3DVFH+: Real-Time Three-Dimensional Obstacle Avoidance Using an Octomap

Simon Vanneste, Ben Bellekens, and Maarten Weyn

CoSys-Lab, Faculty of Applied Engineering
Paardenmarkt 92, B-2000 Antwerpen
{ben.bellekens,maarten.weyn}@uantwerpen.be

Abstract. Recently, researchers have tried to solve the computational intensive three-dimensional obstacle avoidance by creating a 2D map from a 3D map or by creating a 2D map with multiple altitude levels. When a robot can move in a three-dimensional space, these techniques are no longer sufficient. This paper proposes a new algorithm for real-time three-dimensional obstacle avoidance. This algorithm is based on the 2D VFH+ obstacle avoidance algorithm and uses the octomap framework to represent the three-dimensional environment. The algorithm will generate a 2D Polar Histogram from this octomap which will be used to generate a robot motion. The results show that the robot is able to avoid 3D obstacles in real-time. The algorithm is able to calculate a new robot motion with an average time of 300 μs.

Keywords: Three-Dimensional Obstacle Avoidance, Octomap, Navigation and Planning, ROS, Robotics.

1 Introduction

Robotic applications such as airborne or underwater robots need to avoid obstacles in a 3D environment. These robots need to create a 3D map from the environment to locate obstacles. This can, for example, be accomplished by using a 3D laser scan Simultaneous Localization and Mapping (SLAM) algorithm [1] or a RGB-D SLAM algorithm [2,3]. These algorithms will build a 3D map from an unknown environment while at the same time locate the robot within this map. The 3D maps can be represented more efficiently by the octomap framework [4]. In this research we will use this octomap to determine the locations of obstacles so the robot can avoid them.

Researchers [5–7] have tried to solve the 3D obstacle avoidance problem with a 2D robot by creating a 2.5D height map from the original 3D map or by projecting the 3D map on a 2D map. This technique can only be used if the robot can move in 2D. When the robot can move in 3D, researchers [8, 9] solved the 3D obstacle avoidance problem by planning and re-planning a 3D path by using the A* [10] and D* lite [11] algorithm. These techniques have the disadvantage that they are computational expensive. In this paper, we presents the Three

Dimensional Vector Field Histogram (3DVFH+) algorithm. This is an real-time obstacle algorithm, that will generate a robot motion to avoid obstacles in 3D.

The 3DVFH+ algorithm is based on the 2D VFH+ algorithm [12]. It will make a 2D polar histogram from an octomap of the environment. The algorithm consists of five stages to calculate a new 3D robot motion.

This research paper is organised in the following order. To begin with, the related researches will be described in Section 2. Then in Section 3, we will discuss the octomap framework. Section 4 will describe the multiple stages of the 3DVFH+ algorithm. Section 5 follows with an analysis of the results of this research. Next, we will come to a conclusion in Section 6. Finally, Section 7 proposes future improvements to the 3DVFH+ algorithm.

2 Related Work

Ulrich et al. [12] presented the VFH+ algorithm on which the 3DVFH+ algorithm is based. This obstacle avoidance system is able to plan local paths for a robot that can move in 2D. The VFH+ algorithm generates a smooth trajectory in real-time and takes the physical characteristics such as size and turning speed of the robot into account.

Hrabar [13] described a probabilistic roadmap planner. The planner generates a probabilistic roadmap which will be used to plan or re-plan a path with the D* lite algorithm [11]. When a path needed to be re-planned, it took on average 0.15s.

Burgard et al. [9] described a method for an autonomously quad-copter for indoor use. This research will plan a 3D path by using the D* lite algorithm [11] but only consider 2D actions of the robot. For these 2D points the multi-level map will be used to find the surface elevation under the robot to determine the change in altitude cost.

Maier et al. [5] developed a system that allows a Nao humanoid robot [14] to plan its path in a 3D environment and map new obstacles. The path planning algorithm uses a part of the octomap (vertical size of the robot) and project it to a 2D map. Next the path planning algorithm will calculate a path in two dimensions with an A* algorithm [10].

Nieuwenhuisen et al. [8] presents a local multiresolution path planning system. This system will use a multiresolution grid map that will be used for local path planning. This planner will plan a new path to the intermediate goals generated by the global path planner. This path will be planned by using an A* algorithm [10].

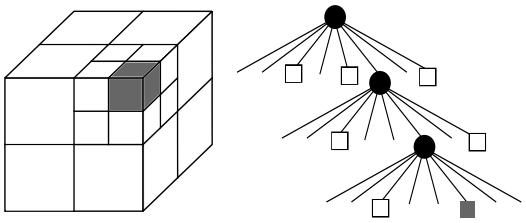

Fig. 1. Octree structure: white octree leaf nodes have status free, gray octree leaf nodes have status occupied

3 Octomap

The octomap [4] data structure is an efficient way to represent a 3D environment. The octomap can, for example, be accomplished by using a 3D laser scan SLAM algorithm [1] or a RGB-D SLAM algorithm [2, 3]. With the octomap the 3DVFH+ algorithm can calculate a robot motion. This section will describe the basic principles of the octomap. The octomap framework is a 3D occupancy grid mapping framework based on the octree structure. The octree data structure is a hierarchical structure containing multiple nodes. These voxels (also called nodes) are cubic volumes in space. Every voxel can contain 8 or 0 child voxels. When the voxel contains 8 occupied child voxels the octomap will reduce these 8 child voxels to one parent voxel that is occupied. If no child voxels are occupied the voxels will be reduced to one parent voxel that is free. All the combinations in between will be stored by using the 8 child voxels. The number of layers in this hierarchical structure (also called the resolution of the octree) determine the precision and size of the octree (see Figure 1). This way the entire environment is represented by voxels.

Robot applications often use a probabilistic representation of the environment because input sensors introduce noise and therefore uncertainty into the system. The octomap leaf voxels contains a probabilistic number. This allows the octomap to have a probabilistic representation of the environment. This likelihood will be used to determine if te voxel is occupied or if voxel is free.

The octomap framework does not directly include information on the voxel location. The location can be calculated when descending the octomap tree. The root voxel has a fixed location and the number of the child voxel that is explored determines the location of the child voxel.

4 3DVFH+

3DVFH+ is an enhanced version of the two-dimensional VFH+ algorithm [12] which is used to work in a 3D environment. The algorithm uses an octomap to

determine where the obstacles are located given the robot pose in a 3D environment. The 3DVFH+ algorithm uses five stages to calculate a new robot motion. These stages are described in the following subsections.

4.1 First Stage: Octomap Exploring

The octomap data structure makes use of the Octree data structure (see Figure 1). When the robot moves around in a large environment it is not possible to explore all the voxels due to computational limits. So the Octomap's exploring stage will only research voxels that lie within a bounding box around the robot. This bounding box has a size of $w_s * w_s * w_s$ (width, height and depth) and contains the Vehicle Center Point (VCP) as centroid. The VCP is the center point of the robot and the whole robot will be represented by this point. When a voxel lies within the bounding box, the voxel is an active cell $C_{i,j,k}$ with i, j, k coordinates within the active region C_a. The exploring stage only needs to find voxels that lie within the boundaries of the bounding box. The algorithm needs to use the location of the voxels to determine which voxels need further exploration or which voxels can be ignored. But the location of the voxels is not implemented in the octomap data structure to reduce memory overhead [4]. These locations will be calculated when exploring the octomap tree. Voxels that are too far from the bounding box will not be explored. This principle also works on high level voxels, which will result in entire branches that can be ignored and improve the exploring speed without losing relevant information. When a voxel is found by the first stage, the second stage will use the information from the located voxel to make a 2D primary polar histogram

4.2 Second Stage: 2D Primary Polar Histogram

The second stage will add the information from the voxel (which has been found by the first stage) into the 2D primary polar histogram. This histogram (as shown in Figure 2) is a polar histogram where the location of an active voxel is determined by two angles. These angles are determined by the position of the voxel and the VCP. A weight that is calculated based on the voxel will be inserted into the 2D primary polar histogram with the two angles as coordinates.

First, the list of active cells will further be reduced by making a bounding sphere within the bounding box. This can be accomplished by calculating the euclidean distance $d_{i,j,k}$ between the VCP and each voxel. When the euclidean distance is larger then the radius of the bounding sphere $d_{i,j,k} > w_s/2$, the active cell will be ignored. A boundary sphere is necessary to create a rotation independent 2D polar histogram.

Next, the algorithm will calculate the two angles to determine the coordinates within the 2D primary polar histogram H^p. These angles are the azimuth angle β_z (x-axis of the 2D primary polar histogram) and the elevation angle β_e (y-axis of the 2D primary polar histogram). These angles are shown in Figure 3

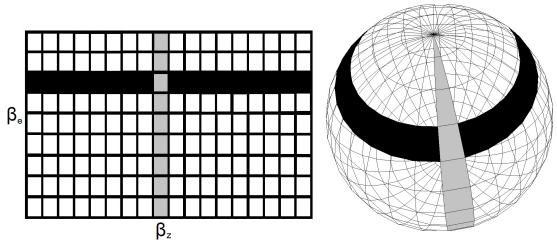

Fig. 2. 2D Polar Histogram

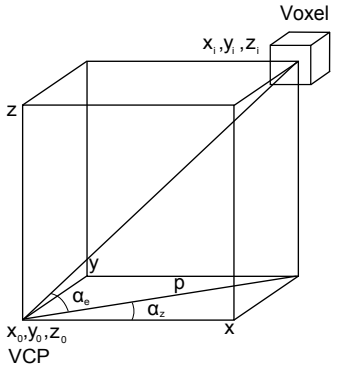

Fig. 3. An image that clarifies the angle calculations

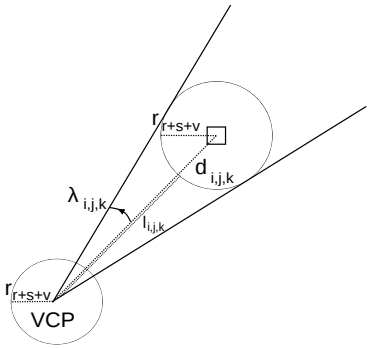

Fig. 4. Enlargement of the voxels

as α_e and α_z.

The azimuth angle β_z is calculated by using (1) which is also used in the VFH+ algorithm [12]. The combination of the floor function and the partition by the resolution of the 2D polar histogram α will create a natural number (requirement for 2D primary polar histogram). The resolution is determined by the difference between the largest angle and the lowest angle that will lead to the same cell within the 2D polar histogram.

$$\beta_z = \text{floor}\left(\frac{1}{\alpha}\arctan\frac{x_i - x_0}{y_i - y_0}\right) \tag{1}$$

The elevation angle needs to be calculated in a different way (see (2) and (3)) because we need the elevation angle independent of the azimuth angle. This can be implemented by calculating p based on the x and y coordinates of the VCP and the voxel, see Figure 3.

$$\beta_e = \text{floor}\left(\frac{1}{\alpha}\arctan\frac{z_i - z_0}{p}\right) \qquad p = \sqrt{(x_i - x_0)^2 + (y_i - y_0)^2} \tag{2}$$

this results in

$$\beta_e = \text{floor}\left(\frac{1}{\alpha}\arctan\frac{z_i - z_0}{\sqrt{(x_i - x_0)^2 + (y_i - y_0)^2}}\right) \tag{3}$$

The previous calculations used the VCP and the center of the voxel to calculate the two angles. The size of the robot is implemented by enlarging all the active voxels in the 2D primary polar histogram with r_{r+s+v} to compensate for the robots size, see Figure 4. This enlargement factor is calculated by adding the radius of the robot r_r, the safety radius r_s and the voxel size r_v. The voxels are enlarged so the algorithm can calculate the maximum and minimum angle in which the voxel lies, see (4).

$$\lambda_{i,j,k} = \text{floor}\left(\frac{1}{\alpha}\arcsin\frac{r_{r+s+v}}{d_{i,j,k}}\right) \tag{4}$$

The minimum distance between the voxel and the VCP can be calculated by subtracting the enlargement from the euclidean distance, see (5).

$$l_{i,j,k} = d_{i,j,k} - r_{r+s+v} \tag{5}$$

Now the algorithm has calculated which cells in the 2D primary polar histogram are influenced by the voxel, the algorithm needs to calculate the weight that the voxel will add to the 2D primary histogram. The weight of a voxel is calculated based on the euclidean distance $l_{i,j,k}$ and their occupancy certainty $o_{i,j,k}$, see (6).

$$H^p_{z,e} = \sum_{i,j,k \in C_a} \begin{cases} (o_{i,j,k})^2(a - bl_{i,j,k}) & \text{if } e \in [\beta_e - \frac{\lambda}{\alpha}, \beta_e + \frac{\lambda}{\alpha}] \\ & \text{and } z \in [\beta_z - \frac{\lambda}{\alpha}, \beta_z + \frac{\lambda}{\alpha}] \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

Two constants a and b are calculated by using (7) to obtain a balanced a and b. The value of these constants is unimportant, only the relative size or fraction is important.

$$a - b\left(\frac{w_s - 1}{2}\right)^2 = 1 \tag{7}$$

The weight of a voxel will be used during stage 4 to obtain the 2D binary polar histogram.

4.3 Third Stage: Physical Characteristics

After the second stage, the third stage will calculate and add new information into the 2D primary polar histogram based on the physical characteristics of the robot and location of the voxel. When the robot has a certain heading θ retrieved by the RGBD-SLAM algorithm and speed, it cannot change direction instantly,

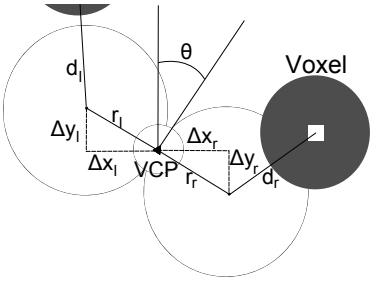

Fig. 5. Turning circles of the physical characteristics stage

so it will need to turn. When the robot is moving slowly, this stage will be ignored because it can change its direction instantly. The robot's turning trajectory depends on the robot's forward velocity, turning speed and the climbing speed. The turning speed and robot velocity are implemented in the VFH+ algorithm [12] so the same method can be used. The climbing acceleration has no influence on this part of the calculation.

First, the two center points of the turning circle need to be calculated (see Figure 5). This can be achieved in the same way as in the original VFH+ algorithm [12] (see (8) and (9)).

$$\Delta x_r = r_r \cdot sin\theta \qquad \Delta y_r = r_r \cdot cos\theta \tag{8}$$

$$\Delta x_l = -r_l \cdot sin\theta \qquad \Delta y_l = -r_l \cdot cos\theta \tag{9}$$

Secondly, the algorithm needs to check every active cell $C_{i,j,k}$ to see if it lies on the turning circle. This will be achieved by calculating the distance $d_{l,r}$ between the centre of the turning circle and the voxel, see (10). To detect if the voxel lies on the turning circle, the algorithm will compare this distance with the safety range r_{r+s+v} and the diameter of the turning circle $r_{l,r}$, see (11). The function $\Delta x(i)$ will calculate the x distance between the voxel and the VCP.

$$\begin{aligned} d_r &= \sqrt{(\Delta x_r - \Delta x(i))^2 + (\Delta y_r - \Delta y(j))^2} \\ d_l &= \sqrt{(\Delta x_l - \Delta x(i))^2 + (\Delta y_l - \Delta y(j))^2} \end{aligned} \tag{10}$$

$$d_r < (r_r + r_{r+s+v}) \qquad d_l < (r_l + r_{r+s+v}) \tag{11}$$

Thirdly, the algorithm also needs to take the climbing motion of the robot into account. The algorithm will calculate which cells within the 2D masked polar histogram are blocked by the voxel. This will be achieved by calculating for all the following α_e angles after the object which α_z angle is blocked by the voxel. First, the climbing motion constant f needs to be calculated. This is achieved by calculating the turning distance t to the obstacle, see (12). Next, the climbing

motion constant is calculated with the altitude difference $z_i - z_0$ and the turning distance, see (13).

$$t = \frac{2.\pi.r(2.\alpha_z)}{360} \tag{12}$$

$$f = \frac{z_i - z_0}{t} \tag{13}$$

Next, a new unreachable altitude $z_{\alpha z}$ needs to be calculated for every α_z angle after voxel. The turning distance $t_{\alpha z}$, see (14) to the next α_z needs to be calculated to determine the unreachable altitude, see (15).

$$t_{\alpha z} = \frac{2.\pi.r(2.\beta_z.\alpha)}{360} \tag{14}$$

$$z_{\alpha z} = f t_{\alpha z} \tag{15}$$

Finally, the unreachable elevation angle β_e will be calculated using the altitude difference and the euclidean distance $l_{\beta z}$, see (16) and by using (17).

$$l_{\alpha z} = \sqrt{r^4 - 2.r^2.\cos(270 - 2.\alpha_z)} \tag{16}$$

$$\beta_e = \frac{1}{\alpha} \arctan(\frac{z_{\alpha z}}{l_{\alpha z}}) \tag{17}$$

These calculations can be extended so that the algorithm will calculate unreachable cells with a given forward speed and maximum climb speed.

4.4 Fourth Stage: 2D Binary Polar Histogram

The previous stages will generate a 2D primary polar histogram based on the voxels of the octomap. The fourth stage will reduce the information further by creating a 2D binary polar histogram based on the 2D primary polar histogram. This will be accomplished by comparing every cell in the 2D primary polar histogram with two thresholds τ_{low} and τ_{high}. When a value is higher than τ_{high} the point will be 1 in the 2D binary polar histogram. When a value is lower than τ_{low} the point will be 0 in the 2D binary polar histogram. If a point lies between the two thresholds the value next to the point will be used in the 2D binary polar histogram. The two thresholds allow the algorithm to distinguish real obstacles and measurement errors.

Because the 3DVFH+ algorithm uses a 2D polar histogram, the thresholds τ_{high} and τ_{low} need to change when using a different elevation angle β_e. The cells of the 2D polar histogram do not have the same size (as shown in Figure 2) so to compensate for this, different thresholds are required for different elevation angles, see (18). The size of these thresholds depends on the robot, the robot's speed, the window size of stage five, the octomap resolution and the bounding sphere size.

$$H^b_{\beta_z,\beta_e} = \begin{cases} 1 & \text{if } H^p_{\beta_z,\beta_e} > \tau_{\beta_e high} \\ 0 & \text{if } H^p_{\beta_z,\beta_e} < \tau_{\beta_e low} \\ H^p_{\beta_{z-1},\beta_p} & \text{otherwise} \end{cases} \tag{18}$$

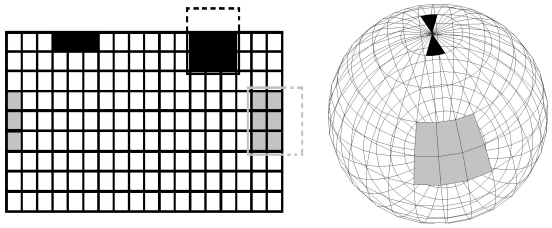

Fig. 6. Moving window of the fifth stage

4.5 Fifth stage: Path detection and selection

The fifth stage searches for available paths in the 2D binary polar histogram and selects the path with the lowest path weight. To determine which paths are available the algorithm will detect openings in the 2D binary polar Histogram by moving a window around the 2D binary polar histogram. This window will mark the path passable, if all the elements in the window are equal to 0. This way only large openings will be threaded as available paths. When implementing this window the algorithms needs to take the polar properties of the histogram into account. When the window crosses the boundaries of the histogram, the window will use elements that are connected by the rules of a 2D polar histogram.

When the window crosses the upper or lower boundary of the histogram (see black markings in Figure 6) the cells that need to be checked are located in the same elevation angle but 180 degrees rotated over the azimuth angle. When the window crosses the left or right border the window will check the cells on the other side of the histogram.

Next, three path weights will be calculated and combined into a single path weight for the candidate direction, see (19). The first path weight will be calculated based on the difference between the target angle k_t and the candidate direction v. The second path weight is the difference between the rotation of the robot θ and the candidate direction v. The last path weight is the difference between the previous selected direction k_{i-1} and the candidate direction v. The variable μ is used to select which of the three path weight has a larger impact on the final path weight. We used for our goal-oriented robot $\mu_1 = 5$, $\mu_2 = 2$ and $\mu_3 = 2$ as proposed by Ulrich et al. [12]. The path weight function can be adapted to enable a different behavior. For example the path weight function can give a preference to a turning motion instead of a climbing motion.

The function $\Delta(v_1, v_2)$ will calculate the difference between the two elevation angles and the two azimuth angles. From these differences the function will generate a weight based on the two calculated angle differences.

$$k_i = \mu_1.\Delta(v, k_t) + \mu_2.\Delta(v, \frac{\theta}{\alpha}) + \mu_3.\Delta(v, k_{i-1}) \tag{19}$$

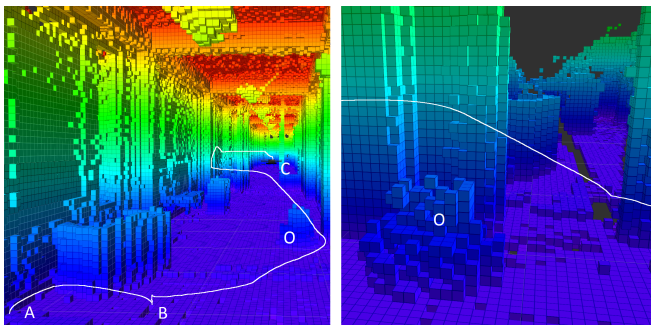

Fig. 7. Left: The full traveled path from point A to C. Right: A part of the traveled path from B to C. The robot avoided an obstacle by flying over the obstacle *O* and by turning.

When the path weight of all the candidate directions is calculated the algorithm can select the direction with the lowest weight. The chosen direction will be converted into a robot motion. The conversion can easily be implemented by making a decision tree that will generate a robot motion based on the coordinates of the calculated direction.

5 Results

In this section we will evaluate the 3DVFH+ algorithm. The requirements for the 3DVFH+ algorithm are that it needs to avoid obstacles in a 3D environment and perform these calculations in real time. These results are generated by using the Robot Operating System (ROS) [15] framework and the simulation software gazebo [16].

5.1 3D Obstacle Avoidance Result

The 3DVFH+ algorithm is a 3D obstacle avoidance algorithm, so to test the algorithm we simulated a quad-copter in gazebo. The quad-copter needed to fly from point A to point B and from point B to point C without bumping into obstacles (as shown in Figure 7). For this simulations, the dataset FR-079 corridor [17] is used. This octomap has a resolution of 0.05 m and is publicly available.

The left image in Figure 7 shows how the robot will go from point A to point B and from point B to point C. The right in Figure 7 image shows how the robot meets an obstacle *O* and will avoid it by a combination of flying over the obstacle and turning to avoid the wall.

5.2 Performance Result

This section will describe the performance of the 3DVFH+ histogram is measured by using the same dataset and the same intermediate points as in the

previous section, see Figure 7. For these experiments we used a standard laptop with an Intel i7-3720QM processor with a NVIDIA GeForce GT 650M. The simulations were executed on a different machines to ensure that the simulation software did not influence the results. When performing the simulation the robot was flying around at a low speed, so the third stage was ignored. The third stage did not impact the performance measurements. The following parameters has the most influences to the performance of the algorithm: the bounding box size, the moving window size, the thresholds of stage four, the robot speed, and the octomap resolution. Due to the increasing complexity and the amount of voxels, the performance will raise or slow down.

We found that the average time that the algorithm needs to calculate a new robot motion is 326 μs. As a result, that the algorithm is capable of calculating on average a new robot motion 3061 times per second. The first three stages mainly determine the speed of the algorithm because these stage speed depends on the number of voxels within the bounding sphere. Based on multiple experiments we found that it takes 300 ns for every voxel to calculate the 2D primary polar histogram. The speed of the fourth and fifth stage are negligible with a speed of 0.2 μs for every robot motion. In practice the algorithm is limited to the other algorithms that the robot uses to calculate its pose and the octomap.

6 Conclusion

The 3DVFH+ algorithm is a real-time three-dimensional obstacle avoidance algorithm that uses an octomap to determine the obstacles locations. The algorithm can determine the location of these obstacles in real-time because the algorithm will only take obstacles into account that are located close to the robot. From the location of the obstacles the algorithm will make a 2D primary polar histogram based on the pose of the robot and location of the obstacles. Next, the algorithm will take the physical capability of the robot into account. In this 2D binary polar histogram the algorithm will find multiple paths, give them a path weight and determine the path with the lowest path weight. This path will be used to calculate a motion for the robot. By using these techniques the algorithm is able to calculate the robot motion real time in 3 dimensions.

7 Future work

Before the 3DVFH+ algorithm can be used, the algorithm needs to be configured based on the robot's size, robot's speed, octomap's resolution and multiple levels of thresholds to generate a 2D binary polar histogram. These parameters are chosen empirically. In future work this process could be automated.

In this system the thresholds that generate a 2D binary polar histogram are static. But when the robot is moving with a different speed, different thresholds could and should be used.

References

1. D. M. Cole and P. M. Newman, "Using laser range data for 3d slam in outdoor environments," in Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on. IEEE, 2006, pp. 1556-1563.
2. F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard,"An evaluation of the rgb-d slam system," in Robotics and Automation (ICRA), 2012 IEEE International Conference on. IEEE, 2012, pp. 1691-1696.
3. P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments," in the 12th International Symposium on Experimental Robotics (ISER), vol. 20, 2010, pp. 22-25.
4. A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, "Octomap: An efficient probabilistic 3d mapping framework based on octrees," Autonomous Robots, vol. 34, no. 3, pp. 189-206, 2013.
5. D. Maier, A. Hornung, and M. Bennewitz, "Real-time navigation in 3d environments based on depth camera data," in Humanoid Robots (Humanoids), 2012 12th IEEE-RAS International Conference on. IEEE, 2012, pp. 692-697.
6. J. Chestnutt, Y. Takaoka, K. Suga, K. Nishiwaki, J. Kuffner, and S. Kagami, "Biped navigation in rough environments using on-board sensing," in Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on. IEEE, 2009, pp. 3543-3548.
7. J.-S. Gutmann, M. Fukuchi, and M. Fujita, "3d perception and environment map generation for humanoid robot navigation," The International Journal of Robotics Research, vol. 27, no. 10, pp. 1117-1134, 2008.
8. M. Nieuwenhuisen and S. Behnke, "Hierarchical planning with 3d local multiresolution obstacle avoidance for micro aerial vehicles," in Proceedings of the Joint Int. Symposium on Robotics (ISR) and the German Conference on Robotics (ROBOTIK), 2014.
9. S. G. G. G. W. Burgard, "A fully autonomous indoor quadrotor."
10. P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," Systems Science and Cybernetics, IEEE Transactions on, vol. 4, no. 2, pp. 100-107, 1968.
11. S. Koenig and M. Likhachev, "Fast replanning for navigation in unknown terrain," Robotics, IEEE Transactions on, vol. 21, no. 3, pp. 354-363, 2005.
12. I. Ulrich and J. Borenstein, "Vfh+: Reliable obstacle avoidance for fast mobile robots," in Robotics and Automation, 1998. Proceedings. 1998 IEEE International Conference on, vol. 2. IEEE, 1998, pp. 1572-1577.
13. S. Hrabar, "3d path planning and stereo-based obstacle avoidance for rotorcraft uavs," in Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on. IEEE, 2008, pp. 807-814. Real-Time Three-Dimensional Obstacle Avoidance Using an Octomap 13
14. D. Gouaillier, V. Hugel, P. Blazevic, C. Kilner, J. Monceaux, P. Lafourcade, B. Marnier, J. Serre, and B. Maisonnier, "Mechatronic design of nao humanoid," in Robotics and Automation, 2009. ICRA'09. IEEE International Conference on. IEEE, 2009, pp. 769-774.
15. "http://www.ros.org/", 2014, Page retrieved on 21/05/2014.
16. "http://gazebosim.org/", 2014, Page retrieved on 21/05/2014.
17. "http://ais.informatik.uni-freiburg.de/projects/datasets/octomap/", 2013, Page retrieved on 18/05/2014.