Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Institute for Control Engineering of Machine Tools and Manufacturing Units

University of Stuttgart
Seidenstraße 36
D–70174 Stuttgart

Diplomarbeit Nr. 3742

Design and implementation of Next-Best-View algorithms for automatic robotic-based (dis)assembly tasks

Viktor Zielke

Course of Study:	Informatik
Examiner:	Prof. Dr. rer. nat. Marc Toussaint
Supervisor:	M.Eng. Christian Friedrich
Commenced:	February 17, 2016
Completed:	August 18, 2016
CR-Classification:	I.2.9, I.2.10

Abstract

Robots tasked with the autonomous interaction of objects, such as assembly and disassembly tasks, in a dynamic environment require the ability to explore their environment and detect objects for interactions. State-of-the-art methods exist which can handle these tasks separately. This work describes a method for combining both tasks and therefor reduce the amount of costly operations like motion and sensing. A next-best-view system is developed which incrementally builds a map of the environment and enables the selection of view poses for an eye-in-hand robot system. The system and the performance of the selected view poses is evaluated on a robotic system. The evaluations showed that the method selected view poses which explored the environment and detected objects.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

In order to manipulate objects in the environment, a robot requires knowledge of the environment and the object. Planning motions to interact with an object in the environment have to be safe, i.e. avoid any collisions of the robot with the environment, to avoid damaging the robot or the environment. As robots are employed in dynamic environments, where the location of objects can change or they are exposed to new environments, an automatic exploration of the environment is required. As the environment in which a robot acts can change, algorithms used for the exploration need to be able to explore with a minimal amount of a priori knowledge. Robots tasked with the assembly or disassembly of objects in the environment require an object recognition system to correctly classify objects which need to be assembled or disassembled in the environment. Modern object classification algorithms still suffer from misclassification of objects, either due to inaccurate sensor measurements or the limitation of a single view of the object. As an object can be occluded during the single view or the view does not provide a sufficient amount of features to classify an object unambiguously, multiple views of an object are required [DCB04]. State-of-the-art methods exist which can explore the environment or plan views for an object recognition task. As measuring the environment and motions of the robot are considered expensive operations, this work formulates an approach which combines both tasks into one view. With this, views of the environment and the object are provided and the overall amount of views and motions required to accomplish both tasks is reduced. The problem of finding a view pose which provides information for a task is known as the Next-Best-View (NBV) problem and was introduced by Connolly [Con85]. In this work, the exploration and object recognition tasks are formulated as NBV problems and solved with a local optimization algorithm. An environment map is build from sensor measurements and a mapping framework. The environment is used for the evaluation of view poses for solving the NBV problem. The approach is evaluated with an eye-in-hand robot system. The evaluations show that the combination of both tasks leads to view poses of the robot system which can explore the environment and provide relevant image data for the object recognition task.

Outline

The outline of this work is given below:

Chapter 2 – Background: The background chapter discusses relevant mathematical methods and related background information. It also provides an overview of view planning methods.

Chapter 3 – Related work: This chapter presents related work and gives a short discussion.

Chapter 4 – Next-best-view system: Gives details of the approach presented in this work and developed system.

Chapter 5 – Experiments Presents the hardware used in the experiments, evaluates the approach and discusses the results of the experiments.

Chapter 6 – Discussion and conclusion Gives a discussion of the complete work and discusses limitations and improvements for the presented approach.

2 Background

The following chapter presents background information and methods used in this work.

2.1 Optimization Problem

In general, a mathematical optimization problem has the form:

Let $x \in \mathbb{R}^n,\ f : \mathbb{R}^n \to \mathbb{R},\ g : \mathbb{R}^n \to \mathbb{R}^m,\ h : \mathbb{R}^n \to \mathbb{R}^l$. Find

$$\min_x f(x) \quad \text{s. t.} \quad g(x) \leq 0,\ h(x) = 0. \tag{2.1}$$

Vector x is the optimization variable, $f(x)$ the objective function, $g(x)$ are inequality constraints and $h(x)$ are equality constraints.
An optimization variable x^* is called *optimal*, if it has the smallest objective value among all possible optimization variables x_i which satisfy the constraints, i.e. for all x_i with $g(x_i) \leq 0$ and $h(x_i) = 0$: $f(x_i) \geq f(x^*)$.

Optimization methods which try to solve the optimization problem can be roughly categorized by the information of $f(x)$ they use:

Gradient-based methods: Use $\nabla f(x)$ to find an optimal solution.

2nd order methods: Use $\nabla^2 f(x)$ to find an optimal solution.

Convex optimization: $f(x)$ is a convex function, i.e. it holds that

$$f(ax + (1-a)y) \leq af(x) + (1-a)f(y), \text{ for all } x, y \in \mathbb{R}^n \text{ and } a \in [0, 1].$$

Black Box optimization: Only $f(x)$ can be evaluated.

These methods can be further categorized by the optimal solution they find:

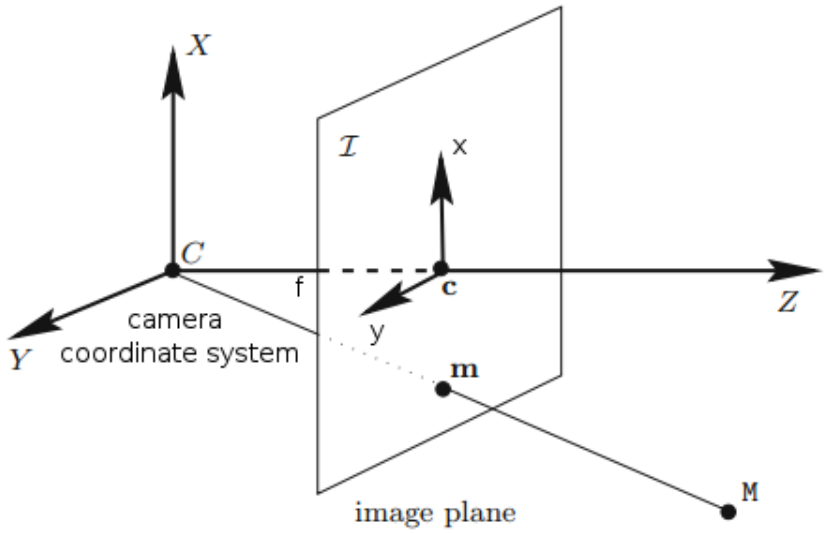


Figure 2.1: Pinhole camera model with a 3D Point M projected on the image plane as point m [Stu14].

Local optimization: Instead of seeking a solution over all feasible optimization variables, local optimization is concerned with finding a solution which is locally optimal, i.e. it is the minimal solution among feasible points which are near the minimal solution. This does not guarantee that there exists a better solution among all feasible points.

Global optimization: Global optimization considers all feasible optimization variables and finds the optimal solution. Usually, the trade-off to local optimization is efficiency. This trade-off is worth in situations where the value of an optimal global solution is higher than computation costs (e.g. for high value or safety-critical systems).

2.2 Pinhole Camera Model

The pinhole camera model (see Figure 2.1) describes the projection of a 3D point onto the image plane. The image plane is set at a distance f (called focal length) in front of the center of project C (also called optical center) of the camera coordinate system. A 3D point $M = (X, Y, Z)$ is projected onto the image plane as image point $m = (x, y)$. The coordinates of m can be computed with the following equations:

$$\frac{f}{Z} = \frac{x}{X} = \frac{y}{Y} \tag{2.2}$$

$$y = \frac{fX}{Z} \tag{2.3}$$

$$x = \frac{fY}{Z} \tag{2.4}$$

With homogeneous coordinates for m this relationship can be described with

$$m = \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}, \text{ with scaling factor } w \neq 0, \; u := wx \text{ and } v := wy. \tag{2.5}$$

The occurring matrix is called the projection matrix P.

The assumption of an ideal image plane are not present for real sensor and several properties might be differ more the ideal assumption. If the principle point c does not lie on the intersection of Z and the image plane, point m has to be translated to the shifted image plane center (u_0, v_0). The pixel dimensions h_u and h_v might be scaled. The image axes x and y might not be orthogonal to each other and have a skew angle θ. These five parameters are called the intrinsic camera parameters and can be described in the intrinsic matrix H.

$$H = \begin{bmatrix} h_u & -h_u \cot\theta & u_0 \\ 0 & h_v / \sin\theta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

If the camera coordinate system does not coincide with the world coordinate system, an additional rotation R and translation t has to be considered. This can be expressed with the extrinsic matrix E.

$$E = [R \mid t] = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \tag{2.7}$$

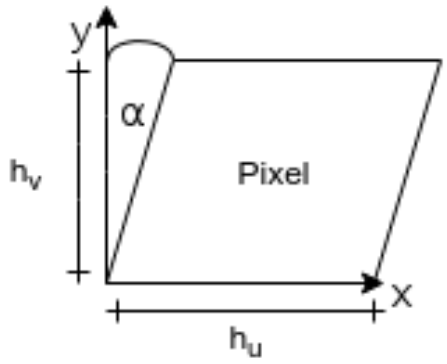


Figure 2.2: Illustration of skewed pixel and the related angle α.

The full projective mapping can then be expressed as:

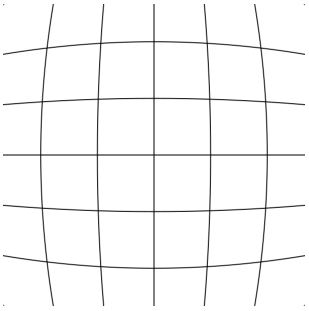
$$\begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} h_u & -h_u \cot\theta & u_0 \\ 0 & h_v / \sin\theta & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.8}$$

For camera calibration the project matrix P and the intrinsic matrix H are concatenated to form a matrix I.

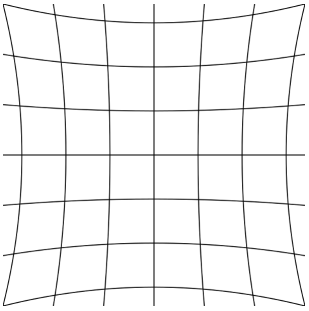
$$I = \begin{bmatrix} f_u & s & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

Here $s = f_v \tan\alpha$ is called the skew parameter (see Figure 2.2) and f_x and f_y are the focal distance parameters in pixel dimensions. Note $\alpha = \frac{\pi}{2} - \theta$ (with θ being the skew angle of the intrinsic matrix H, refer to equation (2.6)).
Another property of a real sensor are lens distortions (see Figure 2.3). The curvature of a lens leads to a distortion of the ray between a 3D point M and the optical center C. To compensate for this distortion in the projection, the distortion can be modeled as radial and tangential distortion:

$$\delta_x = \hat{x}(k_1 r^2 + k_2 r^4 + k_3 r^6 + \cdots) + [p_1(r^2 + 2\hat{x}^2) + 2p_2\hat{x}\hat{y}](1 + p_3 r^2 + \cdots) \tag{2.10}$$



(a) Barrel distortion.



(b) Pincushion distortion.

Figure 2.3: Two common lens distortion models [Hug10].

$$\delta_y = \hat{y}(k_1 r^2 + k_2 r^4 + k_3 r^6 + \cdots) + [2p_1 \hat{x}\hat{y} + p_2(r^2 + 2\hat{y}^2)](1 + p_3 r^2 + \cdots) \tag{2.11}$$

The radial distance r can be computed with $r = \sqrt{\hat{x}^2 + \hat{y}^2}$, k_i are the radial distortion coefficients and p_i are the tangential coefficients (also called decentering coefficients). With the distortion corrections (δ_x, δ_y) and the distorted image coordinates $(\hat{x}.\hat{y})$ the undistorted image coordinates (x, y) can be calculated via:

$$x = \hat{x} + \delta_x \tag{2.12}$$

$$y = \hat{y} + \delta_y \tag{2.13}$$

2.3 View Planning Methods

This section provides an overview of different view planning methods. Generally, these can be categorized into model-based and non-model-based methods. Model-based methods use a priori knowledge of an object or the environment to compute a view plan offline. This leads to a faster execution of taking sensor measurements as no new views have to be computed between view poses. Non-model-based methods use minimal to no a priori knowledge and select a view based on observed properties of the environment or an object. The advantage of non-model-based methods is their application to provide views in unknown environments. This summary is based on the paper of Scott et al. [SRR03].

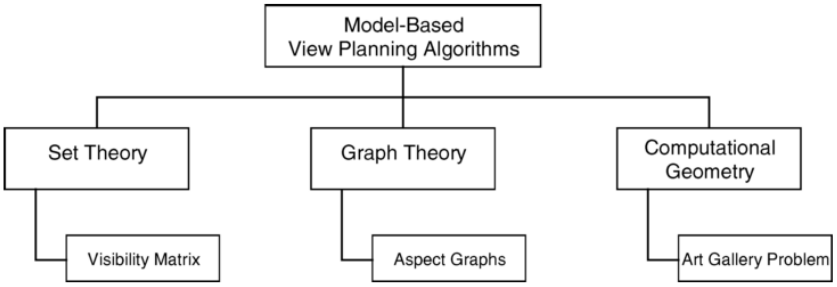


Figure 2.4: Overview of model-based view planning methods [SRR03].

2.3.1 Model-Based View Planning

Model-based view planning can be categorized based on the representations used for the model. Generally, model-based view planning methods compute a complete sequence of view poses with the model as a priori knowledge. The effort required for the model creation is usually compensated by shorter view plans, i.e. smaller amount of different view poses are required to achieve a task compared to non-model-based methods. Usually, these approaches are used for high fidelity scanning of multiple similar objects with complex geometries e.g. for quality inspection [MBND10].

Set Theory - Visibility Matrices

A discrete space around the object is encoded with some visibility metric in a single data structure (visibilty matrix) for each pose in a quantized viewpoint space. This visibility metric can be generalized to a measurability metric, which encodes measurement quality for each element in the discrete space. In "Performance-Oriented View Planning for Model Acquisition" [SRR00], the view planning task to find a set of viewpoints measuring the object surface is formulated in a set-theoretic manner as:

$$\text{Find } \{v_j\} \text{ s. t. } \{S_j\} \supseteq S. \tag{2.14}$$

Here v_j are viewpoint positions of the viewpoint space V and S_j is a set of object surface elements measurable by a viewpoint v_j. S_j is a subset of the object surface space S. In [TG95] this problem was identified to be isomorphic to the set covering problem, which makes the view planning problem a NP-complete problem.
This method requires an accurate object model for the computation of a visibility matrix before any view-planning algorithm can be used.

Graph Theory Method - Aspect Graphs

Aspect graphs provide a complete enumeration of all possible “distinct” views of an object in a defined viewpoint space. The aspect graph is commonly defined as follows [EBD92]:

- A node represents a view of the object which is part of the viewpoint space and
- an arc represents a transition between two neighboring views.

Algorithms for aspect graphs differ in their definition of the viewpoint space and, what a “distinct” view of the object is and how the transition between views is defined. The two basic models for viewpoint space are the viewing sphere and a general model of poses in 3D space. In the viewing sphere model, a unit sphere is placed in the center of the object and a point on the sphere defines a viewpoint direction toward the object origin. In the general 3D space model, a viewpoint is defined through its position and orientation in 3D space. In [HK89], S. Hutchinson and A. Kak define an aspect, or distinct view, of an object as an edge or multiple edges of the object. Arcs in their graph represent adjacent viewpoints. Views which see the same features are grouped in the same node. To determine the best sensing strategy, S. Hutchinson and A. Kak use the aspect graph to search for the minimal amount of views which will see all aspects of an object. To compute the aspect graph for an object requires either an exhaustive examination of the object or an analytical approach.
As aspect graphs encode a large amount data for complex object geometries, the computational complexity of finding a next-best-view is a major disadvantage. The offline calculation of the view-planning strategy allows for no reactive action of the view planning system in uncertain or dynamic environments.

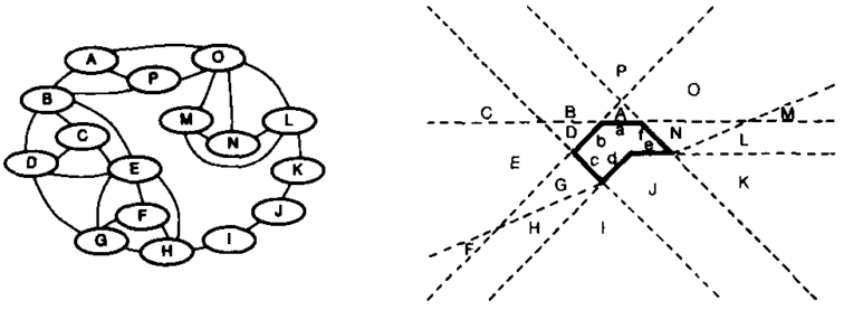


Figure 2.5: An aspect graph (left) with nodes being viewpoints gazing at edges of the object (right) and arcs connecting nodes representing viewpoint adjacency [HK89].

Computational Geometry Methods - Art gallery problem

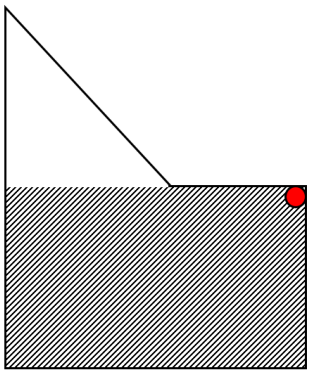


Figure 2.6: An “art gallery” with a guard (red dot) and the area it can view (shaded area).

The question: "Given the complete geometry of a scene, what are the “best” viewpoints for data acquisition?" is a well-known problem in two dimensions, namely the art gallery problem (also called museum problem or watchmen problem). The art gallery problem states: “Given a polygon P (floor plan of an art gallery), what is the upper bound on the number of ‘guards’ and their pose represented as points such that all interior walls of P are visible.” Solving this problem for 3D and a CAD model of an object can be used to acquire view poses which can see the complete surface of an object.
In the general art gallery problem there is no measure of visibility, no modeling of the view capabilities of the “guard” and the problem is restricted to 2D polygons. Even though view poses can be computed which would see the complete surface of an object, the problem lacks a measure to assess the quality of such view poses.

2.3.2 Non-Model-Based View Planning

Non-model-based view planning methods can be classified by their domain of reasoning of viewpoints (see Figure 2.7). These are: *surface-based*, *volumetric* and *global* [SRR03]. The following section will give an overview of these categories and approaches which use the different domains.

Surface-based Methods

Surface-based methods use features and properties of the surface of the object of interest for view-planning. In the following paragraphs, three methods will be introduced which

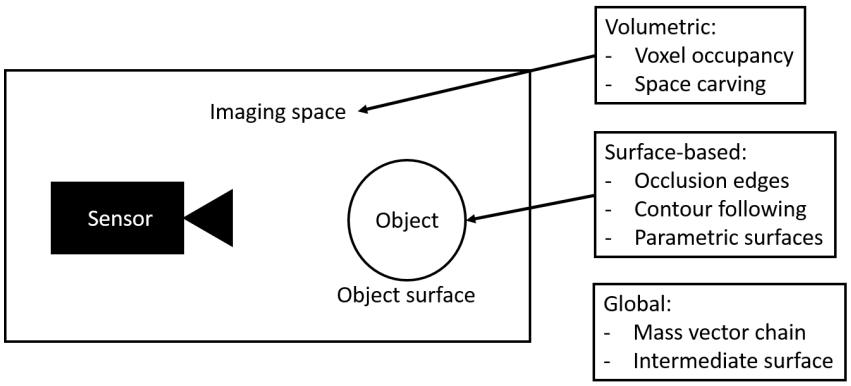


Figure 2.7: Overview of non-model-based view planning methods [SRR03].

use different properties of features of the surface of an object to compute a view of the object.

Occlusion Edge Methods An occlusion edge method utilizes geometric jump edges. These are caused by volumes which obstruct the view of other volumes. As such, they can be used as an indicator for the next-best-view pose to position the view in such a pose as to expose the obstructed volume in the next view. Another kind of edge are boundary (occlusion) edges, which represent the boundary of a volume (see Figure 2.8). This approach requires an extensive computation of visibility and a correct and reliable classification of edges as boundary edges or geometric jump edges. Occlusion edge methods can be included in extensive view-planning methods to, for example, penalize views which lead to occlusion. This can be seen in the work “Occlusion-free path planning with a probabilistic roadmap” [BDL+08].

Contour Following Having located boundary edges of an object, a contour following method will follow the contour of the object by keeping a certain distance between the camera and the object (see Figure 2.9).
One of the major drawbacks of this method is the requirement of a collision-free environment for the camera as there is no direct way in the method to detect collisions. The second major drawback is execution time for acquiring a complete view of the object. The advantage is the simplicity of the method.

Parametric Surface Representation - Superquadric model Superquadric models can be used to represent observed object surfaces. Given the view of an object, the parameters of the superquadric model can be iteratively improved by selecting views where the current superquadric model does not fit the image data very well. This process continues

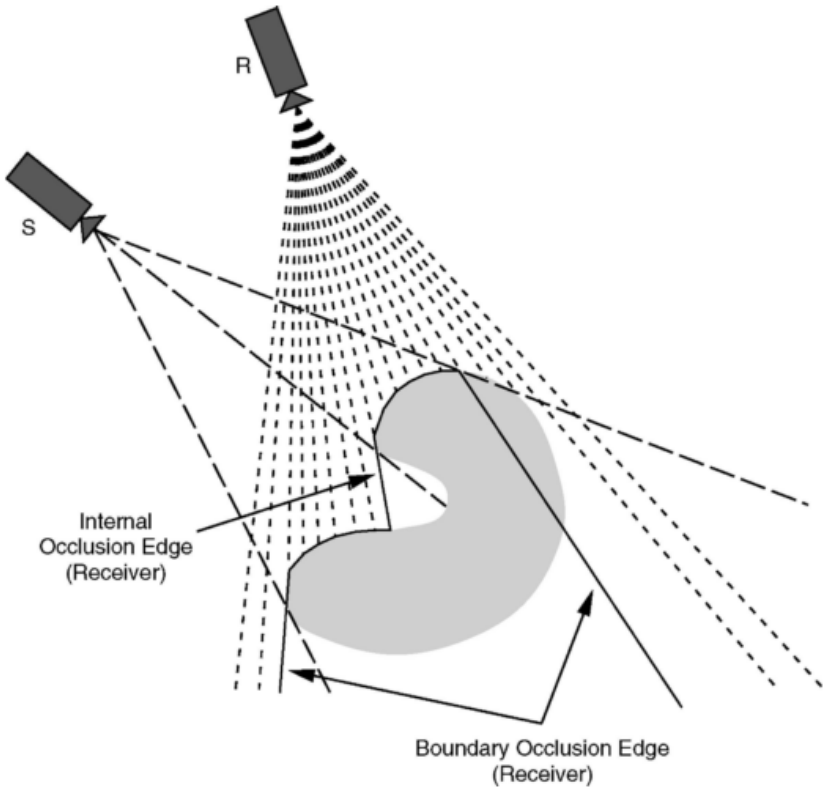


Figure 2.8: Example of an occlusion edge caused by the obstruction of the view with another volume and boundary edges which represent the boundary of a volume [SRR03].

until the superquadric model represents the sensed object with a satisfactory quality. The approach requires a segmentation of the object data and the fitting of superquadric models to the segmentated parts, as computing a superquadric model for object with complex geometries is not feasible.

Volumetric-based Methods

Volumetric-based methods use their knowledge of the imaging space to plan the next-best-view. The goal is to choose the next-best-view which provides the greatest reduction in uncertainty about the imaging space.

Voxel Occupancy Methods Space occupancy is encoded by a voxel occupancy grid (see Figure 2.11) or an octree (see Figure 2.12). A next-best-view method can utilize the occupancy representation of seen and unseen space to determine the next view.

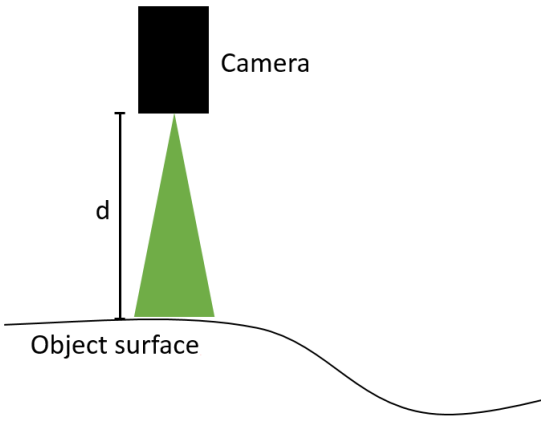


Figure 2.9: Setup for a contour following method, where the camera keeps a certain distance d from the surface of the object.

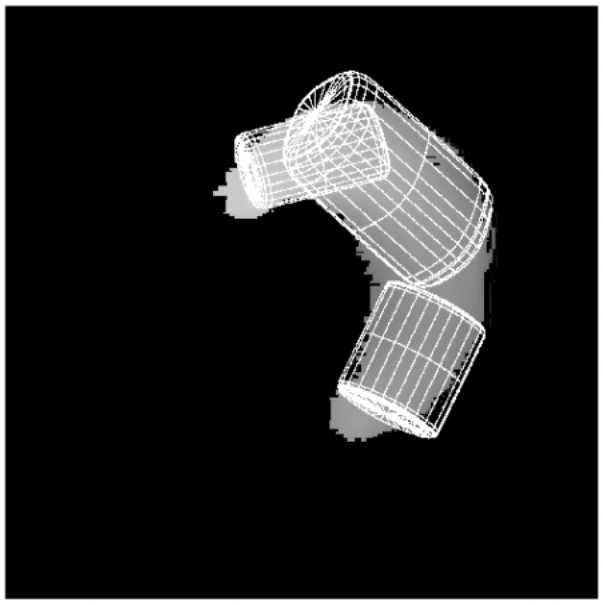


Figure 2.10: Three superquadrics (white wireframe models) fitted to a segmented part of a range image [LJS97].

It's suitable for a coarse representation of surfaces but not useful for high-precision modeling. Grid occupancy storage can have high memory requirement for a small voxel size and big grid dimensions but this can be reduced when using an octree structure to store voxel occupancy. Due to the discretization of the space, one should be aware of misalignment errors.

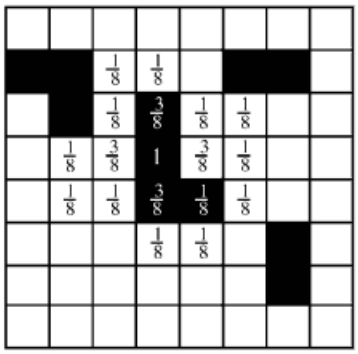


Figure 2.11: Voxel occupancy grid encoding different occupancy probability of voxel in the "L"-shape which might occur for a rotated L-shape [Gre02].

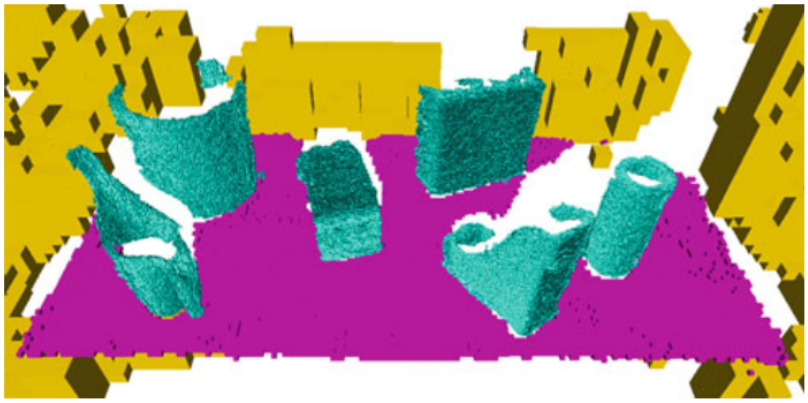


Figure 2.12: Octomap which uses the octree data structure for voxel occupancy mapping. Three octomaps with different resolutions are displayed in different colors [HWB+13].

Space Carving Using a preplanned search pattern (see Figure 2.13), the vision sensor is moved through the imaging space while taking measurements.
The pattern has to include a preplanned collision avoidance or an active collision avoidance as the search pattern is preplanned. Through continuous movement in the pattern this generates views with a high degree of overlap.

Global-based Methods

Global-based methods derive a view-planning strategy from global characteristics of the geometric data of the object.

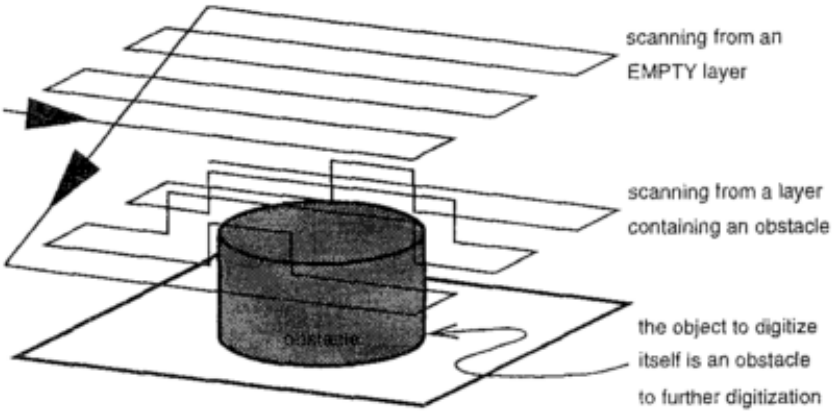


Figure 2.13: Spacecarving with a preplanned search pattern (zig-zag pattern) as well as collision avoidance [PS97].

Mass Vector Chain A mass vector chain is a series of weighted vectors. In [Yua95], X. Yuan represents surface areas as such weighted vectors. For closed surface boundaries, a mass vector chain form a closed chain. A next-best-view direction can be selected as the negative of the computed mass vector chain, which will eventually lead to a closed chain (see Figure 2.14).
Although this method only estimates a view direction, it can be used as a coarse initialization step or a cue for a viewing direction for more refined view-planning strategies.

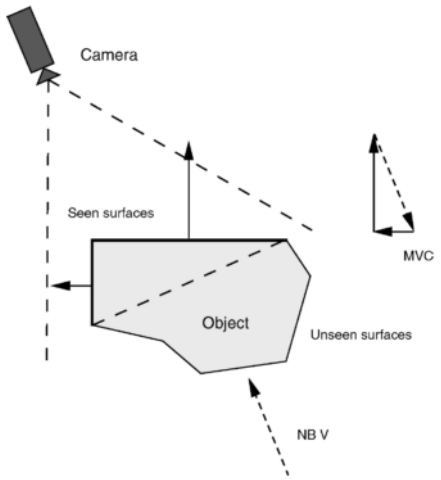


Figure 2.14: Exemplary calculation for a NBV using the mass vector chain. [SRR03].

Intermediate Space Representation An intermediate space representation encodes information of visibility of an object surface related to a sensor placement in a virtual

surface positioned in the imaging space. With the help of such an intermediate representation of visibility, unseen surfaces of the object are chosen for the next-best-view.
An advantage of this approach is the decoupling of computational complexity of visibility measure from the size of the viewpoint space.

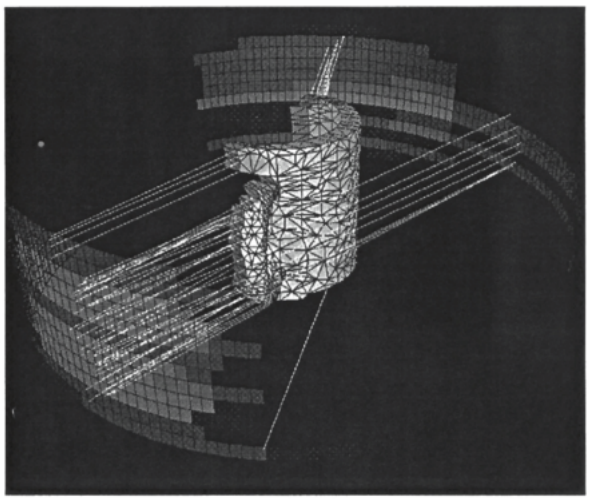


Figure 2.15: Intermediate surface representation of a partially observed coffee mug [Pit95].

For this work, a volumetric method is used. As the environment is assumed to be dynamic, a non-model-based approach is required which uses minimal a priori knowledge of the environment for view planning. AS the task of exploration requires the knowledge and representation of unknown space to determine a view which would see such unknown space. Another advantage of a volumetric method for this work, is the representation of occupied space, which a collision avoidance process can utilize to compute safe motions of a robotic system in the environment.

2.4 Robot Operating System - ROS

The Robot Operating System (ROS) is a “meta” operating system for robots and is currently maintained by Willow Garage[1]. It provides a collection of algorithms relevant for robotic applications and software building tools. Various sensors and actuators are supported through drivers. It is open source software and software is developed by the robotic community. It provides an architecture for inter-process communication to

[1]Willow Garage, *Willow Garage*, [Online] Available: http://www.willowgarage.com/ [Accessed: 14.08.2016]

Figure 2.16: Three ROS nodes (displayed as oval nodes) communicate with each other over two topics (rectangular nodes). The directed edges connecting the nodes and topics indicate the direction of the data flow.

enable the sharing of data between software components. The communication is realized as asynchronous RPC-style communication and synchronous message communication with services. The communication is handled in a peer-to-peer network, called the *Computation Graph* (see Figure 2.16). The basic components of the Computation Graph are nodes, Master, Parameter Server, messages, services, topics and bags. Their functionality will be briefly summarized.

Nodes: Nodes are the essential computation unit of the network. Usually, a system has many nodes which exchange data and execute computations. For example, one node can provide vision sensor measurements as point clouds which are used by another node for point cloud specific computations.

Master: The ROS Master acts as a nameservice and provides registration and lookup for the rest of the Computation Graph.

Parameter Server: Nodes use the parameter server to store and access parameters at runtime. Its intended use is to provide global access to configuration parameters and is executed as part of the ROS Master.

Messages: Messages are simple data structures containing typed fields. Primitive types (like integers, boolean, etc.) as well as array of primitive types are supported. Nodes use messages for data exchange with other nodes.

Topics: Topics are named buses for transporting messages between nodes. They utilize the publish-subscribe pattern. Nodes can publish data on a named topic and nodes which require that data can subscribe to the named topic. The ROS Master keeps track of these topics and enables nodes which publish/subscribe to a topic, to communicate with each other. Topics support multiple subscribers and publishers. Communication with topics is one-way, from publisher to subscriber.

Services: As an alternative to the asynchronous one-way communication of topics, services provide a synchronous request-reply communication style. Requests and replies are defined as message structures. Nodes provide a named service and other nodes can send a request to that named service and wait for a reply message.

Bags: Bags store ROS message data. They can be played back and used for developing and testing algorithms.

3 Related work

In [DDN03] an active object recognition approach is presented which selects viewpoints based on reinforcement learning approach. In reinforcement learning, the goal is to select an action a given a state s to increase a reward r. F. Deinzer et al. formulate the state s as a series of images and camera poses, actions a as the execution of a camera movement and the reward r is given by the object recognition which measures the quality of a given viewpoint. The NBV problem is then formulated as choosing an action a_t at a given time t which will maximize the accumulated future rewards, called the return R. As these future rewards can not be observed at time t, an action-value function is trained which estimates the expected return for a given state and action at time t. The viewpoint selection is done in two steps. First, the training of the action-value function and then in the second step for an observed state s, an action is chosen which maximizes the action-value function. In the experiments, the training of the action-value function took around 600 pairs of state and actions for the classification of two types of cups. Camera movements were restricted to a circular motion. It could be shown that given the autonomously trained action-value function, the NBV component chose a sequence of viewpoints which were near a theoretical optimal viewpoint sequence.
The approach has shown that a sequence of viewpoints for object recognition could be chosen with no prior knowledge of the object classifier or the objects. Although, a short sequence of viewpoints could be chosen which is near a theoretic optimum, the trade-off is the training of an action-value function which requires a large amount of training data.

In the approach of M. Trummer et al., presented in [TMD10], view poses are chosen by uncertain feature point poses. The goal of their work is to produce an accurate 3D reconstruction of unknown objects. They combine the 3D reconstruction process with a guided KLT (GKLT) feature tracker. For each feature detected by the GKLT, the pose of the feature is computed and the covariance matrix for the feature pose is determined. A view pose is determined which is perpendicular to eigenvector v with the corresponding largest eigenvalue λ of the covariance matrix. This approach is an extension to the statistical E-Criterion. With this approach, the largest eigenvalue λ is reduced. As the largest eigenvalue is an estimation of the uncertainty of a feature point pose, a view perpendicular reduces this uncertainty. A camera is moved towards this view pose while continuously taking pictures of the object. In the experiments an eye-in-hand robot

setup was used with an intensity camera. During an initialization step, five pictures of the object were taken from random view poses to receive an initial set of feature points and their 3D pose. It could be shown that compared to random view pose selection, the NBV planning of a view pose with respect to the extended E-Criterion reduced the reconstruction error faster and produced overall a smaller reconstruction error.
The approach shows the selection of viewpoint direction based on the uncertainty of feature point poses but gives no method for handling reachability of the viewpoint or handling occlusions in the viewpoint pose. Camera movement is restricted to a dynamic spherical motion model, which allows for changes in the sphere size and position. The camera is always directed at the center of the sphere.

The thesis of S. Wenhardt [Wen13] presents an approach for 3D reconstruction which utilizes an extended Kalman filter to formulate the 3D reconstruction process as a state estimation problem. A state z is defined as a set of 3D reconstructed points with an observation o defined as 2D points observed with a camera. The method reduces the estimated quadratic error of an estimated state $\hat{z}$ and the actual state z. The estimated quadratic error was identified as the trace of the a posteriori covariance matrix of the state z. Using D-,T- or modified E-Criterion of covariance matrices, an optimal view pose is determined which reduces the estimated quadratic error. The modified E-Criterion is formulated as the minimization of the average maximal eigenvalues of the 3D points. Furthermore, the optimization formulation considered viewability of points with a sensor and occlusions in the scene. The set of candidate viewpoints is determined by the reachable workspace of the robotic system, which is described with the Denavit-Hartenberg matrix. An optimal viewpose D-,T- or modified E-Criterion is determined by a global evaluation of the candidate viewpoint set. With this approach, the global optimal viewpoint can be determined. For an initial estimation of the state variable, the camera is moved to predefined positions. During the experiments various objects with increasing geometric complexity were used to compare the D-,T- and E-Criterions for the reduction of the 3D reconstruction error. These were also compared to an approach which put the camera at random positions and one with uniformly sampled camera positions. The experiments showed that the three optimal criterions outperformed the random camera position process and the uniformly sampled camera position approach. Among the three optimal criterions, the modified E-Criterion showed the best performance regarding complex geometric objects.
The use of the Denavit-Hartenberg matrix allows this approach to formulate a reachable workspace. This is an important consideration for eye-in-hand robotic systems, as only view poses which can be reached can be sensed. No information regarding the computation time for the selection of the viewpoint was given.

An approach for environment exploration was presented in [MAC16]. The environment for view planning was determined as a sphere around a point of interest. The point of interest represents the location an object in the environment which was removed

from the environment, placed at a new location or a new object was introduced to the environment through human interaction. The location of these points of interest was determined by using a hand-tracking system. To determine the next view pose, an approach was used which is based on the approaches presented in [BWDA00] and [Con85]. Here, a spherical sensor motion model is used, with the sensor viewpoint directed at the centre of the sphere. The environment is modeled as a 3D volumetric voxel grid. The voxels can have occupancy states which are determined by a truncated signed distance function. After a sphere of interest is determined, the voxels contained in the sphere are set to the occupancy state unknown. View poses were evaluated by the amount of unknown voxel which are in the field of view of the sensor and inside the sphere of interest. Furthermore, the NBV uses the GPU for a more time efficient computation of raycasts in the voxel map. The experiments showed a significant computation time improvement, around 200%, of the NBV computation using a GPU versus a CPU for evaluation.
This approach is similar in formulation of the information gain of a view pose to this work. The area of exploration is determined dynamically due to detected changes in the environment. In this work, the complete environment is considered to be unknown and sensor motions are not restricted to a sphere. Furthermore, no collision avoidance for reachability of view poses is considered.

In the thesis of M. Suppa [Sup07] the exploration of configuration space (C-space) and the environment is combined in one exploration system. The goal of C-space exploration is to increase the knowledge of manoeuvrability of a robot in a partially unknown environment. From vision sensor measurements the environment is sensed and robot configurations are identified which can be in a collision state or in a collision-free state. The exploration of the environment is concerned with the task of object modelling. Additional tasks include object tracking and object recognition, which are mentioned in the work but no explicit method for achieving these objectives is given. To combine the different tasks for the selection of a sensor pose s, each task computes a task-specific information gain I based on a world model which all tasks use. Entropy of the information gain is used as an indicator for the NBV selection. Each information gain and therefore each task can be weighted for NBV selection. Resulting in the formulation of the NBV, for the different tasks i and task-specific weights w_i

$$s_{\text{max}} = \arg\max_{s} \sum_{i} w_i I_i \tag{3.1}$$

This results in a viewpose s_{max}, which is optimal with regard to weighted information gain metrics. For a viewpose acquired from the NBV component, a robot configuration q is determined and verified that it is a collision-free configuration. In case the configuration is in a collision state, the measurement can not be taken and no further strategy is given in such a case. If the configuration is in an unknown space with no known configuration

state, a C-space exploration at that viewpose is required. In case the configuration is collision-free, a motion planner seeks a path between the current configuration and the goal configuration. If a path is found then the robot moves to the goal configuration and continues the exploration process. If no path is found due to obstacles, the execution is stopped. The C-space exploration and environment exploration were tested in a simulation framework and further evaluated on a real robot system. The experiments showed an increase of the knowledge of the configuration space but could not explore the complete C-space.
This is an interesting approach and is, to the authors best knowledge, the first approach to combine multiple tasks into one view for view planning. The approach requires task-specific information gains which can be combined to an overall information gain of a pose. This is not the case in this work and different tasks can have different metrics for the evaluation of a pose. The combination of a of task-specific view poses is independent of the task-specific evaluations. Furthermore, the task-specific weights in the approach of M. Suppa are fixed and do not adapt to the state of the environment or the task-specific goals. Although object recognition is mentioned in this work and an architecture is given for the incorporation of the object recognition task, no further details were given. An interesting part of this approach is the exploration of the configuration space.

4 Next-best-view system

The following chapter will present the approach and gives an overview of the system architecture.

4.1 General overview

Autonomous exploration of the environment is achieved through a non-model-based view planning approach which uses a volumetric representation of the environment. To combine the task of object recognition and autonomous exploration of the environment, a next-best-view system is developed. A next-best-view component computes view poses based on a volumetric environment map, which is derived from a volumetric mapping system. The next-best-view system consists of a RGB-D sensor (see Section 4.2), an

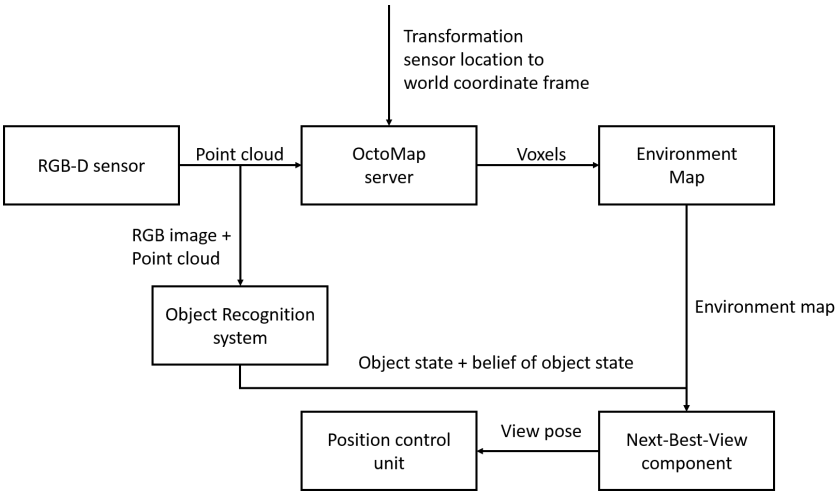


Figure 4.1: Basic overview of the next-best-view system.

environment mapping system (see Section 4.3), an environment map (see Section 4.4) derived from the environment mapping system and the next-best-view component (see Section 4.5). Additionally, an object recognition system and a position control unit for the robotic system are present, but were not developed in this work. The RGB-D sensor

sends its depth measurement as a point cloud to the mapping framework OctoMap which computes occupied and free voxels based on the provided point cloud. Given the transformation of the sensor location to world coordinate frame, the OctoMap server computes the location of the voxels in world coordinates. With the resolution and location of the occupied and free voxels, an environment map is updated with the new measurements (see Figure 4.1). Based on this map, a new sensor pose is computed in the next-best-view component to explore the environment and improve the object recognition. This pose is given to a position control unit, which moves the robotic hand with the mounted RGB-D sensor to the given pose. From the new view pose, new measurements of the environment are made. The position control unit for the positioning of the robotic arm in the workspace already exists and was implemented in MATLAB. An object recognition system is also provided in the available robotic system. The environment map and next-best-view component are implemented in MATLAB, while the communication with the RGB-D sensor is achieved through a ROS node as well as the OctoMap server.

The following tasks were done to implement the next-best-view system:

- Integration of Kinect v2 in the existing system with the IAI Kinect2 package (kinect2_bridge ROS node) [Wie15].
- Camera calibration of Kinect v2.
- Hand-Eye calibration for Kinect v2 and robotic arm.
- Integration of OctoMap by using octomap_server package available in ROS.
- Implementing 3D grid map (environment map) in MATLAB and validating communication between kinect2_bridge, octomap_server and MATLAB.
- Implementation of next-best-view component in MATLAB which provides a viewpoint pose for the Kinect v2 mounted on the robotic arm.
- Validation of the complete system and experiments regarding exploration.
- Integration of object recognition metric into next-best-view system.
- Validation and experiments of next-best-view system with object recognition metric.

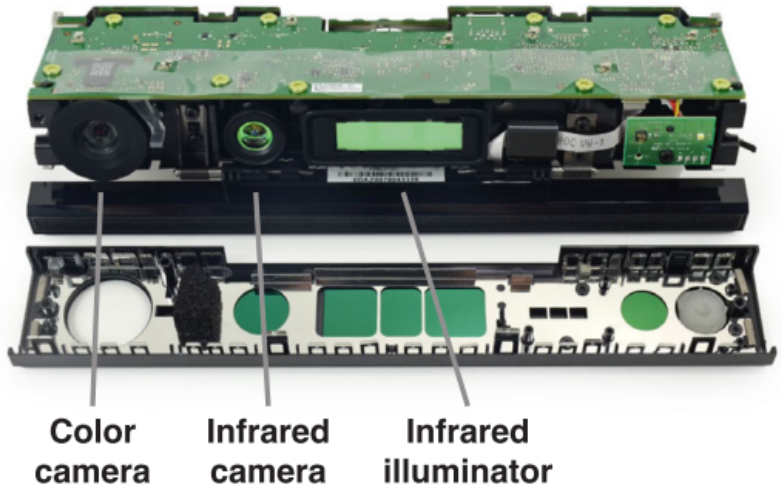


Figure 4.2: Kinect v2 sensor with the front panel removed. The location of the RGB camera (color camera) and the depth sensor (Infrared camera) are shown [BR14].

4.2 RGB-D Camera - Kinect v2

For the acquisition of depth measurements the second generation of the Kinect sensor, following called Kinect v2 (also called Kinect for Xbox One), was used. It is equipped with an RGB sensor, IR emitter and receiver for depth measurements and a microphone array. The technical specifications can be seen in Table 4.1 [1]. Compared to the first generation of the Kinect, Kinect v1 (also called Kinect for Xbox 360), the Kinect v2 has a larger RGB and depth resolution, with a larger field of view as well as a smaller minimal depth range and higher maximal depth range by default [Zen14]. This enables the Kinect v2 to sense a larger space with a higher resolution. For measuring depth, the Kinect v2 employs the Time-of-Flight (ToF) method. The ToF method is based on measuring the time a light, emitted by an illumination unit, requires to travel from the illumination unit to an object where it is reflected and received by a sensor (see Figure 4.3). For further technical details, please refer to “The Xbox One System on a Chip and Kinect Sensor” [SO14]. The sensor exhibits an oscillating depth distortion error of ±6 mm [BR14] and requires a pre-heating phase of around 20 minutes for reliable depth measurements [LMM+15].

[1] Microsoft, *Kinect hardware*, [Online] Available: https://developer.microsoft.com/en-us/windows/kinect/hardware [Accessed: 25.06.2016]

RGB resolution [px]	1920 x 1080
Depth resolution [px]	512 x 424
Depth FOV (h x v) [°]	70 x 60
Depth range [m]	0.5 - 4.5
Minimum latency [ms]	20
Dimensions (l x w x h) [mm]	249 x 66 x 67 (+/- 0,3175)
Weight [kg]	(approx.) 1.4
Connection Type	USB 3.0

Table 4.1: Summarized technical specification of Kinect v2.

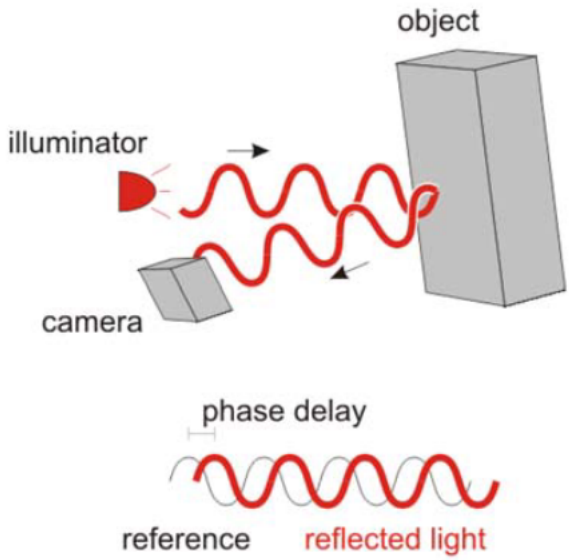


Figure 4.3: Basic concept of the ToF method [Esp12] with phase modulation which is internally used by the Kinect v2 sensor.

4.2.1 Kinect v2 connection to ROS

To connect the Kinect v2 into ROS, the package *IAI Kinect 2* [Wie15] developed T. Wiedermeyer was used. It consists of a camera calibration tool for the RGB and IR sensor, a bridge (kinect2_bridge) for publishing relevant Kinect v2 data (e.g. point clouds, RGB images or camera info) on ROS topics, a viewer for the RGB image data and point cloud data and a library for the registration of depth measurements and RGB data.
Registered point clouds are provided in Full HD resolution (1920 x 1080 px) and quarter Full HD (QHD) resolution (960 x 540 px). The raw depth images from the sensor are also available with a resolution of 512 x 424 px. The raw depth data requires no further calibration as internal sensor data is used for registration of RGB and IR data. Full

HD and QHD point clouds require a camera calibration for a correct registration of RGB and IR data. The point clouds are published as '*sensor_msgs/PointCloud2*' ROS messages [2]. The *kinect2_bridge* has several parameters which influence the properties of the registration process and properties of the point cloud. For this work, the relevant parameters are:

publish_tf = [*bool*] Publishes the static transformation frames *kinect2_link*, *kinect2_rgb_optical_frame* and *kinect2_ir_optical_frame*. The *kinect2_rgb_optical_frame* corresponds to the location of the RGB sensor of the Kinect v2 and *kinect2_ir_optical_frame* corresponds to the location of the IR receiver. The *kinect2_link* is used as the camera base link and is located at the same location as *kinect2_rgb_optical_frame*.

fps_limit = [*double*] Defines the limit on the frames per second (fps) for publishing the point cloud data. If no value is set, the kinect2_bridge tries to publish the point cloud as fast as possible which puts a heavy load on the CPU.

max_depth = [*double*] Sets the maximal depth [meters] for the depth measurement. It can be used to clip the range of the depth measurement to a relevant maximal depth and remove data not relevant for point cloud processing.

4.3 OctoMap

OctoMap is a 3D mapping framework which utilizes octrees to generate 3D volumetric models of the environment. It is available as a C++ library[3] as well as a ROS package[4] and is presented in the work "OctoMap: an efficient probabilistic 3D mapping framework based on octrees" [HWB+13].

An octree is a graph-based hierarchical data structure. Each node in an octree represents a cubic volume, usually called voxel. These voxels are recursively divided into eight children until a given minimum voxel size is reached (see Figure 4.4). The minimum voxel size determines the resolution of the octree (see Figure 4.5). Voxels are marked as occupied according to the measurements acquired from a range sensor. Free voxels are determined as the area between the sensor origin and measurement end points. Any area not marked as free or occupied is implicitly modelled as unknown area.

[2]ROS, *sensor_msgs/PointCloud2*, [Online] Available: http://docs.ros.org/api/sensor_msgs/html/msg/PointCloud2.html [Accessed: 25.06.2016]
[3]K.M. Wurm, A. Hornung, *OctoMap*, [Online] Available: http://octomap.github.io/ [Accessed: 25.06.2016]
[4]ROS, *octomap*, [Online] Available: http://wiki.ros.org/octomap [Accessed: 25.06.2016]

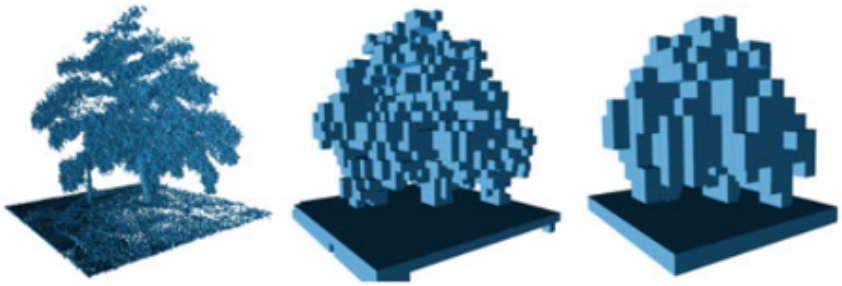


Figure 4.4: Representation of voxels at different resolutions [HWB+13].

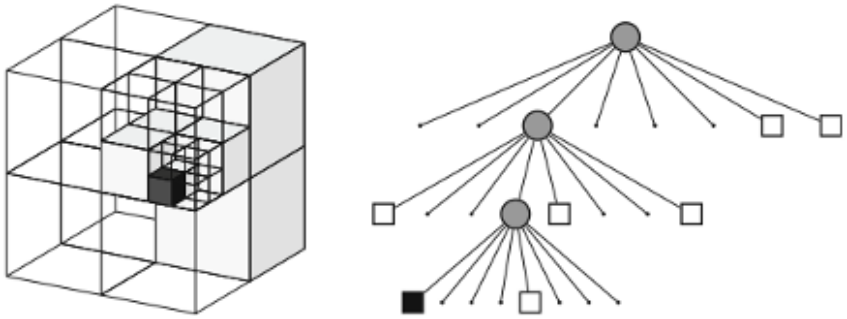


Figure 4.5: Octree representations once in a volumetric representation (*left*) and the corresponding tree structure (*right*). Occupied voxels are shaded *black* and free voxels are shaded *white* [HWB+13].

Occupancy in the OctoMap is represented probabilistically. Volumetric elements (Voxels) can have three states: *unknown*, *occupied* and *free*. This is a useful classification for this work as it enables to restrict movement of a robotic system to only free space to avoid collisions with objects in the environment, which have been measured and mapped as occupied. Furthermore, unknown space can be avoided as it can contain objects not yet sensed by the sensor and therefor avoid collisions with those objects. Additionally, the representation of unknown space is of special interest for visual exploration tasks as the task involves the planning of actions which reduce the amount of unknown space. As the next-best-view component in the system only provides a pose for the sensor in a free voxel, the collision avoidance with the environment can only be guaranteed for this component and not the whole geometry of the robotic arm. The probabilistic representation of occupancy is an important property of the OctoMap as sensor measurements are afflicted by different error sources. The noisy sensor measurements can be taken into account as underlying uncertainties. By fusing multiple measurements, a robust estimation of the true state of the environment can be made. The sensor fusion is implemented as log-odds for numerical stability. Given previous sensor measurements

$z_{1:t-1}$ and the current sensor measurement z_t, the occupancy for voxel n given $z_{1:t}$ can be formulated with log-odds as:

$$L(n \mid z_{1:t}) = L(n \mid z_{1:t-1}) + L(n \mid z_t) \tag{4.1}$$

The inverse sensor model $L(n \mid z_t)$ depends on the specific sensor used for making measurements in the environment. In [HWB+13], the sensor is assumed to be beam-based, i.e. endpoints of a measurement correspond with objects in the environment that reflect the beam. This is also true for the Kinect v2 sensor used in this work. The traversed voxels by a beam are determined by a ray-casting algorithm. For a voxel n and a measurement z_t the inverse sensor model can be described as:

$$L(n \mid z_t) = \begin{cases} l_{occ} \text{ if beam is reflected in voxel n} \\ l_{free} \text{ if beam traversed voxel n} \end{cases} \tag{4.2}$$

4.3.1 OctoMap in ROS

In this work the *octomap_server* ROS package is used, which computes OctoMaps and publishes them as '*octomap_msgs/Octomap*' ROS messages[5]. In the '*octomap_msgs/Octomap*' message the OctoMap is serialized for data transportation and can be deserialized to a OctoMap with functions in the C++ library or with the ROS package '*octomap_msgs*'[6]. The *octomap_server* subscribes to the topic '*cloud_in*' with message type '*sensor_msgs/PointCloud2*'. This is provided by the *kinect2_bridge* node. The *octomap_server* also requires a transformation from the sensor data frame to a static *map* frame. With this transformation the point cloud received from the sensor can be transformed into world coordinates frames for the proper computation of the voxels center points in world coordinates in the OctoMap. In this work the *map* frame is set to be equal with the *world* frame. The *world* frame is located at the base of robotic arm and is in the same plane as the objects of interest. The location of the sensor data frame (*kinect2_link*, see Subsection 4.2.1) in world coordinates is determined by a hand-eye calibration (see Section 5.3).
The *octomap_server* publishes the following topics:

octomap_binary as '*octomap_msgs/Octomap*': The compact binary version of the OctoMap which stores only free and occupied voxel states.

[5]ROS, *octomap_msgs/Octomap*, [Online] Available: http://docs.ros.org/api/octomap_msgs/html/msg/Octomap.html [Accessed: 25.06.2016]
[6]ROS, *octomap_msgs*, [Online] Available: http://wiki.ros.org/octomap_msgs [Accessed: 25.06.2016]

octomap_full as '*octomap_msgs/Octomap*': Publishes the full state of the OctoMap.

occupied_cells_vis_array as '*visualization_msgs/MarkerArray*': The center of all occupied voxels as box markers with different scaling. It is used for visualization of occupied voxels in RViz.

free_cells_vis_array as '*visualization_msgs/MarkerArray*': The center of all free voxels as box markers. By default, this is not published but can be enabled with the *publish_free_space* parameter.

octomap_point_cloud_centers as '*sensor_msgs/PointCloud2*': The center points of all occupied voxels as a point cloud. As points in a point cloud have no volume (opposed to the box markers used in occupied_cells_vis_array), the visualization will contain gaps between voxel center points. The gaps vary in size with the different voxel size resolutions.

projected_map as '*nav_msgs/OccupancyGrid*': A 2D downprojected occupancy map of the 3D OctoMap.

The following parameters are of special interest for this work:

resolution = [float] Defines the minimal resolution (in meters) of voxels in the map.

sensor_model/max_range = [float] Maximum range of measurements integrated into the map from the provided point cloud in the topic '*cloud_in*'. This can be set to a value relevant for the mapping area.

sensor_model/(hit | miss) = [float] Probabilities for the inverse sensor model for hitting (occupied voxel) and missing (free voxel). The default values are 0.7 for hit and 0.4 for a miss which correspond to $l_{occ} = 0.85$ and $l_{free} = -0.4$ (refer to equation (4.2))

publish_free_space = [bool] Enables the publishing of *free* voxels on the topic '*free_cells_vis_array*'.

4.4 Environment Map

The environment map is implemented as a 3D grid map with evenly spaced voxels with a constant voxel resolution (see Figure 4.6). A voxel in the map can have three occupancy states, namely *unknown*, *free* and *occupied*. A voxel, not yet sensed by the RGB-D sensor has the state *unknown*. Voxels which have been sensed by the RGB-D sensor are marked *occupied* and voxels between the sensor viewpoint and *occupied* voxels are marked as *free* voxels. Their location is given by the OctoMap server. The states of *occupied* voxels

and *free* voxels is also given by the OctoMap server. The environment map is initialized with voxels in the *unknown* state. The environment map is implemented in MATLAB to be utilized not only for the next-best-view system but also for the object recognition system and a collision avoidance system.

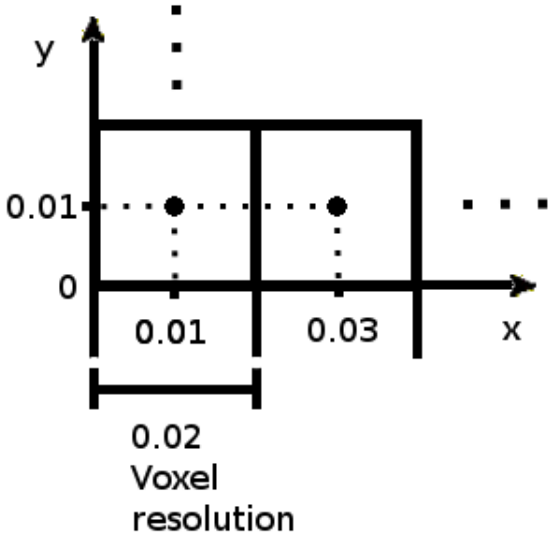


Figure 4.6: Center points for voxel in a grid map (displayed in 2D) with a voxel resolution of 0.02 meters.

Although, MATLAB has support for the '*octomap_msgs/Octomap*' ROS message and can receive those messages, MATLAB has no direct way of accessing the data in the OctoMap as they are serialized. No direct way of deserializing the data in MATLAB was found. The topic *octomap_point_cloud_centers* publishes the location of *occupied* voxels as a '*sensor_msgs/PointCloud2*' ROS message, but no information of *free* voxels can be received through this message and would require a computation of these *free* voxels. With these difficulties present, it was chosen to use the visualization markers *occupied_cells_vis_array* and *free_cells_vis_array*, which provide the location of *occupied* and *free* voxels on two topics. These topics provide no information regarding the probabilistic occupancy value of the voxels, which could be useful for next-best-view algorithms which utilize these values. As the proposed approach makes only use of the occupancy state of voxels, this was considered an acceptable loss of information.

4.5 Next-best-view component

To choose a new sensor pose, a possible new pose has to be evaluated regarding its capability to gain additional information about the environment and provide useful data

for the object recognition task. For this purpose an evaluation function is formulated. To evaluate a pose for exploration of the map, a voxel raycast is performed from the pose to raycast endpoints. These endpoints are determined by the sensor view frustum. With this design, the computed rays represent the possible measurements of a sensor in a pose. For a ray being cast from a position to a raycast endpoint, each voxel traversed by that ray is evaluated. A voxel is evaluated with the environment map and represents the state of the voxel in the environment map. If along a traversed ray an occupied voxel is encountered, the traversal is stopped and the next ray is cast and evaluated. Each traversed voxel which is evaluated as unknown is added to the total sum of unknown voxels revealed by the given position. This sum represents the function value of the evaluation function.
Let $\mathbf{M}$ be the set of all voxels v in the sensor view frustum for a pose p. Let EM be an evaluation function of the environment map which returns the state of a voxel in the environment map. Then the evaluation function for exploration f for a given pose p can be written as:

$$f(p) = \sum_{i=1}^{n=|\mathbf{M}|} [EM(v_i) = \textit{unknown}] \tag{4.3}$$

For computing the traversed voxels by a ray, a 3D grid implementation of the Bresenham algorithm was used which is based on the work "A Fast Voxel Traversal Algorithm for Ray Tracing" of J. Amantides and A. Woo [AW87].

A possible new sensor pose has to fulfill the following three constraints before it is evaluated:

Free: A possible new pose should only be in a location marked in the map as *free*. This avoids possible collisions of the sensor in its new pose. As this can not be guaranteed for unknown space, unknown marked voxels are excluded as possible new poses.

Reachable: The reachability is defined as a pose which can be reached by the robotic system without a self-collision.

Distance to object: As the sensor system should always be able to view the object for the object recognition task, the view range of the used sensor has to be considered. For the Kinect v2 sensor the minimal distance for the depth sensor is 0.5m. This minimal distance also avoids any collision of the sensor hardware with the object.

With the evaluation function given in equation (4.3), the next-best-view problem of finding a sensor pose p^*_{exp} which maximizes the information gain IG in all possible sensor placements P

$$p^*_{\text{exp}} = \arg\max_{p \in P} IG(p) \tag{4.4}$$

can be rewritten as

$$p^*_{\text{exp}} = \arg\max_{p \in P} f(p) \tag{4.5}$$

To find a solution for equation (4.5) the twiddle algorithm (see Algorithm 4.1) is used. Given an initial pose, the twiddle algorithm will choose a local optimal pose, i.e. a pose locally near the initial pose which will view the most unknown voxels in the map.
The twiddle algorithm is given an initial position in environment map coordinates $p_M = (x_M, y_M, z_M)$ and a stepsize α_i for each map dimension. Starting at the initial pose p_M, a new pose p'_M is computed by once adding and subtracting the stepsize for one dimension and evaluating the new pose p'_M. If a step taken in a dimension is an improvement, the stepsize is increased, otherwise the stepsize is decreased. The twiddle algorithm then continues to evaluate positions in the voxel map in the direction which have a higher evaluation function value. The algorithm terminates in a position where locally no other position has a higher evaluation function value.

Algorithm 4.1 Twiddle algorithm

Input: Initial pose $p_M \in Map$, initial stepsize α_i for the n Map dimensions
Output: local optimal pose $p^*_M \in Map$
repeat
  for $i = 1 : n$ **do**
    $p_M \leftarrow \arg\max_{p'_M \in \{p_M - \alpha_i, p_M, p_M + \alpha_i\}} f(p'_M)$
  end for
  Increase α_i if p_M changed; decrease α_i otherwise
until p_M converges

4.5.1 Object recognition system

The object recognition system uses a CAD model of the object to express the state of the model in the environment. As the environment can be dynamic and parts of the object can be occluded or not present, a formulation of the belief of the state of the object o in the environment is required. To increase the belief of the object state in the environment, sensor measurements z of the object in the environment are required to confirm or refute the object state. These sensor measurements are used together with an object recognition algorithm to classify parts of an object in an image and determine their location. The information provided by the CAD model and the measurements are fused to a belief of the object state in the environment. It is assumed that the object

state does not change over time in the environment. The belief over the object state is expressed with a binary bayes filter with static state [Thr02].

$$\text{bel}_t(o) = p(o|z_{1:t}) \tag{4.6}$$

Here the belief bel_t of the object state depends on a series of t sensor measurements z. Using log odds ratio to express the belief, reduces the computational effort. The log odds ratio is defined as the logarithm of the probability of an object state divided by its negate

$$l(o) = \log \frac{p(o)}{p(\neg o)} = \log \frac{p(o)}{1 - p(o)} \tag{4.7}$$

The belief can be computed from the log odds ratio with the following equation:

$$\text{bel}_t = 1 - \frac{1}{1 + \exp(l_t)} \tag{4.8}$$

Here l_t denotes the log odds ratio of the object state at time t and can be computed with

$$l_t(o) = l_{t-1}(o) + \log \frac{p(x|z_t)}{1 - p(x|z_t)} - \log \frac{p(o)}{1 - p(o)} \tag{4.9}$$

$P(o)$ is the prior probability of the state of the object and is derived from the CAD model of the object.

4.5.2 Combining exploration task and object recognition task

The combination of the exploration task and the object recognition task is achieved through a weighting of the view poses selected by each task. This requires an evaluation of a view pose for the object recognition task. In this approach, a view pose is considered "good" in case the sensor can see the location of the objects recognized in the object recognition system. The object recognition system keeps a database of recognized objects and their location throughout all views. These locations are used with the sensor model to determine if and how many objects a potential view pose can see. This is determined through ray tracing in the environment. If a ray cast from the potential view poses traverses a voxel which is located near an object location, it is assumed that the view pose can see the object location. If the ray traverses an *occupied* voxel and that *occupied* voxel is not near an object location, it is assumed that this ray can not see the object. If no ray for a view pose can see an object, then the view pose is considered a "bad" view pose. These considerations are in effect a visibility measure of the object location for a

view pose. The goal of the visibility measure $g(s)$ is to find a sensor pose p which can see as many objects present in the object database as possible. This can be formulated as

$$p^*_{\text{vis}} = \arg\max_{p \in P} g(p) \tag{4.10}$$

To solve the optimization problem in equation (4.10), the Twiddle algorithm is used. A view pose for exploration can have a low visibility metric and the same can be true for a view pose for the object recognition task for the exploration metric. A combined view pose is acquired by dynamic weights for both view poses. As more of the environment map is explored, less focus has to be given to an exploration view pose and more focus can be given to the object recognition view pose

The weight $e \in [0 \dots 1]$ for the exploration pose depends on the amount of remaining *unknown* voxel in the environment map. It can be determined with the amount of *free* voxel $\mid v_{\text{free}} \mid$ and the amount of *occupied* voxel $\mid v_{\text{occ}} \mid$ in the following formulation

$$e = 1 - \frac{\mid v_{\text{free}} \mid + \mid v_{\text{occ}} \mid}{\mid v_{\text{total}} \mid} \tag{4.11}$$

For the object recognition task a threshold P_{max} for the object state belief is used. If an object state belief is higher than the threshold P_{max}, the object state is considered true. For the weight $o \in [0 \dots 1]$ of the object recognition task, this is considered in the following formulation

$$o = 1 - \frac{\sum_{i=1}^{N} P_i(o_i|z_i)}{\sum_{i=1}^{N} P_{\text{max},i}} \tag{4.12}$$

Given the weights e and o, the exploration view pose p^*_{exp} and the object recognition view pose p^*_{vis}, a combined view pose p^*_{comb} can be computed with the following equation:

$$p^*_{\text{comb}} = \frac{e}{e+o} p^*_{\text{exp}} + \frac{o}{e+o} p^*_{\text{vis}} \tag{4.13}$$

5 Experiments

This chapter will give a brief description of the robotic arm used during the experiments, camera calibration results required for the RGB-D sensor, the hand-eye calibration which is required for the correct estimation of the sensor pose as well as experiments for the evaluation of the developed NBV system.

5.1 Robotic arm

(a) Top view of the robotic arm.

(b) Side view of the robotic arm.

Figure 5.1: Technical drawing of the Schunk LWA 4P with dimensions of links and visualization of work space. [Sch16].

The robotic arm used in the experiments is the Schunk LWA 4P[1]. It consists of three links, one of which is grounded, and three joints with two degrees of freedom (DOF) each. From it's ground point, it can reach a height of 0.8692 meters and has a horizontal reach of 0.6549 meters [Sch16]. As can be seen in Figure 5.1, the robotic arm can not make a full circular motion but only one of $340°$ degrees. This motions restrictions are considered in the selection of candidate view poses which are reachable.

[1] Schunk, *Powerball Lightweight Arm LWA 4P* [Online], Available: http://mobile.schunk-microsite.com/en/produkte/products/powerball-lightweight-arm-lwa-4p.html [Accessed: 16.07.2016]

5.2 Camera calibration

The process of camera calibration is concerned with the estimation of the intrinsic and/or the extrinsic camera parameters (refer to Section 2.2). It is required to accurately identify 3D world points sensed in the camera to their corresponding 2D image points. For a 3D world point $M = (X, Y, Z, 1)^T$ the corresponding 2D image point $m = (u, v, 1)^T$ can be computed with the equation:

$$m = IEM = \underbrace{\begin{bmatrix} f_u & s & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{intrinsic matrix I}} \underbrace{\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix}}_{\text{extrinsic matrix E}} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{5.1}$$

Lets assume that all sensed world points M are on a plane with $Z = 0$ in world coordinates. Then equation (5.1) with $Z = 0$

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_u & s & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 0 \\ 1 \end{bmatrix} \tag{5.2}$$

can be reduced to

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} f_u & s & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix}}_{\text{Homography H}} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} \tag{5.3}$$

As image point m and world point $\hat{M} = \begin{bmatrix} X & Y & 1 \end{bmatrix}^T$ both lie on planes, they are related by a homography H.

$$m = H\hat{M} \tag{5.4}$$

Let h_1, h_2 and h_3 be the column vectors of H and with equation (5.3) follows:

$$H = \begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix} = \underbrace{\begin{bmatrix} f_u & s & u_0 \\ 0 & f_v & v_0 \\ 0 & 0 & 1 \end{bmatrix}}_{\text{I}} \underbrace{\begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & t_3 \end{bmatrix}}_{\begin{bmatrix} r_1 & r_2 & t \end{bmatrix}} \tag{5.5}$$

This can be further simplified with r_1, r_2 and t being the column vectors of the extrinsic matrix:

$$\begin{bmatrix} h_1 & h_2 & h_3 \end{bmatrix} = I \begin{bmatrix} r_1 & r_2 & t \end{bmatrix} \tag{5.6}$$

As $\begin{bmatrix} r_1 & r_2 & r_3 \end{bmatrix}$ form an orthonormal basis, following conditions hold true:

$$r_1^T r_2 = 0, \ \|r_1\| = \|r_2\| = 1 \tag{5.7}$$

With these two conditions, the following two equations can be formulated. With $r_1^T r_2 = 0$, $r_1 = I^{-1} h_1$ and $r_2 = I^{-1} h_2$:

$$h_1^T I^{-T} I^{-1} h_2 = 0, \tag{5.8}$$

and with $\|r_1\| = \|r_2\| = 1$ follows:

$$h_1^T I^{-T} I^{-1} h_1 - h_2^T I^{-T} I^{-1} h_2 = 0 \tag{5.9}$$

Each homography provides two equations, which can be used to solve a linear system of equations for the intrinsic parameters. Acquiring a homography in camera calibration is done with a special calibration object, usually a chessboard pattern. For a chessboard pattern the amount of squares and their dimensions are known. The corners of the squares of the chessboard are detected and they are assumed to all lie on a plane (see Figure 5.2 and Figure 5.3). The plane is further assumed to have a $Z = 0$ world coordinate. With this setup, the chessboard plane and the image plane are related by a homography H as described above and the intrinsic camera parameters can be estimated through a linear system of equations. To acquire multiple homographies to solve the linear system of equations, the chessboard is moved into different poses in the view of the camera.

The camera calibration in this work was done with the *kinect2_calibration* ROS package provided in the IAI Kinect 2 project. Internally it uses the calibration method used in OpenCV which is based on the method of Tsai [TL89]. The process estimates the intrinsic parameters for the RGB and IR camera separately as well as the tangential and radial distortion parameters. The intrinsic skew parameter s is not estimated during the calibration. After the intrinsic camera parameters are estimated for RGB and IR camera, the extrinsic parameters between both cameras are estimated which is required for the registration of RGB images to depth measurements (point cloud). The estimated

	This work	Lachat	Microsoft SDK	Sarbolandi	IAI Kinect
$f_x[px]$	364.4	364.7	366.0	370.8	366.9
$f_y[px]$	362.7	366.1	366.0	370.2	364.8
$c_x[px]$	250.3	255.8	258.6	263.4	243.0
$c_y[px]$	202.5	203.7	206.5	202.6	207.7
k_1	0.07370	0.08708	0.09357	0.09497	0.09656
k_2	-0.29259	-0.16515	-0.27394	-0.24260	-0.28298
k_3	0.12608	-0.00321	0.09288	0	0.10524
p_1	-0.00088	-0.00345	n.a.	0.00076	0.00016
p_2	0.00286	0	n.a.	-0.00017	-0.00051

Table 5.1: IR camera parameters determined with IR camera parameters from other works. Lachat refers to the work of Lachat et al.[LMM+15]. The Microsoft SDK values were also determined in the work of Lachat [LMM+15]. Sarbolandi refers to the work of Sarbolandi et al.[SLK15] and IAI Kinect refers to the work of Wiedemeyer [Wie15]

.

	This work	IAI Kinect
$f_x[px]$	1554.4	1059.9
$f_y[px]$	1415.9	1053.9
$c_x[px]$	957.9	954.8
$c_y[px]$	537.4	523.7
k_1	-0.04159	0.05627
k_2	0.08877	-0.07419
k_3	-1.8435	0.02411
p_1	-0.005523	0.00143
p_2	-0.00852	-0.00169

Table 5.2: RGB camera parameters determined in this work and the work of Wiedemeyer [Wie15], refered to as IAI Kinect in the table.

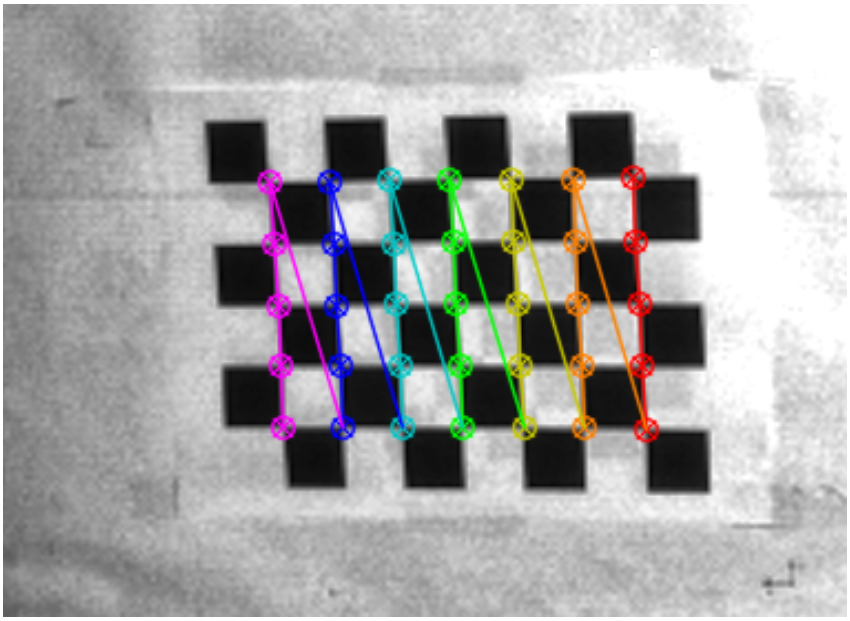


Figure 5.2: Identification of the corners of the chessboard pattern with the IR camera.

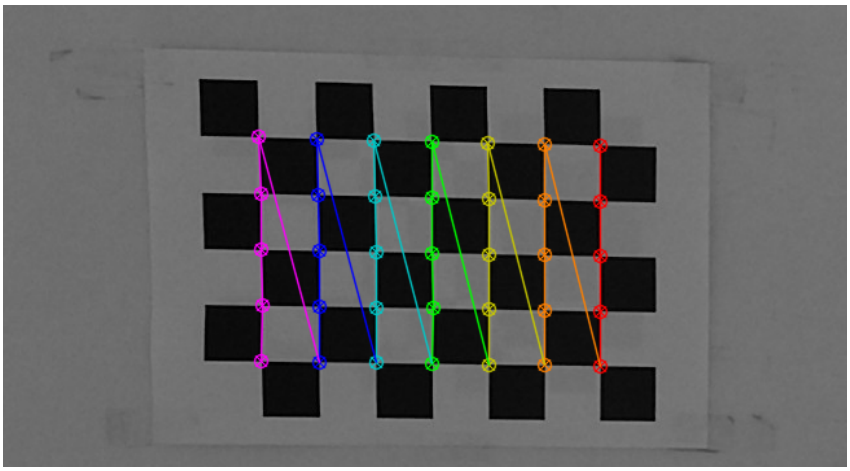


Figure 5.3: Identification of the corners of the chessboard pattern with the RGB camera.

camera parameters are similar to parameters estimated in other works. The IR camera parameters can be seen in Table 5.1 and the RGB camera parameters in Table 5.2.

The transformation of IR camera pose to the RGB camera pose was estimated as well. The transformation matrix is expressed with its rotation matrix R and the translation t separately.

$$R = \begin{bmatrix} 0.997 & -0.0167 & -0.0107 \\ 0.0166 & 0.998 & 0.0018 \\ 0.0108 & -0.0019 & 0.999 \end{bmatrix} \tag{5.10}$$

$$t = \begin{bmatrix} -0.05213 \\ -0.00034 \\ 0.00085 \end{bmatrix} \tag{5.11}$$

This places the IR sensor about 5.2 centimeters to the side of the RGB sensor. This correlates to manual measurements done at the physical sensor and the estimated parameters in the work [Wie15].

5.3 Hand-Eye calibration

The hand-eye calibration problem is the determination of the relative position and orientation of a camera, which is rigidly mounted on the robotic hand, to the hand frame. It is required for the mapping and manipulation task as location of objects and the environment sensed in the camera frame can be transformed into the robot base frame. For the octomap_server ROS node it is important as it is required to compute the transformation from the camera frame to the world frame.

Acquiring the solution of the hand-eye calibration problem, leads to solving a homogeneous transformation equation of the form:

$$AX = XB \tag{5.12}$$

Here X is the unknown transformation between sensor frame and hand frame and has to be determined. The general approach is, where the camera is mounted rigidly on the robotic arm and the robot executes movements to different positions in the robot workspace. The transformation between two hand positions i and j is the transformation A (see Figure 5.4). The transformation from robot hand to robot base can be used to compute A. Let H_i be the transformation from robot hand frame to robot base

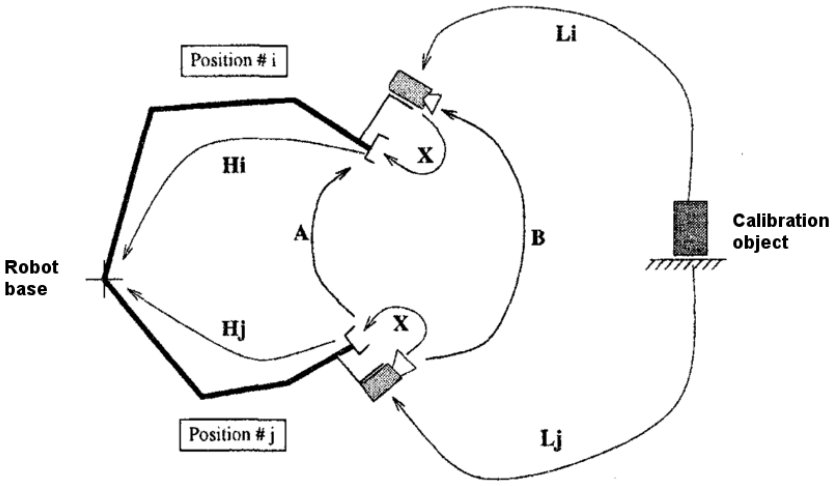


Figure 5.4: Schematic hand-eye setup with relevant transformations for two hand positions i and j. Adapted from [RDLD97].

frame for hand position i and H_j the same transformation for hand position j, then the transformation A can be computed as:

$$A = (H_i)^{-1}(H_j) \tag{5.13}$$

In every position of the hand, the camera is sensing a calibration object which does not change its location during the hand-eye calibration process. The transformation B is the transformation between two camera locations. It can be determined with the transformation L from the calibration object frame to camera frame (also known as extrinsic camera calibration in literature). For two hand positions i and j, the transformation B can be determined from the two transformations L_i and L_j:

$$B = (L_i)(L_j)^{-1} \tag{5.14}$$

With A, B and equation (5.12), the parameters of the transformation matrix X can be estimated. At least three different robot motions are required [TL89].

To solve the hand-eye calibration problem in this work, the project *aruco_hand_eye*[2] was used. The project is based on the *aruco_ros*[3] ROS package which can estimate 3D poses of fiducial markers in the camera frame [GMMM14], i.e. it provides the transformation L from camera frame to calibration object frame. Transformation H is provided by the available robotic system (see Figure 5.5 and Figure 5.6). With these transformations

[2]JHU Laboratory for Computational Sensing and Robotics, *aruco_hand_eye*, [Online] Available: https://github.com/jhu-lcsr/aruco_hand_eye [Accessed: 30.06.2016]
[3]ROS, *aruco_ros*, [Online] Available: http://wiki.ros.org/aruco_ros [Accessed: 30.06.2016]

the ROS package *visp_hand2eye_calibration*[4] is employed to estimate the parameters of transformation X with the method of R. Tsai and R. Lenz presented in their work "A new technique for fully autonomous and efficient 3D robotics hand/eye calibration" [TL89]. The method by R. Tsai and R. Lenz first computes the rotational parameters and then the translational parameters of the transformation X (called closed-form solution). The quality of the solution depends on the quality of the sensor measurements and on a sufficiently accurate intrinsic camera calibration.

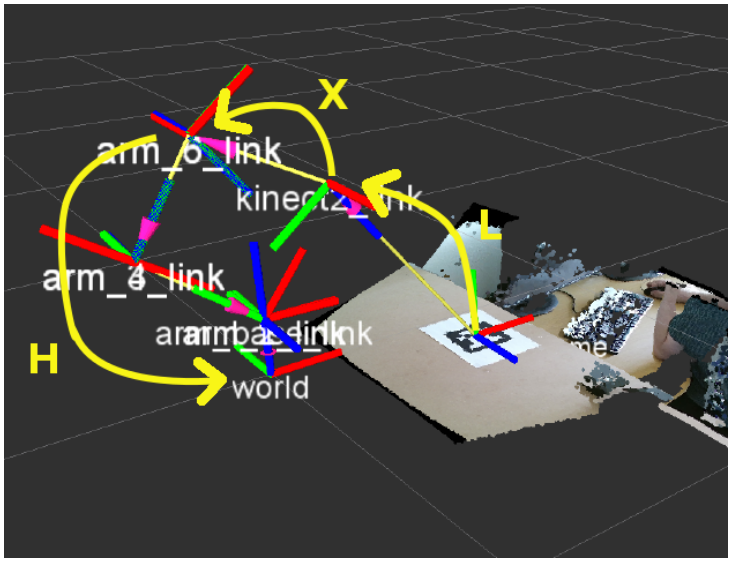


Figure 5.5: A scene showing the transformations H, L and X. The transformation L is estimated by aruco_hand_eye package. The camera is oriented towards the fiducial marker on a table in front of the robotic arm. Several frames of the robotic arm are displayed as well as the camera frame and the calibration object frame.

5.4 Experiments

During the experiments two objects were used for exploration and object recognition. One is a plate fixed onto a box with two screws (see Figure 5.7), henceforth referred to as plate object, and the other is a pressure valve with a pressure cylinder (see Figure 5.8), henceforth referred to as valve object. Both of the objects are placed on a tabletop in front of the robot. These objects were used as they resemble objects in a (dis)assembly

[4]ROS, *visp_hand2eye_calibration*, [Online] Available: http://wiki.ros.org/aruco_ros [Accessed: 30.06.2016]

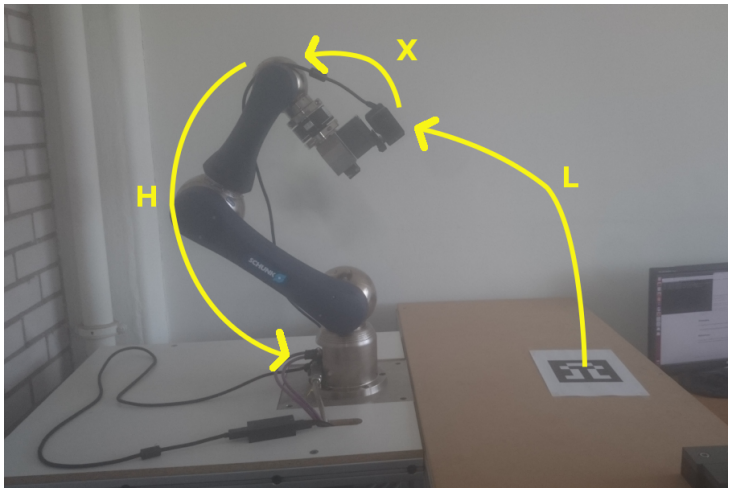
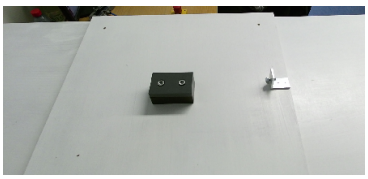
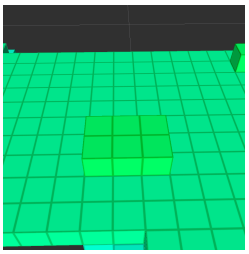


Figure 5.6: Showing the same scene as in Figure 5.5 from an "outside" view.

task of an industrial robot. The experiments consisted of evaluating the exploration approach and the combined approach for their ability to reduce the amount of unknown voxel in the environment map and the time required to determine a new viewpose. The experiments were evaluated on a Ubuntu 14.04 64-bit system with a Intel i7-4790k 4.00 GHz processor and 16 GB memory. The distribution version ROS Indigo Igloo was used.



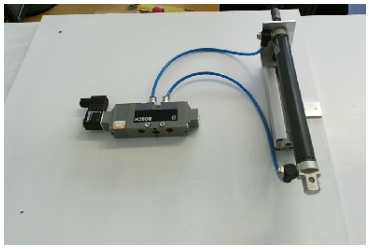
(a) Plate object seen from RGB sensor.



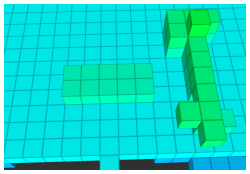
(b) Plate object modeled as *occupied* voxel in the OctoMap.

Figure 5.7: Plate object on a tabletop in front of the robot.

During the experiments the robot was moved into random starting positions which were oriented towards the tabletop scene. The orientation toward the tabletop scene is a requirement of the object recognition system to correctly execute. An unknown box-shaped area which contains the tabletop scene was defined. It contains 3500 unknown voxel at initialization.
The impact of different step sizes in the Twiddle algorithm were evaluated. The step size has impact on the selection of a view pose and therefore an impact on the amount of unknown voxel revealed by that view pose. Figure 5.9 shows the average amount of



(a) Valve object as RGB image.



(b) Valve object as *occupied* voxel in OctoMap.

Figure 5.8: Pressure valve with cylinder on tabletop in front of the robot.

unknown voxel remaining in the environment map after four view poses for different step sizes with the plate object. The average amount of unknown voxel for the valve object can be seen in Figure 5.10. The biggest reduction of unknown voxel in the environment map is achieved in the first view pose. Further view poses reduce the amount of unknown voxel to a lesser and lesser degree. The amount of unknown voxel could not be totally reduced with any approach. This is due to the orientation of the sensor towards the tabletop scene and the reachable points of the robot. Most of these unknown voxel are located above the tabletop scene and on the far side of the box-shaped unknown region. The Twiddle algorithm with step size 3 and 5 performed equally well. A step size of 3 provided on average the least amount of remaining unknown voxel. With a step size of 1 the Twiddle algorithm was in some cases incapable of finding a local optimum. This occurs when the immediate neighbors for an initial starting pose do not improve upon the objective function. If no improve on the objective function is found, the step size is reduced to 0 and the Twiddle algorithm terminates and returns the initial random pose.

This behavior is also responsible for the Twiddle algorithm to terminate the fastest with a step size of 1. This can be seen in Figure 5.11, which depicts the average amount of time the Twiddle algorithm required to find a local optimum with different step sizes. In general, the Twiddle algorithm has no consistent execution time which can be seen in the relative high standard deviation among all step sizes. A global search for the optimum took on average 35.12 seconds. This is approximately 2.6 times slower than the maximal Twiddle execution time of 13.15 seconds, which was measured during the experiments.

The time needed to update the environment map depends on the amount of *occupied* and *free* voxel in the OctoMap. As the amount of these increases with each view pose, the time required to update the environment also increases. This can be seen in Figure 5.12.

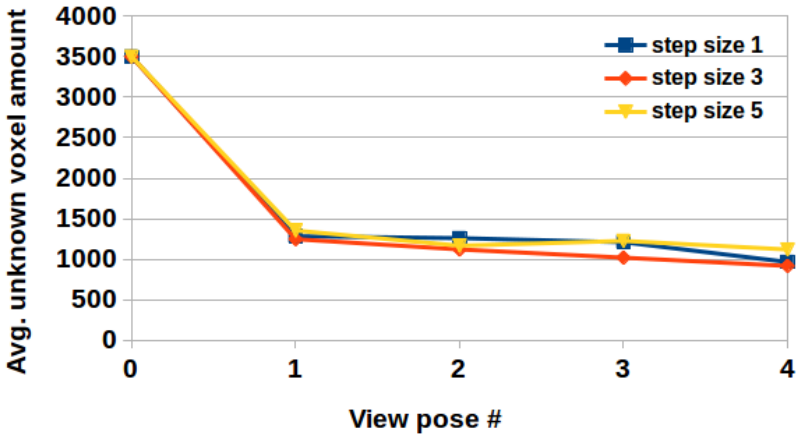


Figure 5.9: Average amount of unknown voxel remaining in the environment map after viewing the tabletop scene with the plate object from view poses determined by the exploration method.

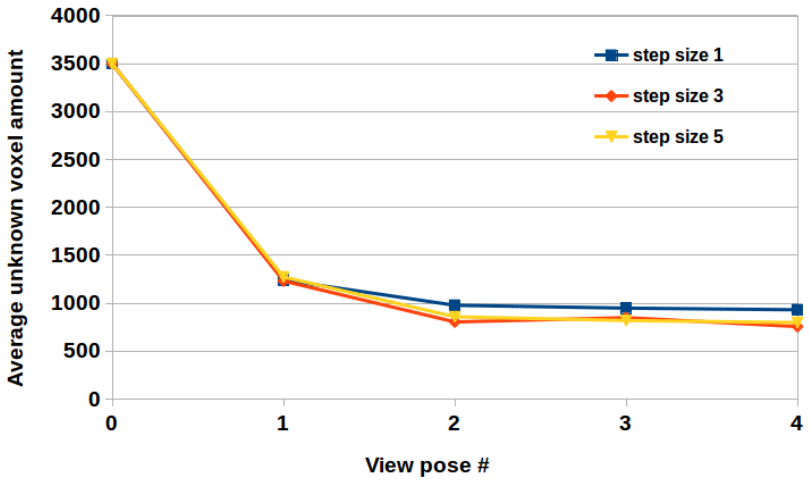


Figure 5.10: Average amount of unknown voxel remaining in the enivronment map for the valve object

The method of combining the environment exploration with the visibility measure for the object recognition was validated in two scenarios. In the first the plate object was placed in front of the robot and in the second an obstacle was placed between the plate object and the robot which partly occluded the plate object from one side (see Figure 5.13). The valve object could not be used during these experiments as the object recognition system was not trained for the valve object at the time of the experiments. The method was compared against the sole exploration method for its ability to explore

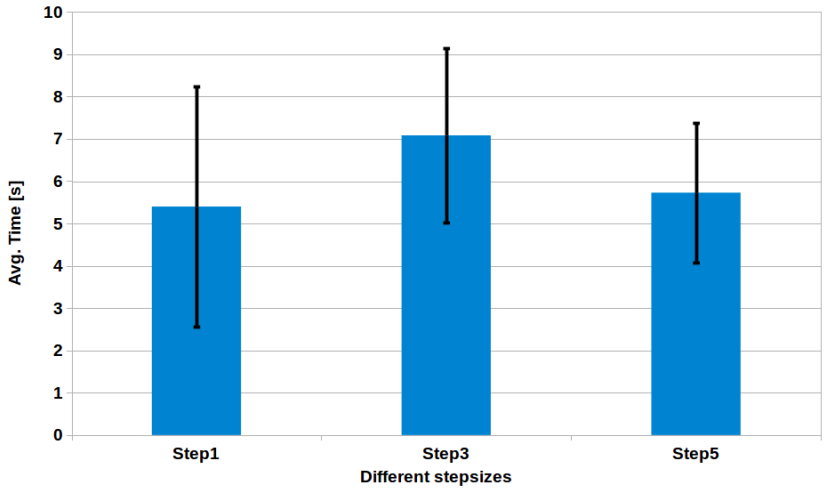


Figure 5.11: Average execution time with standard deviation of the twiddle algorithm with different step sizes.

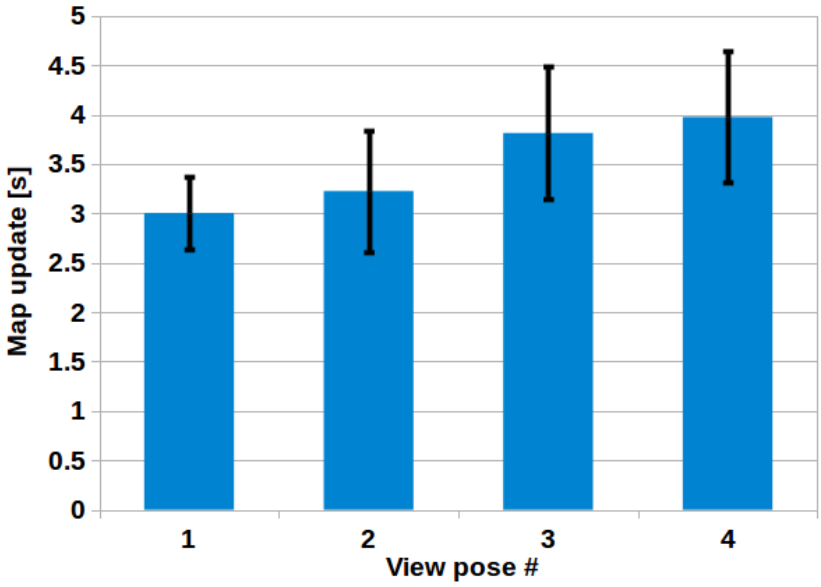
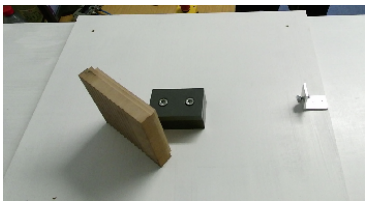


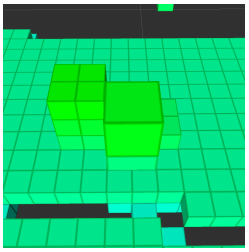
Figure 5.12: Average amount of time required to update the map increases with each new view pose.

the environment. The average amount of unknown voxel remaining in the environment can be seen in Figure 5.14.

The combined approach only was only slightly worse in exploring the environment as compared to the sole exploration approach. This is expected as the combined approach does not solely focus on exploration.



(a) Plate object with obstacle as RGB image.



(b) Plate object with obstacle displayed as *occupied* voxel in OctoMap.

Figure 5.13: Plate object on tabletop with an occlusion object placed between the plate object and robot which partly obstructs the visibility of the plate object.

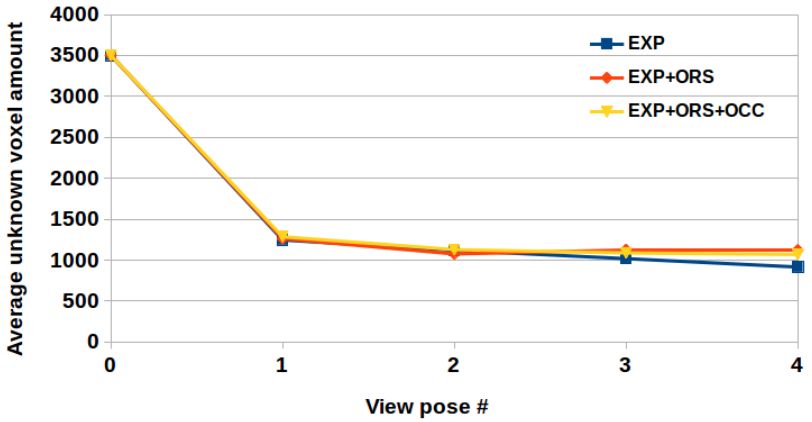


Figure 5.14: Average amount of unknown voxel remaining in the environment map after four view poses. Depicted are the exploration method (EXP), exploration with visibility for object recognition (EXP+ORS) and the exploration with visibility method with on occlusion object (EXP+ORS+OCC).

During the experiments with the visibility measure the object recognition system was running to compute the weights for the view pose selection. Its task was the recognition of screws of the plate object. A correctly recognized scene can be seen in Figure 5.15. The bracket to the right of the plate object in Figure 5.15 got misclassified as a screw in some cases, this can be seen in Figure 5.16.

During the experiments with the occlusion object, a view pose can lose vision of parts of the object. During these view poses no update can be made in the object belief. Figure 5.17 depicts the object belief of the screws recognized in the scene with an occlusion object. During the second view pose a new object was recognized, which was not recognized during subsequent view poses. In the third view pose the second screw

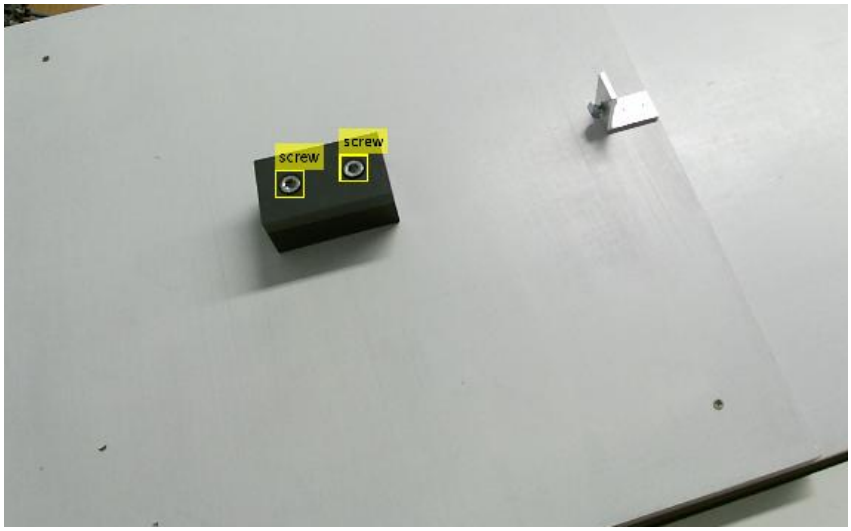


Figure 5.15: The object recognition system correctly identifies both screws on the plate object.

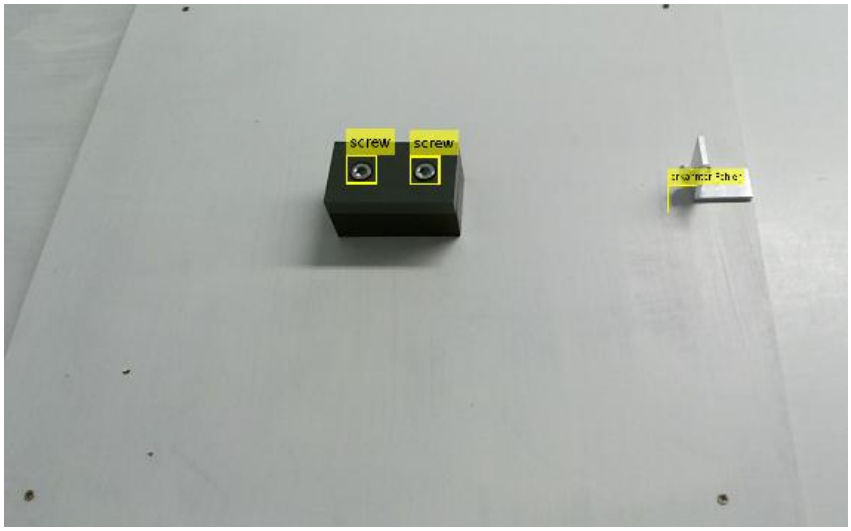


Figure 5.16: A misclassification of the bracket to the right of the plate object as a screw.

could not be correctly recognized and therefore had no update of the object belief. In subsequent views the second screw was again viewable and was recognized.

Finding a local optimal pose with the Twiddle algorithm for the visibility measure of the object recognition took on average 2.00 seconds. This increases the average time to find a local optimum view pose to 9.15 seconds. The average times for finding a local optimum are summarized in Table 5.3.

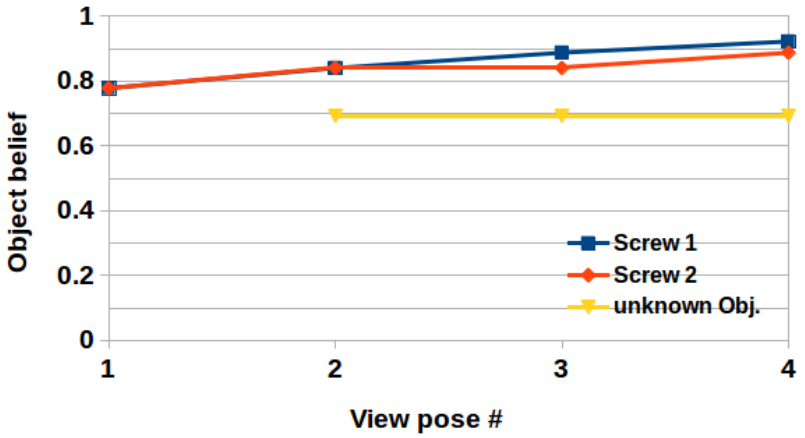


Figure 5.17: Object belief of the screws on the plate object over four view poses during one experiment with occlusion object.

Method	Avg. time [s]
Global search	35.12
EXP	7.15
EXP+ORS	9.15

Table 5.3: Average time required to compute a new view pose across view pose candidates for a global search, sole exploration method (EXP) and exploration with visibility measure (EXP+ORS).

5.5 Discussion of experiments

During the experiments the Twiddle algorithm performed as expected in finding local optimal view poses in varying time. It was expected that a step size of 1 in the Twiddle algorithm would terminate fast and not improve on the initial start pose. This was surprisingly often not the case and local optimal view poses were found relative far from the initial starting pose. To cope with the limitation of a local optimal algorithm to get stuck in local optima, the Twiddle algorithm was initialized with five random starting view poses. Among the five computed local optimal view poses, the view pose which provided the most expected reduction in unknown voxel was chosen as the next view pose for the robot. Although these runs were executed consecutively, they can be easily

parallelized as the Twiddle algorithm does not change the content of the environment map or the set of candidate view poses.
To avoid collisions with the robot itself, the system required constant supervision during motion executions from one view pose to the next view pose. Collisions could as there was no active collision avoidance active during the execution of the movements. Movements which could lead to collision occurred in some cases. These occurred as the Kinect v2 sensor was mounted on a mounting block which was attached to the end effector of the robot. As the Kinect was always oriented towards the tabletop scene, the mounting block and the robot would come into collision the moving robotic system for some movements without human intervention. For a fully autonomous system an active collision avoidance which includes the full robot geometry has to be included during the robot motions.
The orientation towards the tabletop scene of the RGB-D sensor enabled the object recognition system to receive relevant image data with each view pose. With this approach only four view poses are required to increase the object belief above a threshold of 0.9. The disadvantage of the fixed orientation can be seen in the exploration, as not the whole scene could be explored. Although with a wider range of reachable poses of the robot, it can be possible to observe the whole scene. A variable orientation of the sensor pose can allow the sensor to view more of the environment and observe the tabletop scene from different orientations. View poses with a variable orientation can also view scenes in the environment which do not contain the tabletop scene, which would not improve the object recognition task. So this is a trade-off; with a fixed orientation of the sensor towards the tabletop scene, each view has the potential to provide relevant information for the object recognition task and with a variable orientation of the sensor each view pose can explore more of the environment with potential useless information for the object recognition task.
During the experiments a voxel resolution of $0.05m$ was used. The voxel size has a direct impact on the amount of voxel available in OctoMap. With a smaller voxel resolution, the OctoMap represents the environment to a finer degree which would also increase the time required to update the environment map and the evaluation of a view pose with a ray traversal algorithm. With a voxel resolution of $0.01m$ a map update took around 2 minutes on average. This was deemed an unacceptable time as a map update is required for each new view pose. The disadvantage of a larger voxel resolution is a less accurate representation of the state of the environment. An object with dimensions smaller than the voxel resolution is represented as an *occupied* voxel with the dimensions of the voxel resolution. This can especially be of concern in cases where unknown space and occupied space are close to each other. As the occupied space in the environment will be mapped to *occupied* voxel, all unknown space contained in the *occupied* voxel are not known in the environment as they get "swallowed" by the *occupied* voxel. The information of the unknown space gets lost and can not be used in the exploration method. This is another trade-off; a smaller voxel resolution for a finer representation

of the environment and an increased computation time or a loss of information in the environment map for faster computation times in the exploration. The choice for this trade-off depends on the application. The approach developed in this work can be adapted to lower or higher voxel resolution without losing the ability to explore the environment but one should be aware of the trade-off.
During the experiments, a typical error of Time-of-Flight cameras could be observed: so called “flying pixel”. They occur due to depth inhomogeneities [KBKL10]. These cause wrong distance values for measured points. This leads to a false representation of the environment in the environment map and causes an overestimation of the amount of occupied voxel in the environment map (see Figure 5.18).

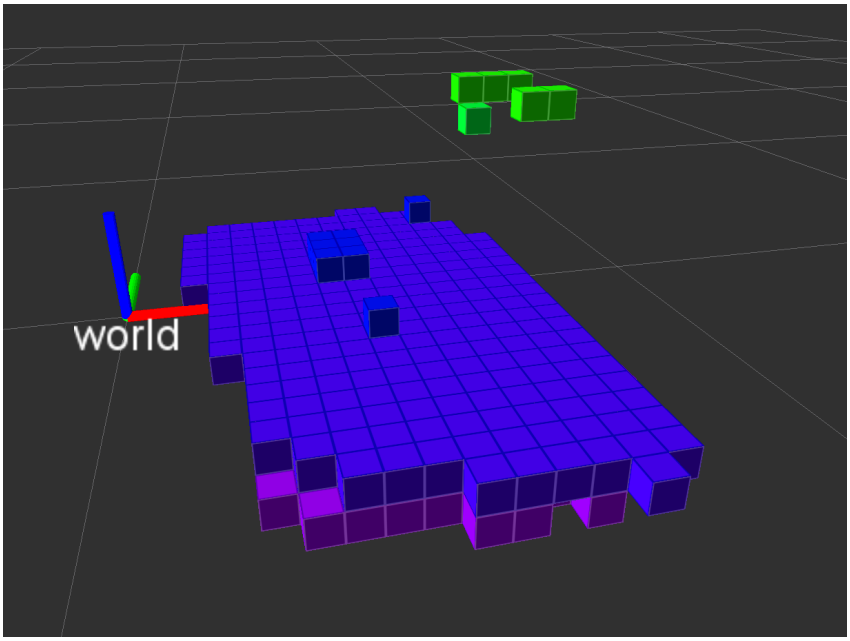


Figure 5.18: *Flying pixels* in the sensor measurements lead to a false representation (green voxel) of occupied space in the environment map. Scene shows a side view of the OctoMap of the tabletop scene.

6 Discussion and conclusion

This work presented a method for next-best-view computation combining exploration of the environment with an object recognition task. The Kinect v2 sensor together with OctoMap was used to create an environment map. This map was used for the evaluation of a set of candidate view poses. To efficiently determine the next-best-view, a local optimization algorithm was used to determine view poses for the exploration and object recognition task. These were combined with dynamic weights to acquire a view pose which can explore the environment and provide image data for the object recognition task. The experiments showed a decrease of unknown environment could be achieved while also providing image data required for the object recognition task. The usage of a local optimization algorithm computed view poses much faster than a global search for the optimal pose.

Although the approach in this work fulfills the requirements posed for this work, several possible adjustments can be made to improve the approach. The set of view pose candidates can be expanded by sampling the complete workspace of the robot. With this bigger set, the full motion capabilities of the robot can be used to acquire view poses which were not present during this work. This can have the potential to improve upon the reduction of the unknown environment as well as provide view poses which can view the object for the object recognition task. As these view poses can lead to self-collision, a collision test for each pose would be required which includes the geometry of the robot, similar to the work of S.Kriegel [Kri15].

The evaluation of a view pose in this work is a rather simple metric and can be further enhanced with more sophisticated evaluation algorithms as presented in [BST15]. Although these algorithms would have a higher computation time compared to the approach in this work, they would be able to estimate the expected amount of reduced unknown voxel more accurately. As these algorithms depend on probabilistic occupancy maps, the environment map developed in this work would need to be adjusted to include occupancy probabilities. This could be achieved by including the OctoMap directly into our approach. During this work, a ROS node was developed which was able acquire this information from the OctoMap ROS node and communicating this information to the MatLab ROS node. This approach was abandoned, as the location of some voxels was incorrect and the cause for these false locations could not be identified.

The execution time of the system can be further improved by using parallelization for the map update and the ray traversal algorithm, similar to the work [MAC16]. The

registration process for the RGB and depth image data, puts a heavy load on the CPU. This can be reduced by using the graphic card for the depth registration. The Kinect v2 ROS node is able to utilize CUDA for this. Although it was tried to enable this feature, the author was not able to get this feature to work.

Bibliography

[AW87] J. Amanatides, A. Woo. “A Fast Voxel Traversal Algorithm for Ray Tracing.” In: *Eurographics* 87.3 (1987), pp. 3–10 (cit. on p. 46).

[BDL+08] M. Baumann, D. Dupuis, S. Leonard, E. Croft, J. Little. “Occlusion-free path planning with a probabilistic roadmap.” In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2008, pp. 2151–2156 (cit. on p. 25).

[BR14] M. Bloesch, D. Rodriguez. “Kinect v2 for Mobile Robot Navigation : Evaluation and Modeling.” In: *International Conference on Advanced Robotics (ICAR)* (2014) (cit. on p. 39).

[BST15] F. Bissmarck, M. Svensson, G. Tolt. “Efficient algorithms for Next Best View evaluation.” In: *IEEE International Conference on Intelligent Robots and Systems* (2015), pp. 5876–5883 (cit. on p. 69).

[BWDA00] J. E. Banta, L. M. Wong, C. Dumont, M. A. Abidi. “A next-best-view system for autonomous 3-D object reconstruction.” In: *IEEE Transactions on Systems, Man, and Cybernetics Part A:Systems and Humans.* 30.5 (2000), pp. 589–598 (cit. on p. 35).

[Con85] C. Connolly. “The Determination of next best views.” In: *IEEE International Conference on Robotics and Automation* 2 (1985), pp. 432–435 (cit. on pp. 15, 35).

[DCB04] S. Dutta Roy, S. Chaudhury, S. Banerjee. “Active recognition through next view planning: a survey.” In: *Pattern Recognition* 37.3 (2004), pp. 429–446 (cit. on p. 15).

[DDN03] F. Deinzer, J. Denzler, H. Niemann. “Viewpoint selection - Planning optimal sequences of views for object recognition.” In: *Computer Analysis of Images and Patterns* 2756 (2003), pp. 65–73 (cit. on p. 33).

[EBD92] D. Eggert, K. Bowyer, C. Dyer. “Aspect graphs: State-of-the-art and applications in digital photogrammetry.” In: *Proceedings of the 17th Congress of the International Society for Photogrammetry and Remote Sensing* (1992), pp. 633–645 (cit. on p. 23).

[Esp12] A. Esposito. “Beyond range: Innovating fluorescence microscopy.” In: *Remote Sensing* 4.1 (2012), pp. 111–119 (cit. on p. 40).

[GMMM14] S. Garrido-Jurado, R. Munoz-Salinas, F. J. Madrid-Cuevas, M. J. Marin-Jimenez. “Automatic generation and detection of highly reliable fiducial markers under occlusion.” In: *Pattern Recognition* 47.6 (2014), pp. 2280–2292 (cit. on p. 57).

[Gre02] M. Greenspan. “Geometric probing of dense range data.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 24.4 (2002), pp. 495–508 (cit. on p. 28).

[HK89] S. A. Hutchinson, A. C. Kak. “Planning Sensing Strategies in a Robot Work Cell with Multi-Sensor Capabilities.” In: *IEEE Transactions on Robotics and Automation* 5.6 (1989), pp. 765–783 (cit. on p. 23).

[Hug10] W. Hugemann. *Correcting lens distortions in digital photographs*. 2010 (cit. on p. 21).

[HWB+13] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, W. Burgard. “OctoMap: an efficient probabilistic 3D mapping framework based on octrees.” In: *Autonomous Robots* 34.3 (2013), pp. 189–206 (cit. on pp. 28, 41–43).

[KBKL10] A. Kolb, E. Barth, R. Koch, R. Larsen. “Time-of-flight cameras in computer graphics.” In: *Computer Graphics Forum* 29.1 (2010), pp. 141–159 (cit. on p. 67).

[Kri15] S. Kriegel. “Autonomous 3D Modeling of Unknown Objects for Active Scene Exploration.” PhD thesis. 2015 (cit. on p. 69).

[LJS97] A. Leonardis, A. Jaklic, F. Solina. “Superquadrics for segmenting and modeling range data.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 19.11 (1997), pp. 1289–1295 (cit. on p. 27).

[LMM+15] E. Lachat, H. Macher, M.-A. Mittet, T. Landes, P. Grussenmeyer. “FIRST EXPERIENCES WITH KINECT V2 SENSOR FOR CLOSE RANGE 3D MODELLING.” In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XL-5/W4.5 (2015), pp. 93–100 (cit. on pp. 39, 54).

[MAC16] R. Monica, J. Aleotti, S. Caselli. “A KinFu based approach for robot spatial attention and view planning.” In: *Robotics and Autonomous Systems* 75 (2016), pp. 627–640 (cit. on pp. 34, 69).

[MBND10] C. Munkelt, A. Breitbarth, G. Notni, J. Denzler. “Multi-View Planning for Simultaneous Coverage and Accuracy Optimisation.” In: *Procedings of the British Machine Vision Conference 2010* (2010), pp. 118.1–118.11 (cit. on p. 22).

[Pit95] R. Pito. “Solution to the next best view problem for automated CAD model acquisiton of free-form objects using range cameras.” In: *Proceedings of SPIE* (1995), pp. 78–89 (cit. on p. 30).

[PS97] D. Papadopoulos-Orfanos, F. Schmitt. “Automatic 3-D digitization using a laser rangefinder with a small field of view.” In: *Proceedings. International Conference on Recent Advances in 3-D Digital Imaging and Modeling (Cat. No.97TB100134)*. 1997, pp. 60–67 (cit. on p. 29).

[RDLD97] S. Remy, M. Dhome, J.-M. Lavest, N. Daucher. “Hand-eye calibration.” In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*. Vol. 2. 1997, 1057–1065 vol.2 (cit. on p. 57).

[Sch16] Schunk. *Technical data sheet LWA 4P*. 2016. URL: http://mobile.schunk-microsite.com/fileadmin/user_upload/broshures/SCHUNK_Technical_data_LWA4P.pdf (cit. on p. 51).

[SLK15] H. Sarbolandi, D. Lefloch, A. Kolb. “Kinect Range Sensing: Structured-Light versus Time-of-Flight Kinect.” In: (2015). arXiv: 1505.05459 (cit. on p. 54).

[SO14] J. Sell, P. O’Connor. “The Xbox One System on a Chip and Kinect Sensor.” In: *IEEE Micro* 34.2 (2014), pp. 44–53 (cit. on p. 39).

[SRR00] W. Scott, G. Roth, J. Rivest. “Performance-Oriented View Planning for Model Acquisition.” In: *The International Symposium on Robotics (ISR 2000)* (2000), p. 8 (cit. on p. 22).

[SRR03] W. R. Scott, G. Roth, J.-F. Rivest. “View planning for automated three-dimensional object reconstruction and inspection.” In: *ACM Computing Surveys* 35.1 (2003), pp. 64–96 (cit. on pp. 21, 22, 24–26, 29).

[Stu14] P. Sturm. “Pinhole Camera Model.” In: *Computer Vision* (2014), pp. 610–613 (cit. on p. 18).

[Sup07] M. Suppa. “Autonomous Robot Work Cell Exploration using Multisensory Eye-in-Hand Systems.” PhD thesis. 2007 (cit. on p. 35).

[TG95] G. Tarbox, S. Gottschlich. *Planning for Complete Sensor Coverage in Inspection*. 1995 (cit. on p. 22).

[Thr02] S. Thrun. “Probabilistic robotics.” In: *Communications of the ACM* (2002) (cit. on p. 48).

[TL89] R. Tsai, R. Lenz. “A new technique for fully autonomous and efficient 3D robotics hand/eye calibration.” In: *IEEE Transactions on Robotics and Automation* 5.3 (1989), pp. 345–358 (cit. on pp. 53, 57, 58).

[TMD10] M. Trummer, C. Munkelt, J. Denzler. “Online next-best-view planning for accuracy optimization using an extended E-criterion.” In: *Proceedings - International Conference on Pattern Recognition* (2010), pp. 1642–1645 (cit. on p. 33).

[Wen13] S. Wenhardt. *Ansichtenauswahl für die 3-D-Rekonstruktion statischer Szenen*. Vol. 37. Logos Verlag Berlin GmbH, 2013 (cit. on p. 34).

[Wie15] T. Wiedemeyer. *IAI Kinect2*. https://github.com/code-iai/iai_kinect2. Accessed July 30, 2016. University Bremen: Institute for Artificial Intelligence, 2014 – 2015 (cit. on pp. 38, 40, 54, 56).

[Yua95] X. Yuan. “A Mechanism of Automatic 3D Object Modeling.” In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 17.3 (1995), pp. 307–311 (cit. on p. 29).

[Zen14] S. Zennaro. “Evaluation of Microsoft Kinect 360 and Microsoft Kinect One for robotics and computer vision applications.” MA thesis. 2014 (cit. on p. 39).

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature