Hierarchical Ray Tracing For Fast Volumetric Next-Best-View Planning

J. Irving Vasquez-Gomez
Dept. of Computer Sciences
INAOE
Puebla, Mexico
Email: ivasquez@ccc.inaoep.mx

L. Enrique Sucar
Dept. of Computer Sciences
INAOE
Puebla, Mexico
Email: esucar@inaoep.mx

Rafael Murrieta-Cid
Mathematical Computing Group
CIMAT
Guanajuato, Mexico
murrieta@cimat.mx

***Abstract*—A mobile robot must have the ability of building a representation of its environment and the objects in it. To build a three-dimensional (3D) model of a physical object, several scans must be taken at different locations. Selecting each location is the next-best-view problem. Search based methods, where candidate views are generated and evaluated by a utility function, are a solution. However, such methods are slow for high resolution models given that the evaluation requires visibility computation in 3D. We propose a scene representation by octrees with a hierarchical ray tracing that reduces the visibility computation time. Such method performs a coarse ray tracing, except for the interesting volumes where a finer resolution is applied. The method decreases the computation time at least one order of magnitude. Saving time with this method leads to evaluate more constraints and more candidate views in high resolution models.**

I. Introduction

In robotics, a mobile robot must have the ability of building a representation of its environment and the objects on it. Three-dimensional (3D) representations from such objects have several applications, such as object recognition, pose estimation, grasping, etc. The task of building a 3D model from an object is known as 3D object reconstruction [1]. In order to accomplish this task a range sensor is placed at several locations to see the full object surface.

Automated 3D object reconstruction is a cycling process of observing and deciding where to see next. First, a range sensor is placed by the mobile robot at a certain location. After that, a range image (point cloud) is obtained from the sensor. Then, if there are images taken from previous iterations, the new one is registered (aligned in a common reference frame) and merged (a unique point cloud without redundancy is created). A partial model, beyond the range images, is updated with the images. Finally, the next sensing location is planned, based on the current information, or the process is finished if a stop criterion is satisfied.

Planning the next sensing location refers to determining a new configuration which provides as much as possible new information about the object, meanwhile several constraints are satisfied (Section II-C details the constraints). Such problem is called in the literature as next-best-view planning (NBVP) [1]. The planning must be executed in running time given that the object shape is not known in advance.

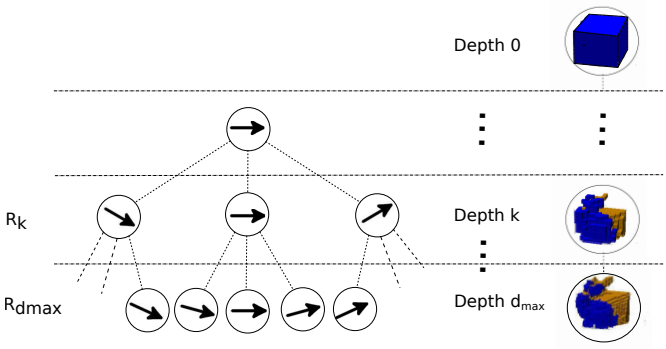


Figure 1. Hierarchical ray tracing. The object is represented with a probability occupancy octree of several resolutions (right). The rays to be traced are organized in a tree structure where each level corresponds to a resolution of the octree (left). Image labels are explained in section V.

Therefore, occlusions generated by the sensor field of view or object auto-occlusions are also unknown. In this paper, our main goal is to propose a method to determine the next-best-view in a short time.

In general there are two types of methods for planning the next-best-view (NBV): surface based and search based. Surface based methods analyze the scanned surfaces to determine the next-best-view, e.g. [2]. Such methods are fast, but sometimes are unable to overcome auto-occlusions or fulfill registration constraints, as the case of computing normals in [3]. On the other hand, search based methods generate a set of candidate views, then their visibility is computed, and they are evaluated by a utility function. For both types of methods, visibility is required, given that is the only way to assure that possible occlusions from the same object, the robot or the environment will not interfere with the target surface.

We propose a NBV algorithm for solving the task of autonomous object reconstruction. The main contribution of the method is the hierarchical ray tracing for fast visibility computation. Hierarchical ray tracing is based on tracing few rays in a rough resolution map, then, when interesting volumes are touch the resolution is increased for observing details. See figure 1. Notice that the obtained

result corresponds to the finest resolution without computing exhaustively rays for all voxels in the finest resolution. Such strategy typically reduces the computational running time needed to evaluate a view in at least one order or magnitude (10 times). This reduction in time allows us to evaluate more constraints and more views than other methods.

II. Background

A. Range Sensor

A sensor configuration x is defined by position and orientation. The device is able to get a 2 1/2 image (z) from the scene, a set of 3D points with respect of the sensor's reference frame. We assume that the range sensor has a perspective geometry, where there is a common origin for all rays that pass trough the image plane. The sensor's field of view (FOV) is determined by a cone or a pyramid depending on the emitted rays. We represent the sensor as follows:

$$S = \{\overrightarrow{o}, \overrightarrow{d}, R\}$$

where $\overrightarrow{o}$ is the origin, $\overrightarrow{d}$ the director ray and R a set of n rays inside the FOV.

$$R = \{\overrightarrow{u_i} | 0 \leq i < n; \overrightarrow{u_i} = [x_i, y_i, z_i]^T\}$$

B. Probabilistic Occupancy Map

An occupancy map performs a division of the 3D space. If the division is uniform, it is called voxel map, e.g. [4]. If a hierarchical division is performed then it is called octree [5]. The occupancy map is used to store information about the reconstruction. Therefore it is used to reason the next-best-view.

Our work is based on a probabilistic octree called *octomap* [6]. An *octomap* is an octree with probabilistic occupancy estimation to deal with imperfect sensor readings. Each voxel has one of three possible labels: i) **occupied**, which represents surface points measured by the range sensor, ii) **free**, which represents free space and iii) **unknown**, whose space has not been seen by the sensor.

An octomap has two parameters, the resolution, size of a voxel, and the tree depth. If the depth is limited then a coarser resolution octree can be obtained. Our work take advantage of this characteristic to trace rays at different depths.

C. Next-best-view Constraints

The set of candidate views used to perform the next-best-view search is denoted by $V = \{v_0, v_1 \dots v_m\}$. Each view is a tuple of the form: $v_i = (q_i, x_i, H_i, u)$ where q_i is the robot configuration, x_i the workspace of the sensor, $x \in \mathbb{R}^3 \times SO(3)$, H is a homogeneous transformation matrix, which is applied to the sensor rays to obtain the FOV of such view and u is the numerical utility of the view.

Our goal is to select one view $v \in V$ with the following characteristics:

1) New information. The NBV must see unknown surfaces in order to completely observe the object.
2) Positioning constraint. There must be a collision free configuration to place the sensor.
3) Sensing constraint. Surfaces to be seen must be in the camera's field of view (FOV) and depth of view (DOV). Also the angle formed between the sensor's orientation and the surface normal must be smaller that a certain angle defined by the vision angle [7].
4) Registration constraint. To assure that the new scan will be merged with the previous ones there must be an overlap between them [8].

Each selected NBV is denoted by p_i. The initial view, arbitrary selected, is specified by p_0. We assume the existence of an object bounding box (OBB) and a robot bounding box (RBB).

III. Related Work

Since the 80's the next-best-view problem has been addressed. For an extended review of classic methods see [9] and [1]. According to [1], our algorithm is volumetric and search based, so we will mainly review similar methods in this section.

The work of Connolly [3] represents the object with an octree and determines the NBV with one of two approaches. The first one determines the NBV as the sum of normals from unknown voxels. The second method, called planetary, determines the NBV by testing views from a set around the object. Recent work have done improvements in object representation and NBV computation.

The representation of the object has been improved by introducing octrees with probabilistic occupancy estimation [6]. Furthermore, hierarchies of octrees have been proposed to decrease the required storage memory [10]. Einhorn *et. al.* proposed the determination of map resolution and spatial subdivision depending on the acquired data in running time [11]. Our work differs from that of Wurm [10], given that they represent each part of the map with disjoint octrees, and we represent the same object with several resolutions.

The NBV computation using volumetric representations is done by defining a search space and then, with a utility function, decide which configuration is selected. Foissotte *et. al.* [12] propose an optimization algorithm to maximize the amount of unknown data in the camera's FOV. However, they do not specify how to solve object auto-occlusions. Given that NBV constraints are many, the optimization problem is commonly reduced to a search over a discrete set of candidate views, where each one is tested to determine their goodness. Vasquez *et. al.* [4] proposed a search over set of views around the object and a utility function which measures surface, quality and distance. Their problem is that a set of pointing views is not always feasible, i. e. the robot can be in collision with an obstacle or the sensor can be occluded by another object, the environment or the same

robot. Therefore, instead of a fixed set of candidate views some works synthesize a reduced set of candidate views with promising goodness. Dornhege and Kleiner proposed the computation of candidates which point to frontier-void volumes [13]. Krainin *et. al.* [14] proposed a method in which the robot grasps the object and moves it inside the camera's FOV, the candidate views are possible object rotations which are evaluated with a trade-off between view goodness and motion cost.

Most of the related work needs to compute visibility from candidate views, otherwise, there is no guarantee that the new location will overcome occlusions from the environment or the robot. In some methods, e. g. [3], [4], [13], ray tracing provides access to the object representation in order to query overlap, quality and occupancy estimation. Our work allows to compute the ray tracing in a short time, it is based on multi-level ray tracing and coherent ray tracing which have been used for rendering 3D scenes [15], [16]. In those techniques, rays with a common feature are grouped into one single beam. Unlike them, we determine when to expand a ray based on the NBV requirements.

IV. Next-Best-View Algorithm

We propose a fast next-best-view algorithm that is able to build the model of an arbitrary object within a maximum size. The algorithm is based on generating candidate views and rank them by a utility function. In this section, the overall framework (utility function and general algorithm) is described. Section V describes the main contribution of this paper, which consist in a method to compute hierarchical ray tracing. This method provides the information required to evaluate the utility function.

A. Utility Function

The utility function ranks the candidate views according to their goodness for the reconstruction process. We propose a utility function as a product of factors. The utility function is defined by equation (1).

$$u(v) = pos(v) \cdot reg(v) \cdot sur(v) \cdot dist(v) \tag{1}$$

where each factor evaluates a constraint, below we detail each constraint.

1) Positioning: $pos(v)$ is 1 whether all voxels inside the robot bounding box (RBB) translated by $v.H$ are **free**, otherwise it is 0.

2) Registration: $reg(v)$ is 1 whether a minimum overlap with previous surfaces exist, otherwise it is 0.

$$reg(v) = \left\{ \begin{array}{ll} 0 & \text{if } oc_o(v) < h_{oc} \\ 1 & \text{if } oc_o(v) \geq h_{oc} \end{array} \right\} \tag{2}$$

where $oc_o(v)$ indicates the amount of occupied voxels that are touched by the sensor (R rotated by $v.H$) and lie inside the object bounding box (OBB), and h_{oc} is a threshold.

3) New Surface: $sur(v)$ evaluates a view depending of how much unknown voxels are seen from v. Unknown voxels are important given that could have occluded surfaces. Such function returns values between 1 and 0. It is 1 whether v sees all the unknown voxels in the OBB, see equation (3).

$$sur(v) = \frac{un_o(v)}{un_o} \tag{3}$$

where $un_o(v)$ returns the amount of unknown voxels touched by the sensor R rotated by $v.H$ that lie inside the OBB, and un_o is the total amount of unknown voxels inside the OBB.

Given that un_o remains constant for all the evaluations of one iteration, equation (3) can be reduced to equation (4):

$$sur(v) = un_o(v) \tag{4}$$

4) Distance: Candidate views are also evaluated according to their distance to the current view. The function is shown in eq. (5):

$$dist(v) = \frac{1}{1 + \rho(v.q, p.q)} \tag{5}$$

where ρ is the euclidean distance in the configuration space.

B. Algorithm

The next-best-view algorithm generates candidate views and ranks them with the utility function. Furthermore, a particular evaluation scheme is applied. The scheme is based on the efficient evaluation proposed by Low and Lastra [17]. Some factors are applied as filters. Therefore, the evaluation is a cascade of filters where the views have to pass through.

Algorithm 1 resumes the process that is described in the following lines. First, a configuration step is performed, here OBB and RBB are defined, all voxels inside the OBB are set as unknown. Octree and rays-tree are configured as is explained in section V. Then, an arbitrary view is given as an initial view. After that, a path is computed with a motion planning technique. The path is executed. Then, from the reached configuration a range image is taken, and it is integrated to the octree. The integration is performed via estimating the occupancy of a leaf node given the readings, as described in [6]. Next, a set of candidate views is computed by uniformly sampling the workspace. Then, for each candidate view, the positioning constraint is checked, if it does not satisfy it, then the view is deleted. For the remaining views, hierarchical ray tracing is performed. It determines which volumes from M are visible from a view v and determines the values for the functions oc_o and un_o. After that, if the registration constraint is satisfied, then the utility is computed. Next, the view with the highest evaluation is selected as the NBV. Finally, if it does nos provide new information, that is, if the amount of unknown voxels visible from it is less than a threshold, then the

process is finished; otherwise the process is continued by moving the robot to the planned configuration.

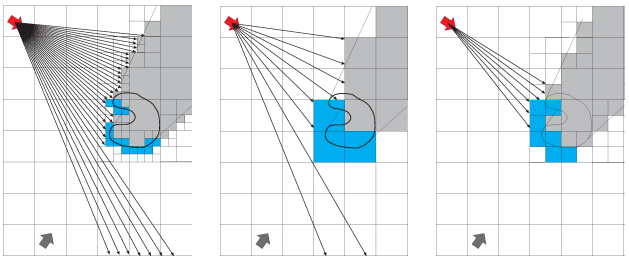
Algorithm 1: NBV for 3D Object Reconstruction

```
Input  : Initial position (p_0)
Output: Object model (M)
Configure octree and rays-tree;
p ← p_0;
while true do
    Plan a path for reaching p;
    Move the robot to p;
    z ← Take range image;
    M ← Update model M using z;
    V ← Generate candidate views;
    foreach v ∈ V do
        if pos(v) then
            HierarchicalRayTracing(M,v);
            if reg(v) then
                v.u ← u(v)
            else
                delete v;
            end
        else
            delete v;
        end
    end
    p ← Best evaluated view from V ;
    if p does not provide information then
        Return M;
        Finish reconstruction;
    end
end
```

V. Hierarchical Ray Tracing

Ray tracing has the purpose of providing visibility to a viewpoint. In computer graphics ray tracing has been widely used to render 3D environments [15]. In NBV planning, ray tracing is a way to predict the sensor readings from a candidate view.

Usually a uniform ray tracing (URT) is performed to compute visibility. Figure 2(a) shows a uniform ray tracing for an octree structure. Each ray from the sensor passes through several voxels until an occupied or unknown voxel is touched. After that, the amounts of hit voxels are returned. Which voxels lies in the way of each ray is computed by Amanatides algorithm [18]. Ray tracing complexity depends on two parameters: voxel resolution and number of rays. Voxel resolution (size of a voxel) affects the complexity given that Amanatides algorithm spans each voxel at finest resolution, so, as the number of voxels increase, more computations are needed. On the other hand, the number of rays depends on the sensor resolution (number of columns times number of rows). Low computation time is obtained with coarse octrees, but accuracy is decreased given that voxel size increases and details are lost.



(a) Uniform ray tracing. All sensor rays are traced in the octree.

(b) Rays traced in a coarse octree.

(c) Ray tracing in a finer octree only for touched occupied voxels.

Figure 2. Examples of uniform ray tracing and hierarchical ray tracing.

The proposed method starts tracing a low density set of rays for a coarse octree (Fig. 2(b)). Then, if a occupied voxel is hit, a higher density set of rays is traced into a finer octree. The higher density set tries to cover only the volume from the occupied voxel that was hit. If an unknown voxel is hit then there is no subdivision to a finer octree. Notice that avoiding subdivision when a unknown voxel is hit allows us to save time. Furthermore, the information about unknown voxels at a rough resolution is enough to know that the sensor must move to explore the associate space. In contrast, occupied voxels are always expanded to the finest resolution. There are two reasons to proceed in this way: 1) A finer resolution might contain unknown voxels. 2) The information of occupied voxels is needed up to the finest resolution, since it is used to decide whether or not there is enough overlapping to register the scan.

In order to trace rays for a given octree resolution, we have computed a tree of rays where each level of the tree correspond to a determined octree resolution. Sections V-A and V-B detail the octree and rays-tree structures, respectively. Section V-C formalizes the method.

A. Octree

We represent the object and the environment with an octomap M of depth d_{max}. Such map can be reduced in resolution by pruning leafs until a desired depth. Therefore, a particular instance of M is m_i where $0 < i < d_{max}$ indicates the depth of the octree.

1) Updating the octree: For each sensor measurement, the octree is updated with the sensor readings according to the occupancy grid mapping model of an octomap [6]. The inner occupancy is updated with the mean occupancy or the maximum occupancy [6]. Maximum occupancy provides a more conservative strategy than mean occupancy.

B. Ray-tree

1) Rays for Resolution: For a given octree depth we have computed a set of rays. Such set has the same sensor aperture, but the rays are separated according to the voxel size of the target depth.

Let's suppose an octree node m_i with resolution l_i, a range sensor with horizontal aperture of a_h and vertical aperture of a_v, and a maximum depth of view of d. Then the reduced set of rays R_i is computed by equation (6).

$$R_i = \{\overrightarrow{u}_{(j,k)} | \overrightarrow{u}_{(j,k)} = [x_{(j,k)}, y_{(j,k)}, z_{(j,k)}]^T\} \quad (6)$$

where the number of image rows is limited by $0 \leq j < \frac{a_v}{\arcsin \frac{l}{d}}$ and the number of the image columns is limited by $0 \leq k < \frac{a_h}{\arcsin \frac{l}{d}}$. The components of each ray are computed as follows (We assume a right-handed coordinate system and a sensor which points to positive z axis):

$$\begin{aligned} x_{(j,k)} &= \cos\beta_j \sin\alpha_k \\ y_{(j,k)} &= \sin\beta_j \\ z_{(j,k)} &= \cos\beta_j \cos\alpha_k \\ \beta_j &= -\frac{a_v}{2} + j * \arcsin\frac{l}{d} \\ \alpha_k &= -\frac{a_h}{2} + k * \arcsin\frac{l}{d} \end{aligned}$$

2) Initial Resolution: Above we have specified the set of rays for a given octree depth. However, it is possible to select an intermediate starting depth k, where $0 < k \leq d_{max}$. When $k = d_{max}$ the hierarchical ray tracing is a uniform ray tracing in the maximum resolution, as k decreases the amount of initial traced rays also decreases.

3) Tree: In order to replace a ray by a higher density set of rays a tree structure is proposed. The union of all rays, $\bigcup_{i=k}^{d_{max}} R_i$, is organized in a tree $T = \{r_0, r_1 ... r_n\}$, each node is defined by:

$$r_j = (\overrightarrow{u_j}, p_j, C_j, lf_j)$$

where $\overrightarrow{u_j}$ is a ray, p_j is a reference to its parent, C_j is a set of references to its children, and lf_j the number of leaves that depends on it.

The tree is organized as follows:

- The root node is the sensor's director ray $\overrightarrow{u_d}$.
- Each level is composed by the set of rays R_i that corresponds to the octree depth i.
- For each ray node r_j, except the root, its parent p_j is the ray node from the superior level with the shortest distance. The distance is computed with the angle formed between rays. If there are equal distances, the parent is selected arbitrary. To the selected parent a reference to r_j is added to its children list C.

C. Hierarchical Ray Tracing Algorithm

Hierarchical ray tracing (HRT) is done recursively. See algorithm 2. The input data are the candidate view, the octree, the ray-tree and the initial depth for ray tracing. The output is a structure with the amounts of each voxel type. First, each ray child of the ray root is traced in the octree. Function `CastRayIn()` throws a ray at the specified depth and returns the label of the hit voxel. Next, if it is occupied then HRT is performed recursively for the children of the rays and the map, until the leaves are reached. But, if the hit voxel is unknown, then, there is no need to perform a finer ray tracing, so the number of leafs from the current rays is added (line 11). Notice that adding the number of leaves sums up to the total number of rays.

Algorithm 2: Hierarchical Ray Tracing (HRT)

```
   Data: v, M, k, rayNode = root
   Result: VoxelAmouts
1  foreach r ∈ rayNode.C do
2      switch CastRayIn(r.u, M, k) do
3          case OCCUPIED
4              if k == d_max ∨ isLeaf(rayNode) then
5                  VoxelAmounts.occupied ++;
6              else
7                  Add HRT( v, M, k + 1, r) to
                   VoxelAmounts;
8          case UNKNOWN
9              VoxelAmounts.unmark += r.lf;
10 Return VoxelAmounts;
```

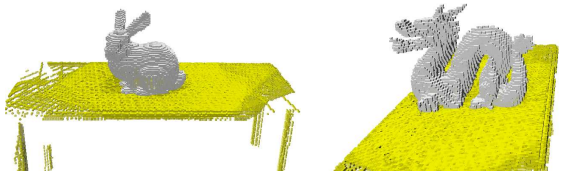
VI. Experiments

A. Simulation

This experiment simulates the reconstruction of several objects. Also, a comparison in terms of efficiency between hierarchical ray tracing (HRT) and uniform ray tracing (URT) is performed. The scene consists of an object over a table inside an empty room. See figure 3. The sensor is a range camera and the positioning system is a point robot able to move free (freeflyer) in the 3 D environment. Thus, our device can be positioned in any collision free point in R^3 and it can be rotated to any direction. 320 candidate views were placed around the object pointing to the object center (views were generated with icosahedron tessellation). A resolution of 0.01 m was established for the octree.

Performance of URT and HRT has been compared in the reconstruction of two complex objects. Fig. 4 shows the output models obtained with the proposed planner. The results are shown in table I. In this table one can see that the required time per iteration is reduced more than 90% when HRT is applied. Furthermore, the main advantage of the method is that it can discard fast views that are occluded, e.g. views below the table are evaluated in 0.05 s compared with 5.6 s obtained with the URT. The number of views is increased given that there is an over estimation of unknown

Figure 3. Simulated reconstruction scene. The objects are placed in the center of the table. Notice that the table occludes the object from several view points.



(a) Bunny. (b) Dragon.

Figure 4. Output octrees from reconstructed objects.

voxels when rays are no further expanded; however this is more than compensated by the important reduction in the time required to analyze the views.

Object	Ray Tracing	Views	NBV time
Bunny	Uniform	10	1796 s
Bunny	HRT	13	97 s
Dragon	Uniform	10	2075 s
Dragon	HRT	15	88 s

Table I
SIMULATION RESULTS.

B. Real Case Reconstruction

This experiment performs the reconstruction of an office fan with a Microsoft Kinect sensor mounted on a mobile robot. See figure 5(a). The objective is to show experimentally that the method can deal with real data in acceptable computation times.

For each iteration, 1500 candidate views were generated randomly (sampling the robot's configuration space and projecting the samples to the workspace). Fifty unknown voxels was established as stop criterion threshold. Initial depth was established to 14 of 16. Voxel resolution was 2 cm.

The method required seven scans to reach the stop criterion. Figure 5 shows some snapshots of the reconstruction. The fan was seen from different locations around it. However, some unknown voxels were not seen given that they are not reachable for the robot, e.g. above the fan. The time required per iteration is shown in table II. The values are average time per iteration. It is worth to say that evaluating 1500 views only takes 10.3 s, it is a short time considering that all reconstruction constraints are checked. Previous results, shown in [4], perform the evaluation of 80 views in 6.2 s. It is worth to say that the speed up in time is greater than in the simulation experiment, given that there are many views that do not see the object, so they are discarded fast.

In the case of the experiment with the real robot, we present preliminary results, in which collision free robot's paths are computed with a Biased Rapidly Exploring Random Tree [19].

Modeling	Scanning	0.1 s
	Octree update	6.3 s
NBV Planning	Views Evaluation	10.3 s
	Path Planning	0.3 s
Total time		17.0 s

Table II
AVERAGE NBV COMPUTATION TIME BY ITERATION.

VII. CONCLUSIONS AND FUTURE WORK

We have presented a next-best-view algorithm for 3D object reconstruction. The main contributions are i) a way to perform hierarchical ray tracing, which can efficiently discard occluded views, ii) a utility function for next-best-view planning and iii) an integrated system for object reconstruction. Using this new hierarchical ray tracing, the evaluation of hundreds of candidate views can be achieved in seconds. Therefore, more restrictions can be evaluated and a better view can be selected. In this work, we have considered a point robot able to move free (freeflyer) in the collision free space of a 3 D environment. As a future work we want to consider a robot with no trivial geometry, in particular we want to address the problem of computing the next best view for object reconstruction with a mobile manipulator robot, equipped with an eye-in-hand-sensor. For this problem is essential to include a motion planning technique in order to find collision free paths for the robot to reach a next best view configuration.

ACKNOWLEDGMENT

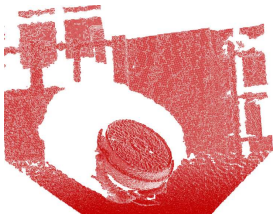
This work was supported by Conacyt and the National Institute of Astrophysics, Optics and Electronics (INAOE)

REFERENCES

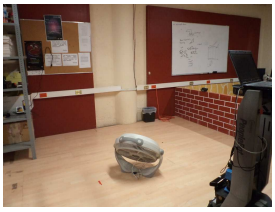
[1] W. R. Scott, G. Roth, and J.-F. Rivest, "View planning for automated three-dimensional object reconstruction and inspection," *ACM Comput. Surv.*, vol. 35, pp. 64–96, March 2003.

[2] J. Maver and R. Bajcsy, "Occlusions as a guide for planning the next view," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 15, pp. 417–433, 1993.

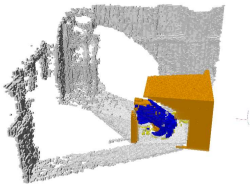
(a) Initial configuration.



(b) Point cloud acquired from initial configuration.

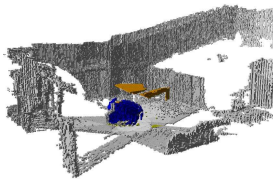


(c) First planned configuration for the robot.



(d) Octree representation after two scans. Object is painted in blue. Unknown voxels of the object bounding box are painted in yellow.

(e) Last planned configuration.



(f) Octree after seven scans.

Figure 5. Office fan reconstruction with a mobile robot and a Kinect sensor.

[3] C. Connolly, "The determination of next best views," in *IEEE International Conference on Robotics and Automation 1985, ICRA85*, vol. 2, 1985, pp. 432–435.

[4] J. I. Vasquez, E. Lopez-Damian, and L. E. Sucar, "View Planning for 3D Object Reconstruction," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS09)*, 2009, pp. 4015–4020.

[5] D. Meagher, "Geometric modeling using octree encoding," *Computer Graphics and Image Processing*, vol. 19, no. 2, pp. 129–147, 1982.

[6] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard, "OctoMap: A probabilistic, flexible, and compact 3D map representation for robotic systems," in *Proc. of the ICRA 2010 Workshop on Best Practice in 3D Perception and Modeling for Mobile Manipulation*, Anchorage, AK, USA, May 2010.

[7] S. Chen, Y. Li, J. Zhang, and W. Wang, *Active Sensor Planning for Multiview Vision Tasks*. Springer-Verlag, 2008.

[8] P. Besl and N. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, pp. 239–256, 1992.

[9] K. Tarabanis, P. Allen, and R. Tsai, "A survey of sensor planning in computer vision," *IEEE Transactions on Robotics and Automation*, vol. 11, pp. 86–104, 1995.

[10] K. M. Wurm, D. Hennes, D. Holz, R. B. Rusu, C. Stachniss, K. Konolige, and W. Burgard, "Hierarchies of octrees for efficient 3d mapping," in *IEEE/RSJ International Conference on Intelligent Robots and Systems 2011*, 2011, pp. 4249–4255.

[11] E. Einhorn, C. Schroter, and H.-M. Gross, "Finding the adequate resolution for grid mapping - cell sizes locally adapting on-the-fly," in *IEEE International Conference on Robotics and Automation (ICRA), 2011*, may 2011, pp. 1843 –1848.

[12] T. Foissotte, O. Stasse, A. Escande, P.-B. Wieber, and A. Kheddar, "A two-steps next-best-view algorithm for autonomous 3d object modeling by a humanoid robot," in *Proceedings of the 2009 IEEE international conference on Robotics and Automation*, ser. ICRA'09. Piscataway, NJ, USA: IEEE Press, 2009, pp. 1078–1083.

[13] C. Dornhege and A. Kleiner, "A frontier-void-based approach for autonomous exploration in 3d," in *IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*, 2011.

[14] M. Krainin, B. Curless, and D. Fox, "Autonomous generation of complete 3d object models using next best view manipulation planning," in *ICRA*, 2011, pp. 5031–5037.

[15] A. S. Glassner, *An introduction to ray tracing*. London, UK, UK: Academic Press Ltd., 1989.

[16] I. Wald, W. R. Mark, J. Günther, S. Boulos, T. Ize, W. Hunt, S. G. Parker, and P. Shirley, "State of the art in ray tracing animated scenes," in *STAR Proceedings of Eurographics 2007*, D. Schmalstieg and J. Bittner, Eds. The Eurographics Association, Sep. 2007, pp. 89–116.

[17] K.-L. Low and A. Lastra, "Efficient constraint evaluation algorithms for hierarchical next-best-view planning," *3D Data Processing Visualization and Transmission, International Symposium on*, vol. 0, pp. 830–837, 2006.

[18] J. Amanatides and A. Woo, "A fast voxel traversal algorithm for ray tracing," in *In Eurographics '87*, 1987, pp. 3–10.

[19] S. M. LaValle and J. J. Kuffner, "Randomized kinodynamic planning," *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, 2001.