Accurate floor detection and segmentation for indoor navigation using RGB+D and stereo cameras

Muhammad Emaduddin, Khalid Al-Mutib, Mansour AlSulaiman, Hedjar Ramdane
Dept. of Computer Engineering
College of Computer Science and Information
King Saud University, Riyadh, Saudi Arabia
{memaduddin, muteb, msuliman, hedjar}@ksu.edu.sa

Ebrahim Mattar
Electrical & Electronics Engineering Dept.
College of Engineering, University of Bahrain
Kingdom of Bahrain
ebmattar@eng.uob.bh

Abstract

Real world indoor environments are rich in planar surfaces. Floor detection or ground-plane detection is a crucial requirement for a robotic navigation task. Despite frequent successes in this area, problems with detection of navigable floor with multiple planar and non-planar slopes at multiple heights still exist. For robust and safe navigation, such small variations such as floor joins, carpet deformities, raised steps and floor gradients need to be detected and robot path and kinodynamics plan must be adjusted accordingly. The authors suggest a recursive RANSAC segmentation based algorithm that estimates the dominant and sub-dominant plane models for all the navigable planes within a detected floor or a ground plane. The algorithm also divides the input point clouds intelligently into multiple regions of interest for both efficiency and accuracy enhancement. The recursive estimation approach for determining plane parameters helps to detect multiple planes within each region. Among other benefits of this approach, reduction of search space size for the estimation of plane parameters stands out to be the most striking result of this work. This region wise plane estimation approach also helps to reduce the computational load by selectively dropping less significant floor sections from estimation process. The floor estimation technique coupled with sensor response functions for two different point cloud generators further investigates into the robustness of the method when deployed on two distinct sensors i.e. RGB+D sensor and a stereo vision camera. In our experiments we segment navigable floor planes in real-time for a slowly moving sensor. The location and geometrical parameters of the floor planes are updated in a global coordinate systemwhenever a change their location is detected. The planes are associated to a grid map which serves as a path-planning reference to a mobile robot used in our experiments. The results of floor detection and the precision of floor anomaly detection are compared sensor-wise and with the ground truth defined by obstacle heights and configuration.

1. INTRODUCTION

Ground-plane or floor detection and segmentation constitute a fundamental step in any AGV (Automated Guided Vehicle) path planning process. As mobile robot applications are finding their way fast into our household and officespace, reliable and safe navigation is facing more challenges to address in terms of dealing with increasingly complex obstacle space. Planar surfaces have a very wide range of geometrical and non-geometrical properties, for example their orientation, their shape and size, texture and color. All these properties have either a positive or negative impact on the accuracy in 3D information produced bya sensing device. In our experiments we employ two different kinds of 3D sensors based on non-overlapping technologies, namely Microsoft Kinect RGB+D sensor and Point Grey Bumblebee stereo vision camera. This allows us to test our ground-plane detection technique over two 3D datasets having distinct sensor response to objects at the same distance and accuracy thresholds.

From the standpoint of the authors, the estimation of the floor planes needs to be highly accurate in order to enable the indoor AGV to be able to differentiate between floor and obstacles as low as just 1.0 cm. Indoor AGVs can usually navigate through wheelchair accessible surfaces thus throughout our experiments we categorize wheelchair accessible pathways as navigable.

It may be mentioned here that 3D sensors employed in our experiments produce large amount of data and thus in order to keep the floor detection to follow the real time constraints, the proposed method relies upon an object oriented, threaded implementation of pre-optimized segmentation and filtering techniques from open source libraries such as OpenCV, Mobile Robot Planning Toolkit (MRPT) and Point Cloud Library (PCL).

2. RELATED WORK

Although other related methods emphasize on the accuracy of 3D reconstruction [1], ground plane detection using minimal or noisy 3D data [2] and detection of dominant planes in the environment [3] , none of the works suggest a method that detects both dominant and sub-dominant planes along with the height based classification of obstacles. The proposed method not only addresses the navigability problem associated with variations in floor plane but also presents a real-time detection technique that can compromise on accuracy given the available computational resources. The compromise occurs in four respects (i) the amount of 3D data to be processed, (ii) the amount of plane detection

search-space under consideration, (iii) the fineness in the resolution of the floor plane and (iv) the number of possible inclination angles for the planes. This highly customizable technique allows optimized use of precious onboard and remote computational resources. Among other obstacle detection methods some ([4][5]) completely ignore the significance of detecting the ground plane and variations in within it while others [6] use pixel based region segmentation and classification techniques that may or may not be able to classify floor anomalies. In contrast to all discussed approaches, the proposed method has the reliability and accuracy advantage since it processes the dense point clouds in real-time in order to detect floor planes with varying resolution and angle accuracy. The method effectively compresses the dense point clouds into compact *surfels* [7] thus contributing to the domain of compression for navigation data.

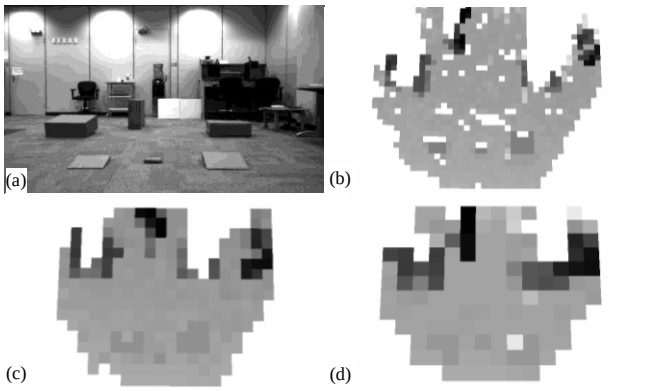

Figure 1 (a) Obstacle test bed (b) corresponding grid-map at 0.1m resolution (b) at 0.2m resolution and (c) at 0.3m resolution.

3. PROBLEM DESCRIPTION

3.1 Grid-map

A certain map location is identified by Cartesian coordinates (in meters) for a particular location within a global map on which AGV path needs to be planned. These Cartesian coordinates are actually the centroid of a square shaped region logically defined on a map stored at the robot memory. A collection of such square shaped adjacent regions constitute the global grid-map for our problem. The length of the side of a squared region is termed as the "map resolution" for our problem. A snapshot of a global grid-map is represented in figure. 1.

We assume that a Grid-map is available for a robot that needs to navigate through an indoor environment. The Grid-map can be converted in shape of an eight connected graph $G = (V, E)$, a set of vertices and edges, which can be later used for implementing path planning algorithms. The unit of space for the grid-map is a vertex as defined in the next section. The dimensions of the grid-map unit exactly represent the physical space. Each vertex can be assigned a value μ between 0 and 1. Here 1 represents the absolute belief that the location is navigable while 0 represents the exact opposite of this belief. Any intermediate value represents the degree of the belief regarding the navigable status of a particular location as per (i).

$$\mu_{x,y} = C \; where$$
$$C = \{c \; \epsilon \; R | \forall c (c > 0) AND (c < 1)\} \quad \ldots(i)$$

The criterion for marking a certain vertex as accessible includes calculating the difference in the height of any two adjacent vertices. If the calculated difference *d* is greater than a pre-defined threshold (*T*) then the AGV travel between two adjacent vertices is restricted. The criterion is illustrated by (ii).

$$if \; \left(difference\left(\mu_{x,y}, neighbor(\mu_{x,y})\right) > T\right)$$
$$\{$$
$$\quad mark\_inaccessible\left(\mu_{x,y}, neighbor(\mu_{x,y})\right)$$
$$\} \quad \ldots(ii)$$

3.2 Population of Grid-Map

We define a vertex within graph G as $V_{x,y} = (plane\_points, Z)$. Here *x* and *y* are the index of the vertex on grid-like eight connected graph. Variable $plane\_points$ is a set of 3D points which belongs to a plane detected at a vertex $v_{x,y}$. Variable Z represents the largest z-coordinate value detected by the proposed method among the set *plane_points*. In other words Z represents the highest point from the ground plane within the set *plane_points*. The ultimate objective of the grid-map is to differentiate between traversable and non-traversable vertices. For this purpose the vertices are assigned values based upon the height and slope of the detected plane. The less the height and the slope, the more likely it is for the vertex to be traversable.

3.3 Sensors and Error Modeling

For the Microsoft Kinect sensor, the following expression from [8] is used to predict the random error in depth data from the sensor.

$$\sigma_{Z_k} = \left(\frac{m}{fb}\right) {Z_k}^2 \sigma_{d'} \quad \ldots(iii)$$

where Z_k denotes the distance (depth) of a point k in the object space, b is the base length and f is the focal length of the infrared camera, m is the parameter of a linear normalization for disparity d, with $\sigma_{d'}$ and σ_{Z_k} respectively are the standard deviation of the measured normalized disparity and the standard deviation of the calculated depth. Expression (iii) in essence denotes that the random error of depth measurement is proportional to the square distance from the sensor to the object. The plane detection parameters were adjusted using the monotonically increasing function (iii) as the distance Z of points from the sensor increases. It must be mentioned here that as the distance of detected planes increase from the sensor it becomes harder for the proposed method to detect small variations due to the increasing standard deviation of the calculated Z_k . Thus the proposed work assigns more confidence level to the variations detected within a range of

3 meters ($\sigma_{Z_k} = 1.4\ cm$) from the sensor as compared to the variations detected at the maximum range of 5m, where the standard deviation σ_{Z_k} itself stands at 4cm.

For Bumblebee XB3 color stereo vision camera, an accuracy function is provided by Point Grey Research Inc. (given in expression (iv), detailed in [9]), the plane detection parameters for our algorithm were adjusted to avoid the noise to be considered as obstacles.

$$\partial_{Z_k} = \left(\frac{-{Z_k}^2}{fb}\right) m' \qquad \ldots(\text{iv})$$

Here ∂_{Z_k} represents the standard deviation of calculated depth error, m' denotes the uncertainty in disparity, f, b and Z_k have the same meaning as for expression (iii). Figure 2 represents the increase in standard deviation in Z_k calculation with respect to increase in object distance from the camera. In the case of Bumblebee camera the standard deviation ∂_{Z_k} is relatively very less for first 5 meters. Obstacles or anomalies with height difference as low 1 cm can be detected easily within first 3 meters as errors hover around ± 0.35 cm.

It may be mentioned here that Kinect sensor and Bumblebee stereo vision camera have very different responses to obstacles at the same distance with the same texture or smoothness properties.

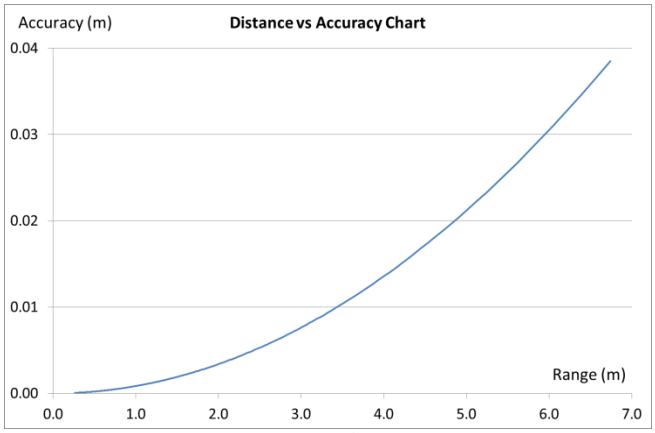

Figure 2 Distance vs. Accuracy chart for Point Grey Bumblebee stereo vision camera for 1280x960 resolution (wide baseline).

3.4 The Problem

The input to proposed method constitutes of a set of 3D points $P = \{p_1, \ldots, p_k\}$. These points are gathered via any one of the two sensors mentioned in detail in the previous section. The objective of the proposed method is to detect the navigable floor planes by fitting all possible planes to the input 3D points consuming minimal time and resources. Furthermore the method should also provide the plane orientation and location information for all the non-traversable planes, which are categorized as obstacles with reference to the presented method.

All those points in the input point cloud are clipped which lie outside the region of interest (ROI). In our experiments, the ROI is a cuboid with the height equivalent to that of the robot (along with sensors) used for navigation (75cm), width equals to 400cm and length equals to 490cm. The location of the sensor with reference to the ROI is shown in figure 5. It is ensured that the sensor location bisects the ROI width. The argument behind choosing an area of roughly 4x5 meters in front of robot is that a mobile robot only needs as much area in front of it to be able to plan its immediate path. A drawback of choosing such an ROI is that a robot has to rely on past data or on some other form of sensors to be aware of terrain in its vicinity that is not part of the ROI. At the same time relatively small size of an ROI allows us to freely apply expensive filtering techniques as well as relay of 3D data over the network for off-board processing.

Figure 3 Sensors mounted on Powerbot for experimentation.

4. THE FLOOR DETECTION AND SEGMENTATION METHOD

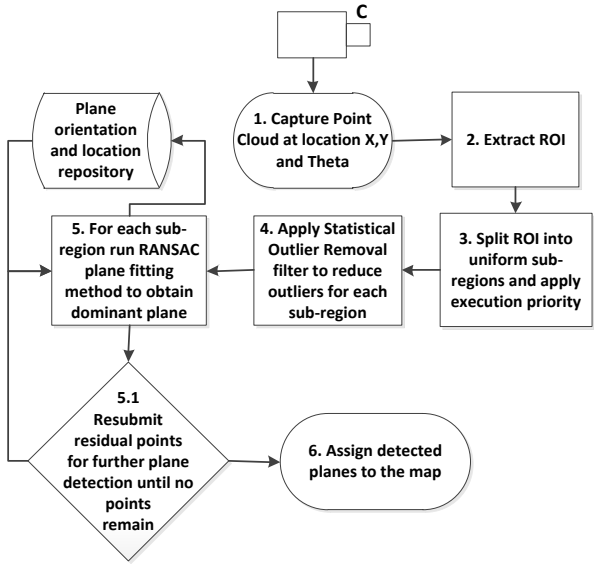

Figure 4 Floor detection and segmentation flowchart

4.1 Capturing Point Cloud at pose *x*, *y* and θ

Although there is no particular innovation involved at this step but it is worth mentioning that sensors involved in the

experiments produce very dense point clouds. The need was felt to down-sample the incoming data but this approach was quickly retreated as any beneficial down-sampling technique proved to be very expensive in terms of execution time. Although dense point clouds are an added burden on computational resources but techniques were developed at later steps in the method to avoid processing unnecessary points. Kinect sensor pose, $Pose_K$ and Bumblebee camera pose, $Pose_{BB}$ are associated with observations from respective sensors. This enables the proposed method to map each point to the vertex $\mu_{x,y}$ of the global grid-map G while the mobile robot mounted with multiple sensors navigates the environment.

4.2 Extract ROI

As discussed in the previous section, all points lying within ROI are extracted and passed on to the sub-sectioning module for further processing. ROI is kept standard for both sensors used in our experiments although sensors with different field of view (FOV) ideally require customized ROI.

4.3 Sub-sectioning ROI and Execution Priority

ROI is split up into a grid of cuboids G'. The x-y-z dimensions of the grid are equal to the dimensions of the ROI as shown in figure. The x resolution (x_{res}) and y resolution (y_{res}) of G' however is a tunable parameter. If the resolution is set too low, the RANSAC based plane fitting algorithm will fail to detect small variations in the floor as the random error in depth data (σ_{Z_k} and ∂_{Z_k}) will surpass floor variations. On the other if the resolution is set too high, the computational burden will render the method non real-time. High resolution nonetheless makes the map nearer to its 3D representation but the foremost priority of the method is to achieve reliable and effective navigation. Thus fast floor detection can be achieved at the expense of lower resolution for G'. As a rule of thumb, the more quicker the rate of growth for functions σ_{Z_k} or ∂_{Z_k}, the more higher the resolution should be for G' as represented by expression (v).

$$O(\partial_{Z_k}) \infty x_{res} * y_{res}$$
$$O(\sigma_{Z_k}) \infty x_{res} * y_{res} \quad \ldots(v)$$

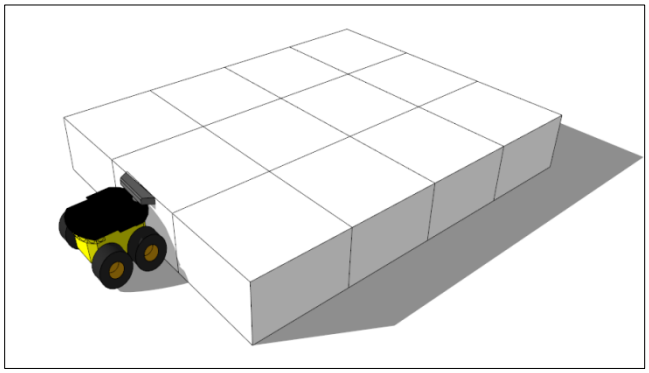

Figure 5 ROI divided into sub-sections. The x-y resolution for dividing ROI is a tunable parameter.

The sub-sectioning approach apart not only serves to detect the floor plane accurately but also helps to reduce the computational load on filtering and segmentations algorithms that can have the complexity up to order $O(n^3)$ where n is the number of 3D points per subsection. Each subsection is assigned an execution priority in order make the proposed method detect the floor planes located closer to the robot much earlier as compared to the floor planes situated away from the robot. The execution priority can be adjusted in a variety of ways in order to facilitate quick path-planning but no experiments were designed to quantify advantages of such an approach. Experiments were conducted however where sub-sections that were the farthest from the robot were skipped from the floor plane detection loop thus saving expensive computational cycles.

4.4 Sparse Outlier removal through Statistical Outlier Removal Filter

Measurement errors from deployed sensors in our experiments produce significant outliers. Such outliers are removed by performing a statistical analysis on the neighborhood of each 3D point. The points which lie outside the noise bounds defined by expression (iii) and (iv) are trimmed from the output of respective sensors. The sparse outlier removal technique used within our method is based on the computation of the distribution of point to neighbors' distances in the input dataset which is generally a standard technique under the given circumstances [10]. A cut-off cardinality threshold is used to determine the n closest neighbors to the 3D point under consideration. Now for each 3D point, mean distance between the point and all its neighbors is computed. The distribution of distances is assumed to be a normal distribution N with a mean μ and a standard deviation σ. Each point whose mean μ is outside an interval defined by the mean of global distances and standard deviation can be considered as outlier and clipped from the output point cloud produced by the sensor. Expression (vi) illustrates the outlier set.

$$out(\varepsilon, K) = \{x: |\mu_x - \mu_{global}| > |\varepsilon\sigma_{global}|\} \quad \ldots(vi)$$

Here, ε is an input parameter which helps to define a quintile of the distribution of distances N(0,1) for each 3D point x. If the value of absolute difference $|\mu_x - \mu_{global}|$ lies outside the $|\varepsilon\sigma_{global}|$ range, point x is considered an outlier. K on the hand is the number of closest neighbors whose distances from point x form the normal distribution N. This step is repeated for all point cloud sub-sections. The filtered point cloud for each sub-section is then processed for plane detection.

4.5 RANSAC based recursive segmentation for dominant plane detection

This step detects multiple planes within a given sub-section of the point cloud by recursively executing RANSAC or "Random Sample Consensus" method for plane fitting. Here the notion of *recursion* means that the same segmentation algorithm is applied to a sub-set of input point cloud repeatedly. This sub-set consists of leftover points (P') from the last iteration which were not assigned to a plane.

Thus these leftover points are fed to the segmentation algorithm again and again unless either no more points can be assigned to a plane or a predefined *recursion iteration limit* is hit as signified by steps 5 and 5.1 in figure 4. The RANSAC based plane fitting algorithm requires three parameters (i) a normal to the plane ($\hat{n}$), (ii) the acceptable error (E) in the Euler angles of the plane whose normal is $\hat{n}$ and (iii) the distance threshold (α) for a 3D point to be part of the plane. The distance threshold is the maximum allowable distance between the point and its projection on the plane for it to be considered as part of the plane. It must be mentioned here that each time a subsection of point cloud $subsection_{xy}(G')$ is submitted to the segmentation algorithm, the algorithm after detecting the dominant floor plane, preserves the normal to the detected plane $\hat{n}_{xy}$. The rest sub-dominant planes within the subsection are stored in a set $\widehat{N}_{xy}$.

4.6 Plane assignment to the grid-map

After the detection of dominant and sub-dominant planes within each $subsection_{xy}(G')$ of the input point cloud from the sensor, the planes need to be assigned to the grid-map G. It is important here to note that a radius is estimated for each detected plane. This radius depends upon the 2D area that the points P' span across. The radius is made a part of vector $\hat{n}_{xy}$. This step effectively converts $\hat{n}_{xy}$ into a *surfel*. The surfels are then evaluated against each vertex of the grid-map G. If a surfel is found to be spanning across certain vertices (or a vertex), each vertex is assigned a value depending upon the height of surfel from the floor. In case the surfel itself represents the floor it is assigned a maximum value of 1.0. For the surfels at maximum height (in experiments this height equals 75cm), the vertex is assigned a minimum value of 0.0. The elevation angle that a surfel makes from the floor further affects he value of each vertex. In experiments each vertex value was passed through a linear function that maps the vertex value to 100 percent if the elevation angle is 0° and to 20 percent if the elevation angle is 60°. It takes many factors to decide the threshold value for navigability of floor, including robot ground clearance, robot Center of Gravity, application and its speed constraints.

Following pseudo code details the recursive floor segmentation algorithm.

4.6.1. Algorithm – Recursive Floor Segmentation

procedure initialize( )

{2.1} **set** all $\hat{n}_{xy}$*=(0,0,1); //a normal facing the direction opposite to the floor*

{2.2} G=initialize_grid_map (0.0, resolution, X*Y);
//all values within grid-map are 0.0 by default .i.e. all nodes are set to be non-traversable. Resolution is set to be at a pre-defined value. X and Y define the total length and width of the grid-map.

{2.3} **set** x_{res} and y_{res};

{2.4} **set** *recursion_iterations;*

{2.5**}** **for** all subsections $G'_{x,y}$

{2.6} **set** error_tolerance ($G'_{x,y}$) $\geq \begin{Bmatrix} \sigma_Z(for\ Kinect) \\ \partial_Z(for\ Bumblebee) \end{Bmatrix}$;

procedure process_pointCloud (*pointCloud P*)

{3.1} P = extract_ROI (P);

{3.2} G'= subsection_pointCloud(P, x_{res}, y_{res});

{3.3} **for** all subsections $G'_{x,y}$

{3.4} {$G'_{x,y}$ = statistical_outlier_removal ($G'_{x,y}$)

{3.5} floor_detection($G'_{x,y}$);}

procedure floor_detection(*subsection graph* G')

{4.1} **for** all subsections $G'_{x,y} \in G'$

{4.2} { **set** *i=0;*

{4.3} $P = \infty$;

{4.4} recursive_plane_segmentation($P \in G'_{x,y}$);

{4.5} update_map(G, $\hat{n}_{xy}$, $\widehat{N}_{xy}$);}

procedure recursive_plane_segmentation (*pointCloud* P')

{5.1} **if** ($P' = \emptyset$) **or** ($i >$ *recursion_iteration*s) where $P' \in G'_{x,y}$

{5.2} { **return** ;}

{5.3} e= error_tolerance ($G'_{x,y}$);

{5.4} **if** (i=0)

{5.5} {($\hat{n}_{xy}, P'$) = RANSAC_planefitting($P', \alpha, e, \hat{n}_{xy}$);}

{5.6} **else**

{5.7} { ($\hat{n}, P'$) = RANSAC_planefitting($P', \alpha, e, \hat{n}$);

{5.8} $\widehat{N}_{xy}$ = ($\widehat{N}_{xy} \cup \hat{n}$);}

{5.9} recursive_plane_fitting(P');

{5.10} $i=i+1$;

procedure main( )

{1.1} initialize( );

{1.2} **forever**

{1.3} process_pointCloud (sensorOutput());

5. EXPERIMENTATION AND PERFORMANCE ANALYSIS

We gathered the point clouds using Kinect sensor with 640x480 resolution@30 FPS and Bumblebee stereo vision camera with two resolutions 1280x960@7.5FPS and 640x480@15FPS (wide baseline). In order to process extra dense clouds from 1280x960 resolution images from Bumblebee camera, the point cloud ROI was serialized and sent over the network to a high-end networked PC for efficient real-time processing. The mobile robotics platform used for our experimentation included PowerBot from Adept Mobilerobots, Intel Dual Core 1.8 GHz Processor onboard the robot and Intel i7 2.20 GHz Processor for off-board processing. In order to deliver real-time performance using the available resources, the search-space for RANSAC plane fitting algorithm is constrained by providing the fitting algorithm with last know floor surface normals $\hat{n}_{xy}$ for each point cloud subsection $G'_{x,y}$. This process is represented by statements {5.4}{5.5} listed in section 4.6.1.

Obstacles of varying heights were placed at varying distance from the sensor as depicted in the figure .6. The ground truth and measured obstacle heights are compared in table. 1. It must be noted that ideal values for x_{res} and y_{res} in the experimental scenario were found to be 4 and 3 for both the Kinect sensor and the Bumblebee camera since the accuracy gain is not feasible enough as compared to the additional computational load if the values are boosted higher than $x_{res} = 4$ and $y_{res} = 3$. The total area that is sub-sectioned in front of the sensor is (6.0x4.2) 25.2 m^2 for Bumblebee stereo camera and (5.5x3.0) 16.5 m^2 for Kinect sensor. Table 1 offers an accuracy comparison of the proposed method on the two sensors used. 20 experiments were conducted with varying obstacle configuration for different (x_{res}, y_{res}) where the proposed method was executed for each sensor separately. The maximum height of each sensor was measured manually and served as ground-truth. The errors between the detected maximum heights of the grid-map vertices and the ground truth were calculated. These average errors are shown in Table 1. It must be mentioned here that as the sub-section resolution increases the added processing to sub-section point cloud and additional function calls starts to affect the execution time of the proposed method.

x_{res}, y_{res}		**Average error in maximum detected height**	
	Obstacle distance from sensor	**< 3 meters**	**>3 meters <5.5meters**
3 , 2	Kinect sensor	±0.51 cm	±5.31 cm
	Bumblebee camera	±0.39 cm	±2.49 cm
4 , 3	Kinect sensor	±0.44 cm	±4.97 cm
	Bumblebee camera	±0.31 cm	±2.21 cm
8 , 6	Kinect sensor	±0.44 cm	±4.88 cm
	Bumblebee camera	±0.32 cm	±2.10 cm
12 , 9	Kinect sensor	±0.40 cm	±4.74 cm
	Bumblebee camera	±0.29 cm	±1.99 cm

Table. 1. Method Accuracy: Sensor wise comparison

6. DISCUSSION

The primary output of the proposed method is a grid-map which can submitted to grid-based path planning method such as A* or D*. Apart from the primary output, the method also returns 3D height maps of the arena. Such maps are relatively much dense as compared to grid-maps and can be compressed and processed for decomposition into Voronoi cells. Each 3D point provided in the input point cloud is associated to a plane (given the *recursion_iteration*s is set to a very large value) by the end of the execution of this method thus the proposed method can be termed as an exhaustive mapping method for a given set of input 3D points. Figure 6 and 7 show the obstacle test bed and the corresponding 3D terrain map generated by the method for $x_{res} = 4$ and $y_{res} = 3$. The x-y resolution of terrain map is 0.2m.

7. CONCLUSION

In this paper, a real-time and accurate floor detection method is proposed that is tested on a mobile robotics platform while using two distinct sensor technologies. The method uses various parameters that can be for scenarios where on-board computational resources are expensive, in order to render real-time floor detection performance. The proposed method sub-sections the floor for overall improved execution time and selective processing of prioritized floor sections. Sensor error models are used to detect floor accurately up to 6 meters away from the sensors and detect obstacles as low as 1cm within the range of 3 meters. The method currently uses surfels to remember floor state between consecutive observations. The surfel parameters are improved as the distance between the detected surfel and sensor decreases. The same concept can be used in future to track the floor surface in noisy environments such as floors crowded with people. Furthermore the presented method also proposes a flexible floor representation (using surfels) whose parameter accuracy can be improved by using multiple sensors simultaneously.

Figure 6 The obstacle test bed consists of flat floor and objects with varying slopes and heights

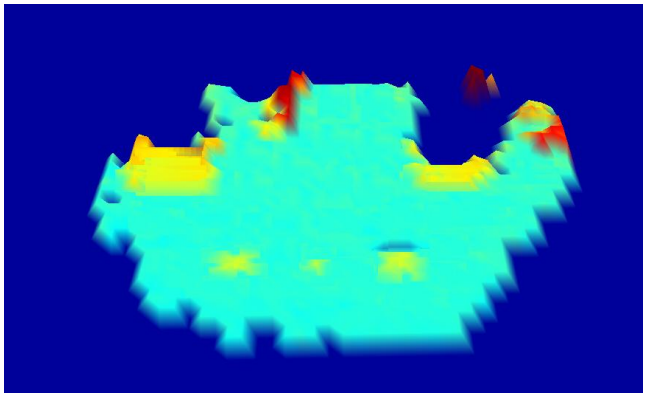

Figure 7 The 3D terrain map representing the detected planes. Color temperature represents height.

8. Acknowledgment:

"This work is supported by National Plan for Science and Technology program at King Saud University, Saudi Arabia Project: Number 08-ELE300

9. REFERENCES

[1] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and others, "KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera," in Proceedings of the 24th annual ACM symposium on User interface software and technology, 2011, pp. 559–568.

[2] M. Heracles, B. Bolder, and C. Goerick, "Fast detection of arbitrary planar surfaces from unreliable 3D data," in Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on, 2009, pp. 5717–5724.

[3] K. Gong and R. Green, "Ground-plane detection using stereo depth values for wheelchair guidance," in Image and Vision Computing New Zealand, 2009. IVCNZ'09. 24th International Conference, 2009, pp. 97–101.

[4] M. Olsson, "Obstacle Detection using Stereo Vision using Unmanned Ground Vehicles," LinkoPing University, Sweden, 2009.

[5] C. D. Pantilie, S. Bota, I. Haller, and S. Nedevschi, "Real-time obstacle detection using dense stereo vision and dense optical flow," in Intelligent Computer Communication and Processing (ICCP), 2010 IEEE International Conference on, 2010, pp. 191–196.

[6] E. Fazl-Ersi and J. Tsotsos, "Region classification for robust floor detection in indoor environments," Image Analysis and Recognition, pp. 717–726, 2009.

[7] P. Henry et al. RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments. In Proc. of the Int. Symposium on Experimental Robotics (ISER), 2010.

[8] K. Khoshelham and S. O. Elberink, "Accuracy and Resolution of Kinect Depth Data for Indoor Mapping Applications," *Sensors*, vol. 12, no. 2, pp. 1437–1454, Feb. 2012.

[9] Point Grey Research Inc, "Stereo Accuracy and Error Modeling," Point Grey Knowledge Base Article. 19-Apr-2004.

[10] I. Ben-Gal, "Outlier Detection," in Data mining : a knowledge discovery approach, New York: Springer, 2005.