ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Optimization-based Motion Blur aware Camera Pose Estimation

Semester Thesis

Johann Diep

Fall Term 2018

Advisor: Peidong Liu

Prof. Marc Pollefeys, Computer Vision and Geometry Group, ETH Zürich

Abstract

A large percentage of mobile robotics applications adopt vision-based estimation of their positions within the environment in order to perform automated or semi-automated tasks. Commonly used methods, such as visual odometry (VO) or simultaneous localization and mapping (SLAM), require the input images to be sharp. In practice, these kinds of methods tend to fail or provide inaccurate results in the present of motion blurring due to rapid movements in combination with a long exposure time.

This semester thesis discusses the implementation of a motion blur aware camera pose tracker. Given a blurred input image, the algorithm estimates the pose at where it was taken from via an optimization process. In order to solve the optimization problem, the algorithm requires a sharp reference image with its corresponding depth map to be available as an initial requirement. Additionally, the ground-truth reference pose, the camera extrinsic and intrinsic data as well as the exposure time must be known. Hence, the work is split up in several iterating modules.

At first, a polygon-mesh is constructed from the reference depth map as a 3D representation of the environment. By constructing 3D pointclouds using a depth map renderer[1] and backprojecting the landmarks on the reference plane, sharp images are generated by sampling the intensities at the corresponding pixel locations in the reference image. Thereby, multiple images are generated along a specific interpolated camera trajectory starting from the reference pose. In a further step, the intensities of the sampled images are averaged in order to create an artificial blurred image. The deviation of the generated blurred image with the blurred input image is measured with an appropriate objective function. Thus, the pose of the blurred input image is estimated using a stochastic gradient descent algorithm.

The presented method was evaluated on a rendered 3D dataset. Given a cardinal directed deflection from the searched pose, the algorithm achieved convergence within 170 seconds[2].

[1] Rendering is the process of generating a 2D image from a 3D representation.

[2] Running on a 2.5 GHz *Intel i5-7300HQ* machine with a *Nvidia GeForce GTX 1050* graphics processor.

Contents

Contents **iii**

1 Introduction **1**
1.1 Motivation 1
1.2 Outline 1
1.3 Modules 2
1.4 Related Work 2

2 Conventions **5**
2.1 Rendered Dataset 5
2.1.1 Coordinate Systems 5
2.2 Pose Representation 6
2.3 Pinhole Camera Model Approximation 7

3 Algorithm **9**
3.1 Initialization 10
3.1.1 Adding Disturbance 10
3.1.2 Logarithmic Mapping 10
3.2 3D Representation 11
3.2.1 Formats 11
3.2.2 Polygon-Mesh Construction 11
3.2.3 Downscaling 13
3.3 Generator 14
3.3.1 Exponential Mapping 15
3.3.2 Depth Rendering 15
3.3.3 Grid-Sampling 16
3.4 Blurrer 17
3.4.1 Blur Model Interpolation 17
3.4.2 Intensity Averaging 18
3.5 Optimization 18

3.5.1 Cost Function 19
3.5.2 Adam Algorithm 19

4 Results **21**
4.1 Runtime-Precision Tradeoff 21
4.2 Robustness against Translational Disturbance 22
4.2.1 L2-norm 22
4.2.2 L1-norm 23
4.3 Influence of the Number of Intermediate Images 24
4.4 Robustness against Angular Disturbance 25
4.5 Pyramid Resolution Approach 26
4.6 Disturbed Reference Depth Map 27

5 Conclusion **29**
5.1 Achievements 29
5.2 Outlook 30
5.2.1 Preprocessing via Convolutional Neural Network . . . 30
5.2.2 Selective Polygon-Mesh Surface Reduction 30
5.2.3 Different Dataset 30

A Appendix **31**
A.1 Repository 31
A.1.1 Installation 31
A.1.2 Code Structure 32
A.1.3 Docker 33

Bibliography **35**

Chapter 1

Introduction

1.1 Motivation

Many current robotics applications, especially autonomous mobile robots, require a robust position and orientation estimation in order to perform autonomous or semi-autonomous tasks. Vision-based techniques such as visual odometry (VO) or simultaneous location and mapping (SLAM) are used to estimate camera movements from a sequence of images. For large scale deployment as the fundamental module for pose estimation, these methods need to work robustly under any circumstances. Unfortunately, they come with practical requirements. In order to extract apparent motions, some conditions must be fulfilled, such as the presence of sufficient illuminations and textures in the environment, large enough scene overlaps between consecutive frames or the dominance of static scenes. The work in the context of this semester thesis is addressing the latter case, in particular the occurrence of motion blur due to rapid camera movements in combination with a long exposure time.

1.2 Outline

This report describes the implementation of an optimization-based motion blur aware camera pose tracking method within the *PyTorch*[1] framework. Given a sharp reference image with its corresponding depth map and ground-truth pose, the intrinsic and extrinsic parameters, the exposure time of the camera and an initial guess on the blurred input image camera pose, the algorithm converges to the true pose where the blurred input image was taken from after a finite number of iterations. The proposed method can be employed as a fail-safe module in conventional VO or SLAM techniques.

[1] Open-source machine learning library for *Python* (Source: https://pytorch.org/ 27. January 2019).

In order to lower the effective runtime, the algorithm is designed to utilize graphics processing unit (GPU) resources by applying *CUDA*[2] tensor types.

1.3 Modules

Based on the reference depth map, a 3D polygon-mesh of the environment is constructed. The subsequent pipeline, which is initialized by an educated guess of the blurred input image camera pose, consists of three interacting modules running sequentially within a loop. These are described in the following (a complete flow chart of the pipeline and further explanation of the implementation keypoints can be found in chapter 3):

- Generator: This module is responsible for generating new images at arbitrary poses based on the reference image. First, a dense depth map at the desired pose is rendered using the constructed polygon-mesh. Corresponding pixels of the new and reference image are matched by creating a 3D pointcloud at the new pose and projecting them back to the reference plane according to the pinhole camera model. Pixel information of the new image are determined by sampling the intensities at the corresponding pixel locations in the reference image.
- Blurrer: In order to generate an artificial blurred image, multiple images along a defined camera trajectory are sampled using the generator and averaged in pixel intensities.
- Optimization: The degree of similarity between the artificial and the blurred input image is measured by a cost function. Thus, the previous guess of the blurred input image camera pose is updated via a stochastic optimization process in an attempt to minimize the objective function.

1.4 Related Work

Template matching methods rely on warping all pixels from a template image, where the goal is to find a set of warping parameters which minimizes the difference between the warped and a reference image. A common method of this kind was described in 1981 by Lucas and Kanade [1]. Due to ideal assumptions, their warp function has trouble working with images affected by motion blur. Park and Lee (2017, [2]) proposed a blur model, which synthesizes a blurred image by approximating intermediate images during the shutter time and along the camera path.

[2] *CUDA* is a parallel computing platform and programming model invented by *NVIDIA*. It enables increases in computing performance by harnessing the power of the GPU (Source: http://developer.nvidia.com/cuda-90-download-archive/ 27. December 2018).

Based on this concept, Aebi, Milano and Schnetzler (2018, [3]) reported a method for camera pose estimation through image warping and optimization[3]. For generating the intermediate images, they implemented a reprojector module. Starting from a sharp reference image and its corresponding depth map, a 3D pointcloud was constructed and backprojected on to the image plane at an arbitrary pose. Since multiple landmarks can be reprojected to the same pixel, they only considered the point with the smallest distance to the image plane. Inevitably, this process leads to sparse images which influences estimation accuracy. Their algorithm was the basis of this thesis which highlights the impact of approximated depths at every frame. A comparison on the estimation precision can be seen in chapter 4.

[3] Developed in the context of the lecture *3D Vision* taught by Dr. Torsten Sattler and Dr. Martin Oswald at *ETH Zurich* in spring term 2018 (Source: http://www.cvg.ethz.ch/teaching/3dvision/ 27. December 2018).

Chapter 2

Conventions

2.1 Rendered Dataset

Figure 2.1: The rendered dataset of an indoor space contains sharp RGB images (left), depth maps (middle) and blurred images (right).

In order to evaluate the proposed algorithm, a rendered dataset of a 3D indoor environment was used[1]. The dataset consists of two different camera (0 and 1) trajectories with 13 timestamped RGB images (serially numbered from 1 to 13) and their corresponding depth and blurred variations respectively (see figure 2.1). The exposure time for each blurred image is set to be constant at 0.04 seconds. Additionally, it includes the intrinsic and extrinsic parameters as well as the timestamped ground-truth poses of the body-fixed frame $\mathcal{B}$ described in the inertial frame $\mathcal{I}$ (see figure 2.2). According to [3], the distance between two consecutive image poses is approximately 26.7 cm (standard deviation of 2.7 cm), whereby the distance traveled during exposure time is approximately 10.6 cm (standard deviation of 1.1 cm).

2.1.1 Coordinate Systems

Four right-handed coordinate systems are used: inertial frame $\mathcal{I}$, body-fixed frame $\mathcal{B}$ and two shifted camera-frames $\mathcal{C}$ (see figure 2.2). The origin of the

[1] Made available by Peidong Liu from the *Computer Vision and Geometry Group*.

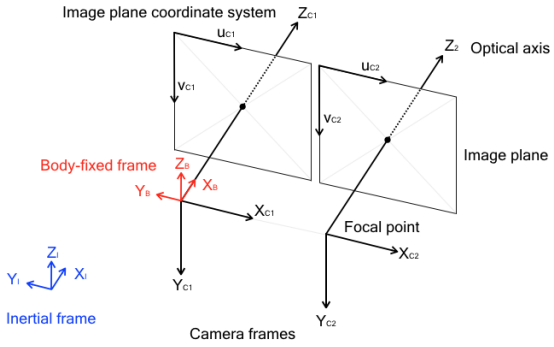

Figure 2.2: Overview of the coordinate systems from in the dataset.

image plane coordinate system is set to be the top left corner.

2.2 Pose Representation

This section elaborates the chosen pose representation and its corresponding mathematical formulations as stated in (Eade, 2017, [4]).

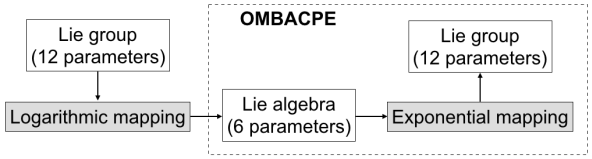

Figure 2.3: The ground-truth poses from the dataset are given in the *Lie group* SE(3)-formulation, which requires 12 parameters for description. A logarithmic mapping transforms this representation to the *Lie algebra* se(3)-formulation with only 6 parameters. In contrast, an exponential mapping does the inverse transformation which is required for the input to an external module (see subsection 3.3.1).

The configuration of a pose is fully described by the position and rotation of the attached body-fixed frame $\mathcal{B}$, w.r.t. an inertial frame $\mathcal{I}$. This is commonly represented in *Lie group* SE(3)-formulation by a homogeneous transformation matrix, which composes of a rotation matrix $R_{\mathcal{IB}}$ and translation vector t_{IB}:

$$T_{\mathcal{IB}} = \left(\begin{array}{c|c} R_{\mathcal{IB}} & t_{IB} \\ \hline 0_{1x3} & 1 \end{array}\right) \in SE(3) \quad R_{\mathcal{IB}} \in SO(3),\, t_{IB} \in \mathbb{R}^3 \tag{2.1}$$

For the design of the optimization process, a pose representation with minimal degrees of freedom is preferred since this limits the number of parame-

ters needed to be estimated during backpropagation (see figure 2.3). Therefore, the pose estimate is defined in *Lie algebra* se(3)-space, which is spanned by six 4 x 4 generator matrices:

$$G_1 = \begin{pmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad G_2 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad G_3 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.2}$$

$$G_4 = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad G_5 = \begin{pmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \quad G_6 = \begin{pmatrix} 0 & -1 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix} \tag{2.3}$$

One element of the se(3)-space can be represented by a linear combination of the translational (equations 2.2) and rotational (equations 2.3) generators:

$$\begin{pmatrix} u_{IB} & \omega_{\mathcal{IB}} \end{pmatrix}^T \in \mathbb{R}^6 \tag{2.4}$$

$$u^1_{IB}G_1 + u^2_{IB}G_2 + u^3_{IB}G_3 + \omega^4_{\mathcal{IB}}G_4 + \omega^5_{\mathcal{IB}}G_5 + \omega^6_{\mathcal{IB}}G_6 \in se(3) \tag{2.5}$$

The matrix exponential on the linear combination (equation 2.5) enables a mapping from se(3)- to SE(3)-formulation which results in the following equations for $R_{\mathcal{IB}}$ and t_{IB}, whereas $[\omega_{\mathcal{IB}}]_x$ denotes the skew symmetric matrix of $\omega_{\mathcal{IB}}$:

$$\theta = \sqrt{\omega_{\mathcal{IB}}{}^T \omega_{\mathcal{IB}}} \tag{2.6}$$

$$V_{\mathcal{IB}} = \mathbb{I} + \frac{1 - \cos(\theta)}{\theta^2}[\omega_{\mathcal{IB}}]_x + \frac{1 - \sin(\theta)/\theta}{\theta^2}[\omega_{\mathcal{IB}}]^2_x \tag{2.7}$$

$$R_{\mathcal{IB}} = \mathbb{I} + \frac{\sin(\theta)}{\theta}[\omega_{\mathcal{IB}}]_x + \frac{1 - \cos(\theta)}{\theta^2}[\omega_{\mathcal{IB}}]^2_x \tag{2.8}$$

$$t_{IB} = V_{\mathcal{IB}} \cdot u_{IB} \tag{2.9}$$

2.3 Pinhole Camera Model Approximation

The mathematical relation between the coordinates of image points and the positions of 3D landmarks are approximated by the pinhole model[2], where the camera aperture is described as a point. The geometric transformation

[2] From the lecture *Vision for Mobile Robots* taught by Prof. Scaramuzza at *ETH Zurich* in fall term 2018 (Source: http://rpg.ifi.uzh.ch/teaching.html 29. December 2018).

between image plane coordinates and camera frame landmarks is given by the following equation:

$$\begin{pmatrix} \lambda u \\ \lambda v \\ \lambda \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_\mathcal{C} \\ Y_\mathcal{C} \\ Z_\mathcal{C} \end{pmatrix} = K \begin{pmatrix} X_\mathcal{C} \\ Y_\mathcal{C} \\ Z_\mathcal{C} \end{pmatrix} \tag{2.10}$$

Here, u and v denote the image coordinates whereas $X_\mathcal{C}$, $Y_\mathcal{C}$ and $Z_\mathcal{C}$ describe the landmark in the camera frame. K is the calibration matrix consisting of focal lengths f_x, f_y and camera optical center c_x, c_y.

In order to transform the landmark from the camera over the body-fixed to the inertial frame, one uses a consecutive homogeneous transformation matrix:

$$\begin{pmatrix} X_\mathcal{I} \\ Y_\mathcal{I} \\ Z_\mathcal{I} \\ 1 \end{pmatrix} = \left(\begin{array}{c|c} R_{\mathcal{IB}} & t_{IB} \\ \hline 0_{1x3} & 1 \end{array}\right) \left(\begin{array}{c|c} R_{\mathcal{BC}} & t_{BC} \\ \hline 0_{1x3} & 1 \end{array}\right) \begin{pmatrix} X_\mathcal{C} \\ Y_\mathcal{C} \\ Z_\mathcal{C} \\ 1 \end{pmatrix} \tag{2.11}$$

The landmark in the inertial frame is denoted by $X_\mathcal{I}$, $Y_\mathcal{I}$ and $Z_\mathcal{I}$.

Chapter 3

Algorithm

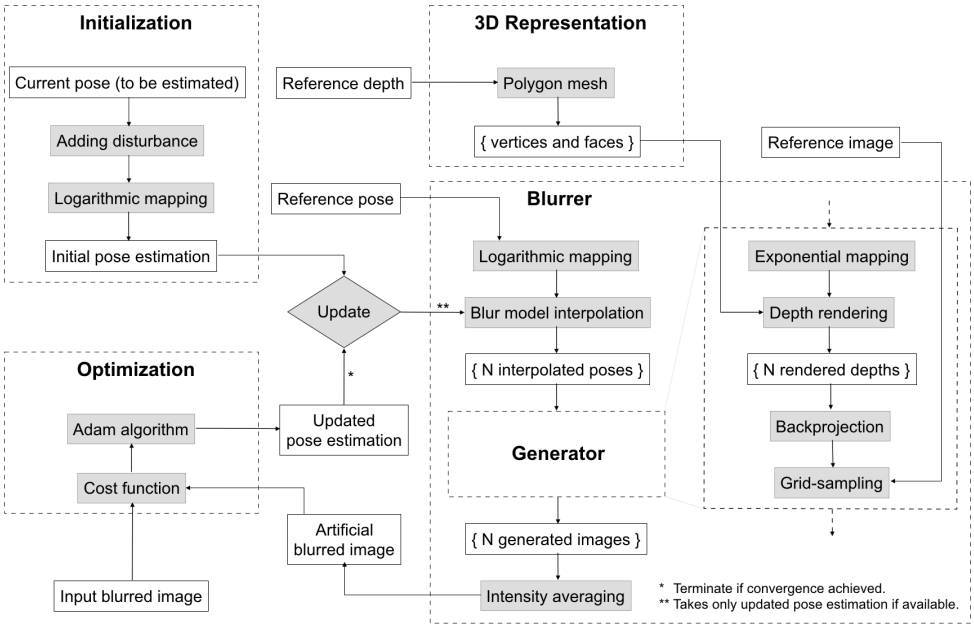

Figure 3.1: Flowchart of the interacting modules.

The algorithm was implemented in *Python 3.6* using the *PyTorch* library and consists of three main part running in iteration. This includes the generator, the blurrer and optimization module. Additionally, there is the initialization and 3D representation part. The former provides a guess for the searched pose and the latter is needed to render depths. The tasks of each module were briefly mentioned in chapter 1.3. This chapter covers the reasoning and implementation. A graphical illustration of the workflow can be found in figure 3.1 which helps understanding the explained concepts.

3.1 Initialization

3.1.1 Adding Disturbance

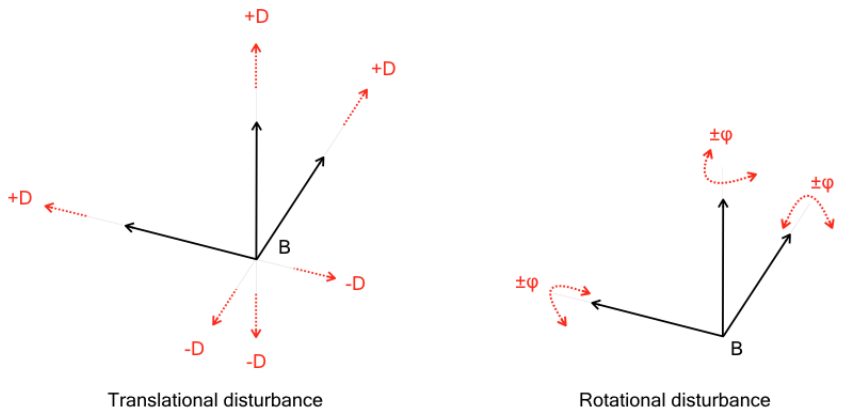

Figure 3.2: Translational and rotational disturbance added to the body-fixed frame.

First, the pipeline is initialized with an educated guess for the blurred input image camera pose. In order to make the process tractable, a disturbance is added to the ground-truth pose of the blurred input image in the inertial-frame. The disturbance is chosen as follows (see figure 3.2):

- for translation, a vector t_d with a desired length and facing in one of the six cardinal directions,
- and for rotation, an unit quaternion q_d corresponding to an elemental rotation with a desired angle.

This provides valuable information on the robustness of the algorithm towards different types of disturbances, which will be discussed in chapter 4.

3.1.2 Logarithmic Mapping

In a subsequent step, the resulting disturbed pose is mapped to *Lie algebra* se(3)-formulation for initialization (see section 2.2). The rotation part from the dataset is given in the unit quaternion form (stored as rotation matrices for simplicity) and mapped to se(3)-formulation by using the methods implemented in the open-source *pyquaternion*[1] library. As for the translation part, the inverse logarithmic mapping $V_{\mathcal{IB}}^{-1}$ is applied (compare with equation 2.9):

[1] *pyquaternion* is a full-featured *Python* module for representing and using quaternions (Source: http://kieranwynn.github.io/pyquaternion/ 15. January 2018).

$$D = \frac{1 - \cos(\theta)}{\theta^2} \tag{3.1}$$

$$V_{\mathcal{IB}}^{-1} = \mathbb{I} - \frac{1}{2}[\omega_{\mathcal{IB}}]_x + \frac{1}{\theta^2}\left(1 - \frac{\sin(\theta)/\theta}{2D}\right)[\omega_{\mathcal{IB}}]_x^2 \tag{3.2}$$

3.2 3D Representation

3.2.1 Formats

There exists multiple formats for representing the camera surroundings. Most commonly used are voxels, pointclouds and polygon-meshes (Kato, Ushiku, Harada, 2018, [5]). Pointclouds are sets of data points in the Euclidean space. Voxels are 3D equivalents of pixels and represents a value on a three-dimensional regular grid. In comparison, polygon-meshes consist of sets of vertices and surfaces (see image 3.3). Since dense depth map rendering at arbitrary poses is required in a subsequent step of the image generation process, polygon-meshes offer an advantage in scalability over voxels or pointclouds. Specifically, in order to represent a triangular surface, a mesh representation only needs to store three vertices and the connections. However, voxels and pointclouds require many sampling points over the face. Because polygon-meshes require less parameters for representing three-dimensional shapes, the size of the representation can be kept small relative to voxels and pointcloud formats. This reduces the storage size for representing large 3D shapes.

3.2.2 Polygon-Mesh Construction

The surfaces of the mesh are generated by connecting each pixel with its cardinal neighbors in a triangular pattern. The dimension of the reference image is 480 x 640 pixels, which results in the total amount of 612'162 faces. This is implemented using the methods from the open-source *meshzoo*[2] library.

The vertices of the polygon-mesh are the projected pixels of the reference image in the inertial frame, which results in the total amount of 307'200 vertices. This is done by applying the inverse from equation 2.10:

$$\begin{pmatrix} X_{\mathcal{C}} \\ Y_{\mathcal{C}} \\ Z_{\mathcal{C}} \end{pmatrix} = \lambda \begin{pmatrix} 1/f_x & 0 & -c_x/f_x \\ 0 & 1/f_y & -c_y/f_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \tag{3.3}$$

[2] *meshzoo* is a collection of meshes for numerical computation (Source: https://pypi.org/project/meshzoo/ 30. December 2018).

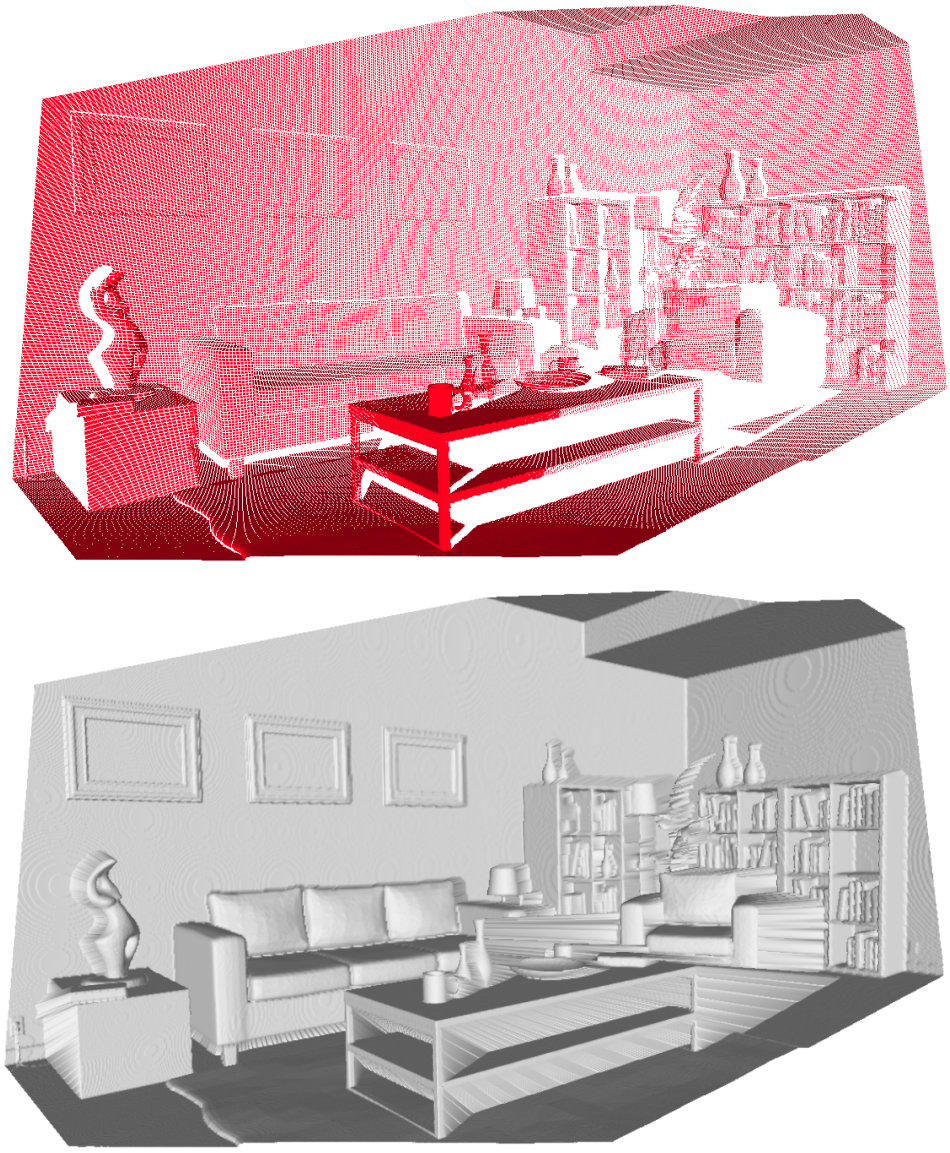

Figure 3.3: The upper image displays only the vertices (307'200 points) of the 3D scene, the lower one includes the corresponding faces (612'162 surfaces). Note that the upper pointcloud leads to a sparse representation of the scene, especially the part which is not visible from the reference image. In comparison, the polygon-mesh on the bottom offers a continuous representation.

Since each value from the dataset reference depth map measures the absolute distance between the optical center and the corresponding point in the camera frame, the inverse equation 3.3 is needed to be adjusted in the following way, where d is the depth value from the dataset:

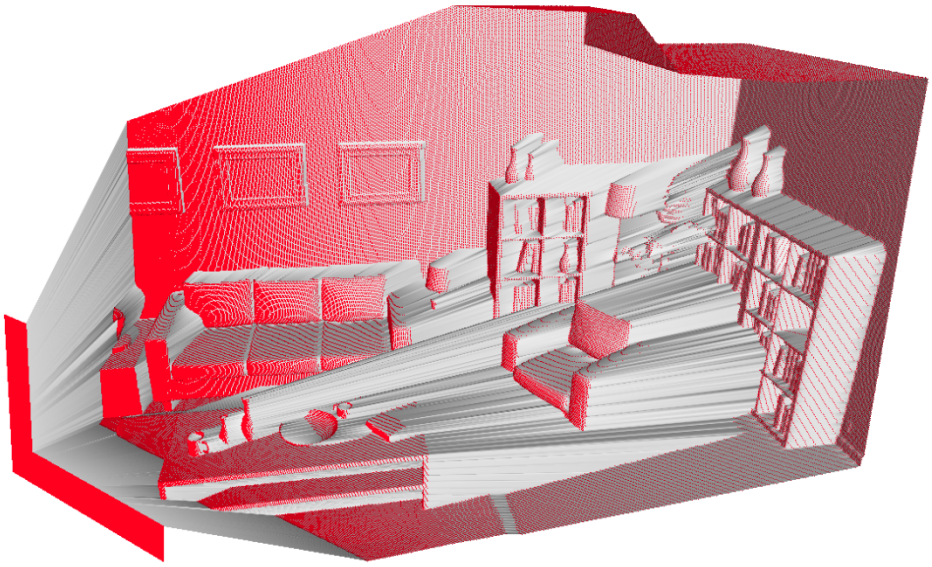

Figure 3.4: Construction of a polygon-mesh by projecting the connected pixels of the image in the 3D space.

$$Z_c = d / \sqrt{(u - c_x)^2 / f_x^2 + (v - c_y)^2 / f_y^2 + 1} \tag{3.4}$$

With the results from equation 3.4, the coordinates X_c and Y_c are calculated by:

$$X_c = Z_c \cdot (u - c_x) / f_x, \quad Y_c = Z_c \cdot (v - c_y) / f_y \tag{3.5}$$

By applying equation 2.11, the landmarks are then transformed in the inertial system for subsequent use.

3.2.3 Downscaling

The construction of the complete polygon-mesh for the 3D representation and its usage for depth map rendering requires a lot of computational resources. In order to reduce the system load, the total amount of vertices, and consequently the total amount of faces, has to be reduced. By subdividing the reference image resolution (height and width), as well as the intrinsic parameters by 2^n, the number of pixels being projected to 3D vertices is lessen (see figure 3.5). This leads to faster computation but simultaneously the overall accuracy of the polygon-mesh and the resulting rendered depth datas are scaled down. Therefore, it results in n to be a hyperparameter which is left for tuning (see section 4.1).

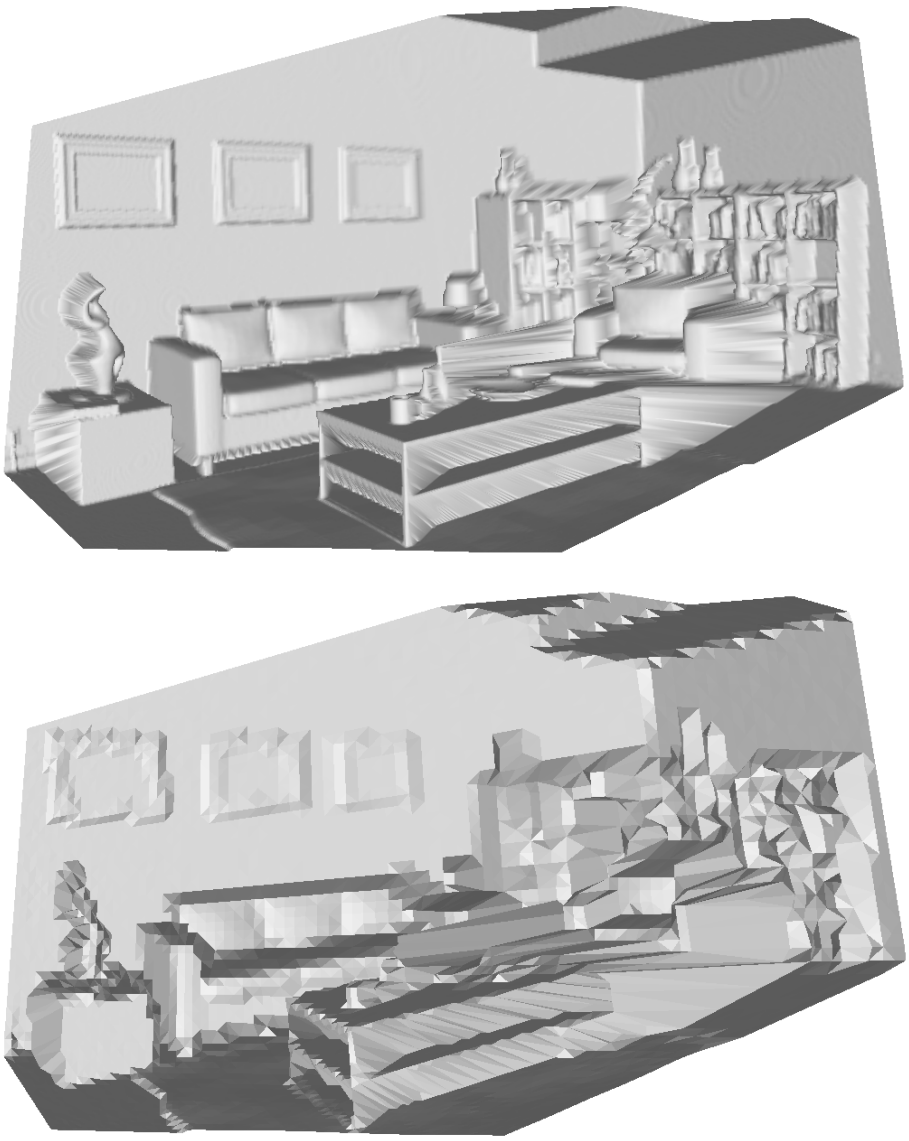

Figure 3.5: The upper mesh is generated by reducing the original mesh by a factor 2^1 (76'800 points, 152'482 surfaces), the lower one by a factor 2^3 (4'800 points, 9'322 surfaces). With increasing exponent, the computational runtime is exponentially shorten at the cost of lower precision of the 3D representation and the rendered depth datas. Therefore, the runtime-precision tradeoff was considered when tuning the exponent.

3.3 Generator

In this section the process of generating sharp images at arbitrary poses based on the reference image is described. This module is then called by the blurrer, which generates a set of sharp images along a specific camera trajectory.

3.3.1 Exponential Mapping

The generator makes use of a depth rendering method which takes the pose

$$\left(\; R_{\mathcal{CI}} \; \middle| \; t_{CI} \; \right) \in SE(3) \tag{3.6}$$

as an input variable. Specifically, it requires the transformation from the inertial $\mathcal{I}$ to the camera $\mathcal{C}$ frame in SE(3)-formulation. Since the input u_{IB} and $\omega_{\mathcal{IB}}$ to the generator module is in se(3)-form, equations 2.6 to 2.9 are used for the exponential mapping. A consecutive transformation with the camera extrinsics results in $R_{\mathcal{IC}}$ and t_{IC} which are subsequently inverted by:

$$R_{\mathcal{CI}} = R_{\mathcal{IC}}^{-1} \tag{3.7}$$

$$t_{CI} = -R_{\mathcal{IC}}^{-1} \cdot t_{IC} \tag{3.8}$$

3.3.2 Depth Rendering

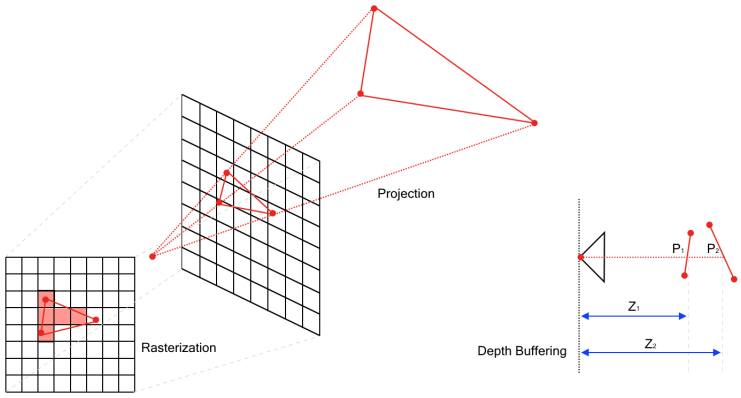

Figure 3.6: Rendering consists of projecting the vertices on to the image plane and applying rasterization to gather the pixel values (left). In order to solve the visibility problem, where multiple surfaces are overlapping each other after projection, a method called smallest depth buffering is considered. Thereby, only the surface with the lowest z-coordinate value is rasterized (right).

The rendering module is an external module pre-implemented in [5][3]. It consists of projecting the vertices of a 3D polygon-mesh (and their corresponding triangular surfaces) onto the image plane and generating an image via a method called rasterization. Former step is solved by applying the perspective equation 2.10, the latter uses regular grid-sampling (not to be mixed-up

[3] The authors proposed a *3D mesh renderer* able to be integrated into neural networks (Source: https://github.com/hiroharu-kato/neural_renderer/ 27. December 2018).

with the process that will be explained in subsection 3.3.3). In particular, if the center of the pixel is within the projected triangle, the pixel takes the color of the overlapping face as displayed in figure 3.6.

Since multiple triangles can be overlapping after projection, the rendering process buffers the z-coordinate corresponding to the distance between the optical center and the nearest face that the projected pixel overlaps[4]. In the case of figure 3.6, as an example, Z_1 is stored in an array the size of the rendered image. Thereby, only the face corresponding to the shortest distance is getting sampled. The resulting rendered depth map is used for following procedures.

3.3.3 Grid-Sampling

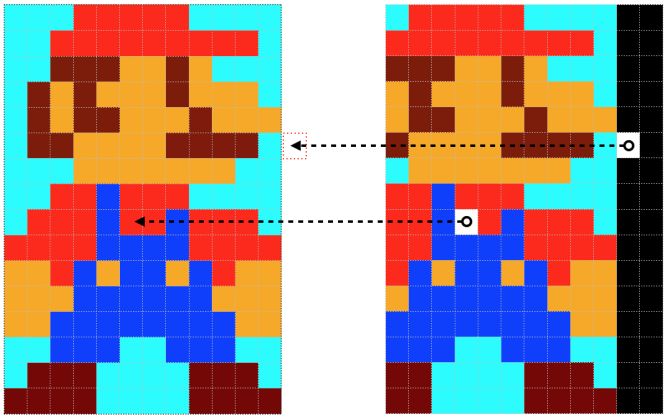

Figure 3.7: This simple example illustrates grid-sampling. The pixel values are sampled from the reference image. If the corresponding pixel in the reference plane is located outside the image boundary, the value is set to zero.

In order to generate an image, the previous rendered depth map is constructed at a desired pose and 3D landmarks are generated via equations 2.10. These landmarks are then backprojected from the current to the reference image plane with the following consecutive homogeneous transformation:

$$T_{\mathcal{RC}} = T_{\mathcal{RI}} \cdot T_{\mathcal{IC}} = \left(\begin{array}{c|c} R_{\mathcal{IR}}^{-1} & -R_{\mathcal{IR}}^{-1} \cdot t_{IR} \\ \hline 0_{1x3} & 1 \end{array}\right) \cdot \left(\begin{array}{c|c} R_{\mathcal{IC}} & t_{IC} \\ \hline 0_{1x3} & 1 \end{array}\right) \tag{3.9}$$

[4] Specifically, the z-coordinate can be found through an inverse interpolation as explained on: http://www.scratchapixel.com/lessons/3d-basic-rendering/rasterization-practical-implementation/ 16. January 2019

Here, $\mathcal{R}$ denotes the camera frame of the reference image. The reprojected pixel coordinates are normalized to $[-1, 1]$ by the image height h and width w with:

$$v_{norm} = 2 \cdot (v_{reproject} - h \cdot 0.5 + 0.5)/(h - 1) \tag{3.10}$$

$$u_{norm} = 2 \cdot (u_{reproject} - w \cdot 0.5 + 0.5)/(w - 1) \tag{3.11}$$

The resulting grid is used to sample the pixel value from the reference image by bilinear interpolation (see figure 3.7). If the grid has values outside the range of $[-1, 1]$, those locations are padded to zero values. This is implemented in the *PyTorch* with the method `grid_sample`[5].

3.4 Blurrer

Figure 3.8: A comparison of a real blurred input image (left) and an artificial generated image (right) according to the simplified blur model. The artificial image is generated with $N_p = 5$ intermediate poses and without downsizing in 3D representation (see section 3.2.3). The black pixels at the upper border is due to zero padding in grid-sampling (see section 3.3.3).

This chapter explains the process of generating an artificial blurred image. The result is the basis for defining the objective function for the optimization part, which will be described in section 3.5.

3.4.1 Blur Model Interpolation

The goal is to approximate the real blurred input image that the camera takes while moving from a reference starting pose P_s to an ending pose P_t. Therefore, multiple sharp images at specific points along the camera trajectory between the two poses are generated using the method described in the previous section. In particular, the N_p poses are linearly estimated as

[5] Source: https://pytorch.org/docs/stable/nn.html 1. Januar 2019

described in the blur model proposed in [2] (see figure 3.9). Given the time instant t_t of the ending pose as well as the exposure time t_{exp}, the timestamps of the N_p intermediate poses are estimated by:

$$\tau_i = t_t - t_{exp} + (i-1) \cdot \frac{t_{exp}}{N_p - 1} \quad i \in \{1, ..., N_p\} \tag{3.12}$$

The corresponding poses at each of those timestamps are linearly interpolated in se(3)-space by:

$$P_i(\tau_i) = \log(P_s) - \frac{\tau_i - t_s}{t_t - t_s} \cdot (\log(P_s) - P_t) \tag{3.13}$$

Here, log denotes the logarithmic mapping (see 3.1.2) of the reference pose.

3.4.2 Intensity Averaging

According to [2], a blurred image is assumed to be an accumulation of sharp images from the opening to the closing of the camera shutter, which is modeled by following equation, where t_s denotes the opening, t_t the closing time and I_τ the sharp image at the interpolated pose:

$$B_t(x, y) = \frac{1}{t_t - t_s} \int_{t_s}^{t_t} I_\tau(x, y) d\tau \tag{3.14}$$

For this thesis, the integral 3.14 is approximated by a finite sum of N_p images within the time interval, where N_p is a hyperparameter for controlling the degree of discretization:

$$B_t(x, y) \approx \frac{1}{N_p} \sum_{i=1}^{N_p} I_{\tau_i}(x, y) \tag{3.15}$$

Here, I_{τ_i} denotes the sharp image sampled at pose P_i.

3.5 Optimization

This section explains the process of finding the pose of the blurred input image via an optimization algorithm. The automatic differentiation package of the *PyTorch* framework is used in order to execute backpropagation and to find the minimal solution to a specific objective cost function.

3.5.1 Cost Function

Based on the assumption, that images share attributes if they are taken at poses close to each other, the L2-norm is considered to measure similarity between the synthetic and the real blurred input image. The objective function is defined as follows:

$$L = \sum_{x,y} (B_i(x,y) - B_t(x,y))^2 \tag{3.16}$$

Here, B_i denotes the blurred input image and B_t the image generated by the blurrer module at pose P_t. Similar to [3], the images are assumed to satisfy Lambertian lighting conditions. This demands that the intensities and colors of a surface to an observer remains unchanged regardless of the angle of view.

3.5.2 Adam Algorithm

In order to minimize the cost function 3.16, gradient descent is used in an iterative process. It updates the pose in the negative direction of the objective function w.r.t. the pose parameters. For this work, a stochastic optimization algorithm called Adam proposed in (Kingma, Ba, 2014, [6]) is used, which is already built in *PyTorch*. In comparison with conventional stochastic gradient descent, it adds momentum to the gradient step which accelerates convergence to the minimum. Through trial and error, the learning parameter is set to 0.01 such that fluctuation in the cost is small during optimization. All other parameters are kept at their *PyTorch* default values. The optimization module terminates once it reaches the termination condition, which is set to:

- 1 cm for translational,
- and 0.02 rad for rotational error.

If convergence can not be achieved, the algorithm terminates after 100 iterations. In this case, the module outputs the closest pose to the searched ground-truth.

Figure 3.9: Process on sampling a blurred image with $N_p = 5$ intermediate images. The depth maps on the left and the corresponding generated image on the right column.

Chapter 4

Results

Multiple experiments were conducted on the developed algorithm described in chapter 3. The results, modifications and their interpretations are presented here. The dataset of camera 0 was used where image 1 was set to be reference. The tests were executed on a 2.5 GHz *Intel i5-7300HQ* machine with *Nvidia GeForce GTX 1050* graphics processor.

4.1 Runtime-Precision Tradeoff

Figure 4.1: The generated image of the rendered indoor space for $n = 0$ (left), $n = 3$ (middle) and $n = 5$ (right). As expected, the quality decreases with increasing scale exponent which is due to the inaccuracy of the rendered depths.

For generating the 3D representation, n is a parameter which defines the resolution of the generated polygon-mesh (see section 3.2.3). Values between 0 and 5 were validated on the basis of sampling an image. An exponent of $n = 0$ lead to a runtime of 86 seconds for the construction of the mesh and 27 seconds for sampling a sharp image. In contrast, $n = 5$ drastically shortens the required computational time (2 seconds for mesh construction and 11 seconds for image sampling). However, this decreases the quality of the generated image (as can be seen from figure 4.1). Therefore, $n = 3$ was chosen for subsequent experiments since it offers the best tradeoff between runtime (4 seconds for mesh construction and 11 seconds for image

sampling) and quality (absent of large distortion).

4.2 Robustness against Translational Disturbance

In the following section, the influence of translational disturbance on the convergence to the current pose is discussed. The current pose was set to the ground-truth pose of image 2, 3 and 4. The experiments were conducted for offset vector sizes between 0 cm and 25 cm in positive cardinal directions (+x, +y and +z) of the body-fixed frame. Further increment of the vector size is not necessary, since the distance between two consecutive image is already around 25 cm and therefore can be replaced by taking the corresponding two non-consecutive image pairs as an initialization. The number of sampled intermediate images for blurring was set to $N_p = 5$.

4.2.1 L2-norm

	Current: 2 (25 cm to 1)				Current: 3 (50 cm to 1)				Current: 4 (75 cm to 1)			
	Error [cm]	Error [rad]	Terminate	Time [s]	Error [cm]	Error [rad]	Terminate	Time [s]	Error [cm]	Error [rad]	Terminate	Time [s]
+ x-axis [cm]												
0	0	0	yes	3.18	0	0	yes	3.17	0	0	yes	3.15
5	0.86	0.0014	yes	13.3	0.70	0.0010	yes	13.10	0.65	0.0027	yes	12.64
10	0.96	0.0015	yes	64.77	0.93	0.0026	yes	51.21	1.46	0.0018	no	162.17
15	0.94	0.0019	yes	85.29	0.90	0.0027	yes	61.31	1.49	0.0016	no	162.66
20	0.83	0.0023	yes	87.75	0.79	0.0020	yes	74.14	0.97	0.0009	yes	76.04
25	0.78	0.0019	yes	95.76	0.93	0.0013	yes	90.25	1.60	0.0024	no	162.92
+ y-axis [cm]												
0	0	0	yes	3.17	0	0	yes	3.19	0	0	yes	3.19
5	0.70	0.0003	yes	37.39	0.84	0.0004	yes	37.83	0.83	0.0065	yes	29.80
10	8.81	0.0161	no	159.62	0.96	0.0012	yes	93.69	0.78	0.0078	yes	43.78
15	8.94	0.0164	no	162.14	0.95	0.0018	yes	103.32	1.41	0.0022	no	161.74
20	8.84	0.0162	no	162.87	0.10	0.0033	yes	107.04	1.38	0.0024	no	161.62
25	8.89	0.0164	no	159.57	0.83	0.0007	yes	143.34	1.40	0.0024	no	161.04
+ z-axis [cm]												
0	0	0	yes	3.19	0	0	yes	3.17	0	0	yes	3.15
5	8.87	0.0155	no	162.06	0.96	0.0006	yes	74.05	1.58	0.0015	no	165.39
10	8.85	0.0155	no	161.44	8.39	0.0109	no	161.94	1.49	0.0013	no	164.33
15	8.84	0.0155	no	162.02	8.28	0.0111	no	170.48	1.45	0.0019	no	160.77
20	8.90	0.0167	no	161.32	8.10	0.0113	no	162.36	2.57	0.0038	no	160.50
25	8.49	0.0167	no	160.91	9.74	0.0099	no	161.90	7.08	0.0090	no	160.86

Table 4.1: The results for translational disturbances with the L2 objective function.

First, the vanilla algorithm described in chapter 3 (compare with table 4.1) was analyzed. The non-marked fields denotes successful experiments, where convergence to the current ground-truth pose is achieved (subcentimeter-accuracy). The blue marked fields are experiments, which indeed do not pass according to the termination conditions mentioned in subsection 3.5.2, but still offers a good signal-to-noise ratio. Consider that the distance from the current to the reference pose is roughly 75 cm, the final error is relatively small (between 1-3 cm translational error). Red marks experiments, which

fails with a large margin (up to 8-9 cm translational error).

As one can see, the algorithm has mainly difficulty dealing with y- and z-axis disturbance, wheras x-axis disturbance can be handled robustly. This is expected, since the positve x-axis corresponds to the optical axis of the camera. Pixel motions for y- and z-axis disturbance are much more distinctive than disturbance in the direction of the principal axis. The likelihood of falling in a minima caused by outliers is higher (see figure 4.2). Therefore, the L2 objective function was replaced with the L1-norm in a subsequent experiment, since it penalizes outliers less and therefore offers more robustness against them.

Figure 4.2: This example shows the difference of the final image between an unsuccessful (left) and a successful (right) experiment. The overall difference is minimal. However, the right image entails more zero padded values (black pixels on the top left corner) which contributes more weight as outliers in the L2-norm. This leads to the fact that an image at the wrong pose but with less empty pixels is preferred.

4.2.2 L1-norm

L1-norm as the objective function combats the issues of minimas due to non-existing pixel values. Since the pixel deviation is not squared, empty pixels are not heavily weighted for the overall cost. Compare to L2-norm, the modified algorithm achieves convergence to the current ground-truth pose for all experiments (see table 4.2). Therefore, for any subsequent tests the L1-norm was used.

The results with the L1-norm are also more accurate than [3]. A comparison with y-axis disturbance shows that their proposed method diverges with increasing distance to the reference image pose. This is expected, since they are relying on the given reference depth map to project 3D landmarks. For sampling an image at a desired pose, they backproject these landmarks on the corresponding image plane. In the process, some landmarks are reprojected outside the image plane or to the same pixel. This results in pixel-

	Current: 2 (25 cm to 1)				Current: 3 (50 cm to 1)				Current: 4 (75 cm to 1)			
	Error [cm]	Error [rad]	Terminate	Time [s]	Error [cm]	Error [rad]	Terminate	Time [s]	Error [cm]	Error [rad]	Terminate	Time [s]
+ x-axis [cm]												
0	0	0	yes	3.19	0	0	yes	3.17	0	0	yes	3.13
5	0.81	0.0028	yes	29.66	0.71	0.0021	yes	12.66	0.98	0.0013	yes	66.28
10	0.94	0.0019	yes	45.37	0.93	0.0018	yes	42.43	0.98	0.0024	yes	78.89
15	0.89	0.0021	yes	53.98	0.92	0.0029	yes	55.17	0.99	0.0026	yes	101.13
20	0.68	0.0016	yes	61.12	0.98	0.0024	yes	69.43	0.98	0.0027	yes	112.21
25	0.88	0.0008	yes	76.35	0.97	0.0031	yes	81.94	0.99	0.0037	yes	121.22
+ y-axis [cm]												
0	0	0	yes	3.17	0	0	yes	3.14	0	0	yes	3.19
5	0.90	0.0008	yes	22.47	0.92	0.0006	yes	23.79	0.83	0.0068	yes	20.41
10	0.98	0.0003	yes	56.26	0.85	0.0030	yes	62.97	0.74	0.0036	yes	32.98
15	0.84	0.0005	yes	113.42	0.84	0.0039	yes	72.38	0.61	0.0059	yes	43.98
20	0.94	0.0001	yes	117.48	0.88	0.0052	yes	78.68	0.59	0.0042	yes	51.79
25	0.96	0.0004	yes	113.74	0.99	0.0055	yes	89.30	0.33	0.0068	yes	59.64
+ z-axis [cm]												
0	0	0	yes	3.16	0	0	yes	3.19	0	0	yes	3.15
5	0.10	0.0012	yes	133.00	0.96	0.0016	yes	46.85	0.91	0.0013	yes	72.44
10	0.93	0.0015	yes	126.95	0.81	0.0004	yes	80.76	0.87	0.0019	yes	80.54
15	0.90	0.0014	yes	145.60	0.83	0.0007	yes	111.50	0.95	0.0028	yes	110.41
20	0.79	0.0010	yes	144.97	0.93	0.0003	yes	129.13	0.90	0.0023	yes	119.69
25	2.17	0.0035	no	163.10	1.16	0.0003	no	160.25	0.85	0.0022	yes	141.79

Table 4.2: The results for translational disturbances with the L1 objective function.

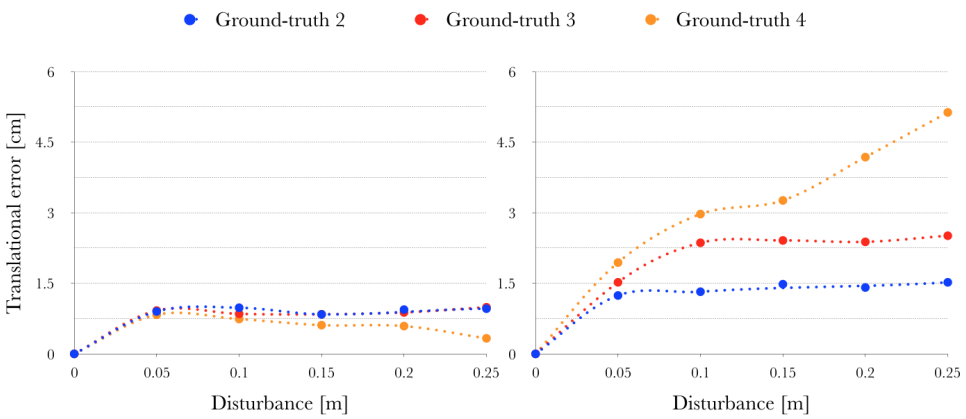

Figure 4.3: A y-axis disturbance comparison with [3] (right). The current pose was set to the ground-truth pose of image 2 (blue), 3 (red) and 4 (orange).

empty images, which increases in sparsity with the distance to the reference pose, resulting in a similar issue as described in figure 4.2.

4.3 Influence of the Number of Intermediate Images

In a further step, the influence of the number of intermediate images generated during open-shutter time was analyzed. It turns out that increasing the number of images can lead to more accurate results. As can be seen in table 4.3, the average translational error over all experiments (only y-axis

	Current: 2 (25 cm to 1)							
	N_p = 5				N_p = 10			
	Error [cm]	Error [rad]	Terminate	Time [s]	Error [cm]	Error [rad]	Terminate	Time [s]
+ y-axis [cm]								
0	0	0	yes	3.17	0	0	yes	3.17
5	0.90	0.0008	yes	22.47	0.78	0.0013	yes	42.33
10	0.98	0.0003	yes	56.26	0.98	0.0008	yes	106.15
15	0.84	0.0005	yes	113.42	0.93	0.0006	yes	230.13
20	0.94	0.0001	yes	117.48	0.85	0.0005	yes	230.87
25	0.96	0.0004	yes	113.74	0.91	0.0005	yes	220.69

Table 4.3: The influence of the number of intermediate images for generating an artificial blurred image. Red marks experiments are the cases, where the accuracy is not decreased.

translational disturbance tested) is decreased. Since generating more images is computational more expensive, the number N_p should be kept at a minimum if the runtime is a priority. In the example, an average decrease of 1 mm in error leads to an increase of computational time by a factor 2.

4.4 Robustness against Angular Disturbance

	Current: 2								
	Error [cm]	Error [rad]	Terminate	Time [s]		Error [cm]	Error [rad]	Terminate	Time [s]
+x-axis [°]					+x-axis [°]				
0	0	0	yes	3.17	30	4.21	0.0068	no	174.70
5	0.80	0.0065	yes	38.41	35	4.10	0.0086	no	168.83
10	0.99	0.0063	yes	56.52	40	5.84	0.0040	no	178.52
15	0.96	0.0030	yes	80.26	45	22.13	0.0286	no	177.42
20	0.87	0.0028	yes	100.32	50	38.36	0.0895	no	195.13
25	0.98	0.0026	yes	111.57	55	106.02	0.1085	no	193.74

Table 4.4: The influence of roll-angle.

Before, only translational disturbance were tested. When it comes to angular disturbance, deviations in camera pitch and yaw behave essentially just like disturbances in y- and z- direction and are therefore covered in experiment 4.2.2. Here, the experiment where the camera experience only a roll motion in positive x-axis was conducted. N_p was set to 5 and the angle disturbance was varied from 0°to 25°. As can be seen in table 4.4, the algorithm shows great robustness against roll-angular disturbance up to 25°. Degrees between 30°and 40°(marked blue) achieves convergence within 6 cm error and degrees from 45°do not converge at all (marked red). One can conclude

that the algorithm is only robust within small angular deviation. Motion blur due to fast angular rotations in x-axis should therefore be avoided.

4.5 Pyramid Resolution Approach

Intuitively, one would expect the computational time to increase with larger deviation. This can be seen in table 4.2, as fixing the current image and increasing the disturbance results in an increase of runtime for each cardinal direction. In contrast, further distance to the reference image does not necessarily lead to higher runtime.

A method to decrease the runtime for large deviated initialization is to follow a coarse-to-sparse approach. The idea is to bring the initial guess as fast as possible close to the ground-truth pose of the current image by starting with an image at a low resolution. Due to lower information density, processing needs much less time and the resulting pose comes closer to the real pose. Subsequently, the next iteration starts at a higher resolution with the last resulting pose as the initialization.

Current: 2 (25 cm to 1)

	Pyramid scale: 2				Pyramid scale: 1				Pyramid scale: 0			
	Error [cm]	Error [rad]	Terminate	Time [s]	Error [cm]	Error [rad]	Terminate	Time [s]	Error [cm]	Error [rad]	Terminate	Time [s]
+ y-axis [cm]												
0	0	0	yes	3.28	0	0	yes	3.27	0	0	yes	3.17
5	0.99	0	yes	14.91	0.84	0.0007	yes	9.37	0.90	0.0008	yes	22.47
10	4.22	0.0053	no	26.22	0.97	0.0006	yes	38.65	0.98	0.0003	yes	56.26
15	4.24	0.0053	no	26.20	0.91	0.0011	yes	45.50	0.84	0.0005	yes	113.42
20	4.17	0.0050	no	26.47	0.99	0.0010	yes	44.84	0.94	0.0001	yes	117.48
25	4.17	0.0048	no	26.45	0.85	0.0010	yes	45.03	0.96	0.0004	yes	113.74

	Pyramid scale: 2				Pyramid scale: 1				Pyramid scale: 0				Results	
	Error [cm]	Error [rad]	Terminate	Time [s]	Error [cm]	Error [rad]	Terminate	Time [s]	Error [cm]	Error [rad]	Terminate	Time [s]	Error [cm]	Time [s]
+ y-axis [cm]														
0	0	0	yes	3.22									0	3.22
5	0.99	0	yes	14.10	0.38	0.0029	yes	8.48	0.25	0.0024	yes	20.30	0.25	42.88
10	4.22	0.0053	no	24.49	1.89	0.0025	no	50.90	0.89	0.0027	yes	21.85	0.89	97.24
15	4.24	0.0053	no	24.54	0.82	0.0008	yes	49.23	0.32	0.0008	yes	21.89	0.32	95.66
20	4.17	0.0050	no	24.47	2.57	0.0034	no	50.94	0.74	0.0010	yes	4.78	0.74	80.19
25	4.17	0.0048	no	24.52	1.12	0.0013	no	50.77	0.93	0.0009	yes	3.14	0.93	78.43

Table 4.5: The top experiment shows the results of different pyramid scales individually ($N_p = 5$). For a scale exponent of 0, the results are the same as in table 4.2. A scale of 1 leads to similar accurate results but with shorter runtime. At a scale of 2, the termination conditions are not achieved for the blue marked cases. The lower experiment follows the pyramid approach starting with scale 2. The termination conditions are halved if the previous lower scale is terminated. As can be seen from the results column, compared with the results from table 4.2, the new approach achieves higher precision and faster runtime performance.

The first test shows that lowering the resolution by 2^n can lead to shorter runtime (see top table 4.5). However, at high exponent the information density is not sufficient for termination. The second test (see lower table 4.5) displays the results of the coarse-to-sparse method. As can be seen, this approach leads to higher precision at shorter runtime.

4.6 Disturbed Reference Depth Map

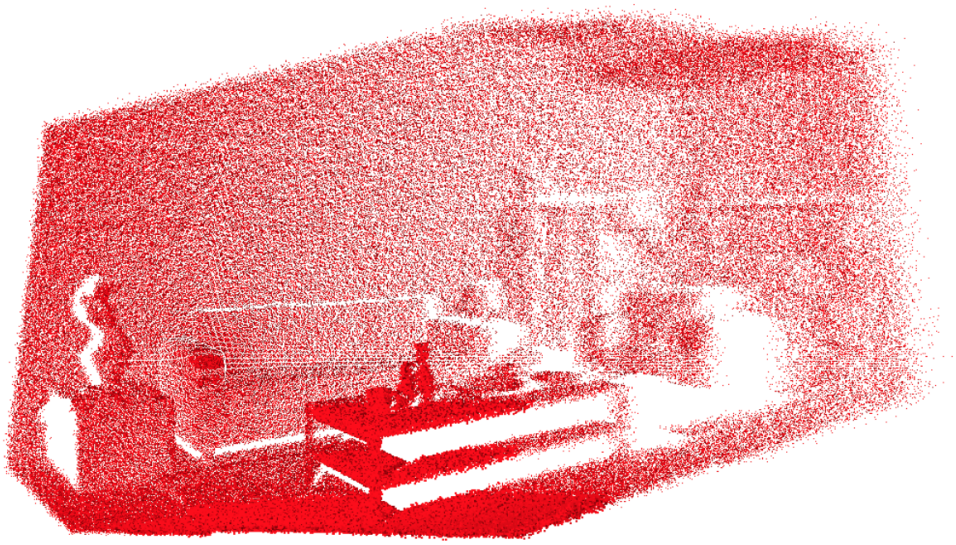

Figure 4.4: Polygon-mesh ($n = 0$) constructed with a perturbed reference depth map with $\sigma = 0.01$ 1/m. At comparison with figure 3.3, the difference of perturbation becomes obvious.

	Current: 2							
	Error [cm]	Error [rad]	Terminate	Time [s]		Error [cm]	Error [rad]	Terminate	Time [s]
Std σ [1/m]					Std σ [1/m]				
0	0.80	0.0006	yes	29.29	0.12	35.66	0.1136	no	175.02
0.02	0.68	0.0004	yes	29.60	0.14	22.34	0.0388	no	197.23
0.04	0.82	0.0011	yes	29.64	0.16	13.58	0.0735	no	158.20
0.06	0.97	0.0008	yes	21.28	0.18	30.91	0.0582	no	160.03
0.08	0.85	0.0051	yes	22.96	0.20	9.78	0.0002	no	160.53
0.1	0.87	0.0061	yes	19.75	0.22	54.11	0.2145	no	161.35

Table 4.6: The influence of a perturbed reference depth map.

For generating the 3D representation, one assumes perfect depth maps, which are not always given in reality. As a further test, reference depth values were inversely perturbed by a normal distribution with zero mean according to:

$$d_p = \left(\frac{1}{d} + \mathcal{N}(0, \sigma)\right)^{-1} \tag{4.1}$$

Here, d_p denotes the disturbed depth value. For the test, the x-axis translational disturbance was set to 0.15 m and the x-axis roll disturbance to 5

degree. As can be seen from table 4.6, the algorithm shows large deviation from the current ground-truth pose starting at variance 0.12 m^{-1}. This is also mirrored in the sampled images (see figure 4.5) as the quality goes down with larger depth perturbation.

Figure 4.5: The generated image at the current pose 2 for 0 m^{-1} (left), 0.02 m^{-1} (middle) and 0.12 m^{-1} (right) standard deviation depth perturbation. As expected, the generated image quality decreases with an increase of inaccuracy in the reference depth map.

Chapter 5

Conclusion

The goal of this semester thesis is to implement an optimization-based motion blur aware camera pose tracker within the *Pytorch* framework.

5.1 Achievements

The proposed algorithm is able to estimate the pose for a motion-blurred input image given an educated guess, a sharp reference image with corresponding ground-truth depth map and pose and the camera information, which includes the intrinsic and extrinsic parameters as well as the exposure time.

A downscaling of the 3D representation by a scale-exponent 3 offers an optimal tradeoff between computational runtime and depth rendering quality. As for the objective function, it turns out that the L1-norm is a better option for similarity comparison than the L2-norm due to its robustness against outliers. The resulting modified algorithm achieves subcentimeter-accuracy for large range of translational disturbance and small range of roll-angle disturbance. By following a coarse-to-sparse approach, one can achieve higher precision and faster runtime performance. Using more intermediate poses for generating artificial blurred images result in slightly more accurate results, but also takes longer time to compute. In particular, an average decrease of 1 mm in translational error leads to an increase of computational time by a factor 2. Since perfect ground-truth reference depth maps are rare in practical implementation, the given reference depth map was perturbed by a normal distribution with zero mean. It turns out that the algorithm is very robust up to a certain variance of perturbation.

Therefore, the proposed algorithm provides a fail-safe solution to conventional VO or SLAM techniques.

5.2 Outlook

5.2.1 Preprocessing via Convolutional Neural Network

As mentioned, the definition of the cost function in this thesis assumes Lambertian lighting conditions. With the given rendered dataset, this assumption created no issues. However, at occurrence of reflective or glossy surfaces, an alternative approach must be considered. One option is to preprocess the images via a convolutional neural network (CNN). CNN is a class of neural networks specifically designed to detect visual patterns directly from images. As a possibility, a 16 layered VGGNet (Simonyan, Zisserman, 2014, [7]) architecture with 3x3 convolutional and 2x2 pooling layers can be used to extract image features for similarity comparison with the L1-norm.

5.2.2 Selective Polygon-Mesh Surface Reduction

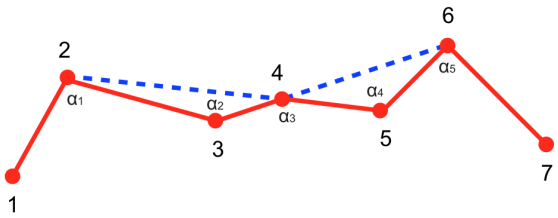

Figure 5.1: With a selective surface reduction approach, the goal is to preserve successive surfaces with intersection angles below a certain threshold which corresponds to surfaces around edges. In the example above, surfaces 1-2, 2-3, 5-6, and 6-7 are preserved whereas 2-3, 3-4, 4-5 and 5-6 are replaced with connective surfaces.

Reducing the amount of faces of the polygon-mesh by downsizing the image resolution is not the most optimal solution. The main issue is the overall loss of details in the 3D representation. Following a selective surface reduction (see figure 5.1) might be a better option.

5.2.3 Different Dataset

Different dataset should be used in order to test the generalization and performance of the algorithm on them. However, the selection of available datasets with blurred images and corresponding depth maps and ground-truth camera poses are limited.

Appendix A

Appendix

A.1 Repository

This *GitLab* repository contains the *Python* code for the optimization-based motion blur aware camera pose tracker:

- https://gitlab.com/jdiep/semester-thesis.git/

A.1.1 Installation

This installation was tested for *Ubuntu 16.04 LTS*. For other operating systems, changes or additional packages might be required. The following packages were used:

- *Python 3.6.5*: https://www.python.org/downloads/source/
- *CUDA Toolkit 9.0*: https://developer.nvidia.com/cuda-90-download-archive/
- *PyTorch*: https://pytorch.org/
- *Neural 3D Mesh Renderer*[1]: http://hiroharu-kato.com/projects_en/neural_renderer.html
- *pyquaternion*: http://kieranwynn.github.io/pyquaternion/
- *meshzoo 0.4.3*: https://pypi.org/project/meshzoo/
- *OpenCV 3.4.4*: https://www.learnopencv.com/install-opencv-3-4-4-on-ubuntu-16-04/
- *tqdm*: https://github.com/tqdm/tqdm

[1] A modified version was used for this project which is not publicly available at the time of writing (1. Feburary 2019).

- *scikit-image 0.13.1*: http://scikit-image.org/docs/0.13.x/install.html
- *matplotlib 3.0.2* (optional): https://matplotlib.org/users/installing.html
- *MeshLab 2016* (optional): http://www.meshlab.net/#download/

A.1.2 Code Structure

The `source_code` folder contains all the essential classes with its corresponding methods:

- `dataset.py`: This class is responsible for reading out the information from the rendered dataset. Additionally, it also contains the methods to return the scaled as well as the perturbed information.
- `framework.py`: This class sets up the framework for the optimization process. It initializes the pose with a disturbance which is subsequently mapped to se(3)-formulation. Further, the objective function is defined here.
- `imagegeneration.py`: This class contains methods to generate rendered depth maps, sharp and blurred images at arbitrary poses.
- `meshgeneration.py`: This class contains the construction of the 3D polygon-mesh.
- `optimization.py`: The optimization with *PyTorch* automatic differentiation package is contained here.
- `posetransformation.py`: Contains the exponential and logarithmic mapping methods.
- `randomizer.py`: Responsible for creating a cardinal directed vector with a defined length as well as an angle axis vector corresponding to an elemental rotation with a defined angle.
- `renderer.py`: Initializes the external renderer module, which is used for generating depth maps.

The `main` folder contains the executables of the classes mentioned above which can be run individually depending on the desired output:

- `pose_estimation.py`: This is the main executable file of this project. It finds the pose of the blurred input image via an optimization process.
- `3d_representation.py`: This program produces a 3D pointcloud and polygon-mesh representation of the environment in the intial frame. The generated txt- and obj-file can be observed in *Meshlab*.

- `image_generator.py`: This program generates sharp and blurry images at arbitrary poses.

A.1.3 Docker

In order to run the code of [3], a *Dockerfile*[2] was written to build an image with all corresponding libraries and dependencies.

[2]Docker is a computer program that performs operating-system-level virtualization (Source: https://www.docker.com/ 1. Feburary 2019).

Bibliography

[1] B. D. Lucas, T. Kanade, *et al.*, "An iterative image registration technique with an application to stereo vision," 1981.

[2] H. Park and K. Mu Lee, "Joint estimation of camera pose, depth, deblurring, and super-resolution from a blurred image sequence," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 4613–4621, 2017.

[3] T. Aebi, F. Milano, and C. Schnetzler, "Motion blur aware camera pose tracking," 2018. 3D Vision report, ETH Zurich.

[4] E. Eade, "Lie groups for 2d and 3d transformations," 2017. Available at http://ethaneade.com/lie.pdf; accessed 14-January-2018.

[5] H. Kato, Y. Ushiku, and T. Harada, "Neural 3d mesh renderer," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3907–3916, 2018.

[6] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[7] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.