Determining Navigability of Terrain Using Point Cloud Data

Stephanie Cockrell, Gregory Lee, *Member IEEE*, Wyatt Newman, *Senior Member IEEE*
Case Western Reserve University
Cleveland, Ohio, United States

***Abstract*-- This paper presents an algorithm to identify features of the navigation surface in front of a wheeled robot. Recent advances in mobile robotics have brought about the development of smart wheelchairs to assist disabled people, allowing them to be more independent. These robots have a human occupant and operate in real environments where they must be able to detect hazards like holes, stairs, or obstacles. Furthermore, to ensure safe navigation, wheelchairs often need to locate and navigate on ramps. The algorithm is implemented on data from a Kinect and can effectively identify these features, increasing occupant safety and allowing for a smoother ride.**

***Keywords*—mobile robotics, obstacle detection, drivable surfaces, Kinect**

I. INTRODUCTION

Previous work in robotics has been focused on finding large obstacles blocking an assumed navigable path, but not looking at the navigation surface itself. However, this approach is not adequate to ensure the safety of a smart wheelchair operator. This paper discusses an approach based on taking gradients from depth data returned by a Kinect, mounted on the front of the robot and pointed down toward the floor.

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 [1]. 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 [2] or Canny and Hough transforms [3]. Progress has also been made using image segmentation techniques and logical operations [4]. 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 [5].

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.

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 [6] 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 [7]. Furthermore, the height information can be subtracted in order to find the gradients of the surrounding area [8] [9] [10]. 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 paper.

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. In the field of mobile robots specifically, the Kinect was found to be an effective tool for target tracking and navigation [11] [12]. It was even found to be useful for localization and mapping for a robotic wheelchair [13].

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 [14]. 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.

978-1-4673-6024-1/13/$31.00 ©2013 IEEE

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 [15]. 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 [16] [17]. 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) [18]. 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.

II. Software and Hardware

The research presented here builds on previous work using ROS (Robot Operating System) for mobile robotics [19] [20] [21] [12]. ROS is a robotics software framework, useful for developing robotics applications. 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 [22] [23].

The Kinect was mounted on the front of a mobile robot, approximately 1 meter above the ground, pointed toward the floor at an angle of approximately 50 degrees from the horizontal. This particular angle and height were chosen based on information about the Kinect's strengths and limitations [23].

III. Classifying Features of the Floor Surface

The Kinect data is transformed into the robot's reference frame, and is used to segment the area in front of the robot according to drivability. The algorithm presented here classifies regions into the following five groups:

Level floor. The robot can drive on this without any problems.

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.

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.

Obstacles. These must be avoided; the robot cannot safely run over them. (The specific thresholds to distinguish obstacles and bumps are dependent on the characteristics of the individual robot. For example, the robot's wheel size directly relates to the size of objects that can be safely run over.) This category also includes dangerous holes in the floor, and the edges of stairs- both going up and going down.

Unknown. If a point cannot be seen by the Kinect, it will be classified as unknown.

This method creates a map of the area in front of the robot, with each (x,y) location having a value to indicate which of the five categories that point falls into.

A. Creating a height map

A two-dimensional array is 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 is then transformed into the robot's reference frame, and the resulting height, z, is stored in the array at the corresponding (x,y) index. It is often the case that multiple points from the point cloud correspond to the same (x,y) entry in the two-dimensional array; in this case, the average height of all such points is stored in that entry of the array. Any cells with no data are filled in with the average height of the surrounding cells, if the surrounding cells contain data.

B. Identifying bumps and obstacles

Bumps and obstacles are characterized by a sudden change in height; they are the easiest features to find and are detected first in this algorithm. ("Bumps" are small enough to be run over, and "obstacles" are too large.) In order to detect them, a second map is created, and populated with values representing the gradient at each point. The gradients are calculated as follows:

$$gradient(i,j) = \sqrt{(d_{x1} + d_{x2})^2 + \left(d_{y1} + d_{y2}\right)^2} \quad (1)$$

$d_{x1}, d_{x2}, d_{y1}, d_{y2}$ 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 as follows:

$$d_{x1} = |height(i,j) - height(i-1,j)| \quad (2)$$

$$d_{x2} = |height(i,j) - height(i+1,j)| \quad (3)$$

$$d_{y1} = |height(i,j) - height(i,j-1)| \quad (4)$$

$$d_{y2} = |height(i,j) - height(i,j+1)| \quad (5)$$

An example of this gradient map is shown in Figure 1b.

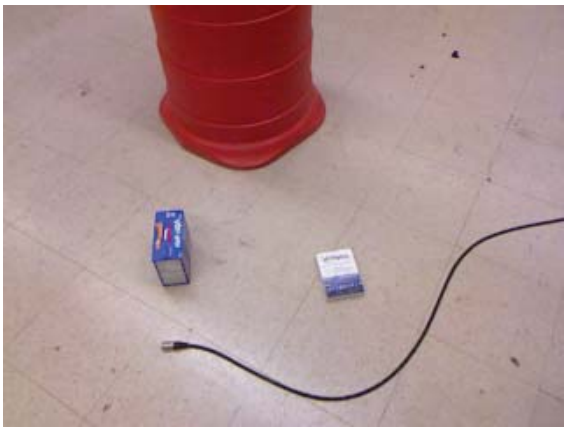
Two threshold values were applied to the gradient data, so that the cells could be divided into 3 categories: floor/ramp, bumps, and obstacles. At this point in the process, all obstacles and bumps have been detected and categorized.

C. Identifying level floors and ramps

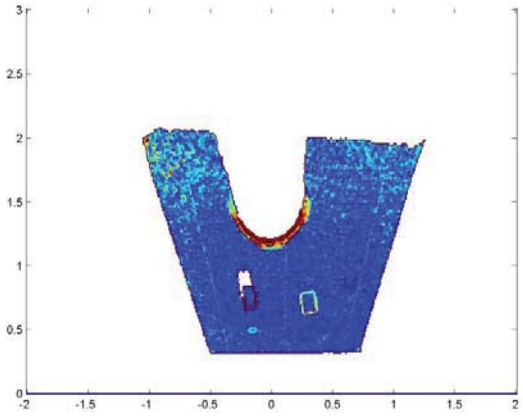
The data is further processed to distinguish the floor from a ramp. Because of the noise, it is only possible to detect a ramp by calculating the slope over a long distance. Mathematically, this can also be accomplished by smoothing the data with a large averaging filter, and then calculating the height differences of adjacent cells. For this application, an averaging filter was used because it could be modified to exclude points which were not part of a level floor or ramp.

The output of this process is a smoothed height map, where each cell contains the average local height. However, only the height data which represents a level floor or ramp is included in the average. Any cells which have already been classified as obstacles, bumps, or unknown are ignored. 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.

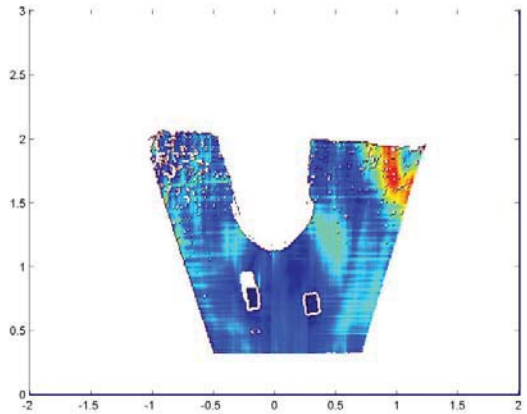
The averaging filter used is a 30x30 square. Initially, every entry in the filter is zero. For each point that needs to be classified as either floor or ramp, the averaging filter is placed over it, so that point is at the approximate center. The corresponding entry in the averaging filter is set to 1. Next, the four cells adjacent to this one are examined. If any of these represents a point containing level floor or



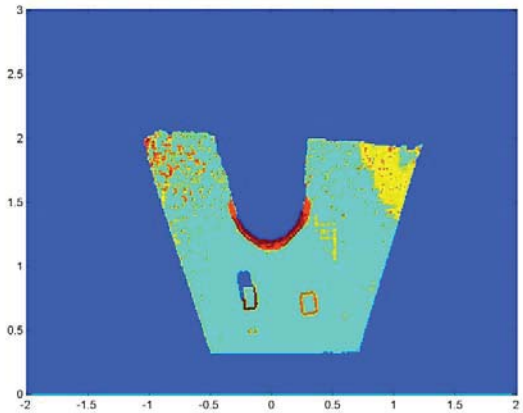
(a)



(b)



(c)



(d)

Figure 1: 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. The robot position is (0,0), the bottom of the image. Units are meters. (a) Image from robot's RGB camera. (b) First gradient map. Detects obstacles and bumps. Red represents high gradients and blue represents low gradients. (c) Second gradient map. Distinguishes ramps from level floor. Red represents high gradients and blue represents low gradients. (d) Final map. Dark blue is unknown space, light blue is level floor, yellow is ramps, orange is bumps, dark red is obstacles.

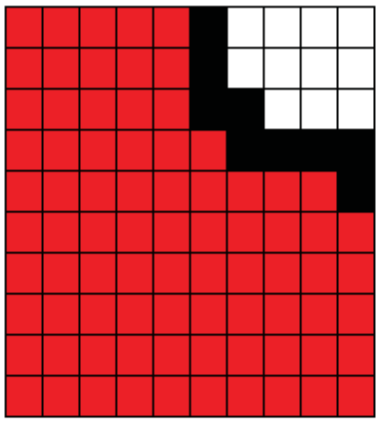


Figure 2: Averaging mask. Only the height data from the red cells is used in the averaging process for the point at the center of the filter. The black cells indicate the edge of an obstacle.

ramp, that entry in the array is set to 1. The process continues, with each iteration setting the entries in the averaging filter to 1 if they are adjacent to a cell already set to 1 and they correspond to a point on the map containing level floor or ramp. In this way, every point within the 30x30 averaging filter is set to 1 if it contains level floor or ramp and it is not separated from the rest of the floor area by the boundary of an obstacle, bump, or unknown area.

In practice, the averaging filter does not need to be regenerated every time. The time required to run the code can be reduced by recognizing that the averaging filter stays largely the same for adjacent points in the map. The values in the filter only need to be shifted, and the newly included points along the edge evaluated. The filter only needs to be completely regenerated when the point at the center crosses over a point marked as a bump, obstacle, or unknown area.

Figure 2 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.

The white cells in Figure 2 are either a level floor or ramp. However, since they are separated from the others by the edge of an obstacle, it cannot be assumed that they are 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.

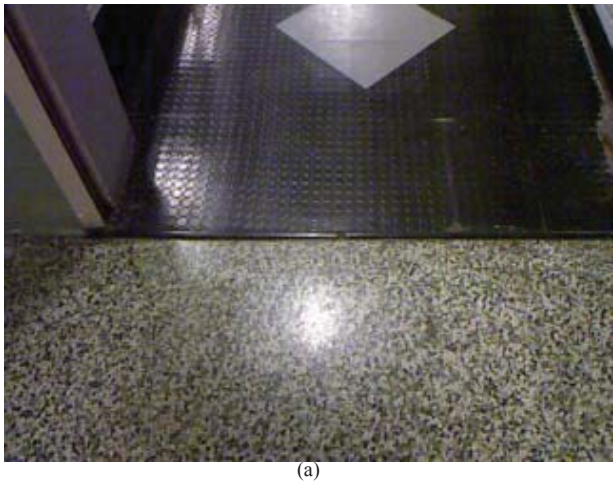
Next, a gradient map was calculated, using an equation similar to equation 1 described above, applied to the smoothed height data. An example of this gradient map is shown in Figure 1c. A threshold was used to separate the floor from ramps. Thus the process is complete and each cell in the map has been classified into one of the five categories.

IV. Results

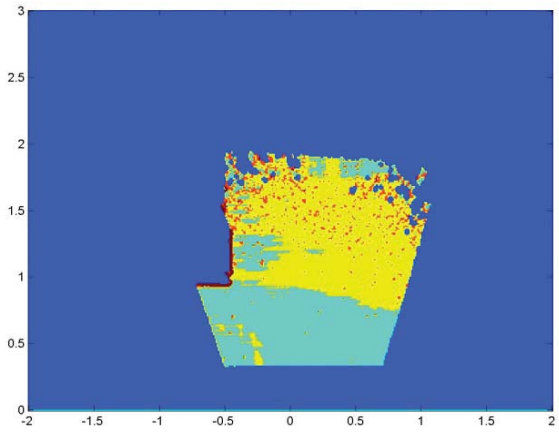
Figure 1d and Figure 3b show the two instances of the results of this algorithm. These images show that the algorithm is able to detect both small and large objects on the floor in front of the robot, as well as distinguishing between a ramp and a level floor.

In Figure 1d, the small box is correctly identified as a traversable object, but the large box is identified as not traversable (based on parameters that define the size of object the robot can safely traverse). The small box has a height of approximately 3 cm; while a wheelchair may be capable of traversing the object, path planning could intervene and use the information to plan a new path which avoids the small box, for a smoother ride. Furthermore, the floor was correctly identified (it was colored light blue).

Figure 3b shows the resulting map when the robot is looking at a ramp sloping upward. The algorithm correctly distinguishes between the level floor (colored light blue)



(a)



(b)

Figure 3: The robot is on a level floor, looking at a ramp that slopes upward. (a) Image from robot's RGB camera. (b) Final map. Dark blue is unknown space, light blue is level floor, yellow is ramps, orange is bumps, dark red is obstacles

	Initialize maps	Transform Kinect data, populate height map	Fill in small gaps in height map	Calculate 1st gradient map, threshold to find bumps/ obstacles	Use averaging mask to smooth height map	Calculate 2nd gradient map, threshold to find level floor/ ramps	Total
	0.01	1.53	0.01	0.00	0.31	0.00	1.86
	0.01	1.61	0.01	0.00	0.29	0.00	1.92
	0.00	1.46	0.01	0.00	0.36	0.01	1.84
	0.00	1.67	0.00	0.01	0.36	0.00	2.04
	0.01	1.78	0.01	0.01	0.35	0.00	2.16
	0.00	1.61	0.01	0.00	0.37	0.00	1.99
	0.01	1.67	0.01	0.00	0.34	0.00	2.03
	0.00	1.58	0.00	0.00	0.37	0.00	1.95
	0.01	1.77	0.00	0.00	0.34	0.00	2.12
	0.01	1.63	0.00	0.00	0.33	0.00	1.97
Mean	0.006	1.63	0.006	0.002	0.34	0.001	1.99
Std dev	0.005	0.09	0.005	0.004	0.02	0.003	0.10

Table 1: Time (in seconds) required to run each of the 6 parts of the algorithm. The time was recorded for 10 runs and the average was 1.99 seconds, with a standard deviation of 0.10 seconds.

and the ramp (colored yellow).

The algorithm also gave accurate results at several other locations, including locations at the top and bottom of a stairwell [23].

A. Speed

The algorithm was run 10 times to test its speed. The results are shown in Table 1.

As Table 1 shows, the average time to run the algorithm was 1.99 seconds. On average, 1.63 seconds is required to transform the data into the correct reference frame, and 0.36 seconds are required for the algorithm itself. The transform is outside the scope of this project; however, in the future the slowness of the transform should be addressed, to improve the performance of the algorithm.

The algorithm produces a map that extends 2 m in front of the robot. This means that theoretically, it could be used on a robot traveling at approximately 1 m/s. In practical sense, however, it would be unsafe to travel at that speed, because the time required to stop the robot's motion must be taken into account. A typical speed for a motorized wheelchair is 0.5 m/s. At this speed, the algorithm presented here is able to identify navigable terrain before the smart wheelchair reaches it.

V. Conclusion

Obstacle detection has always been an important aspect of mobile robotics. In the case of a smart wheelchair robot, it is necessary to identify obstacles and terrain characteristics of the navigation surface itself in order to ensure the safety of the user. It is also essential to identify ramps that the wheelchair will be able to access. Previous systems for obstacle detection often focused on large obstacles in the environment, rather than looking down at 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 navigating a typical indoor environment.

The Kinect is a low-cost sensor that provides reliable three-dimensional depth data of indoor environments, which is useful for robotics. This paper presents an algorithm to use data from 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.

In future work, the algorithm could be integrated with localization and path-planning code. The algorithm would run continuously and provide terrain information which could be used in a world terrain map, which is updated every 2 seconds.

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 the exact location of the robot's wheels when driving over bumps. Path-planning is outside the scope of this paper, but by classifying areas according to drivability, this algorithm outputs the information a path planner needs to calculate a safe route from one point to another.

In summary, the algorithm presented here allows the robot to identify and avoid nontraversable terrain and provide useful information for later use in path planning.

VI. Acknowledgement

We would like to thank Eric Perko, Chad Rockey, Jesse Fish, Toby Waite, Ed Venator, and Bill Kulp for their assistance with the hardware and software for this project. Also, we would like to thank the Joseph P. and Nancy F. Keithley Fellowship Endowment Fund for funding.

VII. References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

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

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

[12] W. Kulp, "Robotic Person-following in Cluttered Environments," *Case Western Reserve University, EECS Dept. Masters Thesis,* 2012.

[13] 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.

[14] 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.

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

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

[17] 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.

[18] 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.

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

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

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

[22] S. Cockrell, "kinect_final," GitHub, 12 Dec 2012. [Online]. Available: https://github.com/scockrell/kinect_final. [Accessed 8 Feb 2013].

[23] S. Cockrell, "Using the Xbox Kinect to Detect Features of the Floor Surface," *Case Western Reserve University, EECS Dept. Masters Thesis,* 2012.