VPL: A View Planning Library for Automated 3D Reconstruction

J. Irving Vasquez-Gomez
Consejo Nacional de Ciencia y Tecnología (CONACYT)
Instituto Politécnico Nacional (IPN),
Centro de Innovación y Desarrollo Tecnológico en Cómputo.
Ciudad de México, México
jivasquezg@conacyt.mx

Abstract

During the last decade, automated three-dimensional (3D) object reconstruction and inspection have become widely used in the same way that 3D sensors and positioning systems have become affordable. For example, nowadays drones or underwater vehicles are capable of building 3D models of complex structures. To achieve an autonomous reconstruction it is essential to determine each pose of the sensor, from where an observation is made. In the literature, this task is called view planning. To develop a custom application or propose a new view planning algorithm requires several software tools that are usually reimplemented each time; this causes an effort duplication and is one of the reasons why view planning research papers usually do not present comparisons. We present VPL, a software framework to develop view planning algorithms and compare their performance versus other approaches. The library is written in C++ and it is released under open source BSD license.

1 Introduction

The objective of a 3D reconstruction is to build a computational representation of the shape of a physical entity; in some cases, the entity is an object or it could be an entire scene, where the concept of 3D mapping is best suited. During the last decade, automated 3D reconstruction has become a core task in many applications, and its success has been leveraged by the price decrement of 3D sensors and positioning systems. Some successful examples are the modeling of historic buildings [16], the reconstruction of home furniture by mobile robots [22] [11] or the exploration of underwater structures [23]. Furthermore, with the spreading of the micro unmanned aerial vehicles (UAV), best known as drones, a wide set of related tasks is emerging; for example, building facade reconstruction [7], object reconstruction and inspection [1] or inspection of the electricity transmission equipment [15].

To achieve an autonomous reconstruction or inspection it is essential to determine each pose of the sensor, from where an observation is made. In the literature, this task is called view planning (VP) [18]. Many contributions have been made towards addressing the problem of VP; accordingly to [5] more than two thousand papers have been published until 2010 in active vision (the generalization of the VP problem) for object reconstruction, inspection and other applications. Additional information can be found in the following surveys: [5, 12, 18, 19]. However, due to the wide spectrum of applications, until now, there is no method that can deal with all particular problems. When a new application rises up, the researchers or developers have to deal with the particular constraints of the problem and their hardware (sensors and positioning systems). So, there are several setbacks that need be addressed in addition to the research: i) implementation of the spatial representation, ii) implementation of the visibility algorithms, iii) incorporation of external algorithms such as motion planning or range sensor simulation and iv) implementation of current approaches in order to compare the performance.

Yet, to our knowledge, no software for development and testing of view planning algorithms is available to the general public. The source code may not be available, the implementations are ad-hoc, hardware dependent or require advance knowledge of the particular method. As consequence, various research groups had to develop their own software, leading to effort duplication and lack of a comparison benchmark.

For the previous reasons, we are presenting VPL (The acronym of view planning library) which provides a platform to develop view planning algorithms and perform comparisons quickly. VPL is written in C++ and it is based on a set of widely used libraries: Boost, Octomap and MRPT. VPL provides the data structures to represent the space, provides visibility algorithms, implements sev-

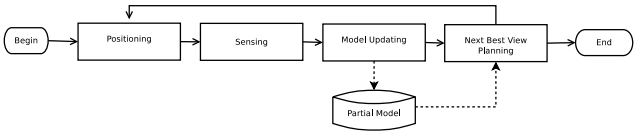


Figure 1. Flow diagram of the automated 3D object reconstruction. The steps are repeated until a stop criterion is satisfied.

eral view planning algorithms reported in the literature and provides flexibility to link with range sensor simulators or motion planning algorithms. VPL has been designed for the problem of object reconstruction, however, it could be applied to 3D scene mapping or object inspection. VPL is available for download at `github.com/irvingvasquez/vpl` under open source BSD license.

1.1 View Planning Problem

The view planning problem (VPP) refers to the computation of a set of views, sensor positions and orientations, so that the gathered observations at those views, images or point clouds, is ideally enough to reconstruct an object or inspect it. The VPP is an NP-Hard problem [18]; the search space is infinite and traditional optimization algorithms tend to fall into local minima [8].

When the object shape is known *a priori*, the problem is defined as inspection instead reconstruction, and it is related to the "art gallery problem" [17]. Ideally, to inspect an object, the set of views is planned in one iteration, before the sensor is moved [3]. However, in the real cases, the gathered information is not sufficient due to positioning errors, or it could be corrupted by noise, increasing the need for a runtime re-planning and a probabilistic representation of the information.

In the reconstruction problem, the object is not known in advance. So the reconstruction is addressed iteratively, in each iteration the steps of positioning, sensing, model updating and planning are executed. Fig. 1 shows the flow diagram of the reconstruction of an unknown object. In the planning step the next sensor view is computed. Besides that computing the optimal view is intractable, the literature has adopted the term of *next best view* planning to the problem of finding a feasible and reasonably good sensor pose that increases the reconstructed surface [6].

2 The View Planning Library

VPL provides tools to develop an automated reconstruction/inspection implementation without dependence on the sensor or positioning system. In VPL, the implementation of a view planning problem is made by integrating

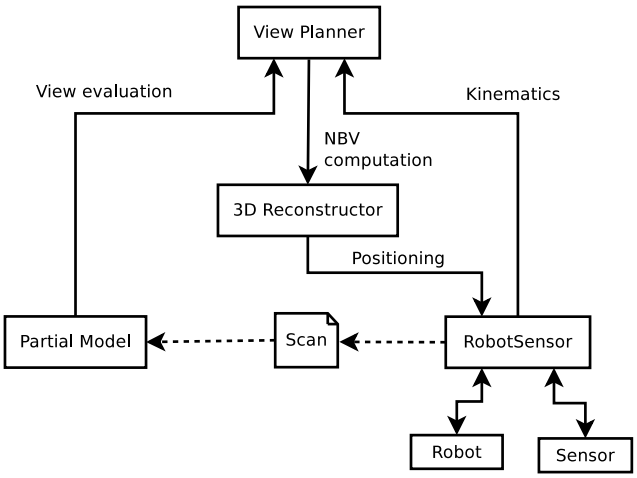


Figure 2. Building diagram of the implementation of an automated 3D reconstruction problem. The arrows show the interaction between components.

four main components: partial model, view planner, robot-sensor and a 3D reconstructor. Fig. 2 shows the relation between them. In this section we describe each component. The core component, view planner, is explained in detail in section 3.

2.1 The sensor

In VPL, the sensor is defined by a "director ray" and a set of "rays". The director ray is a unit vector that describes the direction where the sensor is pointing. The rays are unit vectors that pass trough the image plane.

Let us assume that the sensor has a perspective geometry. Therefore, the sensor, R, is a set of rays that pass though a common origin, see equation (4).

$$R = \{r_i | 0 \leq i < w \times ; r_i = [x_i, y_i, z_i]^T\} \tag{1}$$

where x_i,y_i y z_i are the components of a unit vector.

2.2 View Definition

A view v describes a sensor pose, $v = (x, y, z, \psi, \theta, \phi)$, namely the position of the sensor and orientation of the sensor's reference frame S. Given that, in many cases the view is associated with a robot state, VPL implements a view as a tuple of three elements:

$$V = (v, x, H(x)) \tag{2}$$

where, v is the pose and belongs the to view space, $\mathcal{V}$, which represents all possible combinations of the sensor position

and orientations, $\mathcal{V} \subseteq \mathbb{R}^3 \times SO(3)$; x is a robot state and belongs to robot's state space, $\mathcal{X}$ [14]; and H is a homogeneous transformation matrix that represent the sensor pose instantaneously at state x; so that,

$$^{o}r = H(x)^{s}r \tag{3}$$

where ^{s}r is a ray in the sensors reference frame and ^{o}r is a ray in the global reference frame. H is included explicitly because it reduces the need of computing it for each ray in the sensor.

2.3 Partial Model

The partial model stores the information about the workspace, W, where the entity to be reconstructed or inspected lies. Several partial models have been proposed such as voxel maps [21], octrees [6], point clouds [20], or probabilistic octrees [11] [22]. VPL uses a probabilistic octree however it is not restricted to it.

VPL makes use of the Octomap [10] library to represent W. Octomap implements a probabilistic occupancy estimation based on octrees. This representation has the advantage of low spatial complexity and provides a probabilistic approach to model the uncertainty of the sensor readings. In this representation each voxel has associated a probability of being occupied. In addition, we divide the probability in classes, this allow us to implement a wider set of algorithms. Depending on the probability of been occupied, we classify each voxel with one of three possible classes: 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. Each class has a defined probability interval that can be adjusted accordingly to the needs. The default values are: $[0.45, 0.55]$ for the unknown voxels, less than 0.45 for the free voxels and larger than 0.55 for the occupied voxels.

In addition, the partial model provides functions to get a wider variety of voxel definitions that have been reported in the literature. The additional voxel definitions are: visible unknown, which is an unknown voxel that is adjacent to at least one free voxel, and frontier unknown is a visible unknown voxel that is also adjacent to an occupied voxel. Two voxels are adjacent if they share a face. See Fig. 4. Furthermore, VPL also provides the normals of occupied and visible unknown voxels.

2.3.1 Visibility

One of the functions that is repeatedly used in view planning is the visibility calculation; it computes the surfaces that are visible from a sensor view point. In VPL, it is defined as collecting the voxels that are inside the sensor's frustum, $\mathcal{F}(v)$. VPL implements the visibility calculation with ray tracing.

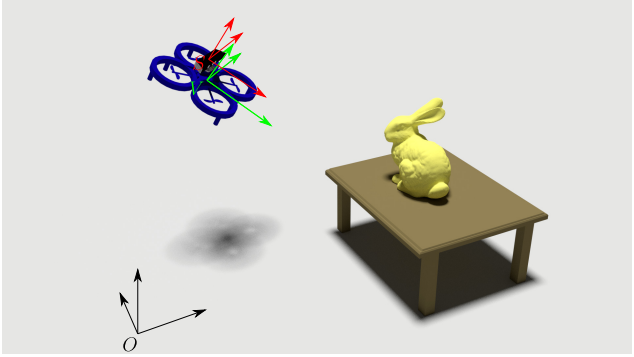


Figure 3. Example of reference frames. Inertial frame: O, body or vehicle frame: V, sensor Frame S.

Let us assume that the sensor has a perspective geometry. Therefore, the sensor, R, is a set of rays that pass though a common origin, see equation (4).

$$R = \{r_i | 0 \leq i < n; r_i = [x_i, y_i, z_i]^T\} \tag{4}$$

where x_i,y_i y z_i are the components of a unit vector.

Then, during the visibility calculation, for each r in the sensor R a ray is traced accordingly to the Amanatides algorithm [2]. In addition, depth of view constraints are also verified.

2.4 The Robot

The robot class encapsulates the communication with the physical or simulated robots. This class standardizes all the robots by providing a unique interface to the planner and reconstructor.

2.4.1 FreeFlyer Robot

A freeflyer robot can be positioned at any configuration. It is represented by the reference frame V. See Fig. 3. In this reference frame we assume that x axis is in the front, y is to the left and z is described upwards.

2.5 The Sensor and the Robot

The moving and sensing capabilities are represented into the Robot and Sensor classes. Such classes have been designed to be replaced by the user on dependency of the user hardware. The RobotSensor class unifies the sensor and the robot into a single object, this allow us to deals with the transformation of the point cloud to global reference frame given the robot configuration.

2.5.1 Simulation

To simulate a reconstruction or inspection it is necessary at least to provide range sensing simulation (robot motion could be bypassed by a direct positioning of the sensor). VPL provides a simplistic range simulator based on ray tracing. The precision of the simulator depends on the octree resolution. This simulator could be replaced by other alternatives such as Glidar [24] or Blensor [9], which can add higher precision or more realistic features as noise. Alternatively, if a more realistic simulation is needed, for example robot motion with dynamics or collision, VPL robot classes can easily be replaced with nodes that communicate with Gazebo or V-Rep simulators though TCP or ROS.

2.6 View Planner

The view planner implements the algorithms to compute the next best view or the set of inspection views. It makes queries to the partial model, in order to evaluate possible views, and also makes queries to the RobotSensor Interface about the kinematics model. Section 3 describes the algorithms that are currently implemented in VPL.

2.7 3D Reconstructor

The 3D reconstructor coordinates the modules and follows the iterative process drawn in Fig. 1. VPL provides a naive iterative reconstructor that implements a stop criterion based on the new information that provides the calculated NBV [22]. More intelligent coordinators could be proposed and added, making the reconstruction robust to unthoughtful situations.

3 NBV Planning Algorithms

VPL implements several NBV planning methods. We hope that this amount will be increased soon with collaboration of the research community. Most of the NBV methods, reported in the literature, lie into a two steps methodology: synthesis and evaluation. In the former step, the partial model is analyzed in order to generate some candidate views; when a single view is generated, then it is taken as the NBV. In the latter step, the generated views are evaluated with a given metric.

3.1 Synthesis Methods

Synthesis methods compute the NBV or a set of promising views by geometrical analysis of the reconstructed object. Next, we will review some of the implemented synthesis methods.

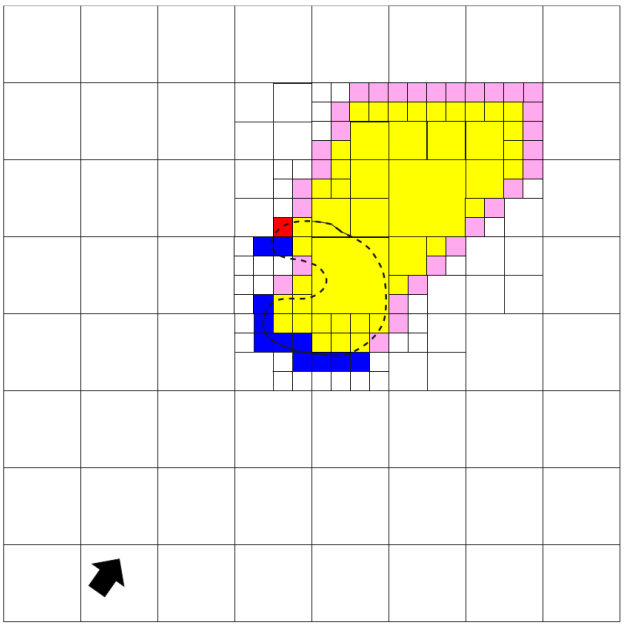


Figure 4. Voxels labels after a scan. Occupied voxels are painted in blue, unknown voxels are painted in yellow, visible unknown voxels are painted in pink and frontier unknown voxels are painted in red.

3.1.1 View Sphere

A naive method that provides a set of points equidistant from the center of the object (See Fig. 6(d)). The view sphere is a wide used method for generating a set of candidate views.

3.1.2 Random Sampling

It provides views by sampling the view space under a uniform distribution.

3.1.3 Frontier normals

The normals of the frontier unknown voxels are used as axis to generate candidate views. Torabi et al. [20] proposed a similar method based on generating patches near to the point cloud boundary.

3.2 View Evaluation Metrics

In this section we describe some of the implemented evaluation metrics. Such metrics determine the goodness of a view depending on their own criteria.

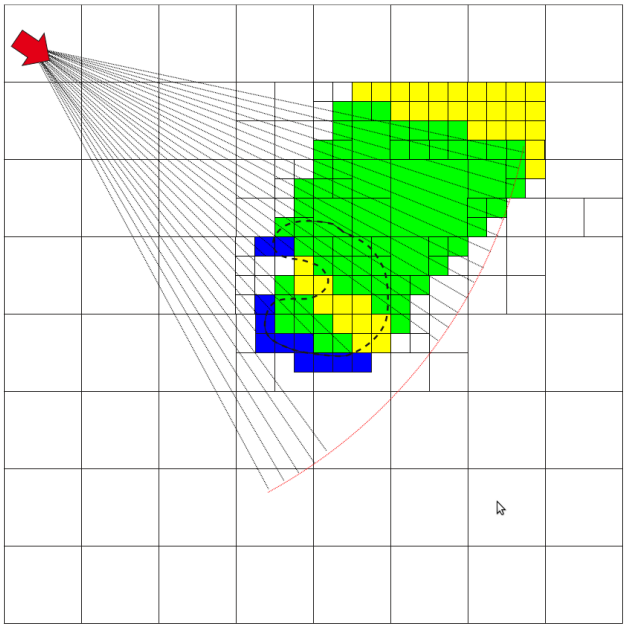


Figure 5. Ray tracing for computing the Frustum Information Gain (FIG). The figure shows in green the voxels that contribute to the frustum information gain.

3.2.1 Voxel Amount trade-off

This approach evaluates a view by comparing the amount of gathered voxels against a desired trade-off of voxels [21]. The idea of the trade-off is to improve the chances of a good registration.

3.2.2 Frustum Information Gain

The frustum information gain (FIG) is a variant of the information gain metric used by [13] and [11]. This approach is based on the asseveration that the information contained in each voxel i can be measured by its entropy [13], see equation (5).

$$I(i) = \sum_{i \in f(x)} -p(i)\ln(p(i)) - (1-p(i))\ln(1-p(i)) \quad (5)$$

where $p(i)$ is the occupancy probability for the voxel i.

The entropy of a voxel, equation (5), is higher when the probability of been occupied is close to 0.5, and it is lower when the probability tends to 0 or 1. Namely, the voxel has more information when it is unknown. Therefore, the frustum information gain sums all the information that is contained in the set of unknown voxels that lie inside the sensor frustum, see equation (6).

$$I_f(x) = \sum_{\forall i | i \in f(x) \wedge i \in M_u} I(i) \quad (6)$$

where $f(x)$ is the set of voxels that are touched by the ray tracing,

$$f(x) = \bigcup_{\forall r \in R_x} M_r \quad (7)$$

, and M_u is the set of unknown voxels. Fig. 5 shows an example of the voxels that contribute to FIG. FIG integrates the entropy for each voxel inside the sensors frustum but avoids multiple integration caused by ray tracing.

3.2.3 Scan Information Gain

The scan information gain (SIG), proposed in [13], calculates the information gain for all voxels touched by the rays of the sensor, see equation (8). Unlike the FIG, in this method each ray contributes to the computation even if the voxels are touched more than one time.

$$I_r(x) = \sum_{\forall r_j \in R_x} \sum_{\forall i \in M_{r_j}} I(i) \quad (8)$$

3.2.4 Visible Unknown Voxels

This approach counts the visible unknown voxels (VUV). See equation (9). The visible unknown voxels are the unknown voxels that are first touched by the ray tracing. In [21] the same type of voxels are called occplane (the contraction of occlusion plane) and they are defined as unknown voxels that are adjacent to a free voxel.

$$I_u(x) = \sum_{i \in f(x)} I_o(i) \quad (9)$$

so that

$$I_o(i) = \begin{cases} 1 & i \text{ is a visible unknown voxel} \\ 0 & i \text{ otherwise} \end{cases} \quad (10)$$

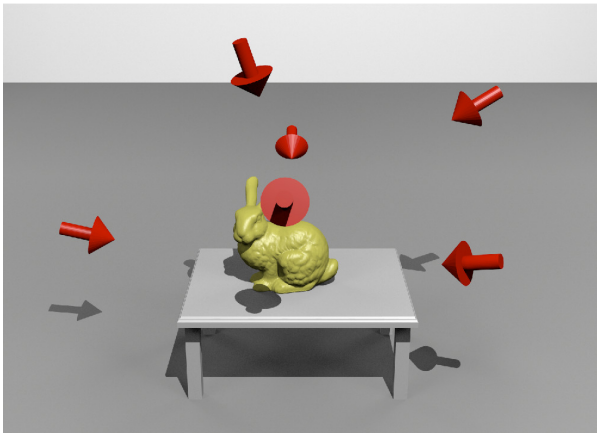
3.2.5 Rear Side Voxel

Rear side voxel metric (RSV) counts the number of voxels behind of a seen surface expecting that such voxels will be part of the object. See equation (11). This metric was proposed in [11], and it shown a surface coverage advantage with respect of other metrics. See [11] for more details.

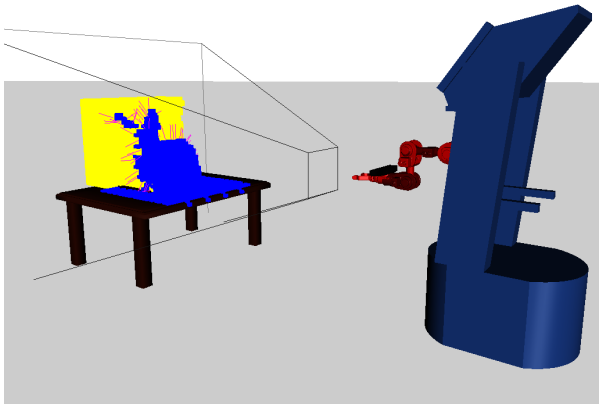
$$I_s(x) = \sum_{\forall r_j \in R_x} \sum_{\forall i \in M_{r_j}} I_b(i) \quad (11)$$

where

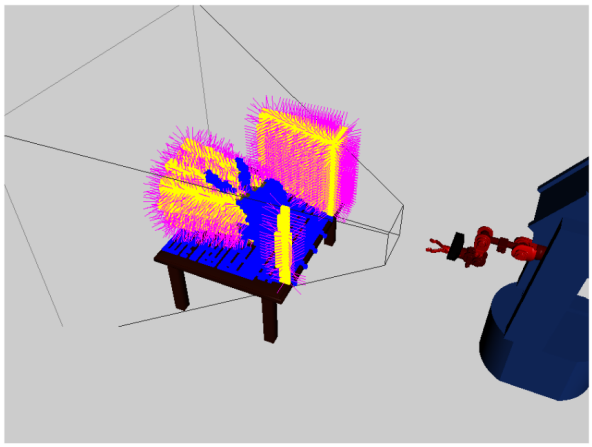
$$I_b(i) = \begin{cases} 1 & i \text{ is a rear side voxel} \\ 0 & i \text{ otherwise} \end{cases} \quad (12)$$



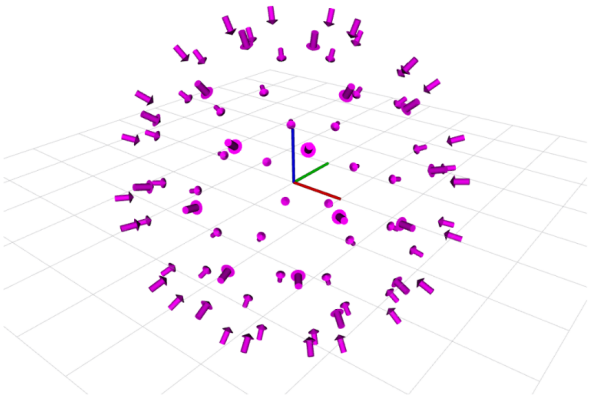
(a) Set of vews that observe an object.



(b) Robot view/state.



(c) Normals of the visible unknown voxels.



(d) View sphere renderization with ROS visualizer (Rviz).

Figure 6. Applications examples of the view planning library. The library is able to handle with the sensor poses (a) or view/sates (b). The library provides several view planning algorithms, for example, surface normal based(c) or view sphere synthesis (d).

4 Conclusions and Future Work

VPL has been tested on several Ubuntu versions and with several hardware configurations. Fig. 6 illustrates some uses of VPL. The library does not require a GPU but a better CPU is desirable in order to make computations faster. The software requires some external libraries that are open source and are easy to find and download. We will do our best, to maintain the library up to date with new NBV algorithms and updates of its dependencies. A ROS compatible version is being prepared and it will be released soon. It is our hope that our library will be useful to others and we invite to researchers and developers to contribute with their own implementations.

5 Acknowledgments

During the development of the library, J. Irving Vasquez-Gomez was supported by a CONACYT PhD. scholarship and by CONACYT Catedras 1507 project.

References

[1] Kostas Alexis, Christos Papachristos, Roland Siegwart, and Anthony Tzes. Uniform coverage structural inspection path–planning for micro aerial vehicles. In *Intelligent Control (ISIC), 2015 IEEE International Symposium on*, pages 59–64. IEEE, 2015.

[2] John Amanatides and Andrew Woo. A fast voxel traversal algorithm for ray tracing. In *In Eurographics '87*, pages 3–10, 1987.

[3] Andreas Bircher, Mina Kamel, Kostas Alexis, Michael Burri, Philipp Oettershagen, Sammy Omari, Thomas Mantel, and Roland Siegwart. Three-dimensional coverage path planning via viewpoint resampling and tour optimization for aerial robots. *Autonomous Robots*, 40(6):1059–1078, 2016.

[4] José-Luis Blanco et al. Mobile robot programming toolkit (mrpt), 2011.

[5] Shengyong Chen, Youfu Li, and Ngai Ming Kwok. Active vision in robotic systems: A survey of recent developments. *The International Journal of Robotics Research*, 30(11):1343–1377, 2011.

[6] C.I. Connolly. The determination of next best views. In *Proc. IEEE Int. Conf. on Robotics and Automation*, volume 2, pages 432–435, St. Louis, MO, USA, 1985.

[7] Shreyansh Daftry, Christof Hoppe, and Horst Bischof. Building with drones: Accurate 3d facade reconstruction using mavs. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3487–3494. IEEE, 2015.

[8] T. Foissotte, O. Stasse, A. Escande, and A. Kheddar. A next-best-view algorithm for autonomous 3D object modeling by a humanoid robot. In *Proc. of International Conference on Humanoid Robots, 2008.*, pages 333–338, 2008.

[9] Michael Gschwandtner, Roland Kwitt, Andreas Uhl, and Wolfgang Pree. Blensor: blender sensor simulation toolbox. In *International Symposium on Visual Computing*, pages 199–208. Springer, 2011.

[10] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard. Octomap: an efficient probabilistic 3d mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, 2013.

[11] Stefan Isler, Reza Sabzevari, Jeffrey Delmerico, and Davide Scaramuzza. An information gain formulation for active volumetric 3d reconstruction. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3477–3484. IEEE, 2016.

[12] M Karaszewski, M Adamczyk, and R Sitnik. Assessment of next-best-view algorithms performance with various 3d scanners and manipulator. *ISPRS Journal of Photogrammetry and Remote Sensing*, 119:320–333, 2016.

[13] S. Kriegel, C. Rink, T. Bodenmuller, A. Narr, M. Suppa, and G. Hirzinger. Next-best-scan planning for autonomous 3d modeling. In *Proc. IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, pages 2850–2856, Vilamoura, Algarve, Portugal, 2012.

[14] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.

[15] Chang-an Liu, Rui-fang Dong, and Hua Wu. Flying robot based viewpoint selection for the electricity transmission equipment inspection. *Mathematical Problems in Engineering*, 2014, 2014.

[16] Maurice Murphy, Eugene McGovern, and Sara Pavia. Historic building information modelling–adding intelligence to laser and image based surveys of european classical architecture. *ISPRS journal of photogrammetry and remote sensing*, 76:89–102, 2013.

[17] Joseph O'Rourke. *Art Gallery Theorems and Algorithms*. Oxford University Press, 1987.

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

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

[20] L. Torabi and K. Gupta. An autonomous six-dof eye-in-hand system for in situ 3d object modeling. *The International Journal of Robotics Research*, 31(1):82–100, 2012.

[21] J. I. Vasquez, E. Lopez-Damian, and L. E. Sucar. View Planning for 3D Object Reconstruction. In *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4015–4020, St. Louis, MO, USA, 2009.

[22] J Irving Vasquez-Gomez, L Enrique Sucar, and Rafael Murrieta-Cid. View/state planning for three-dimensional object reconstruction under uncertainty. *Autonomous Robots*, pages 1–21, 2015.

[23] Eduard Vidal, Juan David Hernández, Klemen Istenič, and Marc Carreras. Online view planning for inspecting unexplored underwater structures. *IEEE Robotics and Automation Letters*, 2(3):1436–1443, 2017.

[24] John O Woods and John A Christian. Glidar: An opengl-based, real-time, and open source 3d sensor simulator for testing computer vision algorithms. *Journal of Imaging*, 2(1):5, 2016.