LUND UNIVERSITY

Computational Methods for Reconstruction: Signed Distance Functions, Low Rank Models and Depth Sensors

Bylow, Erik

2016

Document Version:
Publisher's PDF, also known as Version of record

Link to publication

Citation for published version (APA):
Bylow, E. (2016). *Computational Methods for Reconstruction: Signed Distance Functions, Low Rank Models and Depth Sensors*. Lunds Universitet, Centre for Mathematical Sciences.

Total number of authors:
1

General rights
Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: https://creativecommons.org/licenses/

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Computational Methods for Reconstruction: Signed Distance Functions, Low Rank Models and Depth Sensors

Erik Bylow

LUND UNIVERSITY

Faculty of Engineering
Centre for Mathematical Sciences
Mathematics

Mathematics
Centre for Mathematical Sciences
Lund University
Box 118
SE-221 00 Lund
Sweden

http://www.maths.lth.se/

Licentiate Theses in Mathematical Sciences 2016:1
ISSN 1404-028X

ISBN 978-91-7623-768-7 (print)
ISBN 978-91-7623-769-4 (electronic)
LUTFMA-2033-2010

© Erik Bylow, 2016

Printed in Sweden by MediaTryck, Lund 2016

Preface

"I think you will end up with a doctor's hat on your head". That was one of the last things my grandfather told me the last time I met him in September 2010. Six years later I can present my Licentiate Thesis. The contents of the thesis is based on the following papers:

Main papers

- E. Bylow, J. Sturm, C. Kerl, F. Kahl, D. Cremers, Real-Time Camera Tracking and 3D Reconstructions Using Signed Distance Functions, RSS, Berlin, 2013.
- E. Bylow, C. Olsson, F. Kahl, Robust Camera Tracking by Combining Color and Depth Measurements, ICPR, Stockholm, 2014.
- V. Larsson, C. Olsson, E. Bylow, F. Kahl, Rank Minimization With Structured Data Patterns, ECCV, Zürich, 2014.
- E. Bylow, C. Olsson, F. Kahl, Robust Online 3D Reconstruction Combining Model Information and Sparse Feature Points, To be submitted, Lund, 2016

Subsidiary papers

- J Sturm, E Bylow, C Kerl, F Kahl, D Cremers, Dense tracking and mapping with a quadrocopter, UAV-G, Rostock, 2013.
- J Sturm, E Bylow, F Kahl, D Cremers, CopyMe3D: Scanning and printing persons in 3D, in Proc. GCPR, Saarbrücken, 2013.

Acknowledgements

I would like to thank my colleagues at the Centre for Mathematical Sciences and in particular the Computer Vision Group for all their support and help. I want to give a special thank to my supervisors Carl Olsson and Fredrik Kahl for fruitful discussions and help in general with all types of problems.

Last but not least I want to thank my family for their support during all times. I want to give a special thought to my late mother and grandfather, who both always supported me and my sisters' studies.

This work has been funded by the Swedish Research Council (grant no. 2012-4213), the Crafoord Foundation and Scientific Research Council, project no. 2012-4215.

Contents

1 Introduction 1
1.1 Content . . . 1
1.2 Organisation of the Thesis . . . 1
1.3 Contributions . . . 2

2 Rigid 3D Reconstruction 5
2.1 Introduction . . . 5
2.2 Basics . . . 6
2.2.1 Pinhole Camera Model . . . 6
2.2.2 Depth Cameras . . . 9
2.2.3 Representation of 3D Models . . . 10
2.2.4 Camera Tracking . . . 13
2.3 Related Work . . . 16

3 Estimating the Camera Pose and Creating a 3D Model 21
3.1 Updating a TSDF for a New Depth Image . . . 21
3.2 Estimating the Camera Pose Using Geometry Information From the 3D Model . . . 26
3.2.1 Results and Experiments . . . 35
3.3 Robust Estimation of the Camera Pose . . . 44
3.3.1 Invoking Color in the Camera Pose Estimation . . . 45
3.3.2 Efficient Use of Memory . . . 48
3.3.3 Experiments and Results . . . 49
3.3.4 Quantitative Results . . . 51
3.4 Combining Sparse and Dense Tracking . . . 51
3.4.1 Qualitative Results . . . 54

3.4.2 Quantitative Evaluation 55
3.5 Discussion 55

4 Low-Rank Approximation of Matrices 59
4.1 Introduction 59
4.1.1 Structure from Motion 59
4.1.2 Missing Data Problems 63
4.2 Developing the Convex Envelope 64
4.2.1 Missing Data 68
4.3 Optimization 73
4.3.1 Proximal Operator 73
4.3.2 Experiments 76
4.3.3 Discussion 85

Chapter 1

Introduction

Computer vision is a broad research area with several different fields within it. Each field has its own focus and applications, but all have in common that they are working with images. This thesis study mainly how 3D models can be created using special cameras known as depth sensors, but we also study how a matrix of low rank can be estimated. The latter problem occurs in several applications of computer vision, for example in Structure from Motion and denoising.

The former topic is less abstract and there are several applications for 3D models. The models themselves has a value and can for example be used in refurbishment. Another application is robotic where one want to use the 3D model so that the robot can navigate by its own in a room to complete its task.

1.1 Content

The first part of this thesis deals with 3D reconstruction from RGB-D cameras. In particular, we study how the camera trajectory can be robustly estimated. This is crucial in many applications, for example in robotics where the robot uses the camera pose for navigation. In the second part, we study low rank approximations of matrices, commonly used in several applications, not only in computer vision but other fields as well.

1.2 Organisation of the Thesis

Chapter 2 In this chapter we give a brief introduction to rigid 3D reconstruction. This chapter also contains basic information about 3D

reconstruction and computer vision, intended for the reader who is not so familiar with the topic. For example, the pinhole camera model is explained and we also describe how depth sensors works and different methods for creating 3D models.

Chapter 3 In this chapter the focus is on online camera pose estimation. We study how one can use the 3D model itself to estimate the pose of the camera by extracting the information in a Truncated Signed Distance Function. We also study how more information from the estimated 3D model can be used together with information in the images. The purpose is to make the trajectory estimation more robust and accurate.

Chapter 4 In this part we show how one can extract certain patterns in a measurement matrix to obtain a convex envelope of the rank function plus a data term. We show that this gives superior results compared to the nuclear norm, which is the standard way of relaxing the rank function.

1.3 Contributions

In Chapter 3, we show how the information in a Truncated Signed Distance Function, (TSDF), can be used to estimate the pose of the camera. Experimental results show that this leads to superior results compared to KinectFusion [23]. This part was made in collaboration with Jürgen Sturm (TUM), Christian Kerl (TUM), Fredrik Kahl (LTH) and Daniel Cremers (TUM). I did all the code and most of the writing of the paper. Further, in chapter 3 it is shown that the texture information in the 3D model can be used together with the distance information in order to compute the camera pose. This makes camera pose estimation significantly more robust compared to just using geometry. This method is then extended further by using the color and depth images separately to make the algorithm robust to situations with hardly any structure or texture. A thorough quantitative evaluation is also carried out. This was done together with Carl Olsson and Fredrik Kahl. I wrote all the code and wrote most of the paper. In the final part Chapter 4 the contribution is the derived convex envelope of a function consisting of a rank

term and a least squares data term. This was done together with Viktor Larsson, Carl Olsson and Fredrik Kahl. Viktor and Carl did most of the theory and writing of the paper. I contributed with writing some code and recording some datasets for the experiments.

Chapter 2

Rigid 3D Reconstruction

2.1 Introduction

The capability to reconstruct a scene from a set of images has been one of the major challenges in computer vision. It is still an active research area and many challenges remain. In computer vision this research field is known as Structure from Motion (SfM) and in robotics as Simultaneous Localization and Mapping (SLAM).

In SfM, the pipeline is typically to first find a set of key-points in different images and to match these. Then one computes the optimal camera poses and 3D scene geometry using bundle-adjustment to get global consistency. This results in a sparse 3D model, where only the detected key-points can be reconstructed.

To create dense 3D models using monocular cameras, an approach is to use stereo and try to estimate depth maps from image pairs. To create each depth map one needs to optimize an energy function for each image pair. Often these energy functions require some form of regularization to decrease noise and to get more smooth surfaces.

However, the camera technology has evolved as well. Today there are different so called depth cameras such as the Microsoft Kinect and the Asus Pro Live Sensor available on the market. Thanks to these depth cameras, it is possible to create dense 3D models using the depth images generated from these cameras. The advantage compared to techniques like SfM is that one gets the depth for each pixel, resulting in a dense point cloud with correct scale. Disadvantages with these depth cameras is that they do not work outdoors when structured light is used and the range is limited to a couple of meters.

Nevertheless, these new cameras open up for new applications for

indoor environments. The focus of the first part of the thesis is on how we can robustly estimate 3D models using these sensors.

The motivation behind this is that to do accurate 3D-reconstruction from a set of images, it is necessary to have an accurate estimation of the camera pose. For several applications, for instance in robotics, the robot can use the current position and 3D model to localize itself in the room. If the camera position is poorly estimated, then the robot will make an incorrect estimation of its position and might not be able to perform its task.

In other applications such as refurbishment, we might want to measure the dimensions of our room. This could then be used to see how a new sofa would fit into a living room. This requires an accurate 3D model to get consistent measurements. The main contribution with this part of the thesis is that we develop a robust and accurate method for tracking the pose of the depth sensor in real-time. By real-time we mean that the pose is estimated and the model is created as we acquire new depth- and color images.

2.2 Basics

In this section concepts from computer vision is presented. We will describe some camera models and illustrate the idea behind some different well-known methods for computing the camera pose and representing the 3D model.

2.2.1 Pinhole Camera Model

The pinhole camera model is probably the most common model of a camera. Let $\mathbf{x}$ be a point in the world, seen by the camera. Then under the pinhole camera model the point is projected onto the image plane by following the ray between the point $\mathbf{x}$ and the camera center $\mathcal{O}$. This is illustrated in Figure 2.1. The projected coordinates for a 3D point (x_1, x_2, x_3) are $(\frac{x_1}{x_3}, \frac{x_2}{x_3}, 1)$, assuming the image plane is a unit distance from the camera center along the z-axis. We assume that the image plane is parallel with the xy-plane, and we are only interested in the coordinates $(\frac{x_1}{x_3}, \frac{x_2}{x_3})$ on the image plane, as illustrated in Figure 2.2. In

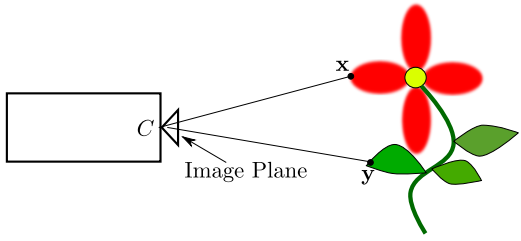


Figure 2.1: The points $\mathbf{x}$ and $\mathbf{y}$ on the flower are projected onto the image plane along the ray between the point and the camera center $\mathcal{O}$.

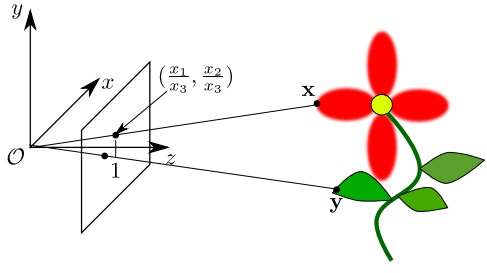


Figure 2.2: When the center of the image plane is a unit distance from the optical center, then the projection onto the image plane is just the coordinates $(\frac{x_1}{x_3}, \frac{x_1}{x_3}, 1)$. The coordinates on the image plane are $(\frac{x_1}{x_3}, \frac{x_2}{x_3})$.

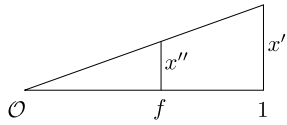


Figure 2.3: If a 3D point has the x-coordinate x' on a unit distance from the origin, then the corresponding coordinates on the image plane will be fx'.

practice, the distance between the image plane and the camera center will of course be significantly less than, say 1 meter, if that is 1 unit. The distance between the camera center and the image plane is known as the focal length f, and by similar triangles it can be seen how to go from $(\frac{x_1}{x_3}, \frac{x_2}{x_3})$ to image coordinates. By looking at Figure 2.3 we see that

$$\frac{x''}{x'} = \frac{f}{1} \Leftrightarrow x'' = fx'. \tag{2.1}$$

Since $x' = \frac{x_1}{x_3}$, the coordinate for the 3D point on the image plane will be $x'' = \frac{fx_1}{x_3}$. Similarly, the projection of the y-coordinate can be computed the same way. To go from ordinary coordinates to pixels we simply translate the projected point $(\frac{fx_1}{x_3}, \frac{fx_2}{x_3})$ by (c_x, c_y), where c_x is half the width of the image plane in pixels, and c_y half the height in pixels. To get the pixel coordinates of the 3D point x we compute

$$(p_x, p_y) = (\frac{fx_1}{x_3} + c_x, \frac{fx_2}{x_3} + c_y). \tag{2.2}$$

In practice, the pinhole camera model will not be the best model. Instead we use the CCD camera model with different focal lengths in the x- and y-direction. More information about different models can be found in [14]. We now define a function that takes a 3D coordinate to a pixel coordinates:

Definition 2.2.1. Let $\pi : \mathbb{R}^3 \mapsto \mathbb{R}^2$ be the function that takes a 3D point to pixel coordinates:

$$\pi(\mathbf{x}) = (\frac{f_x x_1}{x_3} + c_x, \frac{f_y x_2}{x_3} + c_y). \tag{2.3}$$

The process of 3D reconstruction can be thought of as inverting the projection on the image. Assume we know the projection coordinates, the calibration of the camera, how do we get back the original 3D point? If the the pixel coordinates (p_x, p_y) are known, we simply do the reverse calculations and get

$$x_1 = \frac{p_x - c_x}{f_x} t \tag{2.4}$$

$$x_2 = \frac{p_y - c_y}{f_y} t \tag{2.5}$$

$$x_3 = t, \tag{2.6}$$

where $t \in \mathbb{R}$ is an arbitrary scalar corresponding to depth. However, we have an ambiguity here because we can take any depth t $\in \mathbb{R}$ and we will get a 3D that projects onto the same pixel coordinates (p_x, p_y).

When using depth sensors, we get around this ambiguity since we get the actual distance to the object in the world.

2.2.2 Depth Cameras

Depth cameras differ from regular cameras in the sense that it generates two different images, one ordinary color image and one depth image. Each pixel in the depth image contains distance information between the object and the camera. In Figures 2.4a and 2.4b a depth image with its corresponding color image are shown.

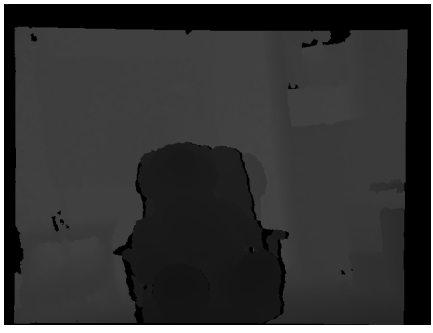
Let us denote the depth image by I_d, then we can for each pixel (p_x, p_y) read the depth value

$$z = I_d(p_x, p_y). \tag{2.7}$$

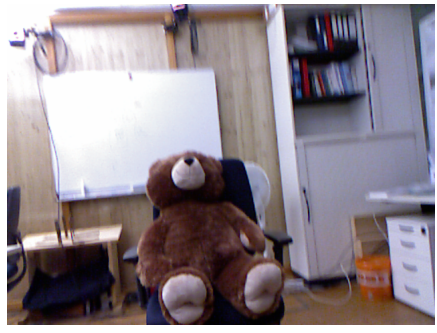
Using this we resolve the ambiguity of the depth in equation (2.6). We can define the following function that takes a pixel (p_x, p_y) to its 3D coordinates:

Definition 2.2.2. Let $\rho : \mathbb{R}^2 \times \mathbb{R} \mapsto \mathbb{R}^3$ be the function that transforms a pixel (p_x, p_y) to its 3D coordinates by

$$\rho(p_x, p_y, z) = (\frac{p_x - c_x}{f_x} z, \frac{p_y - c_y}{f_y} z, z), \tag{2.8}$$



(a) An example of a depth image



(b) Corresponding color image.

Figure 2.4

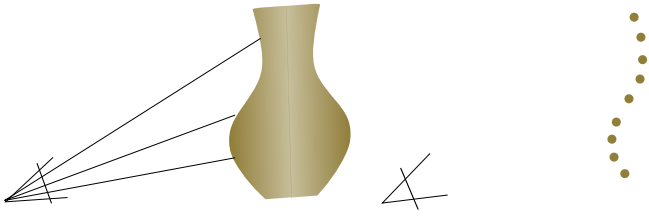


Figure 2.5: In the image to the left the depth sensor takes a depth image of the vas, to the right we can see how the point cloud could look like by reconstructing the 3D points.

where $z = I_d(p_x, p_y)$ and f_x, fy, c_x and c_y are intrinsic camera parameters.

An illustration of how this works is given in Figure 2.5, where the points to the right are reconstructed 3D-points of the vase.

2.2.3 Representation of 3D Models

Using the depth image alone, we get a point cloud. However, we would like to estimate a surface, not just a set of points. There exist several methods to do this. For example, one can use occupancy grids through octrees. Another popular method is to use a so called Truncated Signed Distance Functions, (TSDF), which represents the model surface implic-

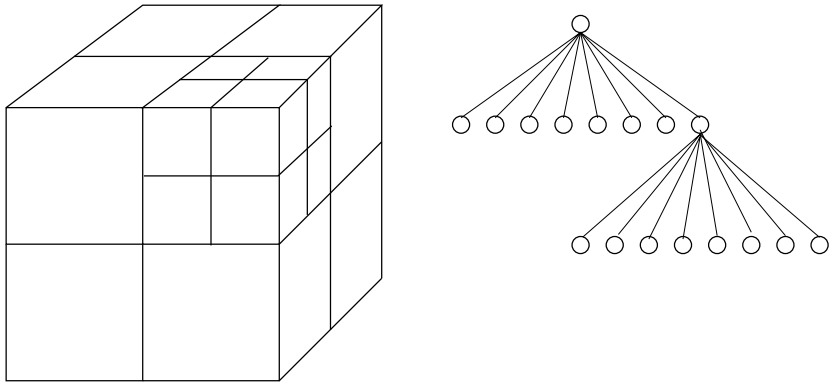


Figure 2.6: In an octree, each node has eight children, this can be used to subdivide space with more nodes in some parts than in others.

itly through the zero level-set.

Octrees

One method to represent a surface is to use a probabilistic occupancy grid as in [36], where the grid is represented via an octree. In short, an octree is a tree data structure where each parent has exactly eight children. The idea is that one can subdivide the space with high resolution close to the surface and with lower resolution where there is only empty space, as illustrated in Figure 2.6.

A cell can be either occupied or free. An occupied cell means surface intersects the cell and non-occupied means free space. This approach has several advantageous, one is that it is very memory efficient, which makes large scale representation possible. The other is that it models both free space and occupied space which is important for robotic applications. However, even though the octree representation is memory efficient and we can make the resolution high, the surface will be built up by non-smooth cubes. For applications where we might want good looking and smooth models, this might not be the best choice.

Signed Distance Functions

An alternative representation, which is commonly used in conjunction with RGB-D cameras, [34], [16], [23], [29], is to use a TSDF.

The basic idea of an implicit representation is illustrated in Figure 2.7. Looking at the figure, there is a red area and a blue area and a white circle between the two areas. This is an implicit representation of a circle where points outside the circle have a negative distance, (blue), and points inside the circle, (red), have a positive distance to the surface.

The theory for level set methods and surface representations through signed distance functions are thoroughly treated in [24]. Here we present some basic definitions from [24] to get a brief overview of signed distance functions.

Definition 2.2.3. Let $\partial\Omega$ be a surface in $\mathbb{R}^n$ and let $\phi : \mathbb{R}^n \to \mathbb{R}$ be an implicit representation of the surface. Then we define $\Omega^+ = \{\mathbf{x} \in \mathbb{R}^n : \phi(\mathbf{x}) > 0\}$ and $\Omega^- = \{\mathbf{x} \in \mathbb{R}^n : \phi(\mathbf{x}) < 0\}$ and $\Omega = \Omega^+ \cup \partial\Omega \cup \Omega^-$

Definition 2.2.4. A distance function $d(\mathbf{x})$ is defined as

$$d(\mathbf{x}) = \min(\|\mathbf{x} - \mathbf{x}_I\|) \quad \forall \mathbf{x}_I \in \partial\Omega, \mathbf{x} \in \Omega.$$

We can now define a general signed distance function:

Definition 2.2.5. A signed distance function ϕ is an implicit function ϕ with $|\phi(\mathbf{x})| = d(\mathbf{x})$ $\forall \mathbf{x} \in \Omega$. Thus, $\phi(\mathbf{x}) = d(\mathbf{x}) = 0$ for all $\mathbf{x} \in \partial\Omega$, $\phi(\mathbf{x}) = -d(\mathbf{x})$ for all $\mathbf{x} \in \Omega^-$ and $\phi(\mathbf{x}) = d(\mathbf{x})$ for all $\mathbf{x} \in \Omega^+$. Since d is a Euclidean distance, we have $\|\nabla d(\mathbf{x})\| = \|\nabla \phi(\mathbf{x})\| = 1$ for all $\mathbf{x} \in \Omega$.

The last property is typical for signed distance functions and it is not true for a general implicit function.

With this we have an idea of what a signed distance function is and how we can represent the geometry through it. The obvious drawback is that one needs to estimate the distance to the surface for each point in (a discretized) space. For a general 3D model, no closed form solution exists.

To achieve an approximation of the signed distance function, we use a uniform voxel grid and estimate the distance between each voxel and the surface. With this representation we can find the distance to the

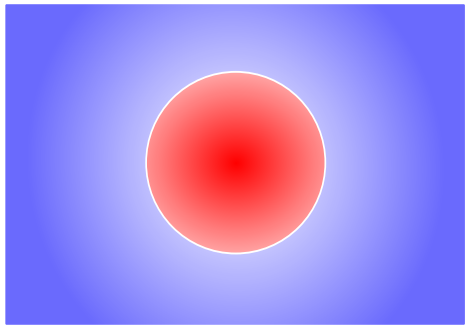


Figure 2.7: A circle (white) is represented implicitly through a signed distance function. The points with red color are inside the circle, having a positive distance, and the blue points are points outside the circle and they have a negative distance

surface for any point in the grid through interpolation. Typically one uses a uniform grid. That means we have 3-dimensional grid where the voxels are laid out equidistantly.

An advantage with this is that one can obtain smooth surfaces by averaging several measurements for each voxel. A drawback is that the uniform grid requires a lot of memory, which mostly represents empty space.

The challenge to estimate the distance between the voxels and the surface remains. In section 3.1 we present a method of how to estimate the signed distance function.

2.2.4 Camera Tracking

The problem of estimating the pose of the camera has been studied for a long time. Here two methods are described in order to give a basic understanding of how they work.

ICP

In the beginning of the nineties, [3] was published. It is a paper about how to register two point clouds. The technique is known as Iterated Closest Point, (ICP). The method has become a standard approach for

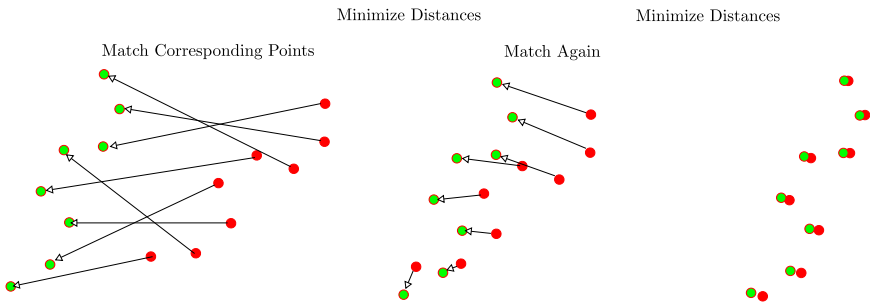


Figure 2.8: In ICP we have two point clouds and the goal is to find a transformation that takes the red point cloud to the green point cloud in an optimal way. One alternates between finding correspondences and minimizing the distance between these correspondences.

3D registration and there are almost infinitely many versions of it, a review can be found in [27]. In short, ICP aims to find a rotation and translation which transforms one point-cloud into another using minimal distance in some norm, often L_2, as illustrated in Figure 2.8.

From Figure 2.8 we can extract some key components of ICP. Given a point cloud and the correspondences between the points, we aim to minimize

$$E(R, \mathbf{t}) = \sum_{i=1}^{n} \|R\mathbf{x}_i + \mathbf{t} - \mathbf{y}_i\|_p^p \tag{2.9}$$

which means we are seeking the rotation and translations which minimize the sum of residuals, often $p = 1$ or $p = 2$.

With no noise and perfect correspondences this is easily solved. However, given two point clouds, finding good correspondences is a problem in its own. Typically we cannot expect to find a correspondence for all points, but just a subset of points and not all of these might be correct. To handle this we start with a set of correspondences, then solve (2.9). Then we can apply the transformation on the point cloud and recompute the correspondences and solve (2.9) again, until convergence. This is local optimization with no guarantee of finding the global optimum, but when the point clouds are close to each other, one often gets good solutions.

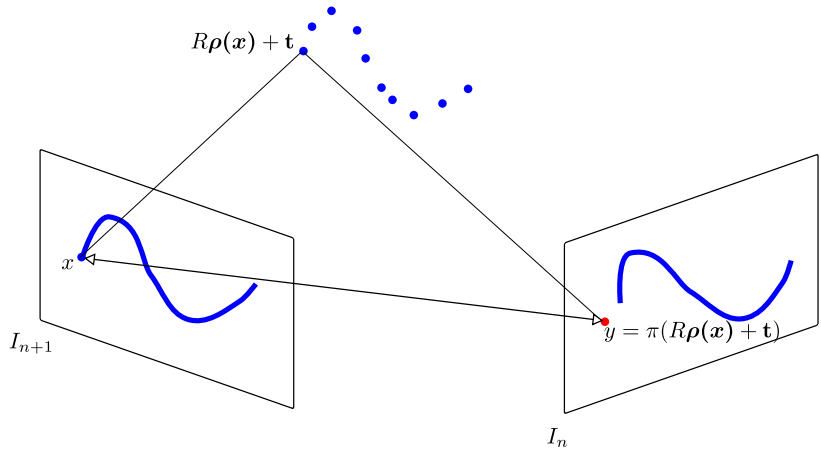


Figure 2.9: A pixel x is recomputed to 3D by $R\rho(x)+\mathbf{t}$ and then projected on to the image I_n and gets pixel coordinates $\pi(R\rho(x) + \mathbf{t})$. One then compares the intensity values at pixel x and $\pi(R\rho(x) + \mathbf{t})$.

There are several variations of this procedure, a common method is to estimate a normal to each point and minimizing the projection to the normal instead of the difference between the point pairs $R\mathbf{x}_i+\mathbf{t}-\mathbf{y}_i$, [10]. We would then minimize

$$\sum_{i=1}^{n} \|(R\mathbf{x}_i + \mathbf{t} - \mathbf{y}_i)^T \mathbf{n}_i\|_p^p. \tag{2.10}$$

This metric is known as the point-to-plane metric, and corresponds to minimizing the distance between the point $\mathbf{x}_i$ and the tangent plane at $\mathbf{y}_i$.

A general drawback of ICP is that most methods rely on the presence of varied geometry, meaning that if all points lies on a plane, there is no unique minimizer of (2.9) or (2.10).

Intensity Based Methods

Some other approaches which work well and have some nice properties have also been developed. The basic idea is that photo consistency between images shall be maximized. That is, if a 3D point for a pixel in one

image is projected onto another image, then the intensity in these two pixels shall be similar. We call the difference in intensity between these pixels for the intensity error. In particular [28] introduced a frame-to-frame tracking approach which uses both the depth image and the RGB image. This was later extended in [29] to handle more general situations.

Intensity based methods such as this are suitable for RGB-D cameras. The idea is to create a point cloud from one image. Then we project the points into the second image. The rotation R and translation $\mathbf{t}$ is assumed to be correct if the intensity error is 0. In practice perfect color matching cannot be achieved. Therefore the intensity difference is typically minimized instead.

This differs from ICP in that we are not working with aligning two point clouds, but instead we are trying to maximize photo consistency. As illustrated in Figure 2.9, we have a 3D point $\rho(x)$ obtained from the left image. Then we use the estimated relative transformation between image I_n and I_{n+1} and transforms $\rho(x)$ to the second camera frame. The point is then projected onto pixel y in I_n and the difference in intensity between pixel x in I_{n+1} and y in I_n is evaluated. This process can be formulated as the following energy function:

$$E(R, \mathbf{t}) = \sum_{i=1}^{m} \sum_{j=1}^{n} \|I_{k+1}(i,j) - I_k(\pi(R\rho(i,j,z_{ij}) + \mathbf{t}))\|^2,$$

where z_{ij} is the depth at pixel (i,j) in the depth image $I_d(i,j)$. With this technique we find the relative transformation between two frames.

The advantage with this approach is that we can use all color information available and need not to find corresponding points as in ICP. The disadvantage is that errors are quickly accumulated which can lead to poor results if we do not find a way of reducing these errors.

2.3 Related Work

As mentioned earlier, 3D reconstruction is not a new topic and there are many approaches to solve the problem of estimating the pose and the geometry. Both in SfM and in SLAM a lot of research has been done. Lately, since the advent of the Microsoft Kinect and the Asus Pro Live

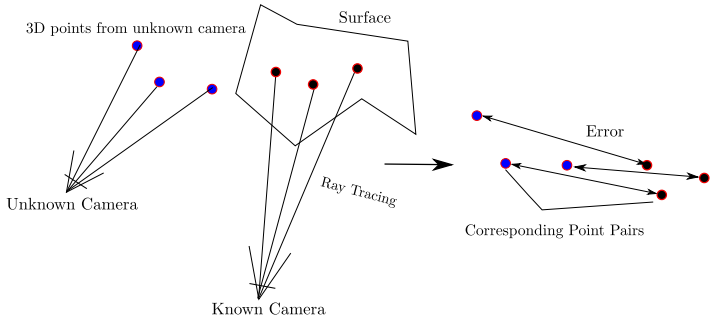


Figure 2.10: Given a new depth image, one takes the latest known camera and perform ray tracing to find points on the actual surface. With this global point cloud one can perform ICP to find the correct transformation between the last known camera position and the new unknown camera.

sensor, a very active research area has been how to create 3D-models using depth cameras.

The most famous work is probably KinectFusion [23], which was likely the first system capable of creating 3D models in real-time using these sensors. Their main contribution was that they demonstrated how a TSDF can be used to robustly track the camera movement for medium-sized reconstructions. To track the camera they use ICP and to create the 3D models, they use the method from [11]. In ICP [3], [27], one wants to align two point clouds. What makes ICP more robust in KinectFusion is that they render the 3D model directly. This results in a global point cloud which the new point cloud obtained from the new depth image is aligned to. An illustration of this is shown in Figure 2.10.

In [28], a different approach to estimate the camera pose is taken compared to KinectFusion. Instead of minimizing the geometric error between point clouds, one seeks to maximize photo-consistency between two consecutive images as described in Section 2.2.4. Here the pose estimation is independent of the model. Actually no model is estimated in [28]. This idea was improved in [17] and [18]. A drawback is that these methods easily drifts away. Drifting means that due to accumulated errors, the estimated camera pose deviates more and more from the true path. These intensity based methods do not require a 3D model to work, which makes it possible to use other methods to decrease drift such as

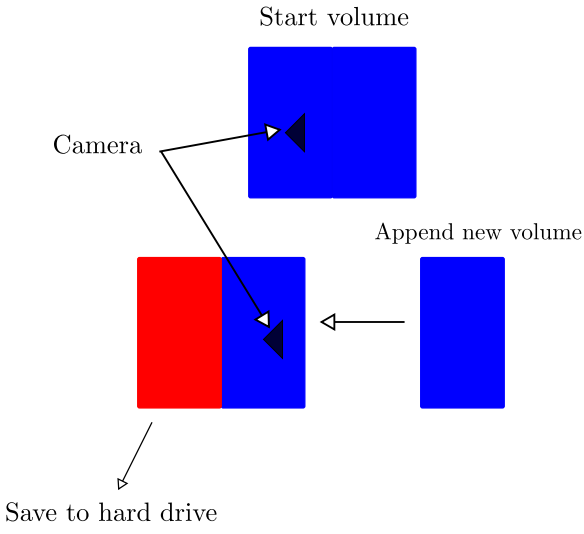


Figure 2.11: When the camera approaches the border of the volume, a new volume is created and appended to the active part. The other half is saved to the hard drive.

loop-closure and bundle-adjustment.

Another way to estimate the camera pose is the one by [12], where they find key points between pairs of images and use RANSAC to compute a relative transformation between the pair. These transformations are then added to a graph which optimizes the camera pose globally to reduce drift. This method has the advantage compared to ICP that it requires only keypoints to be found, so it can handle scenes with little texture as long as it can find keypoints. A drawback is that the 3D model cannot be created until the entire scene is recorded. To represent the 3D model, they use a probabilistic Octree, [36], which is created after all images has been recorded and the graph is globally optimized. The advantage with such a representation is that it is very memory efficient. However, the reconstructions are harder to make smooth since a voxel is either entirely surface or not surface. An advantage for robotic applications is that both occupied and non-occupied space is represented.

The approaches described above have been successfully used in other well-known methods such as Kintinuous [32], [33] and lately [35]. A

combination of the KinectFusion based ICP [23] and the intensity based methods by [28] is used to estimate the camera trajectory. The aim with these works is to create large-scale online reconstructions, that is, the model is created as the images are recorded. For large-scale methods one meets other challenges. Firstly, one must reduce drift and secondly the memory consumption must be reduced so that the entire model can be represented on a computer. To handle this [34] uses a rolling volume. This means one has a uniform grid as in KinectFusion and represents the surface with a TSDF. However, when reaching the border of this grid the other half is saved to the hard drive and a new empty grid is appended to the volume where the camera is, as depicted in Figure 2.11.

In [35] a new interesting approach is taken. There Surfels [25] are used to represent the model. Surfels are basically small surface elements that contain information about position, size, orientation and eventually texture. This makes it easier to recompute the 3D model online if drift is detected and adjusted for which is done in [35] with impressive results. To track the camera the KinectFusion based ICP is used together with an error term that takes photo consistency into account. This makes it robust to scenes where there is either only texture or structure.

Chapter 3

Estimating the Camera Pose and Creating a 3D Model

The main contribution with this part of the thesis is that we investigate how the model represented as a TSDF can be used to estimate the camera pose. Evaluation on benchmarks demonstrates that when using the global model in a different way than KinectFusion [23, 16] does, the estimated pose is even more robust. These conclusions are drawn by evaluating an open-source implementation of KinectFusion, known as KinFu [1], together with our method on publicly available datasets [30].

As described in previous sections, there are some different ways of creating 3D models and estimating the camera pose. To get an accurate and reliable 3D model, it is important to have a method that can estimate the pose accurately and robustly. In the first part of this chapter we show how the TSDF can be estimated with known camera position. Thereafter, we explore how the TSDF representation can be used to estimate the pose. Later we see that we can invoke more information to make the pose estimation more robust. In particular we investigate how to recover the pose when there is not so much geometry in the scene.

3.1 Updating a TSDF for a New Depth Image

As described in the introduction we can use a TSDF to represent the 3D model. Here we go into detail of how the grid is updated as we get new measurements for each new frame.

The goal is that each voxel shall contain the best estimation of the distance between the surface and the voxel. This is generally a hard problem and to get an exact signed distance function the constraint

$\|\nabla\phi\| = 1$ must be satisfied. Here we aim to do real-time 3D reconstruction similar to KinectFusion. Since the Kinect delivers images at a rate of 30 frames per second, one has approximately 33 ms to find the position of the camera and update the grid with new information. Therefore, more advanced methods to estimate the distance function are not suitable. Instead we follow the heuristic by [11], which is trivial to parallelize, allowing for a considerable speed up using a modern GPU.

To start with we have a voxel grid, which is a 3 dimensional discretization of a volume in space. Each voxel has a unique index $(i, j, k) \in \mathbb{N}^3$ and we refer to one voxel at index (i, j, k) as V_{ijk}. Each voxel stores data used to represent the distance function. The data in this work is:

- D - estimated distance to surface
- W - estimated weight of the measured distance
- R - estimated intensity in the red channel
- G - estimated intensity in the green channel
- B - estimated intensity in the blue channel.
- W_c - estimated weight for the color.

A data value for a voxel at (i, j, k) will be referred to with subscript ijk, for example the distance at voxel V_{ijk} will be denoted D_{ijk}. We set the origin of the global coordinate system to be in the center of the voxel grid. Since the distance between the voxels is known and the voxels are fixed in space and the origin of the global coordinate system is known, the global coordinates for a voxel V_{ijk} can easily be computed.

With the above definitions we can now describe how we can estimate the signed distance to the surface for each voxel. Assuming that we have the global position of the camera $\mathbf{C}_n$, i.e. we know the global rotation R and translation $\mathbf{t}$ of the camera, we can express the coordinates of the voxel V_{ijk} in the cameras frame of view by computing

$$\mathbf{x}_L = R^T \mathbf{x}_G - R^T \mathbf{t}, \tag{3.1}$$

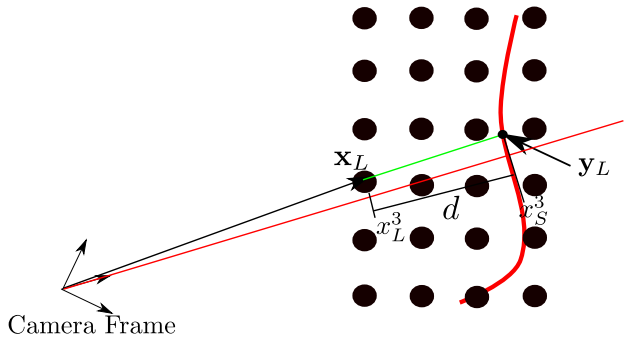


Figure 3.1: Instead of measuring the distance between $\mathbf{x}_L$ and $\mathbf{x}_S$, we measure the distance long the principal axis.

where $\mathbf{x}_G$ is the known 3D coordinates for the voxel V_{ijk} in the global frame. Then, provided that $z_L > 0$ which means that the voxel is in front of the camera, we can find which pixel the voxel is projected onto by computing

$$\begin{aligned} p_x &= \frac{f_x x_L}{z_L} + c_x \\ p_y &= \frac{f_y y_L}{z_L} + c_y. \end{aligned} \tag{3.2}$$

The surface point given at this pixel lies on the ray between the camera center and V_{ijk}. We can read the depth for the surface point at pixel (p_x, p_y) by

$$z = I_d(p_x, p_y). \tag{3.3}$$

With known depth z, the 3D point for the surface point $\mathbf{x}_S$ can be computed by

$$\mathbf{x}_S = \begin{pmatrix} \frac{(p_x - c_x)z}{f_x} \\ \frac{(p_y - c_y)z}{f_y} \\ z \end{pmatrix}. \tag{3.4}$$

It is now easy to compute the distance between the voxel and the surface point along the ray between the camera and the voxel via

$$d = \|\mathbf{x}_S - \mathbf{x}_L\|. \tag{3.5}$$

The sign of the distance is obtained by comparing z_L and z_S. In practice, it is easier to approximate the distance by just taking the difference between the z-coordinates, i.e.

$$d = z_L - z_S.$$

With this approximation, a voxel will have negative distance if the voxel is in front of the surface ($z_L < z_S$) and a positive distance if it is behind. Note that this distance is an approximation of the projective distance, but in practice it does not matter. The advantage is that one saves some computations in the implementation which makes it a bit faster. The idea is illustrated in Figure 3.1.

Since we are estimating the distance with the projected distance, it can happen that a surface point close to the voxel is missed if it is not on the ray between the voxel and the camera. Instead one might get a measurement for a surface point far away from the voxel. To reduce the impact of such erroneous measurements, the estimated distance is truncated at a threshold δ. We get the approximated distance

$$d^t = \begin{cases} -\delta, & d < -\delta \\ d, & |d| \leq \delta \\ \delta, & d > \delta. \end{cases} \tag{3.6}$$

This makes the potential error in the measurement limited to the bandwidth $[-\delta, \delta]$. There is also an uncertainty for the measurements when the voxel is behind the observed surface. A voxel might be close to another surface which is not observed in that frame. Therefore we also introduce a weight function for the uncertainties in the measurements. Since we cannot see behind surfaces, a lower weight is assigned to measurements behind a surface, and the further behind the surface is, the lower the weight is. The weight function is defined as follows

$$w(d) = \begin{cases} 1, & d \leq \epsilon \\ e^{-\sigma(d-\epsilon)^2}, & d > \epsilon \\ 0, & d > \delta. \end{cases} \tag{3.7}$$

The distance measurements are made for all voxels in the grid. For every frame with corresponding known global pose of the camera we get a new measurement for each voxel. For a voxel V_{ijk} we get a measurement D_{ijk} for each image. The question is, how do we obtain a TSDF which takes all measurements into account? As proposed by [11], we can formulate the optimization problem

$$E(D_{ijk}) = \sum_{l=1}^{n} w_l (D_{ijk} - D_{ijk}^l)^2, \tag{3.8}$$

where w_l is the weight of the measurement D_{ijk}^l and n is the number of images. Taking the derivative of this function and setting it to zero one gets

$$\begin{aligned} \sum_{l=1}^{n} w_l (D_{ijk} - D_{ijk}^l) = 0 \\ \Leftrightarrow \\ \frac{\sum_{l=1}^{n} w_l D_{ijk}^l}{\sum_{l=1}^{n} w_l} = D_{ijk}. \end{aligned} \tag{3.9}$$

The optimal measurement for a voxel V_{ijk} is therefore the weighted average of all measurements. Since each voxel is independent of the others, one can easily obtain an optimal TSDF by computing the weighted average for each voxel. For each voxel, we do the following update

$$\begin{aligned} D^{i+1} &= \frac{W^i D^i + w(d_{i+1}) d_{i+1}^t}{W^i + w(d_{i+1})} \\ W^{i+1} &= W^i + w(d_{i+1}). \end{aligned} \tag{3.10}$$

As we can see, for every new image we can simply update the entire grid by these computations in order to get the current best approximation of the 3D model. Furthermore, each computation only needs reading and writing from one voxel, so this procedure is straightforward to implement in parallel.

Similarly, the color for voxel V_{ijk} can be estimated by extracting the RGB-vector (r, g, b) from the color image I_c^{i+1}. For each new image we

obtain a measurement which we can integrate into the voxel by computing

$$R^{i+1} = \frac{W_c^i R^i + w_c r_{i+1}}{W_c^i + w_c} \tag{3.11}$$

$$G^{i+1} = \frac{W_c^i G^i + w_c g_{i+1}}{W_c^i + w_c} \tag{3.12}$$

$$B^{i+1} = \frac{W_C^i B^i + w_c b_{i+1}}{W_c^i + w_c}, \tag{3.13}$$

where w_c is the weight of the new measurement defined as

$$w_c = \cos(\theta) \cdot w(d_{i+1}), \tag{3.14}$$

where θ is the angle between the optical axis and the light ray. (r^{+1}, g^{i+1}, b^{i+1}) are the measured intensities in the new color image I_c^{i+1}. These measurements assigns a RGB-vector to every voxel. This color vector can be used to colorize the model.

3.2 Estimating the Camera Pose Using Geometry Information From the 3D Model

We now have a simple method of integrating a new depth frame into the 3D model, given that we already know the pose of the camera with respect to the global coordinate system. Simple as it may sound, it is not so easy to obtain the rotation and translation of the camera. The main contribution of this part of the thesis is how we tackle the problem of finding the camera pose using the model directly, rather than doing an ICP-like procedure or maximizing photo-consistency in a frame-to-frame manner. The key idea is to use the distance information embedded in the signed distance function itself.

Assume that we after N images have found a (correct) representation of the 3D model through the signed distance function. We illustrate this in Figure 3.2.

Given a new image, I_{N+1}, we get new measurements of the surface. Without knowing the rotation and translation of the camera, a guess will

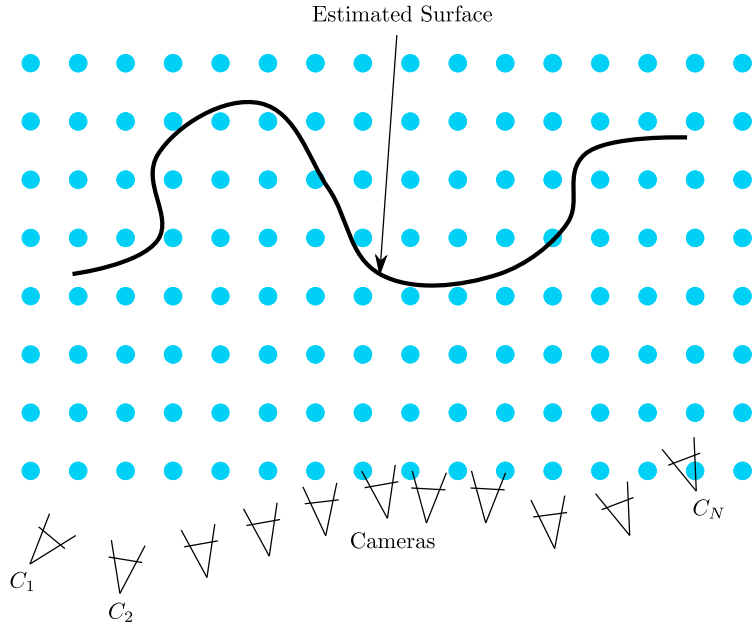


Figure 3.2: Assuming we know the rotation and translation of the first N camera positions a surface can be created, represented in a voxel grid.

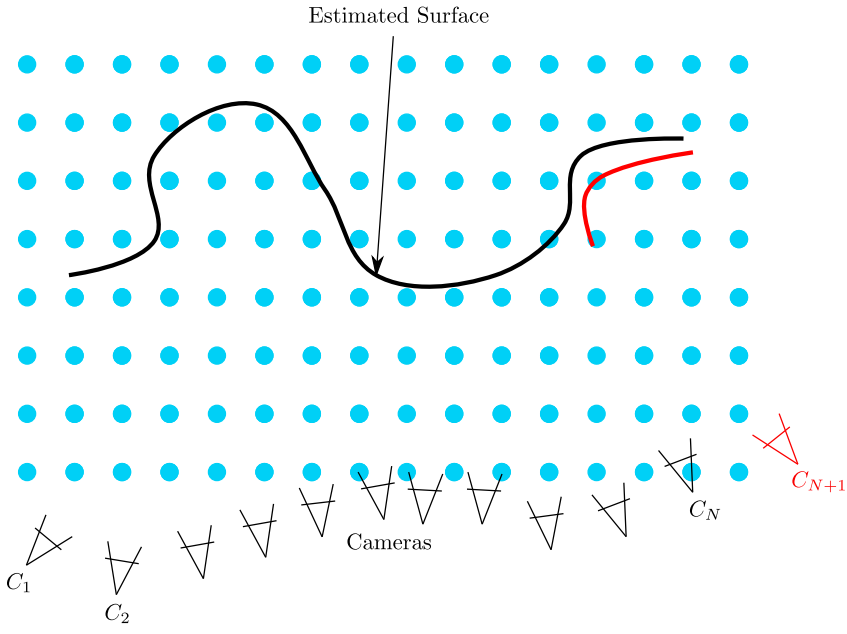


Figure 3.3: Without correct rotation and translation, the newly observed surface cannot be correctly aligned to the estimated surface from the first N surfaces.

most likely not align the new surface onto the estimated (assumed true) surface, as depicted in Figure 3.3.

The key idea to find the correct configuration of the camera is to use the distance information obtained for each 3D point from the new depth image. In reality, what we get from each new depth image is a point cloud, and by a guess of the global pose of the position of the camera, one can reconstruct the point cloud into the voxel grid. For each 3D point, we can find out where in the voxel grid it is located. Using the distance information in the voxels a distance between the 3D point and the actual surface can be computed, as shown in Figure 3.4. Assuming a small camera motion, most of the structure seen in the new frame will be the same as observed in the previous frames. Therefore, to find the correct pose of the camera, it is reasonable to find the rotation and translation which minimizes the distances between the point cloud and

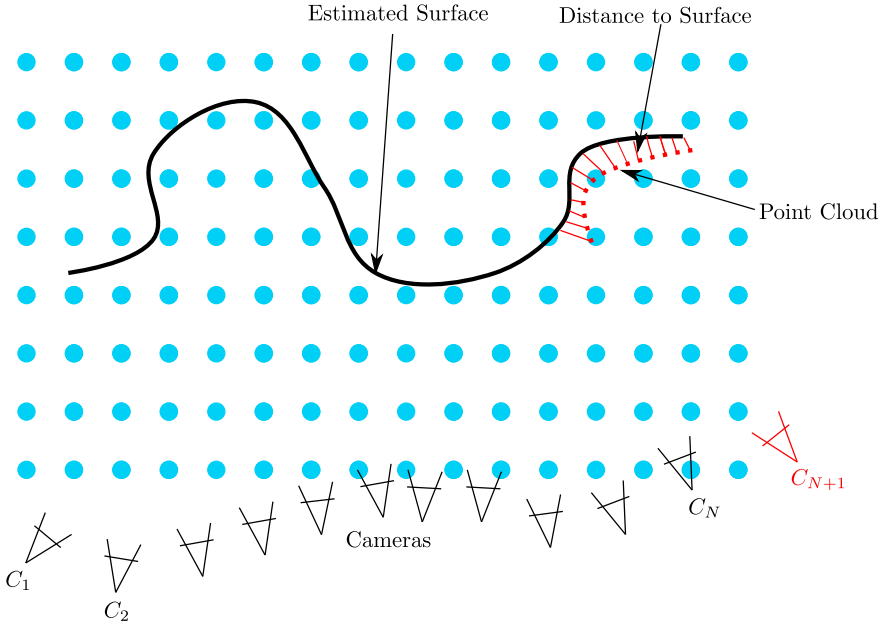


Figure 3.4: The point cloud can be reconstructed into the voxel grid, and for each point we can estimate the distance between the surface and the 3D point.

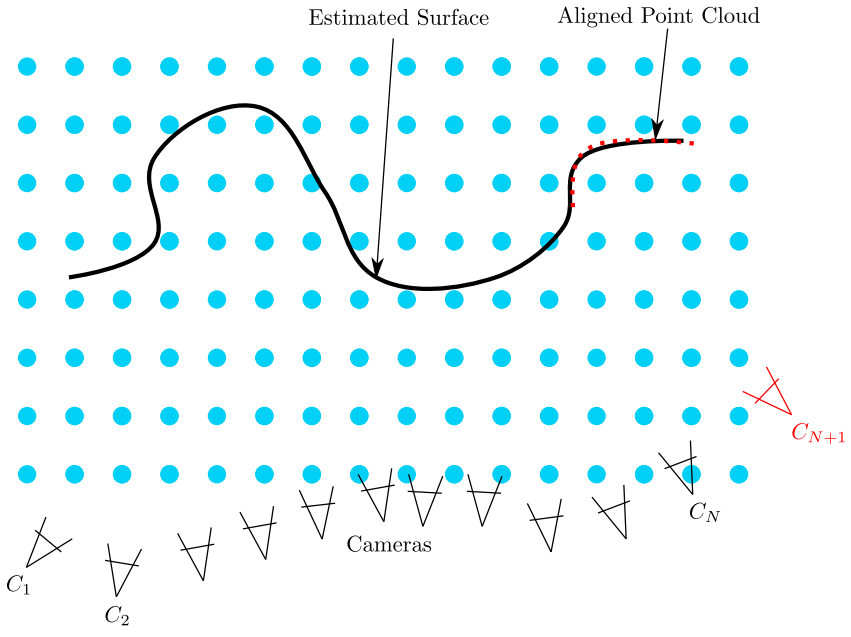


Figure 3.5: With the correct rotation and translation of the camera, as many points as possible should be reconstructed onto the surface as possible.

the surface, as illustrated in Figure 3.5. We now define the function for obtaining the distance to the surface.

Definition 3.2.1. Let $\phi : \mathbb{R}^3 \mapsto \mathbb{R}$ be the function which takes a 3D point $\mathbf{x}$ and returns the value in the voxel grid at that position in the grid.

The challenge now is to find this pose. By observing that in a distance function, the distance between a 3D point and the surface is zero on the surface, and increasing the further away from the surface we move, the following error function can be defined

$$E(R, \mathbf{t}) = \sum_{i=1}^{M} \sum_{j=1}^{N} \phi(R\mathbf{x}_{ij} + \mathbf{t})^2. \tag{3.15}$$

Here R and $\mathbf{t}$ denotes the global rotation and translation and $\mathbf{x}_{ij}$ is the local 3D point and ϕ is the TSDF. M and N are the number of rows and

colons in the image. If this error function is 0 for some R and $\mathbf{t}$, all points are reconstructed on the surface. Due to noise and earlier unobserved structure, the error function will in practice never become zero, so we need to find the minimal error. The task is now to solve

$$\min_{R,\mathbf{t}} \sum_{i=1}^{M} \sum_{j=1}^{N} \phi(R\mathbf{x}_{ij} + \mathbf{t})^2. \tag{3.16}$$

However, we have no analytic expression of the signed distance function which makes it hard to minimize directly. To parametrize R and $\mathbf{t}$ we use the Lie Algebra representation [22]. With this representation, it is possible to represent the entire rigid transformation via a 6-dimensional vector

$$(\boldsymbol{\omega}, \mathbf{t}) = (r_x, r_y, r_z, t_x, t_y, t_z). \tag{3.17}$$

Here r_x, r_y and r_z represents the rotation around the three axes and t_x, t_y and t_z is the translation. To go from this vector representation to a rigid transformation $C \in \mathcal{SO}(3)$, one computes the exponential matrix

$$R = e^{\hat{\omega}}, \tag{3.18}$$

where

$$\hat{\omega} = \begin{bmatrix} 0 & -r_z & r_y \\ r_z & 0 & -r_x \\ -r_y & r_x & 0 \end{bmatrix}. \tag{3.19}$$

With this representation, we can rewrite the error function as

$$E(\boldsymbol{\omega}, \mathbf{t}) = \sum_{i=1}^{M} \sum_{j=1}^{N} \phi(g(\boldsymbol{\omega}, \mathbf{t}, \mathbf{x}_{ij}))^2, \tag{3.20}$$

where

$$g(\boldsymbol{\omega}, \mathbf{t}, \mathbf{x}) = e^{\hat{\omega}}\mathbf{x} + \mathbf{t}. \tag{3.21}$$

In order to optimize the above energy, we use the Gauss-Newton method. The Gauss-Newton method is suitable since the distance between two

Data: Depth image: I_d^{N+1}, SDF: ϕ, Previous camera position: $(\boldsymbol{\omega}_N, \mathbf{t}_N)$
Result: Camera position $(\boldsymbol{\omega}_{N+1}, \mathbf{t}_{N+1})$
$\boldsymbol{\omega}_{New} = \boldsymbol{\omega}_N$;
$\mathbf{t}_{New} = \mathbf{t}_N$;
$\boldsymbol{\omega}_{Old} = \boldsymbol{\omega}_N$;
$\mathbf{t}_{Old} = \mathbf{t}_N$;
while converged == false do
  $\boldsymbol{\omega}_{Old} = \boldsymbol{\omega}_{New}$;
  $\mathbf{t}_{Old} = \mathbf{t}_{New}$;
  $A = \sum_{i,j} \nabla\phi(g(\boldsymbol{\omega}_{New}, \mathbf{t}_{New}, \mathbf{x}_{ij}) \nabla^T \phi(g(\boldsymbol{\omega}_{New}, \mathbf{t}_{New}, \mathbf{x}_{ij})$;
  $\mathbf{b} = \sum_{i,j} \phi(g(\boldsymbol{\omega}_{New}, \mathbf{t}_{New}, \mathbf{x}_{ij}) \nabla\phi(g(\boldsymbol{\omega}_{New}, \mathbf{t}_{New}, \mathbf{x}_{ij})$;
  $\begin{pmatrix} \boldsymbol{\omega}_{New} \\ \mathbf{t}_{New} \end{pmatrix} = -A^{-1}\mathbf{b} + \begin{pmatrix} \boldsymbol{\omega}_{New} \\ \mathbf{t}_{New} \end{pmatrix}$;
  if $\|(\boldsymbol{\omega}_{New}, \mathbf{t}_{New}) - (\boldsymbol{\omega}_{Old}, \mathbf{t}_{Old})\|_\infty < \epsilon$ then
    converged = true;
  end
end
$\boldsymbol{\omega}_{N+1} = \boldsymbol{\omega}_{New}$;
$\mathbf{t}_{N+1} = \mathbf{t}_{New}$

Algorithm 1: The algorithm for computing the new camera pose $(\boldsymbol{\omega}_{N+1}, \mathbf{t}_{N+1})$.

consecutive camera positions will be small under normal circumstances due to the high frame-rate. Hence by using the last known position, we will be quite close to the optimal point already. Linearizing the error function around the current guess of the camera position $(\boldsymbol{\omega}_0, \mathbf{t_0})$, we get the following error function

$$E(\boldsymbol{\omega}, \mathbf{t}) \approx \sum_{i,j} (\phi(g(\boldsymbol{\omega}_0, \mathbf{t}_0, \mathbf{x}_{ij})) + \nabla_{(\boldsymbol{\omega},\mathbf{t})} \phi(g(\boldsymbol{\omega}_0, \mathbf{t}_0, \mathbf{x}_{ij}))^T \begin{pmatrix} \boldsymbol{\omega} - \boldsymbol{\omega}_0 \\ \mathbf{t} - \mathbf{t}_0 \end{pmatrix})^2. \tag{3.22}$$

Now we can optimize the approximated error function by taking the gradient of it with respect to $(\boldsymbol{\omega}, \mathbf{t})$ and setting it equal to 0. We get

$$\sum_{i,j} \phi(g(\boldsymbol{\omega}_0, \mathbf{t}_0, \mathbf{x}_{ij})) \nabla \phi_{(\boldsymbol{\omega},\mathbf{t}))} (g(\boldsymbol{\omega}_0, \mathbf{t}_0, \mathbf{x}_{ij}) +$$

$$\nabla \phi(g(\boldsymbol{\omega}_0, \mathbf{t}_0, \mathbf{x}_{ij})) \nabla^T \phi(g(\boldsymbol{\omega}_0, \mathbf{t}_0, \mathbf{x}_{ij})) \begin{pmatrix} \boldsymbol{\omega} - \boldsymbol{\omega}_0 \\ \mathbf{t} - \mathbf{t}_0 \end{pmatrix} = 0 \tag{3.23}$$

which is easily solved if the resulting matrix

$$A = \sum_{i,j} \nabla \phi(g(\boldsymbol{\omega}_0, \mathbf{t}_0, \mathbf{x}_{ij})) \nabla^T \phi(g(\boldsymbol{\omega}_0, \mathbf{t}_0, \mathbf{x}_{ij})) \tag{3.24}$$

is invertible. This will be the case as long as there is enough different structure in the scene. Then there will be a unique point $(\boldsymbol{\omega}^*, \boldsymbol{t}^*)$ such that error is minimized. In case of planar scenes, this will fail since we do not have enough constraints to determine the rotation and translation uniquely. This is a general shortcoming of purely geometric based tracking approaches, such as ICP.

So to find the camera position, we use the method described above. Initializing with camera position from the last frame, we start by solving the linearized error function and take the newly found camera pose and re-linearize again until convergence. To compute the gradient of ϕ, we use the chain rule. The method is summarized in Algorithm 1.

Note that we are computing the gradient for each 3D point obtained from pixel (i, j), and the computations are independent of each other. Hence, these computations can mostly be done in parallel, it takes some more refined techniques to implement it compared to integrating the 3D

models though. That is because at the end all matrices A_{ij} and b_{ij} computed for each pixel (i, j) need to be summed up to one matrix A and one vector b. To do that on the GPU, we have to use tree-reduction and use shared memory, otherwise, it is quite straightforward.

The main difference between our approach and KinectFusion [23] is that we make use of all 3D points and we make no data association, which is done in KinectFusion and their ICP-framework.

In summary, we now have a way of estimating the camera pose given a 3D model and a new image. Moreover, we also know how to integrate this image into the model in order to update it. This can be used to create an algorithm capable of creating a 3D model on the fly. The work flow for the method is presented in Algorithm 2.

```
Data: Depth Image: I_d^k, SDF: φ
Initialize;
(ω_0, t_0) = (ω_init, t_init);
φ = updateSDF(φ, ω_0, t_0, I_d^0);
k = 1;
while stop == false do
    I_d^k = acquireDepthImage();
    (ω_k, t_k) = getCameraPose(ω_{k-1}, t_{k-1}, I_d^k, φ);
    φ = updateSDF(φ, ω_k, t_k, I_d^k);
    k++;
    if stopping criteria fulfilled then
        stop = true;
    end
end
```

Algorithm 2: Work flow for the system to create a complete 3D model. To stop the procedure one can for instance pre- define how many images one should use. Other stop criteria are of course possible.

Figure 3.6: 3D reconstruction of a room, top view.

3.2.1 Results and Experiments

Qualitative Results

To evaluate this system we perform several experiments and we evaluate it on the public benchmark [30].

The advantage of using a TSDF is that it requires no constraints on the topology of the surface. This means that it should be possible to create arbitrary 3D models, provided there is enough structure to get the tracking to work. To demonstrate this, look at Figures 3.6 and 3.7. This is a smaller room which have been reconstructed using the proposed method. As one can see, it looks quite decent. For instance the black keys on the keyboard are distinguishable, which indicates a correct estimation of the trajectory. Also the figures on the wall to the right in Figure 3.7 have distinct edges and are correctly reconstructed, if the tracking had failed, they would have been smeared out. However, looking at the wall in the left corner at the window, there seems to be some artifacts and the motive on the pictures are not distinguishable. Though the picture frames are sharp rectangles and correctly reconstructed, there are still

Figure 3.7: Same model as in Figure 3.6, note that the black keys on the keyboard are distinguishable from the white keys. That is an indication of a consistent tracking.

some challenges to be met.

A disadvantage with the uniform grid representation of the TSDF is that it requires a lot of memory, and to make detailed reconstructions, the object cannot be too large. For instance, a grid of size 4 m^3 would have a spatial resolution of approximately 1.6 cm if 256^3 voxels are used. To get detailed reconstructions one has to use small objects which fits into a small voxel grid. That the reconstruction can be made more detailed is clearly illustrated in Figure 3.8, here the emblem on the book is clearly distinguishable for example. The high quality of the model itself is of course proof of a good estimation of the camera trajectory. To verify this we look at Figure 3.9 where we can see that there is very little drift in the sequence due to the closed loop. Note that the reconstruction could not have such high resolution if the scene were larger. Here the volume is approximately 1 m^3.

An application mentioned in the introduction of real-time 3D reconstruction could for instance be autonomous flights of a quadrocopter. We investigate this further by performing several experiments using a quadrocopter together with an Asus sensor and base station with GPU capabilities. The first experiments were made to see if a quadrocopter was capable of following a pre-defined orbit. The model was initialized

Figure 3.8: A reconstruction of a smaller object. Due to the small size of the grid, the reconstruction gets significantly more detailed. Look on the emblem on the book, it is clearly distinguishable.

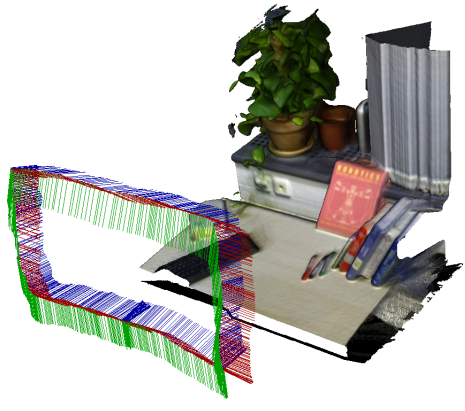


Figure 3.9: Looking at the trajectory and the quality of the 3D model, it is apparent that there is little drift in this sequence.

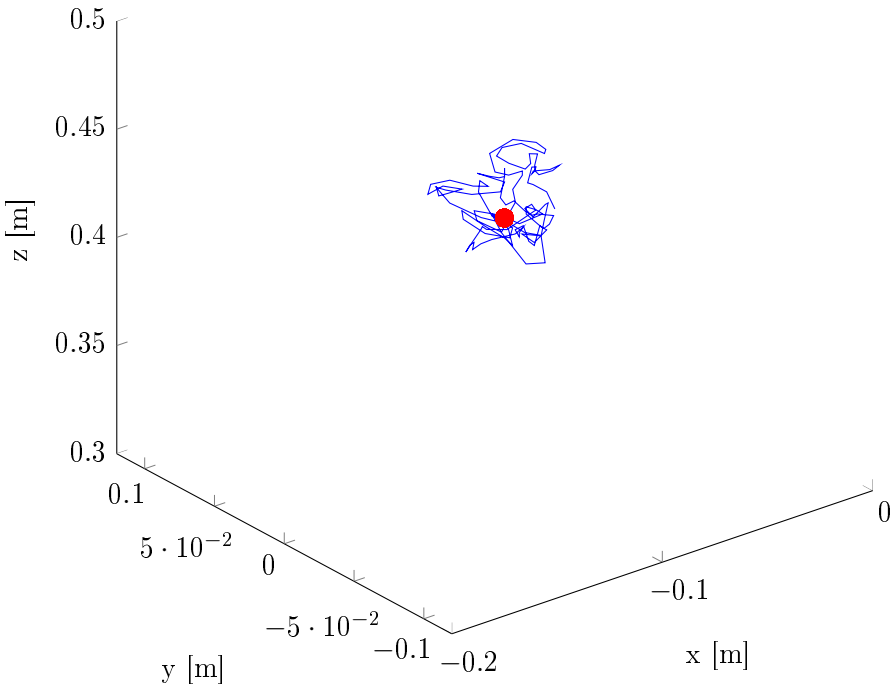


Figure 3.10: In the first quadrocopter experiment the task of the quadrocopter was to hover at the same point. As one can see in the image, it is capable of staying at the same position reasonably good.

while the quadrocopter was staying on the ground, then a signal was sent to start and lift to a certain altitude. Thereafter the mode was switched to autonomous control and the quadrocopter should follow a predefined orbit by computing its position using the depth sensor and our algorithm. The first experiment was just to hover on the same position. In this experiment, the quadrocopter had an average standard deviation of 2.1 cm while hovering, the experiment is demonstrated in Figure 3.10. The error was measured between the set goal position and the estimated position of our algorithm. A more advanced trajectory was also tested. The task was to navigate in a rectangle and follow the path for several rounds. As one can see in Figure 3.11, this was successfully accomplished. The blue line is the pose the quadrocopter flew and the

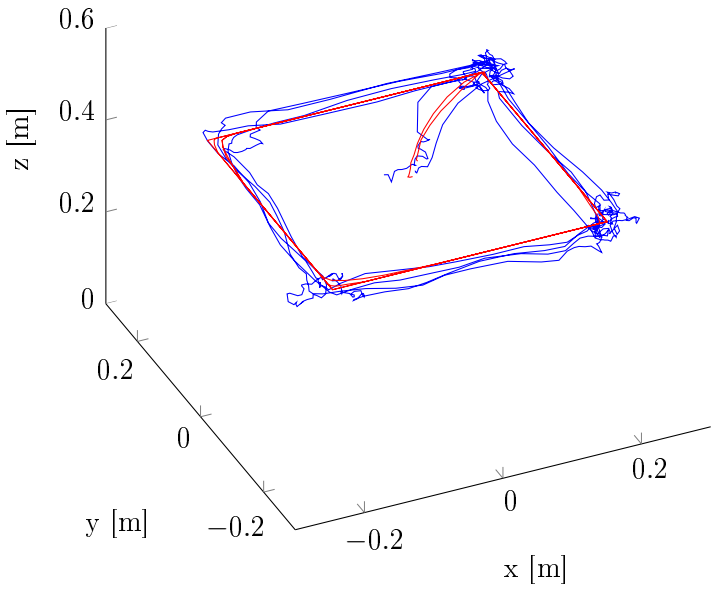


Figure 3.11: The path the quadrocopter should follow is in red and the estimated trajectory in blue. As one can see in the image, it manages to follow the same trajectory for several rounds.

red line is the path it was supposed to take. In another experiment, the quadrocopter was in assisted mode and a user should specify way points and the quadrocopter should use its estimated pose to navigate to the specified way points. The resulting 3D model is decent as can be seen in Figure 3.12 and Figure 3.13. This again shows that the trajectory is correctly estimated.

The algorithm can also be used to create 3D models of persons. By sitting on a swivel chair and rotate 360° one gets a complete scan of the upper body. A result can be seen in Figure 3.14. Looking at the pose in Figure 3.14 we see that the loop closes nicely, which it is supposed to do.

Figure 3.12: Side view of the resulting 3D reconstruction of a room using the assisted mode on the quadrocopter.

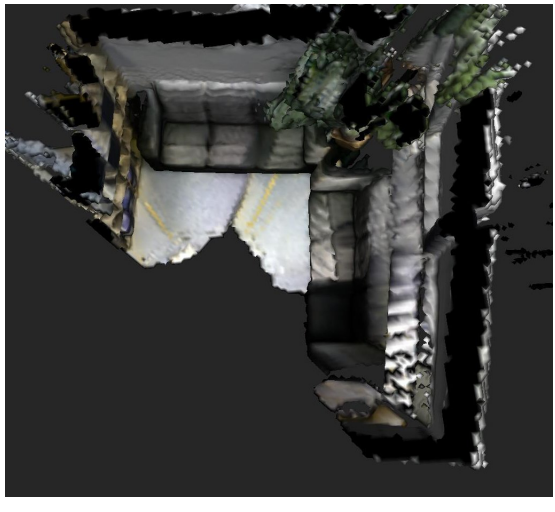


Figure 3.13: Top view of the 3D model obtained from assisted mode using the quadrocopter.

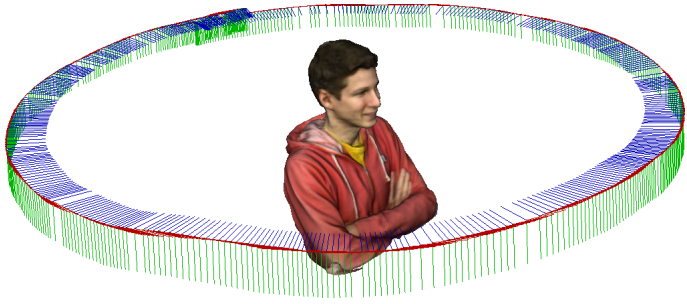


Figure 3.14: A 3D scan of a man sitting on a swivel chair. The colored lines is the estimated pose of the camera.

Quantitative Results

To make a more rigorous evaluation of the system, we evaluate the algorithm on public available benchmarks [30]. To get a better understanding of the quality of our work, we also compare ourselves to the PCL-implementation of KinectFusion [1], known as KinFu. In [7], we do also compare ourselves to RGB-D SLAM [12] as well. After publication, we understood that they first estimate the camera positions for all images, reducing for drift using loop-closures and other techniques. When all camera positions have been optimized globally, [12] creates the 3D model by fusing all images into a probabilistic octree representation [36]. This is a simplified problem because it makes it possible compute all poses optimal before creating a map. In contrast we and KinectFusion [23] create the model online, which makes it substantially more difficult to maintain an accurate estimation of the camera since we do not update it afterwards. Therefore, we do not consider it fair to compare KinFu and our method to RGB-D SLAM. The frame rate with a voxel grid of 512^3 voxels was about $15 - 16$ Hz, using an NVidia Geforce GTX 770, Intel i7 3.4 GHz processor and 16 GB RAM.

Looking at Table 3.1, we see that our method clearly outperforms KinFu on almost all datasets. Either KinFu works poorly, or not at all. This might be a bit surprising due to the impressive videos shown on youtube[1]. There might be some difference between the KinFu implemen-

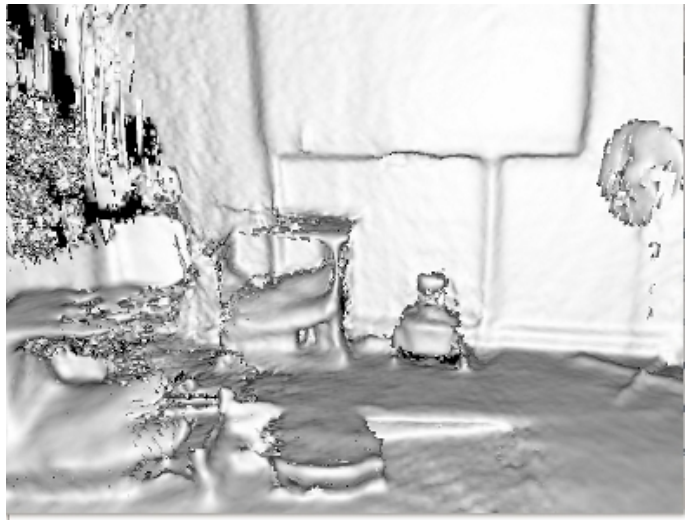
[1] https://www.youtube.com/watch?v=quGhaggn3cQ

Table 3.1: The root-mean square absolute trajectory error for KinFu and our method for different resolutions, metrics and datasets.

Method	Res.	Teddy	F1 Desk	F1 Desk2	F3 House
KinFu	256	0.156 m	0.057 m	0.420 m	0.064 m
KinFu	512	0.337 m	0.068 m	0.635 m	0.061 m
Point-To-Point	256	0.075 m	0.037 m	0.064 m	0.037 m
Point-To-Point	512	**0.072** m	**0.035** m	**0.055** m	**0.035** m
Method	Res.	F1 360	F1 Plant	F1 RPY	F1 XYZ
KinFu	256	0.913 m	0.598 m	0.133 m	0.026 m
KinFu	512	0.591 m	0.281 m	0.081 m	0.025 m
Point-To-Point	256	0.553 m	0.048 m	**0.042** m	0.022 m
Point-To-Point	512	**0.131** m	**0.044** m	0.045 m	**0.022** m

tation and the original KinectFusion. Moreover, most scenes in demonstrations of KinectFusion are for smaller grids, whereas the datasets are larger, and the trajectory possible more challenging to recover. Due to the ray casting technique for visualization, one only sees the model from the current frame and not the entire reconstruction at once, where drift and inconsistencies are more prominent. This is clearly visible for datasets where loop-closure is present since one can see how the previous visited areas gets destroyed due to drift in the camera pose. Inaccuracies in the estimation of the camera position leads to errors in the TSDF which ultimately destroys the implicit surface, which is shown in Figure 3.15. However, this evaluation does not give the whole truth, even though our tracking method is superior in the experiments, there are certain advantages with the KinectFusion approach. One advantage for example is that when using the ICP method, one can reduce the truncation in TSDF estimation. This gives a better estimation of the distances, as long as the camera position is correctly estimated.

The main reason to why our method is superior to KinectFusion, is probably that we make use of more information when we compute the rotation and translation of the camera. In KinectFusion, a point cloud is created by performing ray casting on the current 3D model.



(a) Reconstruction of Fr1 Teddy using KinFu

(b) Reconstruction of Fr1 Teddy using our method.

Figure 3.15: Comparison of the reconstruction of Fr1 Teddy using our method and KinFu. Note that the Teddy Bear in the top figure is completely gone.

Then the 3D points in this point cloud is associated to the points in the new point cloud obtained from the new depth image, through the fast Data Association algorithm [4]. In this process, potential matches are rejected, so at the end, KinectFusion uses less 3D points to compute the camera transformation. In contrast, our approach uses all 3D points which are reconstructed in the grid where we have measurements of the distance to the surface. This way we make use of more data, which makes tracking more robust and accurate. It might also be that we get a better estimation of the error between the surface and the reconstructed 3D point compared to KinectFusion. In the data association there might be false matches which will have a negative effect on the tracking.

There are also some drawbacks of our approach. When we compute the gradients we must be within the threshold in the distance function, otherwise the distance is thresholded to δ and the gradient would be zero and of no use. Empirically we found that the threshold must be rather big, $\delta = 0.3$ m, to make our method perform well on all benchmarks. This has the effect that voxels further away from the surface can get a rather poor estimation of the distance to the surface. We also use more voxels behind the surface. In contrast, KinectFusion has a rather tight threshold, which undoubtedly gives a visually more appealing surface when the tracking works. That is a trade of we have to do. With a tight threshold, many points will be reconstructed in the thresholded area which makes our method work poorly. In contrast KinectFusion just performs raycasting and then does ICP, so their tracking is not at all that sensitive.

3.3 Robust Estimation of the Camera Pose

The 3D models shown are colorized and this is done by integrating all color information in each voxel as described in Section 3.1. So far this has only been used for visualization purposes. As mentioned earlier, the tracking approach proposed in Algorithm 2, does only work when there is enough structure in the scene. This is a shortcoming. In many practical applications we might at least partially be in scenes where there is little structure. In this part we address this issue with two different approaches. The key idea behind both approaches is to invoke

color information in the tracking algorithm. Color information has the advantage that only texture is required, so even if the scene is planar, texture should be enough to recover the camera pose. On the other hand, invoking only texture has the disadvantage of being sensitive to scenes with no texture, so the tactic here is to extend our tracking method so that we make use of both color information and geometric information.

3.3.1 Invoking Color in the Camera Pose Estimation

We saw in Section 3.1 how the color of the 3D model can be estimated by computing a weighted average of color measurements for each voxel. Now when we have a textured 3D model represented in a voxel grid, we want to use this information to improve the camera tracking. The idea is based on photo consistency between the current estimation of the 3D model and the newly obtained color image I_c. By using a guess of the transformation $[R\ \mathbf{t}]$, we can reconstruct a 3D point into the voxel grid by

$$\mathbf{x}_G = R\mathbf{x}_L + \mathbf{t}, \tag{3.25}$$

where

$$\mathbf{x}_L = \begin{pmatrix} \frac{(p_x - c_x)z}{f_x} \\ \frac{(p_y - c_y)z}{f_y} \\ z \end{pmatrix}. \tag{3.26}$$

From $\mathbf{x}_G$, we can easily find out where in the voxel grid the point is located, and we can then compute the color (R_s, G_s, B_s) of the surface there. By obtaining the RGB-vector of the model (R_s, G_s, B_s), we can compare this with the color intensities in the pixel coordinates (p_x, p_y) in the RGB image I_c. Ideally, these should match for each pixel, as illustrated in Figure 3.16.

So by computing the difference between the intensities in the image and the color on the surface for each pixel, we get the error

$$E_{color}(R, \mathbf{t}) = \sum_{ij} \|C(R\mathbf{x}_L + t) - I_c(i, j)\|^2, \tag{3.27}$$

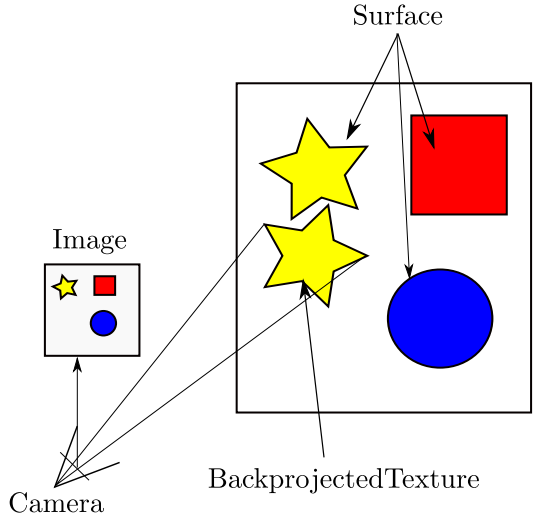


Figure 3.16: By reconstructing the 3D points with a guess of R and $\mathbf{t}$, it is possible to compare the color of the model where the point is reconstructed and the color in the color image I_c. Ideally, it should match.

where $C(R\mathbf{x}_L + \mathbf{t})$ is the RGB-vector in the voxel grid at $R\mathbf{x}_L + \mathbf{t}$ and $I_c(i,j)$ is the RGB-vector in the color image at pixel (i,j). Thus, we have an error metric which takes color information into account. This allows us to integrate color information into our original method where we only use geometric information. Let us first define the color error as

$$\psi(R\mathbf{x}_L + \mathbf{t}) = \|C(R\mathbf{x}_L + \mathbf{t}) - I_c(i,j)\|^2. \tag{3.28}$$

By adding this to the geometric error we get

$$E_{tot}(R,\mathbf{t}) = \phi(R\mathbf{x}_L + \mathbf{t})^2 + \alpha\psi(R\mathbf{x}_L + \mathbf{t})^2, \tag{3.29}$$

where α is the weight of the color error. This term takes into account both the geometric error and error in each color channel. The error is zero when the rotation and translation is such that the point is reconstructed onto the zero level set, and the color on the surface matches the color in the pixel. Note that due to the threshold, the geometric error differs between $[-\delta, \delta]$ and the color error goes between $[0, 1]$. One could scale the color error to lie in the interval $[0, \delta]$. However, the weight α can be tuned to give the desired impact on the tracking instead.

With this, the new error function we want to minimize is

$$E(R,\mathbf{t}) = \sum_{i,j} \phi(R\mathbf{x}_{ij} + \mathbf{t})^2 + \alpha\psi(R\mathbf{x}_{ij} + \mathbf{t})^2, \tag{3.30}$$

where we sum over all pixels (i,j). Again, we change representation of the rigid body motion by using the Lie-algebra representation

$$(\boldsymbol{\omega}, \mathbf{t}) = (\omega_x, \omega_y, \omega_z, t_x, t_y, t_z). \tag{3.31}$$

With this and a local 3D coordinate $\mathbf{x}_{ij}$ for pixel (i,j) we write the residual vector as

$$r_{ij}(\boldsymbol{\omega}, \mathbf{t}, \mathbf{x}_{ij}) = (\phi(g(\boldsymbol{\omega}, \mathbf{t}, \mathbf{x}_{ij}), \sqrt{\alpha}\psi(g(\boldsymbol{\omega}, \mathbf{t}, \mathbf{x}_{ij})))^T, \tag{3.32}$$

and the error function becomes

$$E(\boldsymbol{\omega}, \boldsymbol{t}) = \sum_{i,j} r_{ij}(\boldsymbol{\omega}, \mathbf{t}, \mathbf{x}_{ij})^T r_{ij}(\boldsymbol{\omega}, \mathbf{t}, \mathbf{x}_{ij}), \tag{3.33}$$

which we solve by using the Gauss-Newton method. Thus we now have a method for estimating the camera position by using both geometry and texture information.

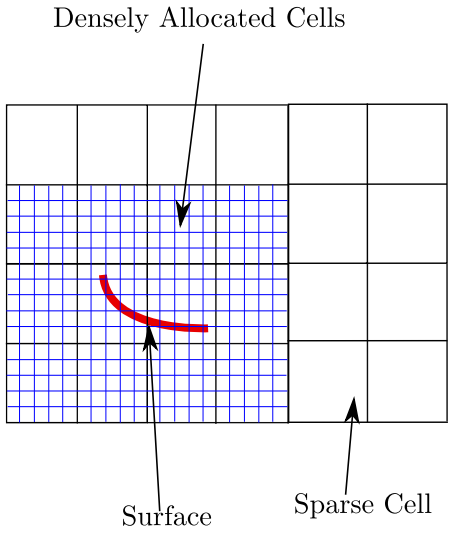


Figure 3.17: Close to the surface we allocate a dense voxel grid, but further away from the surface there are no voxels allocated.

3.3.2 Efficient Use of Memory

Another problem we address is the waste of memory by using a uniform grid. Remember that the memory consumption of a uniform grid grows cubically with the voxel resolution. To get more detailed reconstructions requires small objects, otherwise the memory consumption would explode. To address this we simply notice that we are only interested in the distance function in the vicinity of the surface and therefore, there is no need to allocate a lot of voxels in empty space. The simplest approach would be to implement an octree representation where each node has eight sub-leaves. However, then the resolution quickly decreases as we get far away from the surface. To compute the derivatives, we want to have a high resolution around the surface as well. Therefore, we implement a representation where we have a very coarse voxel grid, with no voxels allocated, then if we detect a surface in any of these voxels, we allocate densely with voxels there. This is illustrated in 3.17. This allows us to get a higher spatial resolution for larger reconstructions without running out of memory.

Figure 3.18: The reconstructed scene when the sensor is only facing the floor.

3.3.3 Experiments and Results

The purpose of invoking color information in the tracking procedure was to address that our original method would not work when there is only planar surfaces. To test our hypothesis, we made a simple experiment by recording data with the depth camera facing the floor only. As one can see in Figure 3.18, everything looks smeared out and the estimated pose is approximately a non-moving one, whereas invoking color information yields a completely different result, as seen in Figure 3.19.

When we use the color information to estimate the camera pose the floor with its texture is clearly visible. Moreover, the edges on the blue squares are clearly distinguishable. The sharp edges indicate that the pose of the camera is correctly estimated. In another experiment, we tested how the method works for larger reconstructions, since we now have a more efficient way of representing the distance function. We tried to reconstruct a part of a corridor, which is quite a challenging task, since there are many scenes with little structure. The result is shown in Figure 3.20. The movement in these scenes is quite simple. Nonetheless, it shows that our tracking is also rather robust over larger scenes. The corridor is approximately 40 m long and even if there is visible drift, it is not that bad. It is inherently prone to drift since an error in the model

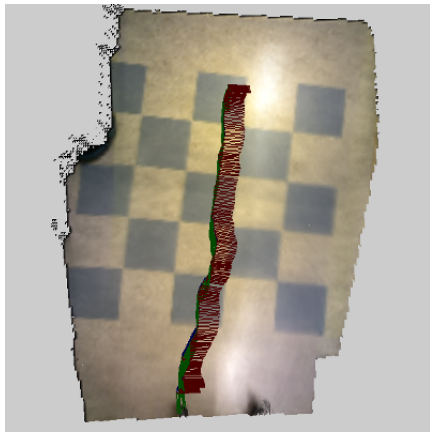


Figure 3.19: The reconstructed scene when the sensor is only facing the floor and color information is invoked in the tracking.

Figure 3.20: The reconstructed corridor, using both color and geometric information in the camera pose estimation.

gives an error in the tracking and then it is impossible to recover without any external technique. Nonetheless, this experiments suggests that the drift is small.

3.3.4 Quantitative Results

To evaluate our method we test it on the benchmarks [30] again. In [8], we compared ourselves to [29], and drew the conclusion that they were superior to us. This was not quite fair, since they separate the problem into two parts. They first compute all camera positions globally to reduce drift, then they integrate all images into the 3D model. In contrast we try to solve the problem of creating the 3D model online, that is, when we get a new image, we compute the pose and then update the model. This is significantly harder, since we cannot recompute positions we have already found easily. Therefore, we do not compare ourselves to them here. Instead, we only compare this new approach with our old approach where we only use geometric information.

It is clear that invoking color information increases the performance on most datasets. In particular, Fr3 No_Structure_Texture_Far shows the strength of invoking color information. With no color the RMSE is 1.36 m, but invoking color information decreases the error to 0.03 m. However, Fr1 Desk2 shows the opposite behvaiour, the more color that is used, the more inaccurate is the tracking. Fr1 Desk2 is considered to be a quite challenging dataset with a fast movement of the camera. It is also clear from the results that different datasets give best result for different values of the weight α. At the moment the weight has to be set manually. It would be interesting to look further into how the weight can be set automatically. The frame rate with a voxel grid of 512^3 voxels was about $11 - 12$ Hz, using an NVidia Geforce GTX 770, Intel i7 3.4 GHz processor and 16 GB RAM.

3.4 Combining Sparse and Dense Tracking

Invoking color information from the global model clearly improves the tracking as seen in Section 3.3.4. Yet, there are still situations where even texture information is not enough. Imagine we have a planar surface with

Table 3.2: The root-mean square absolute trajectory error (m) for different values of the weight α. Note that $\alpha = 0$ corresponds to the pure geometric tracking approach in Section 3.2

Dataset	Weight α				
α	0	0.1	0.2	0.3	0.4
fr1 teddy	0.072	0.067	0.064	0.061	0.059
fr3 str_no_tx_f	**0.034**	**0.034**	0.035	0.036	0.036
fr3 no_str_tx_f	1.36	0.042	0.041	0.031	0.031
fr1 desk	**0.035**	0.036	0.036	0.037	0.037
fr1 desk2	**0.055**	0.064	0.079	0.105	0.117
fr1 360	0.131	0.131	**0.129**	0.147	0.138
fr3 office_house	0.035	0.032	0.027	0.025	**0.024**
fr1 plant	0.044	**0.042**	0.044	0.047	0.048
fr1 rpy	0.045	0.041	0.039	0.038	**0.037**

α	0.5	0.6	0.7	0.8	0.9	1.0
fr1 teddy	**0.058**	0.066	0.072	0.080	0.248	0.223
fr3 str_no_tx_f	0.037	0.037	0.038	0.038	0.039	0.040
fr3 no_str_tx_f	0.030	0.030	0.030	0.030	**0.029**	**0.029**
fr1 desk	0.038	0.038	0.038	0.038	0.039	0.039
fr1 desk2	0.130	0.132	0.135	0.138	0.140	0.140
fr1 360	0.200	0.196	0.199	0.205	0.737	0.759
fr3 office_house	**0.024**	**0.024**	**0.024**	0.025	0.025	0.026
fr1 plant	0.050	0.051	0.051	0.051	0.051	0.052
fr1 rpy	**0.037**	**0.037**	**0.037**	**0.037**	**0.037**	**0.037**

texture, but the texture is not so distinct. The estimated model is then likely to be smeared out and it will be difficult to estimate the pose.

An idea to attack this problem is to use sparse feature points and invoke these into our error function. The approach is to find corresponding points between the new image and a sequence of images already obtained. With corresponding pixel pairs $[(x^{n+1}, y^{n+1}), (x^n, y^n)]$, where (x^{n+1}, y^{n+1}) corresponds to a pixel in image I^{n+1}, and (x^n, y^n) to a pixel in image I^n, we can compute the global position $\mathbf{y}_G$ of the point corresponding to (x^n, y^n) since the camera position for that frame is known. With these global 3D-coordinates of $\mathbf{y}_G$, we want to find R and $\mathbf{t}$ such that

$$R\mathbf{x} + \mathbf{t} = \mathbf{y}_G. \tag{3.34}$$

where $\mathbf{x}$ is the corresponding local point in the new camera's frame of reference. By matching keypoints between a sequence of the K latest image pairs

$$(I^{n+1}, I^n), (I^{n+1}, I^{n-1}), ..., (I^{n+1}, I^{n-K}) \tag{3.35}$$

we get a set

$$(Y_1, Y_2, ..., Y_K) \tag{3.36}$$

of global 3D points and a corresponding set of local 3D points

$$(X_1, X_2, ..., X_K). \tag{3.37}$$

Each Y_i and corresponding X_i consists of M_i number of corresponding 3D points

$$X_i = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_{M_i}\} \tag{3.38}$$

$$Y_i = \{\mathbf{y}_1, \mathbf{y}_2, \ldots, \mathbf{y}_{M_i}\}. \tag{3.39}$$

We can use this to define the error function

$$E(R, \mathbf{t}) = \sum_{i=1} \|R\mathbf{x}_i + \mathbf{t} - \mathbf{y}_i\|^2 \tag{3.40}$$

where the sum is over all found corresponding 3D-points.

This error function is not dependent on what the model looks like so the idea is that this term shall help finding the correct pose even if the model does not provide enough information. By adding (3.40) to (3.30) we get

$$E(R,\mathbf{t}) = \sum_{i,j} \phi(R\mathbf{x}_{ij} + \mathbf{t})^2 + \alpha\psi(R\mathbf{x}_{ij} + \mathbf{t})^2 + \mu \sum_{l=1} \|R\mathbf{x}_l + \mathbf{t} - \mathbf{y}_l\|^2, \tag{3.41}$$

where we sum over all pixels (i, j) and all found corresponding key points and α and μ are the weights.

Linearizing each sum separately and then adding the resulting matrices for the optimization, we can use the Gauss-Newton as earlier to minimize the error function.

3.4.1 Qualitative Results

This new approach was tested qualitatively on several challenging recordings. In the first sequence, images from a floor with hardly any texture was recorded. This is very challenging due to the fact that it is completely planar, and has little texture, so there is very little information to work with. Therefore, a purely geometric based tracker would never work, and even photo consistency is hard since there is little texture and that is likely to be smeared out in the model. By also invoking sparse feature points into our tracker, the hypothesis is that this shall give more information. Looking on Figures 3.21a and 3.21b, it is clear that the extra information we get from the feature points are very helpful in these extreme situations. In the scene the recording starts in the lower left corner and ends after the chess board pattern in Figure 3.21a. When only using the model in the TSDF for tracking, the result is as in Figure 3.21b. The trajectory cannot be correctly estimated between the starting point and the blue pattern, whereas invoking the sparse feature points in the tracking gives satisfying results.

One can ask oneself, do we really need the texture information in the model now? The feature points gives constraints when we have planar structures. By recording a poster on a wall with plenty of texture, we used our proposed method with both geometry, texture and SURF

points [2] and compared it to the obtained reconstruction where we only used sparse key points. In Figure 3.21d, only SURF points were used and the reconstruction looks rather smeared out due to drift in the tracking. In contrast, the reconstruction using the information from the model in the TSDF as well, clearly gives a better reconstruction, as can be seen in Figure 3.21c. This can be seen by looking at the poster, the distinct details indicate that the trajectory is better estimated. Clearly, the information in the model in the TSDF reduces drift which shows that we cannot exclude texture information from the model in the tracking. Instead one should take all information into account to obtain a tracking algorithm which can work under as many circumstances as possible.

3.4.2 Quantitative Evaluation

To see measure the effect of this new approach with sparse feature points, we benchmarked it on [30]. The results are shown in Table 3.3. The parameter α in (3.41) was set to $\alpha = 0.4$. Then we let μ go between 0 and 1. As can be seen in the evaluation, adding the feature points to the camera pose estimation does not improve the accuracy for most datasets. The only noticeable difference was in datasets Desk2 and 360. Both these datasets are quite challenging and the camera movement faster than in other sequences. However, for most sequences it gives little improvement on the benchmark. The advantage of invoking the feature points in the camera tracking lies in the improved robustness as seen in Section 3.4.1. The frame rate with a voxel grid of 512^3 voxels was about 4-5 Hz, using an NVidia Geforce GTX 770, Intel i7 3.4 GHz processor and 16 GB RAM.

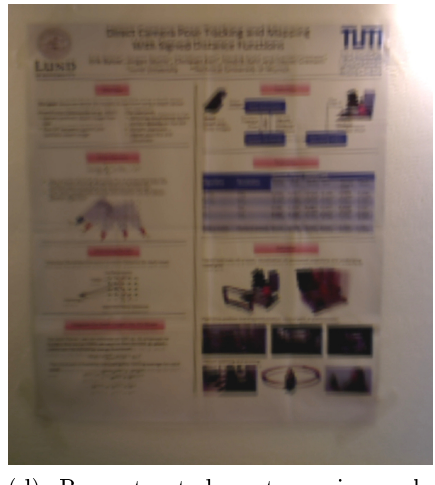
3.5 Discussion

In this part of the thesis we have seen how a 3D model represented as a TSDF can be used for estimation of the camera pose. Evaluations on benchmark shows that our method gives good results on many different datasets with a reasonable speed. The more information we use, the slower tracking works, but the more robust the tracking becomes. Several issues remains, for example for larger reconstructions, one must find a

(a) Reconstructed floor using both the model and sparse feature points

(b) Reconstructed floor using the model only.

(c) Reconstructed poster with plenty of texture using both SURF-points and the model.



(d) Reconstructed poster using only SURF points.

Figure 3.21

Table 3.3: The root-mean square absolute trajectory error (m) for different values of the weight μ, α was set to 0.4.

Dataset Weight μ					
μ	0	0.1	0.2	0.3	0.4
fr1 teddy	**0.059**	**0.059**	0.060	0.060	0.060
fr3 str_no_tx_f	**0.036**	**0.036**	**0.036**	**0.036**	**0.036**
fr3 no_str_tx_f	0.031	**0.030**	**0.030**	**0.030**	**0.030**
fr1 desk	**0.037**	**0.037**	**0.037**	**0.037**	**0.037**
fr1 desk2	0.117	0.107	0.087	0.077	0.075
fr1 360	0.138	0.130	0.128	0.129	0.128
fr3 office_house	0.024	0.024	0.024	0.024	**0.023**
fr1 plant	0.048	0.048	0.047	0.047	0.047
fr1 rpy	0.037	0.037	0.036	0.035	0.035

μ	0.5	0.6	0.7	0.8	0.9	1.0
fr1 teddy	0.061	0.061	0.061	0.060	0.060	0.060
fr3 str_no_tx_f	**0.036**	0.037	0.037	0.037	0.037	0.037
fr3 no_str_tx_f	**0.030**	**0.030**	**0.030**	**0.030**	**0.030**	**0.030**
fr1 desk	**0.037**	**0.037**	**0.037**	**0.037**	**0.037**	**0.037**
fr1 desk2	0.071	0.069	0.066	**0.065**	0.066	**0.065**
fr1 360	0.121	0.118	0.117	0.115	0.112	**0.110**
fr3 office_house	**0.023**	**0.023**	**0.023**	**0.023**	**0.023**	**0.023**
fr1 plant	0.046	0.046	0.046	0.045	0.045	**0.044**
fr1 rpy	0.035	0.034	0.034	0.034	**0.033**	**0.033**

way of reducing errors in the model and the tracking. Now a badly estimated camera pose gets integrated into the model and that way errors are accumulated. For online reconstructions, this must be done in real-time, so that the model is correctly updates as new images are obtained. Another interesting problem is how to set the weights α and μ optimally.

Chapter 4

Low-Rank Approximation of Matrices

4.1 Introduction

Another problem studied in this thesis is how to obtain a low-rank approximation of a measurement matrix. For the reader who is not familiar with this subject we start with giving some examples of problems that eventually lead to a low-rank approximation problem.

4.1.1 Structure from Motion

One of the more familiar applications of low-rank approximation, or more specifically in this case, matrix factorization, is the work by [31].

Assume we have a set of 3D points $Q = \{\mathbf{x}_1, ..., \mathbf{x}_n\}$ where each $x_i \in \mathbb{R}^3$. With an affine camera model $P = \begin{pmatrix} A & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix}$, $A \in \mathbb{R}^{3\times3}$ and $\mathbf{t} \in \mathbb{R}^{3\times1}$. A 3D-point $\mathbf{x}$ is projected onto the image plane with pixel coordinates

$$\begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = P \begin{pmatrix} \mathbf{x} \\ 1 \end{pmatrix} \tag{4.1}$$

where $P \in \mathbb{R}^{3\times4}$. Because of the special shape of the affine camera matrix the third coordinate of the projection will be one as long as the point that we are projecting is not at infinity. Therefore we can ignore this coordinate if we assume that we are only dealing with regular points.

That means the ray between a 3D-point and the pixel coordinates is parallel to the normal of the image plane, as shown in Figure 4.1. So the

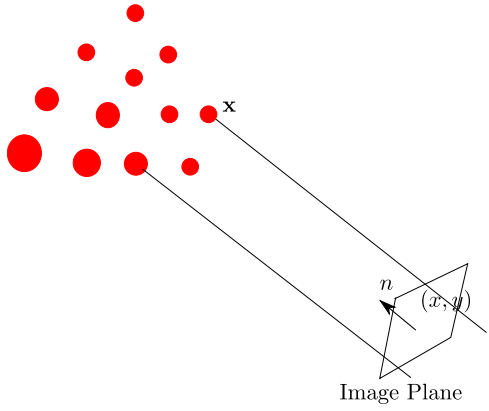


Figure 4.1: An affine camera model, all 3D-points are projected parallel to the image plane. In other words, the camera center is at infinity.

camera center is where all rays meet is a point at infinity. Under many circumstances this is not a realistic model, but when all 3D-points belong to the same object and are far from the camera, the approximation can be decent.

Assume now that we have taken a sequence of K images of the same object and have tracked the image coordinates $(p^i_{x_k}, p^i_{y_k})^T$ for each 3D-point $\mathbf{x}^i$ and each images I_k. We can stack all pixel-coordinates in the matrix M, the rows $2(k-1)+1$ and $2k$ in M contains the coordinates for all pixels in frame I_k and is of size $2K \times N$,

$$M = \begin{pmatrix} x_1^1 & \dots & x_N^1 \\ y_1^1 & \dots & y_N^1 \\ \vdots & \vdots & \vdots \\ x_1^K & \dots & x_N^K \\ y_1^K & \dots & y_N^K \end{pmatrix}. \tag{4.2}$$

Now we want to find the positions of the cameras and the 3D-coordinates of the tracked points. Starting with finding the translation for each camera, it can be shown that

$$t_i = \begin{pmatrix} \bar{x}_i \\ \bar{y}_i \end{pmatrix} - A_i \bar{x}, \tag{4.3}$$

where $\begin{pmatrix} \bar{x}_i \\ \bar{y}_i \end{pmatrix}$ is the mean of the observed pixels in frame i and $\bar{\mathbf{x}}$ is the mean of the 3D points. By withdrawing the corresponding mean coordinates for each pixel, we can assume that the translation is zero. Thus each pixel (x_j^i, y_j^i) in frame i can be written as

$$\begin{pmatrix} x_j^i \\ y_j^i \end{pmatrix} = A_i \mathbf{x}_j \tag{4.4}$$

where $A_i \in \mathbb{R}^{2\times 3}$. We want to factorize $M \in \mathbb{R}^{2K\times N}$ into two matrices A and X such that

$$M = \begin{pmatrix} x_1^1 & \dots & x_N^1 \\ y_1^1 & \dots & y_N^1 \\ \vdots & \vdots & \vdots \\ x_1^K & \dots & x_N^K \\ y_1^K & \dots & y_N^K \end{pmatrix} = \begin{pmatrix} A_1 \\ \vdots \\ A_K \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 & \dots & \mathbf{x}_N \end{pmatrix} = AX, \tag{4.5}$$

where $A_i \in \mathbb{R}^{2\times 3}$ and $\mathbf{x}_j \in \mathbb{R}^{3\times 1}$. Hence given a set of measurements in a matrix M the goal is to find a decomposition of M where one part corresponds to the camera and the other corresponds to the 3D points. We can immediately conclude that the rank of the measurement matrix M is at most 3, since $A \in \mathbb{R}^{2K\times 3}$ and $X \in \mathbb{R}^{3\times N}$.

Now this is an ideal case, dealing with reality you never have perfect measurements, instead the will be noise as well. The best you can do then is to find an approximation of the original data which is close to the measures you have obtained in some norm, and also fulfills certain constraints. In this case we want to optimize

$$\begin{aligned} \underset{W}{\text{minimize}} \quad & \|W - M\|_F^2 \\ \text{subject to} \quad & \operatorname{rank}(W) = 3. \end{aligned} \tag{4.6}$$

In other words we have a problem where the goal is to find a low-rank approximation of a matrix. To solve this particular problem we use von Neumanns trace theorem

$$|tr(AB)| \leq \sum_{i=1}^{n} \sigma_i(A)\sigma_i(B), \tag{4.7}$$

with equality when $A = U\Sigma_A V^T$ and $B = U\Sigma_B V^T$, where U and V are unitary. That equality holds is seen by using the properties of the scalar product

$$\langle A, B\rangle = \langle U\Sigma_A V^T, U\Sigma_B V^T\rangle = \langle \Sigma_A V^T, U^T U\Sigma_B V^T\rangle = \langle \Sigma_A V^T V, \Sigma_B\rangle = \langle \Sigma_A, \Sigma_B\rangle. \tag{4.8}$$

Here we have used the standard scalar product for matrices

$$\langle A, B\rangle = tr(A^T B). \tag{4.9}$$

Now back to (4.6),

$$\begin{aligned} &\|W - M\|_F^2 = \|M\|_F^2 - 2\langle W, M\rangle + \|M\|_F^2 = \\ &\sum_{i=1}^{n} \sigma_i(M)^2 - 2\langle W, M\rangle + \sum_{i=1}^{n} \sigma_i(W)^2 \geq \\ &\sum_{i=1}^{n} \sigma_i(M)^2 - 2\sigma_i(M)\sigma_i(W) + \sigma_i(W) = \\ &\sum_{i=1}^{n} (\sigma_i(W) - \sigma_i(M))^2. \end{aligned} \tag{4.10}$$

To make $\|W - M\|_F^2$ as small as possible we see that we can choose $W = U_M \Sigma_W V_M^T$ where U_M and V_M are unitary and obtained from SVD of M. From (4.10) we conclude that we must choose the three largest singular values of M and put them into Σ_W and set the rest of the diagonal to zero. This is because of the constraint that we want a solution of rank 3. The decomposition P and X we are seeking can be obtained by extracting the 3 largest singular values from Σ_M and corresponding right and left singular vectors from U_M and V_M and putting them into U', Σ' and V', where $U' \in \mathbb{R}^{2K\times 3}$, $\Sigma' \in \mathbb{R}^{3\times 3}$ and $V' \in \mathbb{R}^{3\times N}$. Then we can define

$$\begin{aligned} P &= U'(\Sigma')^{1/2} \\ X &= (\Sigma')^{1/2} V' \end{aligned} \tag{4.11}$$

to be the camera matrices and 3D-points. This is a well-known result from [31] and is an example of how low-rank approximations can be of use in computer vision. This approach was extended in [6] to the non-rigid setting, again using an affine camera.

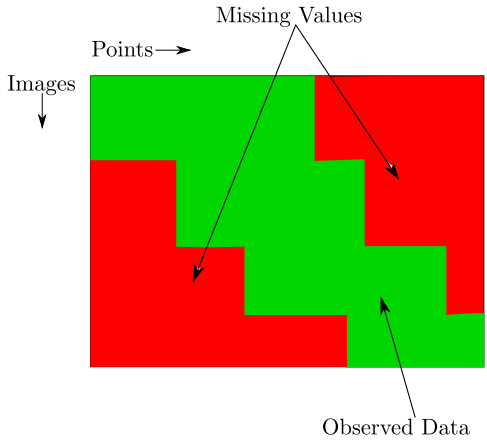


Figure 4.2: When tracking data points for example, the points will not be seen in all images. The green pattern is observed data and the red area is where the points have not been seen. In this illustration the points to the left are seen in the beginning of the sequence and the points to the right are seen in the end of the sequence.

4.1.2 Missing Data Problems

Now assume you have a set of images and through these images you have tracked a number of feature points. Most likely, you will not see all the points in all images throughout the whole sequence. If you put all coordinates in two rows, and then stack them on top of each other for each image, you will get a matrix looking like in Figure 4.2.

To reconstruct the scene, we want to estimate the position of the points in the images where it is not seen. A way of solving this problem would be to minimize an objective of the form

$$\min_X \quad \mu \operatorname{rank}(X) + \|W \odot (X - M)\|_F^2, \tag{4.12}$$

where $W_{ij} = 0$ if M_{ij} is missing and 1 otherwise and $\odot$ denotes element wise multiplication. Typically, $W \odot M$ has a high rank. The first term in (4.12) favors solutions with low rank.

With no missing data in M, (4.12), this could be solved by singular value decomposition of M and choosing the first k significant singular

values. However, missing data makes things much more complicated. A way to tackle this was suggested by [13], where it is shown that the nuclear norm is the convex envelope of the rank function on the set $\{X \in \mathbb{R}^{m\times n} | \sigma_{max}(X) \leq 1\}$. Instead of minimizing (4.12), the rank term can be replaced by the nuclear norm

$$\min_X \quad \|X\|_*^2 + \|W \odot (X - M)\|_F^2. \tag{4.13}$$

In [26, 9] it is shown that this will yield an optimal solution if the location of the missing entries are random. In computer vision though, the data is often highly correlated, and patterns as in 4.2 are common. For example in SfM it appears because tracked points typically occur in a consecutive sequence of images, and then the points go out of view of the camera. Therefore, the nuclear norm might not be the best solution for some problems in computer vision. Moreover, we do not want to restrict ourselves to the set $\{X | \sigma_{max}(X) \leq 1\}$, but instead solving the problem over all matrices. We propose a better convex relaxation of (4.12) with no missing data. Since it is convex, it can be combined with other convex constraints and solved using existing methods for convex optimization.

4.2 Developing the Convex Envelope

To find a solution of (4.12), we derive the convex envelope of

$$f(X) = \mu \operatorname{rank}(X) + \|X - M\|_F^2. \tag{4.14}$$

The tactic to find a solution of (4.12) is to divide the measurement matrix M into sub-blocks with no missing data. On these sub-blocks we will be able to use our convex envelope of (4.14).

In this part we show how we can find the envelope of (4.14). To derive the envelope of (4.14), we start with computing the Fenchel conjugate. Thereafter we compute the conjugate again, which gives us the bi-conjugate which is the convex envelope of f.

The Fenchel conjugate is defined as

$$f^*(Y) = \max_X \quad \langle X, Y \rangle - f(X). \tag{4.15}$$

Writing this as

$$f^*(Y) = \max_X \quad \langle X, Y\rangle - (\mu \operatorname{rank}(X) + \|X - M\|_F^2), \tag{4.16}$$

we can complete squares to obtain

$$\begin{aligned} \langle X, Y\rangle - \|X\|_F^2 + 2\langle X, M\rangle - \|M\|_F^2 = \\ \langle X, Y + 2M\rangle - \|X\|_F^2 - \|M\|_F^2 = \\ -(\|X\|_F^2 - \langle X, Y + 2M\rangle) - \|M\|_F^2 = \\ -\|X - (\frac{1}{2}Y + M)\|_F^2 + \|\frac{1}{2}Y + M\|_F^2 - \|M\|_F^2. \end{aligned} \tag{4.17}$$

The matrix X is of the same size as M, $m \times n$, so the rank cannot be higher than $\min(m, n)$, so by maximizing for each rank k, and over all k, the Fenchel conjugate can be written as

$$\max_k \max_{\substack{X \\ \operatorname{rank}(X)=k}} \quad -\mu\, k - \|X - (\frac{1}{2}Y + M)\|_F^2 + \|\frac{1}{2}Y + M\|_F^2 - \|M\|_F^2. \tag{4.18}$$

Here two terms are independent of X and in the inner maximization the rank k is fixed. For fixed k we can maximize with respect to X by doing SVD of $\frac{1}{2}Y + M = U\Sigma V^T$ and setting $X = U\Sigma_k V^T$, where Σ_k only contains the k largest singular values. Inserting this into (4.18) gives

$$f^*(Y) = \max_k \quad \|\frac{1}{2}Y + M\|_F^2 - \|M\|_F^2 - \sum_{i=k+1}^{n} \sigma_i^2(\frac{1}{2}Y + M) - \mu k \tag{4.19}$$

$$= \max_k \quad \|\frac{1}{2}Y + M\|_F^2 - \|M\|_F^2 - \sum_{i=k+1}^{n} \sigma_i^2(\frac{1}{2}Y + M) - \sum_{i=1}^{k} \mu. \tag{4.20}$$

Looking at equation (4.20), we can deduce that the optimal k must be chosen such that

$$\sigma_k^2(\frac{1}{2}Y + M) \geq \mu \geq \sigma_{k+1}^2(\frac{1}{2}Y + M). \tag{4.21}$$

Using this we can write the conjugate function as

$$f^*(Y) = \|\frac{1}{2}Y + M\|_F^2 - \|M\|_F^2 - \sum_{i=1}^{N} \min(\mu, \sigma_i^2(\frac{1}{2}Y + M)). \quad (4.22)$$

The next step to find the convex envelope of the original function (4.14) is to compute the bi-conjugate. The bi-conjugate is defined as

$$\begin{aligned} f_X^{**} &= \max_Y \langle X, Y\rangle - f^*(Y) \\ &= \max_Y \langle X, Y\rangle - (\|\frac{1}{2}Y + M\|_F^2 - \|M\|_F^2 - \sum_{i=1}^{N} \min(\mu, \sigma_i^2(\frac{1}{2}Y + M))). \end{aligned} \quad (4.23)$$

With the change of variables

$$Z = \frac{1}{2}Y + M, \quad (4.24)$$

this becomes

$$f^{**}(X) = \max_Z \; 2\langle X, Z - M\rangle - \|Z\|_F^2 + \|M\|_F^2 + \sum_{i=1}^{N} \min(\mu, \sigma_i^2(Z)). \quad (4.25)$$

By completing squares, this can be written as

$$f^{**} = \max_Z \; \|X - M\|_F^2 - \|Z - X\|_F^2 + \sum_{i=1}^{N} \min(\mu, \sigma_i^2(Z)). \quad (4.26)$$

The term $\|X - M\|_F^2$ is independent of Z, so we can write it outside the maximization

$$f^{**}(X) = \|X - M\|_F^2 + \max_Z (\sum_{i=1}^{N} \min(\mu, \sigma_i^2(Z)) - \|Z - X\|_F^2). \quad (4.27)$$

We also have that $-\|Z-X\|_F^2 = -\|Z\|_F^2 + 2\langle X, Z\rangle - \|X\|_F^2$, and by von Neumann's trace theorem $\langle X, Z\rangle \leq \sum_{i=1}^{N} \sigma_i(X)\sigma_i(Z)$ we get that

$$-\|Z-X\|_F^2 = -\|Z\|_F^2 + 2\langle Z, X\rangle - \|X\|_F^2 \tag{4.28}$$

$$= -(\sum_{i=1}^{N}(\sigma_i^2(Z) + \sigma_i^2(X))) + 2\langle Z, X\rangle \tag{4.29}$$

$$\leq \sum_{i=1}^{N} -\sigma_i^2(Z) - \sigma_i^2(X) + 2\sigma_i(Z)\sigma_i(X). \tag{4.30}$$

To maximize (4.30), Z should have the same unitary matrices U and V as X have. The problem is now reduced to

$$\max_{Z} \sum_{i=1}^{N} \min(\mu, \sigma_i(Z)) - (\sigma_i(Z) - \sigma_i(X))^2. \tag{4.31}$$

To find the optimal singular values, we can maximize each term individually, since they are separable. Therefore,

$$\max_{\sigma_i(Z)} \min(\mu, \sigma_i^2(Z)) - (\sigma_i(Z) - \sigma_i(X))^2 \tag{4.32}$$

should be solved. There are two cases:

(i) $\sigma_i^2(Z) \leq \mu$ which gives

$$\max_{\sigma_i(Z)} 2\sigma_i(X)\sigma_i(Z) - \sigma_i(X)^2, \tag{4.33}$$

and since $\sigma_i(X) \geq 0$ we have $\sigma_i(Z) = \sqrt{\mu}$.

(ii) $\mu \leq \sigma_i^2(Z)$, which gives the optimization problem

$$\mu - (\sigma_i(Z) - \sigma_i(X))^2, \tag{4.34}$$

which we can trivially solve by setting $\sigma_i(Z) = \sigma_i(X)$.

Together we get the optimal solution

$$\sigma_i(Z) = \max(\sqrt{\mu}, \sigma_i(X)) \, \forall i. \tag{4.35}$$

We now want to use this to derive the bi-conjugate. By using that

$$\min(\mu, \sigma_i^2(Z)) = \min(\mu, \max(\mu, \sigma_i^2(X))) = \mu \tag{4.36}$$

and that

$$\|Z - X\|_F^2 = \sum_{i=1}^{n} \|\sigma_i(Z) - \sigma_i(X)\|_F^2 =$$
$$\sum_{i=1}^{n} \|\max(\sqrt{\mu}, \sigma_i(X)) - \sigma_i(X)\|_F^2 = \sum_{i=1}^{n} [\sqrt{\mu} - \sigma_i(X)]_+^2. \tag{4.37}$$

Note that we also assume that the singular values are sorted in a decreasing order. We can now finally derive the convex envelope of the original optimization problem (4.14). The convex envelope, or bi-conjugate is

$$f^{**}(X) = \|X - M\|_F^2 + \sum_{i=1}^{n} (\mu - [\sqrt{\mu} - \sigma_i(X)]_+^2), \tag{4.38}$$

where

$$[x]_+ = \max(0, x). \tag{4.39}$$

To simplify notation later, we define

$$\mathcal{R}_\mu(X) = \sum_{i=1}^{n} (\mu - [\sqrt{\mu} - \sigma_i(X)]_+^2). \tag{4.40}$$

4.2.1 Missing Data

Even though we have now found a convex relaxation of (4.14), it assumes that the measurement matrix is full, i.e. we have no missing data. For many computer vision problems this assumption is unrealistic and we need to tackle this problem. The key idea behind our approach is to divide the measurement matrix M into sub-blocks, where each sub-block is full. See Figure 4.3 for an illustration. Then we solve the optimization problem for each sub-block, and when all approximations are found, we use these blocks to build the entire matrix X. Later we will see that the

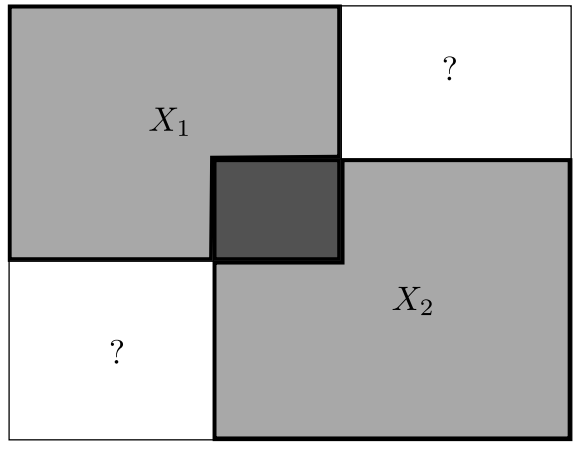


Figure 4.3: To handle missing data we divide the measurement matrix into sub-blocks. To recover the blocks with no data we must have an overlap between the blocks.

rank of the entire matrix is connected to the rank of the blocks we use to create X from.

Let R_i and C_i, $i = 1, \ldots, K$ be a subset of row and column indices for each block i. We define the linear operator $\mathcal{P}_i : \mathbb{R}^{m \times n} \mapsto \mathbb{R}^{|R_i| \times |C_i|}$, which extracts the elements at indices $R_i \times C_i$ and creates a sub-matrix of size $|R_i| \times |C_i|$ with no missing data.

Instead of trying to solve the original problem (4.12), we aim to solve

$$\min_X \sum_{i=1}^{K} \mu_i \operatorname{rank}(\mathcal{P}_i(X)) + \|\mathcal{P}_i(X) - \mathcal{P}_i(M)\|_F^2, \tag{4.41}$$

by replacing it with its convex relaxation

$$\min_X \sum_{i=1}^{K} \mathcal{R}_{\mu_i}(\mathcal{P}_i(X)) + \|\mathcal{P}_i(X) - \mathcal{P}_i(M)\|_F^2. \tag{4.42}$$

By solving the optimization problem we end up with a bunch of sub-blocks. However, we have still not found the entire matrix X, but only a part of it. To recover the whole matrix we use the following lemma

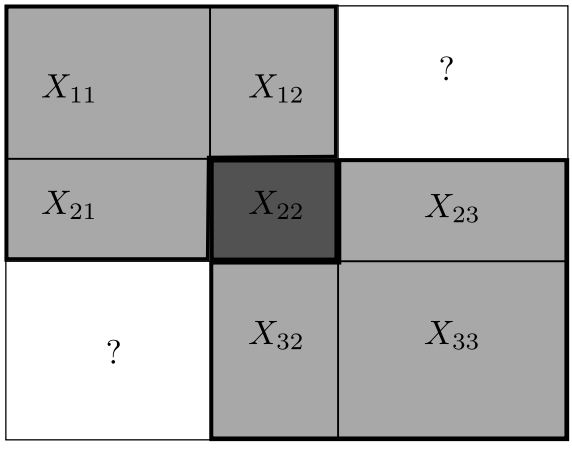


Figure 4.4: Two sub-blocks of a matrix with overlap X_{22}.

Lemma 1. Let X_1 and X_2 be two given matrices with overlap matrix X_{22} as shown in Figure 4.4, and let $r_1 = \text{rank}(X_1)$ and $r_2 = \text{rank}(X_2)$. Suppose that $\text{rank}(X_{22}) = \min(r_1, r_2)$, then there exists a matrix X with $\text{rank}(X) = \max(r_1, r_2)$. Additionally if $\text{rank}(X_{22}) = r_1 = r_2$ then X is unique.

Proof. We will assume (w.l.o.g) that $r_2 \leq r_1$, and look at the block X_2. The overlap X_{22} is of rank r_2 so there are r_2 linearly independent columns in $[X_{22}^T\, X_{32}^T]^T$ and rows in $[X_{22}\, X_{23}]$. Now the rank of X_2 is r_2 and we can find coefficient matrices C_1 and C_2 such that

$$\begin{bmatrix} X_{23} \\ X_{33} \end{bmatrix} = \begin{bmatrix} X_{22} \\ X_{32} \end{bmatrix} C_1 \text{ and } \begin{bmatrix} X_{32} & X_{33} \end{bmatrix} = C_2 \begin{bmatrix} X_{22} & X_{23} \end{bmatrix}. \tag{4.43}$$

We therefore set $X_{13} := X_{12}C_1$ and $X_{31} := C_2X_{21}$. To determine the rank of the resulting X we first look at the number of linearly independent columns. By construction, the columns $[X_{13}^T\, X_{23}^T\, X_{33}^T]^T$ are linear combinations of the other columns, and similarly, the rows $[X_{31}\, X_{32}\, X_{33}]$ are linear combinations of the other rows. Hence, the number of linearly independent columns (or rows) has not increased. Therefore has the same rank as X_1.

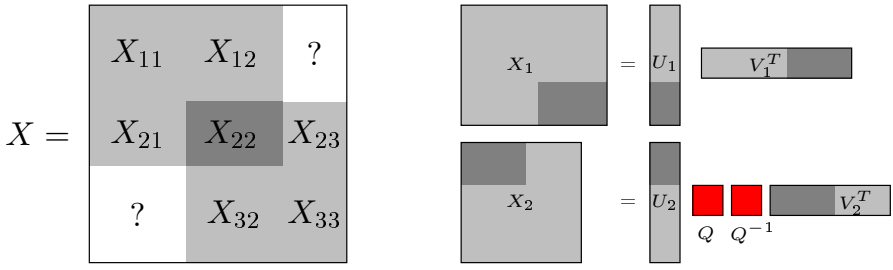


Figure 4.5: Left: The matrix X contains two overlapping blocks X_1 and X_2. The goal is to fill in the missing entries X_{13} and X_{31} such that $\text{rank}(X)$ is kept to a minimum. Right: The low-rank factorizations of the two blocks X_1 and X_2. The overlap is marked in both the blocks and the factorizations.

If $\text{rank}(X_{22}) = \text{rank}(X_1) = \text{rank}(X_2) = r$, then C is unique. To prove this, assume that it is not unique, then

$$X_{13} = X_{12}C_1 \text{ and } \hat{X}_{13} = X_{12}\hat{C}_1. \tag{4.44}$$

Since both C_1 and $\hat{C}_1$ solves

$$\begin{bmatrix} X_{23} \\ X_{33} \end{bmatrix} = \begin{bmatrix} X_{22} \\ X_{32} \end{bmatrix} C \tag{4.45}$$

it follows that $C_1 - \hat{C}_1$ lies in the nullspace of

$$\begin{bmatrix} X_{22} \\ X_{32}. \end{bmatrix} \tag{4.46}$$

By assumption $\text{rank}([X_{12}^T\, X_{22}^T\, X_{32}^T]^T) = \text{rank}([X_{22}^T\, X_{32}^T]^T)$. This implies that they share the same nullspace, so $C_1 - \hat{C}_1$ must lie in the nullspace of X_{12}. This gives $X_{13} = X_{12}C_1 = X_{12}\hat{C}_1 = \hat{X}_{13}$, a contradiction. □

The next step is now to extend the solution on the blocks to the entire matrix. We start with finding a rank r_{max} factorization of X_1 and X_2 using SVD,

$$X_1 = U_1V_1^T \text{ and } X_2 = U_2V_2^T \text{ where } U_k \in \mathbb{R}^{m_k \times r_{max}}, V_k \in \mathbb{R}^{n_k \times r_{max}}. \tag{4.47}$$

However, we have an ambiguity here, because for any non-singular matrix Q, we have

$$U_1 V_1^T = (U_1 Q)(Q^{-1} V_1^T). \tag{4.48}$$

To resolve this, we consider the singular value decompositions of X_1 and X_2. Looking at Figure 4.5, we can see that we can create X_{22} from the sub-matrices by

$$X_{22} = \hat{U}_1 \hat{V}_1^T \tag{4.49}$$
$$X_{22} = \hat{U}_2 \hat{V}_2^T, \tag{4.50}$$

where $\hat{U}_i \hat{V}_i^T$ is the restriction of of the $U_i V_i^T$ to X_{22}.

Due to ambiguity, $\hat{U}_1$ will in general not equal $\hat{U}_2$ and $\hat{V}_1$ will be different from $\hat{V}_2$. Instead we try to find an invertible transformation Q which transforms $\hat{U}_1$ into $\hat{U}_2$, i.e. we seek to solve

$$\hat{U}_1 = \hat{U}_2 Q \tag{4.51}$$
$$\hat{V}_1 = Q^{-1} \hat{V}_2. \tag{4.52}$$

In this case we solve the optimization problem

$$\min_Q \quad \|\hat{U}_1 + \hat{U}_2 Q\|_F^2 + \|Q\hat{V}_1 - \hat{V}_2\|_F^2 \tag{4.53}$$

in a least square sense. With this Q, we can concatenate U_1 and U_2 to $U = [U_1 Q^{-1}, \tilde{U}_2]^T$ where $\tilde{U}_2$ contains the elements which are not common to $U_1 Q^{-1}$. The same way we construct V, and obtain a complete matrix X by

$$X = UV^T. \tag{4.54}$$

In this way we can iteratively combine the sub-blocks and create a single matrix X, and thanks to Lemma 1, we know that our matrix X does not have higher rank then the rank among the sub-matrices.

4.3 Optimization

To actually find a low rank approximation of each sub-block, we must do some optimization. If we would optimize each sub-block independently, we would have no guarantee that they would agree on the overlap. Thus, we must enforce consistency on the overlap as a constraint. This results in the (convex) optimization problem

$$\sum_{i=1}^{K} \mathcal{R}_{\mu_i}(X_i) + \|X_i - \mathcal{P}_i(M)\|_F^2 \tag{4.55}$$

$$\text{subject to}$$

$$\mathcal{P}_i(X) = X_i \quad \forall i = 1...K.$$

An ADMM [5] formulation results in the Lagrangian

$$\sum_{i=1}^{K} \mathcal{R}_{\mu_i}(X_i) + \|X_i - \mathcal{P}_i(M)\|_F^2 + \rho\|X_i - \mathcal{P}_i(X) + \Lambda_i\|_F^2 - \rho\|\Lambda_i\|_F^2. \tag{4.56}$$

At each iteration we solve and update the variables by

$$X_i^{t+1} = \arg\min_{X_i} \quad \mathcal{R}_{\mu_i}(X_i) + \|X_i - \mathcal{P}_i(M)\|_F^2 + \rho\|X_i - \mathcal{P}_i(X^t) + \Lambda_i^t\|_F^2, \tag{4.57}$$

for $i = 1...K$ and

$$X^{t+1} = \arg\min_{X} \sum_{i=1}^{K} \rho\|X_i^{t+1} - \mathcal{P}_i(X) + \Lambda_i^t\|_F^2 \tag{4.58}$$

$$\Lambda_i^{t+1} = \Lambda_i^t + X_i^{t+1} - \mathcal{P}_i(X^{t+1}). \tag{4.59}$$

4.3.1 Proximal Operator

To find X^{t+1} is a simple least squares problem, whereas to find the optimal sub-block X_i^{t+1} is more complicated. We need to solve

$$\min_{X_i} F(X_i) = \min_{X_i} \mathcal{R}_{\mu_i}(X_i) + \|X_i - \mathcal{P}_i(M)\|_F^2 + \rho\|X_i - \mathcal{P}_i(X^t) + \Lambda_i^t\|_F^2. \tag{4.60}$$

This objective can be rewritten as

$$\sum_{j=1}^{N}(-[\sqrt{\mu}-\sigma_j(X_i)]_+^2)+(1+\rho)\|X_i\|_F^2-2\langle\mathcal{P}_i(M)+\rho(\mathcal{P}_i(X^t)-\Lambda_i^t),X_i\rangle \tag{4.61}$$

and simplified to

$$F(X_i)=G(X_i)-2\langle Y,X_i\rangle. \tag{4.62}$$

where

$$G(X_i)=\sum_{j=1}^{n}(-[\sqrt{\mu}-\sigma_j(X_i)]_+^2)+(1+\rho)\|X_i\|_F^2 \tag{4.63}$$

$$Y=\mathcal{P}_i(M)+\rho(\mathcal{P}_i(X^t)-\Lambda_i^t). \tag{4.64}$$

Since F is convex, it is sufficient to find where $0\in\partial F$.

For this we need some theory about unitary invariant matrix functions, but we start with defining the function

$$g_i(\sigma)=-[\sqrt{\mu}-|\sigma|]_+^2+(1+\rho)\sigma^2. \tag{4.65}$$

To see that $g_i(\sigma)$ is convex, one sees that g_i is a special case of $f^{**}(X)$, where $X\in\mathbb{R}^{1\times 1}$. Since the bi-conjugate $f^{**}(X)$ is convex, g_i must be convex. With our definition of g_i, we can now define

$$g(\boldsymbol{\sigma})=\sum_{i=1}^{n}g_i(\sigma_i), \tag{4.66}$$

which is absolutely symmetric and convex, since g_i is convex for all $i=1...n$. Now $G(X)=g\circ\boldsymbol{\sigma}(X)$, where

$$\boldsymbol{\sigma}(X)=(\sigma_1(X),\sigma_2(X),...,\sigma_n(X))^T. \tag{4.67}$$

To derive the subgradients we make use of the following lemma in [20]:

Lemma 2. (Characterization of Subgradients) Let us suppose that the function $f:\mathbb{R}^q\to(-\infty,\infty]$ is absolutely symmetric, and that the $m\times n$ matrix X has $\sigma(X)$ in dom f. Then the $m\times n$ matrix Y lies in $\partial(f\circ\sigma)(X)$

if and only if $\sigma(Y)$ lies in $\partial f \circ \sigma(X)$ and there exists a simultaneous singular value decomposition of the form

$$X = V(\mathrm{Diag}(\sigma(X)))U, \tag{4.68}$$
$$Y = V(\mathrm{Diag}(\sigma(Y)))U \tag{4.69}$$

where U and V are unitary matrices. In fact

$$\partial(f \circ \sigma)(X) = \{V(\mathrm{Diag}(\mu))U | \mu \in \partial f(\sigma(X)), X = V(\mathrm{Diag}(\sigma(X)))U\}. \tag{4.70}$$

So to compute the subdifferential of $G(X)$, we compute the subdifferential of $g(\boldsymbol{\sigma}(X))$ instead. Since $g(\boldsymbol{\sigma}(X))$ is a sum of one-dimensional functions we can compute the subgradient by treating each term individually. For each g_i we have

$$\partial g_i(\sigma) = \begin{cases} 2\mathrm{sgn}(\sigma)[\sqrt{\mu} - |\sigma|]_+ + 2(1+\rho)\sigma & \sigma \neq 0 \\ [-2\sqrt{\mu}, 2\sqrt{\sigma}] & \sigma = 0. \end{cases} \tag{4.71}$$

Now, the aim is to solve $0 \in \partial F(X)$, which is equivalent of solving $2Y \in \partial G(X)$. Remember that $F(X) = G(X) - 2\langle X, Y\rangle$.

If $Y = U\mathrm{Diag}(\sigma(Y))V^T$ then we can verify that $X = U\mathrm{Diag}(\sigma(X))V^T$ where

$$\sigma_i(X) = \begin{cases} \frac{\sigma(Y)}{1+\rho} & \text{if } \sigma_i(Y) \geq (1+\rho)\sqrt{\mu} \\ \frac{\sigma_i(Y) - \sqrt{\mu}}{\rho} & \text{if } \sqrt{\mu} \leq \sigma_i(Y) \leq (1+\rho)\sqrt{\mu} \\ 0 & \text{if } \sigma_i(Y) \leq \sqrt{\mu} \end{cases}, \tag{4.72}$$

is such that $2Y \in \partial G(X)$.

Since

$$\partial G(X) = U\mathrm{diag}(\partial g \circ \boldsymbol{\sigma}(X))V^T \tag{4.73}$$

we have to check that

$$2\sigma_i(Y) \in \partial g_i(\sigma_i(X)) \tag{4.74}$$

for all i.

For all $\sigma_i(X) \neq 0$ the subdifferential for positive σ_i is

$$\frac{\partial g_i}{\partial \sigma}(\sigma) = 2[\sqrt{\mu} - \sigma]_+ + 2(1+\rho)\sigma. \tag{4.75}$$

In case $\sigma_i(X) = 0$ the subdifferential is the interval

$$\frac{\partial g_i}{\partial \sigma}(0) = [-2\sqrt{\mu}, 2\sqrt{\mu}]. \tag{4.76}$$

Now there are three cases to check:
1. $\sigma_i(Y) \geq (1+\rho)\sqrt{\mu}$: $\sigma_i(X) = \frac{\sigma_i(Y)}{1+\rho}$ gives

$$\frac{\partial g_i}{\partial \sigma}\left(\frac{\sigma_i(Y)}{1+\rho}\right) = 2\left[\sqrt{\mu} - \frac{\sigma_i(Y)}{1+\rho}\right]_+ + 2(1+\rho)\frac{\sigma_i(Y)}{1+\rho} = 2\sigma_i(Y). \tag{4.77}$$

2. $\sqrt{\mu} \leq \sigma_i(Y) \leq (1+\rho)\mu$: $\sigma_i(X) = \frac{(\sigma_i(Y)-\sqrt{\mu})}{\rho}$ gives

$$\frac{\partial g_i}{\partial \sigma}\left(\frac{(\sigma_i(Y)-\sqrt{\mu})}{\rho}\right) = \tag{4.78}$$

$$2\left(\sqrt{\mu} - \frac{(\sigma_i(Y)-\sqrt{\mu})}{\rho}\right) + 2(1+\rho)\frac{(\sigma_i(Y)-\sqrt{\mu})}{\rho} = 2\sigma_i(Y) \tag{4.79}$$

If $\frac{(\sigma_i(Y)-\sqrt{\mu})}{\rho} = 0$, then

$$\frac{\partial g_i}{\partial \sigma}(0) = 2\sqrt{\mu} = 2\sigma_i(Y), \tag{4.80}$$

and it is in (4.76).

3: $\sigma_i(Y) \leq \sqrt{\mu}$: Here $\sigma_i(X) = 0$ and therefore $2\sigma_i(Y) \leq 2\sqrt{\mu}$ is contained in the subdifferential (4.76).

4.3.2 Experiments

To evaluate this method we start with using synthetic data and compare the results with other methods, such as the nuclear norm and other local optimization methods. First the convex relaxation is evaluated using synthetic data. We define

$$f(X) = \sum_{i=1}^{K} \mu_i \text{ rank}(\mathcal{P}_i(X)) + \|\mathcal{P}_i(X) - \mathcal{P}_i(M)\|_F^2 \tag{4.81}$$

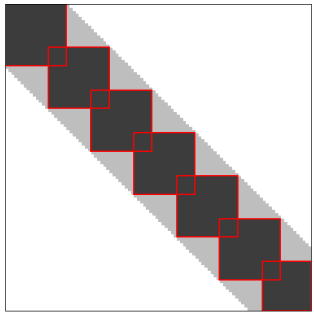


Figure 4.6: The pattern in the synthetic experiments and the overlap.

and our derived convex relaxation

$$f_{\mathcal{R}}(X) = \sum_{i=1}^{K} \mathcal{R}_{\mu_i}(\mathcal{P}_i(X)) + \|\mathcal{P}_i(X) - \mathcal{P}_i(M)\|_F^2. \tag{4.82}$$

Then random rank 3 matrices was generated of dimension 100×100 by sampling $U, V \in \mathbb{R}^{100\times 3}$ from a Gaussian distribution with zero mean and unit variance. Then M was formed by $M = UV^T$. The observation matrix W was chosen to be a band-diagonal matrix with bandwidth 40 similar to Figure 4.6. The blocks were laid out such that the overlap was 6×6 and contained no missing data. Then the solution

$$X_{\mathcal{R}}^* = \arg\min_X f_{\mathcal{R}}(X), \tag{4.83}$$

where μ_i was set to 1 $\forall i$ was found. To M, we added different levels of Gaussian noise, and the test was repeated 1000 times for each noise level. In Figure 4.7 we plot the average error for each noise level of $f(X_{\mathcal{R}}^*)$ and $f_{\mathcal{R}}(X_{\mathcal{R}}^*)$, note that if $f(X_{\mathcal{R}}^*) = f_{\mathcal{R}}(X_{\mathcal{R}}^*)$, then the global minimizer is found.

To compare with other methods we substitute the rank function with the nuclear norm. The nuclear norm is also convex, so it can also be used in our block decomposition framework. Hence, the same experiment was made with the objective

$$f_N(X) = \sum_{i=1}^{K} \mu_i \|\mathcal{P}_i(X)\|_* + \|\mathcal{P}_i(X) - \mathcal{P}_i(M)\|_F^2. \tag{4.84}$$

The results are shown in the top graph of Figure 4.7. Note that the constraint $\sigma_{max}(\mathcal{P}_i(X)) \leq 1$ can be violated, so f_N is not necessarily a lower bound on f.

However, there also exists other methods for obtaining low-rank approximations. We compare to two non-convex methods, namely OptSpace [19] and Truncated Nuclear Norm Regularization (TNNR), [15]. Both OptSpace and TNNR are local methods, that is, they are not convex. OptSpace is based on local optimization on Grassmanian manifolds and in TNNR an energy which penalizes the last $(n - r)$ singular values is minimized. The experiments was made the same way as above with randomly chosen matrices $U, V \in \mathbb{R}^{3\times 100}$. Both these approaches tries to estimate a fixed rank approximation, that is the rank is set before hand. In contrast our method is a trade-off between the rank and the data-term $\|X - M\|_F^2$. We therefore iterate our method over μ_i to get the same rank. The average values of $\|W \odot (X - M)\|_F^2$ are shown in 4.7. Even though the plots suggests that our method is better, often both OptSpace and TNNR gives similar or better results than our method. However, local minima results in a higher average than our method, but on the other hand, since OptSpace and TNNR minimizes the original function directly, their solution will be at least as good as ours when it does not get stuck in a local minima.

Real Data

As described earlier in this chapter, one application for low-rank approximation is rigid structure from motion. We test our method on the well-known Oxford dinosaur sequence. The measurement matrix M will contain the tracked 2D coordinates of the tracked 3D points, as described in the beginning of this chapter. Since we cannot handle outliers, we choose an outlier free subset of this dataset consisting of 321 3D points where each point is seen in at least six images. The observation matrix W is shown in the left of Figure 4.9 and clearly demonstrates the band diagonal pattern for these structure from motion problems. For comparison we also solve the problem using the nuclear norm and bundle adjustment. The nuclear norm formulation is

$$\min_X \mu\|X\|_* + \|W \odot (X - M)\|_F^2 \tag{4.85}$$

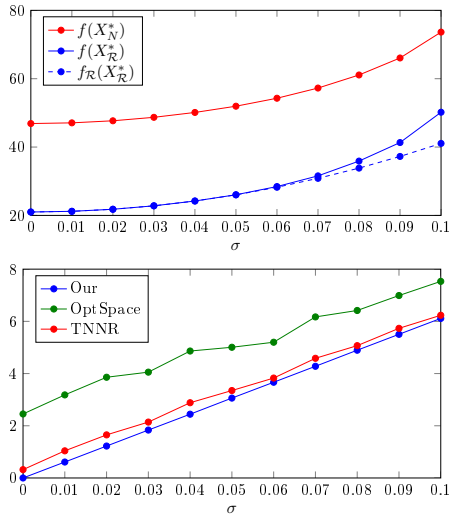
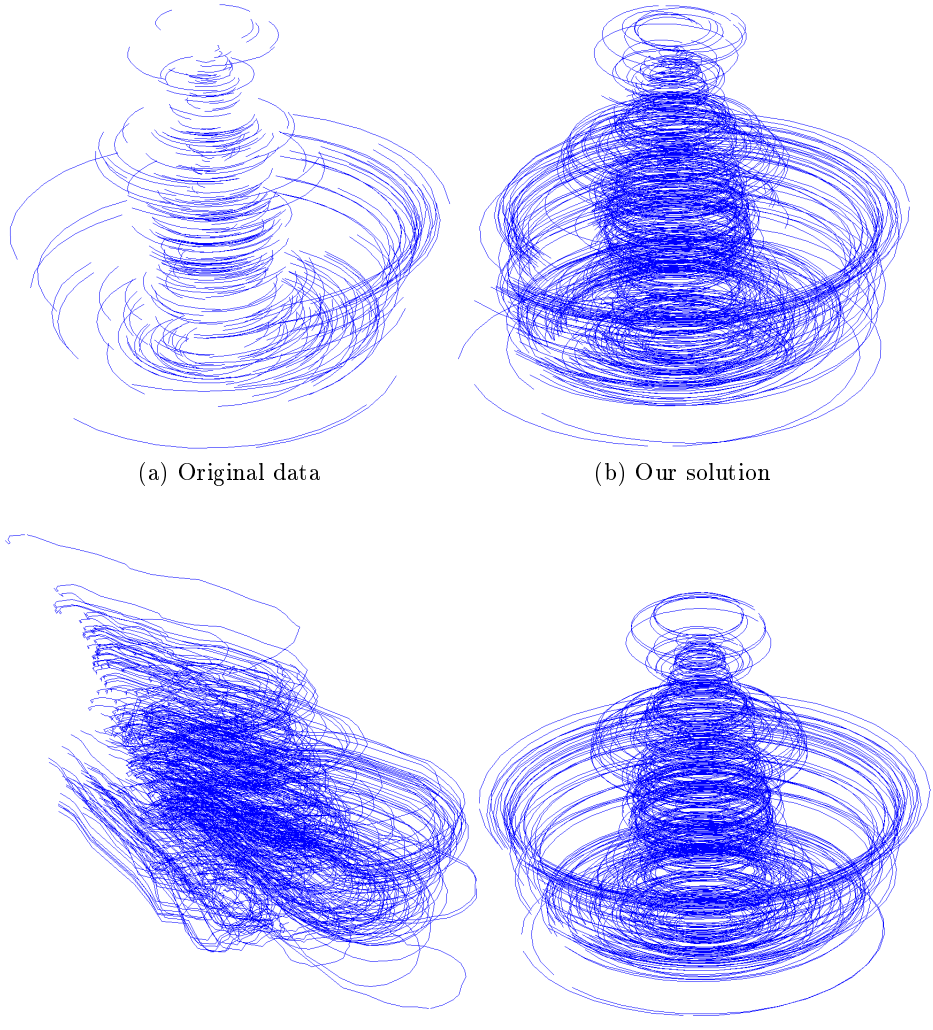


Figure 4.7: Evaluation of the proposed formulation on synthetic data with varying noise levels. Top: Comparison of our convex relaxation solution $X_\mathcal{R}^*$ with the nuclear norm X_N^*. Bottom: The error $\|W \odot (X - M)\|_F$ for varying noise levels.

which optimize with the shrinkage operator and ADMM. The results are show in Figure 4.8. As seen in the figure, the nuclear norm performs much worse than our method. This clearly shows the effect of penalizing all singular values rather than just the singular values smaller than $\sqrt{\mu}$. The perspective projection gives a more visually appealing result. The errors $||W \odot (X - M)||_F$ for the three solutions were 73.2 (our), 1902.5 (nuclear) and 116.2 (perspective), so even if the perspective projection looks better, our solution is a better low-rank approximation of the original data.

Linear Shape Models

Another application is linear shape models, which are common in non-rigid structure from motion. Let X_f be the 2D- or 3D- coordinates of N tracked points in frame f. The model we have assumes that in each frame the coordinates are a linear combination of some unknown shape



(a) Original data

(b) Our solution

(c) Nuclear norm

(d) Perspective projection

Figure 4.8

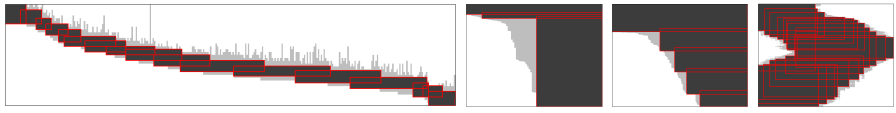
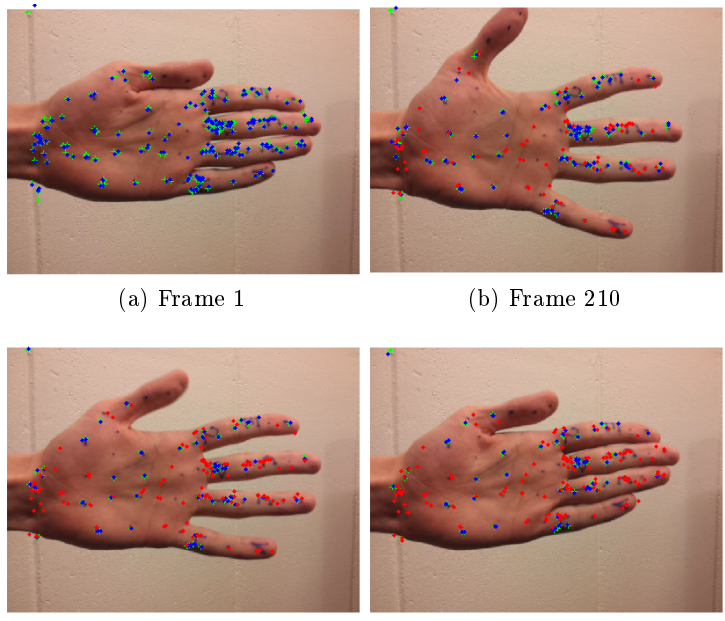


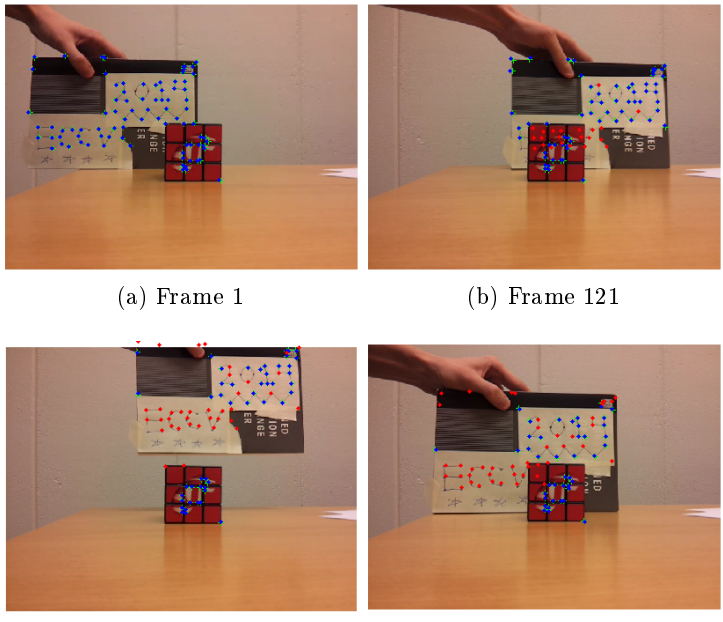
Figure 4.9: Structured data patterns of observations (matrix W) and sub-blocks for the dino, book, hand and banner sequences.



(a) Frame 1 (b) Frame 210

(c) Frame 340 (d) Frame 371

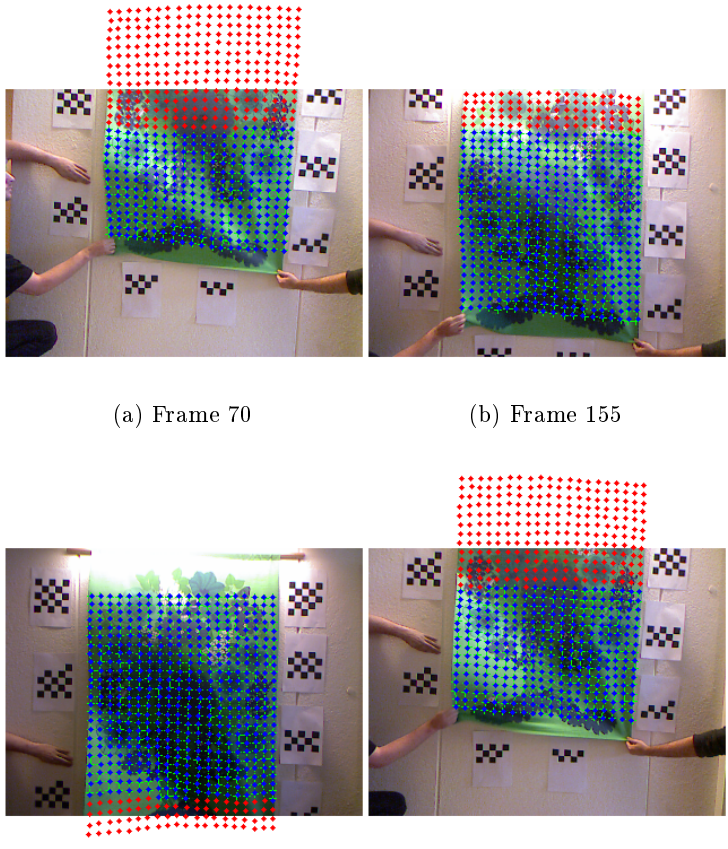
Figure 4.10: Frames 1, 210, 340 and 371 of the hand sequence. The solution has rank 5.



(a) Frame 1 (b) Frame 121

(c) 380 (d) 668

Figure 4.11: Frames 1, 121, 380 and 668 of the book sequence. The solution has rank 3.

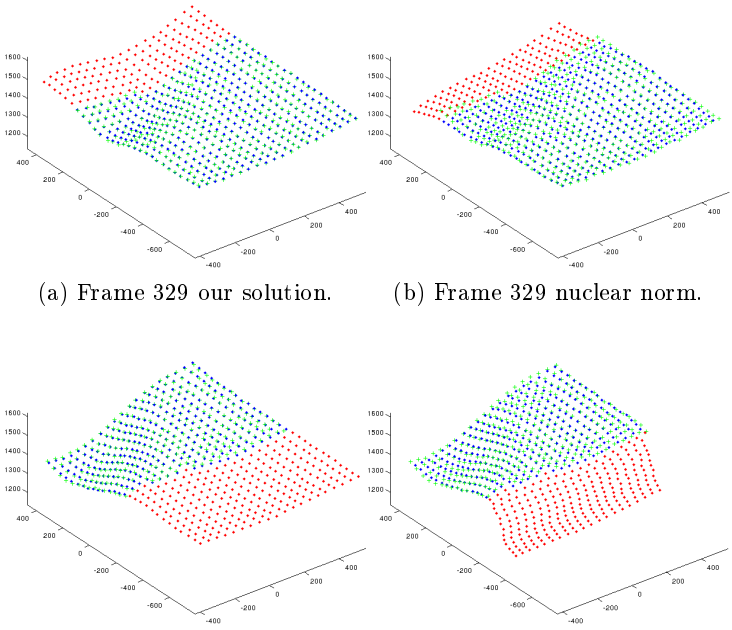


(a) Frame 70 (b) Frame 155

(c) Frame357 (d) Frame 650

Figure 4.12: Frames 70, 155, 357 and 650 of the banner sequence. The solution has rank 9.

Figure 4.13: The result from the banner experiment.



(a) Frame 329 our solution. (b) Frame 329 nuclear norm.

(c) Frame 650 our solution. (d) Frame 650 nuclear norm.

Figure 4.14: From left to right: Our solution (frame 329), nuclear norm solution (frame 329), our solution (frame 650), nuclear norm solution (frame 650).

basis vectors, i.e.

$$X_f = \sum_{k=1}^{K} C_{fk} S_k, \quad f = 1..F, \tag{4.86}$$

where S_k is the shape basis and C is the coefficient matrix. The measurement matrix M is obtained by stacking all tracked points for each frame on top of each other. In other words, the first two rows in M are the tracked points for frame 1 and so on. The elements of the observation matrix W indicates if the point was successfully tracked in that frame or not. We do not want more basis elements than necessary to describe the movement. Consequently, we seek a low-rank approximation of the measurement matrix M.

In the first two datasets, Book and Hand, we track points through a video sequence using the standard KLT-tracker [21]. Due to tracking failure and occlusion, we get missing data which forms the patterns shown in Figure 4.9 where the selected blocks are shown as well. Using (4.42), we find a low rank approximation for each dataset. The results can be seen in Figure 4.10 and Figure 4.11. The blue points are the reconstructed points that were successfully tracked through the entire sequence. The red dots are the reconstructed missing data, and the green crosses are the original measured data. As can be seen in both sequences, the derived solution gives reasonable results.

In the third experiment, we used a Kinect camera to record a video of a moving piece of fabric. This gives a 3D-grid of points and missing data comes from a limited field of view and missing depth values in the depth image. To register all points in a common coordinate system, we track the cameras using the patterns on the wall (Figure 4.13). The obtained solution can be seen in Figure 4.13 and Figure 4.14. For comparison, the results for the nuclear norm is also provided. It is clear that our solution yields a more realistic reconstruction of the movement, in contrast to the nuclear norm which has a bias toward small singular values.

4.3.3 Discussion

In this section we have derived a convex envelope for an energy function

$$\mu \text{ rank}(X) + \|X - M\|_F^2, \tag{4.87}$$

which can be used in situation with missing data, when the missing data has a regular pattern from which sub-blocks can be extracted. The advantage with our method is that it is convex and that it finds a low rank approximation close the the measured matrix M. This is possible since our algorithm only penalizes singular values smaller than μ, where the nuclear norm penalizes all singular values. For larger problems, a major obstacle can be to choose parameter μ_i for each block. Our method is sensitive to outliers, which is also a drawback.

Bibliography

[1] KinectFusion Implementation in the Point Cloud Library (PCL). http://svn.pointclouds.org/pcl/trunk/. 21, 41

[2] Herbert Bay, Tinne Tuytelaars, and Luc Van Gool. Surf: Speeded up robust features. In In ECCV, pages 404–417, 2006. 55

[3] P.J. Besl and N.D. McKay. A method for registration of 3-D shapes. IEEE Trans. Pattern Anal. Mach. Intell., 14(2):239–256, 1992. 13, 17

[4] G. Blais and M.D. Levine. Registering multiview range data to create 3D computer objects. IEEE Trans. Pattern Anal. Mach. Intell., 17:820–824, 1993. 44

[5] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. Found. Trends Mach. Learn., 3(1):1–122, January 2011. ISSN 1935-8237. doi: 10.1561/2200000016. URL http://dx.doi.org/10.1561/2200000016. 73

[6] Christoph Bregler, Aaron Hertzmann, and Henning Biermann. Recovering non-rigid 3d shape from image streams. pages 690–696, 1999. 62

[7] E. Bylow. Camera tracking using a signed distance function. Master's thesis, 2012. 41

[8] Erik Bylow, Carl Olsson, and Fredrik Kahl. Robust camera tracking by combining color and depth measurements. In Pattern Recogni-

tion (ICPR), 2014 22nd International Conference on, pages 4038–4043. IEEE, 2014. 51

[9] Emmanuel J. Candès, Xiaodong Li, Yi Ma, and John Wright. Robust principal component analysis? J. ACM, 58(3):11:1–11:37, June 2011. ISSN 0004-5411. doi: 10.1145/1970392.1970395. URL http://doi.acm.org/10.1145/1970392.1970395. 64

[10] Y. Chen and G. Medioni. Object modelling by registration of multiple range images. Image Vision Comput., 10(3):145–155, 1992. 15

[11] B. Curless and M. Levoy. A volumetric method for building complex models from range images. In SIGGRAPH, 1996. 17, 22, 25

[12] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. An evaluation of the RGB-D SLAM system. In ICRA, May 2012. 18, 41

[13] Maryam Fazel, Haitham Hindi, and Stephen P Boyd. A rank minimization heuristic with application to minimum order system approximation. In American Control Conference, 2001. Proceedings of the 2001, volume 6, pages 4734–4739. IEEE, 2001. 64

[14] R. I. Hartley and A. Zisserman. Multiple View Geometry in Computer Vision. Cambridge University Press, ISBN: 0521540518, second edition, 2004. 8

[15] Yao Hu, Debing Zhang, Jieping Ye, Xuelong Li, and Xiaofei He. Fast and accurate matrix completion via truncated nuclear norm regularization. IEEE Trans. Pattern Anal. Mach. Intell., 35(9): 2117–2130, 2013. doi: 10.1109/TPAMI.2012.271. URL http://doi.ieeecomputersociety.org/10.1109/TPAMI.2012.271. 78

[16] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera. In Proceedings of the 24th annual ACM symposium on User interface software and technology, UIST '11, pages

559–568, New York, NY, USA, 2011. ACM. ISBN 978-1-4503-0716-1. doi: 10.1145/2047196.2047270. URL http://doi.acm.org/10.1145/2047196.2047270. 12, 21

[17] C. Kerl, J. Sturm, and D. Cremers. Robust odometry estimation for rgb-d cameras. In ICRA, 2013. 17

[18] Christian Kerl, Jürgen Sturm, and Daniel Cremers. Dense visual slam for rgb-d cameras. 2013. 17

[19] Raghunandan H. Keshavan, Andrea Montanari, and Sewoong Oh. Matrix completion from a few entries. IEEE Trans. Inf. Theor., 56 (6):2980–2998, June 2010. ISSN 0018-9448. doi: 10.1109/TIT.2010.2046205. URL http://dx.doi.org/10.1109/TIT.2010.2046205. 78

[20] A. S. Lewis. The convex analysis of unitarily invariant matrix functions, 1995. 74

[21] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. pages 674–679, 1981. 85

[22] Y. Ma, S. Soatto, J. Kosecka, and S. Sastry. An Invitation to 3D Vision: From Images to Geometric Models. Springer Verlag, 2003. 31

[23] R.A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A.J. Davison, P. Kohli, J. Shotton, S. Hodges, and A.W. Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. October 2011. 2, 12, 17, 19, 21, 34, 41

[24] Stanley Osher and Ronald P. Fedkiw. Level set methods and dynamic implicit surfaces. Applied mathematical science. Springer, New York, N.Y., 2003. ISBN 0-387-95482-1. URL http://opac.inria.fr/record=b1099358. 12

[25] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In

Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH '00, pages 335–342, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co. ISBN 1-58113-208-5. doi: 10.1145/344779.344936. URL http://dx.doi.org/10.1145/344779.344936. 19

[26] Benjamin Recht, Maryam Fazel, and Pablo A. Parrilo. Guaranteed minimum-rank solutions of linear matrix equations via nuclear norm minimization. SIAM Rev., 52(3):471–501, August 2010. ISSN 0036-1445. doi: 10.1137/070697835. URL http://dx.doi.org/10.1137/070697835. 64

[27] Szymon Rusinkiewicz and Marc Levoy. Efficient variants of the ICP algorithm. In Third International Conference on 3D Digital Imaging and Modeling (3DIM), June 2001. 14, 17

[28] 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. 16, 17, 19

[29] F. Steinbruecker, C. Kerl, J. Sturm, and D. Cremers. Large-scale multi-resolution surface reconstruction from rgb-d sequences. In IEEE International Conference on Computer Vision (ICCV), Sydney, Australia, 2013. 12, 16, 51

[30] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In IROS, 2012. 21, 35, 41, 51, 55

[31] Carlo Tomasi and Takeo Kanade. Shape and motion from image streams under orthography: A factorization method. Int. J. Comput. Vision, 9(2):137–154, November 1992. ISSN 0920-5691. doi: 10.1007/BF00129684. URL http://dx.doi.org/10.1007/BF00129684. 59, 62

[32] T. Whelan, H. Johannsson, M. Kaess, J.J. Leonard, and J.B. McDonald. Robust tracking for real-time dense RGB-D mapping with Kintinuous. Technical report, MIT, 2012. 18

[33] T. Whelan, H. Johannsson, M. Kaess, J.J. Leonard, and J.B. McDonald. Robust real-time visual odometry for dense RGB-D mapping. In IEEE Intl. Conf. on Robotics and Automation, ICRA, Karlsruhe, Germany, May 2013. 18

[34] T. Whelan, J. McDonald, H. Johannsson, M. Kaess, and J.J. Leonard. Robust tracking for dense RGB-D mapping with kintinuous. In Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA), 2013. 12, 19

[35] Thomas Whelan, Stefan Leutenegger, Renato F Salas-Moreno, Ben Glocker, and Andrew J Davison. Elasticfusion: Dense slam without a pose graph. In Robotics: Science and Systems (RSS), 2015. 18, 19

[36] Kai M. Wurm, Armin Hornung, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems. In ICRA 2010 workshop, 2010. 11, 18, 41