ÖREBRO UNIVERSITY

International Master's Thesis

Scene Representation, Registration and Object Detection in a Truncated Signed Distance Function Representation of 3D Space

Daniel Ricão Canelhas
Technology

Studies from the Department of Technology at Örebro University
örebro 2012

Scene Representation, Registration and Object Detection in a Truncated Signed Distance Function Representation of 3D Space

Studies from the Department of Technology
at Örebro University

Daniel Ricão Canelhas

Scene Representation, Registration and Object Detection in a Truncated Signed Distance Function Representation of 3D Space

Supervisors:	Dr. Todor Stoyanov
	Assoc. Prof. Achim J. Lilienthal

Title: Scene Representation, Registration and Object Detection in a Truncated Signed Distance Function Representation of 3D Space

ISSN

Abstract

This thesis presents a study of the signed distance function as a three-dimensional implicit surface representation and provides a detailed overview of its different properties. A method for generating such a representation using the depth-image output from a Kinect camera is reviewed in detail. In order to improve the quality of the implicit function that can be obtained, registration of multiple sensor views is proposed and formulated as a camera pose-estimation problem.

To solve this problem, we first propose to minimize an objective function, based on the signed distance function itself. We then linearise this objective and reformulate the pose-estimation problem as a sequence of convex optimization problems. This allows us to combine multiple depth measurements into a single distance function and perform tracking using the resulting surface representation.

Having these components well defined and implemented in a multi-threaded fashion, we tackle the problem of object detection. This is done by applying the same pose-estimation procedure to a 3D object template, at several locations, in an environment reconstructed using the aforementioned surface representation.

We then present results for localization, mapping and object detection. Experiments on a well-known benchmark indicate that our method for localization performs very well, and is comparable both in terms of speed and error to similar algorithms that are widely used today. The quality of our surface reconstruction is close to the state of the art. Furthermore, we show an experimental set-up, in which the location of a known object is successfully determined within an environment, by means of registration.

Acknowledgements

I wish to thank my supervisors for their support and guidance, not only along the way, but also in granting me freedom to choose the topic of my thesis in accordance to my curiosity. I also wish to express my gratitude to Richard Newcombe, Steven Lovegrove and Jürgen Sturm for their kindness and patience in all our discussions. A big thank you is owed to Alstom Grid for looking after my best interests as their employee and supporting my career choices without restrictions. Lastly, I wish to thank family and friends for their joyful contributions to daily life and Susanne, for everything.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

1.1 Registration and SLAM

The problem of building a consistent representation of an environment, from sensor data, is one that relies heavily on an accurate estimate of the sensor's position relative to that environment for success. Often the inverse is also true, defining what is known as *Simultaneous Localization and Mapping* (SLAM). This is one of the most fundamental problems in mobile robotics and has received much attention since its formulation at the 1986 IEEE Robotics and Automation Conference [9].

It is largely held that SLAM is still an open problem, in spite of an abundance of algorithms that appear to perform exceptionally well at it. This claim is chiefly motivated by the restrictions that have to be imposed on either what is implied by "localization" or the significance of "mapping". Naturally, neither map nor position can be encoded to perfect detail in a truly unlimited sense, but even for the small confines of an apartment, it would be difficult to account for dynamic and static elements of the environment from a single observation without knowledge of objects and their properties.

Another claim on the openness of the SLAM problem may stem from the expectations on what sort of information a map is to provide about the environment. For example, it is often the case that a map is not an end in itself, but only a tool to be used in a subsequent step. This subsequent step might require only to know which parts of the map are occupied, as is often the case in navigation. In some cases, however, the map is expected to provide much more information, such as which room is most likely to contain a particular object. In the first case, many solutions exist. In the other; very few, if any.

In this work we pose the creation of maps, as a camera-tracking problem, using a Kinect camera and approach the problem of object detection by means of registration. By revisiting the problem of SLAM and attempting to extend it with object detection, we aim to take a step towards semantic maps, 3D model synthesis of real-world objects or performing 3D mapping in robotic applications.

Until recently, much of the research in camera tracking has focused on either summarizing the available data from sensors, or discarding all but the most salient features in order to estimate the camera's position. During the past couple of years, we have seen the advent of dense methods that make use of all the available data both for representing the world, as well as solving the camera pose estimation problem under real-time constraints. The feasibility of such methods is owed to some degree to the lower cost and availability of more powerful graphics hardware and multi-core CPUs, but also to innovations in parallel algorithms.

1.2 Related Work

This work bears many similarities to that of Newcombe et al. in KinectFusion [25] in the methods used for generating a 3D signed distance function representation of space from depth-images. It differs in that our implementation follows an alternative approach to solving the implied registration problem and in that our objective is ultimately an attempt at object recognition. Our method for registration is more akin to one presented as Fast ICP [8], which computes a distance transform of one set of points to aid in the alignment of another. For sake of speed and in order to facilitate the inclusion of new data as we build our map, we choose to avoid a full distance transform. Registration, using a distance function representation of objects has also been extended to deal with non-rigid template matching by Fujiwara et al. [15], which is related in light of research directions we wish to explore in our future work.

Although SLAM is not our main motivation, the method outlined in this work can be used for such purposes too. The 3D representation of our choice for this work is a variant on the Signed Distance Function, as proposed by Curless and Levoy [7].

1.3 Outline

The remainder of this thesis is organized as follows:

Chapter 1 is a short introduction to SLAM and a very brief overview of related work. It also contains **this** Outline.

Chapter 2 gives a thorough definition of Signed Distance Functions in the context of implicit three dimensional surface representation. Some noteworthy properties are explained in detail, as well as a method for how such an implicit surface representation can be generated from depth data.

Chapter 3 presents a method for estimating the 6 DoF camera pose within an intuitive and effective convex optimization framework. Practical considerations are made and notes about implementation are also included

Chapter 4 proposes a simple method for registration-based object detection, giving a motivation and a direction for more complex algorithms

Chapter 5 shows experimental results and benchmarks

Chapter 6 offers our conclusions

Chapter 2

Signed Distance Functions

The Signed Distance Function (SDF), also referred to as the Signed Distance Transform, or simply Distance Transform has been widely applied to the processing or visualization of volumetric 3D data. Commonly used in the field of computer graphics as an acceleration structure for speeding up ray-casting operations [16] it can also be used as a 3D model representation. Other applications include collision detection [14] and haptic feedback [23], among others [19][38].

The SDF is usually implemented as a voxel-based (or pixel-based in the two-dimensional case) representation, in which each cell contains the distance to the nearest surface in the scene. The signed part indicates whether the voxel (or pixel) is on the outside (positive) or inside (negative) an object.

2.1 Definition

We will now elaborate on the above definition. Let there be a function

$$D(\boldsymbol{x}) : \mathbb{R}^N \to \mathbb{R},$$

mapping positions in N-dimensional space to scalar values. For our purposes we will most often be interested in $N = 3$, but in some cases we will make analogies in $N = 2$, for illustrative purposes. Consider, for example, the circle defined by

$$r = 2,$$
$$x^2 + y^2 = r^2, \tag{2.1}$$

or, to use a notation more consistent with our function definition,

$$\|\boldsymbol{x}\|_2^2 = r^2, \tag{2.2}$$

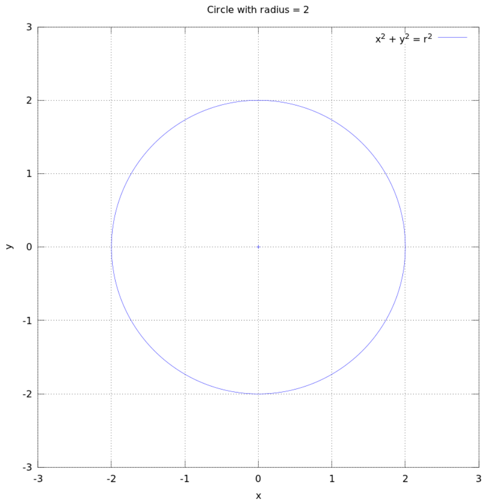


Figure 2.1: This figure shows a plot of all the points with distance equal to r from the origin (r = 2)

equivalent to,

$$\|\boldsymbol{x}\|_2 - r = 0, \tag{2.3}$$

with $\boldsymbol{x} = \begin{bmatrix} x \\ y \end{bmatrix}$ and $\|\cdot\|_2$ signifying the L2-norm (Euclidean distance).

This equation can be expressed, verbally, as the geometric locus in $\mathbb{R}^2$ that has a distance of r units from the origin, r being the circle's radius. Figure 2.1 shows a plot of the points that satisfy equation 2.3.

However, let us focus on the first term of the equation and plot the value of $\|\boldsymbol{x}\|_2$ as a function of x and y. The result, as can be seen in Fig. 2.2, is a smoothly varying gradient that becomes lighter (higher-valued) further away from the origin, in every direction.

If we now subtract r from $\|\boldsymbol{x}\|_2$, as in 2.3 and plot the resulting values, we have the result seen in Fig. 2.3. Now the distance is relative to the edge of the circumference. It is a positive value whenever outside, negative whenever inside and zero exactly on the edge of the circle. This means that the common definition for a circle conforms precisely to the definition of a signed distance function. The intuition behind this is that the equation for a circle can be thought of as the intersection between a plane and a cone standing on its tip. Everything below the plane is negative and above, positive.

In fact, expressions exist for several primitive geometries, also in higher dimensions [26][13]. Furthermore, composing several such geometric shapes together is very easily achieved using min and max operations on the outputs

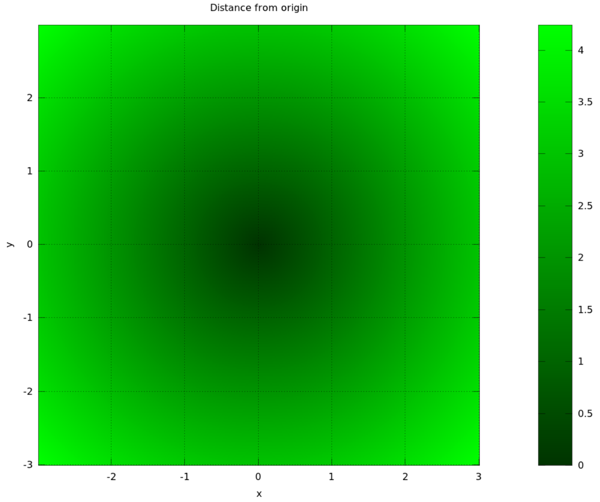


Figure 2.2: Points in 2D, plotted with brightness proportional to the L2-norm of vector that points to them

of their respective distance functions (SDF's are closed under min and max operations).

In the general case, and especially when dealing with sensor data, we do not have an explicit formulation for the SDF, needing to estimate it from discrete samples in some N-dimensional metric space.

2.2 Properties

In this section we will describe some properties of the SDF that will be of use to us later.

2.2.1 Intersection test

Since the SDF represents a surface, it is desirable to have a test to determine where a given ray intersects this surface. This is necessary for rendering images of the representation (including depth-images). It is also useful for sampling points on the surface without resorting to a sweep through the entire voxel-space. Therefore, given a ray,

$$\alpha\boldsymbol{\rho} = \alpha \begin{bmatrix} \rho_1 \\ \rho_2 \\ \rho_3 \end{bmatrix}$$

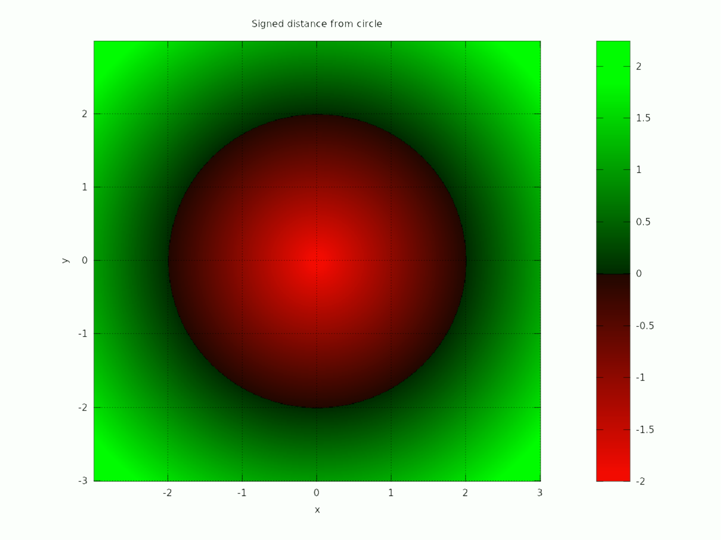


Figure 2.3: Points in 2D, plotted with brightness proportional to the L2-norm of vector that points to them, minus the radius

where $\boldsymbol{\rho}$ is a unit-norm vector and α a scalar. We wish to find a scalar value $\alpha^\star$ such that $D(\alpha^\star \boldsymbol{r}) = 0$. There is a simple and elegant procedure called sphere-tracing that does exactly this. Algorithm 1 has some similarities to Newton's Method for successive approximation of roots (Secant Method). In essence it iteratively rescales a ray by adding the current value of the SDF to α.

Algorithm 1 Sphere tracing.

```
Require: $\rho$
$\alpha_0 \leftarrow 0$
for $k = 1$ to $k_{max}$ do
    $\alpha_k \leftarrow \alpha_{k-1} + D(\alpha_{k-1} \cdot \rho)$
    if (
    $D(\alpha_{k-1} \cdot \rho) > 0$ and $D(\alpha_k \cdot \rho) \leq 0)$ or
    $(D(\alpha_{k-1} \cdot \rho) < 0$ and $D(\alpha_k \cdot \rho) \geq 0))$ then
        return $\alpha_{k-1} - \frac{D(\alpha_{k-1} \cdot \rho)}{D(\alpha_k \cdot \rho) - D(\alpha_{k-1} \cdot \rho)} \cdot [\alpha_k - \alpha_{k-1}]$
    end if
end for
return $+\inf$.
```

The algorithm terminates after a maximum number of steps or if the function has been evaluated at two successive points with opposite signs. This means that the ray has stepped across the surface (either from the inside, going out or the outside, going in). The returned scaling of the ray that represents the intersection point is obtained by linear interpolation (step 5, Algorithm 1). If the maximum number of steps was reached, $+\inf$ is returned, indicating that the ray didn't intersect a surface in the given number of steps.

Since the SDF is defined as the distance to the nearest surface, each step along the ray can be thought of as moving to the edge of the largest sphere that fits at the current point in space. An illustration of the algorithm is given in Fig. 2.4. When searching in this way, for a surface, it is practical to have an early stopping condition at some smallest allowed step-size is set. This early stopping (at a positive value) can speed up rendering, but can also be used to **dilate** objects, making them appear arbitrarily thicker. Conversely, late-stopping can be used to make objects thinner.

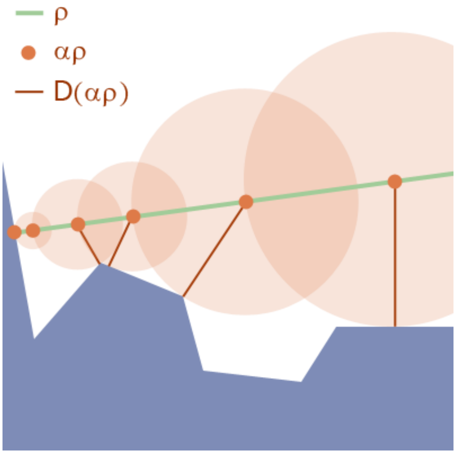


Figure 2.4: Sphere tracing. The dots represent the points at which the function $D(\boldsymbol{x})$ is evaluated and the maroon lines represent the scalar value returned by the function. Note that the information of where the nearest surface is located is not available, only the distance to it.

2.2.2 Normal vector estimation

Having a method for computing the intersection is enough to generate a depth-image or point-cloud from any given viewpoint in the SDF. To safely grasp an object with a manipulator, or to perform object recognition based on 3D features, the orientation of the surface is often needed. For an SDF with an analytical expression, the components of the normal vector are simply obtained by partial derivatives of the SDF with respect to each spatial dimension[1].

$$\boldsymbol{n}(\boldsymbol{x}) = \nabla_{\boldsymbol{x}} D(\boldsymbol{x})^T = \begin{bmatrix} \partial D(\boldsymbol{x})/\partial x_1 \\ \partial D(\boldsymbol{x})/\partial x_2 \\ \partial D(\boldsymbol{x})/\partial x_3 \end{bmatrix} \tag{2.4}$$

For our circle example we have:

$$\frac{\partial}{\partial x_1} \|\boldsymbol{x}\|_2 - r = \frac{x_1}{\|\boldsymbol{x}\|_2},$$

$$\frac{\partial}{\partial x_2} \|\boldsymbol{x}\|_2 - r = \frac{x_2}{\|\boldsymbol{x}\|_2},$$

resulting in

$$\boldsymbol{n} = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \cdot \frac{1}{\|\boldsymbol{x}\|_2}.$$

We note here that there is no restriction on the domain of $\nabla_{\boldsymbol{x}} D(\boldsymbol{x})$, i.e., the gradient is not only defined at points pertaining to surfaces, but can be computed wherever $D(\boldsymbol{x})$ is defined.

As mentioned before, an explicit expression like the one given for the circle will often not be available. In such cases the gradient vector can be found by finite differences. For our voxel-based SDF we will use central differences to compute gradients.

2.2.3 Inside - Outside test

A test to see if a given point is inside or outside an object can be made simply by evaluating the sign of the SDF. A function for an inside-outside test can be defined as the following:

$$Inside(\boldsymbol{x}) = \begin{cases} 1 & D(\boldsymbol{x}) < 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

2.2.4 Curvature

If the first derivative of the SDF, with respect to position, produces the gradient toward the surface, then the second derivative (a measure of how this gradient changes with position) is the *curvature.* More precisely,

$$Curv(\boldsymbol{x}) = \nabla^2_{\boldsymbol{x}\boldsymbol{x}} D(\boldsymbol{x}). \tag{2.6}$$

[1] We adopt the convention that the derivatives, relative to to each dimension, are stored in separate columns, resulting in a row-vector (hence the transpose).

The second-order derivative of a vector field, also known as the Laplacian. In 3D, it can be computed as

$$\nabla^2_{\boldsymbol{x}\boldsymbol{x}} D(\boldsymbol{x}) = \frac{\partial^2 D(\boldsymbol{x})}{\partial x_1^2} + \frac{\partial^2 D(\boldsymbol{x})}{\partial x_2^2} + \frac{\partial^2 D(\boldsymbol{x})}{\partial x_3^2}. \tag{2.7}$$

In practice, our finite-difference approach to evaluating gradients requires several memory look-ups. Derivatives (especially of higher orders) therefore tend to be costly to compute. An approximation that gives an indication of curvature, and also has the benefit of being normalized in the range $[0, 1]$ is the projection of adjacent gradient vectors onto each-other. So instead we have, for the curvature along dimension x_i,

$$\nabla^2_{\boldsymbol{x},i} D(\boldsymbol{x}) \approx 1 - \frac{[\boldsymbol{n}(\boldsymbol{x} + \boldsymbol{d}_i)]^T \boldsymbol{n}(\boldsymbol{x})}{\|\boldsymbol{n}(\boldsymbol{x} + \boldsymbol{d}_i)\|_2 \cdot \|\boldsymbol{n}(\boldsymbol{x})\|_2}, \tag{2.8}$$

where $\boldsymbol{n}(\boldsymbol{x})$ is the surface normal at $\boldsymbol{x}$. The vector $\boldsymbol{d}_i$ is simply a displacement consisting of zeros in all but the component denoted by the subscript, the latter being a small positive value instead. The normalized dot product is equal to the cosine of the angle between the vectors. So the result of this operation is that if two nearby points in space have gradients oriented in different directions, the measure of curvature will be high (closer to one). If the adjacent gradients instead have the same orientation, the curvature will be closer to zero. Computing the curvature for each dimension and storing the result as the entries in a 3-element vector, produces a vector that indicates the direction of "bending" at that point in space. The divisor in the above expression can be omitted if it can be ensured that the norm of the gradient is always equal to one. In practice, this is not always the case.

2.3 Signed Distance Functions From Depth Images

Now that we have defined signed distance functions and some of their properties, we shall continue with a method that can be employed for computing a SDF from depth images. This method is proposed in [7] and is also employed in [25] with some minor differences.

2.3.1 Input data

The input data is a depth image (also called depth-map). A depth image is very much like an ordinary gray-scale image in the sense that it is a two-dimensional array of elements, i.e., pixels or picture elements. Each pixel in a depth image stores a numeric value that either directly or through some conversion corresponds to the distance at which a surface was measured along

Figure 2.5: Microsoft Kinect camera

a ray passing through that particular pixel. These rays are usually arranged so that they originate from a common point, called the sensor origin. The diagram in Fig. 2.6 shows a drawing, explaining the relationship between depth images and the measured space.

Although we say that measurements are made along rays (in green), it is customary to re-scale these range values by the cosine of the angle that these rays make with the view axis. This produces what is called 'z-depth' which, in the drawing, is shown as being the points along the view axis where it is intersected by the vertical blue lines. By convention, depth images denote images with measurements along the viewing axis. Images that contain measurements along rays are usually referred to as 'range-images'. Throughout this work we shall only be concerned with depth images, however.

Note that to recover the original three-dimensional coordinate of the surface measurement, from a depth image, the focal length of the sensor must be known.

In this work, input data are obtained from a Kinect [24] camera, as depicted in Fig. 2.5 though any other device (e.g. [27][2]) that outputs depth images could be used just as well. In fact, for the purposes of the algorithms in this work, sensor systems, such as time-of-flight (ToF) cameras or actuated laser range-finders can potentially be used, with similar expected performance. It has been shown by Stoyanov et al. that Kinect-like sensors, time of flight cameras and actuated Laser Range finders are comparable in terms of error under certain conditions [34]. The main requirement for the success of the method we propose in this thesis is that depth image data is provided from viewpoints with a sufficient amount of overlap.

The Kinect camera is a type of sensor known as a structured (or coded) light sensor. The measurement principle is (very simply stated) based on projecting a light pattern whose appearance, when viewed by a camera, can be related to a distance. As an example, in Fig. 2.7 a depth and color image of an office

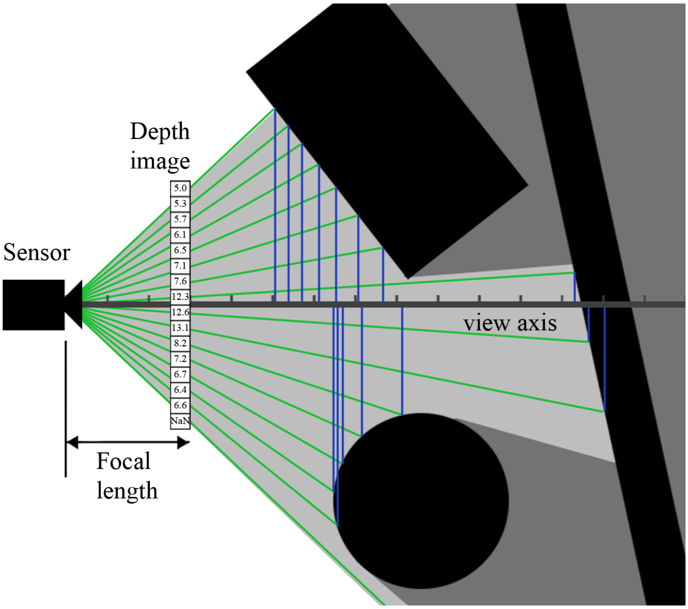
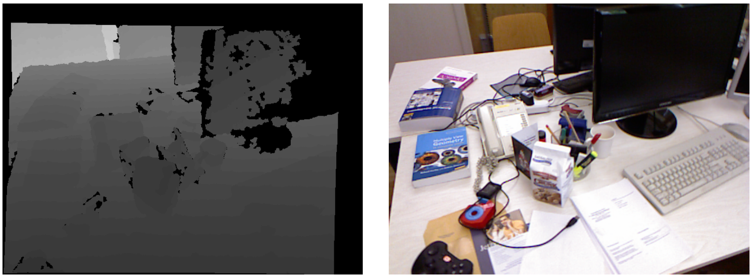


Figure 2.6: Diagram showing the relationship between a depth image and the surfaces at which measurements are taken, under a pinhole camera model



(a) depth image

(b) color image

Figure 2.7: Depth and color images of the same office desk

desk are shown. The images are part of a publicly available dataset [35]. The depth image has been contrast-adjusted to improve visualization.

2.3.2 Notation

To introduce some useful notation and formalize the previous statements, we will consider an image as a subset $\boldsymbol{M}$ of the two-dimensional plane $i.e, \boldsymbol{M} \subset \mathbb{R}^2$.

- Let there be a scalar function z_n that assigns a 'z-depth' value to each element of $\boldsymbol{M}$ for every n^{th} image. Since measurements cannot be obtained from behind or exactly at the sensor origin, z_n is a strictly positive function.
$$z_n : \boldsymbol{M} \to \mathbb{R}_+$$
- Let $\boldsymbol{s}_n$ be a (vector-valued) function that defines a 3D surface point associated with each element on the image plane and its depth, when given the n^{th} depth image, or formally
$$\boldsymbol{s}_n : \mathbb{R}^2 \times \mathbb{R} \to \mathbb{R}^3,$$
$$\boldsymbol{s}_n(\boldsymbol{m}) = \begin{bmatrix} \frac{m_1 - c_x}{f_x} z_n(\boldsymbol{m}) \\ \frac{m_2 - c_y}{f_y} z_n(\boldsymbol{m}) \\ z_n(\boldsymbol{m}) \end{bmatrix}, \tag{2.9}$$
where $\boldsymbol{m} = (m_1, m_2) \in \boldsymbol{M}$, and $c_x, c_y, f_x, f_y \in \mathbb{R}$ are intrinsic camera parameters of a pinhole camera model. Here c_x and c_y are the coordinates in $\boldsymbol{M}$ where the view-axis crosses the image-plane. The parameters f_x and f_y are the horizontal and vertical focal lengths (see Fig. 2.6). Usually, c_x and c_y are half the number of columns and rows of $\boldsymbol{M}$, respectively. Furthermore, if the pixels are square, we also have $f_x = f_y$.

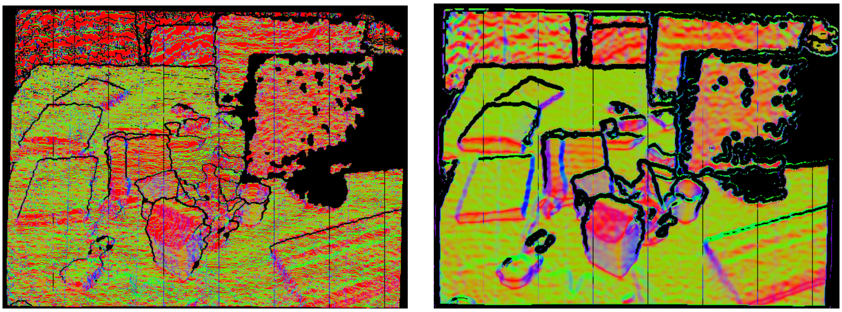
  In addition (for convenience) we define $\bar{\boldsymbol{s}}_n(\boldsymbol{m})$ as a homogeneous coordinate vector
$$\bar{\boldsymbol{s}}_n(\boldsymbol{m}) = \begin{bmatrix} \boldsymbol{s}_n(\boldsymbol{m}) \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{m_1 - c_x}{f_x} z_n(\boldsymbol{m}) \\ \frac{m_2 - c_y}{f_y} z_n(\boldsymbol{m}) \\ z_n(\boldsymbol{m}) \\ 1 \end{bmatrix}. \tag{2.10}$$
- Let $\boldsymbol{\pi}$ be a (vector-valued) function that perspective-projects 3D points to the image plane, or formally
$$\boldsymbol{\pi} : \mathbb{R}^3 \to \mathbb{R}^2,$$
$$\boldsymbol{\pi}(\boldsymbol{x}) = \begin{bmatrix} \frac{x_1}{x_3} f_x + c_x \\ \frac{x_2}{x_3} f_y + c_y \end{bmatrix}, \tag{2.11}$$
where $\boldsymbol{x} = [x_1, x_2, x_3]^T \in \mathbb{R}^3$. Note that $\boldsymbol{m} = \boldsymbol{\pi}(\boldsymbol{s}_n(\boldsymbol{m}))$.

Normal vector estimation from input data

Given the above notation, we can formally present the standard method for estimating normal vectors at surface points, directly from an input depth image. Although we have a method for computing normal vectors from signed distance functions, we shall see that estimating normal vectors from the raw sensor data can be a useful step in generating the SDF itself. The estimation of surface normals here is based on the assumption that each surface point is located within a locally planar region. We can obtain two vectors that span the plane by evaluating the difference between $\boldsymbol{s}_n(\boldsymbol{m})$ and two surface points adjacent to it. The estimated normal vector $\hat{\boldsymbol{n}}_n(\boldsymbol{m})$ is the cross product of these vectors:

$$\hat{\boldsymbol{n}}_n(\boldsymbol{m}) = (\boldsymbol{s}_n(m_1+1, m_2) - \boldsymbol{s}_n(\boldsymbol{m})) \times (\boldsymbol{s}(m_1, m_2+1) - \boldsymbol{s}_n(\boldsymbol{m})) \tag{2.12}$$

To reduce the amount of noise present in the final estimate, the surface normals can be computed, instead, from a bilateral-filtered [36] version of the depth image, as proposed by [25]. Bilateral filtering is an edge-preserving blurring operation.



(a) Estimated surface normals

(b) Estimated surface normals from a smoothed depth image

Figure 2.8: The cross product between vectors pointing to adjacent pixels produces an estimate for the normal vector at each surface point. In the images, the three spatial dimensions (x,y,z) of the vectors have been mapped to red green and blue, respectively, to give a qualitative indication of the estimated surface orientation. Note that there is noise in the result, even with smoothing. The vertical bands are an artefact of the sensor output.

2.3.3 Creating the SDF

The goal of this section is to define an implicit signed distance function based on sensor-data. Let us consider this a function that maps points in three-dimensional space to scalar values,

$$D(\boldsymbol{x}) : \mathbb{R}^3 \rightarrow \mathbb{R},$$

and, for later convenience,

$$\bar{D}(\bar{\boldsymbol{x}}) : \mathbb{R}^4 \rightarrow \mathbb{R},$$

$$\bar{D}(\bar{\boldsymbol{x}}) := D(\boldsymbol{x}) \tag{2.13}$$

The second definition merely states that when $\bar{D}(\bar{\boldsymbol{x}})$ is evaluated with a vector represented in homogeneous coordinates, it discards the last element of the input and returns the result of $D(\boldsymbol{x})$.

To create a SDF from a depth image, we first initialize a discrete voxel grid of extension $size_x \times size_y \times size_z \in \mathbb{N}_+$ with a spatial resolution of τ. This to say that each voxel represents a cube in space, each cube measuring τ meters in length, width and height.

We shall once again rely on two-dimensional analogies for visualization purposes. In Fig. 2.9 we illustrate a newly initialized voxel grid, next to the depth image from the previous example. Note that the focal length of the sensor, in this example, is three units and τ is assumed to be one unit.

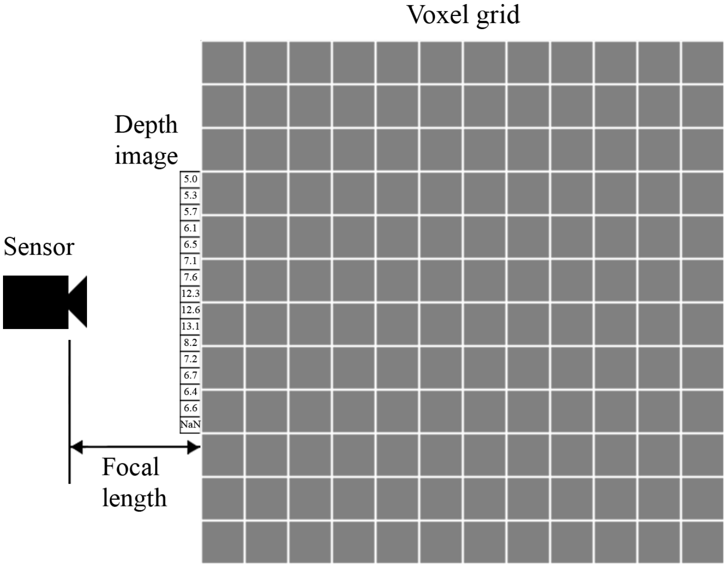


Figure 2.9: The voxel grid is initialized over the 3D space from where measurements were obtained.

The spatial coordinates to the center of each voxel are then perspective projected, using $\boldsymbol{\pi}$, into the image plane (see Fig. 2.10). For all coordinates whose projection falls within the subset of $\mathbb{R}^2$ occupied by $\boldsymbol{M}$, we look up the value $z_n(\boldsymbol{m})$ stored in the nearest pixel. We then compute the difference between this value and the *distance to the voxel* (measured along the view axis). The resulting difference is stored in the voxel itself. See Fig. 2.11 for a graphical example. Formally, our definition of an element $D(\boldsymbol{d})$ is,

$$D(\boldsymbol{d}) = z_n(\boldsymbol{\pi}(\boldsymbol{d})) - d_3 \tag{2.14}$$

where $\boldsymbol{d} = [d_1, d_2, d_3]^T \in \mathbb{N}_0^3$ (i.e., the natural numbers, including zero) are the integer coordinates of each voxel ranging from zero to $size_x$,$size_y$,$size_z$.

Evidently, this produces values that are positive, zero or negative depending on whether the center of the voxel is outside, at or behind surfaces, respectively. Voxels whose perspective projection falls into pixels with bad measurement data or outside the depth image are not updated. In the illustrations, these cases are labelled with question marks. Lastly, the values stored in the grid are not in strict adherence to the definition of a true signed distance function, i.e., that the value represents the (Euclidean) distance to the nearest surface. This is because distances are computed along the line of sight. Euclidean metrics are therefore only produced towards surfaces that are exactly perpendicular to the viewing angle. However, by incorporating measurements from several viewpoints we can still construct a function that decreases monotonically towards surfaces, at which the value of the function is zero.

By interpolating between voxels and drawing a surface at the boundary between positive and negative values, we get the result seen in Fig. 2.12. Compared to the arrangement of (2D) objects in Fig. 2.6 we note that much of the resulting surface is false.

To avoid this, we truncate the values that can be written into the voxel grid at a small positive value D_{max} and a small negative value D_{min}. If we would consider, for instance D_{max} and D_{min} to be $+1$ and -1 we would get the surfaces seen in Fig. 2.13. This has the added benefit of allowing local changes without the need for updating distance values in remote voxels. Throughout the remainder of this thesis we will refer to the signed distance function (SDF), meaning this truncated version.

It is important to mention that D_{max} and D_{min} need **not** be symmetric around zero. It might be desirable, for instance, to have a large value for D_{max} for collision avoidance or to speed up rendering, by allowing the sphere-tracing algorithm to make larger jumps. The value of D_{min}, on the other hand, is often desired to be as small as possible, since it will determine the minimum thickness of objects that can be reconstructed.

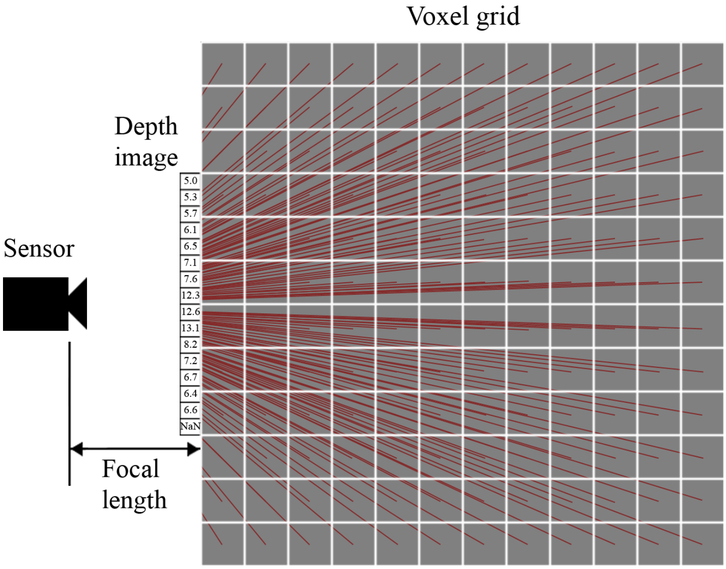
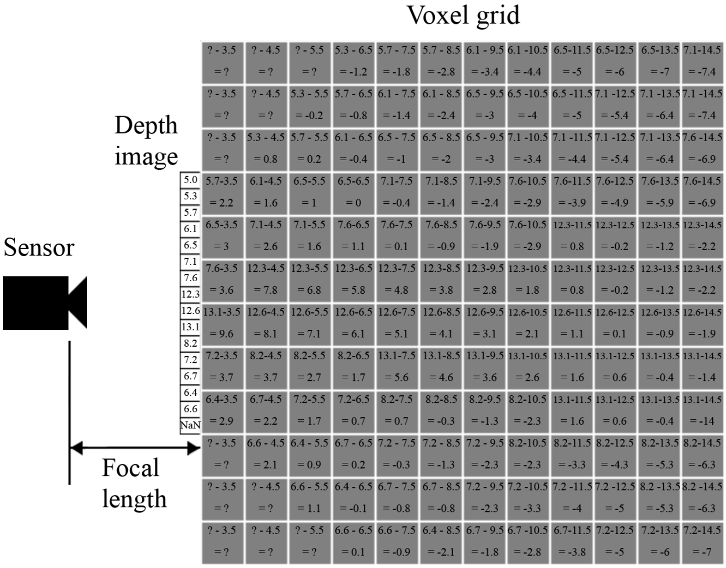
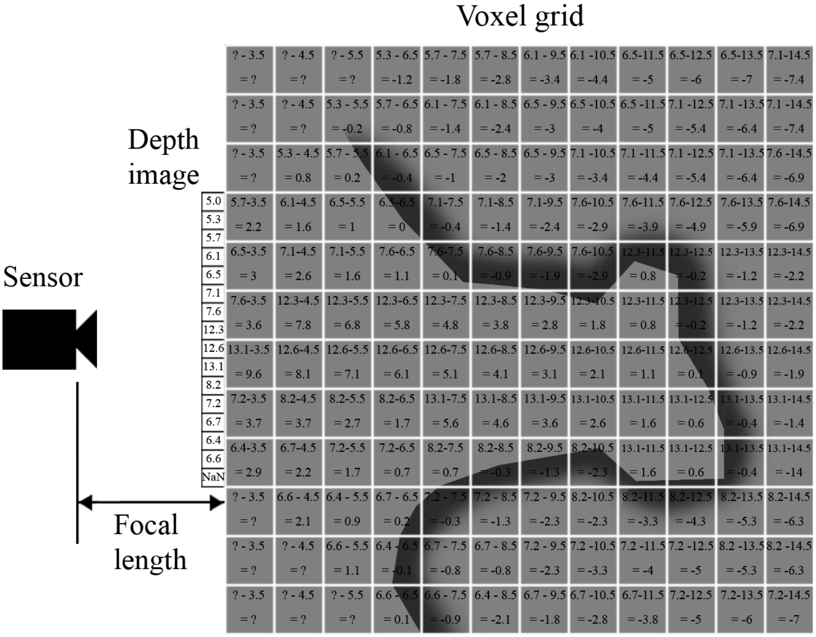


Figure 2.10: The coordinates of each voxel center is projected into the image plane. Note that not every projection will fall within the subset of the plane where the depth image is defined.



? - 3.5 = ?	? - 4.5 = ?	? - 5.5 = ?	5.3 - 6.5 = -1.2	5.7 - 7.5 = -1.8	5.7 - 8.5 = -2.8	6.1 - 9.5 = -3.4	6.1 -10.5 = -4.4	6.5-11.5 = -5	6.5-12.5 = -6	6.5-13.5 = -7	7.1-14.5 = -7.4
? - 3.5 = ?	? - 4.5 = ?	5.3 - 5.5 = -0.2	5.7 - 6.5 = -0.8	6.1 - 7.5 = -1.4	6.1 - 8.5 = -2.4	6.5 - 9.5 = -3	6.5 -10.5 = -4	6.5 -11.5 = -5	7.1 -12.5 = -5.4	7.1 -13.5 = -6.4	7.1 -14.5 = -7.4
? - 3.5 = ?	5.3 - 4.5 = 0.8	5.7 - 5.5 = 0.2	6.1 - 6.5 = -0.4	6.5 - 7.5 = -1	6.5 - 8.5 = -2	6.5 - 9.5 = -3	7.1 -10.5 = -3.4	7.1 -11.5 = -4.4	7.1 -12.5 = -5.4	7.1 -13.5 = -6.4	7.6 -14.5 = -6.9
5.7-3.5 = 2.2	6.1-4.5 = 1.6	6.5-5.5 = 1	6.5-6.5 = 0	7.1-7.5 = -0.4	7.1-8.5 = -1.4	7.1-9.5 = -2.4	7.6-10.5 = -2.9	7.6-11.5 = -3.9	7.6-12.5 = -4.9	7.6-13.5 = -5.9	7.6-14.5 = -6.9
6.5-3.5 = 3	7.1-4.5 = 2.6	7.1-5.5 = 1.6	7.6-6.5 = 1.1	7.6-7.5 = 0.1	7.6-8.5 = -0.9	7.6-9.5 = -1.9	7.6-10.5 = -2.9	12.3-11.5 = 0.8	12.3-12.5 = -0.2	12.3-13.5 = -1.2	12.3-14.5 = -2.2
7.6-3.5 = 3.6	12.3-4.5 = 7.8	12.3-5.5 = 6.8	12.3-6.5 = 5.8	12.3-7.5 = 4.8	12.3-8.5 = 3.8	12.3-9.5 = 2.8	12.3-10.5 = 1.8	12.3-11.5 = 0.8	12.3-12.5 = -0.2	12.3-13.5 = -1.2	12.3-14.5 = -2.2
13.1-3.5 = 9.6	12.6-4.5 = 8.1	12.6-5.5 = 7.1	12.6-6.5 = 6.1	12.6-7.5 = 5.1	12.6-8.5 = 4.1	12.6-9.5 = 3.1	12.6-10.5 = 2.1	12.6-11.5 = 1.1	12.6-12.5 = 0.1	12.6-13.5 = -0.9	12.6-14.5 = -1.9
7.2-3.5 = 3.7	8.2-4.5 = 3.7	8.2-5.5 = 2.7	8.2-6.5 = 1.7	13.1-7.5 = 5.6	13.1-8.5 = 4.6	13.1-9.5 = 3.6	13.1-10.5 = 2.6	13.1-11.5 = 1.6	13.1-12.5 = 0.6	13.1-13.5 = -0.4	13.1-14.5 = -1.4
6.4-3.5 = 2.9	6.7-4.5 = 2.2	7.2-5.5 = 1.7	7.2-6.5 = 0.7	8.2-7.5 = 0.7	8.2-8.5 = -0.3	8.2-9.5 = -1.3	8.2-10.5 = -2.3	13.1-11.5 = 1.6	13.1-12.5 = 0.6	13.1-13.5 = -0.4	13.1-14.5 = -14
? - 3.5 = ?	6.6 - 4.5 = 2.1	6.4 - 5.5 = 0.9	6.7 - 6.5 = 0.2	7.2 - 7.5 = -0.3	7.2 - 8.5 = -1.3	7.2 - 9.5 = -2.3	8.2-10.5 = -2.3	8.2-11.5 = -3.3	8.2-12.5 = -4.3	8.2-13.5 = -5.3	8.2-14.5 = -6.3
? - 3.5 = ?	? - 4.5 = ?	6.6 - 5.5 = 1.1	6.4 - 6.5 = -0.1	6.7 - 7.5 = -0.8	6.7 - 8.5 = -0.8	7.2 - 9.5 = -2.3	7.2 -10.5 = -3.3	7.2 -11.5 = -4	7.2 -12.5 = -5	8.2 -13.5 = -5.3	8.2 -14.5 = -6.3
? - 3.5 = ?	? - 4.5 = ?	? - 5.5 = ?	6.6 - 6.5 = 0.1	6.6 - 7.5 = -0.9	6.4 - 8.5 = -2.1	6.7 - 9.5 = -1.8	6.7 -10.5 = -2.8	6.7-11.5 = -3.8	7.2-12.5 = -5	7.2-13.5 = -6	7.2-14.5 = -7

Figure 2.11: Voxels are updated with the difference between the depth image value and the distance to the respective voxel from the sensor, along the viewing direction.



? - 3.5 = ?	? - 4.5 = ?	? - 5.5 = ?	5.3 - 6.5 = -1.2	5.7 - 7.5 = -1.8	5.7 - 8.5 = -2.8	6.1 - 9.5 = -3.4	6.1 -10.5 = -4.4	6.5-11.5 = -5	6.5-12.5 = -6	6.5-13.5 = -7	7.1-14.5 = -7.4
? - 3.5 = ?	? - 4.5 = ?	5.3 - 5.5 = -0.2	5.7 - 6.5 = -0.8	6.1 - 7.5 = -1.4	6.1 - 8.5 = -2.4	6.5 - 9.5 = -3	6.5 -10.5 = -4	6.5 -11.5 = -5	7.1 -12.5 = -5.4	7.1 -13.5 = -6.4	7.1 -14.5 = -7.4
? - 3.5 = ?	5.3 - 4.5 = 0.8	5.7 - 5.5 = 0.2	6.1 - 6.5 = -0.4	6.5 - 7.5 = -1	6.5 - 8.5 = -2	6.5 - 9.5 = -3	7.1 -10.5 = -3.4	7.1 -11.5 = -4.4	7.1 -12.5 = -5.4	7.1 -13.5 = -6.4	7.6 -14.5 = -6.9
5.7-3.5 = 2.2	6.1-4.5 = 1.6	6.5-5.5 = 1	6.5-6.5 = 0	7.1-7.5 = -0.4	7.1-8.5 = -1.4	7.1-9.5 = -2.4	7.6-10.5 = -2.9	7.6-11.5 = -3.9	7.6-12.5 = -4.9	7.6-13.5 = -5.9	7.6-14.5 = -6.9
6.5-3.5 = 3	7.1-4.5 = 2.6	7.1-5.5 = 1.6	7.6-6.5 = 1.1	7.6-7.5 = 0.1	7.6-8.5 = -0.9	7.6-9.5 = -1.9	7.6-10.5 = -2.9	12.3-11.5 = 0.8	12.3-12.5 = -0.2	12.3-13.5 = -1.2	12.3-14.5 = -2.2
7.6-3.5 = 3.6	12.3-4.5 = 7.8	12.3-5.5 = 6.8	12.3-6.5 = 5.8	12.3-7.5 = 4.8	12.3-8.5 = 3.8	12.3-9.5 = 2.8	12.3-10.5 = 1.8	12.3-11.5 = 0.8	12.3-12.5 = -0.2	12.3-13.5 = -1.2	12.3-14.5 = -2.2
13.1-3.5 = 9.6	12.6-4.5 = 8.1	12.6-5.5 = 7.1	12.6-6.5 = 6.1	12.6-7.5 = 5.1	12.6-8.5 = 4.1	12.6-9.5 = 3.1	12.6-10.5 = 2.1	12.6-11.5 = 1.1	12.6-12.5 = 0.1	12.6-13.5 = -0.9	12.6-14.5 = -1.9
7.2-3.5 = 3.7	8.2-4.5 = 3.7	8.2-5.5 = 2.7	8.2-6.5 = 1.7	13.1-7.5 = 5.6	13.1-8.5 = 4.6	13.1-9.5 = 3.6	13.1-10.5 = 2.6	13.1-11.5 = 1.6	13.1-12.5 = 0.6	13.1-13.5 = -0.4	13.1-14.5 = -1.4
6.4-3.5 = 2.9	6.7-4.5 = 2.2	7.2-5.5 = 1.7	7.2-6.5 = 0.7	8.2-7.5 = 0.7	8.2-8.5 = -0.3	8.2-9.5 = -1.3	8.2-10.5 = -2.3	13.1-11.5 = 1.6	13.1-12.5 = 0.6	13.1-13.5 = -0.4	13.1-14.5 = -14
? - 3.5 = ?	6.6 - 4.5 = 2.1	6.4 - 5.5 = 0.9	6.7 - 6.5 = 0.2	7.2 - 7.5 = -0.3	7.2 - 8.5 = -1.3	7.2 - 9.5 = -2.3	8.2-10.5 = -2.3	8.2-11.5 = -3.3	8.2-12.5 = -4.3	8.2-13.5 = -5.3	8.2-14.5 = -6.3
? - 3.5 = ?	? - 4.5 = ?	6.6 - 5.5 = 1.1	6.4 - 6.5 = -0.1	6.7 - 7.5 = -0.8	6.7 - 8.5 = -0.8	7.2 - 9.5 = -2.3	7.2 -10.5 = -3.3	7.2 -11.5 = -4	7.2 -12.5 = -5	8.2 -13.5 = -5.3	8.2 -14.5 = -6.3
? - 3.5 = ?	? - 4.5 = ?	? - 5.5 = ?	6.6 - 6.5 = 0.1	6.6 - 7.5 = -0.9	6.4 - 8.5 = -2.1	6.7 - 9.5 = -1.8	6.7 -10.5 = -2.8	6.7-11.5 = -3.8	7.2-12.5 = -5	7.2-13.5 = -6	7.2-14.5 = -7

Figure 2.12: By interpolating between voxels we can obtain a surface, defined as the boundary between positive and negative values.

In this example however, we use the aforementioned values and note that the resulting surface fits much better with the measured data. The truncated distance is formally defined as,

$$D_t(\boldsymbol{d}) = max(min(z_n(\boldsymbol{\pi}(\boldsymbol{d})) - d_3, D_{max}), D_{min}) \tag{2.15}$$

When querying $D_t(\boldsymbol{x})$, $\boldsymbol{x}$ may be out of bounds (since the underlying voxel grid only contains a subset of $\mathbb{R}^3$). In this case we simply return the value D_{max}. When $\boldsymbol{x}$ is within bounds, the returned value is a tri-linear interpolation between the nearest 8 values of $D_t(\boldsymbol{d})$. So even if the actual data-structure of $D_t(\boldsymbol{x})$ is a discrete array, we can treat it as a continuous function in $\mathbb{R}^3$ for queries.

Furthermore, since we are working with sensors that provide video streams, we receive new data that affect the information currently used to represent $D_t(\boldsymbol{x})$. Instead of simply replacing old values with new, it is proposed by [7] to compute a weighted average of the data coming from the sensors. For this purpose, let there be a function $W(\boldsymbol{w})$ with $\boldsymbol{w} = [w_1, w_2, w_3]^T \in \mathbb{N}_0^3$,

$$W : \boldsymbol{w} \to \mathbb{R}$$

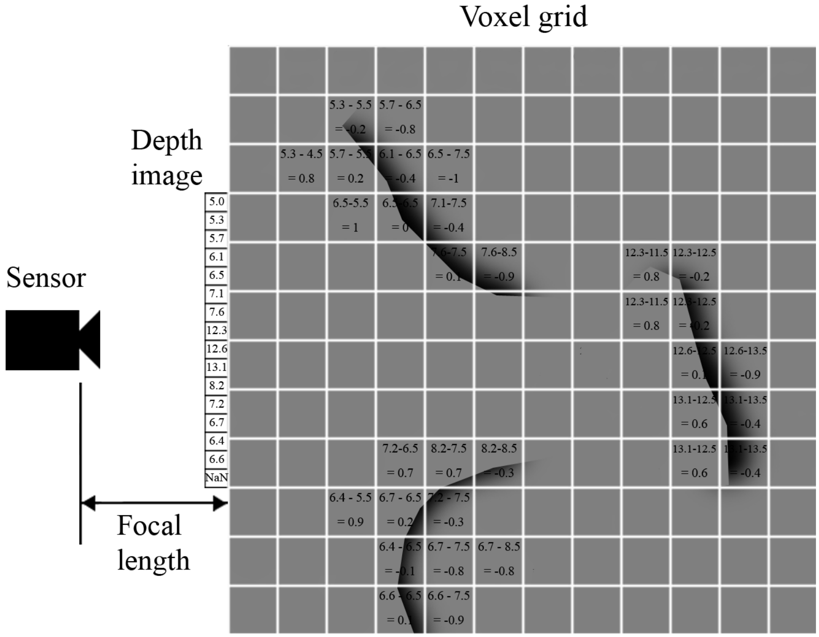


Figure 2.13: By interpolating between voxels we can obtain a surface, defined as the boundary between positive and negative values, only values in the range $[-1, 1]$ are considered.

representing the weight of the data stored in D. An update of a single element in D and W is then done by

$$D_t(\boldsymbol{d})_{n+1} = \frac{D_t(\boldsymbol{d})_n W(\boldsymbol{w})_n + D_t(\boldsymbol{d}) W(\boldsymbol{w})}{W(\boldsymbol{w})_n + W(\boldsymbol{w})}, \tag{2.16}$$

$$W(\boldsymbol{w})_{n+1} = min(W(\boldsymbol{w})_n + W(\boldsymbol{w}), W_{max}) \tag{2.17}$$

$W(\boldsymbol{w})$ may be computed according to an error model for the sensor, for each $z_n(\boldsymbol{m})$. For structured light sensors we have,

$$W(\boldsymbol{w}) = \frac{cos\theta}{err(z_n(\boldsymbol{\pi}(\boldsymbol{w})))} \tag{2.18}$$

where θ is the angle between the estimated surface normal and the ray through $\boldsymbol{m}$ from the sensor origin (the green rays in Fig. 2.6). Note that the surface normal here has to be estimated directly from the raw input data.

This tells us that we are most certain about measurements perpendicular to surfaces that are close to the sensor and that error increases according to an error model for the sensor that varies with distance. For structured light technologies in general this can be modelled as a linear function of z [37]. Specifically for the Kinect sensor, a more accurate model, taking into account quantization effects, has been proposed [30], which is approximately quadratic with respect to z.

$$err(z) = \frac{q_{pix} \cdot b \cdot f}{2} \cdot \left[\frac{1}{round(\frac{q_{pix} \cdot b \cdot f}{z} - 0.5)} - \frac{1}{round(\frac{q_{pix} \cdot b \cdot f}{z} + 0.5)} \right] \quad (2.19)$$

where

q_{pix}	subpixel resolution of the device (8 for Kinect)
f	focal length of the device, in pixels (ca. 520 for Kinect)
b	baseline, between projector and IR camera, in meters (0.075 for Kinect)

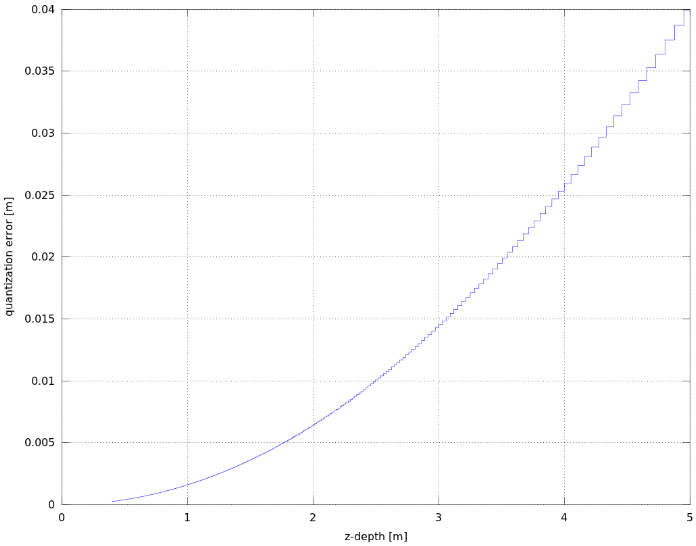


Figure 2.14: Model of the systematic error of the kinect sensor, with respect to depth.

With regard to the computation of W; a simplifying assumption that we make use of, instead of Eq. (2.18), is to set $W = 1$. We thereby reduce the weighted update to a rolling average (rolling, due to the saturation at W_{max}). This saves the cost of having to produce an estimate $\hat{\boldsymbol{n}}$ of the surface normals from the depth image, which would be needed to compute $cos\theta$.

Before we go through an algorithmic overview of this section, a few words about the initialization of D_t and W. Since we choose to truncate positive distances at D_{max}, this value is also our most natural choice for the value representing *empty space*. The initial weight attributed to each voxel is just as naturally chosen as being zero since, at the start, nothing has been measured. Although an empty and unobserved space are both represented by a distance of D_{max}, the weight of zero (in the unseen space) can be used to distinguish between the two cases.

Algorithm 2 SDF initialization.

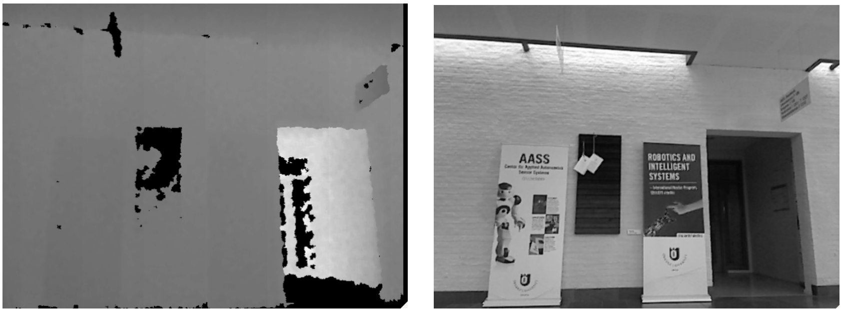
```
for all $\boldsymbol{d} \in D_t$ do
    $D_t(\boldsymbol{d}) \leftarrow D_{max}$
end for
for all $\boldsymbol{w} \in W$ do
    $W(\boldsymbol{w}) \leftarrow 0$
end for
```

Algorithm 3 Truncated SDF update function, with rolling average.

```
Require: $z_n(\boldsymbol{M}) \neq NaN$
for all $\boldsymbol{d} \in D_t$ do
    $\boldsymbol{w} \leftarrow \boldsymbol{d}$
    $weight \leftarrow 1.0$
    $distance \leftarrow max(min(z_n(\boldsymbol{\pi}(\boldsymbol{d})) - d_3, D_{max}), D_{min})$
    $D_t(\boldsymbol{d}) \leftarrow \frac{D_t(\boldsymbol{d})W(\boldsymbol{w}) + weight \cdot distance}{W(\boldsymbol{w}) + weight}$
    $W(\boldsymbol{w}) \leftarrow min(W(\boldsymbol{w}) + weight, W_{max})$
end for
```

To give a concrete example, consider Fig. 2.15; an example depth image of a wall with an adjoining corridor. We initialize a voxel grid of 200 × 200 × 200 elements, with a spatial resolution $\tau = 0.05m$. We allow the SDF update function to include new measurements for several minutes, and there is no *visible* change in the implicit surface represented by $D_t(\boldsymbol{x}) = 0$.

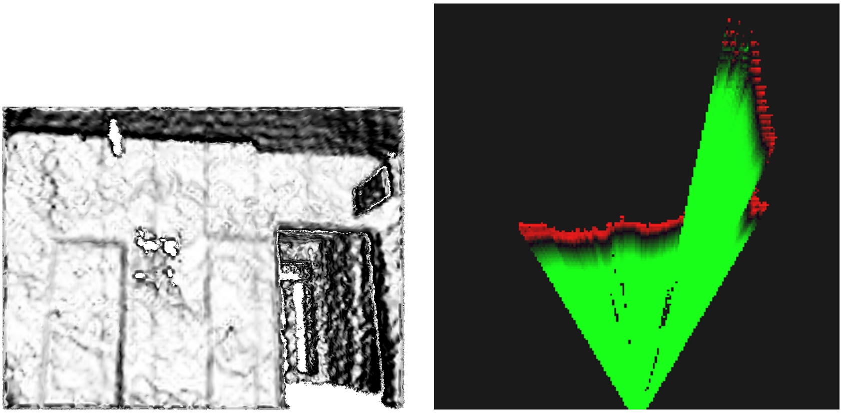
For visualization we use algorithm 1 (sphere-tracing) to produce an image from a virtual camera. The colour at each point is given by the dot product between the vectors ρ and $\boldsymbol{n}(\boldsymbol{x})$ as defined in section 2.2. Another example is given in Fig. 2.16, made by taking a 200 × 200 slice from the voxel grid and treating it as a two-dimensional image. The same colour coding scheme is used as before (see the circle example in section 2.1). Negative values are mapped to red (brighter further from zero), positive values are mapped to green, with the same interpretation for variation in brightness. Completely unseen voxels are mapped to dark gray.



(a) depth image

(b) grayscale image (not used)

Figure 2.15: Depth image of a hallway. The grayscale image is included as a visual aid for the reader, but not actually used in any of our algorithms. Black values in the depth image are unsuccessful measurements and are returned as NaN (not a number).



(a) SDF rendered through sphere-tracing

(b) A slice of the voxel grid, displayed as an image

Figure 2.16: Visualizations of the generated SDF.

2.4 Discussion

We have reviewed one of the standard methods for generating a signed distance function from depth images, as a voxel-based representation of three-dimensional surfaces. As can be seen from the results presented in this chapter, the quality of the reconstructed surface leaves a lot to be desired. First of all, we note that even though the generated SDF is based on the average of many depth images, the resulting surfaces are not smooth. A possible source for this error, one could imagine, is that we pick only the closest pixel in the depth image when performing the perspective projection. Why not interpolate between the pixels in the depth image instead? From experiments we have noted that this does not result in any visible improvement to surface quality, but comes with a number of drawbacks. One is that interpolation over surface discontinuities produces what is known as *jump-edge* points. Jump edge points are measurements that erroneously appear in the empty space between two separate surfaces, along their boundaries. Even if we interpolate only between pixels that are relatively close in depth, it would still have the effect of rounding off sharp corners, blurring out details that might be of interest in the reconstruction. We would also have to exclude interpolation between valid pixels and NaN's.

A more likely explanation for the lack of surface smoothness is the limited detail in the structured light sensor's illumination pattern, as well as quantization errors in its depth estimation, which is a systematic (repeatable, non-stochastic) error.

Another point that is of interest is that the resulting distance function is not Euclidean, as can be seen from the orientation of the gradient (note how the dark stripes all point to the origin of the green "cone") in Fig. 2.16, b. The resulting distance field is better described as consisting of *projective* or *line of sight* distances.

The projective distances can be corrected, close to planar surfaces, by multiplying the distance values by $cos\theta$ [12] (defined earlier in this section). This correction can be applied when integrating the sensor data into the SDF, based on the estimated surface normals (see Fig. 2.8). We include the information about how projective distances can be corrected for completeness and note that the additional computation required to produce the estimate (and filtering) of the normal vectors is not justified by any increase in performance in subsequent camera pose-estimation (according to benchmarks done with an RGBD dataset [35] with known ground-truth).

The surface reconstruction can be vastly improved by making measurements from different points of view. However, to have any use for data at different view-points, we need to know the position of the camera. This will be the topic of Chapter 3.

Chapter 3

Registration

This chapter presents the camera pose-estimation as a series of convex optimization problems that, each in turn is no harder than solving a simple system of linear equations. To arrive at these easy to solve problems, we start with an initial formulation that very literally states what we wish to achieve. We then make one key simplifying assumption about the nature of the problem and devise an approximation based on our 3D representation.

3.1 Problem statement

To register two (or more) sets of surface measurements, is to *align* them so that the parts that are commonly seen in both sets overlap as well as possible. The problem of registration can more precisely be stated as that of finding a *transformation* $\boldsymbol{T}$ that will put these corresponding parts into alignment. Additionally, due to the often impossible task of perfect *alignment*, we look for a transformation that minimizes some distance measure between corresponding parts in the measurements of the surfaces instead. Let $\boldsymbol{T} \in \mathbb{R}^{4\times4}$ denote a homogeneous transformation matrix *i.e.*,

$$\boldsymbol{T} = \left[\begin{array}{cc} \boldsymbol{R} & \boldsymbol{t} \\ \boldsymbol{0}^T & 1 \end{array} \right], \tag{3.1}$$

where $\boldsymbol{R}$ is a 3×3 rotation matrix, $\boldsymbol{t} \in \mathbb{R}^3$ is a translation vector and $\boldsymbol{0}^T$ is a row-vector of appropriate dimensions. This matrix represents a complete three-dimensional, rigid-body transformation, including both rotation and translation. To transform an arbitrary 3D point, in frame of reference $\mathcal{A}$, e.g. $\boldsymbol{p}_{\mathcal{A}} = (x, y, z)^T$ to frame of reference $\mathcal{B}$ we simply re-write $\boldsymbol{p}_{\mathcal{A}}$ in homogeneous coordinates, i.e., $\bar{\boldsymbol{p}}_{\mathcal{A}} = (x, y, z, 1)^T$. The transformation from $\mathcal{A}$ to $\mathcal{B}$ is then,

$$\bar{\boldsymbol{p}}_{\mathcal{B}} = \boldsymbol{T}_{\mathcal{B}\mathcal{A}}\bar{\boldsymbol{p}}_{\mathcal{A}}.$$

When computing the optimal transformation $\boldsymbol{T}$ (an optimality criterion will follow shortly) it will be convenient to have a more compact representation for it than the 12 combined parameters of the rotation matrix and translation vector. Since we know that there are only actually 6 DoF (yaw, pitch, roll, up-down, back-forth and left-right) we would like to have 6 parameters as well. One such parametrization can be made using Lie groups and Lie algebra (see [28] pp. 34–42, [21] pp. 34–37). The 3D rigid body transformation $\boldsymbol{T}$ belongs to the *Special Euclidean* Lie group of dimension 3,

$$\boldsymbol{T} \in \mathbb{SE}(3),$$

and can be parametrized in a neighbourhood around the identity transformation, by $\boldsymbol{\xi} \in \mathfrak{se}_3$, which is the Lie algebra associated with $\mathbb{SE}(3)$. Conceptually $\boldsymbol{\xi}$ can be regarded as a vector "stacking" angular velocities $\boldsymbol{\omega} = (\omega_1, \omega_2, \omega_3)^T$ and linear velocities $\boldsymbol{v} = (v_1, v_2, v_3)^T$,

$$\boldsymbol{\xi} = \begin{bmatrix} \boldsymbol{\omega} \\ \boldsymbol{v} \end{bmatrix}. \tag{3.2}$$

The mapping from the parameter vector $\boldsymbol{\xi}$ to the transformation $\boldsymbol{T}$ is done through the exponential map,

$$\boldsymbol{T}(\boldsymbol{\xi}) = e^{\tilde{\boldsymbol{\xi}}\Delta t}, \tag{3.3}$$

requiring a couple more definitions, namely that

$$\tilde{\boldsymbol{\xi}} = \begin{bmatrix} \tilde{\boldsymbol{\omega}} & \boldsymbol{v} \\ \boldsymbol{0}^T & 0 \end{bmatrix}, \tag{3.4}$$

where $\tilde{\boldsymbol{\omega}}$ is a skew-symmetric (or antisymmetric) matrix *i.e.*,

$$\tilde{\boldsymbol{\omega}} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix}, \tag{3.5}$$

and that $\Delta t \in \mathbb{R}$ denotes a time interval. In our application, we can suppose that Δt is always 1.0 and let any scaling it would have performed on $\hat{\boldsymbol{\xi}}$ be incorporated in the norm of $\boldsymbol{\xi}$ itself. Lastly e denotes the matrix exponential.

The exponential mapping is only guaranteed to yield a *distinct* transformation matrix close to $\boldsymbol{\xi} = \boldsymbol{0}$, so we cannot increment $\boldsymbol{\xi}$ indefinitely. For this reason, we periodically re-set $\boldsymbol{\xi}$ to zero, using it to represent only incremental transformations in pose.

Incrementing one transformation with another is straightforward. Recall the earlier transformation, from frame of reference $\mathcal{A}$ to $\mathcal{B}$. Now, imagine that we want to take a point from $\mathcal{A}$ to $\mathcal{C}$, having already transformed it into frame of reference $\mathcal{B}$,

$$\bar{\boldsymbol{p}}_{\mathcal{C}} = \boldsymbol{T}_{\mathcal{CB}}\bar{\boldsymbol{p}}_{\mathcal{B}}.$$

Since matrix multiplication is associative, we can define the transformation from $\mathcal{A}$ to $\mathcal{C}$ directly as,

$$\boldsymbol{T}_{\mathcal{C}\mathcal{A}} = \boldsymbol{T}_{\mathcal{C}\mathcal{B}}\boldsymbol{T}_{\mathcal{B}\mathcal{A}},$$

without needing to keep any intermediate transformations.

3.2 Objective function

We begin the elaboration on our *objective function*, which is a formal way of expressing "the *thing* we want to minimize", in the context of two sequential depth images taken from slightly different viewpoints. We can intuitively state that our objective should be to minimize the norm of the differences between all the three-dimensional surface points $\boldsymbol{s}_n(\boldsymbol{m})$ and $\boldsymbol{s}_{n+1}(\boldsymbol{m})$, squared. Recall that $\boldsymbol{s}$ is simply a point, obtained by projection of a pixel from a depth image into 3D space (see Equation (2.9)) and that n denotes the nth image in stream of depth images. Let $\boldsymbol{M} = \{m_1, \ldots, m_K\}$, *i.e.*, $\boldsymbol{M}$ has K elements. We then define,

$$\sum_{j=1}^{K} \|\boldsymbol{s}_n(\boldsymbol{m}_j) - \boldsymbol{s}_{n+1}(\boldsymbol{m}_j)\|_2^2, \tag{3.6}$$

To introduce a means for influencing the outcome of this sum, we include $\boldsymbol{T}(\boldsymbol{\xi})$ and use homogeneous coordinates $\bar{\boldsymbol{s}}(\boldsymbol{m})$ to enable manipulating one set of points relative to the other. We now have a proper objective function, stating that we wish to find the transformation $\boldsymbol{T}$ that makes the second set of points move as close as possible to the first set,

$$\underset{\boldsymbol{\xi}}{\text{minimize}} \sum_{j=1}^{K} \|\bar{\boldsymbol{s}}_n(\boldsymbol{m}_j) - \boldsymbol{T}(\boldsymbol{\xi})\bar{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_j)\|_2^2. \tag{3.7}$$

Since we do not have an explicit list of correspondences, we follow an approach similar to the *Iterative Closest Point* algorithm where, instead of trying to match points to their exact correspondences, we simply try to move them to closest point in the previous scan. Since our truncated signed distance function is an approximate measure of the distance to this closest point (and since it is constructed based on previous measurements), we can make the following approximation:

$$\bar{\boldsymbol{s}}_n(\boldsymbol{m}) - \boldsymbol{T}(\boldsymbol{\xi})\bar{\boldsymbol{s}}_{n+1}(\boldsymbol{m}) \approx \bar{D}_t(\boldsymbol{T}(\boldsymbol{\xi})\bar{\boldsymbol{s}}_{n+1}(\boldsymbol{m})), \tag{3.8}$$

so that we instead have the following function,

$$\underset{\boldsymbol{\xi}}{\text{minimize}} \sum_{j=1}^{K} \|\bar{D}_t(\boldsymbol{T}(\boldsymbol{\xi})\bar{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_j))\|_2^2. \tag{3.9}$$

This objective function accurately expresses our desire to minimize the distance between the new surface measurement and the surface *implied* by the previous measurement through a rigid-body transformation applied to the new data. Unlike the initial formulation, however, it does not require explicit correspondences to be fixed between points. Furthermore and due to the accumulation of data into $\boldsymbol{D}_t$, we are implicitly comparing $\boldsymbol{s}_{n+1}(\boldsymbol{m})$ with *all* past $\boldsymbol{s}$ instead of just the previous frame.

3.3 Solution

3.3.1 Linearisation

To make the subsequent notation easier to read we shall define the following alias

$$f(\boldsymbol{\xi}, \boldsymbol{m}) = \bar{D}_t(\boldsymbol{T}(\boldsymbol{\xi})\bar{\boldsymbol{s}}_{n+1}(\boldsymbol{m})). \tag{3.10}$$

We know that the above expression represents a non-linear function. However, for sufficiently small rotations and translations, we can approximate f by a function that is linear with respect to our optimization parameter, $\boldsymbol{\xi}$. This can be done by replacing f with the first two terms of its power series expansion around $\boldsymbol{\xi} = \mathbf{0}$.

$$f(\boldsymbol{\xi}, \boldsymbol{m}) \approx f(\boldsymbol{\xi}, \boldsymbol{m})|_{\boldsymbol{\xi}=\mathbf{0}} + \nabla_{\boldsymbol{\xi}} f(\boldsymbol{\xi}, \boldsymbol{m})|_{\boldsymbol{\xi}=\mathbf{0}}\, \boldsymbol{\xi}, \tag{3.11}$$

remembering that $\boldsymbol{T}(\boldsymbol{\xi}) = e^{\tilde{\boldsymbol{\xi}}\Delta t}$ and the definition behind $f(\boldsymbol{\xi}, \boldsymbol{m})$, we can expand the above equation,

$$f(\boldsymbol{\xi}, \boldsymbol{m}) \approx \bar{D}_t(e^{\tilde{\boldsymbol{\xi}}\Delta t}\bar{\boldsymbol{s}}_{n+1}(\boldsymbol{m}))\Big|_{\boldsymbol{\xi}=\mathbf{0}} + \nabla_{\boldsymbol{\xi}} \bar{D}_t(e^{\tilde{\boldsymbol{\xi}}\Delta t}\bar{\boldsymbol{s}}_{n+1}(\boldsymbol{m}))\Big|_{\boldsymbol{\xi}=\mathbf{0}} \boldsymbol{\xi}. \tag{3.12}$$

Noting that $e^{\mathbf{0}}$ is an identity matrix we can omit it entirely in the above expression (due to the evaluation at $\boldsymbol{\xi} = \mathbf{0}$). It then simplifies to:

$$f(\boldsymbol{\xi}, \boldsymbol{m}) \approx \bar{D}_t(\bar{\boldsymbol{s}}_{n+1}(\boldsymbol{m})) + \nabla_{\boldsymbol{\xi}} \bar{D}_t(\bar{\boldsymbol{s}}_{n+1}(\boldsymbol{m}))\boldsymbol{\xi}. \tag{3.13}$$

We point out that the second term of the function is a matrix, containing the partial derivatives of $\bar{D}_t(\bar{\boldsymbol{s}}_{n+1}(\boldsymbol{m}))$ with respect to the vector $\boldsymbol{\xi}$, multiplied by a vector. This matrix is what is commonly known as the Jacobian matrix (or simply Jacobian) of the function. As it will require some attention, we define it as:

$$\boldsymbol{J}(\boldsymbol{m}) = \nabla_{\boldsymbol{\xi}} \bar{D}_t(\bar{\boldsymbol{s}}_{n+1}(\boldsymbol{m})). \tag{3.14}$$

The linearised version of (3.9) is then

$$\underset{\boldsymbol{\xi}}{\text{minimize}} \sum_{j=1}^{K} \|\boldsymbol{J}(\boldsymbol{m}_j)\boldsymbol{\xi} + \bar{D}_t(\bar{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_j))\|_2^2. \tag{3.15}$$

3.3.2 The Jacobian Matrix

The Jacobian matrix, is the derivative of our SDF with respect to the parameter-vector that defines the rigid-body transformation, $\boldsymbol{T}$. We already know, from section 2.2, that we can compute a derivative (or gradient) of the SDF with respect to its spatial dimensions, i.e.,

$$\nabla_{\boldsymbol{x}} \bar{D}_t(\bar{\boldsymbol{x}}) = \frac{\partial \bar{D}_t(\bar{\boldsymbol{x}})}{\partial \bar{\boldsymbol{x}}} = \left[\tfrac{\partial}{\partial x_1} \bar{D}_t(\bar{\boldsymbol{x}}) \quad \tfrac{\partial}{\partial x_2} \bar{D}_t(\bar{\boldsymbol{x}}) \quad \tfrac{\partial}{\partial x_3} \bar{D}_t(\bar{\boldsymbol{x}}) \right], \tag{3.16}$$

by numerically differentiating $\bar{D}_t(\bar{\boldsymbol{x}})$ (e.g. by central differences) over x_1, x_2, and x_3 (again, $\bar{\boldsymbol{x}}$ simply denotes homogeneous coordinates $\bar{\boldsymbol{x}} = [\boldsymbol{x}, 1]^T$. This, however, does not include any derivation with respect to $\boldsymbol{\xi}$. To arrive at a complete expression for $\boldsymbol{J}(\bar{\boldsymbol{x}})$ we use the chain rule:

$$\boldsymbol{J}(\boldsymbol{x}) = \frac{\partial \bar{D}_t(\bar{\boldsymbol{x}})}{\partial \boldsymbol{\xi}} = \frac{\partial \bar{D}_t(\bar{\boldsymbol{x}})}{\partial \bar{\boldsymbol{x}}} \frac{\partial \bar{\boldsymbol{x}}}{\partial \boldsymbol{\xi}}. \tag{3.17}$$

This leaves us with the need for an expression for how the position of a given point varies with our parameter vector. This is easily obtained by analysing the resulting change in $e^{\hat{\boldsymbol{\xi}}\Delta t}\boldsymbol{x}$ with respect to a change in each element of $\boldsymbol{\xi}$. For any given point $\boldsymbol{x}$ we have;

$$\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{\xi}} = \begin{bmatrix} 0 & x_3 & -x_2 & 1 & 0 & 0 \\ -x_3 & 0 & x_1 & 0 & 1 & 0 \\ x_2 & -x_1 & 0 & 0 & 0 & 1 \end{bmatrix}. \tag{3.18}$$

3.3.3 The Normal Equations

Having defined all the components of the objective function in (3.15), we expand it to

$$\underset{\boldsymbol{\xi}}{\text{minimize}} \sum_{j=1}^{K} [\boldsymbol{J}(\boldsymbol{m}_j)\boldsymbol{\xi} + \bar{D}_t(\bar{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_j))]^T [\boldsymbol{J}(\boldsymbol{m}_j)\boldsymbol{\xi} + \bar{D}_t(\bar{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_j))], \tag{3.19}$$

and carry out the product to arrive at:

$$\underset{\boldsymbol{\xi}}{\text{minimize}} \sum_{j=1}^{K} \boldsymbol{\xi}^T \boldsymbol{J}(\boldsymbol{m}_j)^T \boldsymbol{J}(\boldsymbol{m}_j)\boldsymbol{\xi} + 2\boldsymbol{\xi}^T \boldsymbol{J}(\boldsymbol{m}_j)\bar{D}_t(\bar{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_j)) + \bar{D}_t(\bar{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_j))^2. \tag{3.20}$$

Carrying out the summation over $\boldsymbol{M}$ for the terms dependent on $\boldsymbol{\xi}$ we can form the following matrix and vector pair,

$$\boldsymbol{H} = \sum_{j=1}^{K} \boldsymbol{J}(\boldsymbol{m}_j)^T \boldsymbol{J}(\boldsymbol{m}_j) \in \mathbb{R}^{6\times 6}. \tag{3.21}$$

$$\boldsymbol{g} = \sum_{j=1}^{K} \boldsymbol{J}(\boldsymbol{m}_j)^T \bar{D}_t(\bar{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_j)) \in \mathbb{R}^6. \tag{3.22}$$

Clearly $\boldsymbol{H}$ is a symmetric matrix. Furthermore, it is very likely for it to be positive definite due to the large number of linearly independent vectors added (one per pixel in the depth image). For $\boldsymbol{H}$ to be full-rank, we only really need six linearly independent vectors when forming it. Substituting $\boldsymbol{H}$ and $\boldsymbol{g}$ back into (3.20) leads to,

$$\underset{\boldsymbol{\xi}}{\text{minimize}} \;\; \boldsymbol{\xi}^T \boldsymbol{H} \boldsymbol{\xi} + 2\boldsymbol{\xi}^T \boldsymbol{g} + \sum_{j=1}^{K} \bar{D}_t(\bar{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_j))^2. \tag{3.23}$$

This is a quadratic (strictly convex, since $\boldsymbol{H} > 0$) function of $\boldsymbol{\xi}$. To find its minimum we derive the objective function with respect to $\boldsymbol{\xi}$ and equate the resulting expression with zero. This is analogous to finding the inflection point on a parabola with positive curvature (see Fig. 3.1). Finding the optimal $\boldsymbol{\xi}$ can be done by computing the inverse of $\boldsymbol{H}$, as shown in the example, or through any standard linear least-squares solver.

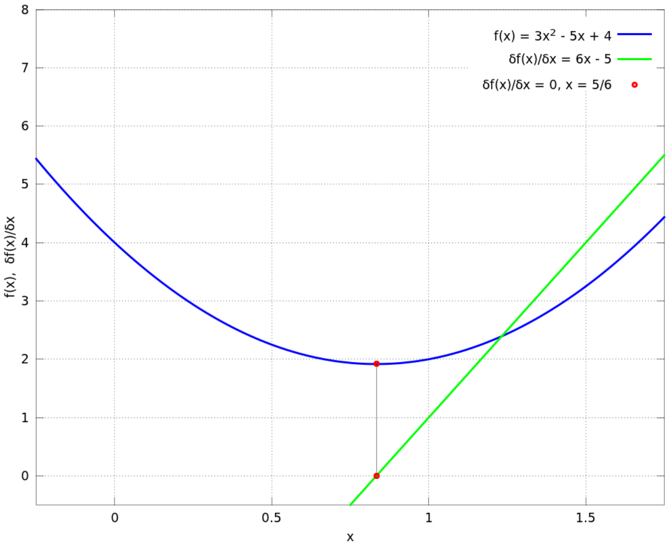


Figure 3.1: As an analogous example, the minimum of a quadratic function can be found by solving its derivative for zero and substituting the resulting value into the original function.

$$2\boldsymbol{H}\boldsymbol{\xi} + 2\boldsymbol{g} = \boldsymbol{0}, \tag{3.24}$$

$$\boldsymbol{\xi}^{\star} = -\boldsymbol{H}^{-1}\boldsymbol{g} \tag{3.25}$$

Since we have made quite a number of simplifications along the way and relied only on local information around each 3D point, this solution is unfortunately only the first step towards an accurate registration. To get a satisfactory result, the transformation parametrized by $\boldsymbol{\xi}^{\star}$ is applied to the points, and the procedure is repeated. A stopping condition is typically set when the change in the parameter falls below some threshold.

3.4 Practical Considerations

3.4.1 For speed

As the reader may have noticed, most of the calculations needed for the SDF generation/update and registration are completely independent of each other. The SDF is updated per voxel (with no regard for what is happening to neighbouring cells), likewise, most of the computations in the registration is done for each pixel in the depth image.

The matrix $\boldsymbol{H}$ and the vector $\boldsymbol{g}$ are the results of sums over all pixels. This implies some sort of synchronization and amalgamation of data, but even this is solved efficiently as a parallel reduction, using a series of intermediary sums. The final step of solving the 6×6 linear system is fast and done relatively few times throughout the process. Due to these characteristics it is very straightforward to implement both the localization and mapping as a multi-threaded work-sharing algorithm.

A common way of both speeding up the solution and avoiding getting stuck in a local minima is to iterate on a coarse to fine level of detail. The standard approach [4] to this implies blurring some image with a Gaussian filter and then sub-sampling it in several steps, forming what is called a “Gaussian Pyramid”. In our case we skip the blurring operation and simplify the coarse to fine iteration scheme by merely computing the derivatives of the SDF (when forming the Jacobian) using central differences between voxels that are further apart and by sub-sampling $\boldsymbol{M}$ to use $1/16, 1/4$ and $1/1$ of the pixels. The result of this is that we can make a larger number of inexpensive iterations that converge fast, and reduce the number of iterations as we include more data. To support this, we perform the registration over several hundred frames and compute the mean rate of convergence. We measure this by the norm of the change in the parameter vector between consecutive iterations. Figures 3.2, 3.4 and 3.2 show the performance when registration is done only on the finest scale, when using sub-sampling on the input data and finally when sub-sampling both the input and computing derivatives in the SDF over a greater number of voxels.

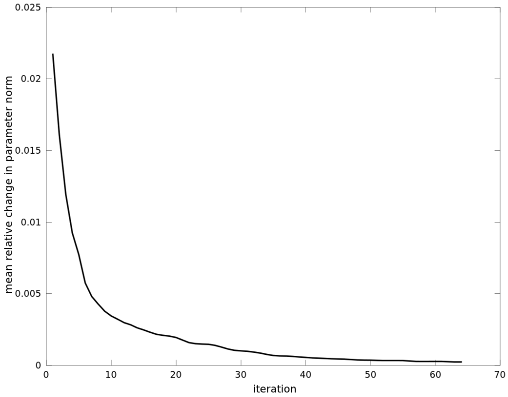


Figure 3.2: The convergence at fine scale is characterized by relatively large initial change, that drops quickly.

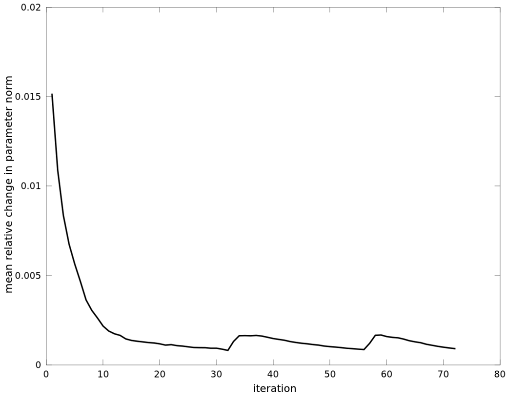


Figure 3.3: The convergence in the coarse-to-fine setting is similar to that of fine-scale registration initially. Note that the change is somewhat larger initially and consider also that the initial steps (on the coarsest level) take only a fraction of the time of the fine-scale registration.

Also, considering the fact that our measure for distance ($\bar{D}_t(\bar{\boldsymbol{x}})$) is truncated at a value D_{max} from the surface, it makes no sense to consider points

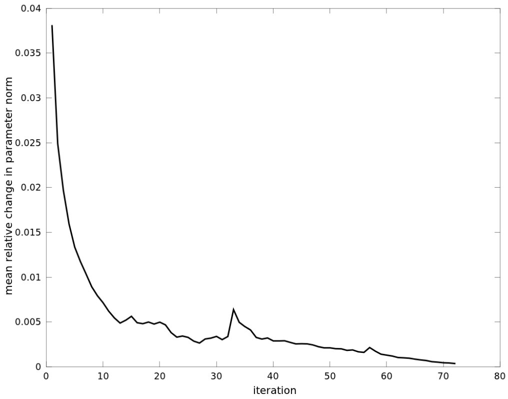


Figure 3.4: The amount of change that occurs in the first iterations of the coarse-to-fine setting, where numerical derivatives are also computed with subsampling, is larger still since it truly overlooks small details in the initial phase. The *rate* of change is not as smooth, however.

beyond that. This is because any gradient computed there is likely to be misleading, and the distance value is not informative. We definitely know that we would like to skip points where the SDF evaluates to D_{max}. To make sure we do not use a truncated value when computing our derivatives also, we tend to be even more restrictive. This is because derivatives are computed with an offset on either side of the point of interest. Skipping points that are invalidated by this distance criteria reduces some of the computational burden but is mainly done to ensure robustness to outliers.

Using our chosen parametrization for $\boldsymbol{T}$, we can extrapolate and interpolate between nearby transformation matrices. We can use this to hot-start the optimization for the next incoming data-frame by initializing $\boldsymbol{\xi}_n$ as

$$\boldsymbol{\xi}_n = s \cdot (\boldsymbol{\xi}_{n-1} - \boldsymbol{\xi}_{n-2})$$

instead of zero. Here, $s \in \mathbb{R}$ is a scalar representing how far we wish to extrapolate the last movement. When the movement is smooth, this leads to much faster convergence, since we tend to start closer to the optimal solution for the next frame. If the movement is non-smooth (or if the frame-rate is very low) this method can lead to over-shoots which do more harm than good.

Finally, as has been mentioned earlier, we do not perform an estimate of surface normals directly from the input data. This means that we also discard any computations that could be based on these. This saves time, and does not have any noticeable effect on the quality of the registration or surface representation.

3.4.2 For accuracy

Robust statistics

Besides rejecting data points that obviously have no benefit to the registration, we can take into consideration that our distance field is not quite Euclidean, except for very close to a surface (where the approximation is better). To do this, we can borrow a concept from statistics, called an M-estimator [39], which can be cheaply implemented as iteratively re-weighted least squares (IRLS). Consider, for example, Tukey's estimator:

$$w(r) = \begin{cases} if|r| \leq c & [1 - (\frac{r}{c})^2]^2 \\ if|r| > c & 0 \end{cases} \tag{3.26}$$

with c being a constant threshold (lower than D_{max}) and r being a measure for the residual (in our case, represented by the SDF) we have a weight that can be computed and factored into each summand of equations (3.21) and (3.22). Note that the weight for points above the threshold is zero. This essentially says that we want to reformulate our parameter estimation problem to disregard points that are not close to surfaces and to reduce the influence of points that are further from the surface when computing our solution.

The use of robust estimators has been shown to widen the convergence basin during registration [8], and comes at virtually no added computational cost, which is why we employ this method in our algorithm.

$$\boldsymbol{H_w} = \sum_{j=1}^{K} w(\bar{D}_t(\bar{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_j)))\boldsymbol{J}(\boldsymbol{m}_j)^T\boldsymbol{J}(\boldsymbol{m}_j), \tag{3.27}$$

$$\boldsymbol{g_w} = \sum_{j=1}^{K} w(\bar{D}_t(\bar{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_j)))\boldsymbol{J}(\boldsymbol{m}_j)^T\bar{D}_t(\bar{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_j)). \tag{3.28}$$

Handling Discontinuities

Another, important fact to consider is that, when computing the gradient of $\bar{D}_t(\bar{\boldsymbol{x}})$ outside a surface, it points toward the surface (which is what we expect). However, when computing the gradient at a point behind a surface, due to the unseen regions being valued D_{max}, this may flip the gradient in the wrong direction. In practice we can solve this by either taking the absolute value of $\bar{D}_t(\bar{\boldsymbol{x}})$ throughout the entire registration step, effectively making it *unsigned.* Or by expanding the negative region of the SDF further (and avoiding the region of the SDF close to D_{min}).

It's worth mentioning that even with an unsigned distance function, the quadratic approach is still a valid alternative. Without it we would have a linear function to minimize, but with no bounds on our parameter. A workaround would be to expand the power series in equation (3.15) to a second order approximation, but this requires less numerically stable (and more costly to compute) second-order derivatives of the SDF.

Regularization

By adding $\|\boldsymbol{\xi}\|_2^2 = \boldsymbol{\xi}^T\boldsymbol{\xi}$ to equation (3.23) we can express that we also want to minimize the norm of $\boldsymbol{\xi}$, as part of the objective function. In other words, making the parameter itself part of the objective. The result of doing so is that the optimal parameter is one that not only minimizes the sum related to distances, but also keeps itself as small as possible. To add different penalties to the elements of the parameter vector, e.g., if we have prior knowledge about the magnitude of translation compared to rotation, we can include this information in the appropriate entries of a diagonal matrix,

$$\boldsymbol{\Gamma} \in \mathbb{R}^{6\times 6},$$

which instead adds a *weighted* norm, i.e., $\boldsymbol{\xi}^T\boldsymbol{\Gamma}\boldsymbol{\xi}$ into the objective function. We can change the expression for $\boldsymbol{H}_w$ to reflect this,

$$\boldsymbol{H}_{\boldsymbol{w}+\boldsymbol{r}} = \sum_{j=1}^{K} \big(w(\bar{D}_t(\bar{\boldsymbol{s}}_{n+1}(\boldsymbol{m}_j)))\boldsymbol{J}(\boldsymbol{m}_j)^T\boldsymbol{J}(\boldsymbol{m}_j)\big) + \boldsymbol{\Gamma}. \tag{3.29}$$

If $\boldsymbol{\Gamma}$ is set to be the identity matrix, the regularization is done on the what is called the unweighted norm of the parameter. If set to zero, we revert to the un-regularized solution. This is a very commonly used regularization method (called Tikhonov regularization) in regression analysis.

3.5 Summary

The algorithm used here is, in its essence, an application of the Lucas-Kanade algorithm in 3D [22]. The mathematical notation and basic problem formulation here is similar to that of Steinbrücker et al. [33]. Due to the many alternatives and possible modifications that can be applied to the algorithms outlined in this work, we hereby offer the reader a summary of the implementation that we use for the main evaluations done in the next section. A simple flowchart is shown in Figure 3.5 and further described in the next paragraphs.

3.5.1 Measurements

The only information computed from the input depth-map are the 3D surface points that each pixel represents, according to equation (2.9). We do not estimate surface normals from the input data. We also keep the depth image, as it is needed in the reconstruction step (after which it may be discarded).

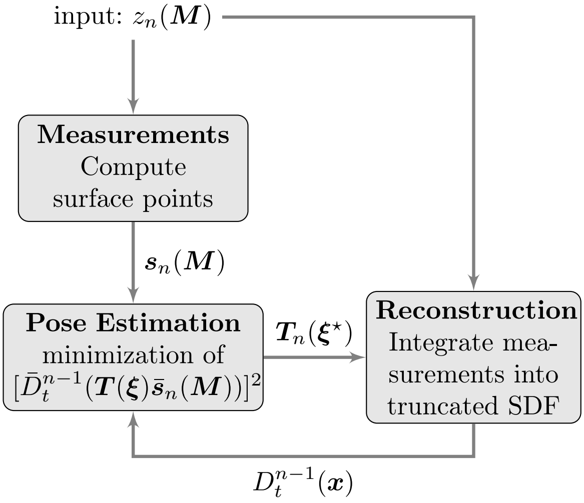

input: $z_n(\boldsymbol{M})$
Measurements
Compute surface points
$\boldsymbol{s}_n(\boldsymbol{M})$
Pose Estimation
minimization of $[\bar{D}_t^{n-1}(\boldsymbol{T}(\boldsymbol{\xi})\bar{\boldsymbol{s}}_n(\boldsymbol{M}))]^2$
$\boldsymbol{T}_n(\boldsymbol{\xi}^\star)$
Reconstruction
Integrate measurements into truncated SDF
$D_t^{n-1}(\boldsymbol{x})$

Figure 3.5: An overview of the main components and variables that make up the algorithm.

3.5.2 Pose Estimation

The pose estimation is done by first forming $\boldsymbol{H}_{w+r}$ and $\boldsymbol{g}_w$, as described in equations (3.29), (3.28) and solving the subsequent system of equations. The resulting transformation is applied to each surface point and the forming/solving step is repeated. We use the coarse-to-fine approach outlined earlier in this section to ensure faster convergence. The actual solution is done using a Robust Cholesky decomposition, provided in Eigen [17], valid for positive semi-definite matrices.

3.5.3 Reconstruction

The update of the truncated signed distance function is performed using Algorithm 3. The weight and distance are computed as indicated there, due to the lack of estimates for incidence angles (since we skipped the estimation of surface normals). We may still choose to set the weight to $\frac{1}{err(z)}$ instead of (2.18), of course.

Chapter 4

Towards Object Detection using SDF models

To perform object detection, we first generate a SDF of the environment in which our object of interest is located. In our method for object detection, we use a reference template representing the object we are looking for. This template is simply an array of 3D points uniformly spaced over the surface of the object, centred in its own coordinate system.

The procedure is then to very coarsely move the object template in fixed steps (roughly half the characteristic length of the object) throughout the SDF and attempt to register it to the environment model at each step. Although this brute-force approach may appear to be an exceedingly time-consuming process, many of the steps will typically initialize in either unseen or empty space (where the entire object is located in the truncated region). Regardless of where the template is initialized, we will have to iterate through each point pertaining to the template, at least once, and evaluate the SDF at each individual point. If every evaluation in the SDF produces a value of D_{max} we can safely skip ahead since this means that the object is not near a surface. The time spent on such an event is proportional to the number of points used to represent the object. Quite possibly, this may be in the same order of magnitude as the number of surface elements contained in a typical depth image (some hundred thousand) or less.

When the template is near a surface, we attempt a registration. Due to the arbitrary orientation that we initialize the template with, we typically have to perform a larger number of iterations in this process, compared to when registering a sequence of frames from the depth-camera. one registration attempt may take between 0.5 to 10 seconds, depending on the number of points, the criteria for convergence and if the coarse to fine registration scheme is employed. The total time spent searching will also be influenced heavily on how much clutter there is in the scene, how large the target object is, and so on. A complete search though a space of a few cubic meters can therefore take anywhere between a few seconds to several minutes.

When the registration is completed, we compute the sum of the residuals of the registration to evaluate how well the template matched to the environment at that particular location. Clearly, it would be ideal if the sum of residuals would be zero for the registration, as this would indicate that we have found a location for which each point on the surface of the template corresponds *exactly* to a surface in the environment. Seeing how most objects do not find themselves suspended in mid-air, we tend to find that objects are partially occluded where in contact with some supporting surface. The presence of other objects or an incomplete view of the environment will lead to further occlusions and unseen regions. The consequence of this is that not all of the surface of our object of interest is available for matching against the template.

Indeed, we can easily imagine a situation in which an object of similar-shaped geometry will do better than the target object itself, especially if the latter is heavily occluded (or even absent).

Before concluding this chapter, we merely note that this method for object detection requires virtually no modification from our present methods of registering frames. An experimental analysis on this approach will be presented in Chapter 5

4.1 Narrowing the Search Space

Searching through a small, mostly empty, space for a large object might be a feasible task, using a brute-force approach. However, in the interest of generalizing our object detection to be applicable to, for instance, finding small objects in large, cluttered spaces we need to employ different methods. An additional argument to support this is that if we attempt to detect objects that are highly variant with regards to orientation, our problem gains an added degree of combinatorial complexity, as we would need to test different locations and orientations. Currently, our registration-based method lacks information about colour and texture, which are known to be important when telling apart instances of objects pertaining to the same class (e.g., a soccer ball from a basketball).

To deal with the issues involved in scaling the object detection up to more complex applications we see the need for exploring the use of feature descriptors.

4.1.1 About Feature Descriptors

Feature descriptors are one or more set of numbers (often arranged in a vector) that encode one or more local properties in a given representation of an object or scene. The representation may be a photograph, in which the feature descriptor is the result of computations done over a neighbourhood of pixels. The representation may also be in 3D, with the corresponding feature being a series of numbers related to properties of the chosen representation.

To give an idea of what a feature descriptor might consist of, the following paragraphs give a brief summary of some common descriptors applicable to 3D representations.

- Fast Point Feature Histogram [31] (FPFH) is a feature based on a 3D point-cloud representation. In our notation a point-cloud can be defined as a vector containing all the $\boldsymbol{s}_n(\boldsymbol{m})$ of a particular depth image, stored in an array, and their estimated normal vectors. The feature descriptor is based on computing pairwise relationships between these surface points and aggregating the results into a histogram.

- 3D Spin Images [18] is the name of a representation that is also developed based on oriented points, i.e., points with associated estimates of surface normals. Spin images are computed by choosing one point as a reference, then parametrizing the position of the surrounding points, using two variables. These two variables are then used as indices to generate a so called Spin Image.

- Depth-Kernel Descriptors [5], are an application of a machine-learning concept called “kernel functions” to object recognition. The kernel functions are defined as measures of similarity between two image patches but can be redefined as features, computed over a single patch. Kernel functions are useful because they make use of the fact that densely grouped data often becomes sparse and easy to separate in higher dimensions. This can make classification problems simpler to solve.

Feature descriptors such as these can be used to avoid exhaustive searches, both of locations in space, and among candidate objects. The standard way of using feature descriptors in object detection can, in broad strokes, be described by the following process;

1. Acquire and pre-process sensor data
2. Find regions of interest in the data
3. Compute features at the regions of interest
4. Combine the features into a descriptor
5. Match the descriptor against a database of descriptors related to objects

Note that several descriptors in the database can be used to indicate a number of different poses of the same object. This means that the above process, when successful, provides an estimate for the location of an object and possibly even a rough initial estimate for its orientation.

In the same way that we find it desirable to use feature descriptors to narrow our search for objects, it is also assumed (as indicated in step 2) that there exists some criteria by which to narrow down the locations at which we compute the feature descriptors themselves.

A criteria by which to do this selection can be based on several properties. For example, in many applications applied to 2D images, features are computed at the crossing between two edges (a corner). In Chapter 5 we show a preliminary result of generalizing the concept of corners to 3D, by employing the measure for curvature that we introduced in Chapter 2.

Which feature descriptor to employ, among the numerous alternatives available, is still a question that remains unanswered. We note that there are many options that directly relate to 3D sensor data, for which standard implementations are available. Considering, however, that our 3D representation is, in essence, the same as an digital image (or a stack of images, to be more precise) we could also consider extending some of the popular 2D descriptors (e.g. [3],[6]) into an additional dimension.

Chapter 5

Results

In this section we present some results, divided into three categories.

1. Localization. We test our algorithm on a well-known data-set from the CVPR lab at the Technical University of Munich [35]. This data-set is popular for evaluating and comparing localization and scene reconstruction algorithms based on RGB+Depth data. We choose a video sequence, captured from a Kinect camera, in which the camera rapidly browses an office scene and we compare the pose and trajectory estimation with ground-truth data. The ground-truth is determined from a motion-capture system, using markers on the camera. We make some remarks on the failure modes of the system and robustness to missing data.

2. Mapping. We show some qualitative results from on-line 3D reconstructions of different scenes and illustrate the properties detailed in Chapter 2. Some comments are given on the numerous parameters of the system and what effect the change of these parameters seem to have on the quality of mapping (which is inextricably linked to the quality of the localization).

3. Object detection. We show the result of attempting to detect a simple object, in an environment, through registration.

5.1 Localization

Since the tracking works by querying an implicit function, represented by discrete samples, it comes as no great surprise that the quality of any position estimate, based on this function will be dependent on its discretization interval τ (voxel resolution). However, since our SDF returns linear interpolations

Table 5.1: Statistics about the translational and rotational errors along the path of the camera, relative to ground truth data.

	e_{rmse}	e_{mean}	e_{median}	$e_{std.dev.}$	e_{min}	e_{max}
Absolute Translational[m]	0.0402	0.0332	0.0279	0.0228	0.0019	0.1139
Relative Translational[m]	0.0477	0.0391	0.0273	0.0273	0.0037	0.1304
Relative Rotational[°]	4.545	3.308	0.041	3.117	0.192	15.110

between the values in its cells, this error is related to how well the represented geometry can be described by planar approximations. Large flat structures such as floors and walls can be very well approximated with relatively low resolution for this reason.

In the benchmarks presented here, we chose to use a fixed number of iterations on each level of sub-sampling. The levels being defined by taking every 4th, every 2nd and every pixel in each direction with the respective number of iterations on each level equal to $12, 8$ and 2. We found that this worked better for this particular data-set than running the optimization until absolute convergence is achieved. The reason behind this is that some sequences in the data set depict scenes that do not adequately constrain the solution. In these cases we gain much more in terms of absolute trajectory error from stopping early, than we lose at not running the optimization to convergence in all other scenes.

What we mean by an adequately constrained solution is that there needs to be enough variation in the geometry in the scene to lock down each of our six degrees of freedom (related to our six parameters). For example, given only a plane (such as a large featureless wall or closely cropped view of a table), we can only lock down three degrees of freedom. In-plane rotation and translation in two directions is still possible. In these cases, the matrix that represents the system of equations to be solved during the registration would be close to singular (rank-deficient). A more flexible solution, that looks for these under-constrained cases specifically, could be devised based on this knowledge. The result of the camera pose estimation can be seen in Fig. 5.1 and 5.2. Statistics about the errors computed along the trajectory are shown in table 5.1. The graphs and numerical results were produced by a script, provided along with the data-set.

Apart from the number of iterations in the registration algorithm, we must mention some other parameters that influence the system performance. The number of voxels storing the SDF was set to $200 \times 200 \times 200$ and the resolution, τ, equal to $2cm$. This means that the map created and used for localization corresponds to a metric space of $4m \times 4m \times 4m$. Lower errors are possible, given higher resolutions, more cells and more iterations on the fine scale. All at the

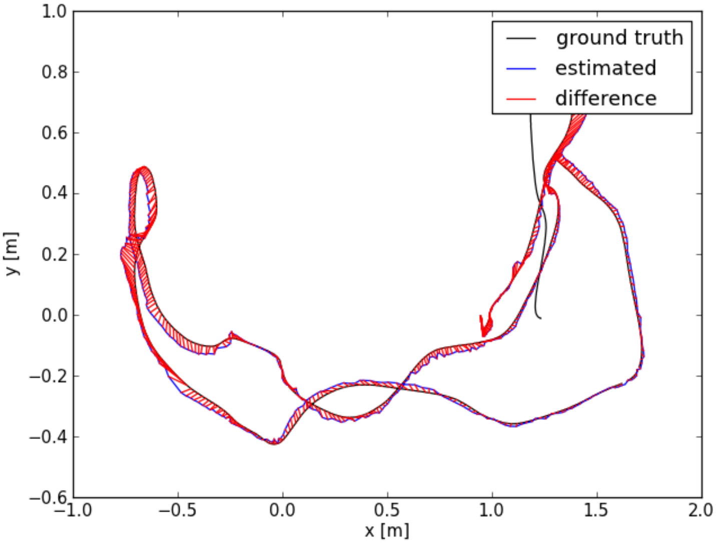


Figure 5.1: The absolute trajectory error is the integral of all the relative errors in translation made throughout the entire trajectory. The estimated poses are centred over the ground truth to account for differences in initial starting points.

expense of computation time, however. The outlined configuration produces an algorithm that runs reliably at 3 frames per second (fps) on the 4 cores of an Intel Core i7 Q720 processor. The following paragraphs give a quick review of similar algorithms and their reported performance on the same data-set.

- RGBD-SLAM [10] Uses a random sample consensus [11](RANSAC)-based registration to estimate poses and builds a map by storing surface measurements as point-clouds with associated camera poses. The final estimate of the camera poses and map relies on a post-processing step of global optimization over camera poses, using g2o [20]. The reported running time is similarly to ours, 2.86 fps with parallel execution on a quad-core CPU. The reported errors on the same data-set are, for relative translation: $e_{rmse} = 0.049m$ and relative rotation $e_{rmse} = 2.43°$. Since global optimization is intended as a post-processing step, sensor data must be kept and therefore tends to accumulate over time.

- NDTF-registration [1], is an algorithm for frame to frame tracking that performs pose estimation based on generating normal distribution transforms around the neighbourhoods of corresponding points. It then mini-

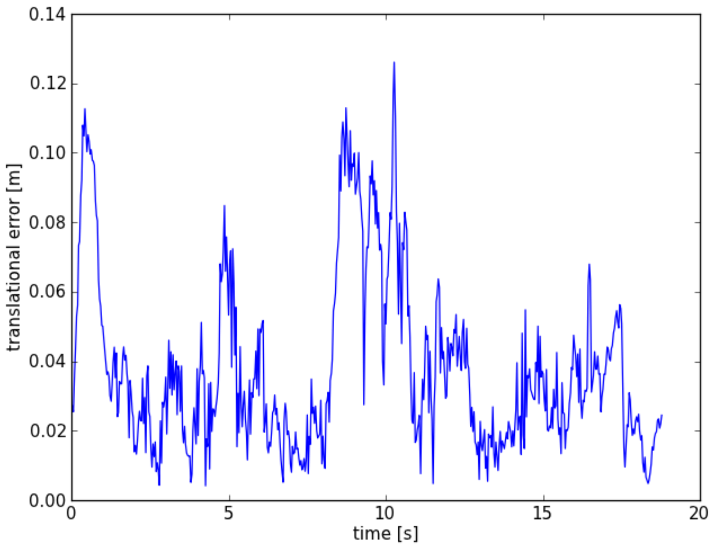


Figure 5.2: The relative pose error is computed as the difference between pairs of consecutive pose estimates.

mizes the distance between the resulting Gaussian mixture models. The reported execution speed is of 14fps on a single core of an Intel Q6600 processor, which is a lot faster than our algorithm. The reported relative translational errors are; $e_{mean} = 0.0165m$, $e_{median} = 0.0122m$. The relative rotational errors are $e_{mean} = 1.1567°$ $e_{median} = 0.909°$.

- Visual Odometry using RGBD [33] is an algorithm for frame to frame tracking that, instead of comparing the distances between points in space, perspective-projects both sets of points into an RGB image. The registration is done by minimizing the difference in brightness between corresponding points in this image plane. The reported frame-rate of their algorithm is of 12.5fps on one core of an Intel Xeon E5520 CPU. The reported relative translational error is $e_{median} = 0.0053m$. The relative rotational error is $e_{median} = 0.0065°$.

- Generalized ICP [32] (GICP) Is a method that performs registration by extending the iterative closest point (ICP) algorithm to include a point-to-plane distance metric, that relies on the estimate of the surface orientations. Generalized ICP is, in the above cases, referred to as the current state of the art, and therefore a good base for comparison. The following values are repeated from [33], tested on the same data-set as our algo-

> rithm. The reported relative translational error is $e_{median} = 0.0103m$. The relative rotational error is $e_{median} = 0.0154°$. The standard GICP implementation takes 7.52 s per match on one core of an Intel Xeon E5520 CPU.

The results show that we are on a similar level of performance as RGBD-SLAM, both in terms of speed and accuracy. This is likely due to the common maintenance of a map structure, which effectively lowers the frame-rate of RGBD-SLAM, as well as ours. In RGBD-SLAM, a volumetric occupancy map is obtained as a post-processing step (after the global optimization). In our case, we can obtain a similar map immediately, by thresholding the SDF at some arbitrarily small distance. Similar to the findings of [25], we note that since we compute a weighted sum of inputs when fusing data into our model we need not store sensor data, meaning that our algorithm can run indefinitely, without needing to discard information. The weighted summation also means that small changes in the scene will be averaged out to represent more recent configurations. Such a mechanism is not easy to implement in a point-cloud based world representation.

Although the relative pose errors and computational time are lower for the remaining algorithms (with the exception of the slower GICP), it would be interesting to also compare the absolute trajectory errors. As can be seen from our results, although the absolute trajectory error increases at times, the availability of a map allows us to reduce this error as the camera re-observes the scene. In pure frame-to-frame tracking applications, this is not possible. A single bad pose-estimate in an otherwise good camera-tracking sequence may therefore have large, undesirable effects on the absolute position estimate.

To test how well our system performs with missing data we can chose to input every other, every third, fourth etc. frame and measure the error to see how much it increases. The reason for doing this is that, unless provided with serious brute-force computing power, the execution time of this algorithm will not be on par with the frame-rate of a typical video-capture device. This limits the field of application to either slow-moving cameras or off-line scene-reconstruction/pose-estimation. In Fig. 5.3 we see how the relative translational and rotational error behaves when an increasing number of frames are omitted. Note that here we use the same fixed number of iterations as before, to make the comparison more meaningful. The increase in error is obviously non-linear with the skipping of frames.

Naturally, we would expect to require more iterations to reach the same quality of registration when the baseline between measurements is larger. This presents a bit of a dilemma, however. If we are having trouble keeping up with the inflow of data, should we spend more time, to attempt to achieve a good registration, on each individual frame, or should we settle for sub-optimal registration in favour of processing more data which will be closer together?

Our experiments with reconstruction in an on-line scenario seem to indicate that the truth is closer to the latter alternative.

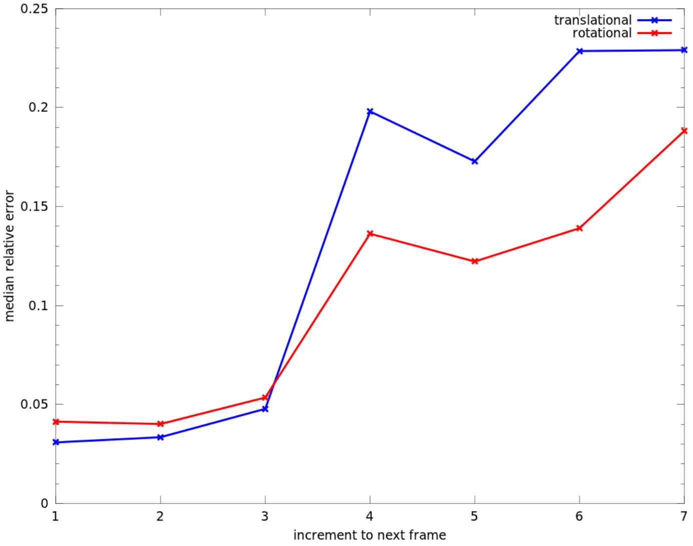


Figure 5.3: Rotational and translational error increase with loss of data. The peak in error that occurs when using every fourth frame is due to a particularly unfortunate combination of inputs, and not a feature of the algorithm.

5.1.1 Failure modes

The scenarios under which we have found the localization to be most error-prone are listed here

- Insufficient constraints. The geometry in the scene does not provide constraints for all six degrees of freedom. To solve this, additional data would be needed, such as texture. This failure can be detected by the near-singularity of the equations that need to be solved during the registration.

- Repetitive patterns. Geometrically identical objects, with dimensions and spacing that roughly correspond to the inter-frame camera motion are very problematic. In this case it is very likely that the solution gets stuck in a local minima. If there are other, large structures in the scene, they may still drive the solution to a correct registration. If the repetitive elements are dominant, even with color information, it would be a hard problem to solve. A possibility is to hot-start the solution of the next iteration registration with some “inertia” from the previous step, but this introduces a smoothness assumption that may not always hold.

- Insufficient overlap. This can be caused by too rapid camera motion leaving too few surfaces, too far apart to achieve a good registration. Another cause for this scenario can be related to the sensor technology. The Kinect camera typically returns very few measurements in cases where highly specular or brightly illuminated surfaces are present.
- A non-static environment. Although the scene reconstruction is able to filter out noise and even large structural changes, over time. It depends on most of the scene being mostly stationary in order to gauge the relative motion of the camera with respect to the environment.
- Small features. If the geometric features in the scene are smaller than the voxel resolution used to represent the SDF, we are again faced with insufficient constraints.
- Although our system displays some robustness to noisy data, prolonged motion blur and distortions due to electronic shutter effects are likely to cause failures too.

5.2 Mapping

This section shows some qualitative results of mapping (or 3D reconstruction). Since the SDF representation stores distances rather than occupancy information it is possible to represent surfaces that lie on a linearly interpolated position between measurements. This allows for the reconstructions of surfaces at arbitrary positions within voxels. However, since we always need at least one negatively signed measurement behind the surface, there is a lower limit on the thickness of the geometry that can be represented. The Figures 5.4 and 5.5 show a mapping of a kitchen environment and a sample from the associated SDF. Here we use the same convention as we established earlier, with brightness indicating distance, color indicating sign and dark regions are labelled as unseen.

The colors in Fig 5.4 are obtained by first using the sphere-tracing algorithm, from a virtual camera, (Algorithm 1) to find a surface intersection. At that point the gradient is computed, using equation (2.4). Since this produces a vector in $\mathbb{R}^3$ we we can associate each of its component with one of the colors red, green and blue. The effect of this is that surfaces facing the similar directions will be visualized with similar colors. Since the color space is limited to positive values, we disregard the sign of the gradient vector’s elements.

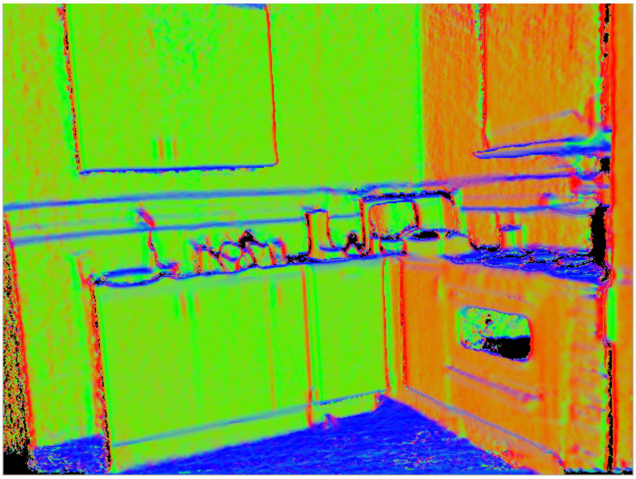


Figure 5.4: A Kitchen. The reconstruction is performed from a single point of view with small panning movements of the camera. This is already sufficient to smoothen surfaces considerably.

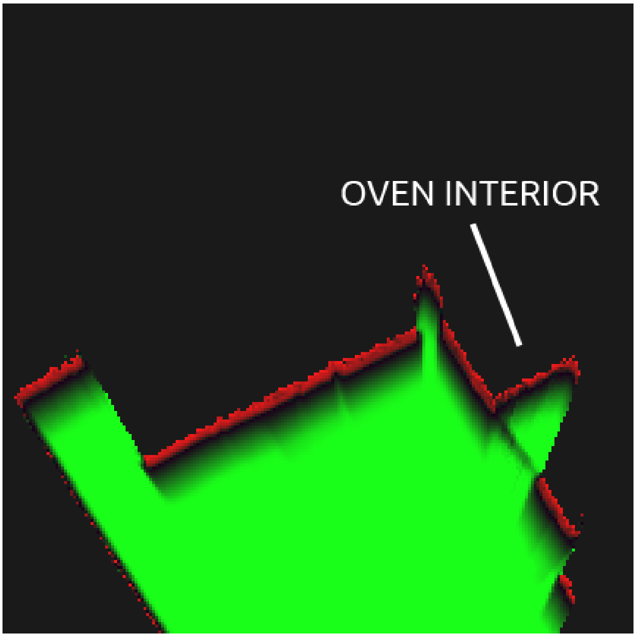


Figure 5.5: By taking a two-dimensional slice from the three-dimensional voxel grid that represents the reconstruction of the kitchen, we can see how the surfaces are defined at the boundary between positive and negative distances.

The remaining figures in this section show further examples from on-line reconstructions of objects and environments, using a hand-held camera, at different scales and resolutions.

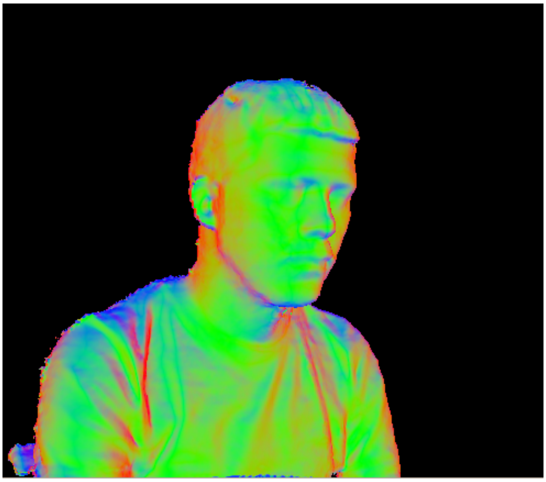


Figure 5.6: A possible application for 3D template-matching is facial recognition.

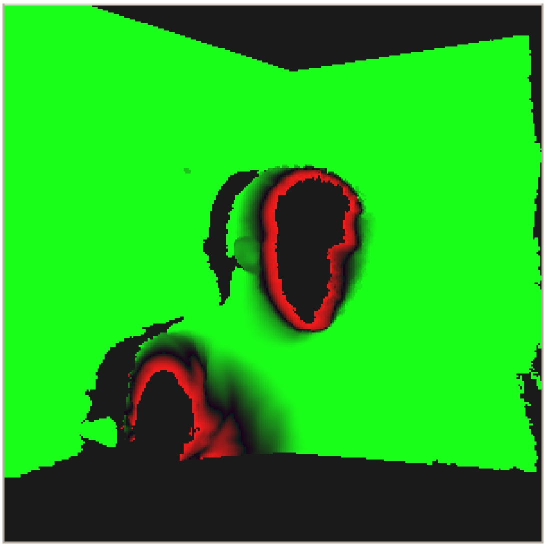


Figure 5.7: A two dimensional slice from the SDF voxel grid, taken in plane orthogonal to the viewing direction.

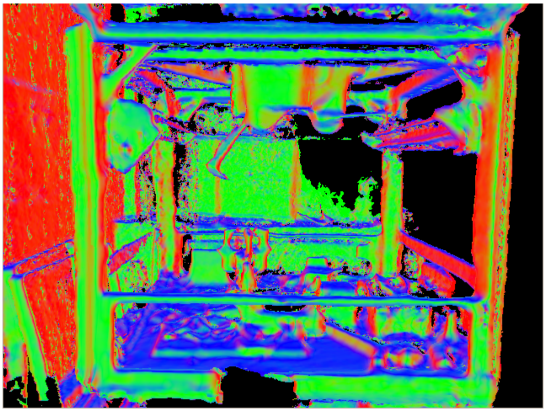


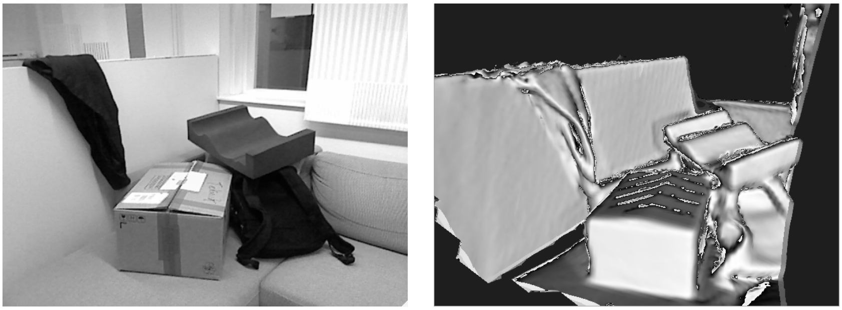
Figure 5.8: Although this scene seems more advanced due to its many details, the fact that it has surfaces in all main orientations and is abundant in non-repetitive geometric structure makes it easier to compute good estimates of the camera pose.

The difference between reconstructing an environment the size of an entire kitchen and a single flower pot is that the resolution (τ), the maximum and minimum values stored in the SDF (D_{max}, D_{min} can be set to much smaller values in the latter. The smaller resolution (and smaller D_{min}) mean that we can better represent fine details and thinner objects. The smaller value for D_{max} has an effect on ray casting (see algorithm 1), making it both slower and more accurate, since we have a lower risk of over-stepping surfaces where the line of sight distances are too large, relative to the angle of the ray. However, D_{max} also influences the maximum translation that the localization algorithm can cope with, since points offset by more than this value will be considered to be in completely empty space. Note that we can still have a large D_{max}, and use the c parameter of the Tukey M-estimator (3.26) to discard points that have too high residuals.

A possibility is to completely decouple the localization and mapping by using separate SDFs for each task. We feel that this somehow defeats the purpose of the proposed solutions here, but it might be a justifiable solution when reconstructing e.g., small objects with a lot of repetitive structure. In such a case, cues from the background (represented at lower resolution, perhaps) will be necessary to achieve a good result.

5.3 Object Detection

Figure 5.9 shows a photo of an environment and a partial reconstruction. The target object is the large cardboard box, for which we have synthetically



(a) Simple scene containing an object of interest

(b) Partial reconstruction of the scene

Figure 5.9: The environment in which we are interested in detecting an object. Here, the object of interest is the large cardboard box.

generated a template based on measurements of its length, height and width. The template object is a set of 3D points, describing the surface of a rectangular prism of the same size as the box. It includes all 6 sides of the object, since no occlusions are assumed to be known in advance.

Initializing the template at 64 uniformly spaced locations throughout the volume and computing the residual of the final registration at each, produces the red graph seen in 5.10. The residual is calculated, relative to the value that would be obtained if each point on the template evaluated to D_{max} within the SDF. Of the 64 locations at which the template was initialized, 33 were skipped due to being too far from any surface.

The blue line, overlaid atop the residuals, indicates how big a fraction, of the total number of points on the template, that was useful. By "useful" we mean that, when queried in the SDF, these returned returned a value other than D_{max}. In other words, the blue line gives a measure of how many points were participated in the final registration. Points that were not used, were either in an empty or unseen region.

In this experiment, the pose for which the *lowest* residual is produced, is shown in Figure 5.11. We see that it corresponds to the correct location of the object of interest. This need not always be the case, however.

One interesting feature to observe about the best-matched attempt (marked with a small black circle in the graph), is that the number of points that are in the truncated region i.e., returned a value of D_{max} when queried in the SDF, and the residual are very nearly symmetric about 50%. In this particular case, it means that the points that did not match against a surface only failed to do so, because the surface where they *would* have matched was unseen. Whether or not a point is in unseen or empty space can be disambiguated by the weight used to update the SDF. Unseen regions remain with a weight equal to zero.

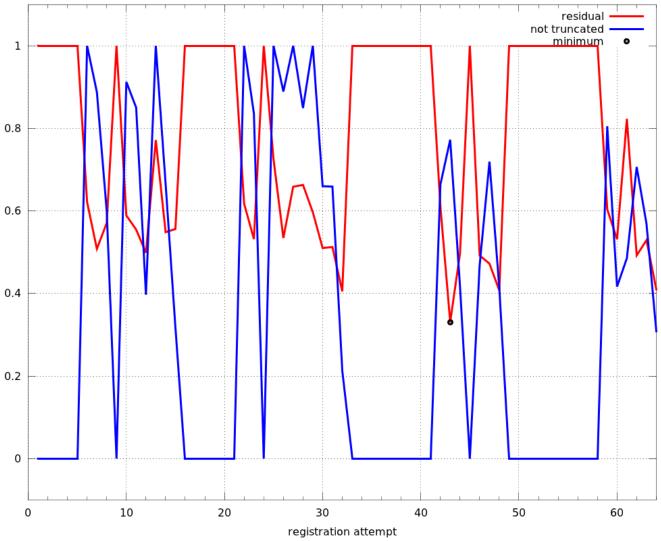


Figure 5.10: The graph shows the attempted matchings in 64 locations inside the reconstructed environment.

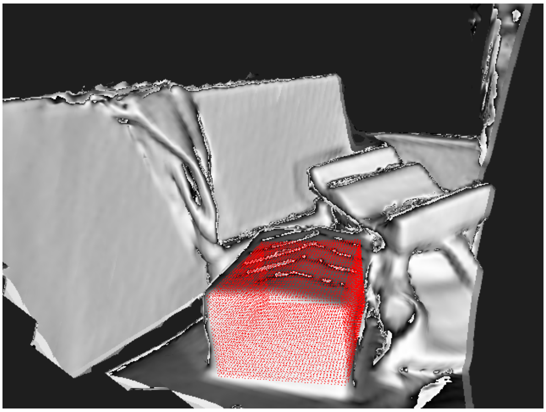


Figure 5.11: The figure shows the estimated pose of the template, inside the environment, at the pose where the minimal residual is produced. We note that it coincides with the object of interest.

Evaluating the registration by this criteria can help us disregard cases that might otherwise wrongly lead to the conclusion that we have successfully

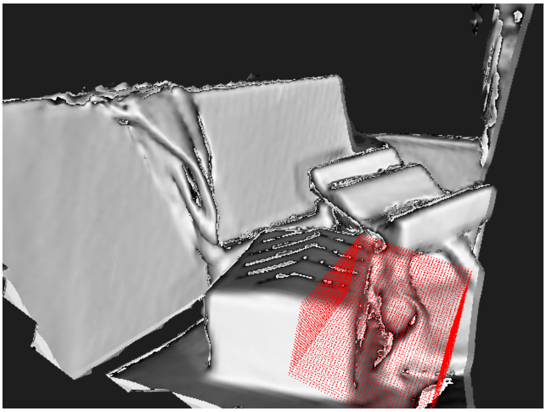
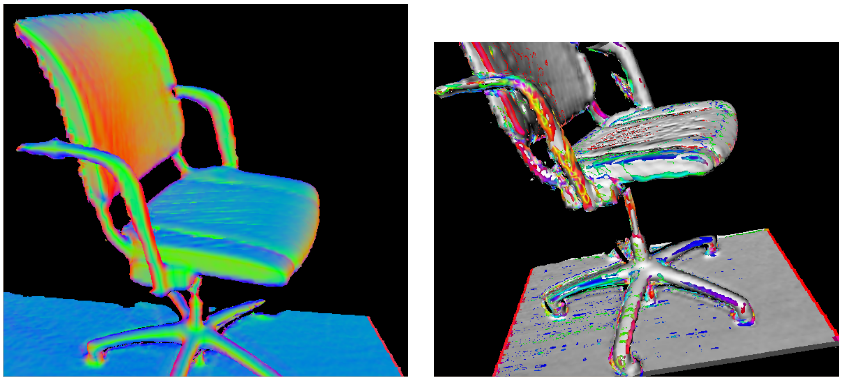


Figure 5.12: The figure shows a situation in which the registration converges at a local minimum near, but not quite coincident, to the target. Most registrations converge to erroneous states.

detected the object. Even through this analysis, we can still not safeguard against some unfortunate situations, such as the template of small box being registered into the corner of a much larger box in the environment, or objects that have similar geometry in general.

5.3.1 Finding salient features

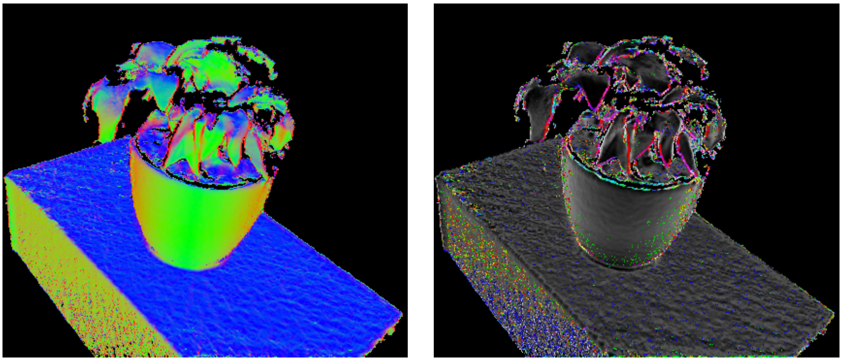
It is apparent that the testing of many possible candidate locations is somewhat inefficient. The results shown in this section have also revealed the that a registration may be inconclusive in determining the presence and type of a particular object. This motivates an interest in computing features about our environment. As an experiment, in search for locations at which to possibly compute descriptive features, we evaluate the surface curvature according to the method defined in section 2.2.4. Applying the same relationship between color and orientation as we have done before, and disregarding curvatures below a certain threshold, we produce the results seen in Figs. 5.13 and 5.14. As stated in Chapter 4, a common choice for a location to compute features in 2D images is typically at "corners". We can define a corner in 3D as being regions where surfaces exhibit strong curvature in more than one direction. In the figures of this section, these are the regions where one color transitions into another.



(a) reconstruction of a chair, color related to surface orientation

(b) estimated curvature, color related to orientation of folding

Figure 5.13: The low voxel resolution makes for large differences in orientation when evaluated at neighbouring points. This results in high curvature values.



(a) reconstruction of a potted plant, color related to surface orientation

(b) estimated curvature, color related to orientation of folding

Figure 5.14: The higher voxel resolution makes for small differences in orientation when evaluated at neighbouring points, highlighting only the sharpest of edges. The small cell-size also leads to noise at a finer scale in the curvature estimation. The plant is part of the same data-set as used for the localization benchmark.

Chapter 6

Conclusion

6.1 Results and Contributions

This work contains a study of the signed distance function (SDF) and the truncated signed distance function as implicit 3D surface representations. The concepts exposed in this study are explained in sufficient depth to serve as a reference for the reader interested in making their own implementation or seeking to better understand work currently done in the field of computer vision and robot perception. We show the results of reconstructing different scenes in this representation, both from a single point of view and by adequately combining data from several vantage points. We show that the latter results in a vast improvement in the accuracy of the reconstructed surfaces.

Another major part of this thesis is the development and multi-threaded implementation of an efficient convex optimization scheme that uses the scene representation directly as part of the estimation of camera poses. Directly using a truncated SDF for on-line pose estimation and scene reconstruction is, to the best of our knowledge a novel contribution. The use of the SDF as a 3D representation has the advantage of both offering a good approximation to the position and orientation of surfaces and, in registration, providing an intuitive and simple objective to minimize. Furthermore, the minimization can be done fast, without the need for explicitly matching corresponding points while being robust to outliers. We identify the discontinuities that occur when considering a truncated SDF representation, and offer solutions to avoid numerical instability when using it for registration.

It is worth noting that during the elaboration of this thesis, a sizeable amount of time was devoted to creating an efficient multi-core implementation of all the system components, to be integrated in the ROS package maintained by the Mobile Robotics & Olfaction lab [29] at Örebro University. This effort has led to insights in memory management and execution scheduling in a parallel programming paradigm. The resulting implementation is fast enough, even in on-line setting, to cope with camera movements that can comfortably be performed by a human operator.

The final section of this thesis is dedicated to an application to object detection. The presented methods represent a preliminary result, which highlights that we are still far from being able to perform well in real-world applications. The evident scalability issues in brute-force object detection approaches motivate a future interest in applying feature descriptors in the SDF space. To this end we present a method for finding salient features in the geometry that may be used to guide the computation of descriptors to points that are more meaningful than the rest.

6.2 Future work

To be able to effectively perform object detection, it would be of great use to have feature descriptors that could narrow down the number of candidate poses and locations to test for a potential registration. Having a good estimate of the surface position, its orientation and curvature could provide a good starting point for shape-based feature descriptors. Enriching the current scene representation with information about color (available from the sensor employed in this thesis) should provide additional discriminative power.

Another interesting field of research that could be explored using a distance-based scene representation is that of complex physics simulation, especially where vector fields and gradients in the empty space around objects are of importance.

References

[1] Henrik Andreasson and Todor Stoyanov. Real Time Registration of RGB-D Data using Local Visual Features and 3D-NDT Registration. In *Semantic Perception, Mapping and Exploration Workshop (SPME)*, May 14 2012. (Cited on page 43.)

[2] ASUS. Xtion pro (live) motion sensor. web, accessed 2012/06. http://www.asus.com/Multimedia/Motion_Sensor. (Cited on page 12.)

[3] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008. (Cited on page 40.)

[4] J.R. Bergen, P. Anandan, K.J. Hanna, and R. Hingorani. Hierarchical model-based motion. pages 237–252, 1992. (Cited on page 31.)

[5] Liefeng Bo, Xiaofeng Ren, and Dieter Fox. *Depth Kernel Descriptors for Object Recognition.* 2011. (Cited on page 39.)

[6] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: binary robust independent elementary features. In *Proceedings of the 11th European conference on Computer vision: Part IV*, ECCV'10, pages 778–792, Berlin, Heidelberg, 2010. Springer-Verlag. (Cited on page 40.)

[7] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. 1996. (Cited on pages 2, 11, and 19.)

[8] Andrew Fitzgibbon Department and Andrew W. Fitzgibbon. Robust registration of 2d and 3d point sets. In *British Machine Vision Conference*, pages 411–420, 2001. (Cited on pages 2 and 34.)

[9] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localisation and mapping (slam): Part i the essential algorithms. *IEEE ROBOTICS AND AUTOMATION MAGAZINE*, 2:2006, 2006. (Cited on page 1.)

[10] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. An evaluation of the RGB-D SLAM system. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, St. Paul, MA, USA, May 2012. (Cited on page 43.)

[11] Martin A. Fischler and Robert C. Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Commun. ACM*, 24(6):381–395, June 1981. (Cited on page 43.)

[12] Sarah F. Frisken and Ronald N. Perry. Efficient estimation of 3d euclidean distance fields from 2d range images. In *Volume Visualization Symposia (VolVis)*, pages 81–88, October 2002. (Cited on page 24.)

[13] Sarah F. Frisken and Ronald N. Perry. Designing with distance fields. In *ACM SIGGRAPH*, pages 60–66, July 2006. (Cited on page 6.)

[14] Arnulph Fuhrmann, Gerrit Sobottka, and Clemens Gross. Abstract distance fields for rapid collision detection in physically based modeling. *International Conference on Computer Graphics and Vision (Graphicon)*, 2003. (Cited on page 5.)

[15] K. Fujiwara, K. Nishino, J. Takamatsu, B. Zheng, and K. Ikeuchi. Locally rigid globally non-rigid surface registration. In *2011 International Conference on Computer Vision*, pages 1527–1534, 2011. (Cited on page 2.)

[16] John C. Hart. Sphere tracing: a geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12:527–545, 1996. (Cited on page 5.)

[17] Benoît Jacob and Gaël Guennebau. Eigen, "a c++ template library for linear algebra: matrices, vectors, numerical solvers, and related algorithms". web, accessed 2012/06. http://eigen.tuxfamily.org. (Cited on page 36.)

[18] Andrew Johnson. *Spin-Images: A Representation for 3-D Surface Matching*. PhD thesis, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 1997. (Cited on page 39.)

[19] M.W. Jones, J.A. Baerentzen, and M. Sramek. 3d distance fields: a survey of techniques and applications. *Visualization and Computer Graphics, IEEE Transactions on*, 12(4):581 –599, july-aug. 2006. (Cited on page 5.)

[20] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A General Framework for Graph Optimization. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Shanghai, China, May 2011. (Cited on page 43.)

[21] Steven Lovegrove. *Parametric Dense Visual SLAM*. PhD thesis, Imperial College London, 2011. (Cited on page 26.)

[22] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. pages 674–679, 1981. (Cited on page 35.)

[23] Karljohan Lundin, Matthew Cooper, Anders Persson, Daniel Evestedt, and Anders Ynnerman. Enabling design and interactive selection of haptic modes. *Virtual Reality*, 11(1):1–13, March 2006. (Cited on page 5.)

[24] Microsoft. Microsoft kinect (for xbox 360 / windows). web, accessed 2012/06. https://www.microsoft.com/en-us/kinectforwindows. (Cited on page 12.)

[25] Richard A Newcombe, Andrew J Davison, Shahram Izadi, Pushmeet Kohli, Otmar Hilliges, Jamie Shotton, David Molyneaux, Steve Hodges, David Kim, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. volume 7, pages 127–136. IEEE, 2011. (Cited on pages 2, 11, 15, and 45.)

[26] Iñgo Quilez. Modeling with distance functions. web, 2008, accessed 2012/06. http://www.iquilezles.org/www/articles/distfunctions/distfunctions.htm. (Cited on page 6.)

[27] PrimeSense. Primesense 3d awareness sensor. web, accessed 2012/06. http://www.primesense.com. (Cited on page 12.)

[28] S. Sastry R. Murray, Z. Li. *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994. (Cited on page 26.)

[29] Mobile Robotics and Sweden Olfaction Lab at the Center for Applied Autonomous Sensor Systems (AASS) at Örebro University. Ros packages from the aass research center at örebro university. web, accessed 2012/06. http://code.google.com/p/oru-ros-pkg/. (Cited on page 55.)

[30] M. Ruhnke, R. Kümmerle, G. Grisetti, and W. Burgard. Range sensor based model construction by sparse surface adjustment. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 2012. (Cited on page 21.)

[31] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *The IEEE International Conference on Robotics and Automation (ICRA)*, Kobe, Japan, 05/2009 2009. (Cited on page 39.)

[32] A. Segal, D. Haehnel, and S. Thrun. Generalized-icp. In *Proceedings of Robotics: Science and Systems*, Seattle, USA, June 2009. (Cited on page 44.)

[33] F. Steinbruecker, J. Sturm, and D. Cremers. Real-time visual odometry from dense rgb-d images. In *Workshop on Live Dense Reconstruction with Moving Cameras at the Intl. Conf. on Computer Vision (ICCV)*, 2011. (Cited on pages 35 and 44.)

[34] Todor Stoyanov, Athanasia Louloudi, Henrik Andreasson, and Achim J. Lilienthal. Comparative Evaluation of Range Sensor Accuracy in Indoor Environments. In *Proceedings of the European Conference on Mobile Robots (ECMR)*, Sep 7–10 2011. (Cited on page 12.)

[35] J. Sturm, S. Magnenat, N. Engelhard, F. Pomerleau, F. Colas, W. Burgard, D. Cremers, and R. Siegwart. Towards a benchmark for rgb-d slam evaluation. In *Proc. of the RGB-D Workshop on Advanced Reasoning with Depth Cameras at Robotics: Science and Systems Conf. (RSS)*, June 2011. (Cited on pages 13, 24, and 41.)

[36] Carlo Tomasi and Roberto Manduchi. Bilateral filtering for gray and color images. In *ICCV*, pages 839–846, 1998. (Cited on page 15.)

[37] Marjan Trobina. Error model of a coded-light range sensor. Technical report, ETH, Zurich, 1995. (Cited on page 21.)

[38] Q.-Z. Ye. The signed euclidean distance transform and its applications. In *Pattern Recognition, 1988., 9th International Conference on*, pages 495 –499 vol.1, nov 1988. (Cited on page 5.)

[39] Zhengyou Zhang, Zhengyou Zhang, Programme Robotique, and Projet Robotvis. Parameter estimation techniques: A tutorial with application to conic fitting. *Image and Vision Computing*, 15:59–76, 1997. (Cited on page 34.)