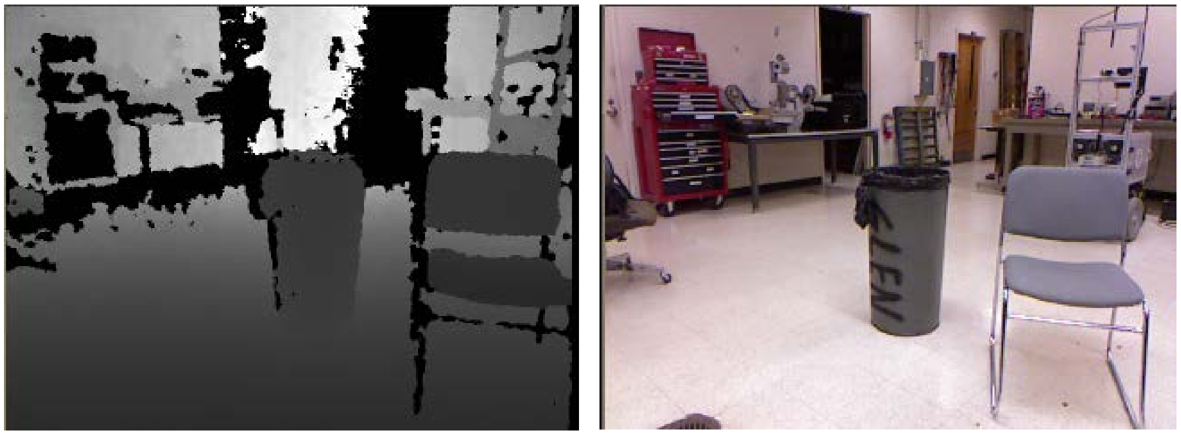
Other algorithms calculated the curvature of different parts of the point cloud. The point cloud could be segmented at the points where there was a change in the curvature [36] [37]. However, a wheelchair robot must identify the slope of a ramp, not a sudden change in curvature.

In another example involving point cloud data, the points were connected into line segments if the vertical distance between them was small enough. The line segments were then joined into surfaces, and the surfaces could then be classified as bare earth, detached objects (like buildings and plants), and attached objects (like ramps and bridges) [38]. However, this method is very computationally intensive and does not address the needs of a wheelchair. This algorithm would categorize a large region as "bare earth" if all the points were connected, but the wheelchair robot needs to know the slope in order to decide whether it is safe to drive there.

The previous work summarized here is effective for certain tasks, but it does not adequately address the needs of a smart wheelchair, particularly the need to detect drivable surfaces. This thesis explores the effectiveness of the Kinect in this particular application.

Chapter 3:Characterization of the Kinect

The Kinect provides high-resolution data about the environment in front of it. It has both a normal RGB camera and a depth camera. Figure 1 shows a sample output from the two cameras:



b) Depth image **a) RGB image**

Figure 1: Sample output from the Kinect's depth camera (left) and RGB camera (right). In the depth image, darker objects are close to the Kinect sensor, and lighter objects are far away.

In the depth image on the left of Figure 1, each pixel in the 640x480 image contains a value which indicates the distance from the Kinect, in meters. Any regions where no data was received are marked as 0 and displayed in black in the above image. The depth data can also be given in the form of a point cloud, which is a 640x480 array of cells, with each cell containing the (x,y,z) coordinate for that particular point. This form of data will be analyzed in this project.

Software and hardware

The research presented here builds on previous work using ROS for mobile robotics [39] [40] [41]. ROS is a robotics software framework, useful for developing robotics applications. It allows different programs to access sensor data, process it, and publish the result for easy access by other programs. Several applications can be run at the same time, and data can be passed between them. For this thesis, ROS was used to view a live feed of RGB and depth images coming from the Kinect, and to pass data from the calibration code to the rest of the code. In future applications, ROS also provides tools for path planning compatible with the result of this thesis.

Additionally, OpenNI, an open-source driver, was used to handle the data received from the Kinect and publish it in a form that could be used by ROS.

The code for this project was written in C++ and run on a Lenovo ThinkPad laptop with an Intel T7300 processor and clock speed of 2GHz. The Kinect itself was mounted on a mobile robot for most of the data collection.

Distance limitations

In order to determine the potential applications of the Kinect, it is necessary to characterize it. First, the accuracy of data at different distances is measured. The Kinect cannot provide accurate data on objects which are less than 60 cm from the unit itself. Figures 2 and 3 demonstrate this limitation:

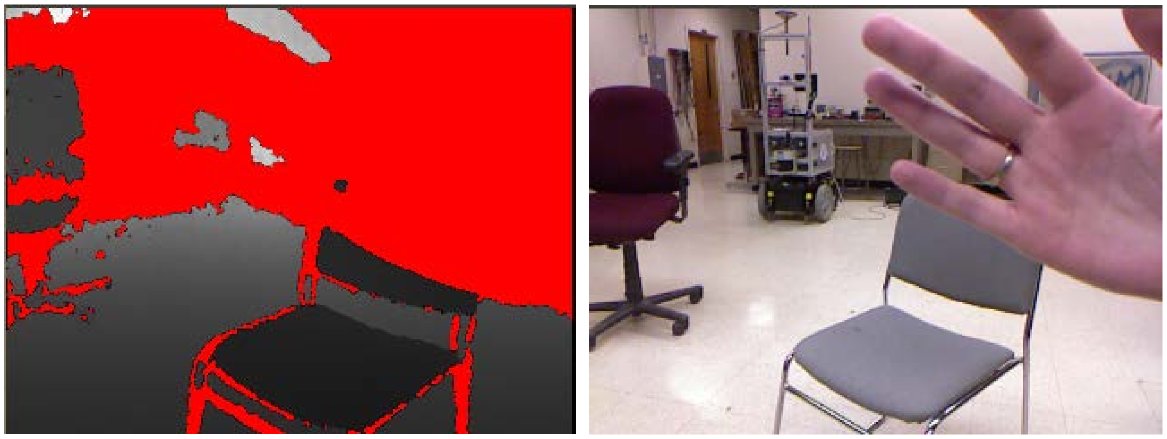


Figure 2: The hand is 20 cm away from the Kinect. Areas with no data have been marked red on the depth image.

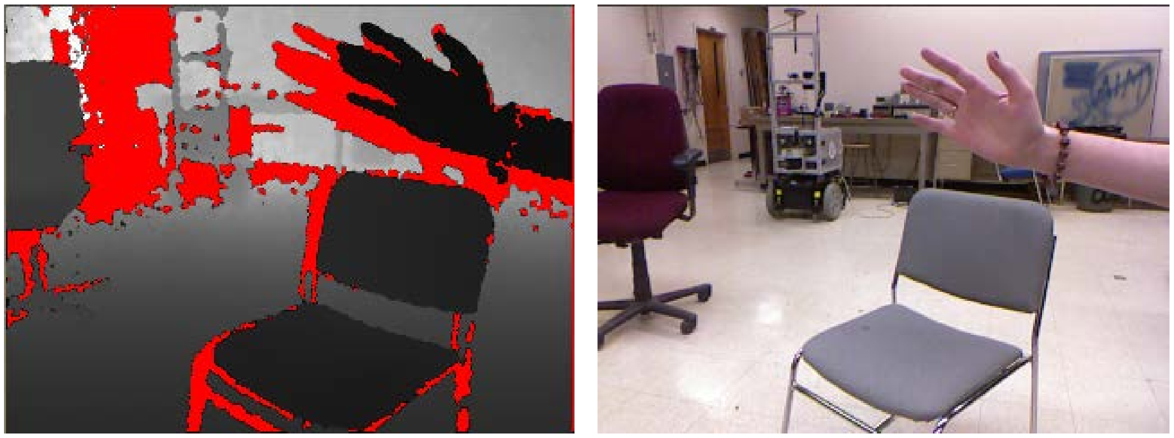


Figure 3: The hand is 60 cm away from the Kinect. Areas with no data have been marked red on the depth image.

These images show that at a distance of 60 cm, the Kinect is able to detect objects and return data about the depth. The hand in Figure 3 is shown in dark gray on the depth map, which means there is data for that area. In Figure 2, the hand shows up red on the depth map, which means the Kinect was unable to collect depth information for that area.

Because of the "blind spot" close to the Kinect, the sensor must be placed such that no objects are expected to come within 60 cm of the Kinect. It is expected that the Kinect can detect any obstacles before they get too close.

Data was also collected to determine the upper limit of the Kinect's range. The following image shows the output from both cameras when the Kinect is in a long hallway:

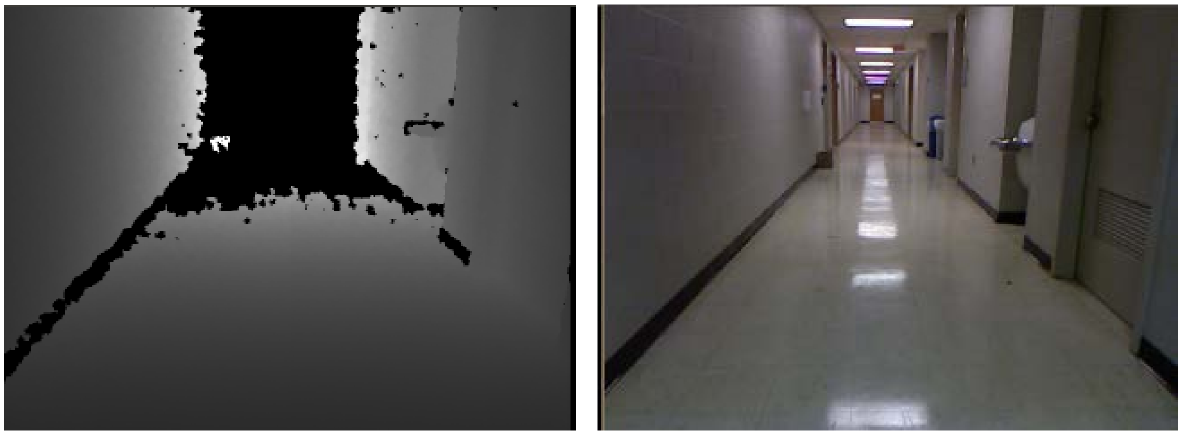


Figure 4: Depth image (left) and RGB image (right) in a long hallway, showing the Kinect has a limitation in seeing long distances.

As Figure 4 shows, everything past a certain distance gets marked as black, indicating that it is out of range. This begins to happen about 6 meters from the Kinect. Beyond 6 meters, the depth camera may be able to see some objects, but not the floor itself, because of the shallow angle of the floor surface. Therefore, the placement of the Kinect on the robot must be such that the field of view does not include regions so far away.

Distortion

Measurements of a known flat surface (wide open floor) were taken. Figures 5, 6, and 7 examine this data to expose warping effects. First, the raw data was plotted in blue. A best-fit plane was fit to this set of data; it is shown in red. All three images are the same set of data, but viewed from different angles.

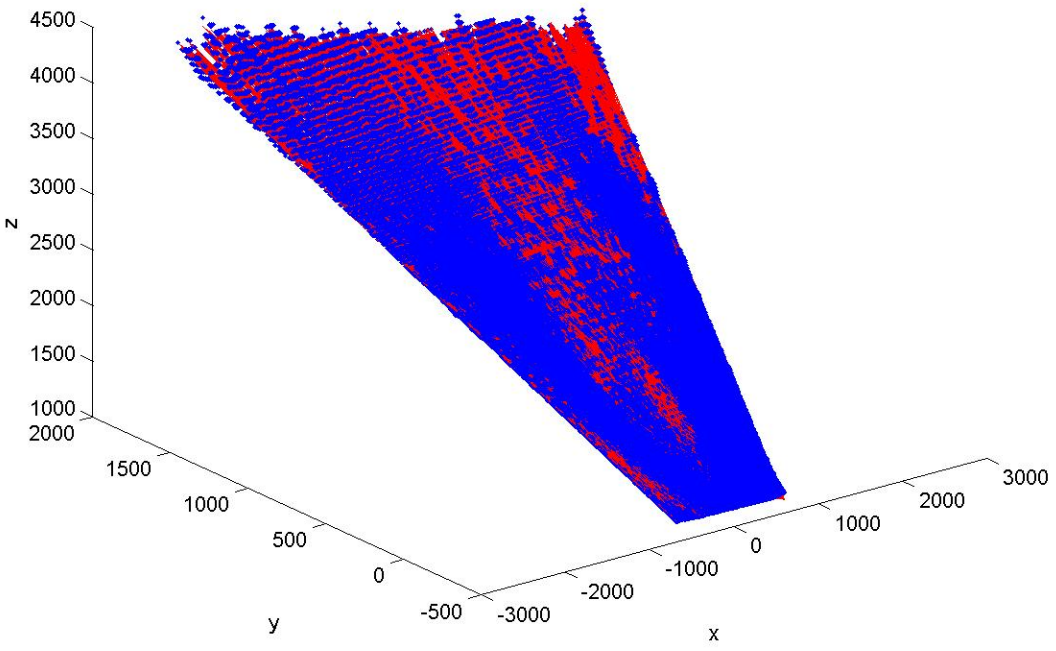


Figure 5: The blue points are raw data; the red ones show the best-fit plane. Units are mm.

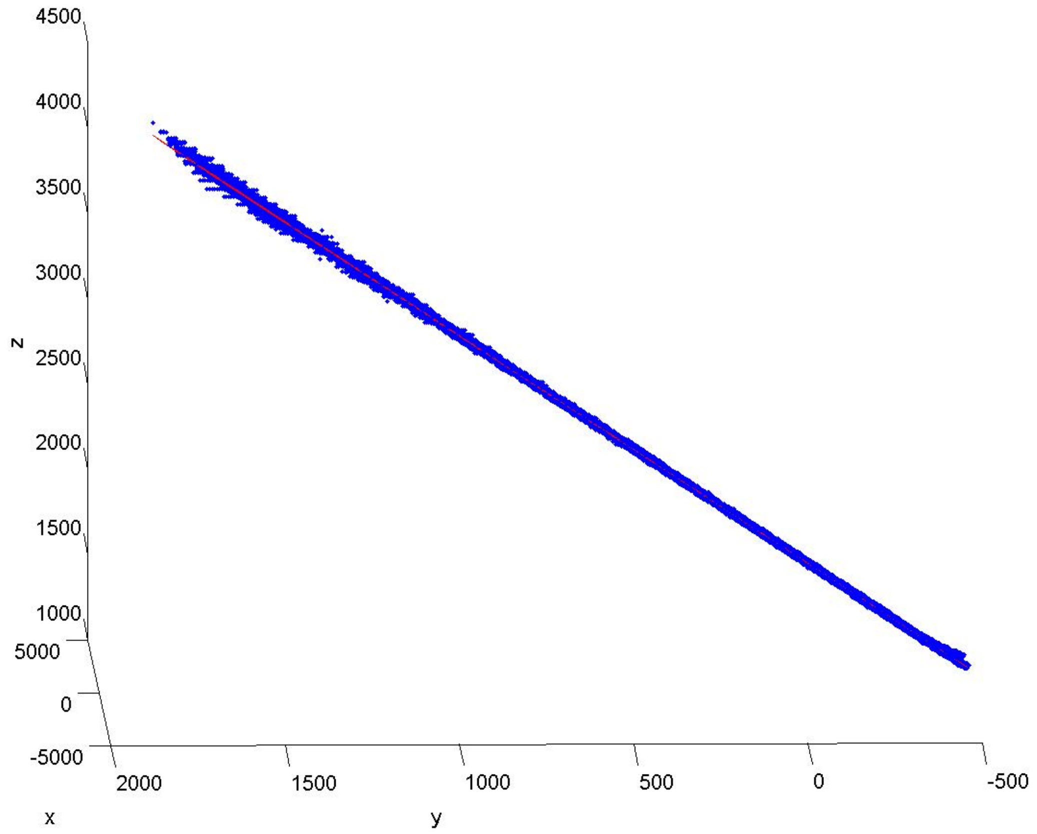


Figure 6: The blue points are raw data; the red ones show the best-fit plane. Units are mm.

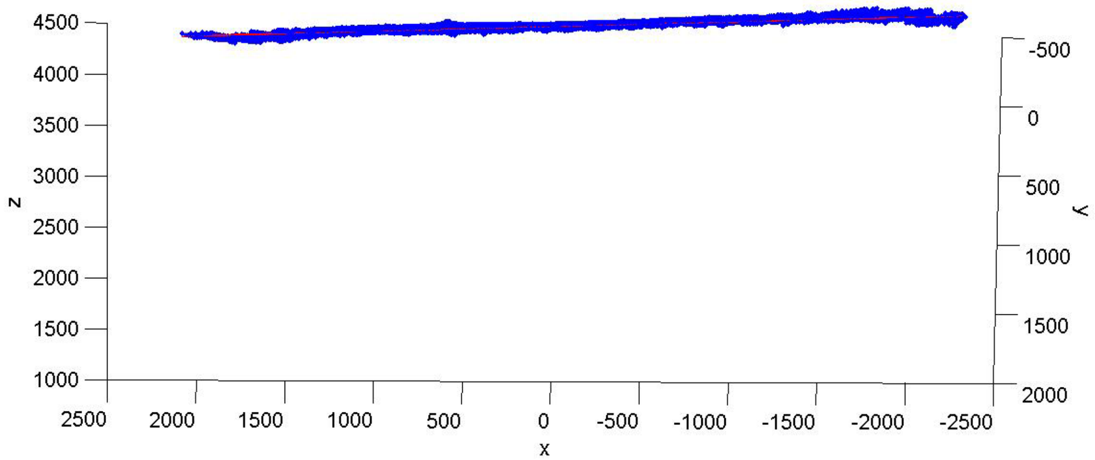


Figure 7: The blue points are raw data; the red ones show the best-fit plane. Units are mm.

For reference, Figure 8 shows the output from the Kinect's RGB camera:

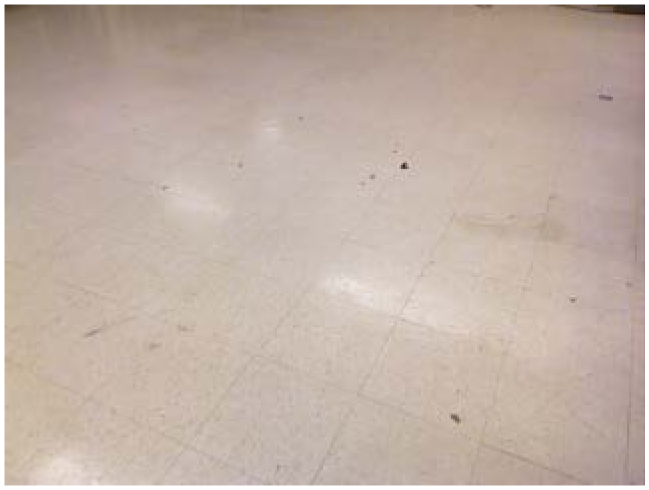


Figure 8: The Kinect's field of view for the data in the previous three images.

A small warping effect is present, but, overall, the data points fit the best-fit plane very well. The R^2 value for the plane fit is 0.9998, which is within the tolerable range for navigation. Any warping from the camera is negligible and does not need to be addressed for the purposes of this project.

Mounting angle

Different mounting angles were tested for data integrity. The depth map image was checked to see if large areas were black, indicating no data. It was found that the more directly the Kinect is pointed at the floor, the more likely it is that depth data will be received for that area. Figures 23 to 29 demonstrate this finding:

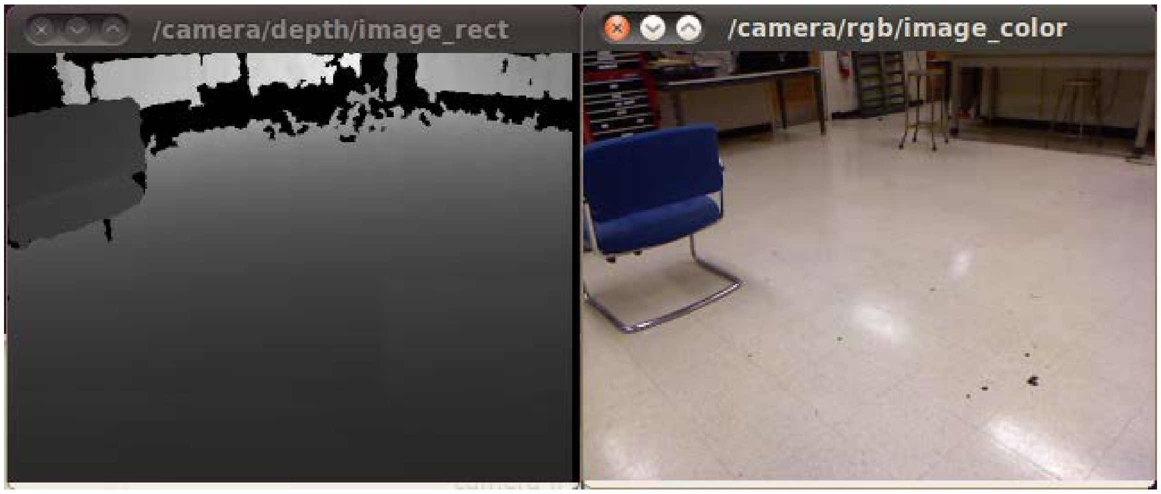


Figure 9: Testing different angles for the Kinect.

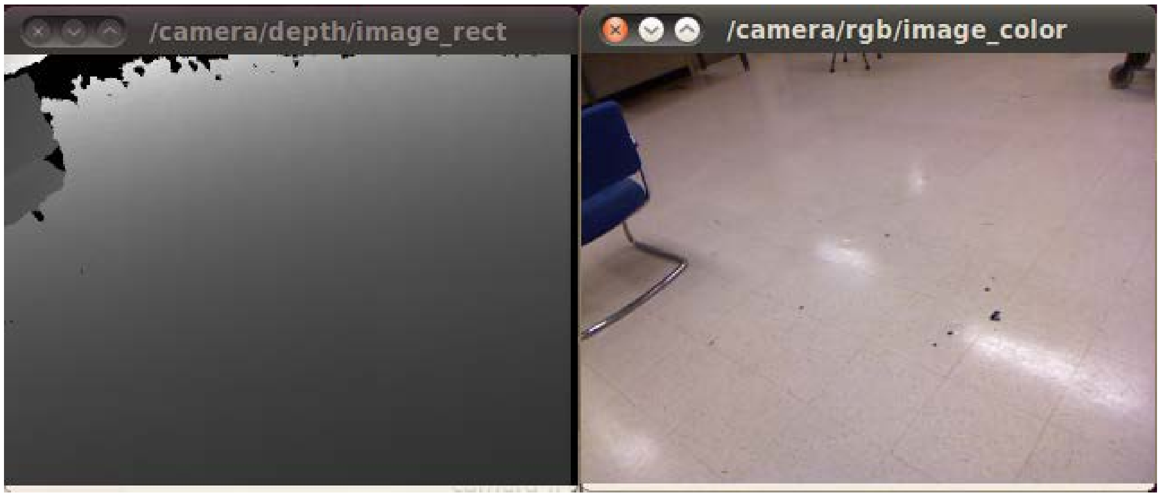


Figure 10: Testing different angles for the Kinect.

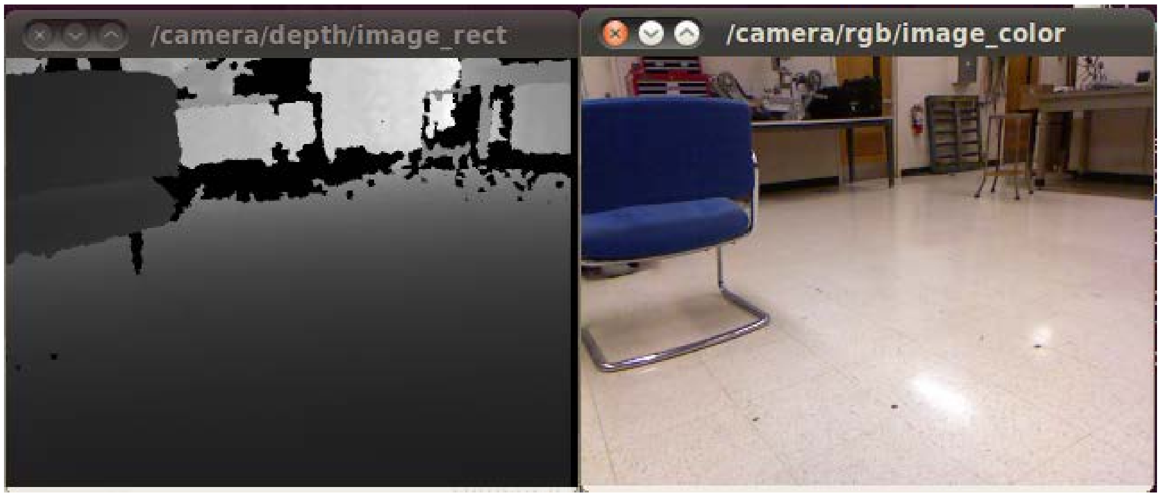


Figure 11: Testing different angles for the Kinect.

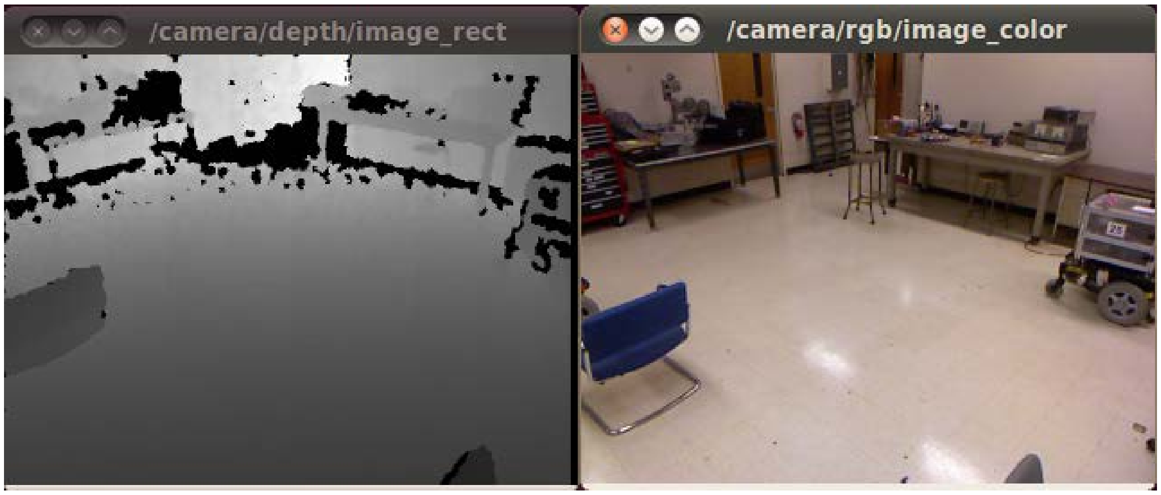


Figure 12: Testing different angles for the Kinect.

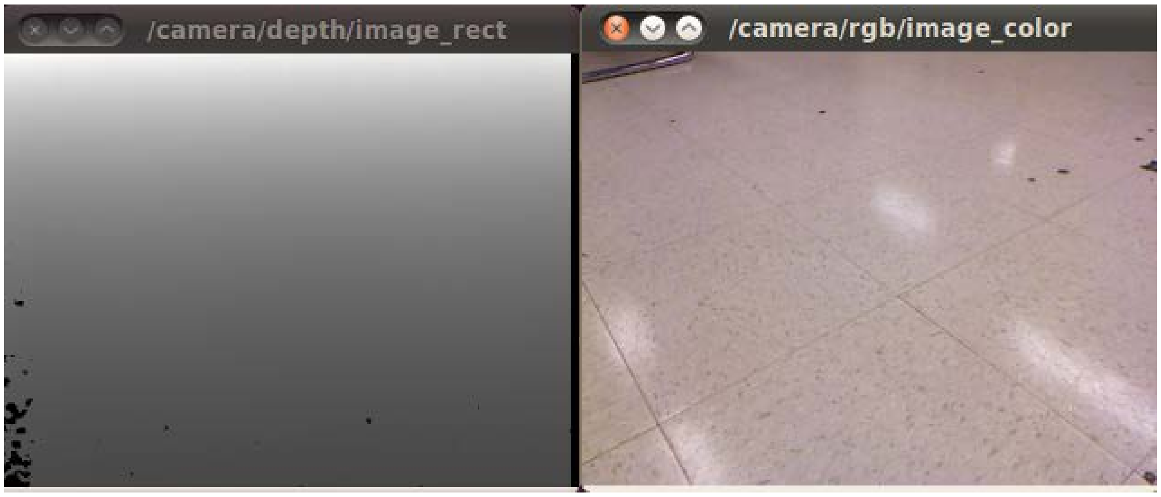


Figure 13: Testing different angles for the Kinect.

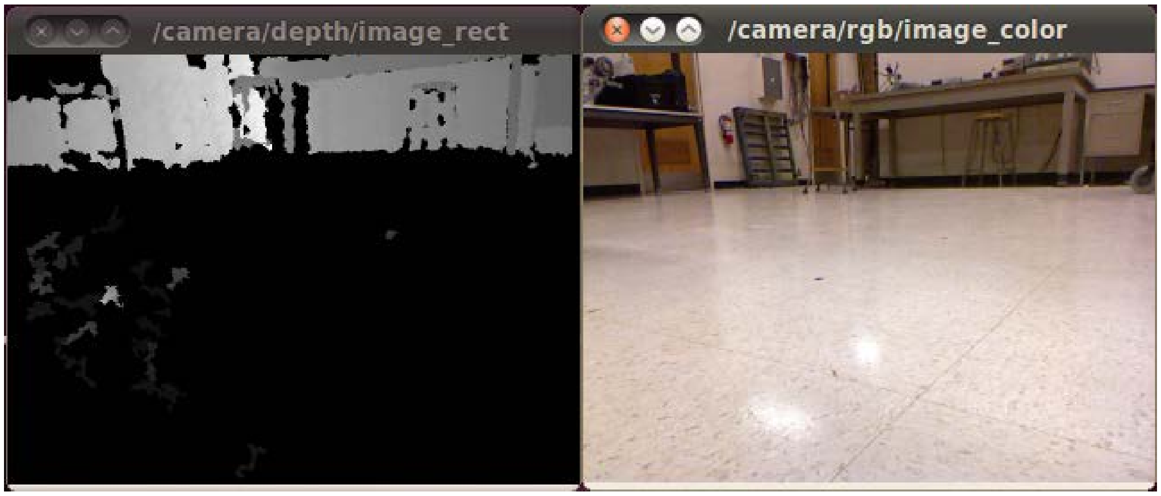


Figure 14: Testing different angles for the Kinect.

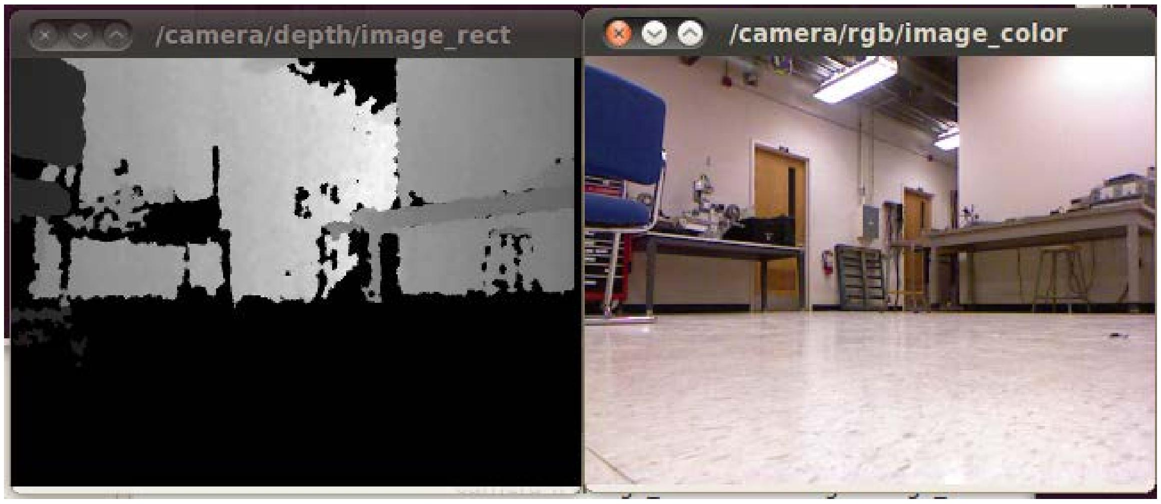


Figure 15: Testing different angles for the Kinect.

In Figures 23 to 27, the floor shows up in various shades of gray in the depth image, which means the Kinect is able to get data for that area. In Figures 28 and 29, the depth map colors the floor black; the angle is too shallow and the Kinect cannot accurately detect the floor surface. This test provides information necessary for determining the final placement of the Kinect on the robot.

Appendix A describes a test that compared the Kinect's depth data for different types of floors. For this test, the Kinect was mounted at a 13 degree angle from the horizontal, and had difficulty detecting black surfaces. However, at steeper angles, the Kinect was successful in detecting the black floor. The following figures show the result of view the black floor using several different angles for the Kinect:

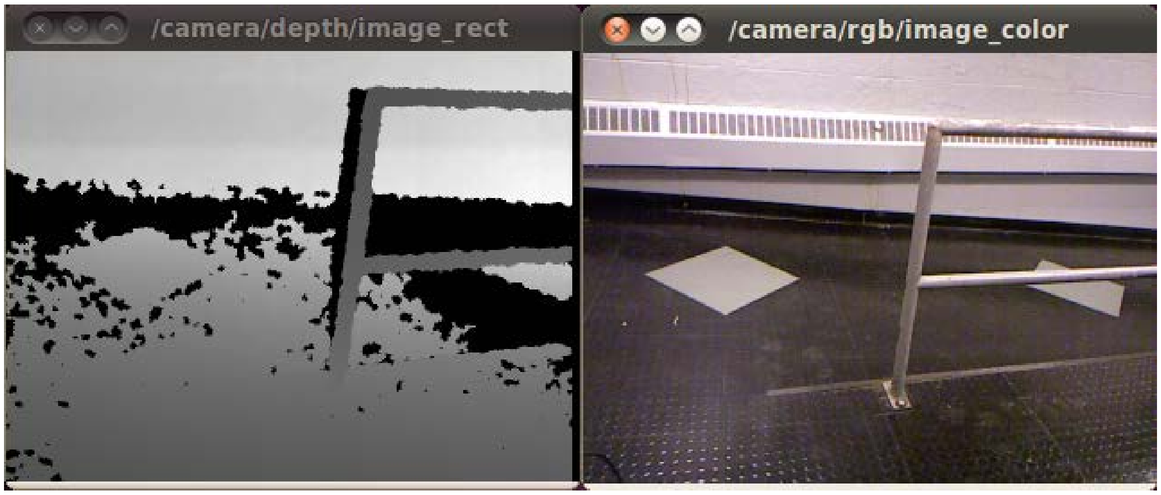


Figure 16: Testing the effectiveness of different angles for viewing a black floor.

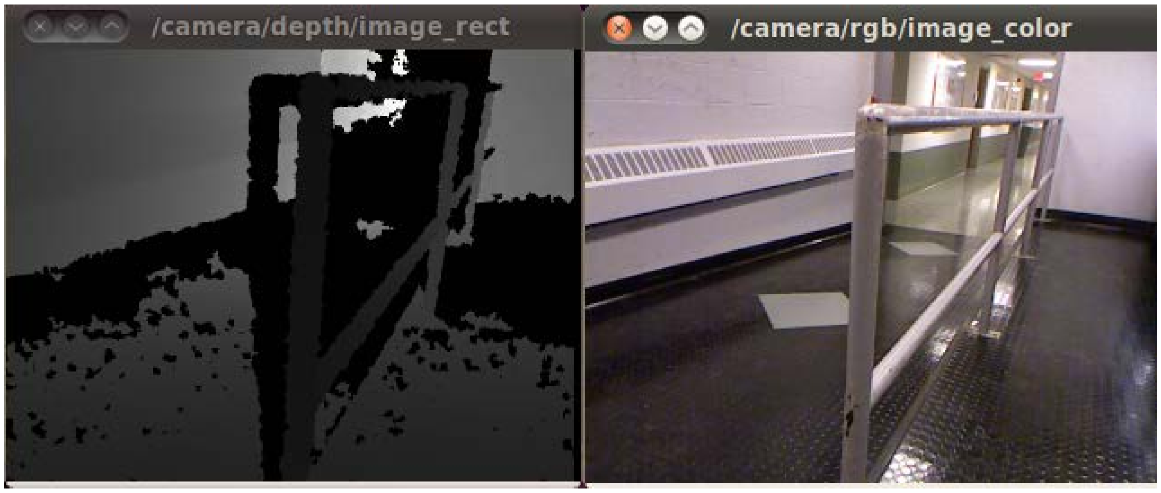


Figure 17: Testing the effectiveness of different angles for viewing a black floor.

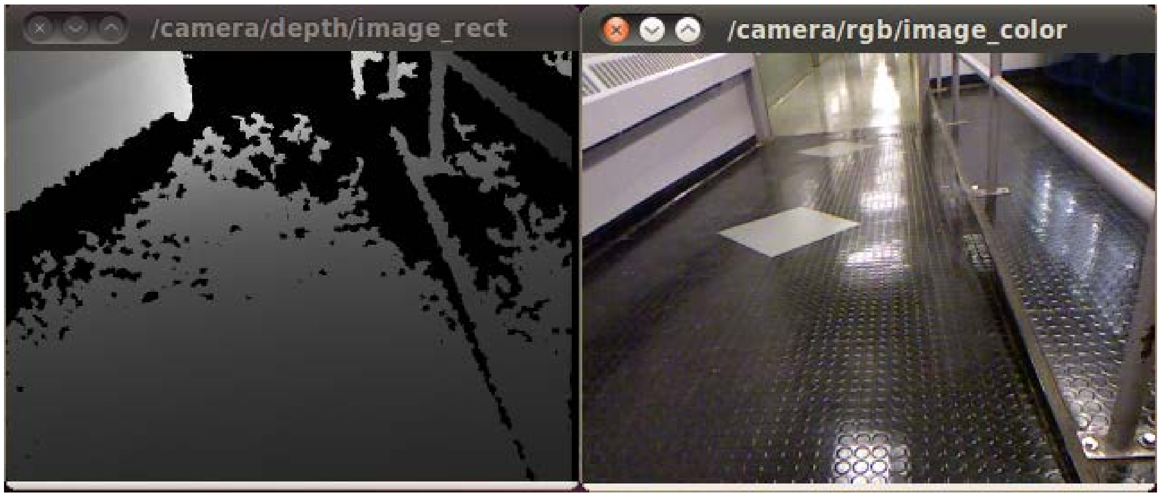


Figure 18: Testing the effectiveness of different angles for viewing a black floor.

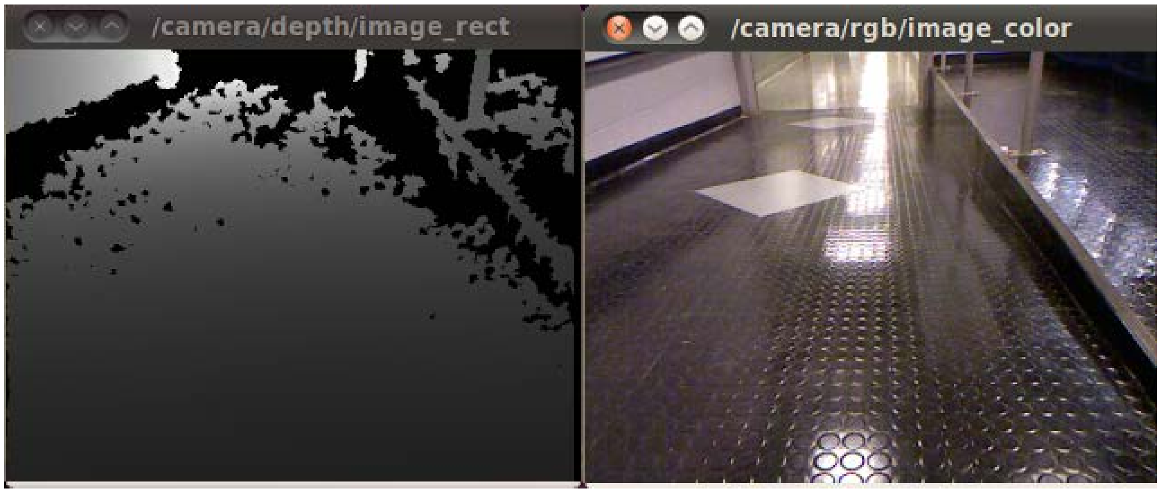


Figure 19: Testing the effectiveness of different angles for viewing a black floor.

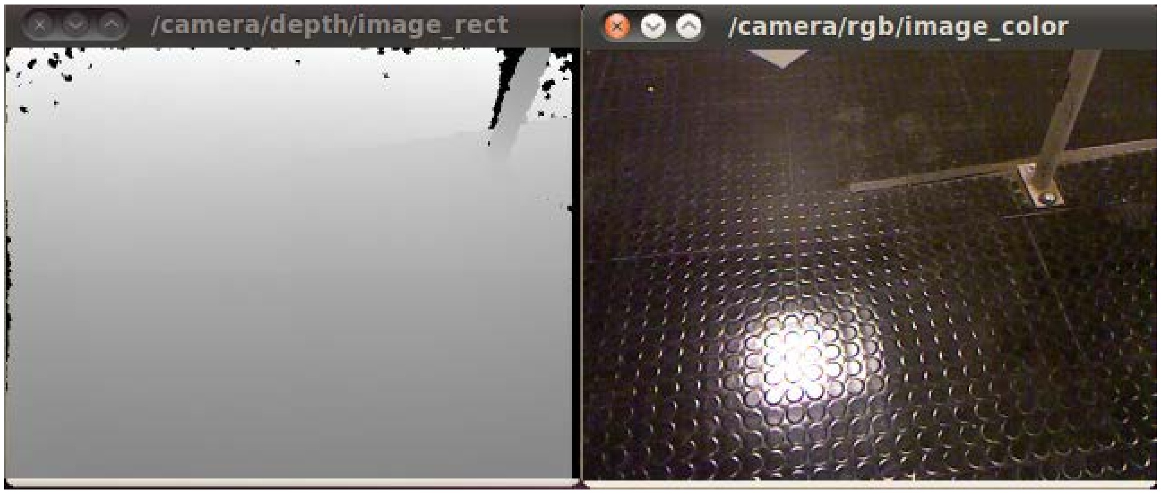


Figure 20: Testing the effectiveness of different angles for viewing a black floor.

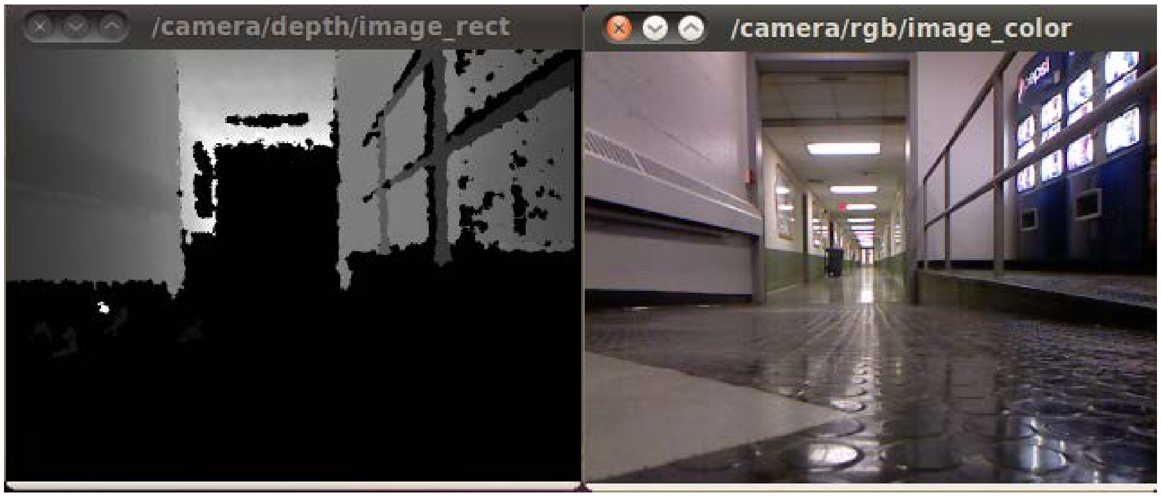


Figure 21: Testing the effectiveness of different angles for viewing a black floor.

Figure 34 shows that even though the black surface is harder for the Kinect to detect, it is not impossible to get data if a very steep angle is used. However, the angle used for that image is impractical for the robot because it greatly limits the field of view. Figure 35 shows that when the Kinect is standing on the floor, it cannot gather any depth information about the floor. This result is the same as that shown in Figure 29, where the Kinect was viewing a white floor, and unable to gather any data about the floor surface. It was able to provide data for some regions of the floor but not others, especially those farther from the Kinect. However, for practical purposes this is sufficient; the purpose of the Kinect in this project is to sense the floor directly in front of the robot. Additionally, it is possible to fill in small regions with no data by interpolating from the existing data points.

The purpose of a robot influences the optimal mounting angle. Since this project is concerned with the Kinect's ability to examine the floor surface immediately in front of the robot, it was decided that the Kinect should be positioned so a large region of its field of view is the floor immediately in front of the robot. It is unnecessary to gather data for areas more than a few meters away. For example, Figure 36 and Figure 37 show the Kinect's field of view when mounted at a 13 degree angle from the horizontal and a 40 degree angle from the horizontal:

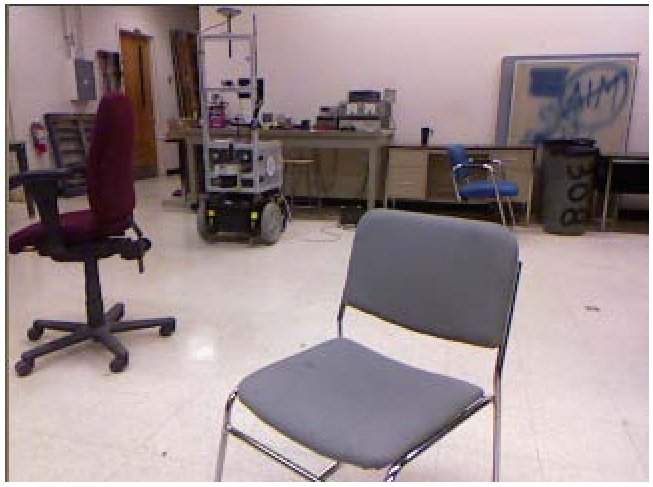


Figure 22: Kinect is mounted at a 13 degree angle from the horizontal

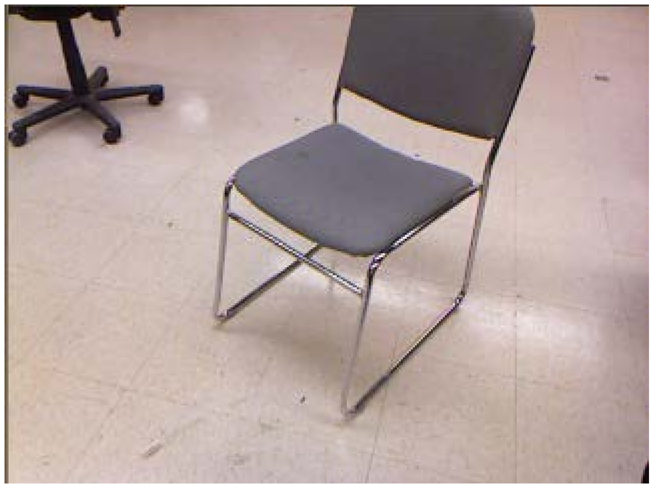


Figure 23: Kinect is mounted at a 40 degree angle from the horizontal

In Figure 36, a significant proportion of the image is taken up by the wall 6 meters away from the robot. Any data from this region is not very useful; it is farther away and therefore less accurate. Also, the robot already has a lidar sensor capable of detecting this kind of obstacle. The purpose of the Kinect is specifically to look for small items or unusual characteristics of the floor.

Figure 37 shows a better result; the Kinect is able to see a region of the floor closer to the robot itself. Also, in this case it is less likely that parts of the floor will show up as black on the depth map because of a lack of data. The angle used is more direct than that of Figure 36, and the field of view is limited to points within approximately 3 meters of the robot.

Conclusion

Taking into account the above information about the Kinect's limitations and its purposes in the wheelchair project, it was decided to mount it on the front of the robot, about 1 meter above the ground, pointed toward the floor at about a 40 degree angle from the horizontal. This height would work very well for a Kinect mounted on the back of the wheelchair, to assist with backing up. For a Kinect on the front of a wheelchair, it would probably need to be mounted higher so it is not in the way of the human operator, but the data-processing technique described in this thesis can be easily adapted to different heights and angles.

The data presented here shows that mounting the Kinect at an angle between 30 and 50 degrees, at a height of about 1 m, will give very reliable data that can be analyzed to determine the transform needed for the Kinect data and the different types of features on the floor in front of the robot, as will be shown in the following chapters.

Chapter 4: Calculating the transform

Because the Kinect is mounted at an angle, the coordinates of the data points returned must be transformed so they can be considered from the frame of reference of the robot as a whole. This is accomplished by using a matrix to rotate the coordinates according to certain roll, pitch, and yaw angles. The angles must be precise in order for the rotation to be effective; otherwise the robot will be unable to distinguish the floor itself from small items on the floor.

Best-fit plane

After some initial processing to select the points which belong to the plane of the floor (explained in the appendix), a plane was fit to the points. Though large matrices can be inconvenient and computationally intensive, this calibration process only needs to be performed one time, so the slowness of the algorithm is not a problem.

Compute a rotation matrix

After the equation for the plane is found, the coefficients are used to create a rotation matrix which will be used later to interpret all of the depth data the Kinect receives.

The rotation matrix is a 3x3 matrix which is used to change the coordinates from the Kinect’s reference frame to the robot’s reference frame. (In addition to this matrix, a translation is also required because the Kinect’s origin is higher than the robot’s base.) The first row of the rotation matrix is the vector that represents the x-axis of the Kinect’s reference frame. (But the coordinates are with respect to the robot’s reference frame.) Similarly, the second row represents the y-axis and the third row represents the z-axis.

If the equation for the plane is $Ax + By + Cz = D$, then one such rotation matrix is as follows:

$$\begin{bmatrix} \sqrt{B^2+C^2} & -\frac{AB}{\sqrt{B^2+C^2}} & -\frac{AC}{\sqrt{B^2+C^2}} \\ 0 & \frac{C}{\sqrt{B^2+C^2}} & -\frac{B}{\sqrt{B^2+C^2}} \\ A & B & C \end{bmatrix}$$

Note that, given only an equation for a plane, there is no way to know the yaw angle. The above rotation matrix is designed such that the yaw angle is 0, because theoretically it should be 0 (the Kinect and the robot are both facing forward). It is possible to precisely calculate yaw only by moving the robot and comparing the Kinect data with the robot's position data. For the purposes of this project, it is not necessary to get such an accurate number for the yaw, and the complexity and tedium of the code required to calculate it is not worth the small improvement in accuracy it may bring.

Therefore, the process to find the transform for the Kinect can be summarized as follows: a plane is fit to the points that are believed to belong to the floor, which are found by a process that first estimates the angle of the Kinect by calculating slopes and choosing the most common one.

Accuracy

Next, the accuracy of this method will be discussed. The error of the plane fit can be found by examining the variation of the data from the plane that was found. Specifically, the rms (root-mean-square) error of the data points from the plane that was found was calculated. In order to evaluate this algorithm, several different sets of data, selected in different ways, were compared to several different plane equations, calculated in different ways.

First of all, it should be noted that not all of the data points returned by the Kinect were used for this process. Because there is slight warping at the far corners of the depth map, only the points within a certain rectangle in the Kinect's field of view were considered. This rectangle's width extended from ¼ to ¾ of the width of the image, and the height was from ¼ to 1. This rectangle is shown in Figure 44, which was taken when the Kinect angle was approximately 40 degrees from the horizontal:

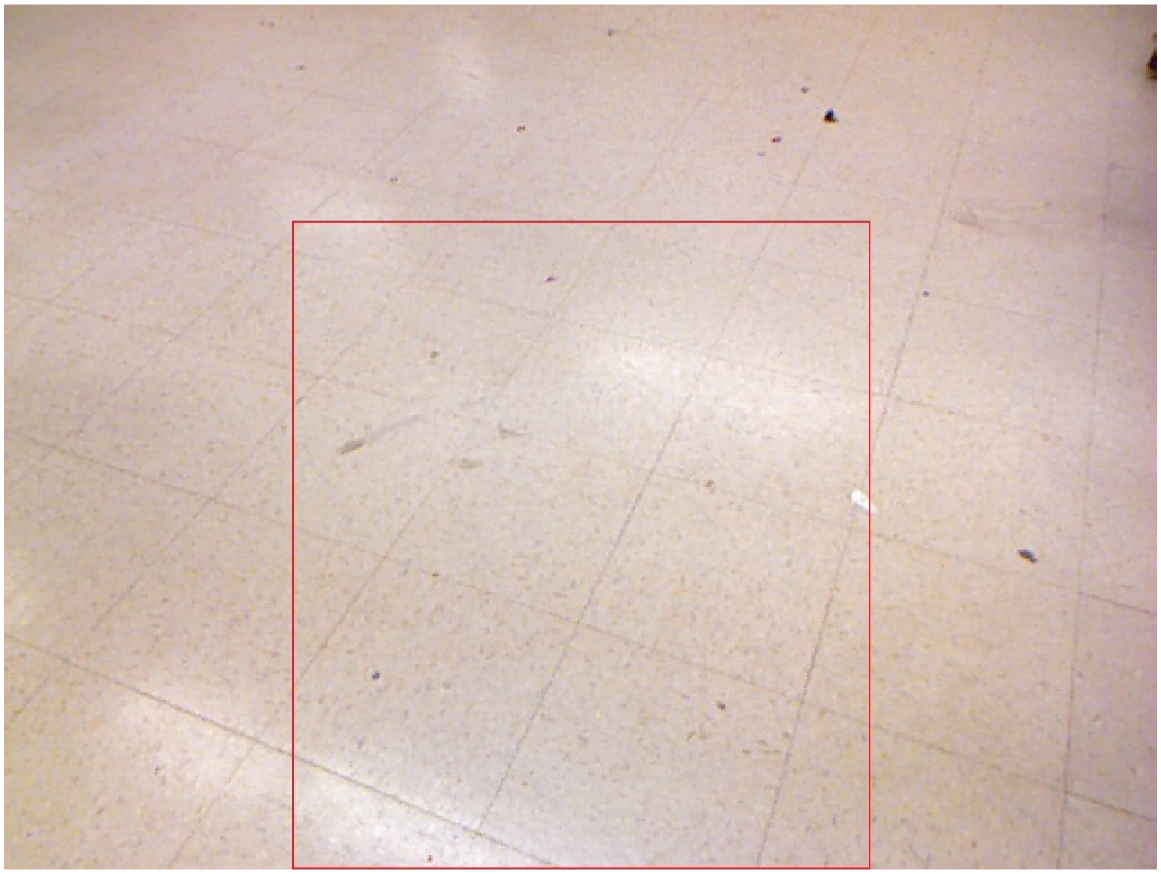


Figure 24: Depth data from the area within the red box was used to calculate the transform to calibrate the Kinect.

Here are the different data sets that were considered and used to judge the accuracy and usefulness of this calibration method:

Data set 1 is the set of points which were judged to belong to the floor, according to the above method where the angle was estimated using slopes, and points were sorted by height. The points belonging to the largest bin, and the 2 bins adjacent to it, were judged to belong to the floor, and are therefore data set 1.

In Figure 45, the blue points are data set 1, and the green points were rejected because their height was not close enough to the calculated height. This image shows that the points that were rejected are only at the extreme corners, and it is reasonable to reject them. (The amount and location of the rejected points varies, but they are always few and near the edges. Figure 45 just shows one typical example.) Therefore a plane fit to the points in data set 1, as was described above, will be very close to the theoretical answer for the plane of the floor.

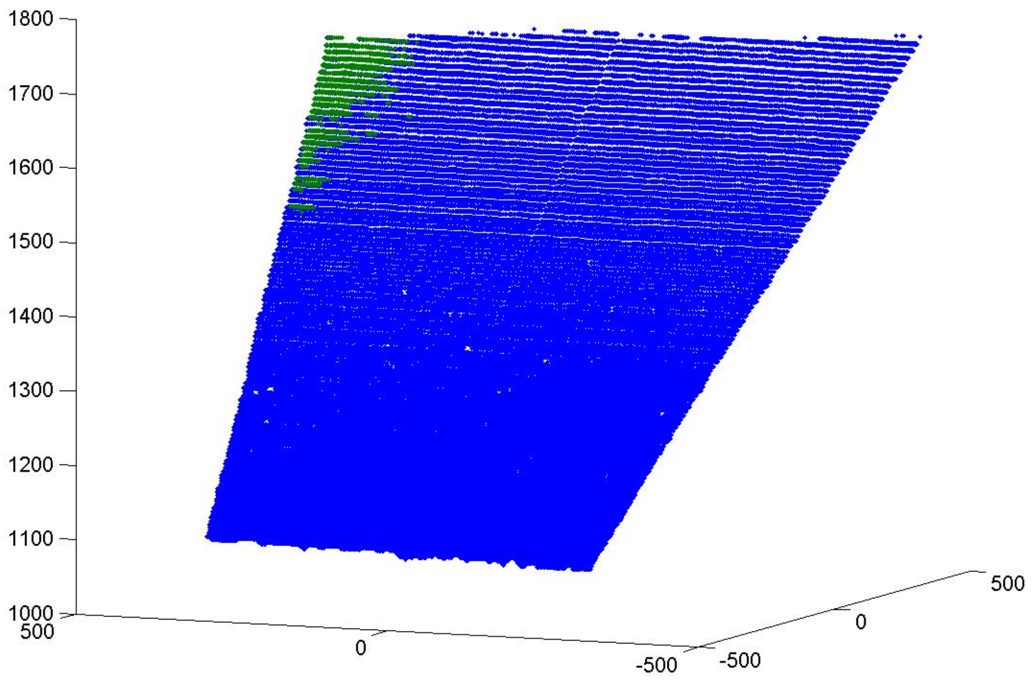


Figure 25: The blue points (data set 1) were judged to belong to the floor. The green points were rejected. The blue and green points together are data set 2.

Data set 2 is all of the points within the rectangle- in other words, both the blue and green points in the above image.

The following 3 planes were used:

Plane 1 is the best-fit plane for the points in data set 1. This plane is the one that was used in the end to transform the data.

Plane 2 is the best-fit plane for data set 2.

Plane 3 is the plane that was constructed using only the angle estimate and the height estimate found in the earlier steps of the method described above. This assumes that the Kinect is rotated only in one dimension. (It assumes that roll=0, and only uses the angle for the pitch.)

Table 4 shows the rms errors of each data set with respect to each plane. The process was repeated 5 times; the table shows the average values. Note that for each run, all of the data sets and planes were taken from the same set of raw data. (The units are mm.)

	Data Set 1	Data Set 2
Plane 1	3.3811	3.5069
Plane 2	3.3891	3.5002
Plane 3	4.2649	4.4900

Table 1: Rms error of best-fit planes calculated in different ways.

In this case, the rms errors were very small. For the above data in particular, the Kinect angle was approximately 52 degrees from the horizontal; steep angles like this tend to result in more accurate results. Regardless, the above data shows that the method previously described (which resulted in the calculation of "Plane 1") is very effective.

In summary, the method described above is an effective method of finding the Kinect's transform with respect to the robot. The above data shows it to be accurate with an error on the order of 5 mm, depending on the angle of the Kinect.

Effect of robot's movement

However, there may be some variation when the robot moves to a different position. If the transform is calculated at one position, and that transform is used to interpret the Kinect's depth data even after the robot moves on to another location, is it still accurate enough? To answer this question, 16 different data samples were taken at 8 different locations. For each data set, a best-fit plane was calculated, using the calibration method described above. Next, the rms errors with each of the 16 data sets with respect to each of the 16 planes were calculated. This indicates how accurate a plane fit at one location would be for interpreting the data from another location.

In Table 5, each column represents a different data set, and each row represents an equation for a plane. The first plane is the one fit to the first data set, and so on. Therefore the diagonal entries will by definition be the smallest numbers; they represent the rms error of the ideal plane for each data set. The units are mm.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	3.95	9.10	7.07	5.31	4.95	7.65	7.90	5.94	5.30	6.45	3.96	4.85	4.71	5.07	5.14	5.07
2	9.15	3.91	12.12	9.12	8.77	5.14	4.70	5.97	11.76	12.52	8.04	9.29	8.90	8.09	7.79	7.69
3	7.15	11.78	3.68	5.23	5.65	9.41	9.94	8.40	6.91	4.98	7.56	4.94	5.17	6.06	6.60	6.74
4	5.35	8.93	5.11	3.89	3.94	6.86	7.25	5.76	6.03	5.57	5.14	3.91	3.85	4.29	4.52	4.59
5	5.01	8.65	5.58	3.96	3.87	6.72	7.06	5.49	5.75	5.72	4.67	4.01	3.89	4.28	4.45	4.49
6	7.56	4.89	9.33	6.76	6.60	4.12	4.08	4.69	9.95	10.08	6.62	6.95	6.61	5.77	5.41	5.37
7	7.86	4.54	9.92	7.21	7.00	4.18	4.03	4.84	10.28	10.57	6.85	7.43	7.07	6.28	5.94	5.88
8	5.82	5.76	8.29	5.62	5.32	4.69	4.75	4.12	8.05	8.58	4.93	5.67	5.34	5.02	4.89	4.82
9	5.30	11.62	6.80	5.98	5.70	9.93	10.22	8.09	3.88	5.36	5.42	5.83	5.87	5.88	5.82	5.82
10	6.23	12.12	4.52	5.30	5.44	9.95	10.40	8.47	5.08	4.21	6.66	5.04	5.25	5.88	6.23	6.33
11	4.21	8.14	7.62	5.29	4.82	6.91	7.07	5.29	5.61	7.00	3.67	5.11	4.88	5.11	5.14	5.05
12	5.01	9.23	4.96	4.07	4.14	7.15	7.56	5.93	5.98	5.45	5.07	3.74	3.72	4.18	4.44	4.50
13	4.91	8.87	5.24	4.05	4.08	6.84	7.24	5.64	6.05	5.68	4.87	3.77	3.70	4.07	4.29	4.34
14	5.41	7.81	6.07	4.57	4.62	5.87	6.25	5.07	7.37	6.91	5.22	4.26	4.09	3.87	3.92	3.96

15	6.17	7.17	6.89	5.29	5.36	5.35	5.70	5.01	8.51	7.97	5.87	5.02	4.81	4.01	3.84	3.86
16	6.10	7.08	6.99	5.32	5.37	5.31	5.65	4.95	8.48	8.01	5.79	5.04	4.82	4.02	3.84	3.86

Table 2: 16 data sets were taken, at 8 different locations. For each data set, the plane of the floor was calculated. This table shows the error of each data set with respect to each plane. Units are mm.

The highest value in this table is 12.52. The average is 6.06 (or 6.20 when the diagonal entries are not included). In other words, if the computer calculates a transform, then moves the robot to another position, one can expect an error of 6 mm between the plane from the calibration and the data for the plane of the floor at the new position.

For the purposes of this project, this error is acceptable. The robot will need to avoid running over people's feet (30 mm tall), and even with the maximum error, 12.52 mm, it is still well within the bounds of being able to detect 30-mm-tall objects.

Next, using the data from all 16 data sets, one best-fit plane was calculated. The rms error was 5.2980 mm. Then the error for each of the 16 data sets with respect to the big plane was calculated. The mean was 5.12 and the maximum was 7.83.

This shows that by taking many snapshots of data and using all of them to fit a best-fit plane and calculate the transform, the result has much lower error than only taking depth data one time. The mean error went from 6.06 to 5.12 and the maximum went from 12.52 to 7.83.

However, either method would be acceptable. In both cases, the errors are very small, on the order of 1 cm. The improved accuracy is not important for most applications; a robot is able to drive over slightly uneven floor anyway. It is not necessary to detect roughness on such a small scale. Therefore, because the coding was simpler, the calibration was run using only one data set. (However, the method was rerun from time to time so the transform could be updated if necessary.)

In summary, the method presented here uses a snapshot of data when the Kinect's field of view mostly consists of a level floor surface, and calculates the necessary angles to use for the transform. This will be essential for the next chapter, which builds a map in the robot's reference frame. Furthermore, it was found that the error between the calculated plane and the actual data is on the order of 5 mm, and that pursuing any greater accuracy than that would be meaningless because of the slight difference in the data itself when the robot moves from one location to another. Overall, this shows that the method is very accurate, and noise is small enough that it will not interfere with detecting small objects.

Chapter 5: Classifying features of the floor surface

After the Kinect is properly calibrated, the data it returns can be used to segment the area in front of the robot according to whether it is drivable. In this thesis, an algorithm is presented which classifies regions into the following four groups:

1. **Level floor. The robot can drive on this without any problems.**
2. **Ramps. It is often necessary to drive on ramps, but they require more careful consideration first. For example, the robot may need to slow down, and may need to approach a ramp from a particular angle. The path planner must determine how to safely navigate the ramp.**
3. **Bumps. These are small objects on the floor, but they are safe to run over. For example, power cords or small cracks in the floor would be classified as bumps. Also, it is usually preferable to avoid driving over the bumps if there exists another path entirely on flat floor. The path planner would then take the information about the location of bumps and determine whether to drive over them or avoid them. If the path planner code is very advanced, it may even plan a path where the robot passes over the bump but the wheels themselves do not contact it.**
4. **Obstacles. These must be avoided; the robot cannot safely run over them. This category also includes dangerous holes in the floor, and the edges of stairs- both going up and going down.**
5. **Unknown. If a point cannot be seen by the Kinect, it will be classified as unknown. This includes areas that are too far away or outside of the angle of the field of view, and also areas which are hidden behind obstacles. However, when this is actually implemented in real time, the**

map would be continually updated as the robot moves, so unknown places would eventually be filled in with data.

The objective of this method is to create a map of the area in front of the robot, with each (x,y) location having a value which tells which of the five categories that point falls into.

Creating a height map

The first step in the process it to make a map showing the height at each (x,y) location.

First, a two-dimensional array was created, with each entry corresponding to a specific (x,y) location in the robot's reference frame. Each point in the Kinect's point cloud was then transformed into the robot's reference frame (using the transform calculated in the previous section), and the resulting height, z, was stored in the array at the corresponding (x,y) index. It is often the case that multiple points from the point cloud would end up at the same entry in the two-dimensional array; in this case, the average height of all such points is stored in that entry of the array.

The problem that arises from this method is some (x,y) locations in the array, particularly those far from the Kinect, will end up with no data. This is not a huge problem; the resolution of the array can be easily chosen such that the area near the robot will accurately store all the data without leaving holes. Furthermore, the data from regions far from the Kinect is less accurate anyway, so it is not of such critical importance to fill in all the holes at that distance. However, it is very simple to check for isolated points where there is no data, and fill them in by averaging the values at the neighboring points. This process will be performed before there is any further analysis on the data.

To accomplish this, every cell with no data was examined. The four cells adjacent to it were checked. If there was data at all of the four cells surrounding it, those four values were averaged, and the cell in the center was updated. Otherwise, if there was data in the two cells above and below, or the two cells to the left and right, those two values were averaged. In this way, holes that are 1 pixel wide are filled in, but it does not create fake data in regions where none exists. The process was iterated until there were no more tiny holes that could be filled in.

This process is shown in Figure 46:

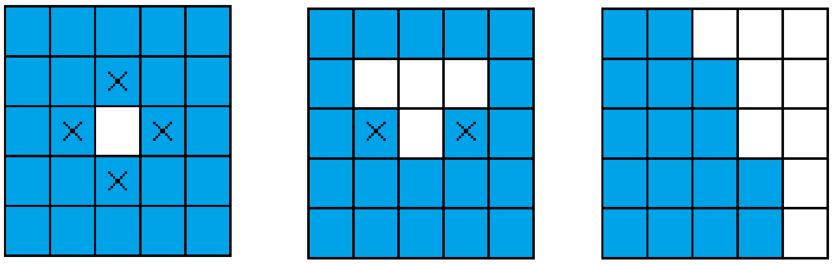
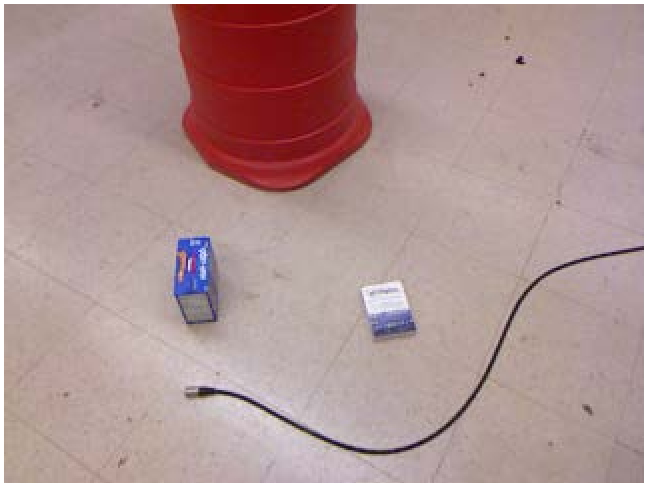


Figure 26: The blue cells have height data and the white cells do not. In the diagram on the left, the white square will be filled in with the average value for the four squares marked with an x. For the center diagram, the white square at the center will be filled in with the average value for the two squares marked with an x. In the diagram on the right, none of the white squares will be filled in.

At this point, the map showing the height of each cell is complete. Figure 47 and Figure 48 show a few examples:



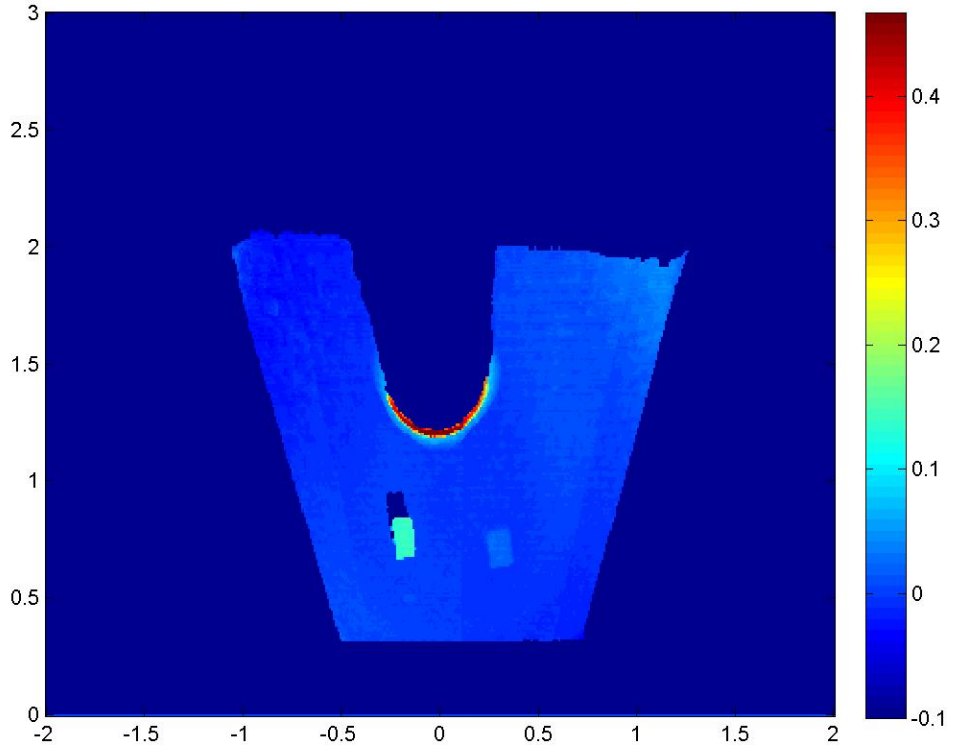
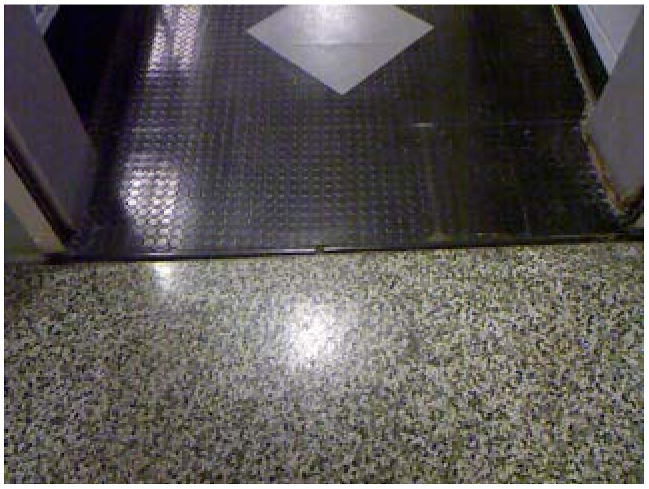


Figure 27: The Kinect is looking at four objects on a level floor. Top: RGB image. Bottom: Height map of the area in front of the robot. The height at each point is indicated by color (with unknown areas marked as -0.1 m and shown in dark blue). Units are meters.



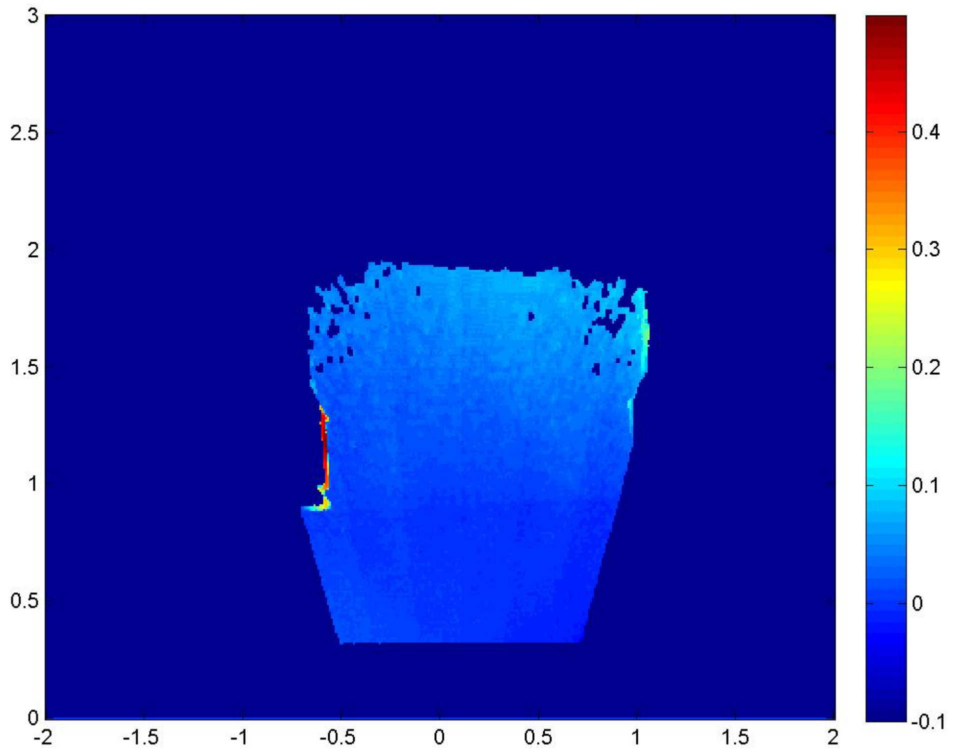


Figure 28: The Kinect is looking at a ramp. Top: RGB image. Bottom: Height map of the area in front of the robot. The height at each point is indicated by color (with unknown areas marked as -0.1 m and shown in dark blue). Units are meters.

Identifying bumps and obstacles

The drivability of a surface is based on its slope, rather than its absolute height. It is completely safe for the robot to drive on a surface that is significantly higher or lower than its current position if there is a gradually-sloped path to get there. Therefore, the following algorithm is based on subtracting heights in order to calculate gradients, rather than applying a single threshold value to the heights.

Due to the noise present in the data, the easiest features to detect are those which correspond to very clear changes in the height data; these are objects sticking up from the floor, which will be classified as bumps and obstacles. “Bumps” are small enough to be run over, and “obstacles” are too large, so the robot must plan a path around them. Although the bumps can be run over, it is often preferable to avoid them in favor of driving on flat floor. The robot’s path planner would be responsible for making those decisions.

Because bumps and obstacles are characterized by a very sudden change in height, they are the easiest features to find and will be detected first in this algorithm. In order to detect them, a second map was created, and filled in with values representing the gradient at each point. The gradients were calculated as follows:

$$gradient(i,j) = \sqrt{(d_{x1} + d_{x2})^2 + \left(d_{y1} + d_{y2}\right)^2}$$

d_{x1}, d_{x2}, d_{y1}, d_{y2} are defined to be the absolute changes in height from the point (i,j) to the 4 points adjacent to it. In other words, they are defined like this:

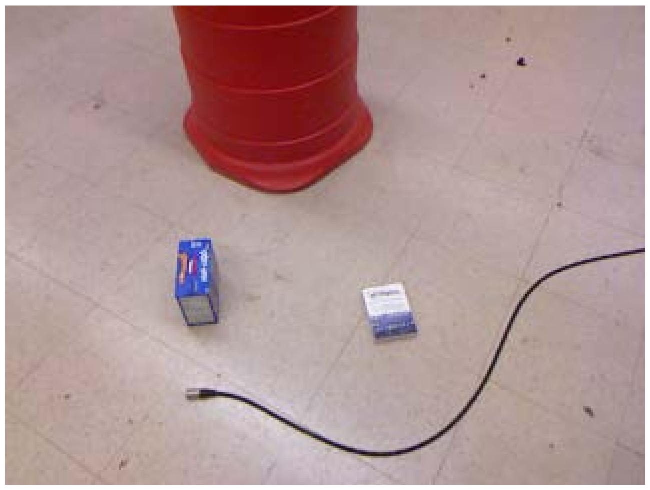
$$d_{x1} = |height(i,j) - height(i-1,j)|$$

$$d_{x2} = |height(i,j) - height(i+1,j)|$$

$$d_{y1} = |height(i,j) - height(i,j-1)|$$

$$d_{y2} = |height(i,j) - height(i,j+1)|$$

Figure 49 and Figure 50 show some examples for the output of this gradient map.



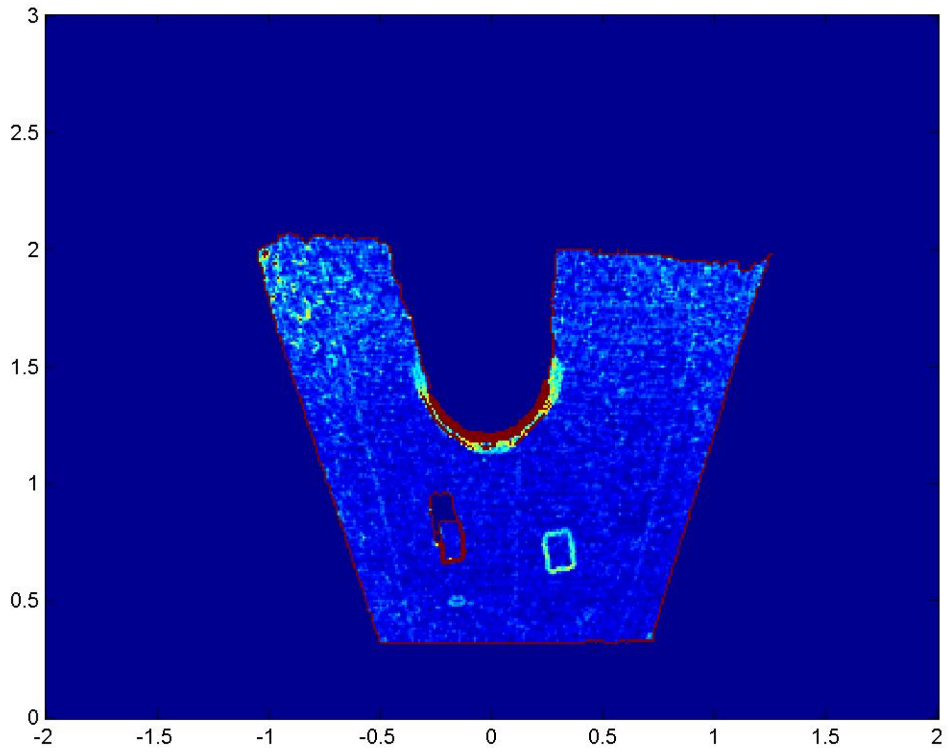
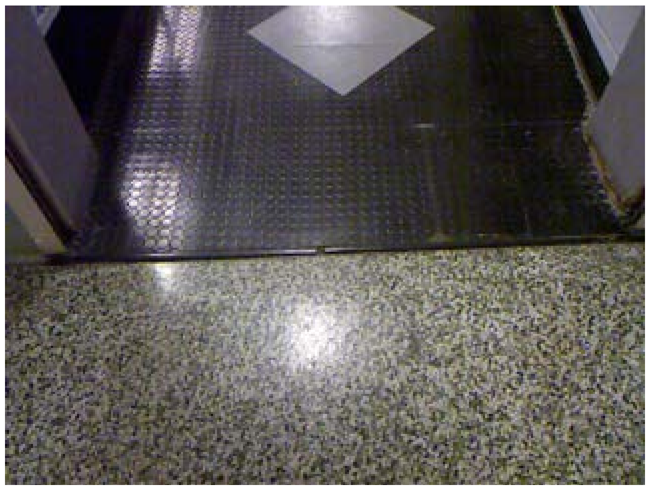


Figure 29: The Kinect is looking at four objects on a level floor. Top: RGB image. Bottom: Gradient map, showing slopes over very short distances. Blue areas have a low slope, red areas have a high slope.



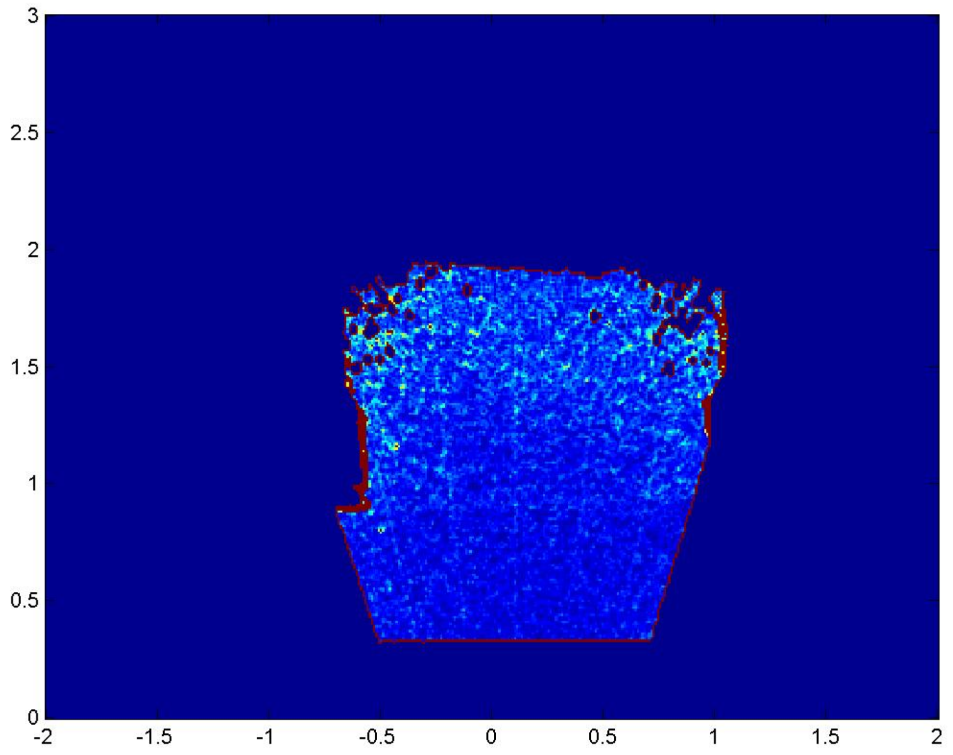


Figure 30: The Kinect is looking at a ramp. Top: RGB image. Bottom: Gradient map, showing slopes over very short distances. Blue areas have a low slope, red areas have a high slope.

Figure 49 shows that the 4 objects are all detected by this process, and a threshold can be applied to divide them into “obstacles” and “bumps”. In Figure 50, there is no clear difference in color between the level floor and the ramp. This is because the data has too much noise to detect a shallow slope between points so close together.

Next, two threshold values were used, so that the cells could be divided into 3 categories: floor/ramp, bumps, and obstacles. At this point in the process, all obstacles and bumps have been detected and categorized; all that remains is judging whether each of the remaining points belongs to a level floor or a ramp.

Identifying level floors and ramps

In the final step, the data will be further processed to distinguish the floor from a ramp. Because of the noise, it is only possible to detect a ramp by calculating the slope over a long distance. Mathematically, this can also be accomplished by processing the data with a large averaging filter, and then calculating the height differences of adjacent cells.

The averaging filter used here is a square 30 cells long and 30 cells wide. For each point that needs to be classified as either floor or ramp, the averaging filter is placed over it, so that point is in the center, and then the average value of all cell heights for floor/ramp cells within the filter is stored in a new map containing all the averaged data. In other words, the filter ignores any bumps or obstacles that have already been detected; if height data from the bumps and obstacles were included in the averaging process, it would destroy any meaningful slope data for the floor area that is near the bumps or obstacles. Points with no data, which are classified as “unknown”, are also ignored by the averaging filter.

Figure 51 shows a simple example of the filtering process (with a smaller averaging mask than the one used in the algorithm described here). The black cells represent the edge of an obstacle. Therefore, only the height data from the cells marked in red will be averaged.

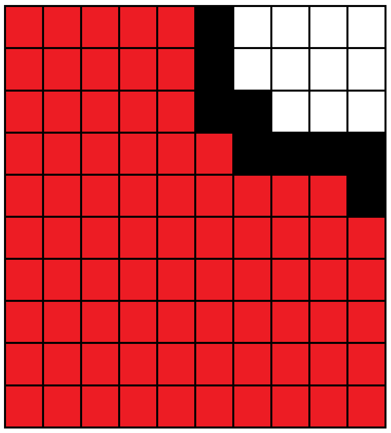


Figure 31: Averaging mask. Only the height data from the red cells is used in the averaging process for the point at the center of the filter. The black cells indicate the edge of an obstacle.

No either a level floor or ramp. However, since they are separated from the others by the edge of an obstacle, it can be assumed that they are not at the same height as the red cells, and therefore their data is not included in the average.

In this way, the averaging mask is used to smooth out the data from the floor surface and remove noise, without blurring the edges of objects on the floor.

Next, a gradient map was calculated, using an equation similar to the one described above, applied to the smoothed height data. Here is the equation:

$$gradient(i,j) = \sqrt{{d_x}^2 + {d_y}^2}$$

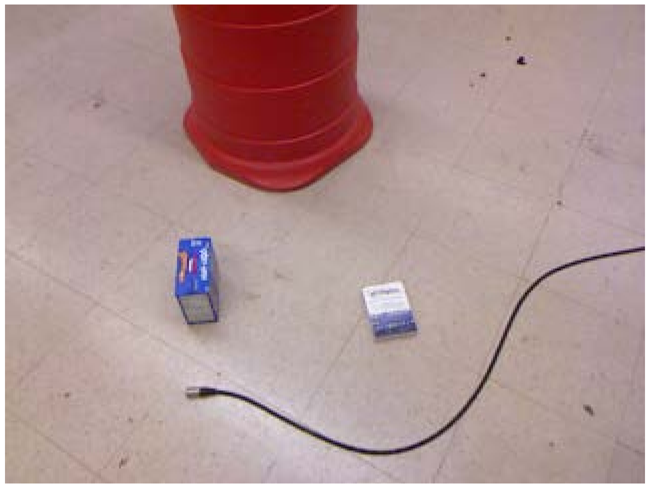
d_x and d_y in the above equation are defined as:

$$d_x = |height(i, j) - height(i - 1, j)|$$

$$d_y = |height(i, j) - height(i, j - 1)|$$

Note that in this case, the heights that are subtracted are from the data after the averaging filter is applied.

Figure 52 and Figure 53 show some examples of the gradient map that was created by this process:



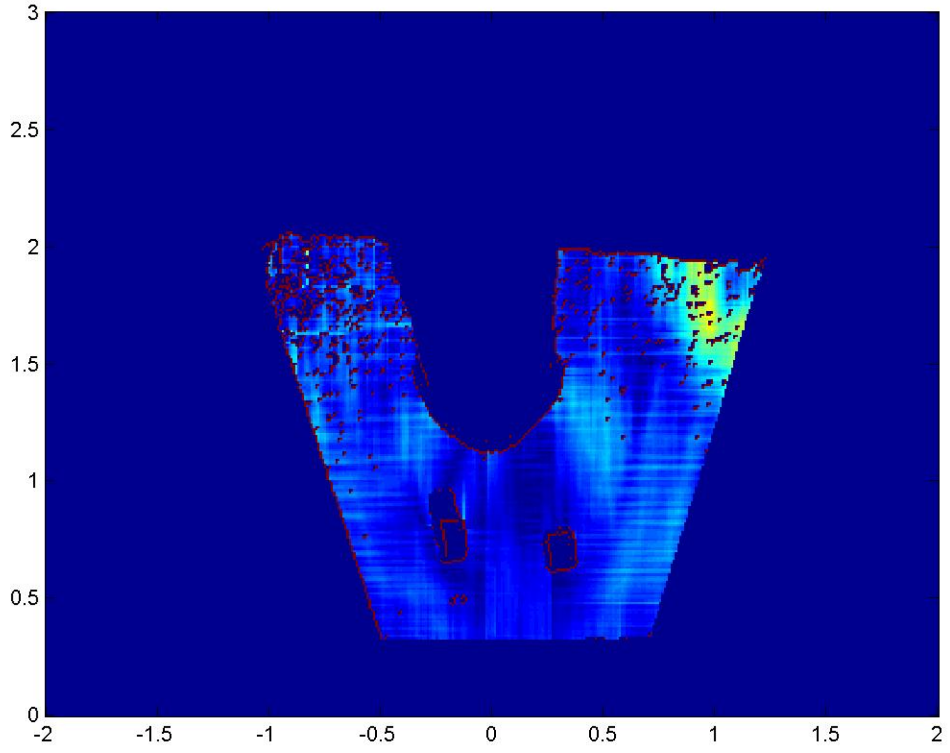
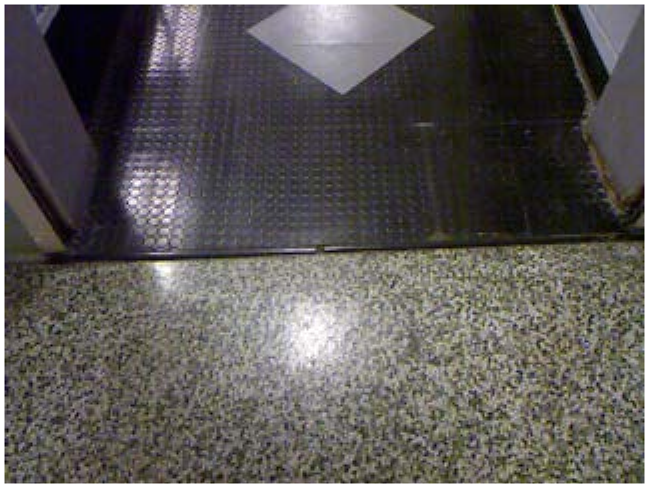


Figure 32: The Kinect is looking at four objects on a level floor. Top: RGB image. Bottom: Gradient map, showing slopes over long distances. Blue areas have a low slope, red areas have a high slope. (The darkest red values show points with no data.)



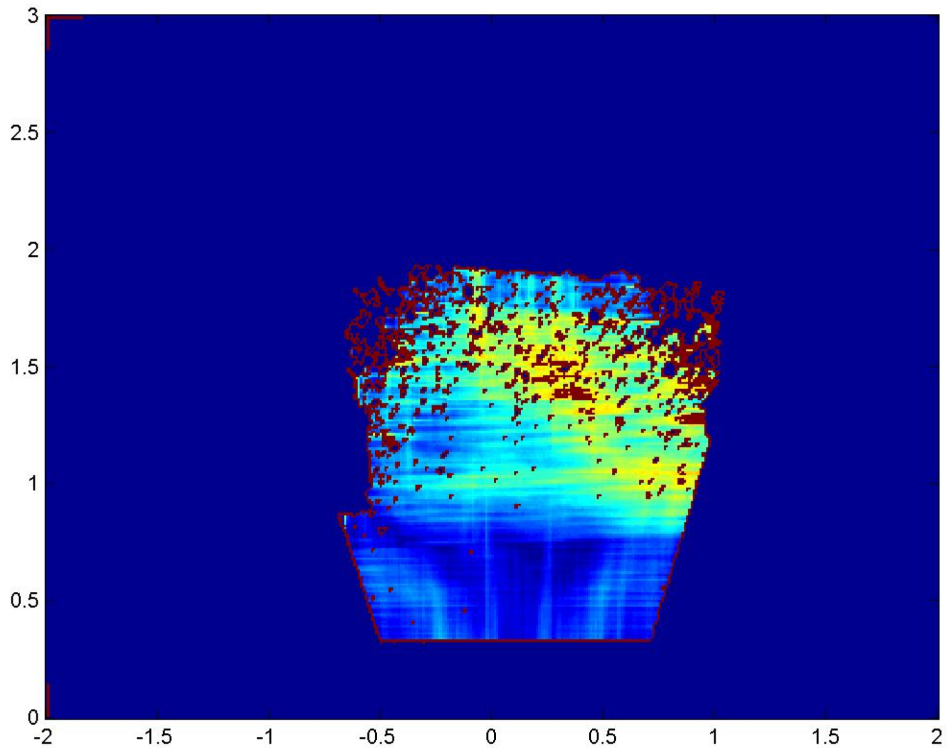


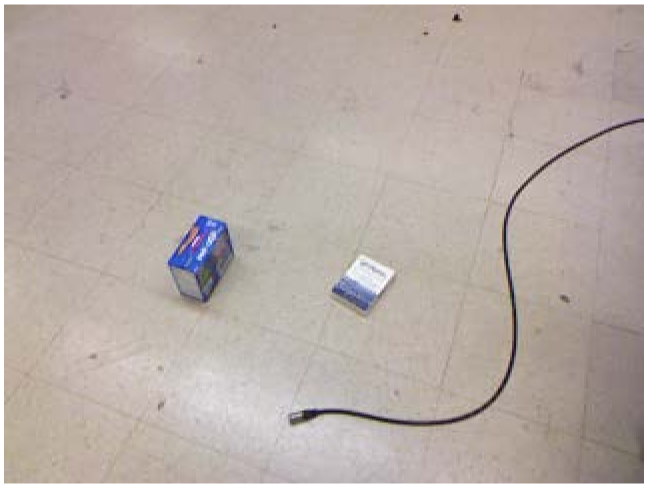
Figure 33: The Kinect is looking at a ramp. Top: RGB image. Bottom: Gradient map, showing slopes over long distances. Blue areas have a low slope, red areas have a high slope. (The darkest red values show points with no data.)

Figure 52 and Figure 53 show how successful this process was. In Figure 53, the sloped area of the ramp is very clearly seen as separate from the level floor, which means this code will be able to reliably detect ramps. Figure 52 shows that the edges of objects categorized as bumps and obstacles were unaffected by the averaging filter. An object sitting on a level floor will not get blurred and incorrectly classified as a ramp.

In the final step, a threshold was used to separate the floor from ramps. Thus the process is complete and each cell in the map has been classified into one of the five categories.

Results

This method took about 2 seconds to run. The following images (Figure 54 to Figure 62) show several examples of the output of this algorithm, along with the corresponding images from the Kinect's RGB camera.



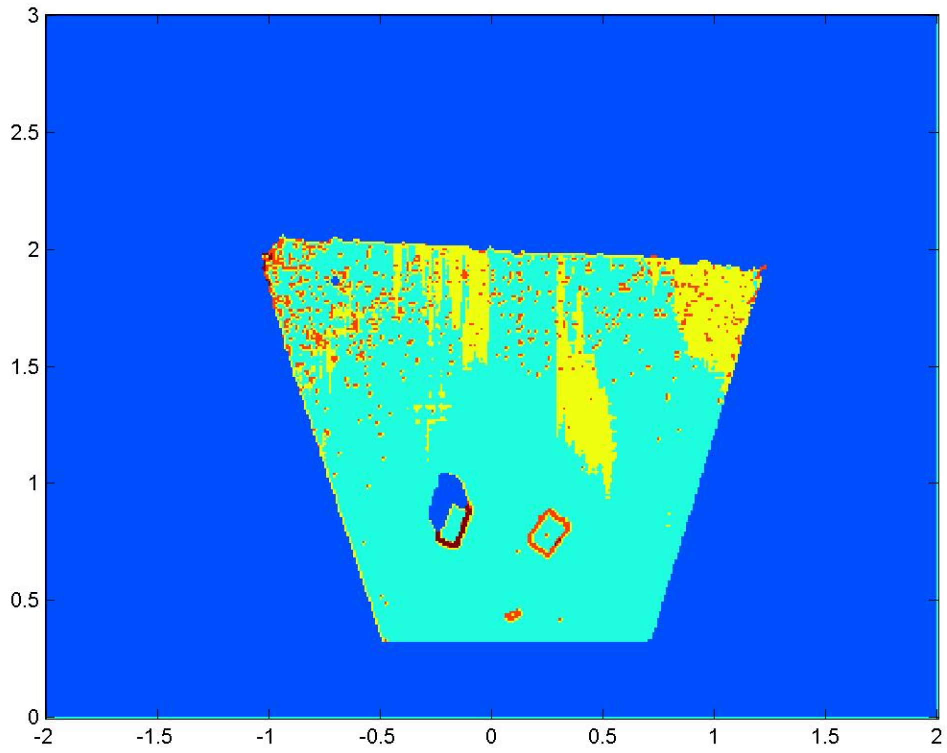
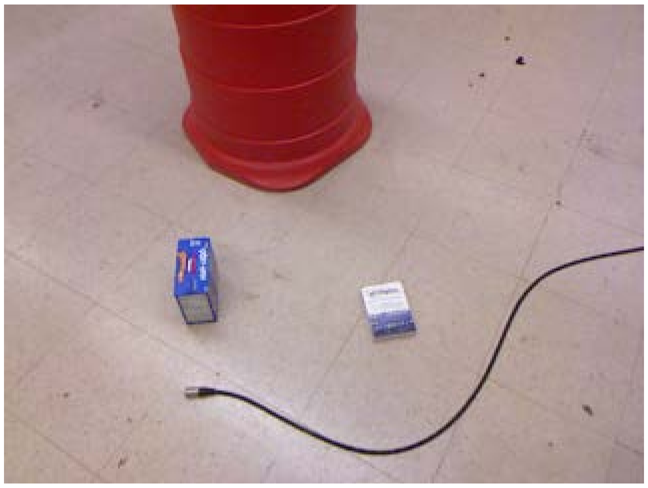


Figure 34: The robot is on a level floor looking at 3 objects. The larger box cannot be run over; it is an obstacle. The smaller box and power cord are small enough to run over; they are classified as bumps. (In each of these images of the map output, the robot position is (0,0), the bottom of the image. Units are meters. Dark blue is unknown space, light blue is level floor, yellow is ramps, orange is bumps, dark red is obstacles.)



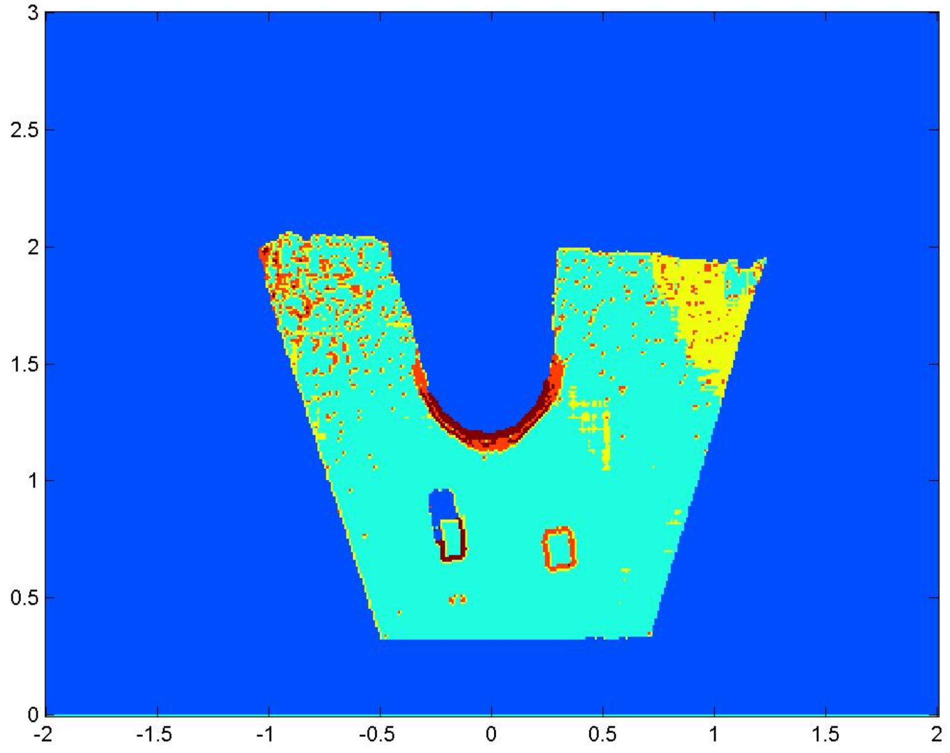
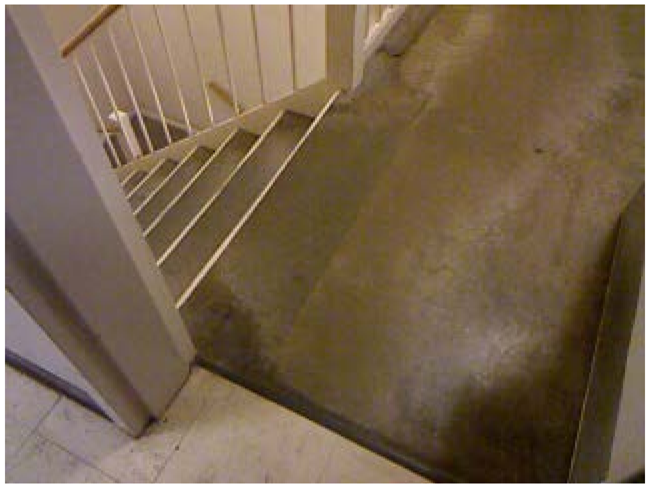


Figure 35: The robot is on a level floor, looking at four objects. The larger box and the orange barrel are obstacles, while the small box and power cord are bumps small enough to run over. (In each of these images of the map output, the robot position is (0,0), the bottom of the image. Units are meters. Dark blue is unknown space, light blue is level floor, yellow is ramps, orange is bumps, dark red is obstacles.)



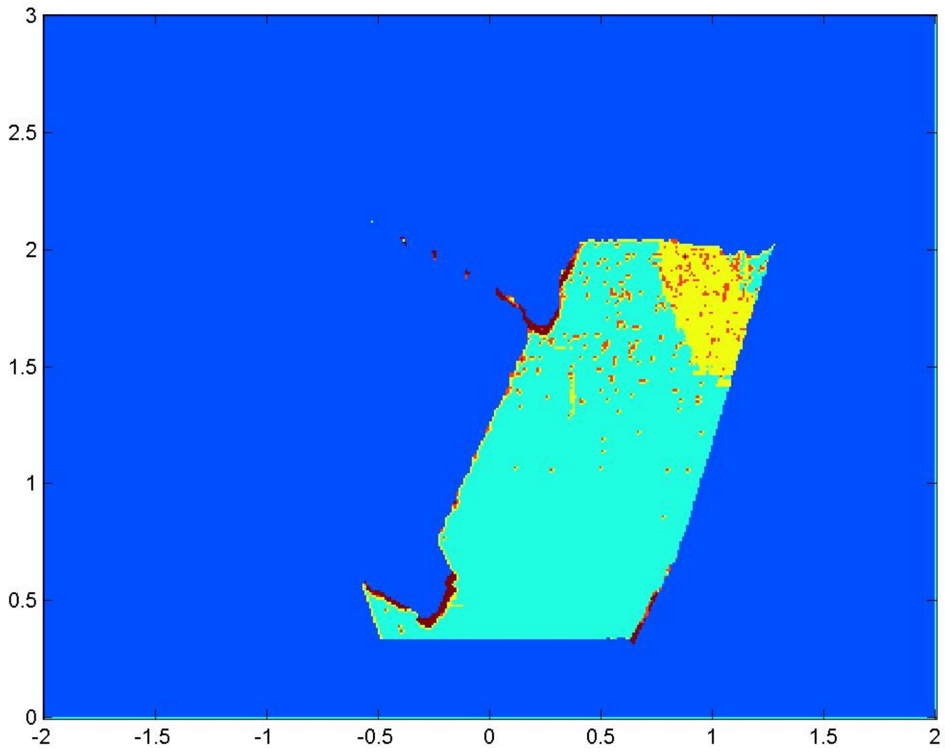
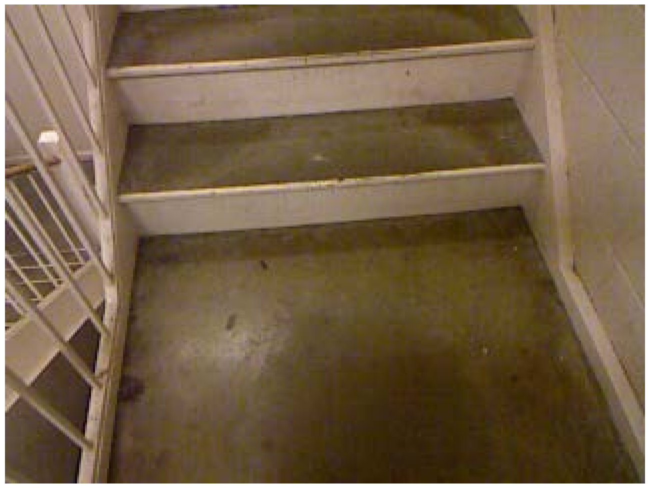


Figure 36: The robot is at the top of a flight of stairs. (In each of these images of the map output, the robot position is (0,0), the bottom of the image. Units are meters. Dark blue is unknown space, light blue is level floor, yellow is ramps, orange is bumps, dark red is obstacles.)



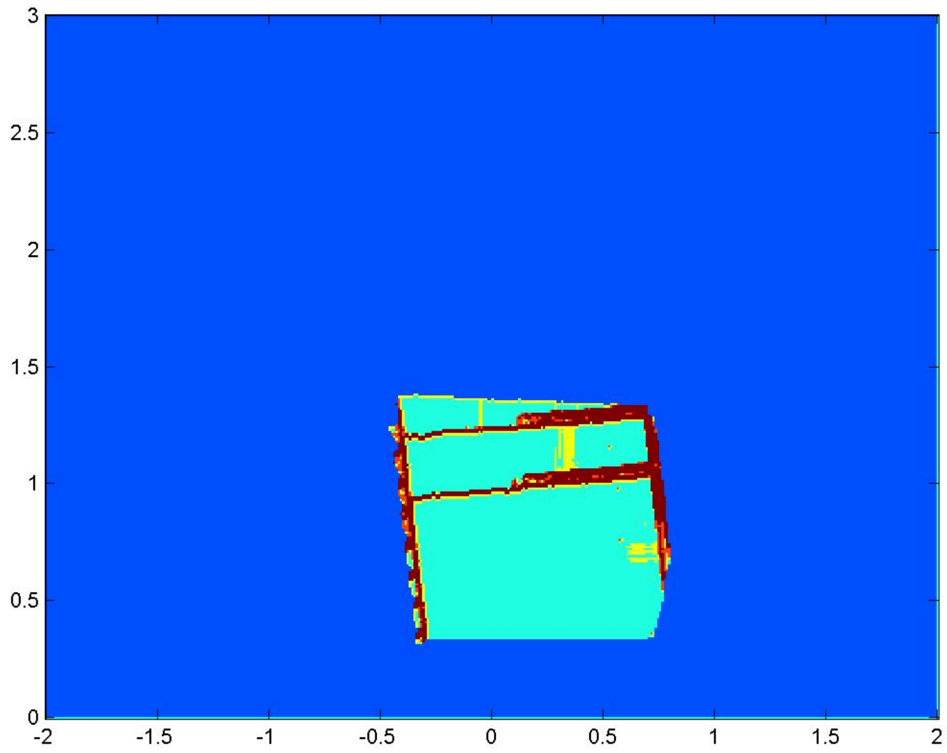
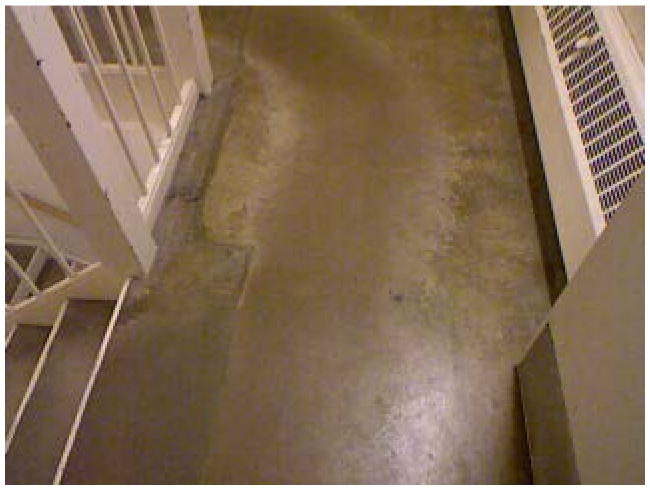


Figure 37: The robot is at the bottom of a flight of stairs. (In each of these images of the map output, the robot position is (0,0), the bottom of the image. Units are meters. Dark blue is unknown space, light blue is level floor, yellow is ramps, orange is bumps, dark red is obstacles.)



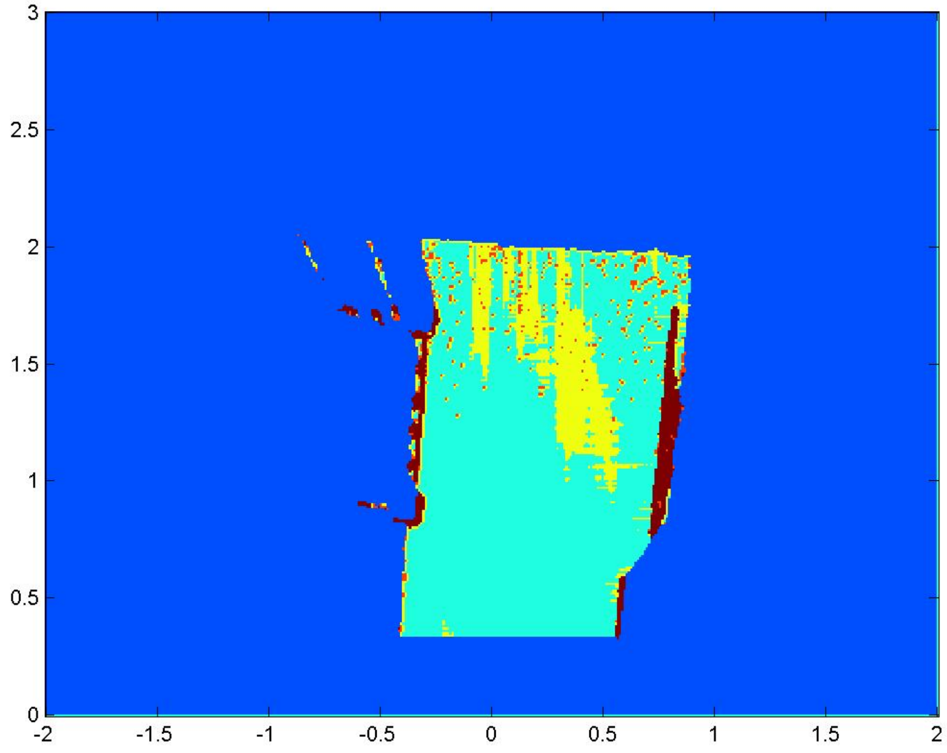
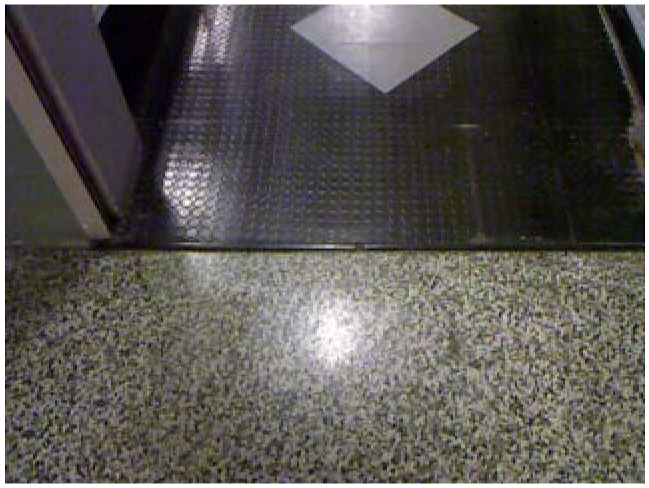


Figure 38: The robot is in a hallway, close to the top of a flight of stairs. (In each of these images of the map output, the robot position is (0,0), the bottom of the image. Units are meters. Dark blue is unknown space, light blue is level floor, yellow is ramps, orange is bumps, dark red is obstacles.)



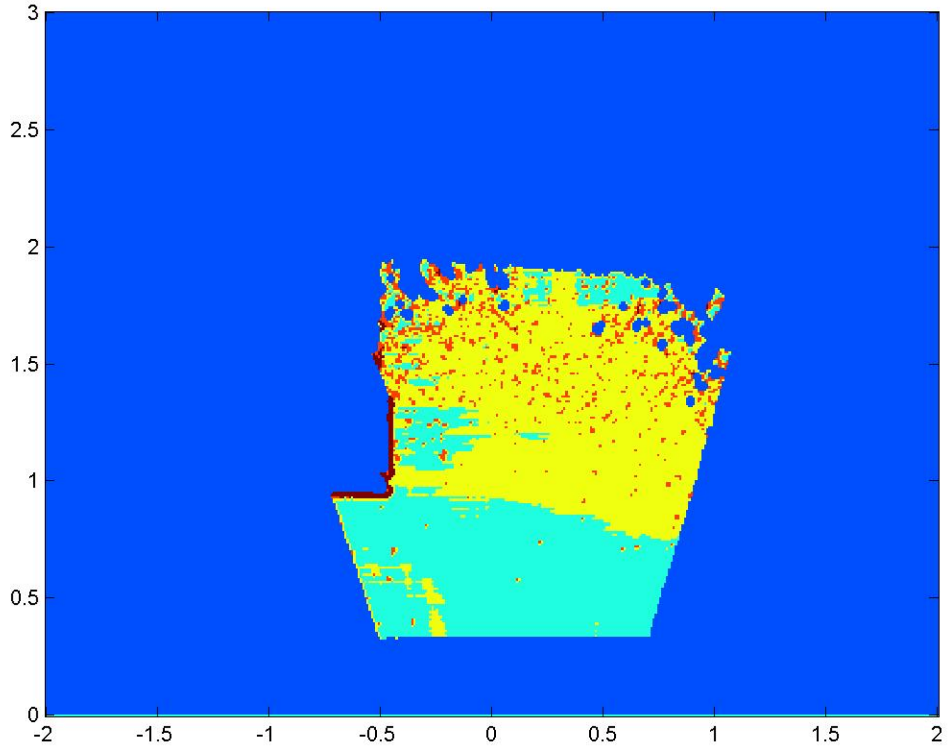
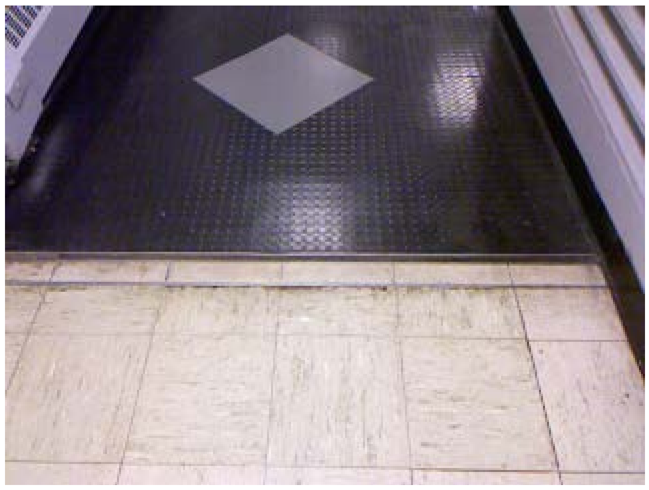


Figure 39: The robot is on a level floor, looking at a ramp that slopes upward. (In each of these images of the map output, the robot position is (0,0), the bottom of the image. Units are meters. Dark blue is unknown space, light blue is level floor, yellow is ramps, orange is bumps, dark red is obstacles.)



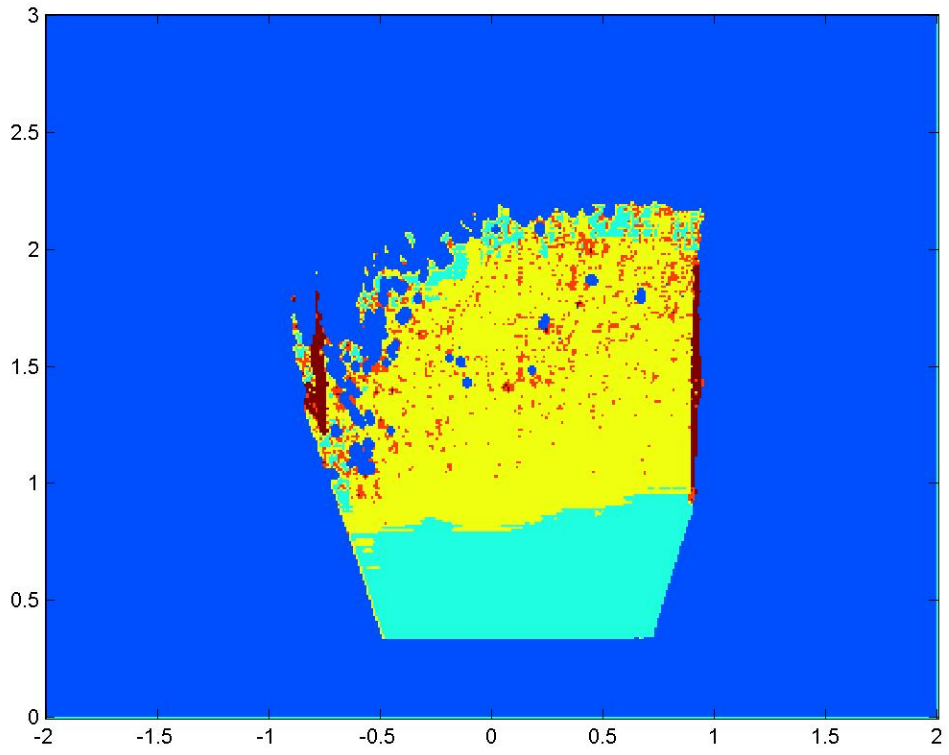
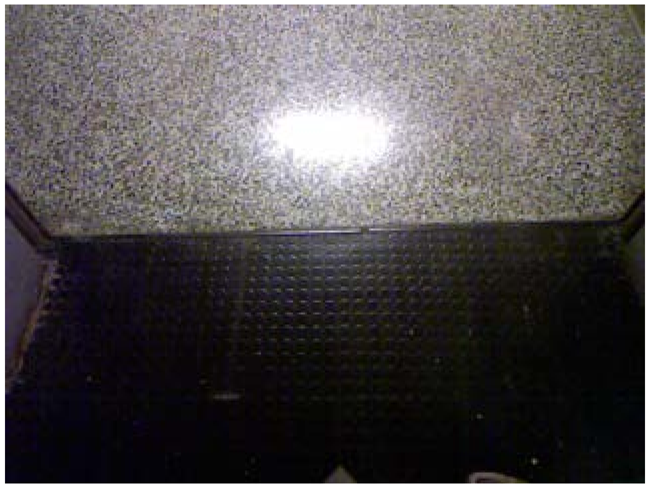


Figure 40: The robot is on level floor, looking at a ramp that slopes downward. (In each of these images of the map output, the robot position is (0,0), the bottom of the image. Units are meters. Dark blue is unknown space, light blue is level floor, yellow is ramps, orange is bumps, dark red is obstacles.)



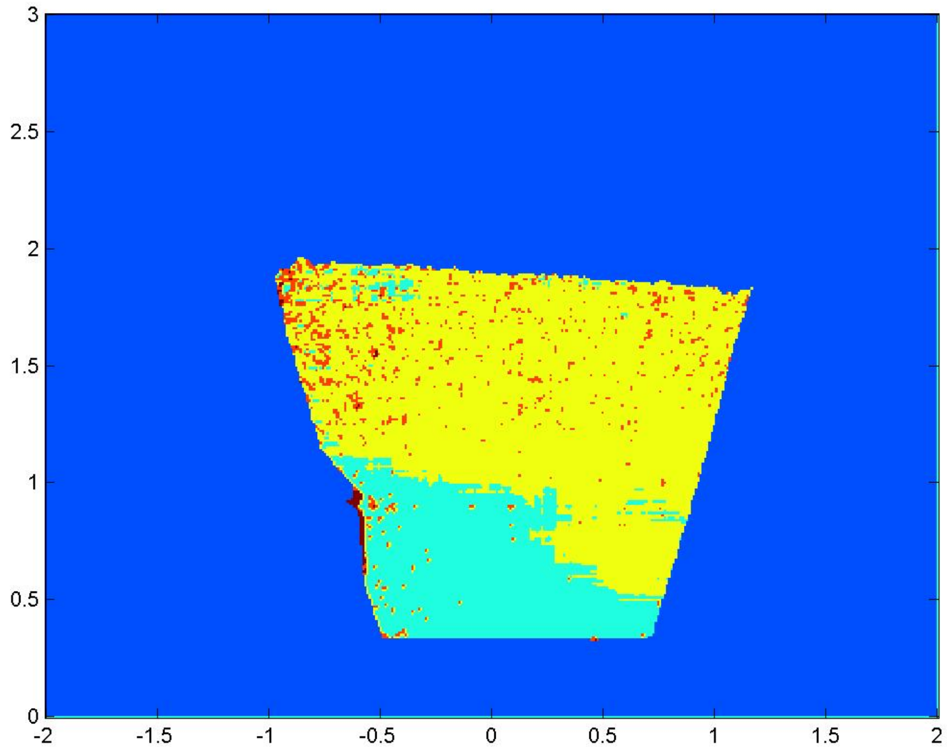
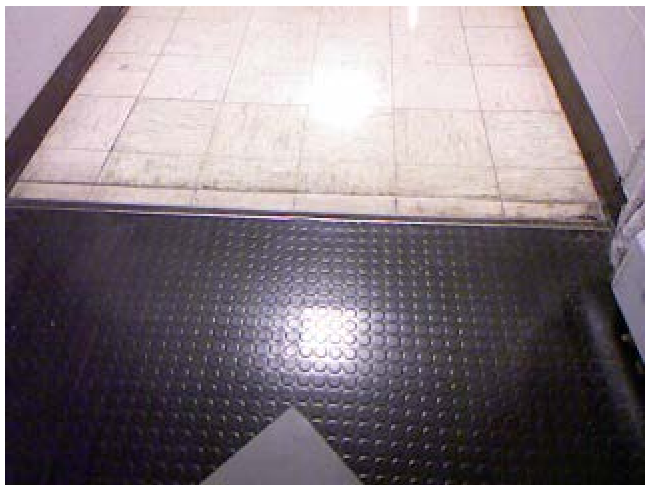


Figure 41: The robot is on a ramp, looking at the level floor at the bottom of the ramp. (In each of these images of the map output, the robot position is (0,0), the bottom of the image. Units are meters. Dark blue is unknown space, light blue is level floor, yellow is ramps, orange is bumps, dark red is obstacles.)



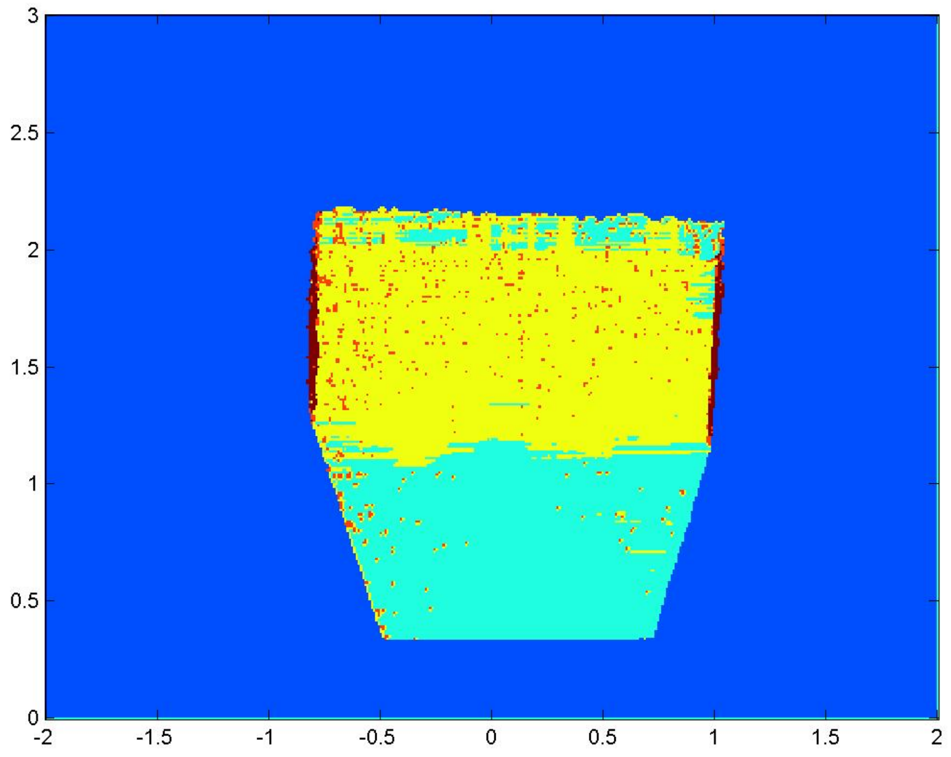


Figure 42: The robot is on a ramp, looking at the level floor at the top of the ramp. (In each of these images of the map output, the robot position is (0,0), the bottom of the image. Units are meters. Dark blue is unknown space, light blue is level floor, yellow is ramps, orange is bumps, dark red is obstacles.)

Figure 54 and Figure 55 show that this code is able to detect both small and large objects on the floor in front of the robot. The algorithm correctly determined that the small box could be run over, but the large box could not. The small box has a height of 2 cm; while it is doable for the wheelchair to run over it, a better option would be to avoid it if possible. It is up to the path planner code to make that decision. Furthermore, these two figures show that the floor itself was correctly identified and colored light blue, with just a little bit of noise in the area farthest from the Kinect sensor.

Figure 56, Figure 57, and Figure 58 deal with stairs. Obviously the robot is not able to travel up or down stairs, and being at the top of a flight of stairs is particularly dangerous. In Figure 56, the area corresponding to the stairs is marked as unknown, presumably because the depth sensor did not return any data for this part of the Kinect's field of view. Because it is marked as unknown, the robot's path planner will not allow the robot to move to that area (and fall down the stairs). Figure 58 shows a similar result. In both of these, the algorithm was able to correctly determine that the stairs were not drivable. This information is essential to the safety of the smart wheelchair's user.

Figure 57 shows the robot at the bottom of a flight of stairs. Interestingly, the horizontal surface of each stair is marked as level floor and therefore drivable. In fact, it would be possible for the robot to drive on an individual stair, if the robot were small enough and had some way of getting there in the first place. Fortunately, the edge of each stair is very clearly marked as an obstacle, so the path planner will know not to go there.

Figure 59, Figure 60, Figure 61, and Figure 62 show how the algorithm is able to very reliably identify ramps. In Figure 59 and Figure 60, the robot is sitting on a level floor, and in Figure 61 and Figure 62 the robot is sitting on the ramp. In each of these examples, the robot's field of view includes a section of level floor and a section of ramp. Notice that the algorithm has correctly detected both ramps sloping up and ramps sloping down. In a practical sense, a different approach may be needed for each type, and this decision is left up to the path planner. Once the path planner receives the data about where the ramp is, it is very easy to determine if the ramp is sloping up or down by simply checking the height data. Alternatively, the path planner may require specific information about the steepness and direction of the ramp, which can also be found by analyzing the height data, perhaps by fitting a plane to it and calculating the normal vector for that plane. Deciding on a specific approach the path planner would use is outside the scope of this project, but the necessary data for making those decisions is contained in the various maps shown here.

Also, it should be noted that in Figure 61 and Figure 62, since the robot is sitting on the ramp, light blue areas are parallel to the plane of the robot, and the yellow areas are sloped, even though the light blue part is actually where the ramp is. This is to be expected, because this algorithm alone has no way of knowing the robot is already on a ramp. When this algorithm is integrated with the rest of the robot's code, the problem can be easily solved by keeping track of transitions that have already been made from a level to sloped surface. Additionally, the robot has an accelerometer, and the data from the accelerometer can later be integrated with the algorithm shown here. The code presented in this thesis only deals with one snapshot of data, so it had no way of knowing the robot was sitting on a ramp. Instead, it reliably identifies planes parallel to the robot's position.

These nine examples show very clearly the effectiveness of this method. However, there is a little noise, particularly in the areas farthest from the Kinect sensor. This noise is not a big concern because the areas immediately in front of the robot had almost no errors, and there are many effective ways to address the noise once the algorithm is implemented on a robot.

For example, once this algorithm is able to run in real time and update the map continuously, it will certainly be able to distinguish noise from actual objects. The errors due to noise would appear and disappear as the map updates, but any objects or features that are truly there will always show up.

Additionally, depending on the specifics of the path planner code, it may choose to ignore any bumps whose size is only a few pixels. This would take care of most of the noise that showed up in the previous images; almost all of the noise errors were cells being marked as bumps when they should not be.

In summary, the method presented in this thesis is effective for classifying different areas into categories based on their drivability. When integrated with the rest of the robot's code, it would be provide valuable information to the path planner and will be very useful for ensuring safety and locating ramps.

Chapter 6: Conclusions and Future Work

Obstacle detection has always been an important aspect of mobile robotics. In the case of a smart wheelchair robot, it is very critical to locate any obstacles or problems with the floor surface itself in order to ensure the safety of the user, and it is also essential to identify ramps that the wheelchair will be able to access. Previous systems for obstacle detection often did not focus on the floor itself; for example, they would not notice when the robot is at the top of a flight of stairs. This is not acceptable for a wheelchair robot; it is far too dangerous. A new method must be developed in order to correctly identify such hazards.

Recently, Microsoft introduced the Xbox Kinect, which provides a point cloud containing three-dimensional depth data. The Kinect is a low-cost sensor which can give high-resolution data; therefore it has huge potential as a sensor in robotics. This thesis addresses the use of the Kinect to classify the floor surface in front of the robot into five categories to indicate drivability: level floor, ramps, bumps, obstacles, and unknown. First the Kinect itself is assessed to find its abilities and limitations, then a method to calculate a transform is shown, and finally an algorithm is produced which would classify different areas.

Overall, the process is successful, and the Kinect proves to be an incredibly valuable sensor. The algorithm presented here runs in about 2 seconds and produces a map that goes to 2 m in front of the robot. This means that theoretically, it could be used on a robot traveling at 1 m/s. In a practical sense, however, it would be unsafe to travel at that speed, because it means obstacles might not be detected before the robot reaches them. A smart wheelchair robot would typically not be going that fast; perhaps it would travel at a top speed of 0.5 m/s. Therefore, the process presented here is certainly applicable to the needs of a smart wheelchair.

This research is significant because though mobile robots can effectively detect obstacles, they typically do not have a sensor to detect features of the floor surface itself. This weakness becomes dangerous when the robot needs to navigate around stairs. In the case of a smart wheelchair robot, where safety is the highest priority, it is essential to have a sensor that can detect stairs or holes in the floor.

Furthermore, the algorithm shown here can locate ramps, which wheelchairs often need to navigate in order to reach their destination. Ramps can be judged by their steepness to determine whether they are safe to travel, and if any special considerations need to be made; for example, the robot may need to reduce its speed, or approach the ramp from a certain angle.

One possible way to implement these results is to put two Kinects on the robot, one facing forward and one facing backward. The backward-facing Kinect is particularly important, because the user of the wheelchair cannot easily see what is behind the wheelchair while backing up.

In future work, this code needs to be integrated with the rest of the code on the robot, especially the localization and path-planning code. The algorithm would be adapted to run continuously and create a map that automatically updates. Currently, the code takes about 2 seconds to process the data and produce the final map, so when the method is integrated with the rest of the robot's code, the map will be able to update every 2 seconds.

Another important future consideration is adapting the thresholds toward the specific needs of the robot. In this thesis, it was assumed that a small box could be run over if it was only 2 cm high, and a threshold value was chosen accordingly; the box was marked as a “bump”. Depending on the characteristics of one’s robot, it may or may not be practical to run over objects of that size, so threshold values can be adjusted accordingly. The same is true for steepness of ramps. If a particular robot is only able to travel up shallow ramps and not steep ones, additional threshold values can be added to the code to mark steeper ramps as non-drivable and ensure the robot operator’s safety.

Furthermore, the robot’s path planner would need to take into account the different features that were classified by this algorithm. A very simple path planner would only avoid obstacles, and treat level floors, ramps, and bumps all as equally drivable. A more complex path planner would evaluate ramps to determine the best speed and angle of approach, and take into account where the robot’s wheels are when driving over bumps. Path-planning is outside the scope of this particular thesis, but by classifying areas according to drivability, the output of this algorithm gives the path planner the information it would need to calculate a safe route from one point to another.

In summary, the results of this thesis suggest that the Kinect may be an effective sensor for mobile robotics, particularly for smart wheelchairs. Though other sensors may be expensive or give data which does not easily lend itself to identifying the plane of the floor, the Kinect is relatively inexpensive and its point cloud data is very well-suited for this task. Algorithms have been presented here to calibrate the Kinect to correct for its angle and height and to classify the features of the floor surface into different categories based on their drivability. When integrated with the rest of the robot’s code, this method will be effective in identifying features of the floor, to ensure the safety of the user.

Appendix A: Comparison of different floors

For several different types of floors, depth information for a small region of the floor was collected and analyzed. Specifically, this region extended from 1.4 m to 2.7 m in front of the robot, and had a width of 1.5 m. This region was chosen because it is small enough that no matter the location of the robot, the robot can be positioned so this particular region of interest is not obstructed by any other objects. Also, the data closest to the robot is most reliable. The region of interest is outlined in red in Figure 9:

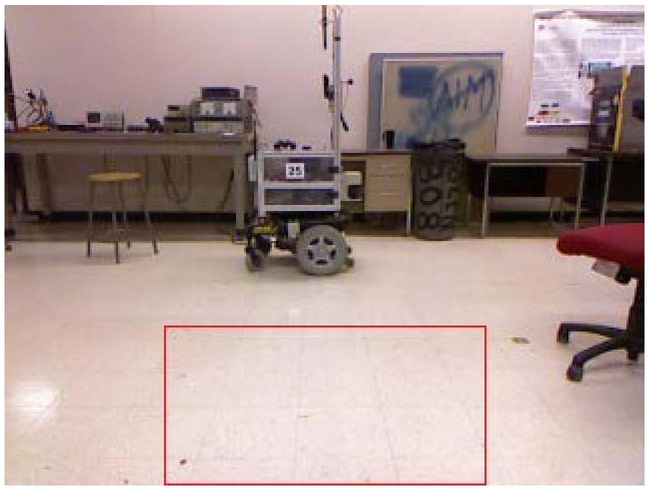


Figure 43: The red box shows the region that was analyzed.

For every point in the region of interest, the coordinates were rotated by 13 degrees to compensate for the angle at which the Kinect was mounted, and the vertical distance from the Kinect to the floor was found. (The angle of the Kinect was known to be approximately 13 degrees. A more accurate method for calculating this angle is described in Chapter 4; in this case the slight error in angle is negligible when comparing different types of surfaces.) All of the vertical distances were then analyzed to find the average and standard deviation for the points in each type of floor surface. Theoretically, the result should be the same for each floor location tested. The purpose of this test was to see if there are differences in the data returned for different surfaces.

The results for running this test are shown in Table 1:

Location	1	2	3	4	5	6	7	8	9	10	11
Mean (mm)	1019	1014	434	1003	1013	1026	1001	1006	1009	999	1023
Std dev (mm)	4.91	4.66	500	6.87	5.89	4.80	5.61	93.69	5.39	5.37	4.74

Table 3: Statistics on the data returned for different types of floors.

The 11 locations are shown in Figures 10 to 20:

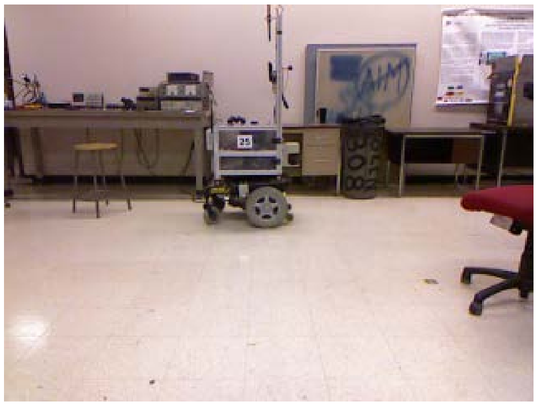


Figure 44: Location 1

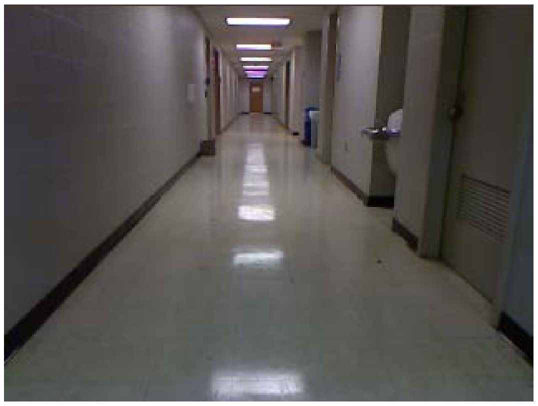


Figure 45: Location 2

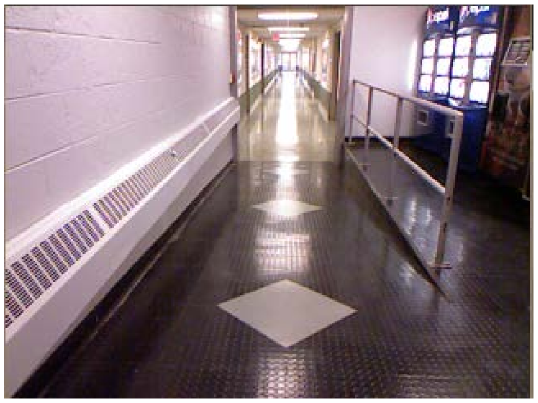


Figure 46: Location 3

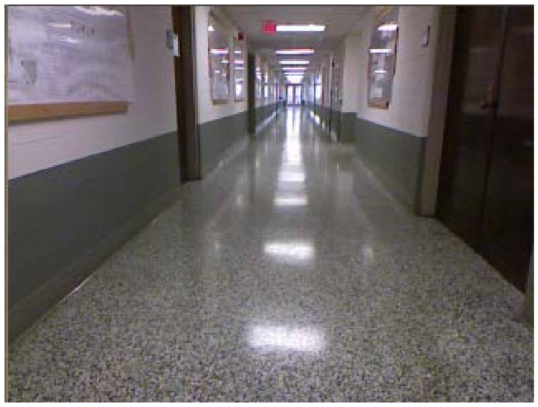


Figure 47: Location 4

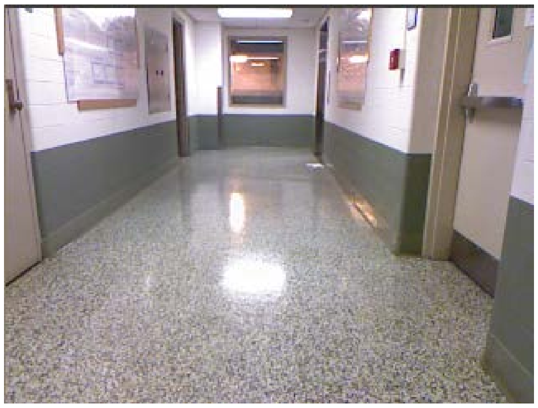


Figure 48: Location 5

Figure 49: Location 6

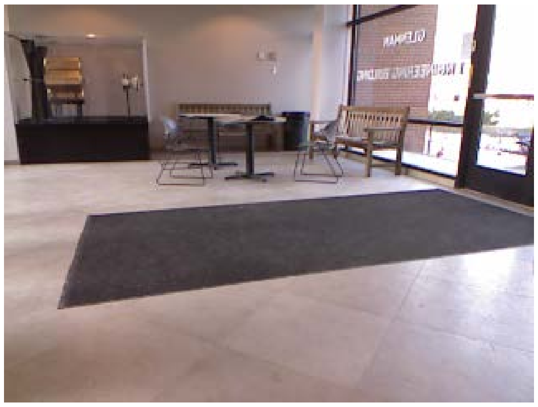


Figure 50: Location 7

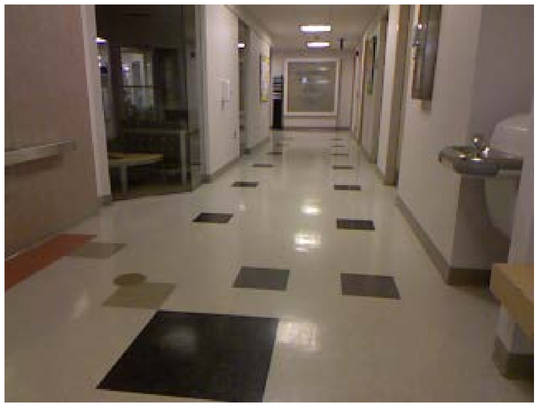


Figure 51: Location 8

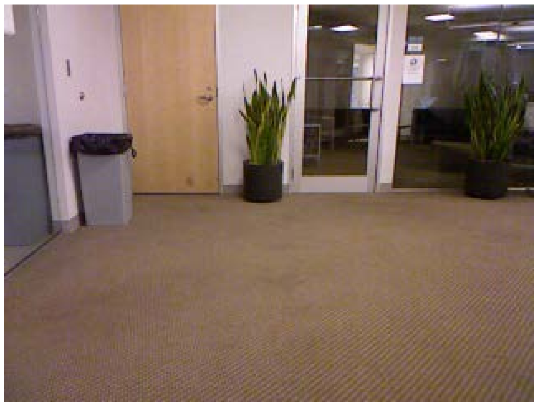


Figure 52: Location 9

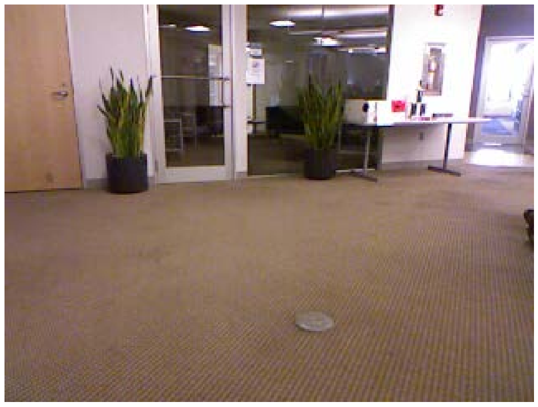


Figure 53: Location 10

Figure 54: Location 11

Table 1 shows that for almost all of the different locations, which had variations in color and texture of the floor itself, as well as variations in lighting, the mean height detected by the Kinect was about 1000 mm, with a standard deviation of about 5 mm. In most of the locations, the Kinect is able to accurately gather data. There were 2 exceptions: locations 3 and 8.

The depth maps for locations 3 and 8 show the reason for the huge error.

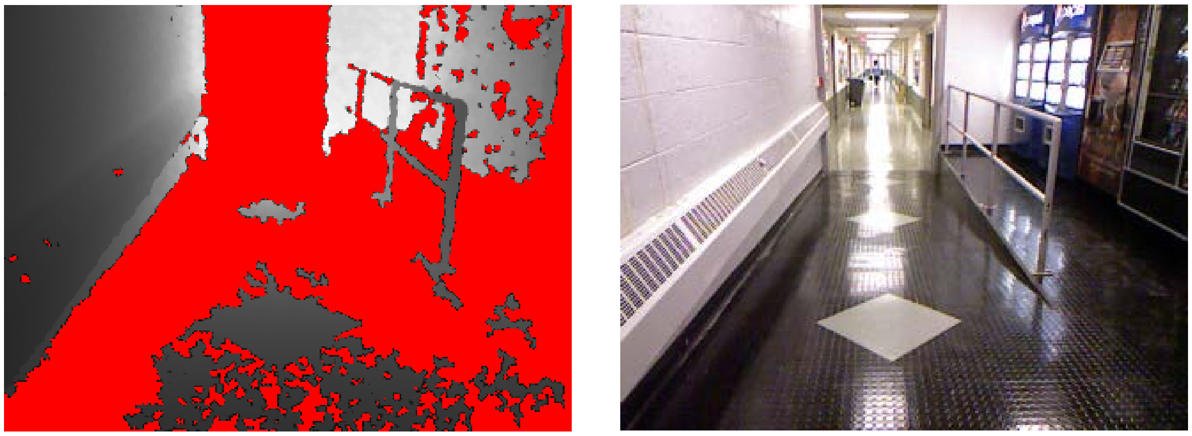


Figure 55: Depth image (left) and RGB image (right) for location 3. The areas with no data have been marked in red on the depth image.

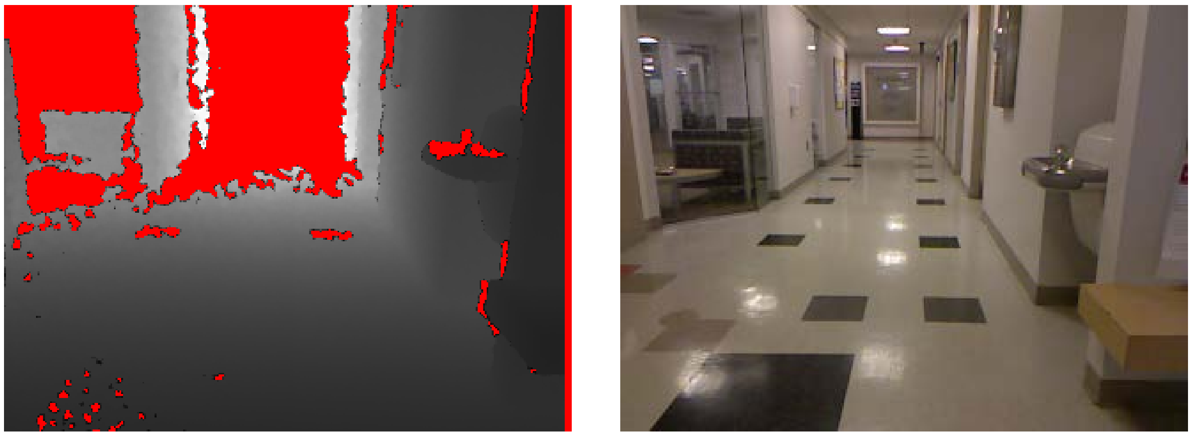


Figure 56: Depth image (left) and RGB image (right) for location 8. The areas with no data have been marked in red on the depth image.

Figure 21 and Figure 22 show two conditions where the Kinect has difficulty collecting depth information. When the angle of the Kinect is 13 degrees, black floors present problems.

Appendix B: Selection of points for calibration

The following steps were used:

1. **For each point in the depth image, calculate the slope and determine an angle by taking the arctangent.**
2. **Sort all of the angles into bins in order to find the most common angle; this gives a good estimate for the angle of the Kinect from the horizontal.**
3. **For each (x, y, z) point returned by the Kinect, rotate by the angle determined in the previous step. Take the z coordinates of the rotated points and sort them into bins to find the most common height. This gives a good estimate for the height of the Kinect.**
4. **Take all the points whose height is close to the calculated height and fit a plane to them. This will give a more accurate result, taking into account roll as well as pitch.**
5. **Use the equation of the plane to find a rotation matrix.**

Estimate the slope at each data point

First, code was written that could determine the angle by finding the floor surface and examining its slope. To do this, the robot was placed such that the Kinect's field of view was largely comprised of the floor. The information from the depth map was analyzed. Each depth value was compared to one several pixels below it, and the slope between them was calculated. This slope value indicates the slope of whatever object was at that position. Theoretically, every point representing the floor should have the same slope, the slope of the floor with respect to the Kinect's line of sight. Once all the slopes are calculated, an arctangent is used to calculate the angle, and they are sorted into bins corresponding to different angles.

In this method, one particularly important concern was noise and smoothing. It was found that when the slope was calculated using pixels that were very close to each other, the results were very noisy and yielded errors that could be as high as 2 degrees. Therefore, several different values for the parameter "smoothConstant" were tested; this parameter indicates the distance between each pixel and the one that it was compared with to find the slope. The results are shown in Figures 38 to 41. In these images, the Kinect's field of view contains a large area of the floor, and the side of a chair can be seen on the right. Every point has a color indicating the angle that was determined by taking the slope at that point. In each of the figures, a different value for the parameter "smoothConstant" is used. For example, in Figure 38, smoothConstant=5, meaning that the slope is computed between a given pixel and the data point 5 pixels above it. (Also, as a result of this method, there is no data for the 5 rows at the top of the image.)

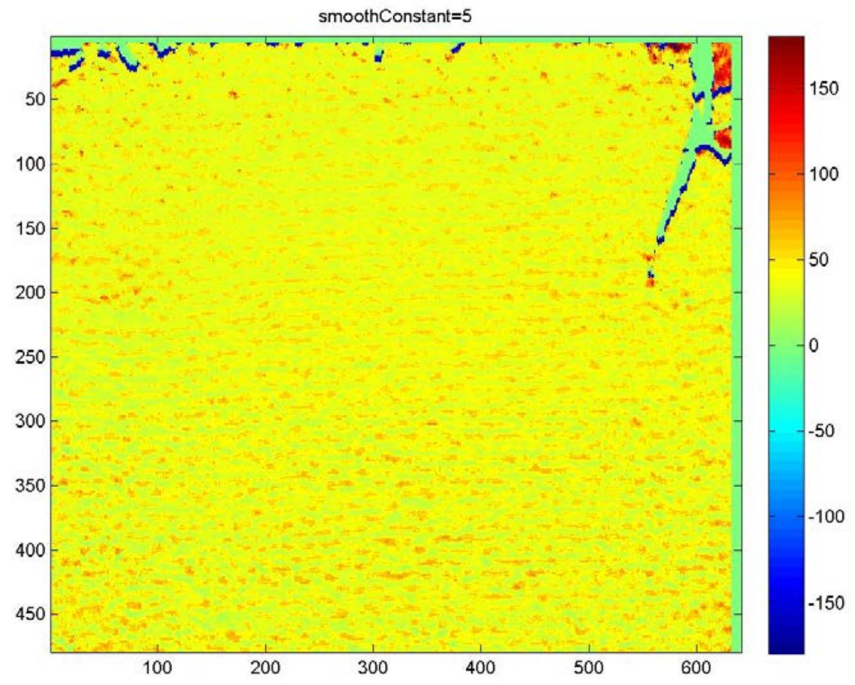


Figure 57: smoothConstant=5. The result for the floor surface is uneven and noisy. The color indicates the angle calculated at each point.

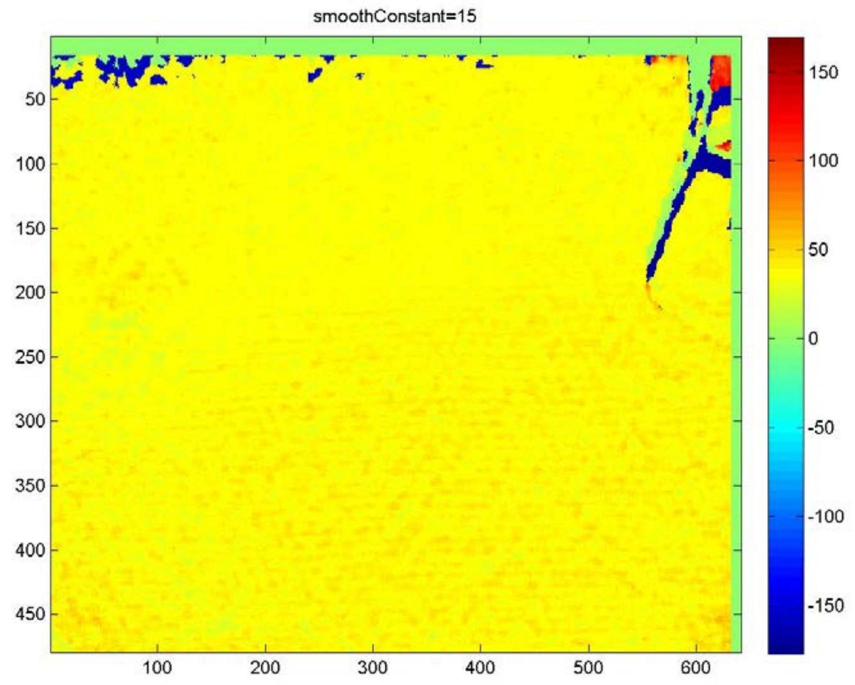


Figure 58: smoothConstant=15. The color indicates the angle calculated at each point.

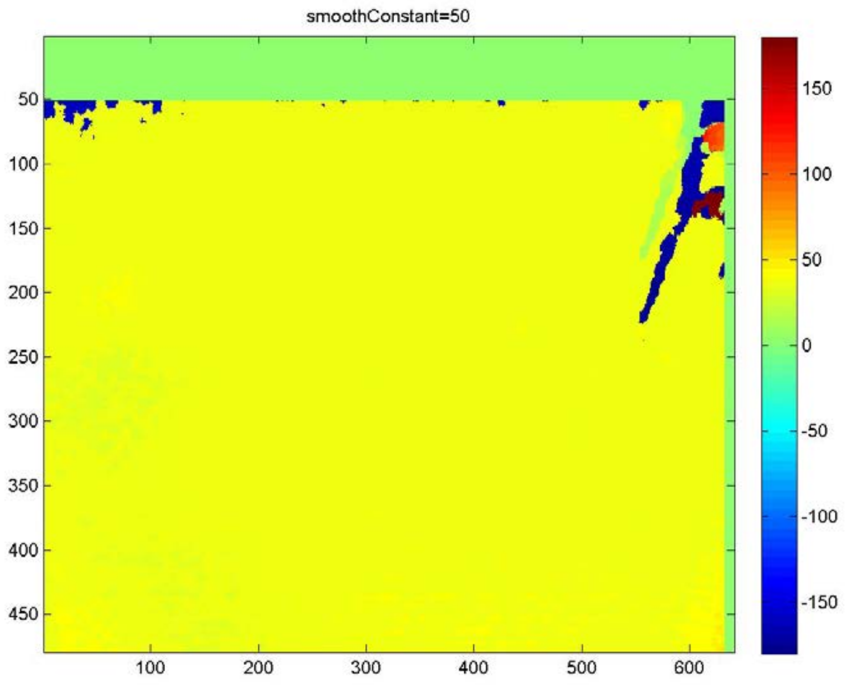


Figure 59: smoothConstant=50. The color indicates the angle calculated at each point.

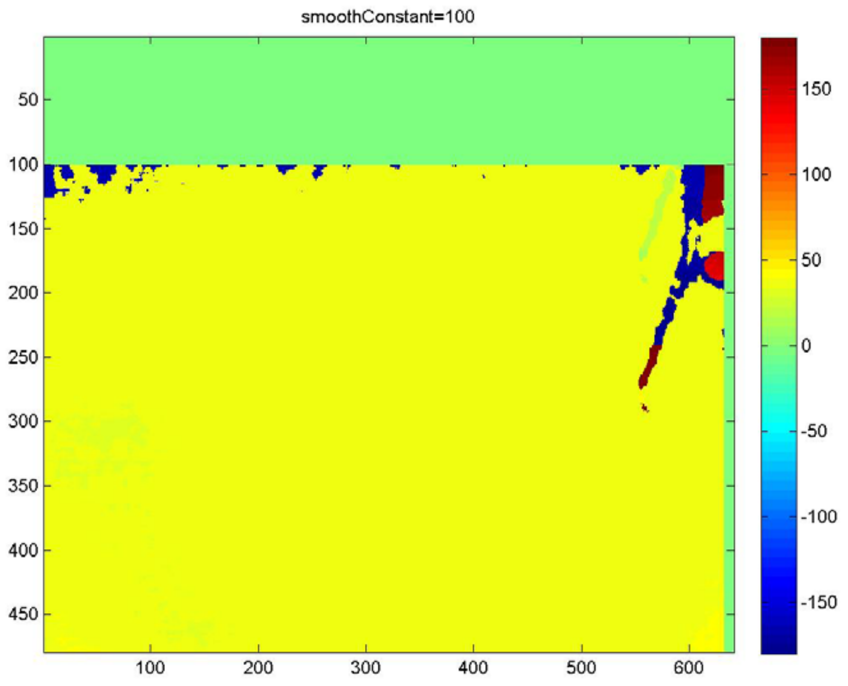


Figure 60: smoothConstant=100. The color indicates the angle calculated at each point.

In each of these cases, the angle and height of the Kinect were computed, and this was compared to nominal result that can be obtained by fitting a plane to the data and extracting an angle and height from that plane (this process is explained in more detail later). Table 2 shows the results:

	Plane fit result	sm const= 5	Sm const= 15	Sm const= 50	Sm const= 100
Angle (degrees)	36.9	38.9	35.6	36.2	36.7
Height (mm)	1069	1106	1055	1057	1067

Table 4: Comparing different values for the parameter, to find the result closest to the best-fit plane.

Therefore, it was decided that 100 was a good choice for smoothConstant. Clearly the above images show that the output is very uniform when 100 is used, and therefore an accurate average can be obtained for the slope of the floor, which in turn tells the angle of the Kinect. The table shows that choosing 100 for smoothConstant yields a result very close to the theoretically correct answer. Additionally, this method is unaffected by other objects in the camera's field of view; Figure 42 shows the result when there is a chair placed in front of the Kinect.

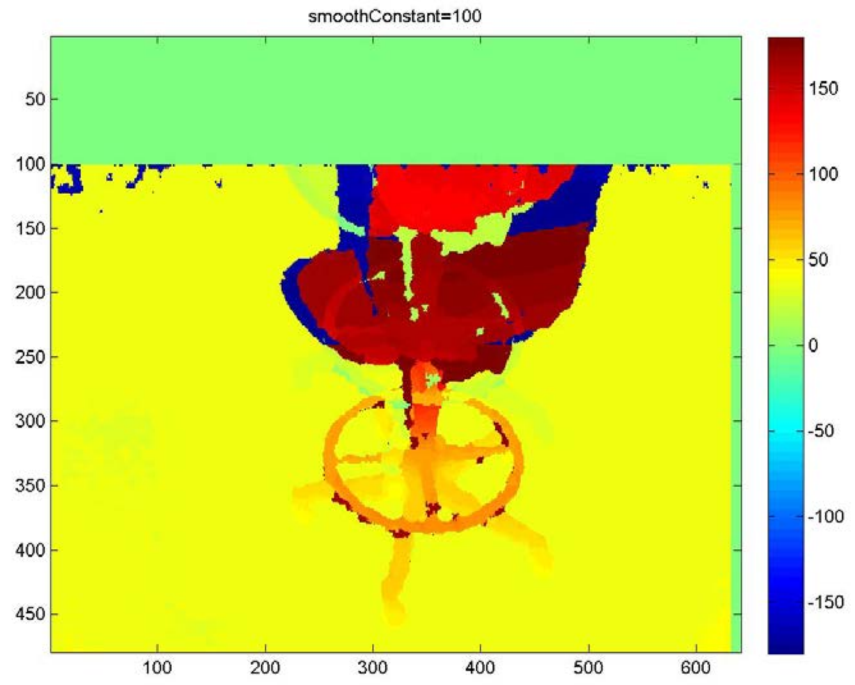


Figure 61: Angle can still be found accurately when an object is in the field of view

Because Figure 42 shows a very large number of points marked in yellow, even when there is a large object in the Kinect's field of view, the algorithm will still be able to give an accurate estimate for the slope of the floor. The strength of this method is that it completely ignores outliers and only concentrates on a range of a few degrees which contain the most points, as will be discussed in the next section.

Obtain an estimate for the angle of the Kinect

All of the values for angles were sorted into bins if they fell between 0 and 90 degrees. Anything outside of this range was assumed to be an error. Figure 43 shows a sample histogram output when the actual angle is about 36 degrees:

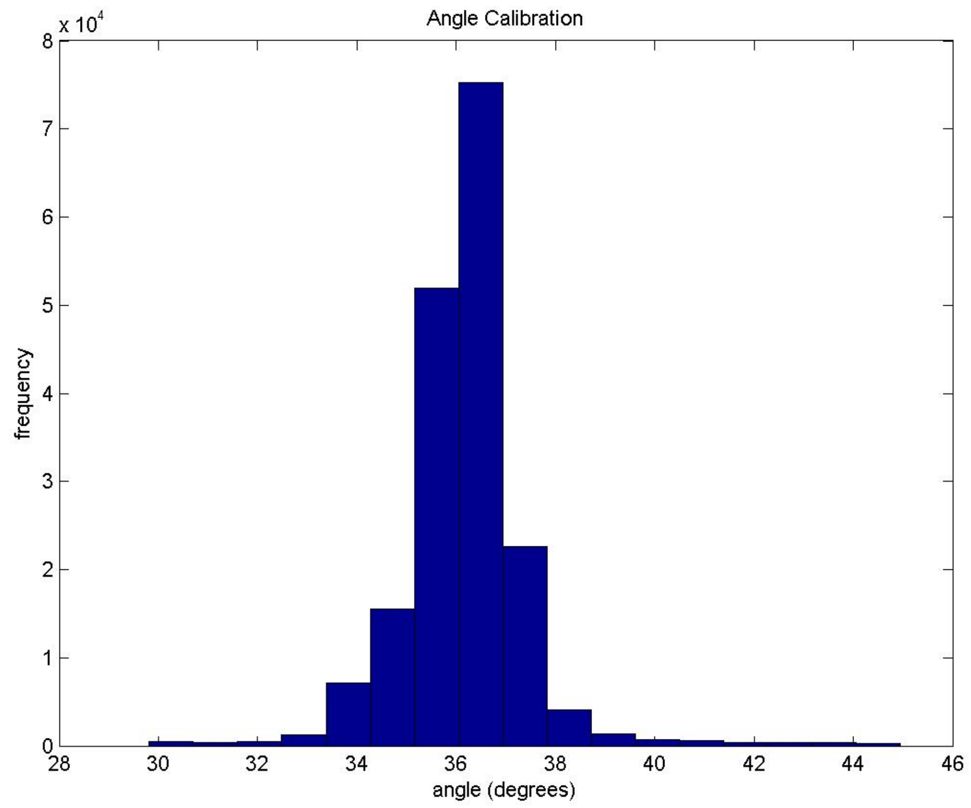


Figure 62: Angles are sorted into bins. The bin with the the highest count gives a good approximation for the angle of the Kinect from the horizontal.

(It should be noted that the algorithm has bins for angles all the way from 0 to 90. The histogram in Figure 43 only goes from 30.26 to 44.5 degrees because beyond that, all of the bins have a very low count.)

In Figure 43, it is obvious that the angle of the floor surface with respect to the Kinect's line of sight is around 36 degrees, since the bins for "36.49" and "35.6" have significantly more points than any of the other bins. The next step in the algorithm is to identify the bin with the most points, and also the 2 bins adjacent to it. In the above example, this means that the bins for 35.6, 36.49, and 37.38 would be selected. It is expected that all of the points in these bins belong to the floor. Next, all the values represented by points within those bins are averaged. This final result is a good approximation (accurate within 0.5 degrees) for the angle of the Kinect. In this example, the final angle was calculated to be 36.7 degrees.

Calculate the Kinect's height

Next, the points that are believed to belong to the floor must be found and stored in an array so a plane can be fit to them. This will give the equation for the plane of the floor. To find those particular points, first all the points are rotated by the angle that was found in the previous step. This corrects for the angle of the Kinect. Then the z-coordinates (height) for each transformed point are sorted into bins, with the most populous bin giving a reasonable estimate for the height of the Kinect.

Check the accuracy at this step

At this point, an angle and height have been calculated, and the accuracy should be examined to see if the result is good enough, or if it would be helpful to go a few steps farther and incorporate a plane-fitting process. To check the accuracy of this method, it was compared to a method which involved fitting a plane to all the data points in the point cloud. A simple plane fit would only work under the condition that all the points in the Kinect's field of view belonged to the floor, so that is how the accuracy was tested. The 3-dimensional coordinates of the points were input and the equation for the plane was found. Then the root-mean-squared error was calculated to determine how well that plane fit the data (this is called rms1 in the table below). Next, the rms error from the algorithm that was just described, which finds angle and height, was calculated (this one is rms2). Then, an angle and height were extracted from the plane that was fit to the data. This is because fitting a plane gives more degrees of freedom than just finding an angle and height, but at this point in the algorithm, only a single angle and height have been calculated. It is assumed that the Kinect has one axis almost parallel to the ground. The rms error from this new angle and height is calculated (this is rms3), and the 3 values for rms error were compared, as shown in Table 3.

	Rms1 (mm)	Rms2 (mm)	Rms3 (mm)
13 degree Kinect	8.2	16.6	13.8
40 degree Kinect	4.2	9.8	8.7

Table 5: Evaluating the rms error calculated by three different methods.

Table 3 shows that the rms2 and rms3 are about the same, indicating that the angle and height that were calculated are very close to the "ideal" angle and height extracted from the best-fit plane.

However, there is still room for improvement. Using a best-fit plane could potentially halve the error.

Best-fit plane

Therefore, the next step is to take all the points whose heights were determined to be in the 3 bins corresponding to the height of the floor, and fit a plane to them. The plane gives a more accurate result because it can take into account rotation by multiple angles. Though large matrices can be inconvenient and computationally intensive, this calibration process only needs to be performed one time, so the slowness of the algorithm is not a problem.

Bibliography

[1] M. Wasif, "Design and implementation of autonomous Lawn-Mower Robot controller," in *2011 7th International Conference on Emerging Technologies (ICET)*, 2011.

[2] C. Ye, S. Ma and B. Li, "Design and Basic Experiments of a Shape-shifting Mobile Robot for Urban Search and Rescue," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006.

[3] Q. Chen, U. Ozguner and K. Redmill, "Ohio State University at the 2004 DARPA Grand Challenge: developing a completely autonomous vehicle," *Intelligent Systems, IEEE,* vol. 19, no. 5, pp. 8-11, Sept-Oct 2004.

[4] G. Bourhis, O. Horn, O. Habert and A. Pruski, "An autonomous vehicle for people with motor disabilities," *Robotics & Automation Magazine, IEEE,* vol. 8, no. 1, pp. 20-28, Mar 2001.

[5] H. Chow and Y. Xu, "Learning Human Navigational Skill for Smart Wheelchair in a Static Cluttered Route," *IEEE Transactions on Industrial Electronics,* vol. 53, no. 4, pp. 1350-1361, June 2006.

[6] A. Filipescu, I. Susnea and G. Stamatescu, "Distributed system of mobile platform obstacle avoidance and control as robotic assistant for disabled and elderly," in *IEEE International Conference on Control and Automation*, 2009.

[7] R. Simpson, D. Poirot and F. Baxter, "The Hephaestus Smart Wheelchair system," *IEEE Transactions on Neural Systems and Rehabilitation Engineering,* vol. 10, no. 2, pp. 118-122, June 2002.

[8] Y. Touati, H. Aoudia and A. Ali-Cherif, "Intelligent wheelchair localization in wireless sensor network environment: A fuzzy logic approach," in *2010 5th IEEE International Conference on Intelligent Systems (IS)*, 2010.

[9] M. Niitsuma, T. Ochi, M. Yamaguchi and H. Hashimoto, "Design of Interaction for simply operating smart electric wheelchair in Intelligent Space," in *2011 4th International Conference on Human System Interactions (HSI),* 2011.

[10] K. Miyazaki, M. Hashimoto, M. Shimada and K. Takahashi, "Guide following control using laser range sensor for a smart wheelchair," in *ICCAS-SICE*, 2009.

[11] B. Wei, J. Gao, K. Li, Y. Fan, X. Gao and B. Gao, "Indoor mobile robot obstacle detection based on linear structured light vision system," in *IEEE International Conference on*

Robotics and Biomimetics, 2009.

[12] X. Wang, K. Ban and K. Ishii, "Estimation of mobile robot ego-motion and obstacle depth detection by using optical flow," in *IEEE International Conference on Systems, Man and Cybernetics*, 2009.

[13] M. Bai, Y. Zhuang and W. Wang, "Stereovision based obstacle detection approach for mobile robot navigation," in *International Conference on Intelligent Control and Information Processing (ICICIP)*, 2010.

[14] M. Vincze, W. Wohlkinger, S. Olufs, P. Einramhof and R. Schwarz, "Towards Bringing Robots into Homes," in *2010 41st International Symposium on Robotics (ISR) and 2010 6th German Conference on Robotics (ROBOTIK)*, 2010.

[15] "Xbox Kinect review, specification & price," 12 February 2011. [Online]. Available: http://www.techibuzz.com/xbox-360-kinect-review/. [Accessed 15 December 2011].

[16] A. Vance, "With Kinect, Microsoft Aims for a Game Changer," 23 October 2010. [Online]. Available: http://www.nytimes.com/2010/10/24/business/24kinect.html?_r=1. [Accessed 15 December 2011].

[17] R. El-laithy, J. Huang and M. Yeh, "Study on the use of Microsoft Kinect for robotics applications," in *IEEE/ION Position Location and Navigation Symposium (PLANS)*, 2012.

[18] P. Rakprayoon, M. Ruchanurucks and A. Coundoul, "Kinect-based obstacle detection for manipulator," in *IEEE/SICE International Symposium on System Integration (SII)*, 2011.

[19] J. L. Raheja, A. Chaudhary and K. Singal, "Tracking of Fingertips and Centers of Palm Using KINECT," in *Third International Conference on Computational Intelligence, Modelling and Simulation (CIMSiM)*, 2011.

[20] P. Benavidez and M. Jamshidi, "Mobile robot navigation and target tracking system," in *2011 6th International Conference on System of Systems Engineering (SoSE)*, 2011.

[21] M. Fallon, H. Johannsson and J. Leonard, "Efficient scene simulation for robust monte carlo localization using an RGB-D camera," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2012.

[22] J.-W. Kim, T.-H. Kim and K.-H. Jo, "Traffic road line detection based on the vanishing point and contour information," in *Proceedings of SICE Annual Conference*, 2011.

[23] Y. Guo, V. Gerasimov and G. Poulton, "Vision-Based Drivable Surface Detection in Autonomous Ground Vehicles," in *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2006.

[24] U. Rasheed, M. Ahmed, S. Ali, J. Afridi and F. Kunwar, "Generic vision based algorithm for driving space detection in diverse indoor and outdoor environments," in *International Conference on Mechatronics and Automation (ICMA)*, 2010.

[25] S. Mostafavi, A. Samadi, H. Pourghassem and J. Haddadnia, "Road boundary extraction under nonuniform light condition using coordinate logic filters," in *IEEE International Conference on Computational Intelligence and Computing Research (ICCIC)*, 2010.

[26] T.-C. Dong-Si, D. Guo, C. H. Yan and S. H. Ong, "Extraction of shady roads using intrinsic colors on stereo camera," in *IEEE International Conference on Systems, Man and Cybernetics*, 2008.

[27] P. Abeles, "Robust local localization for indoor environments with uneven floors and inaccurate maps," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2011.

[28] K. Suwanratchatamanee, M. Matsumoto and S. Hashimoto, "A simple tactile sensing foot for humanoid robot and active ground slope recognition," in *IEEE International Conference on Mechatronics*, 2009.

[29] T. Schamm, M. Strand, T. Gumpp, R. Kohlhaas, J. Zollner and R. Dillmann, "Vision and ToF-based driving assistance for a personal transporter," in *International Conference on Advanced Robotics*, 2009.

[30] Y. Shin, C. Jung and W. Chung, "Drivable road region detection using a single laser range finder for outdoor patrol robots," in *IEEE Intelligent Vehicles Symposium (IV)*, 2010.

[31] C. Guo, W. Sato, L. Han, S. Mita and D. McAllester, "Graph-based 2D road representation of 3D point clouds for intelligent vehicles," in *IEEE Intelligent Vehicles Symposium (IV)*, 2011.

[32] C. Ye, "Polar Traversability Index: A Measure of terrain traversal property for mobile robot navigation in urban environments," in *IEEE International Conference on Systems, Man and Cybernetics*, 2007.

[33] M. Asada, "Building a 3D world model for mobile robot from sensory data," in *IEEE International Conference on Robotics and Automation*, 1988.

[34] O. Gallo, R. Manduchi and A. Rafii, "Robust curb and ramp detection for safe parking using the Canesta TOF camera," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2008.

[35] C. Ye and J. Borenstein, "A method for mobile robot navigation on rough terrain," in *IEEE International Conference on Robotics and Automation*, 2004.

[36] M. Hebert, "Outdoor scene analysis using range data," in *IEEE International Conference on Robotics and Automation*, 1986.

[37] D. Goldgof, T. Huang and H. Lee, "A curvature-based approach to terrain recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence,* vol. 11, no. 11, pp. 1213-1217, 1989.

[38] G. Sithole and G. Vosselman, "Automatic structure detection in a point-cloud of an urban landscape," in *2nd GRSS/ISPRS Joint Workshop on Remote Sensing and Data Fusion over Urban Areas,* 2003.

[39] E. Perko, "Precision Navigation for Indoor Mobile Robotics," *Case Western Reserve University, EECS Department Masters Thesis,* 2013.

[40] C. Rockey, "Low-Cost Sensor Package for Smart Wheelchair Obstacle Avoidance," *Case Western Reserve University, EECS Dept. Masters Thesis,* 2012.

[41] J. Fish, "Robotic Tour Guide Platform," *Case Western Reserve University, EECS Dept. Masters Thesis,* 2012.