

Institutionen för systemteknik

Department of Electrical Engineering

Examensarbete

Indoor 3D Mapping using Kinect

Examensarbete utfört i Datorseende
vid Tekniska högskolan vid Linköpings universitet
av

Morgan Bengtsson

LiTH-ISY-EX--14/4753--SE

Linköping 2014



Linköpings universitet
TEKNISKA HÖGSKOLAN

Indoor 3D Mapping using Kinect

Examensarbete utfört i Datorseende
vid Tekniska högskolan vid Linköpings universitet
av


Morgan Bengtsson

LiTH-ISY-EX--14/4753--SE

Handledare: **Hannes Ovrén**
ISY, Linköpings universitet
Folke Isaksson
SAAB, Vricon Systems

Examinator: **Per-Erik Forssén**
ISY, Linköpings universitet

Linköping, 7 april 2014

	Avdelning, Institution Division, Department	Datum Date
	Datorseende Department of Electrical Engineering SE-581 83 Linköping	2014-04-07

Språk Language <input type="checkbox"/> Svenska/Swedish <input checked="" type="checkbox"/> Engelska/English <input type="checkbox"/> _____	Rapporttyp Report category <input type="checkbox"/> Licentiatavhandling <input checked="" type="checkbox"/> Examensarbete <input type="checkbox"/> C-uppsats <input type="checkbox"/> D-uppsats <input type="checkbox"/> Övrig rapport <input type="checkbox"/> _____	ISBN _____ ISRN LiTH-ISY-EX--14/4753--SE
--	---	---

URL för elektronisk version http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-106145	Serietitel och serienummer Title of series, numbering	ISSN _____
---	---	----------------------

Titel Title	Kartering av inomhusmiljöer med Kinect Indoor 3D Mapping using Kinect
Författare Author	Morgan Bengtsson

Sammanfattning Abstract
<p>In recent years several depth cameras have emerged on the consumer market, creating many interesting possibilities for both professional and recreational usage. One example of such a camera is the Microsoft Kinect sensor originally used with the Microsoft Xbox 360 game console. In this master thesis a system is presented that utilizes this device in order to create an as accurate as possible 3D reconstruction of an indoor environment. The major novelty of the presented system is the data structure based on signed distance fields and voxel octrees used to represent the observed environment.</p>

Nyckelord Keywords	Kinect, mapping, sparse voxel octree, signed distance function, pose estimation
------------------------------	---

Sammanfattning

Under de senaste åren har flera olika avståndskameror lanserats på konsumentmarkanden. Detta har skapat många intressanta applikationer både i professionella system samt för underhållningssyfte. Ett exempel på en sådan kamera är Microsoft Kinect som utvecklades för Microsofts spelkonsol Xbox 360. I detta examensarbete presenteras ett system som använder Kinect för att skapa en så exakt rekonstruktion i 3D av en inomhusmiljö som möjligt. Den främsta innovationen i systemet är en datastruktur baserad på signed distance fields (SDF) och octrees, vilket används för att representera den rekonstruerade miljön.

Abstract

In recent years several depth cameras have emerged on the consumer market, creating many interesting possibilities for both professional and recreational usage. One example of such a camera is the Microsoft Kinect sensor originally used with the Microsoft Xbox 360 game console. In this master thesis a system is presented that utilizes this device in order to create an as accurate as possible 3D reconstruction of an indoor environment. The major novelty of the presented system is the data structure based on signed distance fields and voxel octrees used to represent the observed environment.

Contents

Notation	ix
1 Introduction	1
1.1 Goal	1
1.2 Limitations	1
1.3 System Overview	2
2 Background	3
2.1 The Kinect Sensor	3
2.1.1 Hardware	3
2.1.2 Calibration	4
2.2 Homogeneous Coordinates	5
2.3 Camera Model	5
2.3.1 Extrinsic Camera Model	5
2.3.2 Intrinsic Camera Model	6
2.4 Representation of Orientation	7
2.4.1 Euler Angles	7
2.4.2 Rotation Matrix	8
2.4.3 Unit Quaternions	8
2.5 3D Model Representation	9
2.5.1 Point Cloud	9
2.5.2 Polygon Mesh	9
2.5.3 Voxel Volume	10
2.5.4 Sparse Voxel Octree (SVO)	10
2.5.5 Signed Distance Field (SDF)	12
2.6 Pose Estimation	12
2.6.1 Dense Image Alignment	14
2.6.2 Sparse Image Alignment	14
2.6.3 Iterative Closest Point	16
3 Theory	19
3.1 Overview	19
3.2 Image Collection	20

3.3	Pre-processing	20
3.3.1	RGB	20
3.3.2	Depth	21
3.4	Panorama Construction	21
3.4.1	Feature Descriptors	21
3.4.2	Feature Matching	21
3.4.3	Rotation Estimation	22
3.4.4	Loop Closing	24
3.5	Model Generation	25
3.5.1	Sparse SDF Octree (SSDO)	25
3.6	Panorama-Model alignment	29
3.6.1	Pose Estimation through Feature Matching	30
3.6.2	Mapping between depth and color images	30
3.6.3	Pose Refinement through the Iterative Closest Point algorithm	31
3.7	Visualization	34
3.7.1	Ray-casting	34
3.7.2	Mesh Generation	38
4	Result	39
4.1	Model Accuracy	39
4.2	Memory Usage	42
4.3	Performance	43
4.4	Visual Result	43
5	Analysis and Conclusions	45
5.1	Model Quality	45
5.2	Panorama construction	45
5.3	Panorama Alignment	46
5.4	Sparse SDF Octrees	46
5.5	Future Work	47
5.5.1	Allow translation within a panorama	47
5.5.2	Combine RGB and depth data for pose estimation	47
5.5.3	Improvement of depth data using disparity maps of RGB images	47
5.5.4	GPU implementation of a sparse SDF octree	47
A	Screenshots	51
	Bibliography	57

Notation

NOTATIONS

Notation	Meaning
$\dot{\mathbf{x}}$	Homogenous representation of the vector \mathbf{x} .
\mathbb{R}^n	The n-dimensional space.
\mathbf{fov}	Horizontal and vertical field-of-view in radians.
\mathbf{l}_d	Lens offset.
\mathbf{c}_d	Sensor displacement.
K	Intrinsic camera matrix.
R	Rotation matrix.
T	Extrinsic camera matrix.
$SDF(\mathbf{x})$	Signed distance function.
$\mathbf{f}_{k,A}$	Feature point in frame A.
$\mathbf{d}_{k,A}$	FREAK descriptor of $\mathbf{f}_{k,A}$.
$m_{A,B}(k)$	Feature match between frames A and B.
$H(\mathbf{d}_1, \mathbf{d}_2)$	Bitwise Manhattan (Hamming) distance.
$C_{A,B}$	Inlier set of point correspondences.
$\mathbf{p}_{i,A}$	Vertex observed by frame A.
$\mathbf{n}(\mathbf{p}_{i,A})$	Normal approximation at $\mathbf{p}_{i,A}$.
$d_A(\mathbf{x})$	Depth in mm at image coordinate \mathbf{x} in frame A.
$W_A(\mathbf{v})$	Sparse SDF octree sample weight.

ABBREVIATIONS

Abbreviation	Meaning
CPU	Central Processing Unit
FOV	Field Of View
GPU	Graphics Processing Unit
ICP	Iterative Closest Point
NIR	Near Infra-Red
OPP	Orthogonal Procrustes Problem
RANSAC	Random Sample Consensus
RGB	Red-green-blue
SDF	Signed Distance Function
SIMD	Single Instruction, Multiple Data
SSDO	Sparse Signed Distance Octree
SVD	Singular Value Decomposition
SVO	Sparse Voxel Octree

1

Introduction

The ability to acquire accurate representations of real world environments from camera images is and has for some time been highly desired for multiple reasons. It makes it possible to effectively map areas in three dimensions from aerial imagery, to analyze the structure of objects, or to create realistic virtual worlds from real world data. As the quality of the available methods constantly are improving, new applications are frequently discovered.

1.1 Goal

In this thesis a method is presented that aims to create an as accurate as possible 3D reconstruction of an indoor environment using the Microsoft Kinect sensor. The reconstruction consist of a Signed Distance Function that is sampled at discrete positions by an octree structure henceforth referred to as a Sparse SDF Octree. The system is created with three major goals in mind. The first is to create models of as high accuracy as possible, where measured distances in the models are as close as possible to corresponding measurements in the real world. The second goal is to create a model that closely resembles the real world environment visually. The third goal is to create a system with good performance both related to execution time and memory consumption.

1.2 Limitations

In order to avoid problems introduced by rolling shutter or motion blur, the Kinect sensor is mounted on a tripod in a stable way. Each image is taken while the camera is stationary. In order to simplify the pose estimation problem, over-

lapping images are captured from each position of the tripod in 360° panoramas where the first and the last images overlap. The mapped environment is assumed to be static. The reconstruction is performed offline on a standard PC.

1.3 System Overview

The presented system is built with the task in mind of constructing models of indoor environments using a Kinect sensor which have as high quality as possible. Except from the image capture stage, the process is automatic. Figure 1.1 outlines the process of the system from image capture to the finished model.

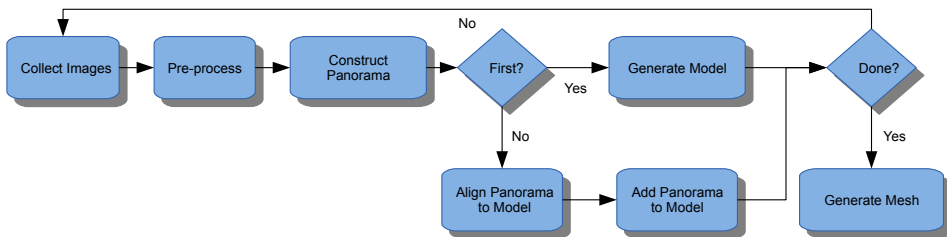


Figure 1.1: Overall flowchart of the system.

Initially the user manually collects pairs of RGB and raw depth data images from a few locations. These images are preprocessed to reduce noise and distortion. The resulting images are then aligned in 360 degree panoramas. This is done by extracting image features using the FAST corner detector ([Rosten and Drummond, 2005] and [Rosten and Drummond, 2006]), which are matched using the FREAK feature descriptor extractor [Alahi et al., 2012] and the RANSAC algorithm [Fischler and Bolles, 1981].

The first panorama is used to create a model representing the surrounding environment. The model consists of a voxel octree where samples of the signed distance field are stored. The other panoramas are then aligned one by one and added to this model using feature descriptor matching together with the ICP algorithm. The complete model is visualized in two ways, through ray-casting of the model and through rasterization of a polygon mesh generated from the model.

2

Background

2.1 The Kinect Sensor

The Kinect sensor was released by Microsoft in 2010 for the Xbox 360 gaming console. The idea was that the user should be able to interact in a more natural way using gestures or voice commands instead of handling a physical controller.

2.1.1 Hardware

The Kinect Sensor is a device with several built-in components. The most important is the depth camera which consists of a near-infrared (NIR) projector and a near-infrared camera. The depth is estimated by projecting a dotted NIR pattern, which is viewed by the NIR camera. By comparing the known projected pattern to the pattern seen by the NIR camera, the depth can be estimated from the displacement of the dots. Due to occlusions of the NIR projector and the NIR camera and surfaces with bad reflectivity, parts of the image lack a valid depth value. The NIR camera has a resolution of 1280 x 1024 pixels and outputs a resulting 11 bit depth image of 640 x 480 pixels. There is also a medium quality RGB camera with a resolution of 1280 x 1024 pixels. Both NIR, color and depth images can be collected from the Kinect. However, due to limitations of the USB bandwidth, NIR images and RGB images can not be collected at the same time. The NIR and RGB images can be collected in either their full resolution or down-sampled to 640 x 480 pixels. If the highest resolution of the NIR or RGB cameras are used the maximum frame rate is about 10 Hz instead of 30 Hz. The RGB and IR cameras can be modeled using standard pinhole camera models. The NIR pattern projector is using an Orthographic fisheye projection [Nordmark, 2012]. The depth image that is received from the Kinect device is constructed using the image from the NIR camera and can therefore be thought of as a standard pin-

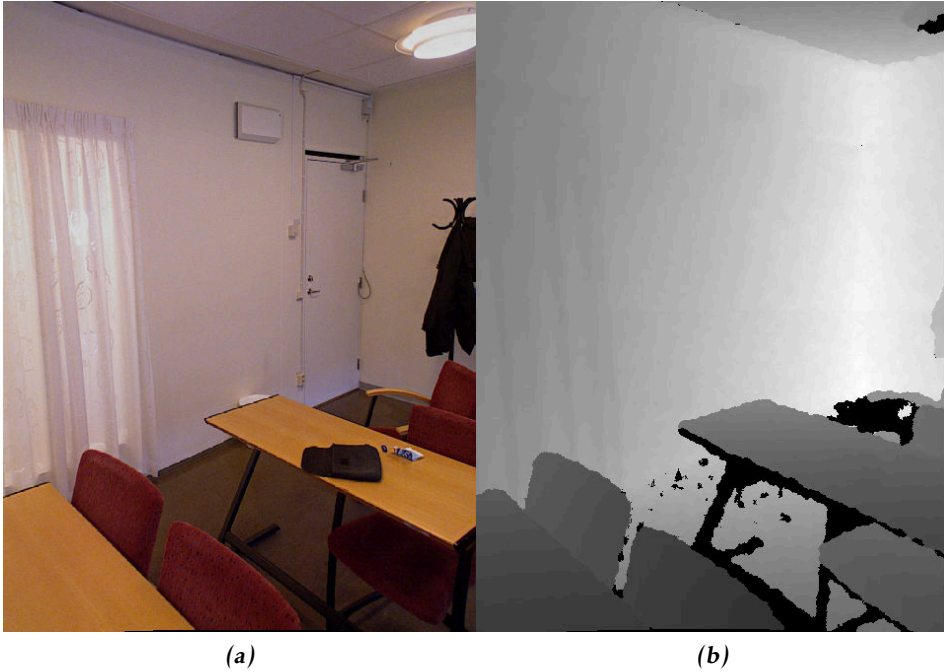


Figure 2.1: A set of depth and RGB images from the Kinect sensor. No depth information is available in the black areas of the depth image.

hole camera projection as well. The Kinect sensor also has four microphones, a LED and a tilt motor. These can be accessed but will not be used in this thesis. Examples of RGB and depth images are shown in figure 2.1. The black areas in the depth image are areas where no depth could be estimated.

2.1.2 Calibration

To convert the NIR pattern displacement map received from the Kinect into real world depth measurements, the mapping (2.1) presented in [Magenat, 2010] is used, resulting in a distance measure in millimeters parallel to the optical axis of the NIR camera.

$$d(x) = k_1 * \tan\left(\frac{x}{k_2} + k_3\right) \quad (2.1)$$

$$k_1 = 123.6, k_2 = 2842.5, k_3 = 1.1863$$

Using the constants k_1 , k_2 and k_3 , a depth value d in millimeter is given by the NIR pattern displacement x . The displacement x is a positive integer with eleven

bit precision returned from the Kinect device.

2.2 Homogeneous Coordinates

In order to represent an arbitrary point in a three-dimensional Euclidean space three components are required. However, in computer graphics and similar applications a homogeneous representation of four components is often more convenient. This makes it possible to represent any affine transformation or perspective projection using matrices. A homogeneous vector $\dot{\mathbf{x}} \in \mathbb{R}^4$ represents the coordinates in three-dimensional Euclidean space given by dividing the first three components with the fourth component;

$$\dot{\mathbf{x}} = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} \rightarrow \mathbf{x} = \begin{bmatrix} x/w \\ y/w \\ z/w \end{bmatrix} \quad (2.2)$$

Throughout this report homogeneous vectors will be denoted with a dot. From (2.2) follows that all scalings of a homogeneous vector represents the same point in Euclidean space.

$$\alpha \dot{\mathbf{x}} \rightarrow \mathbf{x}, \quad \alpha \neq 0. \quad (2.3)$$

The point at infinity can be represented by a homogeneous vector where $w = 0$.

2.3 Camera Model

To be able to convert the data collected from a camera into real world measurements, we need a correct model of how the world is mapped onto this camera. We split this model into two parts; the extrinsic part that describes the position and orientation of the camera, and the intrinsic, which describes how the scene is projected into the camera. The camera models used in this thesis are based on the thesis by [Nordmark, 2012].

2.3.1 Extrinsic Camera Model

The extrinsic parameters, or the pose, of a camera are defined by its position and rotation. In a three dimensional world this gives a total of six degrees of freedom with three degrees each for rotation and position.

A point \mathbf{p} defined in the world coordinate system is converted into the coordinate system of a camera by multiplying the homogeneous representation of the point with the extrinsic camera matrix T which can be decomposed into an orientation and a position.

$$\mathbf{p}' = T * \mathbf{p}, \quad T = \begin{bmatrix} R & -R^T \mathbf{t} \\ 0 & 1 \end{bmatrix} \quad (2.4)$$

In (2.4) the matrix R defines the direction of the camera. \mathbf{t} defines its position in world coordinates.

2.3.2 Intrinsic Camera Model

The intrinsic camera parameters define how a scene is projected onto the image sensor of the camera. The variables to take into account here are the horizontal and vertical fields of view \mathbf{fov}_{xy} , image resolution (w, h) , lens offset \mathbf{l}_d and sensor displacement \mathbf{c}_d .

For a camera that can be described using the pinhole camera model, a three dimensional point $\mathbf{p}_c = (x_c, y_c, z_c)^T$ located in the local coordinate system of the camera is projected onto a point $\mathbf{p}_i = (x_i, y_i)^T$ on the image sensor of the camera as in (2.5).

$$\mathbf{p}_i = \mathbf{p}_0 * (s_x, s_y) + \frac{(w, h)^T}{2}, \quad \mathbf{p}_0 = \frac{(x_c, y_c)^T}{z_c} \quad (2.5)$$

$$s_x = \frac{w}{2 * \tan(\frac{fov_x}{2})}, \quad s_y = \frac{h}{2 * \tan(\frac{fov_y}{2})}$$

Here \mathbf{p}_0 describes the projection of the point into the normalized image plane. $\mathbf{s} = (s_x, s_y)$ describes the scaling from the normalized image plane to the image sensor of the camera. However, real non-ideal pinhole cameras suffer from lens distortion. In this thesis, radial distortion, lens offset and sensor displacement are compensated for using (2.6)

$$\mathbf{p}_i = \left(\frac{(x_c, y_c)^T}{z_c} + \mathbf{l}_d \right) * (1 + k_2 * r^2 + k_3 * r^3 + k_4 * r^4) + \mathbf{c}_d \quad (2.6)$$

In (2.6) $r = \frac{\sqrt{x_c^2 + y_c^2}}{z_c}$ is the distance from the image center. Combining (2.5) and (2.6) where \mathbf{p}_0 is replaced with \mathbf{p}_i the mapping from \mathbf{p}_c onto its corresponding pixel \mathbf{p}_i results in:

$$\mathbf{p}_i = \mathbf{p}_i * (s_x, s_y)^T + \frac{(w, h)^T}{2} \quad (2.7)$$

By forming the intrinsic camera matrix

$$K = \begin{bmatrix} s_x & 0 & \frac{w}{2} \\ 0 & s_y & \frac{h}{2} \\ 0 & 0 & 1 \end{bmatrix}$$

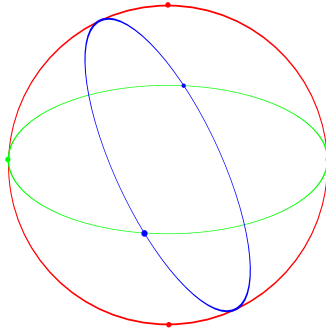


Figure 2.2: A set of three gimbals that can rotate around one axis each, illustrating the idea of Euler angles.

equation (2.7) can be expressed as

$$\dot{\mathbf{p}}_i = K * \dot{\mathbf{p}}_1 \quad (2.8)$$

2.4 Representation of Orientation

Orientation in three dimensions are in comparison to the two dimensional case non-trivial to represent in a both concise and unique way. However, based on a theorem by Leonard Euler from 1775 [Euler, 1775], we can deduce that any orientation can be achieved through a single rotation around one axis.

2.1 Theorem (Euler's Rotation Theorem). *In whatever way a sphere is turned about its centre, it is always possible to assign a diameter, whose direction in the translated state agrees with that of the initial state.*

2.4.1 Euler Angles

One of the more straightforward representations of arbitrary rotations in three dimensions is called Euler Angles. It consists of a set of three angles around three axes by which rotations are performed consecutively. This can be achieved using three gimbals as illustrated in figure 2.2. This construction is rather intuitive and also gives a concise representation of any arbitrary rotation. The resulting orientation is however dependent on the choice of axes and in what order the rotations are applied. In total there are twelve possible combinations. One common convention is to express the rotations in order of yaw, pitch and roll, or rotations around the z, x and y axes consecutively.

Another issue with Euler Angles is that in some configurations, the rotation axes align in the same direction and causes the loss of one degree of freedom as shown in figure 2.3. A way to observe this in the z-x-y case is to view each rotation in

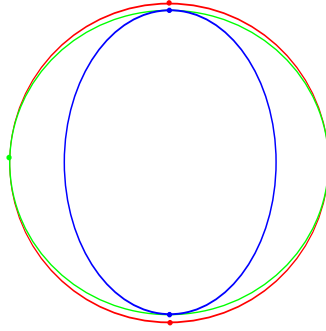


Figure 2.3: When the two outer gimbals align, two of the rotation axes also align and one degree of freedom is lost.

a local coordinate system where the y -rotation is performed first, proceeded by a rotation around x that in effect changes the global orientation of the previous y -axis. If this change in orientation of the y -axis aligns it in the same plane as the z -axis, one degree of freedom is lost. This is called a Gimbal Lock.

2.4.2 Rotation Matrix

When dealing with transformations of three dimensional vectors, rotation matrices are a very convenient tool to use. If expanded to homogeneous point coordinates, any rigid transform can be represented and easily applied using matrix-vector multiplication. Several transformations can also be combined using the matrix product. This is however a very redundant form of representation, and in order for a matrix to represent a proper rotation, further constraints must be met.

2.2 Definition (Rotation Matrix). A matrix R represents a rotation if and only if R is an orthogonal matrix with a determinant equal to one. _____

2.4.3 Unit Quaternions

Quaternions were first described by the Irish mathematician William Rowan Hamilton in 1843 as an extension of complex numbers into a four dimensional space.

2.3 Definition (Quaternion). A quaternion $\mathbf{q} = w + xi + yj + zk$ is a four dimensional vector consisting of one real scalar w and the complex components x , y and z . It is spanned by the basis elements $\mathbf{1}$, \mathbf{i} , \mathbf{j} and \mathbf{k} such that:

$$\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$$

In quaternion algebra the three operations *addition*, *scalar multiplication* and *quaternion multiplication* are defined. Quaternions are often written in the con-

densed notation

$$\mathbf{q} = (w, \mathbf{v}), \quad \mathbf{v} = xi + yj + zk$$

2.4 Definition (Quaternion Conjugate). The conjugate of a quaternion $\mathbf{q} = (w, \mathbf{v})$ is defined as $\bar{\mathbf{q}} = (w, -\mathbf{v})$

2.5 Definition (Unit Quaternion). The Unit Quaternion of \mathbf{q} is defined as

$$U\mathbf{q} = \frac{\mathbf{q}}{\|\mathbf{q}\|}$$

Any rotation in three dimensions can be represented using a unit quaternion $\mathbf{q} = (w, \mathbf{v})$ where the vector \mathbf{v} is parallel to the rotation axis and $w = \cos(\frac{\theta}{2})$ where θ is the magnitude of rotation. The vector $\mathbf{u} \in \mathbb{R}^3$ can be rotated by a unit quaternion \mathbf{q} as $\mathbf{p}' = \mathbf{q}\mathbf{p}\bar{\mathbf{q}}$ where $\mathbf{p} = (0, \mathbf{u})$.

2.5 3D Model Representation

Today, the undoubtedly most common method to represent three dimensional structures is polygon meshes. They have been used for a long period of time in computer graphics and games for which they are well suited. The construction of a model of an environment from a set of depth maps does however pose some challenges. For example, the ability to easily merge new data into a pre-existing model is needed. Uncertainties and noise in the data and areas where no data exists at all also needs to be considered. Many of the techniques used for improvement of low resolution meshes by texture manipulation methods would also be cumbersome, as a highly detailed geometry is needed.

2.5.1 Point Cloud

As the output of the depth camera is a depth image, we can, using a camera model, translate these depth measurements into a discrete set of points in three dimensional space. As we have no way of knowing the volumetric size of these discrete samples, this method is not well suited for visualization. There are however methods to create other types of representations based on assumptions on how these samples are combined into surfaces.

2.5.2 Polygon Mesh

Meshes built from polygons (most commonly triangles) are the most commonly used representation of three dimensional models. They are very efficient for representing large flat areas, both in a memory consumption and in a performance point of view. Partly due to this fact, they have been and are used for a wide range of computer graphic applications. They also allow some degree of freedom in term of deformation and modification of models. They are however quite memory and performance inefficient in the case of finer geometric detail or even

simple curvatures. Because of this textures are often used to manipulate the properties of the flat polygons when applying light and calculating the color of each surface fragment in order to fool the eye.

2.5.3 Voxel Volume

Representing a volume and the structures therein using voxel volumes are arguably the most generic method possible. A voxel volume divides a volume into a dense grid of volume elements called voxels. Each voxel can have different states based on if it contains an object or not and possibly other attributes, for example information about the material of the object it contains. Such a structure could in theory represent any possible environment within that volume with no other limitation than the resolution of the voxel grid. This is however what makes it impractical as it requires immense amounts of data unless voxels of a very large size are used, creating results of unacceptably low levels of detail.

2.5.4 Sparse Voxel Octree (SVO)

A Sparse Voxel Octree (SVO) [Laine and Karras, 2010] utilizes the fact that we only need to represent information about volumes that contain surfaces. There is no need to split empty regions or the inside of objects into a fine grained grid as they do not contain any information of value when rendering. Considering the fact that basically every conceivable scene are dominated by either open space or the interior of objects, this allows huge memory savings. A straightforward method of utilizing this sparsity is to create a voxel octree, where voxels that contains surfaces are split into eight child voxels in a recursive manner until a desired smallest voxel size is reached.

In the extreme example of an empty cube with sides of one meter and a smallest voxel size of one millimeter where a single point sample is added, the sparse voxel octree requires $1 + 8 \lceil \log_2 1000 \rceil = 81$ *voxels* compared to the dense voxel volume where $1000^3 = 10^9$ *voxels* are needed.

A sparse voxel octree does however pose some challenges when constructing one from a point cloud. If the distances between neighboring points that are sampled from the same surface are bigger than the smallest voxel volume we get holes in the surface. This can to some extent be addressed if normal approximations are available for the points in the point cloud. It is however not enough to simply increase the voxel size as this on one hand will decrease the detail level on close ranges and on the other hand not address the issue at large distances or at surfaces with a steep angle towards the direction from where the point samples were captured. A SVO is however easily constructed from a polygon mesh as these represents dense surfaces.

Sparse voxel octrees have attracted some attention from the computer graphics area. One example of this is [Laine and Karras, 2010]. This is due to a combination of their memory efficiency, the possibility to convert to and from meshes, and the ability to perform ray casting efficiently, much faster than when dealing with

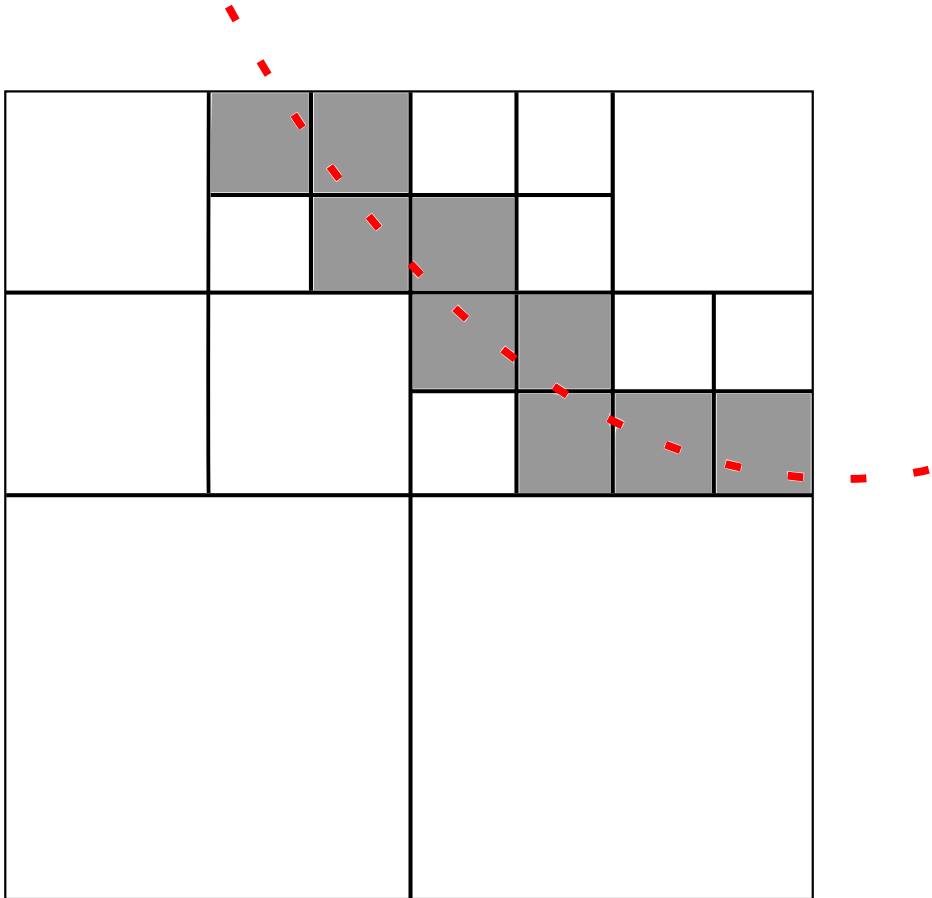


Figure 2.4: A quadtree of area elements representing a one-dimensional line. In the same way an octree of volume elements can be used to represent a two-dimensional plane.

meshes, which makes it possible to use advanced realistic methods of lighting during real-time rendering.

2.5.5 Signed Distance Field (SDF)

The Signed Distance Field or Signed Distance Function is a mathematical construction describing the signed distance in every point \mathbf{x} to the boundary of a set S . [Dapogny and Frey, 2012]

2.6 Definition (Signed Distance Function). Let $S \subset \mathbb{R}^n$ be a bounded set. The signed distance function $SDF_S(\mathbf{x})$ to S in $\mathbf{x} \in \mathbb{R}^n$ is defined as

$$SDF_S(\mathbf{x}) = \begin{cases} -d(\mathbf{x}, \delta S) & \text{if } \mathbf{x} \in S \\ 0 & \text{if } \mathbf{x} \in \delta S \\ d(\mathbf{x}, \delta S) & \text{otherwise} \end{cases} \quad \text{where } d \text{ is the Euclidean distance measure.}$$

If the boundary δS of the set S is defined as the surface of a three dimensional structure, the signed distance function $SDF_S(\mathbf{x})$ can be used to represent this structure.

In figure 2.5 a grid of samples of the signed distance function is illustrated, where red samples represent negative values while the green represent positive values. The represented surface structure can be recreated from these samples by interpolation between neighboring samples with opposite signs.

In the same manner as for the voxel volume, the simplest implementation of a discrete sampling of a Signed Distance Function, utilizes a dense grid. Thus, memory consumption is a problem. However, as the Signed Distance function contains more information for every voxel, it does not need to be sampled as densely as the voxel volume to achieve the same level of detail. The surfaces in the structure are located at the zero level of the signed distance function. Even though it is very unlikely to find any sample with a signed distance value of exactly zero, an approximation of the location of the sampled surfaces can be found by locating opposite signs between neighboring samples and interpolating linearly between them based upon their absolute distance value. Thus sub-sample precision can be achieved.

The SDF is often used with some modifications by systems that utilizes depth maps to construct representations of a three dimensional scene. This is partly due to the elegance when constructing and updating a SDF directly from a set of point samples by approximating the surface distance as the distance to the closest point sample. One example is [Izadi et al., 2011] where a truncated version of the SDF, in short TSDF, is used.

2.6 Pose Estimation

In many computer vision applications, one of the major challenges is the estimation of the relative pose of the observer. This problem is commonly solved using either dense image alignment or sparse feature matching [Szeliski, 2006]. In this

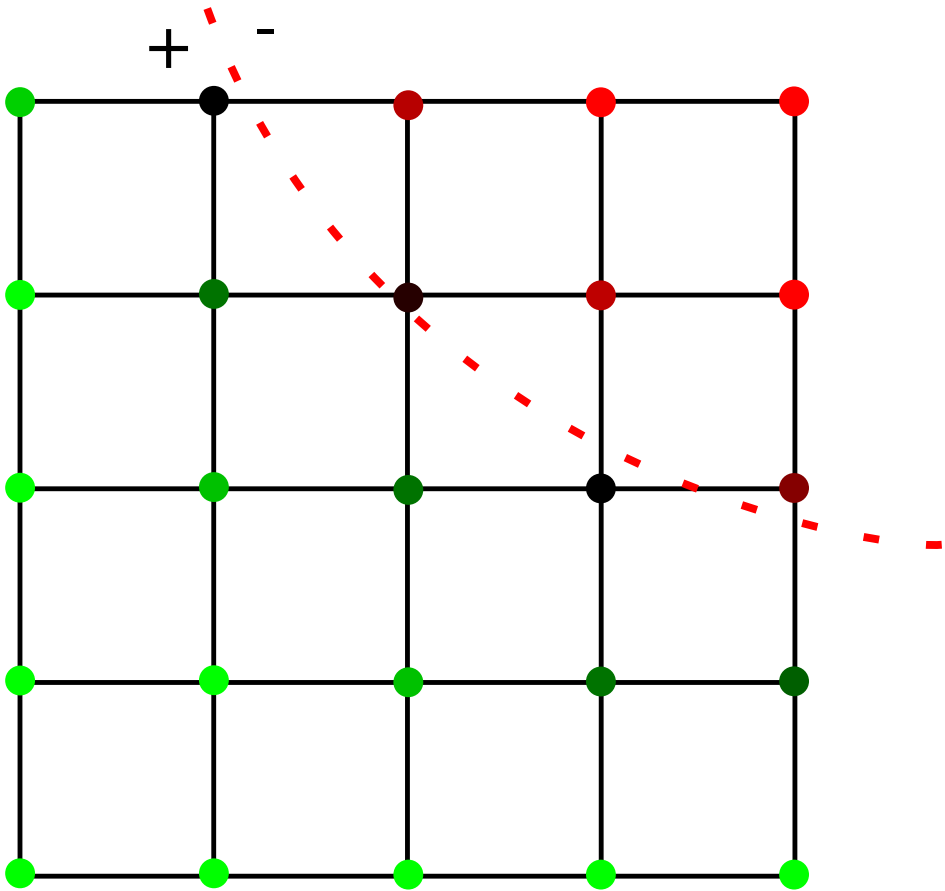


Figure 2.5: A grid of samples of the signed distance function in an area that contains a surface.

thesis, both the RGB and the depth camera of the Kinect device are used to match a limited set of image features. Several methods based on alignment of depth images have been presented, for instance by [Izadi et al., 2011] and [Erik Bylow and Cremers, 2013].

2.6.1 Dense Image Alignment

One way to align to similar images is to utilize the entire images with all their pixels to get a good correlation between them. This is usually done through minimization of a chosen cost measure that locally decreases towards an optimal relative pose where the images match best. If an appropriate cost function is given together with a good enough initial solution, sub pixel precision can be obtained. Due to the fact that any cost function will contain a multitude of local minima, a good initial solution is needed in order to find the global minimum using numerical methods. In order to achieve invariance to changes in illumination, maximization of the normalized cross-correlation is often used.

2.6.2 Sparse Image Alignment

A major advantage of alignment through matching of sparse image features is the ability to find a solution without the need for an initial estimate. Sets of feature points can be extracted from pairs of images and matched across images solely based on image content and not on the spatial locations of the images. These matches can be used to calculate relative poses between the images. Solutions based on variants of this group of methods have been presented in several well known publications, for example [Klein and Murray, 2007] and [Williams et al., 2007]. A wide range of algorithms for extracting robust sets of features and methods for matching these exists. An extensive comparison of the robustness and performance of different methods to detect image features is made in [Tuytelaars and Mikolajczyk, 2008]. In this thesis, the FAST corner detector presented in [Rosten and Drummond, 2005] and [Rosten and Drummond, 2006] is used. For feature matching, the Fast Retina Keypoint descriptor (FREAK) is used [Alahi et al., 2012]. The process of finding a good rigid transform between two images starts with finding robust image features in both images. A descriptor is then calculated for each of these features. These are then used to find good matches between features in the image pair which in turn makes it possible to estimate a relative pose.

Feature Detection

To get a good estimate of a rigid transform between two images a set of point pairs that correspond to the same three dimensional points is needed. It is not necessary to find more than a few correspondences in order to estimate this transformation. A feature detection algorithm is used in order to select a set of good image features that easily are traced between different images. There are several different methods to do this. In this thesis the FAST feature detector has been chosen.

Feature Descriptors

In order to be able to match points in two images that correspond to the same projected three dimensional object, descriptors of these image points are compared. In order to match points from images that are rotated or translated compared to each other, or where the lighting conditions are different, a descriptor that is invariant to rotation, scaling and overall illumination is needed. In this thesis the Fast Retina Keypoint (FREAK) [Alahi et al., 2012] is used. The resulting descriptors are 64- dimensional bit-vectors. Pairs of feature points are matched to each other using the Hamming distance between their descriptors.

Orthogonal Procrustes Problem

The problem of finding the best orthogonal mapping R of a set of points A to the set B is called the Orthogonal Procrustes Problem. A generalized solution to this problem was presented in 1966 by Peter Schönemann in [Schönemann, 1966]. This can be used in the case when the three-dimensional positions of a set of corresponding observations are known for an image pair. In the minimal case only three point correspondences are needed to solve this problem, which gives an exact solution. If more points are used, the transform that minimizes the cost function (2.9) is sought.

$$\epsilon(R) = \sum_{i=1}^n \|\mathbf{r}_{B,i} - R\mathbf{r}_{A,i}\|^2, \quad n \geq 3 \quad (2.9)$$

In (2.9) $\{\mathbf{r}_{A,i}, \mathbf{r}_{B,i}\}$ are the coordinates of a set of point correspondences $\{\mathbf{x}_{A,i}, \mathbf{x}_{B,i}\}$ relative to the centers $\{\mathbf{c}_A, \mathbf{c}_B\}$ of the point sets A and B .

$$\begin{aligned} \mathbf{c}_A &= \frac{1}{n} \sum_{i=1}^n \mathbf{x}_{A,i}, & \mathbf{c}_B &= \frac{1}{n} \sum_{i=1}^n \mathbf{x}_{B,i} \\ \mathbf{r}_{A,i} &= \mathbf{x}_{A,i} - \mathbf{c}_A, & \mathbf{r}_{B,i} &= \mathbf{x}_{B,i} - \mathbf{c}_B \end{aligned} \quad (2.10)$$

The cost function (2.9) is minimized by performing a singular value decomposition of the product $A * B^T$, where A and B are $3 \times n$ matrices with the points $\mathbf{r}_{A,i}$ and $\mathbf{r}_{B,i}$ in their columns. The R that minimizes $\epsilon(R)$ is then given by enforcing orthonormality:

$$R = V * U^T, \quad U * S * V^T = A * B^T$$

This does however only enforce that R is an orthogonal transformation where $\det R = \pm 1$. A negative determinant results in a reflection and thus not a proper rigid transform. In that case there is no proper rotation that fits the given set of corresponding points. A rigid transform that produces the smallest distance between corresponding points can however be found by changing the sign of the column of U that corresponds to the smallest singular value of $A * B^T$, thus

enforcing $\det R = +1$. After a rotation R have been determined the translation vector between the point clouds is given by

$$\mathbf{t} = \mathbf{c}_B - R * \mathbf{c}_A \quad (2.11)$$

RANSAC

Using feature descriptors is a good way of finding corresponding points between two images. It is however hard to attain a set of corresponding points without incorrect matches purely based on local image feature descriptors. This is mainly due to the fact that most images contain repeating structures and multiple similar objects. To select a good subset of correct correspondences from a set of matches the Random Sample Consensus (RANSAC) algorithm first presented in [Fischler and Bolles, 1981] is often used. The basic idea of the algorithm is to from a data set randomly select the minimal number of samples required to uniquely create a model, and then compare all other samples to this model.

Data: data_set, iterations, threshold

Result: best_model

$i = 0$;

best_score = 0;

best_model = 0;

while $i < iterations$ **do**

 subset = select_random_subset(data_set);

 model = create_model(subset);

 number_inliers = count_inliers(model, data_set, threshold);

if $number_inliers > best_score$ **then**

 best_score = number_inliers;

 best_model = model;

end

end

inlier_subset = select_inlier_subset(best_model, data_set, threshold);

best_model = create_model(inlier_subset);

Algorithm 1: The Random Sample Consensus Algorithm (RANSAC)

2.6.3 Iterative Closest Point

The Iterative Closest Point algorithm is widely used for motion estimation in situations when initial estimates of the relative poses between consecutive frames are available, for example in robotics. The main idea is to repeatedly find matches between points in two models based on distance and finding a relative transformation that minimizes an error metric for the distance between these points, and then find new matches after applying the found transformation. There are several variants of the ICP algorithm which uses different methods of selecting subsets of point pairs to be used or different ways to define the error metric be-

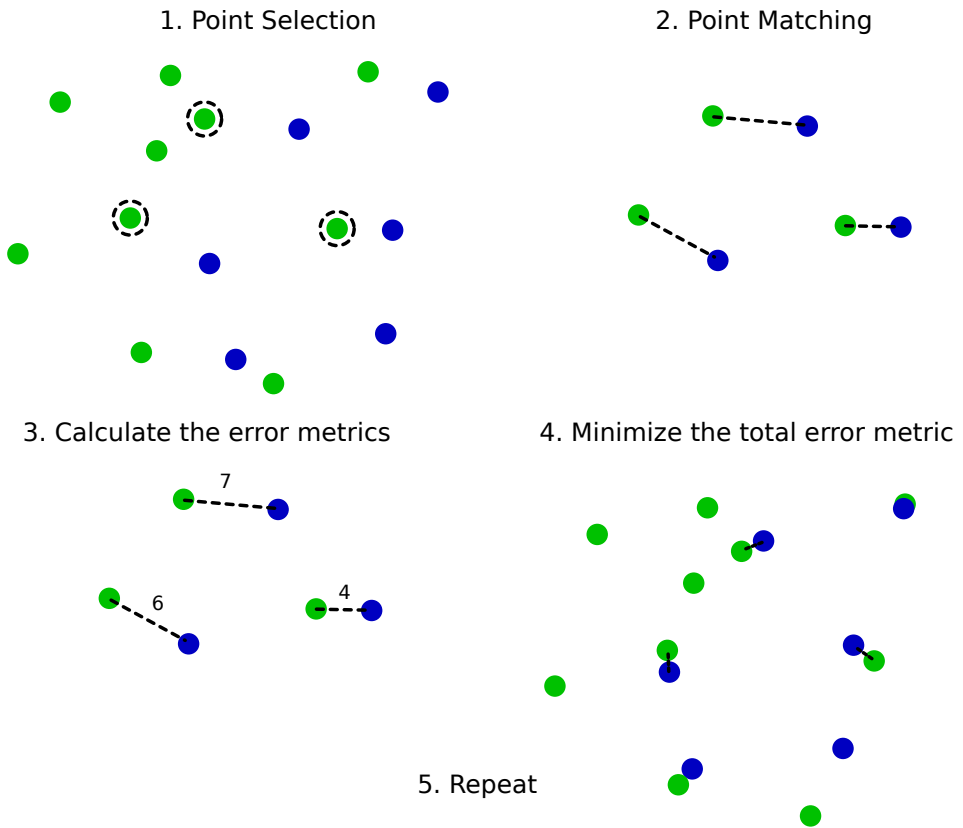


Figure 2.6: An ICP algorithm applied to two sets of points.

tween pairs of points. In figure 2.6 one variant of the ICP algorithm is applied to two sets of points in order to align them to each other. In this implementation, three random points are first selected from one of the point sets. These are then matched to the points in the second set located at the closest Euclidean distances. An error metric is then calculated as the sum of the Euclidean distances between the points of each pair. A rigid transform that minimizes the calculated error metric is then found and applied to the entire second point set. To further improve the result this process is repeated until a sufficiently small error is reached or a maximum number of iterations have been performed.

In [Rusinkiewicz and Levoy, 2001] a number of variants of the ICP algorithm are evaluated and compared to each other. It is possible to choose different methods of selecting the points that are used. All available points can be used, or a randomly or uniformly selected subset. Matching is often done by measuring the Euclidean distance between a point in the first model to the points in the second and pairing it to the closest one in the second model. Other methods of point matching include finding the closest point measured in an image plane or along

```
while  $i < \textit{iterations}$  do  
  1. Select a set of points in both models;  
  2. Match the points in the two models;  
  3. Calculate a chosen error metric between each point pair;  
  4. Minimize the total error metric using a rigid transform  $T$ ;  
end
```

Algorithm 2: The Iterative Closest Point Algorithm (ICP)

a surface normal. The error metric is often defined as a point-to-point distance or a point-to-plane distance, that is; the point-to-point distance along a surface normal. In [Henry et al., 2010] a combination of sparse image feature matching and ICP depth alignment is presented.

3

Theory

The system presented here is built with the task in mind of constructing an as high quality model as possible of an indoor environment using a Kinect sensor. Except from the image capture stage, the process is automatic.

3.1 Overview

In the following section an overview of the system is given. To start with, the user manually collects pairs of RGB and raw depth images from a few locations. These images are preprocessed to reduce noise and distortion. The resulting images are then aligned in 360 degree panoramas using feature matching. The first of the panoramas is used to create a model representing the surrounding environment. The other panoramas are then one by one aligned to this model and added to it.

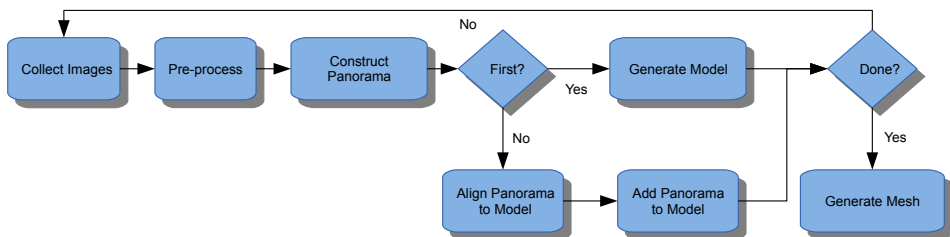


Figure 3.1: Overall flowchart of the system.

3.2 Image Collection

The Kinect sensor is mounted vertically on a camera tripod so that both the RGB and the depth camera have their centers as close as possible to the axis of rotation of the tripod. The tripod is then positioned at a location from where a series of images are collected at different orientations. The Kinect sensor is only rotated around the vertical axis of the tripod. For each orientation of the Kinect sensor a series of raw RGB and depth images are collected. These are used to reduce noise through multi-sampling. Images are taken with an offset of a few degrees so that there is some overlap between each two consecutive images. The field of view of the camera must be considered when choosing the angle offset to get enough overlap. During the evaluation of the system an offset of ten degrees has been used, even though it has shown to be able to handle offsets up to twenty degrees. An illustration of the image collection process is shown in figure 3.2.

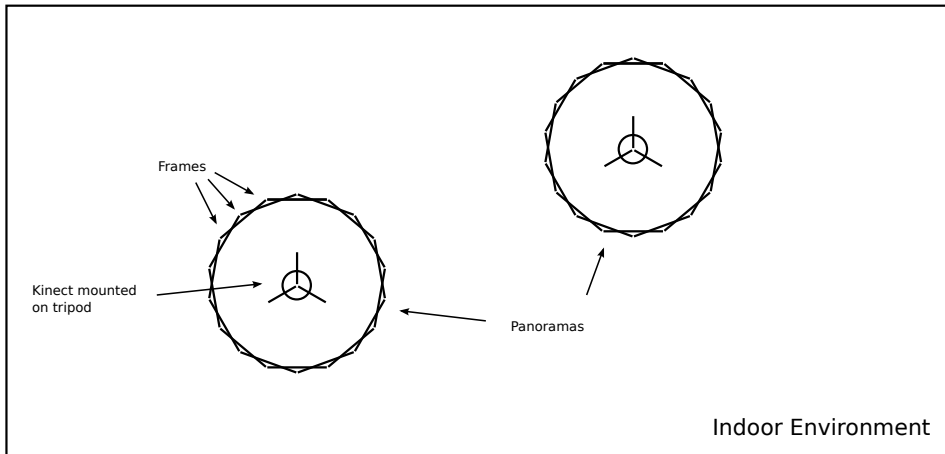


Figure 3.2: Overview of the setup of the system when collecting images.

3.3 Pre-processing

The raw images must be pre-processed in order to reduce camera distortion, reduce noise and translate raw depth values into real-world measures.

3.3.1 RGB

An easy method to reduce the amount of noise in the RGB images is to take multiple images for each orientation and average them into one image. The distortion of the resulting image is then reduced using a given set of parameters defining lens offset, chip offset and radial distortion according to (2.6).

3.3.2 Depth

Averaging a set of depth images does not produce desirable results as this can create unwanted points at depths where there are no objects. Instead a median value in each depth pixel is used. When using an even number of images the lower value of the two center values are used instead of taking an average, in order to avoid introducing any new depth values. As parts of the depth images do not contain any valid depth measurements due to occlusion of the NIR projector or the NIR camera, or to surfaces with bad NIR reflectivity, only valid depth values are considered when calculating the median.

3.4 Panorama Construction

To simplify the process of pose estimation and to reduce the risk of errors, a set of frames is collected from each viewpoint as earlier described and aligned together in 360° panoramas. For each panorama, consecutive overlapping frames are aligned to each other through sparse image alignment of their RGB images, where the positions of the frames are locked at a common origin. This limits the pose estimation problem to finding a set of rotations R_i for every frame in the panorama.

The advantage of using RGB images here is that the RGB images have valid data over the entire image, which is not the case for the depth images. The color images do however contain a substantial amount of noise. This can be reduced by multi-sampling and averaging. This is harder to do with depth images in a good way. Using RGB images is also better if there are large flat areas that contain varying textures, while depth images would be better if there are large uniformly colored areas. As the relative position between the depth and the RGB camera is fixed and known, the camera pose of an RGB image also gives the pose of the corresponding depth image. An example of a set of images aligned into a panorama is shown in figure 3.3.

3.4.1 Feature Descriptors

The panorama construction is initialized by setting the orientation of the first frame $R_0 = \mathbf{I}$. A FAST feature detector is then used in order to find a set of feature points $\mathbf{f}_{i,k}$ for every frame $i \in 1..n$. In order to ensure that sets of image features are evenly distributed the images are first divided into grids of 15x15 cells where the strongest features are found in each cell. For every feature point $\mathbf{f}_{i,k}$ a descriptor vector $\mathbf{d}_{i,k}$ is calculated using FREAK [Alahi et al., 2012].

3.4.2 Feature Matching

When estimating the relative rotation between two overlapping frames A and B in a panorama, at least three correct point correspondences are needed. To find a good set of possible correspondences each descriptor $\mathbf{d}_{k,A}$ in frame A is compared to each descriptor $\mathbf{d}_{l,B}$ in frame B using the Hamming distance.

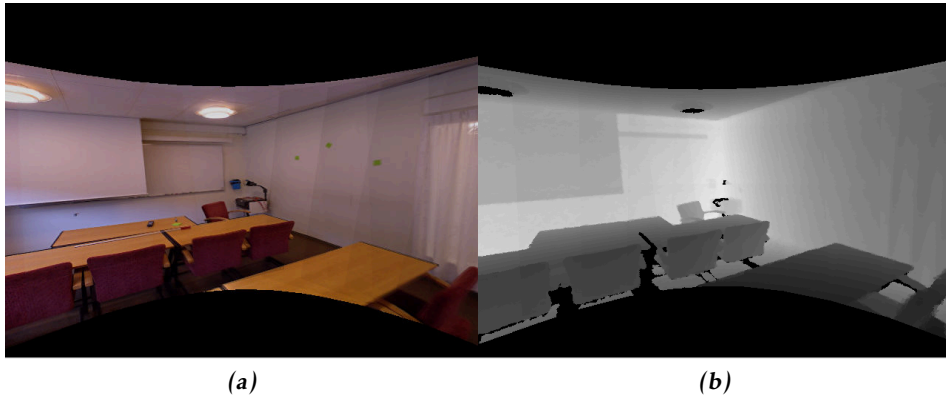


Figure 3.3: A set of depth and RGB images aligned in a panorama.

3.1 Definition (Feature Match). A match in the frame B to the feature point $\mathbf{f}_{k,A}$ in frame A is defined as

$$m_{A,B}(k) = \underset{l}{\operatorname{argmin}}(H(\mathbf{d}_{k,A}, \mathbf{d}_{l,B}))$$

where $H(\mathbf{d}_a, \mathbf{d}_b)$ is the Hamming distance between two descriptor vectors.

In order to screen the set of matches from possible incorrect ones, all matches where $m_{i,j}(m_{j,i}(l)) \neq l$ are ignored.

3.4.3 Rotation Estimation

There is no way of knowing the distance to a point that only is seen from a set of RGB images taken from one common location. Any two points that together aligns with the common camera center will be projected onto the same pixel in each image. This is illustrated in figure 3.4.

Due to this fact, when estimating the rotation between two frames with a common camera center, the three dimensional points which are projected into the images can be assumed to lie on a unit sphere centered around the camera center. Using correspondences given from feature matching, it is thus possible to estimate a rotation by solving the Orthogonal Procrustes Problem 2.6.2 for these points on the unit sphere. It is however inevitable that there will be some incorrect correspondences from the matching of feature descriptors. These needs to be removed before this estimation is done. By implementing the RANSAC algorithm outlined in 2.6.2 where in each iteration a rotation R is generated for a random set of three correspondences, an inlier set can be defined according to definition 3.2. The RANSAC algorithm then aims to find the rotation that creates the largest inlier set.

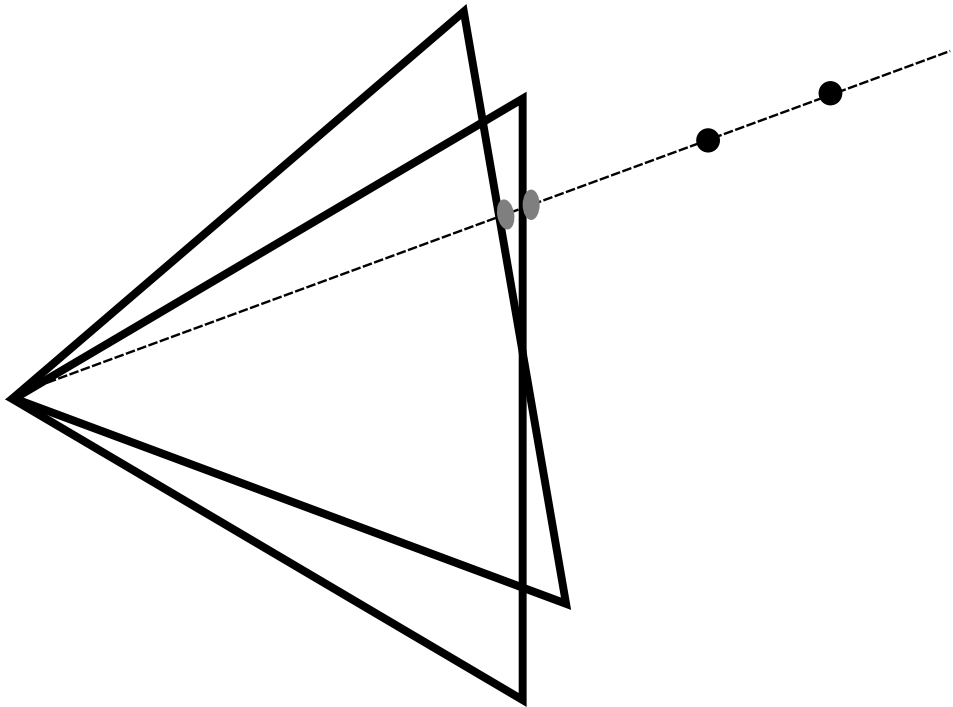


Figure 3.4: Any two points that together aligns with a common camera center will be projected onto the same pixel in each image sharing that camera center.

3.2 Definition (Inlier set - Rotation). The inlier set of correct corresponding points between frame A and frame B given a rotation R is defined as

$$C_{A,B}^R = \{m_{A,B}(k)\}, \quad \|R\mathbf{p}_{k,A} - \mathbf{p}_{m_{A,B}(k),B}\|_2 < \lambda, k = 1 \dots n\}$$

$$\mathbf{p}_{k,A} = \frac{(x_{k,A}, y_{k,A}, 1)^T}{\sqrt{x_{k,A}^2 + y_{k,A}^2 + 1}}$$

where $(x_{k,A}, y_{k,A})$ is the normalized camera coordinate of the feature point $f_{k,A}$. λ is a threshold that determines the largest distance between inlier correspondences after applying the rotation R .

Using the largest inlier set found by RANSAC, a good estimation $R_{A,B}$ of the relative rotation between the frames A and B is found by solving the Orthogonal Procrustes Problem. A threshold of $\lambda = 0.002$ have been found suitable. The resulting inlier set is stored for later use.

As the orientation of the first frame in a panorama is initialized to $R_0 = \mathbf{I}$, the orientation of each new consecutive frame can be calculated using the orientation of the previous frame and their relative rotation (3.1).

$$R_B = R_{B,A} * R_A \quad (3.1)$$

3.4.4 Loop Closing

Even though the estimations of the poses are quite good when looking at a set of poses nearby each other, there are still small errors present which stacks up to a noticeable drift over a larger set of frames. In order to counter this problem the last frame of a circular panorama are matched with the first ones where there is an overlap large enough. The RANSAC algorithm is then performed on these frames. This gives a relative rotation that is disregarded for now, while the resulting set of inliers is stored. Apart from the relative rotations between each pair of neighboring frames in the panorama, a set of inlier point correspondences is available for each pair of neighboring frames. By minimizing the distances between these corresponding points on the unit sphere a better set of frame orientations R_i can be found, which eliminates the drift. The function that should be minimized is thus the sum of all cost functions (3.2) for each pair of neighboring frames.

$$\epsilon_{A,B} = \sum_{k=1}^n \|R_{A,B} * \mathbf{p}_{k,A} - \mathbf{p}_{m_{A,B}(k),B}\|^2, \quad (3.2)$$

As the previously calculated estimations of the orientations of each frame gives a good initial solution, a numerical method can be used to find this minimum. The Levenberg-Marquardt optimization method gives a fast convergence and is used here. During the optimization the orientations are represented using unit

quaternions (see 2.4.3) in order to reduce the number of parameters and more easily enforce proper rotations.

3.5 Model Generation

In several other works with the goal of creating a model from data generated by depth sensors such as the Kinect sensor, implementations of the Signed Distance Field have been used. A few examples are [Izadi et al., 2011], [Erik Bylow and Cremers, 2013] and [Ricao Canelhas, 2012]. These systems are able to produce high quality results for small spaces. However, due to the regular grid structure where the SDF is sampled, the required amount of memory grows rapidly with the size of the mapped volume. To solve this problem an octree-structure of SDF samples have been implemented in this thesis, hereon referred to as a Sparse SDF Octree (SSDO). This section outlines how these are generated.

3.5.1 Sparse SDF Octree (SSDO)

A sparse SDF octree combines the ideas behind SVO and SDF by calculating the signed distance value for every voxel in a SVO in all levels of the octree. This gives the possibility to achieve both sub-sample precision at the same time as the sparse structure of the represented volume is utilized. A SSDO is also easier to construct from a point cloud compared to the SVO. A similar structure to the one used here is presented in [Jamriška]. A significant difference is however that the goal of this thesis is to construct a model representation from a sparse point cloud and not from a mesh. This poses some significant challenges as the structure of surfaces between samples in the modeled environment is not known beforehand, whereas if starting from a mesh, samples can be fetched from any position along any surface. The idea behind the SSDO is to solve this by trying to find surface intersections inside voxels at a coarser level of the octree if no point samples from the original point cloud are found at the finest level.

An issue with the naive implementation of an SSDO where one sample of the signed distance function is stored for the center of each voxel is the difficulty to find the neighbors of a given voxel, which is required in order to be able to interpolate the signed distance function between neighboring voxels. In order to avoid having to do this, the signed distance function is sampled in every corner of each voxel instead of in the center, making it possible to interpolate inside the voxel without having to access any external data. This requires eight times as much memory per voxel but makes the rendering process much simpler. In order to reduce memory usage, only the leaf voxels need to store the signed distances in their corners, as the signed distances in the corners of a parent voxel easily can be fetched from its children. This does however not eliminate the data redundancy between neighboring voxels. The structure of the SSDO is shown in figure 3.5.

Apart from the signed distance, every leaf voxel also store a weight and a color for every corner. The weight is needed when updating the signed distance and

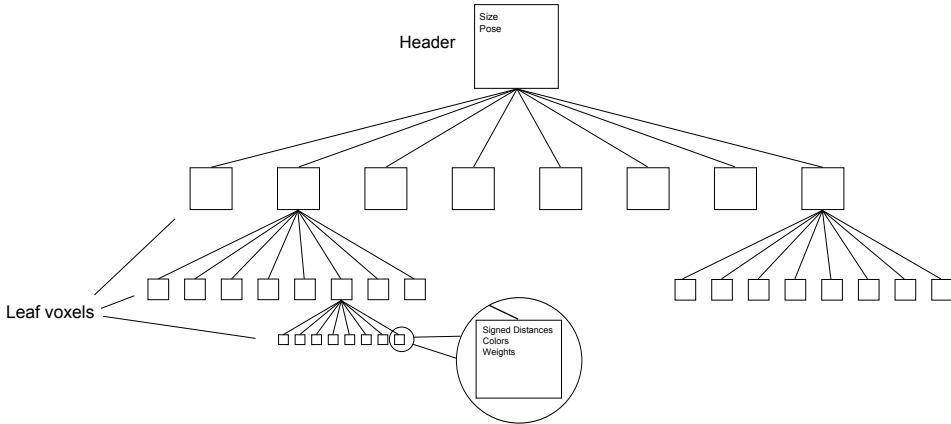


Figure 3.5: Structure of a sparse SDF octree.

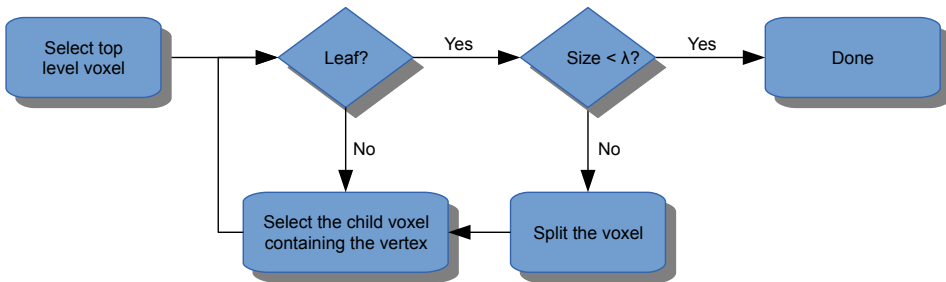


Figure 3.6: Flowchart over how the tree structure is updated when a new vertex is added.

color measures using weighted running averages. Every voxel also store a pointer to its parent and a variable defining where in the parent voxel it is located. One complete panorama at a time is used to create and update the global SSDO. The resulting model is then used to align the next complete panorama, which then in turn can be added, further refining the model.

Tree Generation

The Sparse SDF Octree is built by adding one frame at a time. To begin with the intrinsic and extrinsic camera parameters of a frame A together with its depth image are used to create a set of three-dimensional points $\mathbf{p}_{i,A}$. All these points are samples of surfaces observed in the scene. For each of these points, the leaf voxel in the octree where this point is located is found. If the size of this voxel is larger than a lower threshold λ_s , the leaf is split into eight new leaf voxels. This is repeated until the size of the leaf voxel is smaller than λ_s . The process is outlined in figure 3.6.

Using a smaller threshold λ_s gives a better resolution in the model at the cost

of higher memory consumption. However, the precision of the position of the vertices is dependent on the precision delivered by the Kinect sensor. There is no point in splitting the octree into voxels that are smaller than the resolution of the collected depth values.

Using the mapping between the raw NIR-pattern offset and real-world depth measures in (2.1), the bit precision at a distance d can be defined as its derivate with respect to the NIR-pattern offset.

3.3 Definition (Depth bit-resolution). The bit-resolution $P(d)$ of a depth value $d(x)$ given a NIR-pattern offset $x = 0, 1 \dots n$ is defined as

$$P(d) = \frac{dd(x)}{dx}(x(d))$$

Using the mapping in (2.1) the depth bit-resolution at a given depth d is

$$P(d) = \frac{dd(x)}{dx}(x(d)) = \frac{k_3}{k_1} \cos^{-2}(\arctan \frac{d}{k_3}). \quad (3.3)$$

As expected, the inaccuracy increases monotonically with increasing distance.

Update of the SDF samples

When the vertices $\mathbf{p}_{i,A}$ from the frame A have been used to split the sparse SDF octree V into smaller child voxels, the samples of the SDF in each corner of each voxel in the octree needs to be updated. In order to do so, assumptions must be made concerning the orientation of the surface from which the samples $\mathbf{p}_{i,A}$ are collected. This is necessary in order to distinguish between areas in front of the observed surface to those behind, and what the distance to the surface is. This assumption of the normal of the sampled surface could be made by calculating tangent vectors using the position of nearby vertices, as done in [Erik Bylow and Cremers, 2013]. This does however in turn require assumptions about which vertices that are taken from the same locally flat surface and is sensitive to noise in the depth images. In this thesis, a surface normal parallel and with opposite direction to that of the optical ray is used instead.

$$\mathbf{n}(\mathbf{p}_{i,A}) = \frac{\mathbf{p}_A - \mathbf{p}_{i,A}}{\|\mathbf{p}_A - \mathbf{p}_{i,A}\|_2} \quad (3.4)$$

In (3.4) \mathbf{p}_A is the position of the camera center of frame A .

This is a very rough estimate but gives a result that is good enough, and produces a normal approximation where $\mathbf{n}(\mathbf{p}_{i,A}) \cdot \mathbf{n}_{\text{surf}} > 0$. This is the same approach as used in [Izadi et al., 2011].

To update the SDF in a position \mathbf{v} inside the model using the frame A , the shortest distance from \mathbf{v} to the surfaces seen in A needs to be found. Instead of measuring

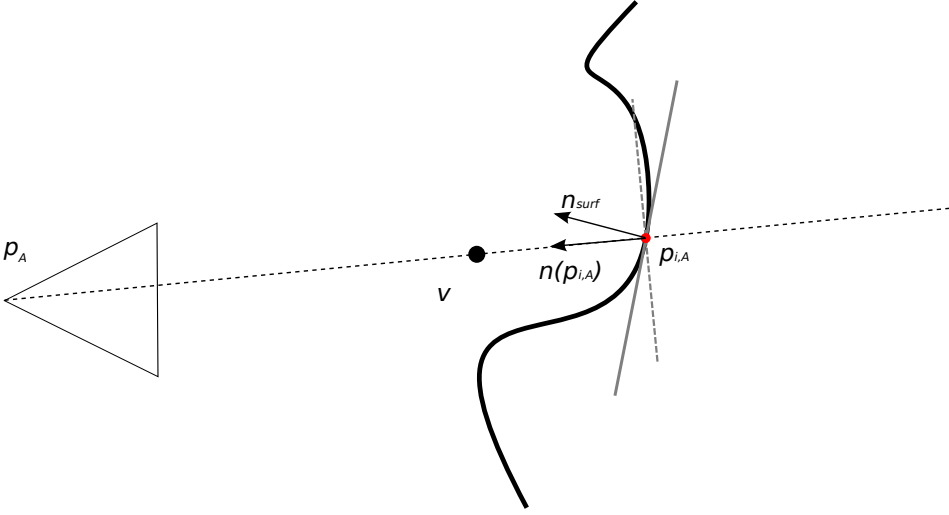


Figure 3.7: The normal of a sampled surface is approximated as parallel to the optical ray from the camera center of the frame from which the surface is observed.

the euclidean distance to all vertices $\mathbf{p}_{i,A}$, the faster method of projecting \mathbf{v} into the depth image of A is used. This projective method does not find the closest vertex but rather the closest vertex-projection as shown in figure 3.7. The signed distance to the surface observed by A is then approximated as the distance along the approximated surface normal $\mathbf{n}(\mathbf{p}_{i,A})$.

$$\begin{aligned} SDF_A(\mathbf{v}) &= (d_A(\mathbf{x}_i) * \dot{\mathbf{x}} - \mathbf{v}_A) \cdot \mathbf{n}(\mathbf{p}_{i,A}) \\ &= d_A(\mathbf{x}_i) * \|\dot{\mathbf{x}}\|_2 - \|\mathbf{p}_A - \mathbf{v}\|_2 \end{aligned} \quad (3.5)$$

$$\dot{\mathbf{v}}_A = T_A * \dot{\mathbf{v}}$$

\mathbf{x}_i : projection of \mathbf{v}_A in frame A according to (2.7).

\mathbf{x} : The point \mathbf{x}_i in normalized image coordinates.

When adding another frame B to the global sparse SDF octree V , a way to combine the signed distances from the new frame $SDF_B(\mathbf{v})$ with the existing ones is needed. The first observation made here is the fact that point samples of the SDF with a negative value are located behind the surfaces seen from the frame. These are in other words occluded and are therefore very uncertain. Another parameter to take into account is the depth value by which the signed distance is calculated, as this affects the depth precision (3.3). In order to get a resulting signed distance for a combined global sparse signed distance octree $SDF_V(\mathbf{v})$ with as high accuracy as possible, a weight $W(\mathbf{v})$ is added to each sample of the sparse SDF octree. In this way the SDF values are updated using a weighted running average. This is also done in [Izadi et al., 2011] and [Ricao Canelhas, 2012], which both uses a

simpler weight $W_A(\mathbf{v}) = 1$ resulting in a plain averaging, and in [Erik Bylow and Cremers, 2013] were a weight defined as a truncated negative exponential of the surface distance is used.

In this thesis the weights are stored as eight-bit unsigned integers and are used to update the SDF as:

$$SDF_{V,n}(\mathbf{v}) = \frac{SDF_{V,n-1}(\mathbf{v})W_{n-1}(\mathbf{v}) + SDF_A(\mathbf{v})W_A(\mathbf{v})}{W_{n-1}(\mathbf{v}) + W_A(\mathbf{v})}. \quad (3.6)$$

$$W_n(\mathbf{v}) = \min(W_{n-1}(\mathbf{v}) + W_A(\mathbf{v}), 255). \quad (3.7)$$

$$W_A(\mathbf{v}) = \begin{cases} 1, & \text{if } SDF_A(\mathbf{v}) < 0 \\ 1 + 254 * 2^{-\frac{P(d_A)}{100}} & \text{otherwise} \end{cases} \quad (3.8)$$

The bit-depth precision $P(d_A)$ used in (3.8) is defined in equation (3.3). Using the weight update method in (3.8) the weight is halved for every decimeter the precision decreases.

The structure of the octree is refined for every new frame that is added. Thus apart from updating all signed distances in the octree based on a new frame, all newly added leaf voxels must also be updated using all previously added frames.

Together with each SDF sample a color is also stored. The images captured by the RGB camera are taken with different exposure due to changing lighting conditions. It is therefore important to average the color at each SDF sample from several images in order to avoid ugly transitions between areas seen in images with high exposure and images with low exposure. The color averaging is done using the same weighted average as for the signed distance:

$$\mathbf{c}_{V,n}(\mathbf{v}) = \frac{\mathbf{c}_{V,n-1}(\mathbf{v})W_{n-1}(\mathbf{v}) + \mathbf{c}_A(\mathbf{v})W_A(\mathbf{v})}{W_{n-1}(\mathbf{v}) + W_A(\mathbf{v})}. \quad (3.9)$$

3.6 Panorama-Model alignment

When building a model of an indoor environment you want to use images taken from different viewpoints in order to capture surfaces that otherwise would be occluded. A method to find the relative position and rotation between images taken from different positions is therefore needed. In systems capturing continuous video with the task of motion tracking this is often solved by making the assumption that two consecutive frames have a rather small relative shift in position and orientation. In the current setup however, no knowledge about the relative position of the origin of two panoramas is available. However, as the rel-

ative rotations between all images within the panoramas are known, all features in these frames can be used in order to find a good transformation that aligns a panorama to a model generated from previous panoramas.

3.6.1 Pose Estimation through Feature Matching

To align a panorama to an existing model through feature matching as done when aligning frames within a single panorama (3.4), all the feature descriptors of the new panorama are matched to all those of the panoramas used to build the model. In this case, the assumption made in 3.4.3 concerning the distances of the feature points from the camera center can not be made, as the transformation between frames from different panoramas also contains a translation. Therefore another method to estimate a relative transformation must be used. One possibility is to use the Eight-point algorithm described in [Longuet, 1981] to estimate an Essential matrix E .

Given an Essential Matrix E a relative rotation can be determined. It is however only possible to extract the direction of the translation. To find the scaling, some real world measurements must be matched to the features used when estimating the Essential Matrix. Because of this, an easier solution is to map a depth value to each image feature point using the depth image of each frame, and then calculate the positions of the image feature points relative to the panorama. This mapping is described in 3.6.2. Given the depth $d_A(\mathbf{f}_{k,A})$ of the feature point $\mathbf{f}_{k,A}$ from frame A , its position $\mathbf{p}_{\mathbf{f}_{k,A}}$ relative to the panorama is defined in (3.10).

$$\begin{aligned} \hat{\mathbf{p}}_{\mathbf{f}_{k,A}} &= T_{RGB} * \hat{\mathbf{v}}, \\ \mathbf{v} &= K_{RGB}^{-1} * \hat{\mathbf{f}}_{k,A} * d_A(\mathbf{f}_{k,A}) \end{aligned} \quad (3.10)$$

When this mapping is done we have two point clouds of vertices with one feature descriptor each. Using the descriptors feature matching is done as described in 3.4.2.

The best transformation between a set of matching vertices can easily be found by calculating the offset between the centroids of the two sets of vertices and solving the Orthogonal Procrustes Problem 2.6.2. By applying this to the RANSAC algorithm for three point correspondences at a time, an rigid transformation $T = \begin{bmatrix} R & \mathbf{t} \\ 0 & 1 \end{bmatrix}$ can be found by defining inliers as point correspondences whose relative distances when applying the transformation T are smaller than a threshold λ .

3.6.2 Mapping between depth and color images

The Kinect RGB camera is located 24 millimeters along the positive x-axis from the NIR-camera. As the pixels in the depth image generated from the NIR-camera can be mapped to real world depth measurements according to (2.1), the position

of the vertices seen in the depth image relative to the position of the RGB camera is known. By projecting these vertices into the RGB camera, a color can be mapped to the pixels in the depth image.

It is a lot harder to find a mapping in the opposite direction. In order to find the pixel \mathbf{p}_d in a depth image that corresponds to a pixel \mathbf{p}_{RGB} in a RGB image, all vertices found in the depth image are projected into the RGB image. The depth value for a color pixel is then found by selecting the vertex from the depth image whose projection in the RGB image is closest to it.

$$\mathbf{p}_d = \underset{\mathbf{p}_d}{\operatorname{argmin}} \|\mathbf{p}_{RGB} - \mathbf{y}\|^2,$$

$$\dot{\mathbf{y}} \sim K_{RGB} * \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} * T_{RGB} * T_d^{-1} \dot{\mathbf{v}}_d, \quad (3.11)$$

$$\mathbf{v}_d = K_d^{-1} \dot{\mathbf{p}}_d * d(\mathbf{p}_d)$$

In equations (3.11) \mathbf{v}_d is the position of an observed vertex at the depth pixel \mathbf{p}_d relative to the depth camera. This vertex is projected into the RGB image pixel \mathbf{y} using the pose and intrinsic camera parameters T_{RGB} and K_{RGB} of the RGB camera. The RGB and the NIR cameras on the Kinect sensor are mounted 24 mm apart from each other along the x-axis. The relative pose between the RGB and the NIR camera is thus fixed and known and can be expressed as:

$$T_{RGB} * T_d^{-1} = \begin{bmatrix} 1 & 0 & 0 & 24 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \quad (3.12)$$

3.6.3 Pose Refinement through the Iterative Closest Point algorithm

The method described in 3.6.1 gives a rough but good initial estimate of the pose of a panorama. In order to get an even better position and orientation the Iterative Closest Point (ICP) algorithm is used with some modifications. The generic ICP algorithm is explained in 2.6.3. Two different variations of the ICP algorithm have been implemented and tested.

RGBD-ICP

The idea of RGBD-ICP is to combine matching of features based upon both feature descriptors and their relative positions. This is inspired by [Henry et al., 2010]. The set of inlier feature matches given from the RANSAC algorithm when the initial relative pose were estimated in 3.6.1 is used in the ICP algorithm in order to ensure that the result of the ICP algorithm does not deviate too much

from the initial solution. All other feature points in the panorama for which no correct feature descriptor matches were found are instead matched based on the shortest Euclidean distance to all the feature points of the panoramas used to build the SSDO model. These two sets of point correspondences are then used together in the ICP algorithm. The point correspondences originating from feature descriptor matching remain the same for each ICP iteration, while the set of correspondences based on Euclidean distances is updated for every iteration. The error metric used is a point-to-plane distance where the surface normal $\mathbf{n}_V(\mathbf{p})$ at a feature point \mathbf{p} in the SSDO model V is calculated using a numerical differential of the signed distance function.

$$\begin{aligned}\epsilon &= \epsilon_{RGB} + \epsilon_D, \\ \epsilon_{RGB} &= \sum_{k=1}^n \dot{\mathbf{n}}_V(\mathbf{p}_{m_{P,V}(k),V}) \cdot (T_{P,V} \dot{\mathbf{p}}_{k,P} - \dot{\mathbf{p}}_{m_{P,V}(k),V}) \\ \epsilon_D &= \sum_{l=1}^m \dot{\mathbf{n}}_V(\mathbf{p}_{m_{P,V}^{euc}(l),V}) \cdot (T_{P,V} \dot{\mathbf{p}}_{k,P} - \dot{\mathbf{p}}_{m_{P,V}^{euc}(l),V})\end{aligned}\quad (3.13)$$

In (3.13) $m_{P,V}$ are the feature descriptor matches between a panorama P and a SSDO model V , while $m_{P,V}^{euc}$ are matches between feature points based on the Euclidean distance.

$$m_{P,V}^{euc}(k) = \underset{l}{\operatorname{argmin}} \|T_{P,V} \dot{\mathbf{p}}_{k,P} - \dot{\mathbf{p}}_{l,V}\|^2 \quad (3.14)$$

In order to avoid incorrect matches, only points at a distance shorter than a threshold λ_{ICPD} are used. When minimizing the error metric, the relative pose converges towards a more accurate result when using the point-to-plane error metric instead of a point-to-point metric. This is illustrated in figure 3.8. The minimization is done numerically through a Levenberg-Marquardt least-square solver.

SDF-ICP

When performing the ICP algorithm, a costly part of the algorithm is the first step where matches between the points in the models are found. There are methods to speed this up, for example using a kd-tree search. However, if the error metric used is the point-to-plane distance, there is no reason to find point matches if there is another method of finding the surface distance between the points in the first model to the surfaces in the second model. This is exactly the case when dealing with the SSDO model, which contain samples of the signed distance function. By interpolating the samples of the SDF at the locations of the feature points of a panorama, an approximate value of the point-to-plane distance in each of these points can be found. The error function (3.15) can be formed and minimized using Levenberg-Marquardt optimization.

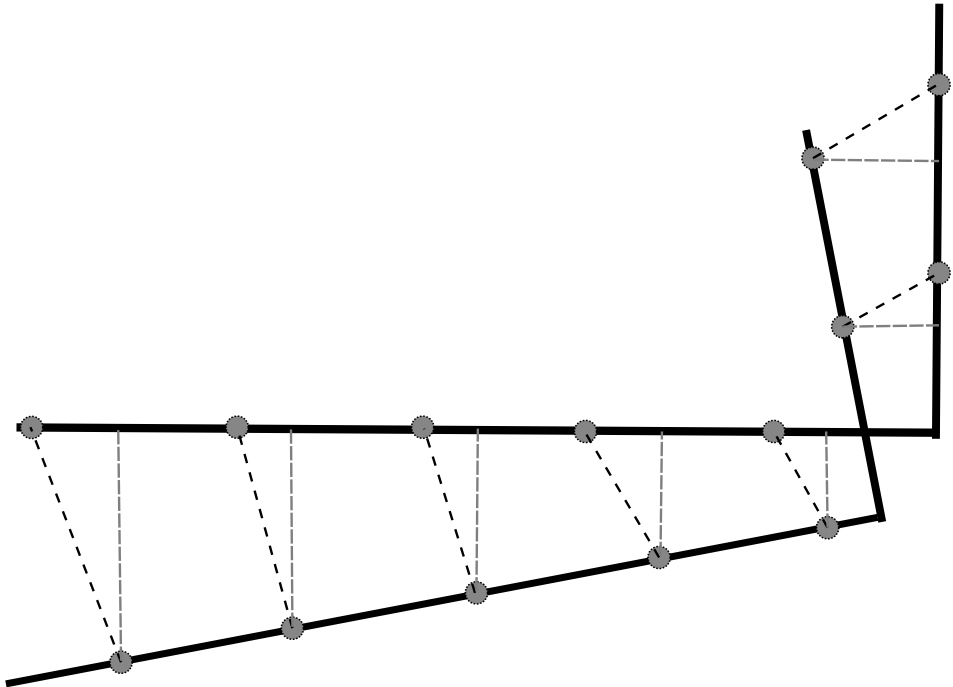


Figure 3.8: When aligning two sets of measurements of the same surface using the ICP algorithm, the result is more accurate if the point-to-plane distance is used as error function instead of the point-to-point distance, as the two sets of samples are located at different positions along the surface.

$$\epsilon = \sum_{k=1}^m (SDF_V(T_{P,V} * \dot{\mathbf{p}}_{k,P}))^2 \quad (3.15)$$

This also makes it unnecessary to calculate the surface normals. This method is much faster but requires that the SDF is sampled with a high resolution in order to get a good result. Due to the speed of this method it is possible to use more vertices than only the feature points.

3.7 Visualization

After a SSDO model has been created as described in 3.5 a rendering method is needed in order to visualize the result. Two methods have been implemented in this master thesis. The first one uses the SSDO structure directly and renders it through ray casting. The other method converts it to a triangle mesh representation and renders it using standard mesh rasterization with OpenGL.

3.7.1 Ray-casting

The sparse SDF octree model is very well suited for performing ray-casting. Ray-tracing is a rendering technique where beams of light are simulated by following them from their origin until they eventually reach an observer after having been reflected and/or transmitted on objects on the way. Ray-casting works in a similar way but instead uses rays originating from the observer that intersects each pixel in its image plane and extends out into the volume that is to be rendered. The depth at each pixel is found by stepping along each ray until a first surface is encountered. The method implemented here is based on [Jamriška] where a method to render distance fields represented in a sparse grid structure is presented.

Ray-casting is a performance demanding method but is heavily parallelizable and scales very well with increased level of detail when using a voxel representation. The SIMD-nature of the problem introduces the possibility of utilizing a GPU. This does however pose challenges concerning the representation of the structure of the octree data on device memory. It is however entirely possible to upload models of large scenes represented using a SSDO data structure to the GPU thanks to its memory efficiency. A ray-casting algorithm has been implemented that runs in parallel on multiple threads on the CPU. A GPU implementation has not been realized.

To perform ray-casting one ray is created for each pixel in a view. Each ray is defined with an origin in the viewpoint and a direction through the position of a pixel in the normalized image plane of the viewer.

The direction of a ray is given as (3.16).

$$\mathbf{r}(\mathbf{p}) \sim R^{-1}K^{-1} * \dot{\mathbf{p}} \quad (3.16)$$

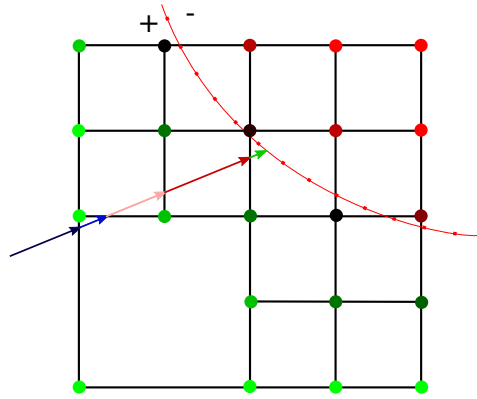


Figure 3.9: A ray cast through a section of a sparse SDF Octree. The step length along the ray depends on the size of the voxels. Surface intersections are found through interpolation of samples of the signed distance function.

where \mathbf{p} is the image coordinate of a pixel, K is the intrinsic camera matrix, and R is the orientation of the viewer.

In figure 3.9 a ray that is cast through a section of a sparse SDF octree is illustrated. The ray is traced along its direction through the voxels in the octree starting at the coarsest level from where the first child-voxel that is intersected is found. This is done through a simple comparison of the current position of the ray and the current voxel.

If the selected voxel does not contain any surface, the ray skips to the exit point of the voxel and restarts the process from the parent voxel starting from the new position. If the ray does intersect a surface inside a voxel, the ray starts stepping through its children until a leaf voxel level is reached. When a leaf voxel is found, the position inside it where the ray intersects a surface is found using the SDF samples in the corners of the voxel. If no surface intersection is found in any of the child voxels, the ray goes back to the parent voxel and tries to find an intersection point using the SDF of that parent voxel. If a surface is found, the depth is given as the distance traveled by the ray.

Surface intersections detected from voxels at coarser levels using their SDF is not very exact as they only are calculated using eight samples of the SDF. This can cause the rays to stop too early, for example close to sharp edges. In order to avoid this the ray does not stop until it has reached a surface in a leaf voxel, or until it exits the entire volume. If the ray does exit the entire volume but has found surface intersections in non-leaf voxels earlier, the intersection that was found inside the smallest voxel is selected to be the true first surface intersection for the ray.

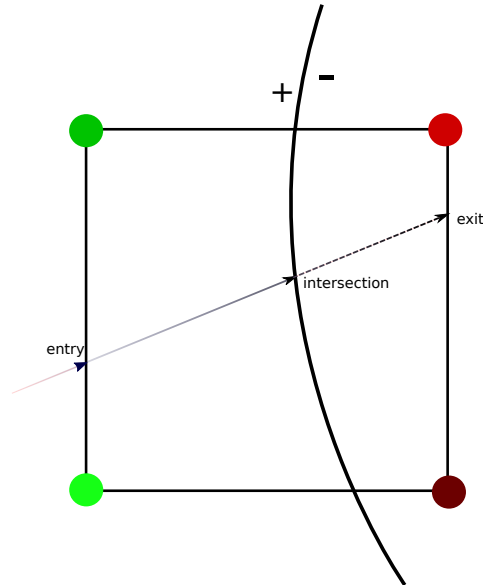


Figure 3.10: The intersection between a ray and a surface inside a voxel is found by first finding the entry and exit point of the ray through the voxel, and then finding the zero level of the signed distance function between these points using the Newton-Raphson method.

SDF Interpolation

When a ray enters a voxel we must find if it intersects with any surface inside it, and if so, locate where the intersection is. To begin with the entry and exit points of the ray are found. The SDF are then linearly interpolated in these locations. The ray is said to intersect the surface if the entry point has a positive signed distance value while the exit point has a negative signed distance value. This does not give the correct result when dealing with very curved surfaces but is a good enough estimation, as we can assume the surface to be rather flat for small enough voxels. If the ray is found to have a surface intersection we want to find the zero level of the SDF along the ray between the entry and exit points. This is done using the Newton-Raphson method over a limited set of iterations to get an approximate value.

Finding voxel-ray intersections

When performing ray casting the steps along the rays should be as large as possible while traversing through the voxels in order to gain performance. At the same time it is not beforehand possible to define a smallest step size to take in order to make sure no voxels are skipped, as the rays intersect voxels at varying angles and distances from the voxel center. So in order to take the right step length, the entry and exit points of a voxel must be found in order to calculate an appropriate step length. To simplify calculations, the starting position of the

ray is first calculated relative to the negative corner of the voxel. The six planes enclosing the voxel are then defined by:

$$\begin{aligned}x &= 0, & x &= l \\y &= 0, & y &= l \\z &= 0, & z &= l\end{aligned}$$

where l is the length of the side of the voxel. The distances to the intersections between the ray and the enclosing planes from a starting point are given as

$$\begin{aligned}t_0 &= -\frac{p_x}{d_x}, & t_1 &= \frac{l-p_x}{d_x} \\t_2 &= -\frac{p_y}{d_y}, & t_3 &= \frac{l-p_y}{d_y} \\t_4 &= -\frac{p_z}{d_z}, & t_5 &= \frac{l-p_z}{d_z}\end{aligned}\tag{3.17}$$

where \mathbf{p} is the starting point of the ray relative to the center of the voxel and \mathbf{d} is its normalized direction. The first intersection is the intersection at the smallest positive distance along the direction of the ray (3.18).

$$t_{min} = \min_{t_i > 0} t_i.\tag{3.18}$$

The intersection point is found at

$$\mathbf{p}_{intersect} = \mathbf{p} + \mathbf{d} * t_{min}\tag{3.19}$$

This method does only give the first intersection between the ray and one of the enclosing planes. If the starting point is located far outside the voxel this method often gives a point intersecting one of the planes outside of the voxel. It is however possible to test if the given intersection is inside the voxel and if not, continue the search from this intersection point.

Voxel Cone Casting

The volume that projects into one pixel using perspective projection has the shape of a cone which widens with the distance from the camera. This is illustrated in figure 3.11. Considering this, there is no reason when casting a ray to search for surfaces in levels in the octree with voxels of a size smaller than the width of this cone. Each ray is therefore assigned an angle θ_r and a width w_r that is increased by $\sin(\theta)$ millimeters for every millimeter along the ray. Voxels with children smaller than the width of a ray is considered to be at the finest level of the octree for this ray. This is also used in [Crassin et al., 2011] to accelerate the rendering of sparse voxel octrees.

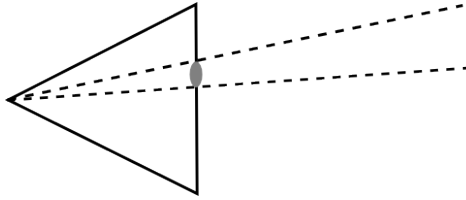


Figure 3.11: Due to the perspective projective property of pinhole cameras, the volume projected into a pixel has the shape of a cone that widens with increased distance.

3.7.2 Mesh Generation

Instead of using ray casting to render the SSDO a polygon mesh can be generated. This is done using the Marching Cubes algorithm [Lorensen and Cline, 1987]. The Marching Cubes algorithm splits the entire volume into a regular grid where each cell is parsed separately. In each cell triangles are created with corners at positions along the cube boundary based upon the value of a scalar field in the corners of the cube. The scalar field in this case is the signed distance field. As the signed distance can be found at any location within the sparse SDF octree through interpolation, the grid used for marching cubes does not have to be aligned to the voxels in the sparse SDF octree. Based upon where the zero level of the signed distance field between the corners of the cell is located, lookup tables presented in [Lorensen and Cline, 1987] are used to generate vertices that are combined into triangles. The speed of the marching cubes algorithm can be increased substantially if cells that are entirely encapsulated by a voxel with the same sign on all its SDF samples, i.e. that does not contain any surface, are skipped.

4

Result

In the following chapter the presented method of indoor mapping using the Kinect sensor is evaluated in terms of precision, performance and memory consumption. In terms of the quality of the produced model, manual visual inspections have been an important tool together with more empiric measurements. For evaluation the seminar room "Algoritmen" at the Department of Electrical Engineering at Linköping University has been used.

4.1 Model Accuracy

The major goal with this thesis is to produce a model with both accuracy and good visual appearance. One way of evaluating the model would be to measure a set of control points and to adjust these so that they align to measurements in the model. This could be done by solving the Orthogonal Procrustes Problem (presented in 2.6.2). However, due to inaccuracies in the real world measurements, this method would be inadvisable as measurement errors would accumulate when adding measurements in order to get absolute positions of the control points. Instead, a set of easily identifiable distances were measured and compared to those found in the model. Distances in the model were measured by calculating the distances between pairs of points in the model. The positions of the points in the model were measured through the ray-casting algorithm described in 3.7.1.

The precision of the real world measurements are estimated at ten millimeters. An error of the same magnitude can be expected for the measurements in the model, as the end points of each measurement are visually selected and may not match perfectly with those in the real world.

Nr.	Location	Real world (mm)	Model (mm)	Diff (mm)	%
1	Room height	2545	2525	-20	0.79
2	Room width	4640	4688	48	1.0
3	Room length	8120	8150	30	0.37
4	Wall - Whiteb.	510	519	9	1.8
5	Floor - Whiteboard	870	880	10	1.1
6	Whiteb. lamp depth	315	265	-50	16
7	Whiteb. - Proj. screen	1474	1456	-18	1.2
8	Proj. screen width	1770	1771	1	0.056
9	Roof - Door	532	538	6	1.1
10	Door width	890	884	-6	0.67
11	Door - Curtains	1615	1602	-13	0.80
12	Roof - Curtains	480	413	-67	14
13	Curtains width	2700	2693	-7	0.26
14	Coathanger height	1665	1598	-67	4.0
15	Roof - Exterior window	680	688	8	1.2
16	Window depth	330	305	-25	7.6
17	Pillar width	240	172	-68	28
18	Table width	1190	1106	-84	7.0
19	Table height	710	751	41	5.8
20	Table depth	490	410	-80	16
21	Chair width	460	415	-45	9.8

Table 4.1: Table of a set of real world and model measurements. The model used has a minimum voxel size of 3 mm and is constructed from three panoramas of 36 frames each.

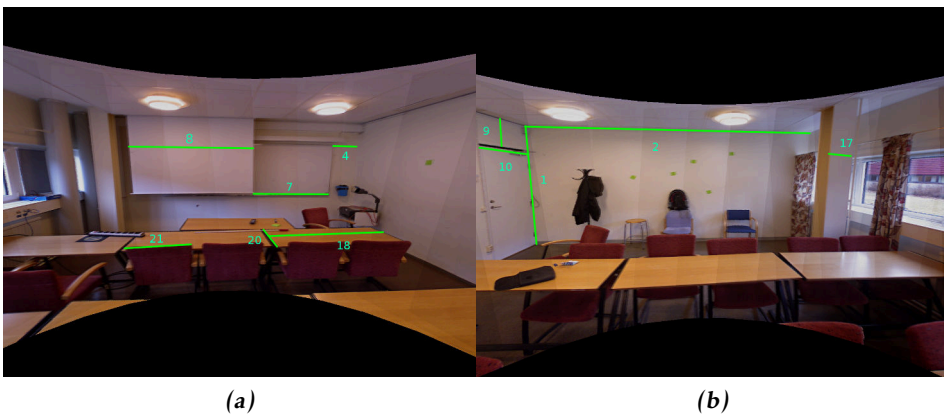


Figure 4.1: Examples of the measures presented in table 4.1, shown in two parts of a panorama.

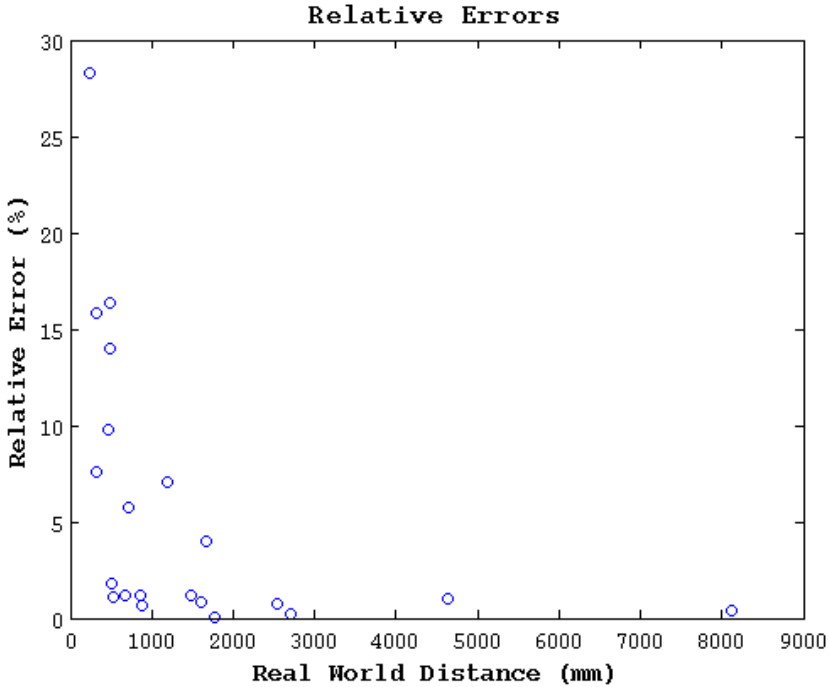


Figure 4.2: Relative model errors of the measures presented in table 4.1.

From table 4.1 the conclusion can be drawn that the global consistency of the model is good, as the dimensions of the room in the model is close to that of the actual room. Also longer distances along flat surfaces have small errors. This gives an indication that the depth values from each frame have good precision and that the estimated relative positions of the different panoramas are accurate. Examples of where the real world measurements have been made is illustrated in figure 4.1. The voxels in the octree are split a maximum of 13 times, giving a minimal voxel size of 3 mm.

Smaller objects and tight corners or partly occluded areas have larger errors. This effect can be observed clearly in figure 4.2 where the relative errors in relation to the real world measurements are plotted. A few examples of large relative errors are the width of the chairs, the height of the coat hanger and the depth of the lamp above the whiteboard. From these the conclusion that edges tends to be notched can be made. This causes objects to shrink in the model compared to their real measurements and gives larger relative errors for small structures. Longer distances, for example the room dimensions, have very small errors. Another indication of the edge issue is the quite small errors for short measurements along larger surfaces where no edges are present. One example is the width of the door. A likely cause to these inaccuracies are poorly aligned frames within the same panorama. As the weight (3.6) used when updating SDF values are larger

in front of observed surfaces than behind, surfaces further away will be favored when new frames which contradicts the current model are added. The effect of this is that poorly aligned images will carve away edges of surfaces seen by other frames. Another source of errors is that most objects are not observed from all angles as only a very limited set of viewpoints (three in this case) are used.

4.2 Memory Usage

When modeling a large room while trying to capture as much details as possible, memory consumption quickly becomes an issue. There is a limit to the amount of system memory available, and too much memory allocations and accesses can be costly in terms of performance. This is in large part what the SSDO data structure attempts to address.

The data stored for a voxel in the octree is a pointer to its parent, a pointer to a list of its children, and a byte of flags which define where inside its parent it is located, and if it is a leaf voxel or if it has children. A leaf voxel also store one floating-point signed distance value, one three-component color and a weight for each corner of the voxel. In total this makes up for 73 bytes for every voxel. When running on a 64-bit system, 80 bytes of system memory is allocated due to memory alignment.

In table 4.2 the number of voxels in the entire octree is shown for when new frames are added to the model. Here 108 frames from three different panoramas are used. The total size of the entire octree used is 24 meters, in order to ensure that the entire room can be covered no matter how the octree is positioned or oriented in the room. This is much larger than the actual size of the room but does not pose any major increase in memory usage as the large empty spaces are represented efficiently at high levels of the octree. The voxels are here split a maximum of 11 times, giving a minimal voxel size of 12 mm.

The number of voxels increase quickly for the first few frames. Later on the increase levels out, as most of the surfaces seen by the following frames already have been seen. Panoramas which are perfectly aligned and sees the same surface will detect points which forms a common surface in the model. If the alignment is bad, their surface samples will not align. This will cause multiple voxels to be generated for the same surface and thus greatly increase the amount of voxels in the octree. Due to this, the speed by which the number of voxels increase can be an indication of how good the alignment between the panoramas is. This comparison have been made to compare the different variants of the ICP algorithm described in 3.6.3.

As can be seen in table 4.2, the number of voxels decrease somewhat using any of these two methods. Compared to each other they produce almost identical results with regard to the amount of voxels. Due to the significantly higher performance cost of the RGBD-ICP algorithm shown in table 4.3, the SDF-ICP variant is preferable. In the table the number of voxels required to construct a dense SDF grid is

Frames	Sparse			Dense
	No ICP	RGBD-ICP	SDF-ICP	SDF
36	2,646,176	2,643,698	2,646,498	60,338,442
72	4,259,242	4,259,158	4,247,328	60,338,442
108	5,552,436	5,507,944	5,505,998	60,338,442

Table 4.2: Table showing the number of voxels in the model using three different methods of panorama alignment. The number of voxels in a dense SDF grid is included for comparison. The number of voxels for the first panorama vary somewhat due to the non-deterministic nature of the RANSAC algorithm.

Min. voxel size	RGBD-ICP	SDF-ICP
3 mm	870.41 s	15.64 s
12 mm	869.45 s	11.64 s

Table 4.3: Table showing the execution time of two variants of the ICP algorithm.

also included for comparison. This number is calculated based on the real-world measurements of the modeled room. It assumes that the grid is perfectly aligned to the room and does not take possible objects outside the room seen through the windows into account. If instead the same size as for the sparse SDF octree would be used, the number of voxels required for the dense SDF grid would increase to over 8 billion.

4.3 Performance

The implemented system is unable to construct a model in real-time during image capture. This was not a goal, but performance is nevertheless an important aspect. The process is entirely executed on the CPU, in contrast to some other systems where GPU:s are used. One example is [Izadi et al., 2011]. The process is however heavily parallelized in many parts and could be rewritten to utilize the SIMD-structure of a GPU. Running on an eight-core AMD FX-8120 at 3.1 GHz the process of constructing a sparse SDF octree of the previously mentioned seminar room from three panoramas using a smallest voxel size of 12 mm took 1,035 seconds. table 4.3 shows the time required to align a second panorama to a sparse SDF octree built using a first panorama for the two ICP variants presented in 3.6.3.

4.4 Visual Result

An important evaluation tool of the presented system is the visual appearance and similarity of the produced model to that of the actual environment. In Ap-

pendix A a few images of the model are shown alongside real photographs taken from the same locations, as well as some images of the model from other angles.

5

Analysis and Conclusions

In the following chapter the produced results of the system are analyzed and some areas where improvements could be made are mentioned.

5.1 Model Quality

The presented method of indoor mapping using the Kinect sensor produces resulting models of the mapped environment which have good global consistency and which are of enough visual quality to easily identify objects from the mapped environments to objects found in the models. When observing measurements of smaller objects in the scene, room for improvement can be found. Edges are often notched, most likely due to imperfect alignment between frames.

5.2 Panorama construction

When panoramas are constructed the assumption is made that the transformations between the individual frames are purely rotational. This requires the camera center of the RGB camera to be perfectly aligned to the axis of rotation. This was not done very carefully when capturing the images used for evaluation of the system. This offset was sufficiently small to construct panoramas with the assumption of pure rotation. This is however most certainly impairing the precision of the model.

The approach of capturing images in fixed positions in sets of panoramas makes this system less flexible compared to other similar systems aimed at mapping using Kinect, such as [Izadi et al., 2011] and [Henry et al., 2010]. The effect of this limitation is that objects in the mapped environments only are captured from

a very limited set of angles. This greatly limits the visual quality and detail of the produced models.

Another issue is that the presence of large uniformly colored areas close to the camera introduces errors or sometimes complete failures when aligning images into panoramas using feature matching. This is due to the lack of good distinct feature points, a problem that systems using alignment through depth data do not suffer from.

5.3 Panorama Alignment

The approach of using feature matching makes it possible to find the relative pose of sets of frames without any initial approximation, as is done in this system when aligning panoramas taken from unknown positions. Other systems often use the pose of the previous frame or a motion model as an approximation for each new frame, and thus requires frames to be captured consecutively with small relative movements. This limits their use to continuous video. When using entire panoramas the problem with a lack of image features due to large uniformly colored areas rarely arise due to the larger image area and field of view. This makes feature matching a very reliable method for this particular task. The use of the ICP algorithm further improves the accuracy of the alignment. The access to the signed surface distance through the Sparse SDF Octree model turned out to be very beneficial for its implementation and greatly increases its speed, which can be seen in the comparison between the two different ICP variants presented in 4.3.

5.4 Sparse SDF Octrees

The Sparse SDF Octree is an interesting method of representation of surface structures considering that it has the advantages of the regular dense SDF while reducing the storage requirements and thus scales better for large environments. The drawback is however the increased computational cost to construct it. The dense SDF is much easier to parse and update, and its complexity does not change. The sparse SDF octree on the other hand changes as new voxels are created for every frame that is added.

The dense grid is also very easy to store and process in parallel on a GPU. This is harder to do with a sparse octree that changes its structure dynamically. Implementations of sparse voxel octrees (SVO) have however been implemented for GPU:s in for example [Laine and Karras, 2010]. As the tree structure of sparse SDF octrees is exactly the same it should be possible to do this for these also. This would open up the possibility of real-time rendering through ray-tracing and maybe also real-time updates of the sparse SDF octree from continuous video.

5.5 Future Work

In the following chapter some suggestions of future studies are listed. These are based on ideas and experiences collected throughout the work with this thesis which were not tested.

5.5.1 Allow translation within a panorama

As it is hard to perfectly align the camera center to a fixed rotational axis, the alignment of frames within a panorama could be improved by allowing translations between the frames and thereby compensating for offsets between the camera center and the rotational axis.

5.5.2 Combine RGB and depth data for pose estimation

When mapping an environment, the poses of the images used must be estimated very accurately in order to get good accuracy in the model. Using both depth and RGB data for pose alignment should make it possible to improve the accuracy of the camera poses further, and above all, be able to produce a more robust system that can handle large uniformly colored images or inaccurate depth values.

5.5.3 Improvement of depth data using disparity maps of RGB images

The distance values from the depth sensor could be complemented with depth data calculated using pairs of RGB images from frames taken from separate locations. If their relative poses are known, pairs of RGB images could be used either to construct disparity maps from which depth values could be extracted, or to triangulate individual vertices after performing feature matching.

5.5.4 GPU implementation of a sparse SDF octree

As the process of both constructing and rendering the sparse SDF octree is parallelizable, it should be possible to greatly improve the system performance by implementing parts of the algorithms on the GPU using CUDA or OpenCL.

Appendix

A

Screenshots

In the following appendix a set of rendered views of a model constructed using the previously presented process is shown. The first three examples are shown alongside actual photographs taken from the same locations.

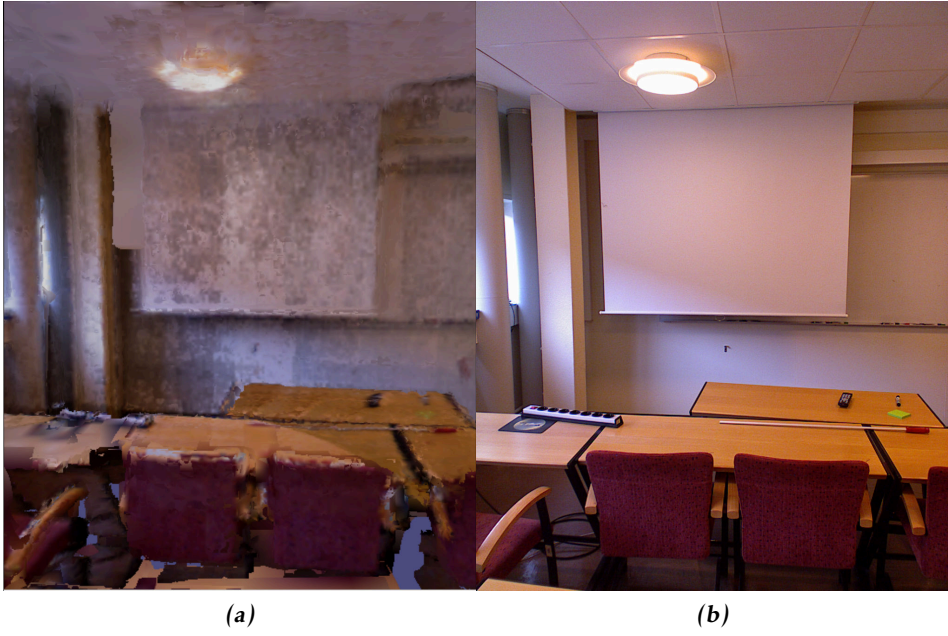


Figure A.1

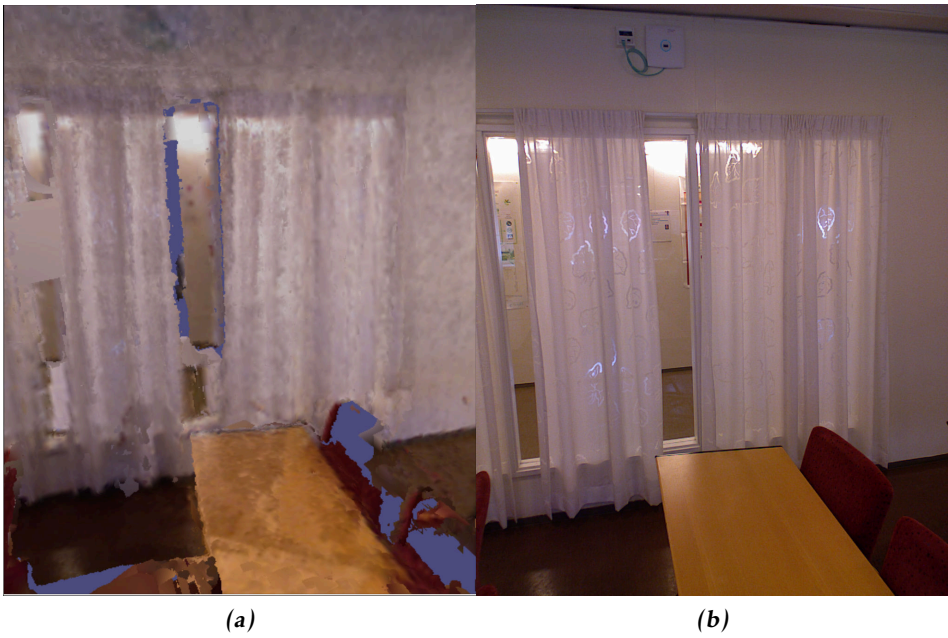


Figure A.2



(a)

(b)

Figure A.3



Figure A.4



Figure A.5



Figure A.6

Bibliography

- A. Alahi, R. Ortiz, and P. Vanderghenst. Freak: Fast retina keypoint. *2012 IEEE Conference on Computer Vision and Pattern Recognition*, 0:510–517, 2012. ISSN 1063-6919. doi: <http://doi.ieeecomputersociety.org/10.1109/CVPR.2012.6247715>. Cited on pages 2, 14, 15, and 21.
- Cyril Crassin, Fabrice Neyret, Miguel Sainz, Simon Green, and Elmar Eisemann. Interactive indirect illumination using voxel cone tracing, sep 2011. URL <http://maverick.inria.fr/Publications/2011/CNSGE11b>. Cited on page 37.
- Charles Dapogny and Pascal Frey. Computation of the signed distance function to a discrete contour on adapted triangulation. *Calcolo*, 49(3):193–219, 2012. doi: 10.1007/s10092-011-0051-z. Cited on page 12.
- Christian Kerl, Fredrik Kahl, Erik Bylow, Jürgen Sturm and Daniel Cremers. Dense localization and mapping from rgb-d data. 2013. Cited on pages 14, 25, 27, and 29.
- Leonhard Euler. Formvlae generales pro translatione quacunque corporum rigidorum. *Novi Commentarii Academiae Scientiarum Imperialis Petropolitanae*, pages 189–207, 1775. Cited on page 7.
- Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981. Cited on pages 2 and 16.
- Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments. In *In RGB-D: Advanced Reasoning with Depth Cameras Workshop in conjunction with RSS*, 2010. Cited on pages 18, 31, and 45.
- Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In *Proceedings of the*

- 24th annual ACM symposium on User interface software and technology*, UIST '11, pages 559–568, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0716-1. doi: 10.1145/2047196.2047270. URL <http://doi.acm.org/10.1145/2047196.2047270>. Cited on pages 12, 14, 25, 27, 28, 43, and 45.
- Ondrej Jamriška. Interactive ray tracing of distance fields. Cited on pages 25 and 34.
- Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. Sixth IEEE and ACM International Symposium on Mixed and Augmented Reality (ISMAR'07)*, Nara, Japan, November 2007. Cited on page 14.
- Samuli Laine and Tero Karras. Efficient sparse voxel octrees – analysis, extensions, and implementation. NVIDIA Technical Report NVR-2010-001, NVIDIA Corporation, February 2010. Cited on pages 10 and 46.
- Longuet. A computer algorithm for reconstructing a scene from two projections. *Nature*, 293:133–135, September 1981. Cited on page 30.
- William E. Lorensen and Harvey E. Cline. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph.*, 21(4):163–169, August 1987. ISSN 0097-8930. doi: 10.1145/37402.37422. URL <http://doi.acm.org/10.1145/37402.37422>. Cited on page 38.
- Stephane Magnenat. Questions regarding code and algorithms. volume [Online] OpenKinect mailing list, 2010. Cited on page 4.
- Anton Nordmark. Kinect 3d mapping. (LiTH-ISY-EX-12/4636—SE), 2012. URL <http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-85158>. Cited on pages 3 and 5.
- Daniel Ricao Canelhas. Scene representation, registration and object detection in a truncated signed distance function representation of 3d space. Master's thesis, Örebro University, School of Science and Technology, Örebro University, Sweden, 2012. Cited on pages 25 and 28.
- Edward Rosten and Tom Drummond. Fusing points and lines for high performance tracking. In *IEEE International Conference on Computer Vision*, volume 2, pages 1508–1511, October 2005. doi: 10.1109/ICCV.2005.104. URL http://edwardrosten.com/work/rosten_2005_tracking.pdf. Cited on pages 2 and 14.
- Edward Rosten and Tom Drummond. Machine learning for high-speed corner detection. In *European Conference on Computer Vision*, volume 1, pages 430–443, May 2006. doi: 10.1007/11744023_34. URL http://edwardrosten.com/work/rosten_2006_machine.pdf. Cited on pages 2 and 14.
- Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the ICP algorithm. In *Third International Conference on 3D Digital Imaging and Modeling (3DIM)*, June 2001. Cited on page 17.

- Peter H. Schönemann. A generalized solution of the orthogonal procrustes problem. *Psychometrika*, 31(1):1–10, 1966. ISSN 0033-3123. doi: 10.1007/BF02289451. Cited on page 15.
- Richard Szeliski. Image alignment and stitching: a tutorial. *Found. Trends. Comput. Graph. Vis.*, 2(1):1–104, January 2006. ISSN 1572-2740. doi: 10.1561/0600000009. URL <http://dx.doi.org/10.1561/0600000009>. Cited on page 12.
- Tinne Tuytelaars and Krystian Mikolajczyk. *Local Invariant Feature Detectors: A Survey*. Now Publishers Inc., Hanover, MA, USA, 2008. ISBN 1601981384, 9781601981387. Cited on page 14.
- B. Williams, G. Klein, and I. Reid. Real-time SLAM relocalisation. In *Proc. International Conference on Computer Vision*, 2007. Cited on page 14.



Upphovsrätt

Detta dokument hålls tillgängligt på Internet — eller dess framtida ersättare — under 25 år från publiceringsdatum under förutsättning att inga extraordinära omständigheter uppstår.

Tillgång till dokumentet innebär tillstånd för var och en att läsa, ladda ner, skriva ut enstaka kopior för enskilt bruk och att använda det oförändrat för icke-kommersiell forskning och för undervisning. Överföring av upphovsrätten vid en senare tidpunkt kan inte upphäva detta tillstånd. All annan användning av dokumentet kräver upphovsmannens medgivande. För att garantera äktheten, säkerheten och tillgängligheten finns det lösningar av teknisk och administrativ art.

Upphovsmannens ideella rätt innefattar rätt att bli nämnd som upphovsman i den omfattning som god sed kräver vid användning av dokumentet på ovan beskrivna sätt samt skydd mot att dokumentet ändras eller presenteras i sådan form eller i sådant sammanhang som är kränkande för upphovsmannens litterära eller konstnärliga anseende eller egenart.

För ytterligare information om Linköping University Electronic Press se förlagets hemsida <http://www.ep.liu.se/>

Copyright

The publishers will keep this document online on the Internet — or its possible replacement — for a period of 25 years from the date of publication barring exceptional circumstances.

The online availability of the document implies a permanent permission for anyone to read, to download, to print out single copies for his/her own use and to use it unchanged for any non-commercial research and educational purpose. Subsequent transfers of copyright cannot revoke this permission. All other uses of the document are conditional on the consent of the copyright owner. The publisher has taken technical and administrative measures to assure authenticity, security and accessibility.

According to intellectual property law the author has the right to be mentioned when his/her work is accessed as described above and to be protected against infringement.

For additional information about the Linköping University Electronic Press and its procedures for publication and for assurance of document integrity, please refer to its www home page: <http://www.ep.liu.se/>