UNIVERSITÄT BONN

Rheinische
Friedrich-Wilhelms-
Universität Bonn

AIS

Institute for Computer Science
Department VI
Autonomous Intelligent Systems

Rheinische
Friedrich-Wilhelms-Universität Bonn

Bachelor Thesis

Improve the Deeper Inverse Compositional Algorithms for RGB-D cameras and LiDAR

Author:
Bruno Scheider

First Examiner:
Prof. Dr. Sven Behnke

Second Examiner:
Dr. Eduard Zell

Date: Aug 16, 2021

Declaration

I hereby declare that I am the sole author of this thesis and that none other than the specified sources and aids have been used. Passages and figures quoted from other works have been marked with appropriate mention of the source.

Bonn, 16.08.2021 [illegible]

Place, Date Signature

Abstract

Image alignment is one of the core components for most vision-based systems to estimate the motion between consecutive images and to establish correspondences with subpixel accuracy. The gold-standard alignment method is the inverse compositional algorithm based on the original Lucas-Kanade algorithm. It has many applications such as stereo reconstruction, tracking, image registration and Simultaneous Localization and Mapping (SLAM).

The underlying non-linearity of the problem and the existence of outliers make it difficult to optimize and decide on how to weigh residuals as well as to find good regularization parameters. Recently, Lv et al. improved the classical inverse compositional approach through learning features, weights and damping factors achieving good results on RGB-D data. Photometric-consistency is still a central assumption within the algorithm and has proven to be less robust than e.g. gradient-based similarity metrics. In this thesis, different error functions will be tested within the framework. In our experiments, we will test the new introduced error functions against the photometric error. In the next step, we will adapt the framework to handle LiDAR data. We will compare the performance of the network with a standard aligning method for point clouds, the Generalized Iterative Closest Point algorithm.

Contents

1 Introduction

When data is captured with a moving camera, the following frames can be used to estimate the transformation of the two camera poses. Obtaining this transformation out of two consecutive images is known as image alignment. The standard alignment algorithm is the Inverse Compositional (IC) [2] algorithm based on the Lucas-Kanade algorithm from 1980. IC algorithm uses iterative optimization of the transformation. It can be robustified by introducing a weight matrix in the optimization step. Finding the right weight matrix requires domain knowledge, which differs with each use case. By additionally introducing a damping term we get the Levernberg-Marquardt update step. Once again the damping term can not be chosen universally. Lv et al. [19] proposed a learned variant of the IC algorithm, which lifts these constraints due to learned features.
In the update step of image alignment algorithms, we try to minimize a residual between two images by finding the transformation between the two images. A simple and effective approach to obtain the residuals is computing the difference between the two images, which is called photometric error. In the first part of this thesis, we evaluate the framework by Lv et al. [19] with different methods to compute the residuals. We will use Cosine Similarity, the Structural Similarity Index, the Gradient Magnitude, and two metrics proposed by Quenzel et al. [23]. These methods tend to be more robust against distortions than the difference, for e.g. against lighting changes. The idea is that the new metrics can lead to a better representation of critical points in the image. With a better representation of the errors, the network could learn more efficiently.
In our experiments, we evaluate the different metrics on the TUM RGB-D [32] dataset, the MovingObjects3D [5] dataset, and the ICL Pering [27] dataset. The real-world dataset TUM RGB-D is often used for the evaluation of different algorithms which gives good comparability with other algorithms. MovingObjects3D and ICL Pering are both synthetic datasets, which give reliable depth estimation. We use the End-Point-Error for training and the Relative-Pose-Error to evaluate the framework.
Other than cameras, LiDAR sensors are common sensors to percept the environment. Aligning point clouds is a key component in mapping and localization tasks. In the second part, we propose a framework to obtain the transformation between

two point clouds. It is based on the Deep Inverse Compositional algorithm [19]. We will use LiDAR range images (LRI) as input for the Two-View feature encoder introduced by [19] to use well-established operations like convolution. We will compare the proposed framework to align LiDAR data against the generalized iterative closest point (GICP) algorithm on the KITTI [11] dataset, and a dataset captured in the DRZ Living Lab [28]. We use the End-Point-Error for training and the Relative-Pose-Error to evaluate the framework.

2 Fundamentals

2.1 Pinhole Camera Model

Cameras are one of the most used sensors for environment perception. Based on this sensor a widely spread variations of algorithms are developed. There are different methods for modeling a camera. Here, we will focus on the pinhole camera model.

The goal of the model is to describe how 3D world objects are projected onto a 2D image plane and vice versa. A barrier with a small hole, the pinhole, is placed between the object and the image. If there would be no barrier, all of the light rays of the object will influence every point on the image. As one can see in Figure 2.1,

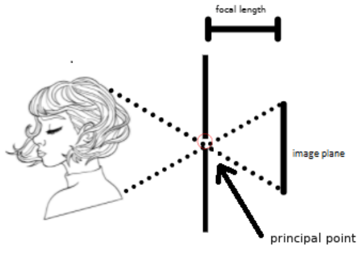

Figure 2.1: Pinhole camera model

the barrier with the pinhole establishes a one-to-one mapping between the object and the image.

The location of the pinhole is referred to as the principal point and is given by x- and y-coordinates in the image plane. The distance of the pinhole to the image is referred focal length. The principal point $(c_x, c_y)^\intercal$ and the focal length (f_x, f_y) can be used to project a 3D point onto a 2D plane. The matrix used for this is named the camera matrix $K \in \mathbb{R}^{3\times3}$:

$$K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}. \tag{2.1}$$

Let $\mathbf{p} = (x, y, z)^\intercal \in \mathbb{R}^3$ be a point in world coordinates. We project the point $\mathbf{p}$ onto the image $(u, v)^\intercal \in \Omega \subset \mathbb{R}^2$ with the camera matrix K:

$$\begin{pmatrix} u' \\ v' \\ w' \end{pmatrix} = K \cdot \mathbf{p}, \tag{2.2}$$

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \frac{u'}{w'} \\ \frac{v'}{w'} \end{pmatrix}. \tag{2.3}$$

This leads to the projection function $f_{proj} : \mathbb{R}^3 \mapsto \mathbb{R}^2$:

$$\Rightarrow f_{proj}(p) = \begin{pmatrix} f_x \frac{x}{z} + c_x \\ f_y \frac{y}{z} + c_y \end{pmatrix}. \tag{2.4}$$

For many applications an image needs to be backprojected into a 3D pointcloud. To achieve this the inverse camera matrix

$$K^{-1} = \begin{pmatrix} \frac{1}{f_x} & 0 & \frac{-c_x}{f_x} \\ 0 & \frac{1}{f_y} & \frac{-c_y}{f_y} \\ 0 & 0 & 1 \end{pmatrix} \tag{2.5}$$

is applied on an image point $\mathbf{x} = (u, v, 1)^\intercal$.

2.1.1 Jacobians of the Pinhole Model

Many computer vision applications require the Jacobians of the projection function 2.4. It is obtained by partial derivation w.r.t. each coordinate:

$$\frac{\partial f_{proj}}{\partial \mathbf{p}} = J_{proj} = \begin{pmatrix} f_x \\ f_y \end{pmatrix} \cdot \begin{pmatrix} \frac{1}{z} & 0 & -\frac{x}{z^2} \\ 0 & \frac{1}{z} & -\frac{y}{z^2} \end{pmatrix}. \tag{2.6}$$

2.2 LiDAR Range Images

Autonomous robots often use LiDARs to perceive distances to objects in their vicinity. These measured distances are more accurate than stereo depth estimates and can be converted to 3D point clouds. This section will explain how a 3D scan of the environments works and how to create a 2D LiDAR range image (LRI).
A LiDAR sends out a focused light pulse and measures the time between sending and receiving the reflected beam. The measured time is used to compute the distance. The distance and the direction of the laser pulse allow to compute the x,

y, z coordinates of the reflecting surface relative to the sensor.
To cover more space with one measurement, multiple lasers are used at once. Often these are ordered vertically. All of them are tilted so that they cover a wide range on the vertical axis. Rotating the sensor around its vertical axis allows to generate a dense point cloud of the surrounding environment. This leads to measurements in all directions. This can be thought of as placing a LiDAR sensor in the middle of a unit sphere, where each laser would create a measurement line on different heights.
Using an LRI instead of a point cloud is a good choice to process LiDAR scans with neural networks since they contain all necessary information and allow to use well-established operations for images like convolutions. The lines of each vertical laser are mapped to the rows of the LRI. A typical number of vertical lasers would be 64. While rotating, each measurement is mapped to a column of the image. A typical number of taken measurements is 1024. This would lead to a 1024x64 pixel image. The frequency of the laser scans results in the resolution on the x-axis. The number of vertical lasers is the resolution on the y-axis.
The spherical coordinates are rather practical for dealing with LRIs, because other than their equivalent Cartesian representations the first two entrys are not depending on the depth of the measurement. One can use the first two entrys to obtain the pixel location within an LRI. Let $\mathbf{p}$ be a point in Cartesian coodinates (x, y, z), to convert $\mathbf{p}$ in its spherical equivalent (θ, φ, r) following formula can be used:

$$\begin{pmatrix} \theta \\ \varphi \\ r \end{pmatrix} = \begin{pmatrix} \arctan(\frac{y}{x}) \\ \arccos(\frac{z}{\|\mathbf{p}\|}) \\ \|\mathbf{p}\| \end{pmatrix} . \tag{2.7}$$

Similar to the pinhole camera matrix, we define a spherical projection matrix:

$$K = \begin{pmatrix} -f_x & 0 & c_x \\ 0 & -f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} , \tag{2.8}$$

with f_x as the inverse of the horizontal resolution of the laser scan and f_y as the inverse of the vertical resolution of the LiDAR scan. The principal point $(c_x, c_y)^\intercal$ is defined as $\left(\frac{(a \cdot w)}{2}, \frac{h}{2}\right)^\intercal$, with w as the width and h as the height of the image. a is the area of the scanner where the tilting angle is positive. c_x is set to be the center of the range image, and c_y is set to match the $0°$ horizontal angle.
The location on the image can now be computed with $K \cdot \mathbf{p}$, with $\mathbf{p}$ as a point in

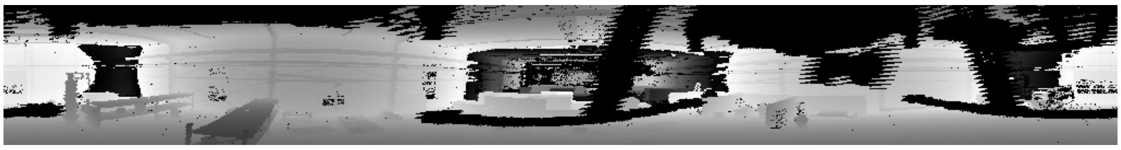

Figure 2.2: Example of an LRI

spherical coordinates:

$$K \cdot \mathbf{p} = \begin{pmatrix} w - \theta \cdot f_x \\ h - \varphi \cdot f_y \\ 1 \end{pmatrix}. \tag{2.9}$$

This would lead to the projection function $f_{proj} : \mathbb{R}^3 \mapsto \mathbb{R}^2$:

$$f_{proj}(\mathbf{p}) = \begin{pmatrix} w - \theta \cdot f_x \\ h - \varphi \cdot f_y \end{pmatrix}. \tag{2.10}$$

While the spherical coordinates are useful for understanding how the pixel locations are determined, Cartesian coordinates are the typical choice to represent coordinates. Figure 2.2 gives an example of an LRI. The pointcloud is captured with a quadrocopter. The black pixel are invalid pixel.

2.2.1 Jacobians LRI

For our application we need the Jacobians of Equation (2.10). To obtain those each part of the projection function needs to be derived w.r.t. x, y and z. We substitude θ and φ in Equation (2.10) following Equation (2.7):

$$f_{proj}(\mathbf{p}) = \begin{pmatrix} w - \arctan(\frac{y}{x}) \cdot f_x \\ h - \arccos(\frac{z}{\|\mathbf{p}\|}) \cdot f_y \end{pmatrix}. \tag{2.11}$$

The Jacobian Matrix w.r.t. $(x, y, z) = \mathbf{p} \in \mathbb{R}^3$ is:

$$\frac{\partial f_{proj}}{\partial \mathbf{p}} = \begin{pmatrix} fx \\ fy \end{pmatrix} \cdot \begin{pmatrix} -\frac{y}{x^2+y^2} & \frac{x}{x^2+y^2} & 0 \\ -\frac{xz}{(x^2+y^2)r^2} & -\frac{yz}{(x^2+y^2)r^2} & \frac{\sqrt{x^2+y^2}}{r^2} \end{pmatrix}, \tag{2.12}$$

with r as norm of $\mathbf{p}$.

2.3 Transformation in 3D

In this thesis we focus on transformations that include arbitrary rotations and translations. We can transform a point $\mathbf{p} \in \mathbb{R}^3$ with the rotation Matrix $\mathbf{R} \in \mathbb{R}^{3\times3}$ and a translation vector $\mathbf{t} \in \mathbb{R}^3$:

$$\tilde{\mathbf{p}} = \mathbf{R}\mathbf{p} + \mathbf{t}. \tag{2.13}$$

$\mathbf{R}$ is a rotation matrix, therefore it is an *orthogonal* matrix ($\mathbf{R}\mathbf{R}^\intercal = \mathbf{R}^\intercal\mathbf{R} = \mathbf{I}$) with $det(\mathbf{R}) = +1$. Matrices with these properties build the group of *special orthogonal* trasformations, known as $\mathbf{SO}(3)$. This group is closed under multiplication.
A transformation with a rotation and a translation can be represented in a tranformation matrix $\mathbf{T} \in \mathbb{R}^{4\times4}$. To apply a transformation matrix on a point $\mathbf{p}$, it needs to be extended by one in the fourth dimension. This homogeneous coordinates will be marked with $[\mathbf{p}]$.

$$[\tilde{\mathbf{p}}] = \mathbf{T} \cdot [\mathbf{p}] = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{pmatrix} \cdot [\mathbf{p}] \tag{2.14}$$

Rotation and translation together form the *special euclidean* group $\mathbf{SE}(3)$. This is the group of affine rigid motions.
$\mathbf{SE}(3)$ is a Lie group. Therefore it has an associated Lie algebra $\mathfrak{se}(3)$. Transformations in $\mathbf{SE}(3)$ have six degrees of freedom (DoF), three for the rotation and three for the translation. An arbitrary element $v \in \mathfrak{se}(3)$ can be represented as a vector in $\mathbb{R}^6$. The log- and exp-map convert between SE(3) and se(3) where the Lie algebra acts as the tangent space for optimization. A detailed introduction to Lie groups and its associated Lie algebra is given in [3].

2.3.1 Jacobians of $\mathbf{SE}(3)$

The Jacobians can be divided into two parts. The first 3x3 matrix is for the rotational part, the second for the translation:

$$J_{warp} = \begin{pmatrix} 0 & z & -y & 1 & 0 & 0 \\ -z & 0 & x & 0 & 1 & 0 \\ y & -x & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{2.15}$$

2.4 Convolutional Neural Networks

2.4.1 Convolution

Artificial intelligence improved significantly in recent years with the convolutional neural network (CNN) paradigm. Inspired by the human visual cortex, CNNs are organized in layers. Each layer is connected to the next by a convolutional kernel. To process data through the network, the convolution operator $* : \mathbb{R}^n \mapsto \mathbb{R}^m$ is used.

$$(I * K)(u, v) = \sum_{x=0}^{X} \sum_{y=0}^{Y} I(u - x, v - y) K(x, y), \tag{2.16}$$

with $K \in \mathbb{R}^{x \times y}$ as kernel and $I \in \Omega \subset \mathbb{R}^2$ as an image. Similar to the human visual system, the kernel values are shared in each layer and are adjustable. The adjustment of the kernel values is known as training.
The kernel is applied locally. Thus, the first layer of a CNN obtains low-level features, such as edges or corners. Later layers create more abstract feature representations.

2.4.2 Fully Connected Layer

Often the convolutional layers are followed by a fully connected layer. As one can see in Figure 2.3, in a fully connected layer, each neuron is connected to each neuron in the next layer. Convolutional layers only use local information. In a fully connected layer, all information is available to every neuron. The convolutional part of such an architecture can be viewed as feature extraction. The fully connected layer interprets the feature and gives a result.

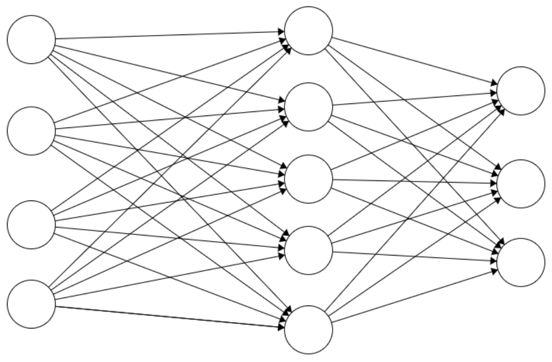

Figure 2.3: Fully connected layer [10]

2.4.3 ReLU

One reason for the success of neural networks is non-linearity. For many years the most popular non-linearity functions were tanh or sigmoid. Recently the Rectified-Linear-Unit is favored because it is much simpler and does not have problems with vanishing gradients in back propagation [1].

$$ReLU(x) = \max(0, x) \tag{2.17}$$

ReLU layer often follow a convolutional layer.

2.4.4 Dilation

The idea behind dilated convolutions $*_l$ is to deal with the local and the global context simultaneously. We achieve this by applying a filter with a greater size on the input data. In addition, this filter is not considering every data point but every l^{th}. The number of parameters does not increase, but the size of the receptive field does increase.

$$(I *_l K)(u, v) = \sum_{x=0}^{X} \sum_{y=0}^{Y} I(u - lx, v - ly) K(x, y) \tag{2.18}$$

For a more detailed explaination we refer to [36].

2.4.5 Batch Normalization

Normalization of the input data is a common way to speed up the learning of a network or handle outlier. Batch normalization is a technique to normalize data between layers of a Neural Network instead of normalizing only the input data. The goal is to lead the data to a mean of zero and a standard deviation of one over one mini batch B with size m:

$$\mu_B = \frac{1}{m} \sum_{i=1}^{m} x_i, \tag{2.19}$$

$$\sigma_B^2 = \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2, \tag{2.20}$$

$$\tilde{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2}}. \tag{2.21}$$

$\tilde{x}_i$ is the normalized neuron value. For a more detailed explaination we refer to [14].

3 Related Work

3.1 Dense Image Alignment

Many different algorithms were developed for image alignment. Kerl et al. [16] use a coarse-to-fine approach to minimize the photometric error to align two images. Sturmberg et al. [30] published a framework that is invariant to lighting changes. Their GN-Net uses a Gauss-Newton algorithm for optimization. Instead of Gauss-Newton, Levenberg-Marquardt can be used to improve the robustness against bad initialization [31]. Another approach for dense image alignment was proposed in 1981, the original Lucas-Kanade (LK) algorithm. Different variations of the LK algorithm were developed including the compositional and the inverse compositional (IC) algorithm [2].

3.1.1 Lucas-Kanade Algorithm

The LK algorithm aims to align an input image $I(\mathbf{x})$ to a template image $T(\mathbf{x})$, where $\mathbf{x} = (u, v)^\intercal$ is a pixel coordinate. To achieve this the LK algorithm tries to find a transformation $W(I, \xi)$ on the input image $I(\mathbf{x})$ to align the input and the template image [2]. ξ is a set of warp parameters and $I(\xi)$ is short for the warped input image using warping function $W(I, \xi)$.
For example let $W(I, \xi)$ be a 2-D translation, with $\xi = (p_1, p_2)$ then:

$$W(I, \xi) = I(\xi) = I(\tilde{\mathbf{x}}), \tag{3.1}$$

with $\tilde{\mathbf{x}} = (u + p_1, v + p_2)$, is the corresponding warped input image.
The goal of the LK algorithm is to minimize the objective function:

$$\min_{\xi} \|I(\xi) - T(0)\|^2. \tag{3.2}$$

T(0) is the template image with no warping applied. Instead of trying to solve this directly the LK algorithm solves

$$\min_{\Delta\xi} \|I(\xi_k + \Delta\xi) - T(0)\|^2 \tag{3.3}$$

iteratively, where ξ_k are the warp parameters at k^{th} iteration. We use first-order Taylor approximation to linearize this expression and to obtain warp update $\Delta\xi$. Therefore, we compute the derivative:

$$\frac{\partial}{\partial \Delta\xi}(I(\xi_k + \Delta\xi) - T(0)). \tag{3.4}$$

$T(0)$ is not depending on $\Delta\xi$, thus, we only need to take the derivative of $I(\xi_k+\Delta\xi)$ into consideration. We iterate over following steps of the LK algorithm [2]:

1. Compute the residual image $r_k(I,T) = I(\xi_k) - T(0)$.
2. Calculate the derivative of the transformed input image w.r.t. the warping update $\nabla I(\xi)\frac{\partial W}{\partial \xi}$.
3. Compute the Hessian Matrix $H = [\nabla I \frac{\partial W}{\partial \xi}]^\intercal[\nabla I \frac{\partial W}{\partial \xi}]$.
4. Obtain the warp update, $\Delta\xi_k = H^{-1}[\nabla I \frac{\partial W}{\partial \xi}]^\intercal r_k$.
5. Update the warp parameters, $\xi_{k+1} = \xi_k \circ \Delta\xi$.
6. Continue with step one until convergence.

The transformation function can be chosen arbitrarily as long as it is differentiable w.r.t. the transformation parameter ξ. Note that computing the gradient of the transformed image in each step is very time-consuming.

3.1.2 Inverse Compositional Algorithm

To avoid the costly gradient recomputing in each step, we apply the warping update $\Delta\xi$ to T instead of I. We avoid computing the gradients of the input image $I(\xi)$ because it does not depend on $\Delta\xi$ anymore. The new objective function is:

$$\min_{\Delta\xi} \|I(\xi_k) - T(0 + \Delta\xi)\|^2. \tag{3.5}$$

To linearize this, we use the first-order Taylor approximation. The derivative of $T(0+\Delta\xi)$ can now be precomputed, because $T(0)$ does not change in each iteration and therefore $\frac{\partial T}{\partial \Delta\xi}$ stays the same. This saves some time and leads to the Inverse Compositional algortihm in [2]:

1. Precompute the gradient of the transformed image $\nabla T \frac{\partial W}{\partial \xi}$.
2. Precompute the Hessian $H = \left[\nabla T \frac{\partial W}{\partial \xi}\right]^\intercal \left[\nabla T \frac{\partial W}{\partial \xi}\right]$.

3. Calculate the residual image $r_k = I(\xi_k) - T(0)$.
4. Obtain warp update $\Delta\xi = H^{-1}\left[\nabla T\frac{\partial W}{\partial \xi}\right]^\intercal r_k$.
5. Update warp parameters $\xi_{k+1} = \xi_k \circ (\Delta\xi)^{-1}$.
6. Continue with step 3 until convergence.

A faster version of the original LK algorithm [2] is obtained, the Inverse Compositional algorithm. Note that we change the manner of updating our warp parameter. Instead of applying the warp update itself, the inverse of the update needs to be applied. The residual of I and T does not change if we apply a transformation on I or apply the inverse transformation on T.

3.1.3 Robust IC Algorithm

Some parts of an image can be less critical for the correct transformation than other parts. For example, it could be that the edge of the first image is not in second image, because the sensor was moved. In the current approach, the residuals of the image will take the edge in the same consideration as the rest of the image, and the resulting high errors distort the result. Therefore, a weight matrix W is introduced to balance the influence of different parts of the image [2]. The objective changes to

$$\min_{\Delta\xi} r_k(\Delta\xi)^\intercal W r_k(\Delta\xi), \tag{3.6}$$

where $r_k(\Delta\xi) = I(\xi_k) - T(\Delta\xi)$ is the residual between I and T. This objective function relies on an appropiate weight matrix W. If no more information is available it can be difficult to choose W.
The main difference compared to the steps of the inverse compositional approach is the calculation of the approximated Hessian matrix, where the weight matrix has to be taken into consideration. Because the approximated Hessian matrix could easily become ill-conditioned a damping term λ is introduced:

$$H = (J^\intercal W J) + \lambda diag(J^\intercal W J), \tag{3.7}$$

with $J = \left[\nabla T\frac{\partial W}{\partial \xi}\right]$. This leads to the Levenberg-Marquardt update step. The following steps apply the robust version of the IC algorithm:

1. Precompute the approximated Hessian matrix $H = (J^\intercal W J) + \lambda diag(J^\intercal W J)$.
2. Compute residuals $r_k = I(\xi_k) - T(\Delta\xi)$.

3. Obtain warp update $\Delta\xi = H^{-1} J^{\intercal} W r_k(0)$.

4. Update warp update $\xi_{k+1} = \xi_k \circ (\Delta\xi)^{-1}$.

5. Continue with step 2 until satified.

Keep in mind that this assumes that we already got suited λ and W.

3.2 Feature Extractor

Many approaches focused on improving feature representations for dense image alignment from handcrafted (SIFT [18], BRIEF [4]) to learned. Some methods with learned features use pre-trained models for feature extraction [6], [15], but these pre-trained feature extractors are not naturally consistent across different views. A more promising approach is to train the feature extractor along with the rest of the framework. Xu et al. [35] divide the feature extractor into three parts. First comes the Two-View encoder, which finds spatio-temporal information in the two images. Second, there is the actual feature extractor which outputs the feature representation. Then they expand the feature extractor with a network that provides an uncertainty map for the images. The weights are shared between the extractor and the uncertainty map. The feature-image and the uncertainty map are used for the least square minimization with a Gauss-Newton algorithm. The whole framework performs on four coarse levels, which leads to a more robust result against large motions. Dusmanu et al. [8] introduce a cross-descriptor for localization and mapping. It addresses the problem of combining different ever-developing features.
Point clouds are used very often in robotics. In the past, the most common approach for using features to align point clouds was to use handcrafted features over learned features. Rusu et al. [26] describe each point with a 16D feature histogram. This is used as starting point for other iterative registration algorithms like ICP. Other approaches use volume descriptor [12], or fast point feature histograms [25]. Dealing with point clouds in Neural Networks can be difficult due to their instability in size and because they are usually given in an unordered way. Nevertheless, some approaches are dealing with point clouds, like PointNet [21]. PointNet uses features for segmentation. Instead of splitting the task of computing a feature representation and aligning, it is built in one network. Du et al. [7] use local and global learned features to obtain a 6 DoF pose for relocalization. This is used for relocalization.

3.3 Iterative Closest Point

Aligning two point clouds to each other is a common problem in SLAM systems. For a long time, ICP was the standard alignment method. The Generalized Iterative Closest Point (GICP) algorithm improves the standard ICP. It combines the point-to-point ICP with the point-to-plane ICP, resulting in a plane-to-plane algorithm. Here we will give a short explanation of ICP and how ICP leads to GICP.

In ICP, two steps are repeated until the convergence criterion is satisfied. First, correspondences between A and B are computed by searching for the nearest neighbour of each point. Second, a transformation is computed which minimizes the distance between corresponding points. A threshold t_{dist} is necessary because we can not assume a full overlap between the two scans. If the nearest point $b_i \in B$ of $a_i \in A$ is further away than t_{dist} this correspondence is not taken into consideration. Then the transformation T is updated by minimizing the distance of $T \cdot a_i$ and its corresponding b_i:

$$T \leftarrow \min_{T} \|T \cdot a_i - b_i\|. \tag{3.8}$$

In the point-to-plane variant of ICP, we update our transformation by minimizing the distance of a_i along the normal direction of a plane spanned by at least three b_is. Therefore, a matching between A and the surfaces that are represented in B is searched.

$$T \leftarrow \min_{T} \|\eta_i \cdot (T \cdot a_i - b_i)\|, \tag{3.9}$$

where η_i is the surface normal at b_i.

In GICP a probabilistic model is attached to the minimization in the update step of T, which considers the covariance of each point. For example, points on a plane are restricted more strongly by the surface normal. This can be used to establish a plane-to-plane correspondence between A and B. For a detailed explanation of this probabilistic model, we refer to Segal et al. [29]. With this plane-to-plane matching, we rather try to find a transformation between the surfaces that are represented by the point clouds than try to match the points themselves.

3.4 Metrics

Iterative methods for visual odometry (VO) and SLAM usually need to compare a transformed image T with a reference image I to align them. For example, after transforming one image, it is compared to another image. This comparison is a metric for the quality of the transformation. It needs to have an optimum if the images are the same. Other than that, different functions will change the importance of different features in the image and therefore influence the results drastically. In direct image alignment, the photometric error is often used for this purpose, which is the difference between two intensity images. Let $I, T \in \Omega \subset \mathbb{R}^{h \times w}$ be two images, and $\mathbf{x} = (u, v)$ be a pixel location. Then the photometric error between I and T is:

$$r_d(\mathbf{x}) = I(\mathbf{x}) - T(\mathbf{x}). \tag{3.10}$$

There are several other metrics to compare two images that robustify the residuals, such as the Huber norm used in [9]. Quenzel et al. [23] propose a new metric for pixel-wise matching which considers the gradient orientation and the magnitude of the intensities. This metric and variations of it are explained in Section 4.1

3.5 Deep Inverse Compositional Algorithm

The Deep Inverse Compositional algorithm [19] was proposed by Lv et al. to integrate learning features into the robust version of the LK algorithm. The learned features lift the consumptions that the damping term λ and weight matrix W is known, and it results in a framework that can be trained in an end-to-end manner. The three main contributions in this framework are the **Two-View Feature Encoder**, the **Convolutional M-Estimator** and the **Trust Region Network**. The framework operates on four coarse levels to handle large motions, each level halves the image size of the previous level.

3.5.1 Two-View Feature Encoder

By using a feature representation of the images, the brightness constancy constraint is relaxed. This leads to a more robust representation than the intensity image. The Two-View feature encoder is a convolutional network Φ which processes the two image I, T to obtain feature representations I_θ, T_θ. To achieve this for each feature representation it takes both images as input.

$$I_\theta = \Phi(I, T), \tag{3.11}$$

$$T_\theta = \Phi(T, I). \tag{3.12}$$

As one can see in Figure 3.1, on each coarse-level, it has three dilated convolutional layers, followed by a batch normalization layer. The obtained features are summed up to gain a one-dimensional feature image on which the algorithm will be performed. By taken input image I and template image T into consideration for the feature representation, both spatial and temporal information are taken into account.

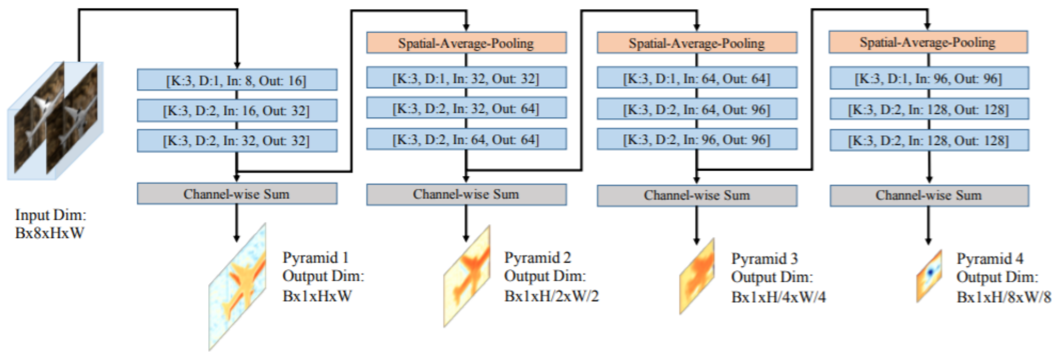
Input Dim:
Bx8xHxW
[K:3, D:1, In: 8, Out: 16]
[K:3, D:2, In: 16, Out: 32]
[K:3, D:2, In: 32, Out: 32]
Channel-wise Sum
Pyramid 1
Output Dim:
Bx1xHxW
Spatial-Average-Pooling
[K:3, D:1, In: 32, Out: 32]
[K:3, D:2, In: 32, Out: 64]
[K:3, D:2, In: 64, Out: 64]
Channel-wise Sum
Pyramid 2
Output Dim:
Bx1xH/2xW/2
Spatial-Average-Pooling
[K:3, D:1, In: 64, Out: 64]
[K:3, D:2, In: 64, Out: 96]
[K:3, D:2, In: 96, Out: 96]
Channel-wise Sum
Pyramid 3
Output Dim:
Bx1xH/4xW/4
Spatial-Average-Pooling
[K:3, D:1, In: 96, Out: 96]
[K:3, D:2, In: 128, Out: 128]
[K:3, D:2, In: 128, Out: 128]
Channel-wise Sum
Pyramid 4
Output Dim:
Bx1xH/8xW/8

Figure 3.1: Two-View Feature Encoder [19]

3.5.2 Convolutional M-Estimator

As mentioned before, the weight matrix W is often hard to find. To deal with this issue, Lv et al. propose a fully convolutional network that predicts the weight matrix W. This convolutional M-Estimator ψ takes the residual of the feature images and the feature images itself as input. It also takes the estimated weight matrix from the coarser level of the algorithm as input. It takes the identity matrix as additional input at the highest resolution because of the lack of a coarser level.

$$W_i = \psi(W_{i-1}, r, I_\theta, T_\theta), \tag{3.13}$$

where i is the coarseness level. The proposed diagonal weight matrix W and especially the underlying robust function ρ are not restricted to a particular error model but on the input data itself. Figure 3.2 gives an overview of the Convolutional M-Estimator.

3.5.3 Trust Region Network

The third core component of [19] is the Trust Region Network, pictured in Figure 3.3. The Trust Region Network tries to find a suited damping term λ_θ to overcome instability in the approximated Hessian matrix. This λ_θ replaces $\lambda diag(J^\intercal W J)$ in

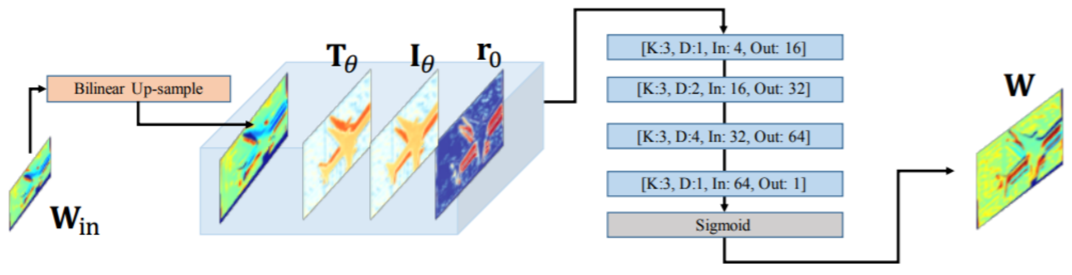

Figure 3.2: Convolutional M-Estimator [19]

the Levenberg-Marquardt update step.
To achieve this first a set of N damping proposals λ_i is sampled on a logarithmic scale. With these hypothetical damping terms, the resulting Levenberg-Marquardt update steps is computed:

$$\Delta\xi = ((J^\intercal WJ) + \lambda_i diag(J^\intercal WJ))^{-1} J^\intercal W r_k(0). \tag{3.14}$$

With these hypothetical update steps residuals are computed.

$$r^i_{k+1} = I_\theta(\xi_k \circ (\Delta\xi_i)^{-1}) - T_\theta(0) \tag{3.15}$$

These N hypothetical residuals as well as the approximated Hessian matrix $J^\intercal WJ$ are fed into the Trust Region Network. It contains 3 fully connected layer each with the ReLu transfer function.

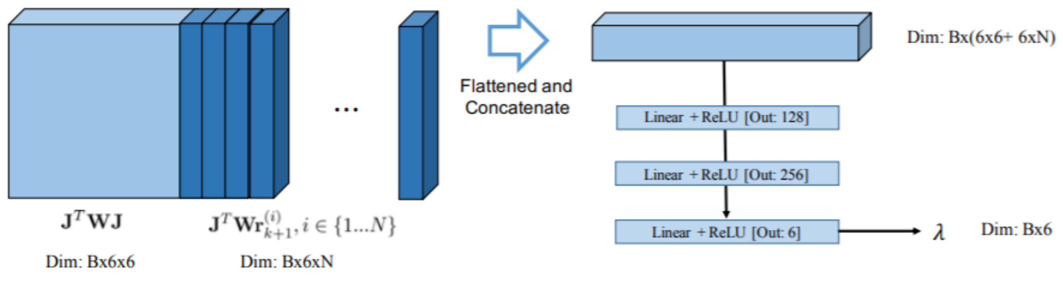

Figure 3.3: Trust Region Network [19]

3.5.4 Summary

Here is an overview over the funcionality of the whole framework [19]:

1. Feed the images [I,T] through the Two-View Feature Encoder to obtain the feature representations I_θ, T_θ for all 4 levels.

2. Compute the residuals $r_0(I_\theta, T_\theta) = I_\theta - T_\theta$.
3. Precompute the Jacobians with respective to the warping parameters $\nabla T \frac{\partial W}{\partial \xi}$.
4. Feed the residuals r_0, the feature images I_θ, T_θ and the warping function as well as the weight matrix W of the coarser level to the Convolutional M-Estimator to gain the weight matrix W.
5. Compute the approximated Hessian matrix $H = J^\intercal W J$.
6. Compute the residuals of the k^{th} iteration $r_k = I_\theta(\xi) - T_\theta$.
7. Sample N damping proposels λ_i and compute the resulting residuals r^i_{k+1}.
8. Feed r^i_{k+1} and r_k to the Trust Region Network to obtain λ_θ.
9. Calculate warping update $\Delta\xi = (H + \lambda_\theta diag(H)^{-1} J^\intercal W r_k$.
10. Update warp parameter $\xi_{k+1} = \xi_k \circ (\Delta\xi)^{-1}$.
11. Continue with step 6 for 12 iterations.

3.5.5 Jacobians

In the following section we will specify how the Jacobians of the residual image w.r.t. the warping parameters are calculated. To get the Jacobians the chain rule is applied, therefore the derivatives of the residual Image w.r.t. the image dimensions u,v, the derivatives of the projection w.r.t. world dimensions x,y,z, and the derivative of the transformation w.r.t. the warping parameter ξ is needed. As we already derived in Section 2.1.1 and Section 2.3.1, here are the Jacobians of the projection and the warping:

$$J_{proj} = \begin{pmatrix} f_x \\ f_y \end{pmatrix} \cdot \begin{pmatrix} \frac{1}{z} & 0 & -\frac{x}{z^2} \\ 0 & \frac{1}{z} & (-\frac{y}{z^2}) \end{pmatrix}, \tag{3.16}$$

$$J_{warp} = \begin{pmatrix} 0 & z & -y & 1 & 0 & 0 \\ -z & 0 & x & 0 & 1 & 0 \\ y & -x & 0 & 0 & 0 & 1 \end{pmatrix}. \tag{3.17}$$

Multiplying these leads to:

$$J_{proj} \cdot J_{warp} = \begin{pmatrix} f_x \\ f_y \end{pmatrix} \cdot \begin{pmatrix} -\frac{xy}{z^2} & 1 + \frac{x^2}{z^2} & \frac{-y}{z} & \frac{1}{z} & 0 & -\frac{x}{z^2} \\ (-1 - \frac{y^2}{z^2}) & \frac{xy}{z^2} & \frac{x}{z} & 0 & \frac{1}{z} & -\frac{y}{z^2} \end{pmatrix}. \tag{3.18}$$

Lv et al. further simplified this by exploiting the inverve depth parameterization $p = (\frac{u}{d}, \frac{v}{d}, \frac{1}{d})$, with (u, v) as image location and d as its associated depth[19]:

$$J_{pw} = J_{proj} \cdot J_{warp} = \begin{pmatrix} f_x \\ f_y \end{pmatrix} \cdot \left(\begin{array}{cccccc} -uv & 1+u^2 & -v & d & 0 & -dv \\ -1-v^2 & uv & u & 0 & d & dv \end{array} \right). \tag{3.19}$$

The desired Jacobians of the template image w.r.t. the warping parameters is the combination of 3.19 and the Jacobians of the residuals. Since the difference is used to compute the residuals, the gradients of the template image can be used:

$$\mathbf{J} = \nabla T \cdot J_{pw}. \tag{3.20}$$

4 Method

In our method, we build on the work by Lv et al. [19] by using their learning approach of the inverse compositional variant of the LK algorithm previously described in Section 3.1.3. The changes to the framework are explained here. Figure 4.1 gives an overview of the original method. We will detail made changes in the following sections.

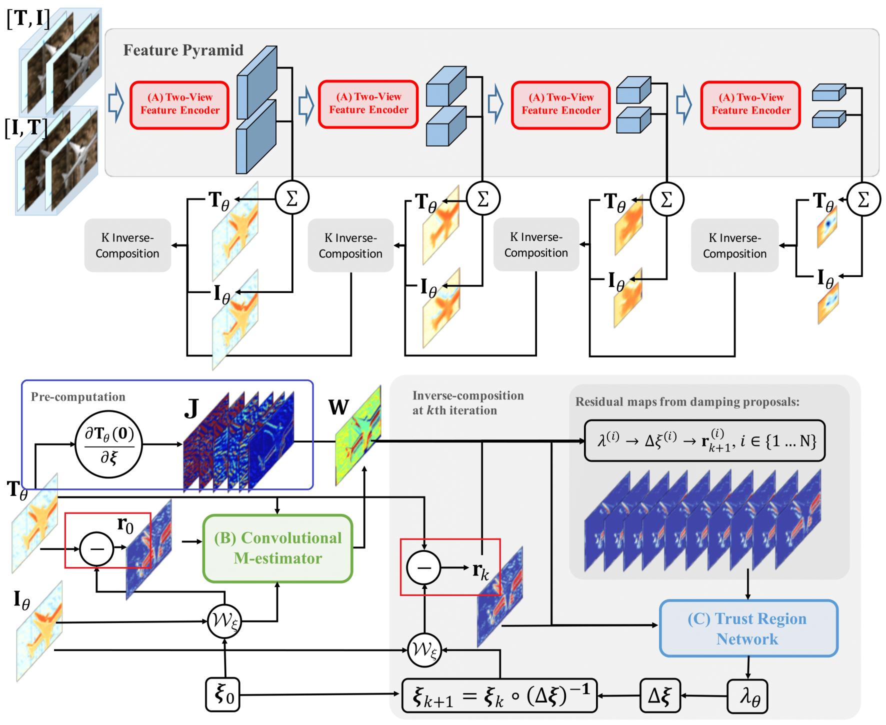
[T, I]
[I, T]
Feature Pyramid
(A) Two-View Feature Encoder
$\mathbf{T}_\theta$
$\mathbf{I}_\theta$
Σ
K Inverse-Composition
Pre-computation
$\frac{\partial \mathbf{T}_\theta(\mathbf{0})}{\partial \xi}$
J
W
$\mathbf{r}_0$
(B) Convolutional M-estimator
$\mathcal{W}_\xi$
ξ_0
Inverse-composition at kth iteration
Residual maps from damping proposals:
$\lambda^{(i)} \to \Delta\xi^{(i)} \to \mathbf{r}_{k+1}^{(i)}, i \in \{1 \dots N\}$
$\mathbf{r}_k$
(C) Trust Region Network
$\xi_{k+1} = \xi_k \circ (\Delta\xi)^{-1}$
$\Delta\xi$
λ_θ

Figure 4.1: Overview of the Framework by [19]

4.1 Residuals

There are multiple methods to point out the differences between image I and image T. A simple but effective approach is to compute the pixel-wise difference between the two images, the photometric error, described in Equation (3.10), which is used by Lv et al. [19]. Figure 4.2 gives an example of a residual computed with r_d. White pixel means there is a high error and black pixel means there is no error. As one can see in Figure 4.1, there are two spots where a residual is needed. These are marked with a red rectangle. r_0 is part of the precomputation, r_k is computed in each step of the iteration. In our method, we test this framework with other metrics to compute the residual image. Each of them having several advantages compared to the difference. However with introducing new metrics to the framework the procedure changes. To give an idea of the residuals each will be performed on Fig. 4.3.

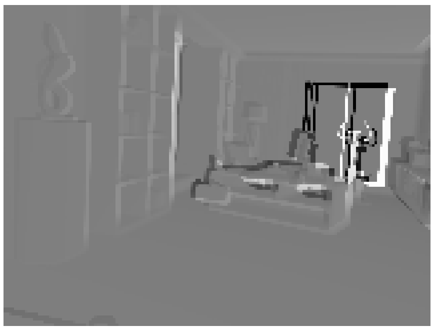
Figure 4.2: difference

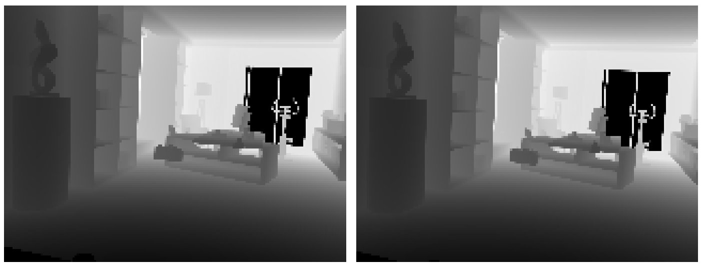
Figure 4.3: depth image of two images out of the ICL dataset [27]

4.1.1 Gradient Magnitude

A more robust version of the difference relies on the gradient:

$$r_{gm} = \|\nabla I(\mathbf{x})\| - \|\nabla T(\mathbf{x})\|, \tag{4.1}$$

where $\mathbf{x} = (u, v)$ is a pixel location. r_{gm} has an advantage over the following residuals, the Jacobians only rely on T:

$$\frac{\partial r_{gm}}{\partial \mathbf{x}} = \frac{\nabla_{\mathbf{x}v} T + \nabla_{\mathbf{x}u} T}{2\sqrt{\nabla_u T^2 + \nabla_v T^2}}. \tag{4.2}$$

In Figure 4.4 is an example for a residual computed with r_{gm}. White pixel means

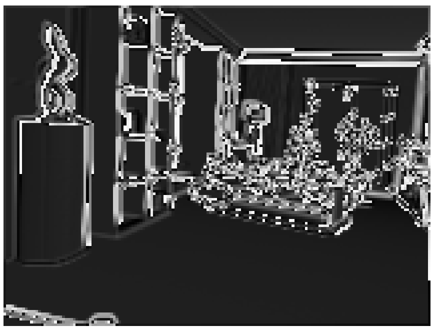

Figure 4.4: residuals with gradient magnitude

there is a high error and black pixel means there is no error.

4.1.2 Cosine Similarity

Cosine similarity is a widely used metric for learning different tasks [17]. To measure the cosine similarity for two images, the dot-product is computed and then scaled by the maximum of the euclidean distances of the single images:

$$r_{cossim}(\mathbf{x}) = \frac{I(\mathbf{x}) \cdot T(\mathbf{x})}{(\|I(\mathbf{x})\| \|T(\mathbf{x})\|) + \epsilon}. \tag{4.3}$$

ϵ is a small constant that ensures stability when $\|I\|, \|T\|$ are close to zero. The advantage of cosine similarity compared to the difference is the robustness in terms of the magnitude of the image. When both image I and T only differ in lighting, e.g. $T = I + \alpha_1$, the gradient of r_d will be strong everywhere. Cosine Similarity avoids this.
The derivation of the cosine similarity w.r.t. to $\mathbf{x}$ is:

$$\frac{\partial r_{cossim}(\mathbf{x})}{\partial \mathbf{x}} = \frac{T(\mathbf{x})}{|I(\mathbf{x})| \cdot |T(\mathbf{x})|} - r_{cossim}(\mathbf{x}) \cdot \frac{I(\mathbf{x})}{|I(\mathbf{x})|^2}. \tag{4.4}$$

4.1.3 $\mathbf{r_{sgf}}$

This metric was proposed by Quenzel et al. [23]. The main advantage compared to other metrics is that it combines a gradient orientation-based metric, such as the cosine similarity, with a magnitude depending scaling term. Quenzel et al. propose several variants of the metric. In our work, we will focus on two of them, the r_{sgf} metric explained here and the r_{sgf3} metric explained in the following section. In r_{sgf} the gradient orientation is aligned with the dot-product following the approach of [20] and [33]. This is more efficient than using the costly tanh operator [23]. A parameter ϵ is introduced to normalize the gradients:

$$\epsilon = \frac{1}{|I|} \sum_{x \in I} \|\nabla I(\mathbf{x})\|^2, \tag{4.5}$$

$$\nabla_\epsilon I = \frac{\nabla I}{\sqrt{\|\nabla I\|^2 + \epsilon}}. \tag{4.6}$$

The orientation of the normalized gradients $\nabla_\epsilon I(\mathbf{x}) \cdot \nabla_\psi T(\mathbf{x})$ will now be scaled by the maximum value, this leads us to the equation:

$$r_{sgf}(\mathbf{x}) = 1 - \frac{\nabla_\epsilon I(\mathbf{x}) \cdot \nabla_\psi T(\mathbf{x})}{\max(\|\nabla_\epsilon I(\mathbf{x})\|^2, \|\nabla_\psi T(\mathbf{x})\|^2, \tau)}. \tag{4.7}$$

We choose ϵ and ψ as normalization parameter to clarify that these are two different values. The normalization parameter is computed on a per image base. τ is a small constant that prevents division by zero. An example for r_{sgf} is given in Figure 4.5.

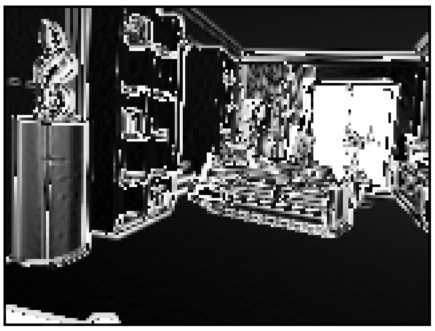

Figure 4.5: r_{sgf}

4.1.4 $\mathbf{r_{sgf3}}$

A simplification of r_{sgf} is r_{sgf3}. It reduces the number of mathematical operations especially the regularization leading to a more slim formula with similar advantages as r_{sgf}. The orientation and magnitude is now combined as follows:

$$r_{sgf3}(\mathbf{x}) = \|\nabla I(\mathbf{x})\| \|\nabla T(\mathbf{x})\|| - n(\mathbf{x}), \tag{4.8}$$

where $n(\mathbf{x}) = \nabla I(\mathbf{x}) \cdot \nabla T(\mathbf{x})$ is the dot-product of I and T for each pixel location. Other than being more slim than r_{sgf}, r_{sgf3} has another advantage for our purpose. The Jacobians w.r.t. the pixels are easier to compute:

$$\frac{\partial r_{sgf}}{\partial \mathbf{x}} = \left(\frac{1}{2} \frac{\|\nabla T\|}{\|\nabla I\|} \nabla I - \nabla T \right) (\nabla_2) I. \tag{4.9}$$

$(\nabla_2)I$ denotes the hessian of the intensity at pixel $\mathbf{x}$ [23]. Figure 4.6 gives an

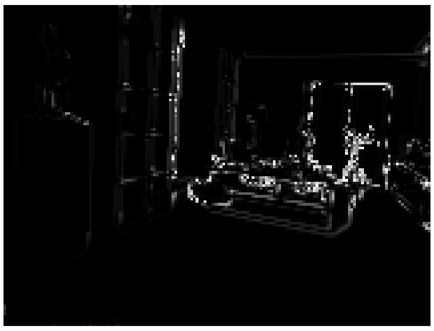
Figure 4.6: r_{sgf3}

example of r_{sgf3}.

4.1.5 Structural Similarity Index

This metric is focused on the structure of the scene in the image. It combines a local luminance term with local measured contrast term and adds a structure term which leads to a overall similarity measure [38]:

$$S(\mathbf{x}) = f(l(\mathbf{x}), c(\mathbf{x}), s(\mathbf{x})). \tag{4.10}$$

These comparison terms should satisfy the following conditions [34]:

1. Symmetry: $S_{I,T}(\mathbf{x}) = S_{T,I} = (\mathbf{x})$.
2. Boundedness: $S(\mathbf{x}) \leq 1$.
3. Unique maximum: $S(\mathbf{x}) = 1$ only if I=T.

For the local luminance term first the mean intensity in a chosen window is computed:

$$\mu_I = \frac{1}{N}\sum_{i=1}^{N} x_i. \tag{4.11}$$

The luminance term is defined as:

$$l(\mathbf{x}) = \frac{2\mu_I\mu_T + C_1}{\mu_I^2 + \mu_T^2 + C_1}. \tag{4.12}$$

C1 is a small constant that ensures stability in case $\mu_I^2 + \mu_t^2$ is close to zero. The contrast comparison function is similar to the luminance function but instead of taking the mean of the intensity the standard deviation in a given window is used.

$$c(\mathbf{x}) = \frac{2\sigma_I\sigma_T + C_2}{\sigma_I^2 + \sigma_T^2 + C_2}, \tag{4.13}$$

where σ_I is computed over the Image with a window size N as follows:

$$\sigma_I = \left(\frac{1}{N-1}\sum_{i=1}^{N}(I(x_i) - \mu_I)^2\right)^{\frac{1}{2}}. \tag{4.14}$$

Once again a constant C_2 is introduced to avoid instability.
The structure comparison is computed as follows:

$$s(\mathbf{x}) = \frac{\sigma_{I,T} + C_3}{\sigma_I\sigma_T + C_3}. \tag{4.15}$$

Note that $s(\mathbf{x})$ can take negative values. The three part terms are combined to the Structural Similarity (SSIM) index between image I and T:

$$\tilde{r}_{ssim}(\mathbf{x}) = [l(\mathbf{x})]^\alpha \cdot [c(\mathbf{x})]^\beta \cdot [s(\mathbf{x})]^\gamma. \tag{4.16}$$

To balance the influence of the three components α, β and γ are to be adjusted. To further simplify the equation α, β and γ are set to one and $C_3 = C_2/2$, this results in:

$$\tilde{r}_{ssim}(\mathbf{x}) = \frac{(2\mu_I\mu_T + C_1)(2\sigma_{I,T} + C_2)}{(\mu_I^2 + \mu_T^2 + C_1)(\sigma_I^2 + \sigma_T^2 + C_2)}. \tag{4.17}$$

To use the Structural Similarity Index as a residual measurement it needs to be ensured that in the best case I=T the residual function is zero. This leads to the residual function:

$$r_{ssim}(\mathbf{x}) = 1 - \tilde{r}_{ssim}(\mathbf{x}). \tag{4.18}$$

Please keep in mind that the SSIM index is computed over a window for each pixel in the images. For the implementation the pixel within the window for a given pixel is weighted in a gaussian manner. So 4.11 and 4.14 is implemented slightly different.
SSIM can be divided in two parts $l(\mathbf{x}) \cdot cs(\mathbf{x})$. The first part is the luminance function (4.12). The second part is the combination of the contrast (4.13) and the structure function (4.15):

$$cs(\mathbf{x}) = \frac{2\sigma_{IT} + C_2}{\sigma_I^2 + \sigma_T^2 + C_2}. \tag{4.19}$$

To obtain the Jacobians of the residuals at each pixel location the derivative w.r.t. each other pixel q in the window W needs to be computed. More formally [38]:

$$\frac{\partial r_{ssim}(W)}{\partial I(q)} = -\frac{\partial \tilde{r}_{ssim}(\mathbf{x})}{\partial I(q)} = -\left(\frac{\partial l(\mathbf{x})}{\partial I(q)} \cdot cs(\mathbf{x}) + l(\mathbf{x}) \cdot \frac{\partial cs(\mathbf{x})}{\partial I(q)}\right). \tag{4.20}$$

With the derivatives of:

$$\frac{\partial l(\mathbf{x})}{\partial I(q)} = 2 \cdot G_{\sigma_G}(x-q) \cdot \left(\frac{\mu_T - \mu_I \cdot l(\mathbf{x})}{\mu_I^2 + \mu_T^2 + C_1}\right), \tag{4.21}$$

and

$$\frac{\partial cs(\mathbf{x})}{\partial I(q)} = \frac{2}{\sigma_I^2 + \sigma_T^2 + C_2} \cdot G_{\sigma_G}(x-q) \cdot \left[(T(q) - \mu_T) - cs(\mathbf{x}) \cdot (I(q) - \mu_I)\right]. \tag{4.22}$$

$G_{\sigma_G}(x-q)$ weights the pixel x with a gaussian centered at q and standard deviation σ_G. Figure 4.7 gives an example for r_{ssim}.

4.2 Jacobians of the Residuals

With the difference as residual, the Jacobians and the resulting Hesse matrix can be precomputed because they only depend on the template image T, which is not changing in each step of iteration. One can see it in the blue rectangle in Fig. 4.1. The newly introduced residuals have more complicated derivatives, which are not only depending on the template image but on the input image I as well. I is changing in each step of the iteration because the warping update is applied to it.

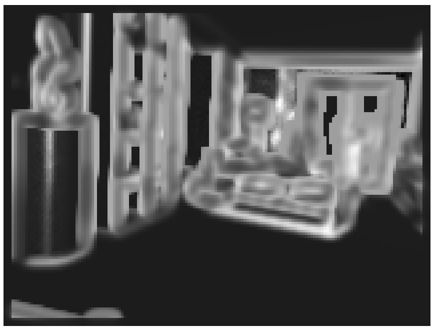
Figure 4.7: r_{ssim}

Therefore the Jacobians and with it the approximated Hesse matrix needs to be recomputed in each step of the iteration. Thus, the advantage of the IC version of the LK algorithm is lost.
The learning modules make the framework flexible. To test this flexibility, in our experiments, we train the network with the new residuals but do not recompute the gradients in each iteration. This will be referred to as the "light" version. Then we compare the light version with the correct version, where gradients are recomputed in each step of the iteration. In theory, the Convolutional M-Estimator should be able to catch some errors caused by the inexact gradients.
The chained Jacobians J_{pw} of the warping and the projection are computed as described in Section 3.5.5. The gradient of the image w.r.t. the warping parameters depend on the used residual:

$$\mathbf{J} = \frac{\partial r}{\partial \mathbf{x}} \cdot J_{pw}. \tag{4.23}$$

4.3 LiDAR

We adapt the framework for LiDAR range images (LRI). The computation of an LRI from a point cloud is described in Section 2.2. LiDARs provide more accurate depth and larger field-of-views (FoV) than RGB-D sensors, but are less dense. We adjust the framework to deal with LRIs.´

4.3.1 Downsampling for LRI

LRIs are relatively small in height because each row requires another laser during scanning as described in Section 2.2. Downsampling an image I of size $w \times h$ in the framework results in $\tilde{I}$ of size $\frac{w}{2} \times \frac{h}{2}$. By dividing an LRI further in height, there are too few data points to perform the algorithm. We decided to downsample the

LRIs only in width.
A common practice for downsampling an image is to perform max- or average-pooling on it. That can lead to problems in LRI. Given that neighboring pixels are not always representing the same object, average-pooling gives a data point between these two objects, but in reality, there might be no object. For this reason, average-pooling is not suitable for downsampling an LRI.
Max-pooling would prefer objects further away over near objects. Assuming that it is easier to learn transformations of closer objects because often they are more detailed and appear broader in the image due to sensor geometry min-pooling is more appropriate to downsample the LRI in the w dimension. In addition to this closer points are prefered by LiDAR. This ensures that closer object are better represented than objects far away.

4.3.2 LRI from Point Clouds

The conversion of a point cloud into an LRI is detailed in Section 2.2. In theory, every point is represented, every point gets represented by one pixel in the image. In praxis, the slightest distortion can lead to errors, such as two points falling on the same pixel leaving one pixel without value. This can lead to problems for the Convolutional Feature Encoder. One such hole-filling method is inpainting [37].
In our experiments, we use the DRZ Living Lab dataset [22], which is already pre-processed to have sparse data. The second dataset we use is the KITTI dataset. Due to the very high resolution of the KITTI dataset in width, another method can be applied. We reduce the image width, which leads to a much denser representation. To record the KITTI dataset a Velodyne HDL-64E is used. It has a approximatly 0.175° horizontal resolution and it has 64 channels. This leads to an image resolution of 2048x64 pixels. Using this resolution results in incomplete data. The upper image in Fig. 4.8 gives an example. If we reduce the image width to 1024x56 pixels, we obtain a much better representation, as one can see in the lower image in Fig. 4.8.

4.3.3 Two-View-Feature Encoder

A good feature representation is crucial for the results of the algorithm. The point clouds exhibit different characteristics compared to RGB-D images. While they lack color information, they have an intensity and reflectivity value per pixel. Additionally, we add the point coordinates in the sensor frame. This information might be helpful to align point clouds.
We feed the LRI together with the cartesian coordinates, the intensity, and the

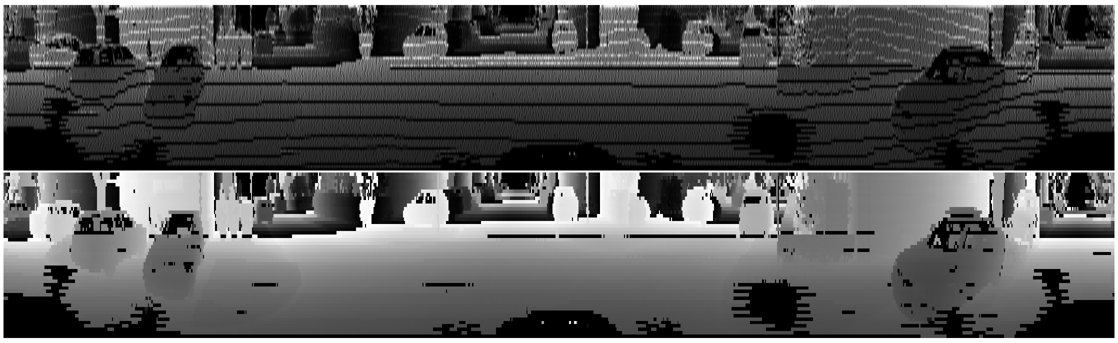
Figure 4.8: Different resolutions of LRI out of the KITTI dataset.

reflection value to the Two-View-Feature Encoder. Except for the input, we do not change the structure of the Two-View-Feature encoder as explained in Section 3.5.1.

4.3.4 Jacobians of LRI

We combine the Jacobian of the LRI projection, as described in Section 2.2.1, with the Jacobian of the warping as follows:

$$\begin{aligned}
J_{pwl} &= J_{proj} \cdot J_{warp} \\
&= \begin{pmatrix} -\frac{y}{x^2+y^2} & \frac{x}{x^2+y^2} & 0 \\ -\frac{xz}{(x^2+y^2)r^2} & -\frac{yz}{(x^2+y^2)r^2} & \frac{\sqrt{x^2+y^2}}{r^2} \end{pmatrix} \cdot \begin{pmatrix} 0 & z & -y & 1 & 0 & 0 \\ -z & 0 & x & 0 & 1 & 0 \\ y & -x & 0 & 0 & 0 & 1 \end{pmatrix} \\
&= \begin{pmatrix} -\frac{zx}{x^2+y^2} & -\frac{zy}{x^2+y^2} & 1 & -\frac{y}{x^2+y^2} & \frac{x}{x^2+y^2} & 0 \\ \frac{yz^2+\sqrt{x^2+y^2}y}{(x^2+y^2)r^2} & \frac{-xz^2}{(x^2+y^2)r^2} - \frac{x\sqrt{x^2+y^2}}{r^2} & 0 & -\frac{xz}{(x^2+y^2)r^2} & -\frac{yz}{(x^2+y^2)r^2} & \frac{\sqrt{x^2+y^2}}{r^2} \end{pmatrix}.
\end{aligned} \tag{4.24}$$

This is combined with the derivatives of the residual image:

$$\mathbf{J} = \frac{\partial r}{\partial \mathbf{x}} \cdot J_{pwl}. \tag{4.25}$$

4.4 Summary

As already mentioned, for some residual, we need to recompute the Jacobians in each step of the iteration. This leads to the following algorithm.

1. Feed the data of both frames [I, T] through the Two-View Feature Encoder to obtain the feature representations I_θ, T_θ for all four levels.

2. Compute the residuals $r_0(I_\theta, T_\theta)$.
3. Feed the residuals r_0, the feature images I_θ, T_θ and the weight matrix W of the coarser level to the Convolutional M-Estimator to gain the weight matrix W.
4. Compute the Jacobians $\frac{\partial r_k}{\partial \xi}$.
5. Compute the approximated Hessian matrix $J^\intercal W J$.
6. Sample N damping proposels λ_i and compute the resulting residuals r^i_{k+1}.
7. Feed r^i_{k+1} and r_k to the Trust Region Network to obtain λ_θ.
8. Calculate warping update $\Delta\xi = (J^\intercal W J) + \lambda_i diag(J^\intercal W J)^{-1} J^\intercal W r_k$.
9. Update warp parameter $\xi_{k+1} = \xi_k \circ (\Delta\xi)^{-1}$.
10. Compute the residuals of the $k + 1^{th}$ iteration $r_{k+1} = I_\theta(\xi) - T_\theta$.
11. Continue with step 4 for 12 iterations.

5 Evaluation

5.1 Datasets

For training and evaluation, we use multiple datasets, synthetic as well as real-world scenes. The datasets should meet some requirements. Firstly, it should provide correct depth estimates. In reality, sensors do not always provide reliable and dense depth. Nevertheless, the algorithm needs to cope with this. As we focus on the comparison of different residuals, a robust residual metric should be able to deal with a few outliers or missing data points.
Secondly, the intrinsic and extrinsic parameters of the sensor are required. Most datasets fullfill this constraint. Thirdly, the scenes and trajectories should vary to such that the network will generalize well and prevent over-fitting. Unfortunately, some datasets such as Hypersim [24] provide camera trajectories with too large gaps between consecutive frames. A central assumption within the framework is to have a similar view-point. Thus, it will likely fail if the overlap is too small. The advantage of synthetic data is that the sensors cause no errors. The network relies on accurate depth information. If there are too many corruptions, the network does not train efficiently. The LiDAR data is very precise in its depth. Following [19] we chose TUM RGB-D [32] and MovingObjects3D [5] for the comparison of different metrics. Additionally, we choose the synthetic ICL [27] dataset. Our adaptation to LIDAR is tested on the DRZ Living Lab [28] and KITTI [11] dataset.

5.1.1 TUM RGB-D

TUM RGB-D is a standard benchmark that is used to compare many RGB-D SLAM frameworks. The TUM RGB-D [32] dataset contains color and depth images of a Microsoft Kinect sensor. The data was recorded at full frame rate (30 Hz) and a resolution of 640x480 pixels per image. Capture was performed within a motion-capture system that provides accurate ground-truth poses.
The ”pioneer360”-sequences in the dataset is obtained by a ground robot and the rest is out of a hand-held perspective. We split the trainings-, validation-, and testing-set like [19]. Most scenes are in the vicinity of one main element like a desk. Fig. 5.1 shows an example image in TUM RGBD.

5.1.2 MovingObjects3D

MovingObject3D is a selection of six objects out of the ShapeNet repository [5]. It contains richly annotated synthetically generated data of single objects like a boat or a car. Reliable depth estimation is a great advantage of synthetic data over real-world datasets. We do not use most of the annotations provided by the dataset, for example, object label, material, or weight. The annotations happen in a hybrid mode. First, they are predicted algorithmically then the annotations are verified by humans. The critical annotations for our purpose (depth, position, rotation) are generated along with the data and are therefore reliable.
The six objects used in Lv et al.'s work are 'boat' and 'motorbike' as test set and 'aeroplane', 'bicycle', 'bus', 'car' as the training set. Lv et al. render 200 video sequences with 100 frames for each object category [19]. We use the same Objects. Figure 5.1 shows an example image of MovingObjects3D.

5.1.3 ICL Pering

Sahid et al. [27] created a high-quality synthetic RGB-D dataset. It contains 16 trajectories of indoor scenes, with some in the perspective of a flying unmanned aerial vehicle (UAV), a walking person and a driving ground robot. ICL has reliable depth estimation such as Moving Objects 3D. As shown in Fig. Figure 5.1, the scenes are more complex and include mirrors and other reflecting objects. Each scene contains between 500 and 2000 images. Each frame has a resolution of 640x480 pixels. In many frames, there is not one main object like in TUM RGB-D and MovingObjects3D. Thus, the dataset contains more challenging scenes than TUM RGB-D and MovingObjects3D.

5.1.4 DRZ Living Lab

Quenzel and Behnke [22] used an unmanned aerial vehicle (UAV) [28] equipped with an Ouster OS-0 LiDAR to record multiple flights through a motion capture volume in the DRZ Living Lab. The environment is an industrial hall filled with some arbitrary objects like shelves and crates. In total 12 sequences with between 650 and 5000 scans were recorded with differing difficulty w.r.t. acceleration and angular velocity. We used seven sequences for training, one for validation, and four for testing. The training poses were provided by registering point clouds against a surfel map [22].

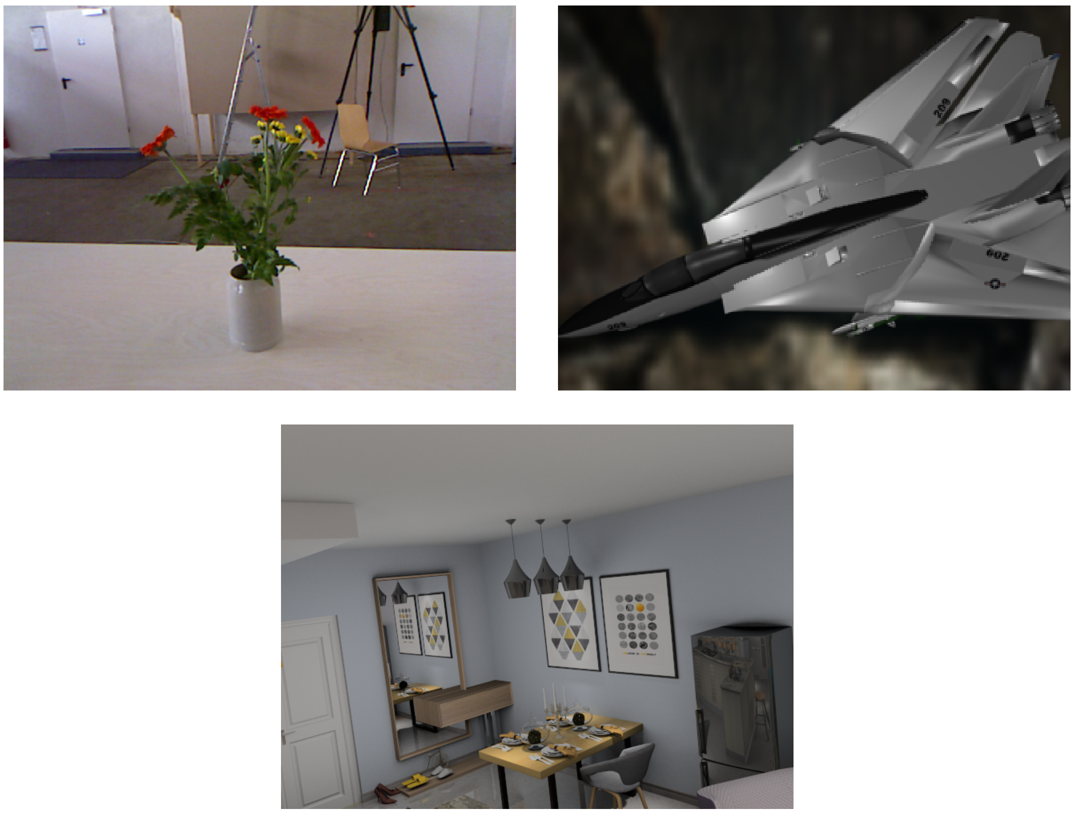

Figure 5.1: Examples of TUM RGB-D(up-lelft), MovingObjects3D(up-right), ICL(bottom)

5.1.5 KITTI

From the automotive context we chose the KITTI dataset [11]. LiDAR scans were acquired by a Velodyne HDL-64 mounted on top of a car driving through rural and urban areas of Karlsruhe. A GPS localization system provides the ground-truth. The dataset contains 22 sequences captured with this setup. We use sequences 0-7 and 9-10 as training set. For validation, we use sequence 8. The rest is used for testing.

5.2 Relative Pose Error

We evaluate and compare different residuals with the Relative Pose Error (RPE). The RPE is a measurement for the local drift of the trajectory. We follow the approach of [19] for comparability and take per frame the first, second, fourth and eighth following frame. It is computed as follows:

$$E_i = (Q_i^{-1} \cdot Q_{i+\Delta})^{-1}(O_i^{-1} \cdot O_{i+\Delta}), \tag{5.1}$$

with Q_j as the ground truth pose on frame $j = i + \Delta$ and O_j as the estimated pose on frame j. The chosen frame interval is given with Δ. The RPE is computed separately for the translation and the rotation.

5.3 End-Point-Error

The End-Point-Error is chosen as loss for training following [19]. The loss balances the influence of rotation and translation.

$$L = \frac{1}{\|I\|} \sum_{p \in I} \sum_{l \in \mathcal{L}} \|T_{gt}p - T(\xi_l)p\|_2^2, \tag{5.2}$$

where $I \in \Omega \subset \mathbb{R}^2$ is the set of points and $\mathcal{L}$ is the set of resolutions. By transforming the points of the same frame and not consider the other frame, the EPE avoids the problem that points that are only present in one frame due to the transformation.

5.4 Experiments

5.4.1 Setup

The framework is trained for a different amount of epochs depending on the residuals. More complex residuals are trained for more epochs. The simpler residual r_d is trained for 30 epochs, a more complex residual like r_{ssim} is trained for 50 epochs. The learning rate decays over time, with a decay ratio of 0.5. The decay rate depends on the number of epochs. The Glorot initialization [13] is used to initialize the weights uniformly following [19].

5.4.2 Different Residuals

	TUM RGB-D			
Δ	1	2	4	8
r_d	**0.403/0.666**	**0.562/1.113**	**0.960/2.199**	**3.016/5.188**
r_{gm}	0.421/0.738	0.611/1.304	1.223/2.713	3.980/ 6.549
r_{sgf1}	0.522/1.073	0.887/1.916	1.711/3.675	4.681/7.775
r_{sgf3}	0.545/1.015	0.898/1.921	1.781/3.802	4.536/7.990
r_{cossim}	1.065/1.164	1.720/2.308	3.129/4.598	6.099/9.164
r_{ssim}	2.037/1.266	2.684 /2.392	4.236/4.697	6.914/9.256

Table 5.1: Angular/Translational RPE evaluation on TUM RGB-D dataset, in [°]/[cm]

	ICL Pering			
Δ	1	2	4	8
r_d	**0.4178/1.901**	**0.761/3.832**	**1.687/8.344**	**4.534/19.171**
r_{gm}	0.830/3.541	1.508/7.078	2.736/14.158	5.706/28.337
r_{sgf1}	0.753/2.341	1.203/4.057	2.054/8.258	5.031/20.820
r_{sgf3}	0.751/3.529	1.231/7.062	2.308/14.129	5.860/28.203
r_{cossim}	1.661/5.013	3.120/9.042	5.968/17.191	11.337/31.535
r_{ssim}	2.571/8.353	3.640/11.404	6.046/17.239	11.320/31.034

Table 5.2: Angular/Translational RPE evaluation on ICL dataset, in [°]/[cm]

First, we want to compare the performance of the different residuals. Table 5.1 - 5.3 are showing the angular and the translation RPE on the different datasets.

	MovingObjects3D			
Δ	1	2	4	8
r_d	**3.705/9.944**	**7.485/20.128**	**14.715**/39.202	**27.132**/70.807
r_{gm}	3.995/10.572	8.003/21.149	15.451/40.729	27.989/72.785
r_{sgf1}	3.961/10.672	7.811/20.857	14.953/39.369	27.230/70.280
r_{sgf3}	3.917/10.128	7.846/20.255	15.074/**38.969**	27.279/**69.991**
r_{cossim}	4.682/11.081	8.631/21.315	15.908/40.322	28.093/71.569
r_{ssim}	4.356/12.080	8.148/21.926	15.390/40.966	27.780/72.832

Table 5.3: Angular/Translational RPE evaluation on MovingObjects3D dataset, in [°]/[cm]

	TUM RGB-D			
Δ	1	2	4	8
r_{sgf3} ”light”	0.552/1.052	**0.880**/1.921	**1.758/3.615**	**4.410/7.594**
r_{sgf3} ”full”	**0.545/1.015**	0.898/**1.916**	1.781/3.802	4.536/7.990
r_{ssim} ”light”	**1.914**/3.242	**2.658**/4.291	**4.192**/6.201	7.084/10.227
r_{ssim} ”full”	2.037/**1.266**	2.684/**2.392**	4.236/**4.697**	**6.914/9.256**

Table 5.4: Angular/Translational RPE evaluation on TUM RGB-D dataset with ”light” and ”full” method, in [°]/[cm]

In almost all the experiments the difference ”r_d” outperforms the other residuals. This does not confirm our hypothesis that more robust residuals improve the frameworks' performance. It is remarkable, that the more complex residuals often lead to worse results. One reason might be that the more complex residuals are more challenging to train. Furthermore, the photometric consistency may perform so well due to the Feature Extractor, which already points out the critical feature of the image. The learned representation may already have learned to deal with changing lighting. Hence, making more robust and complex residuals obsolete. In addition, the difference computation is much faster.

5.4.3 Influence of Jacobian Recomputation

As described in Section 4.2, we need to recompute the Jacobians for the more complex residuals. We test the influence of recomputing the residuals on r_{sgf3} and r_{ssim}. In the ”full” approach, we train the network with recomputing the Jacobians. In the ”light” approach, we ignore the error in the Jacobians in each iteration and keep the precomputed Jacobians. As one can see in Table 5.4, and

	ICL Pering			
Δ	1	2	4	8
r_{sgf3} ”light”	1.137/4.848	2.156/10.140	4.121/ 20.054	8.439/36.693
r_{sgf3} ”full”	**0.751/3.529**	**1.231/7.062**	**2.308/14.129**	**5.860/28.203**
r_{ssim} ”light”	5.973/19.039	6.267/19.826	8.141/25.387	12.723/36.983
r_{ssim} ”full”	**2.571/8.353**	**3.640/11.404**	**6.046/17.239**	**11.320/31.034**

Table 5.5: Angular/Translational RPE evaluation on ICL dataset with ”light” and ”full” method, in [°]/[cm]

Table 5.5, especially in the more challenging ICL dataset the advantage of the recomputing is visible. As expected, recomputation is more important for the complex Jacobians of $r_s sim$ and the more challenging dataset. For the more compact r_{sgf3} on TUM RGB-D the difference between the methods is neglectable. One reason for this could be the M-Estimator. This module should be able to mitigate small errors in the Jacobians.

5.5 LiDAR

To evaluate the framework with our adjustments for LiDAR, we compare against the generalized iterative closest point (GICP) algorithm [29]. The great advantage of GICP is that there is no need for training. It can be performed on arbitrary data. Often GICP relies on a good initialization. For testing, we compute the angular and translation RPE, which is described in Equation (5.1).
Our first experiment was on a small subset of the DRZ Living Lab dataset. We use 600 scans split into 510 for training, 50 for validation, and 40 for testing. The subset allows training the network fast and ensures correct functionality. Table 5.6 shows the results on this subset. Our approach is slightly better than GICP in almost every case. Next, we trained our network on six sequences of the Drz

	DRZ subset			
Δ	1	2	4	8
GICP	0.621/3.677	1.279/6.276	2.448/12.134	3.625/**15.617**
r_d	**0.453/3.037**	**0.867/5.304**	**1.765/11.285**	**2.765**/24.085

Table 5.6: Angular/Translational RPE evaluation on subset of DRZ Living Lab dataset, in [°]/[cm]

dataset. One sequence was used for validation. Then we compared our approach

with GICP on sequence ”12_18_25”, ”13_45_56”, ”14_16_34”, and ”17_29_29”.
As we can see in Table 5.7, GICP outperforms our approach on the DRZ Living

	DRZ			
Δ	1	2	4	8
GICP	**0.072/1.274**	**0.092/1.708**	**0.097/2.307**	**0.111/3.256**
r_d	0.646/5.776	1.025/8.372	1.783/16.495	3.129/42.538

Table 5.7: Angular/Translational RPE evaluation on DRZ Living Lab dataset, in [°]/[cm]

Lab dataset. Especially for higher Δ, the difference is visible.
We can see in Table 5.8 that our approach performs slightly better on the KITTI dataset. Both of the methods do not perform very well. The translation error is over one meter for two following frames for both methods. We have to investigate further, why this error is that high. We will neglect the results on the KITTI dataset in our evaluation.
Using CNNs with with inclompete data can be difficult. While our downsampling

	KITTI			
Δ	1	2	4	8
GICP	0.775/125.245	1.511/255.164	2.86/472.053	5.280/809.872
r_d	**0.697/102.680**	**1.160/106.426**	**2.246/140.084**	**4.306/324.650**

Table 5.8: Ang/Transl RPE on KITTI test data

approach fills most holes in the LRIs, a few remain. A further step to improve our approach could be introducing an impainting module wich takes care of incomplete data.

6 Conclusion

In this thesis, we researched two hypotheses. First, we tried to improve the Deep Inverse Compositional algorithm proposed by Lv et al. [19]. They used three learning modules, the Two-View-Feature Encoder, the Convolutional M-Estimator, and the Trust-Region Network, and integrated them into the Inverse Compositional Algorithm. Lv et al. use the photometric error to compare the transformed first image with the second image within an iterative optimization step. We used the residual functions Cosine Similarity, Structural Similarity index, gradient magnitude, r_{sgf}, and r_{sgf3}. The idea was that the newly introduced metrics give a better representation of the errors in the residuals in terms of robustness. We were not able to confirm this in our experiments on the datasets TUM RGB-D, MovingObjects3D, ICL Pering. Furthermore, it seemed that the more complex metrics perform worse. As an explanation, we listed the Two-View Feature Encoder. The Two-View feature encoder already deals with problems that were addressed by the new residuals. In addition to this, with the new residual functions, the framework might be harder to train due to more complexity.
The new residuals result in a recomputation step of the Jacobians in the iterative part of the algorithm. We tested the influence of recomputation by training the network without recomputation and with recomputation and then compare these two methods. We observed that the recomputation is more important for complex residuals and complex scenes. For more simple residuals and scenes, the two methods achieved similar results.
In the second part of this thesis, we adapted the Deep Inverse Compositional algorithm for LiDAR data. We used Lidar Range images as input for the algorithm to use well-established operations like the convolution in our learning modules. We adapted the computation of the Jacobians to fit LRIs. We were able to propose a new method, build on the work of Lv et al., to align point clouds to each other. In our experiments, we compared our approach against the standard point cloud alignment algorithm generalized iterative closest point (GICP). Our approach performs very well in our first experiment on a subsampled dataset. The following experiment does not confirm this. On the DRZ Living Lab dataset GICP performs very well. Our approach is not able to achieve similar results. One reason for the better results on the subset of DRZ could be that the learned modules in

our method overfitted on the small subset. In addition to this, on average GICP performs much better in the following experiment on the DRZ dataset. The errors on the KITTI dataset were very high, therefore we do not consider the results in our evaluation.
Further steps to improve our methods could be introducing an inpainting module to ensure complete data for our learning modules. In addition, the error on the KITTI dataset needs to be further investigated.

List of Figures

List of Tables

Bibliography

[1] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. “Understanding of a convolutional neural network.” In: *2017 international conference on engineering and technology (icet)*. 2017, pp. 1–6.

[2] S. Baker and I. Matthews. “Lucas-kanade 20 years on: a unifying framework.” In: *International journal of computer vision* 56 (2004), pp. 221–225.

[3] José Luis Blanco-Claraco. “A tutorial on SE(3)transformation parameterizations and on-manifold optimization.” In: *Corr* abs/2103.15980 (2021). arXiv: `2103.15980`. URL: `https://arxiv.org/abs/2103.15980`.

[4] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. “Brief: binary robust independent elementary features.” In: vol. 6314. Sept. 2010, pp. 778–792. ISBN: 978-3-642-15560-4.

[5] Angel X. Chang, Thomas Funkhouser, Leonidas Guibas, Pat Hanrahan, Qixing Huang, Zimo Li, Silvio Savarese, Manolis Savva, Shuran Song, Hao Su, Jianxiong Xiao, Li Yi, and Fisher Yu. *Shapenet: an information-rich 3d model repository.* 2015. arXiv: `1512.03012 [cs.GR]`.

[6] Jan Czarnowski, Stefan Leutenegger, and Andrew Davison. *Semantic texture for robust dense tracking.* 2017. arXiv: `1708.08844 [cs.CV]`.

[7] Juan Du, Rui Wang, and Daniel Cremers. “DH3D: deep hierarchical 3d descriptors for robust large-scale 6dof relocalization.” In: *Corr* abs/2007.09217 (2020). arXiv: `2007.09217`. URL: `https://arxiv.org/abs/2007.09217`.

[8] Mihai Dusmanu, Ondrej Miksik, Johannes L. Schönberger, and Marc Pollefeys. “Cross-descriptor visual localization and mapping.” In: *Corr* abs/2012.01377 (2020). arXiv: `2012.01377`. URL: `https://arxiv.org/abs/2012.01377`.

[9] Jakob Engel, Thomas Schöps, and Daniel Cremers. “Lsd-slam: large-scale direct monocular slam.” In: *Computer vision – eccv 2014*. Ed. by David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars. Cham: Springer International Publishing, 2014, pp. 834–849. ISBN: 978-3-319-10605-2.

[10] *Fully connected layer.* URL: `https://fastaireference.com/tabular-data/fully-connected-layer`.

[11] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. “Vision meets robotics: the kitti dataset.” In: *International journal of robotics research (ijrr)* (2013).

[12] N. Gelfand, Niloy Mitra, Leonidas Guibas, and Helmut Pottmann. "Robust global registration." In: Jan. 2005.

[13] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." In: *Aistats*. 2010.

[14] Sergey Ioffe and Christian Szegedy. *Batch normalization: accelerating deep network training by reducing internal covariate shift*. 2015. arXiv: `1502.03167 [cs.LG]`.

[15] Carlos Jaramillo, Yuichi Taguchi, and Chen Feng. "Direct multichannel tracking." In: *International conference on 3d vision*. Oct. 2017. URL: `https://www.merl.com/publications/TR2017-146`.

[16] C. Kerl, Jürgen Sturm, and D. Cremers. "Robust odometry estimation for rgb-d cameras." In: *2013 ieee international conference on robotics and automation* (2013), pp. 3748–3754.

[17] A. R. Lahitani, A. E. Permanasari, and N. A. Setiawan. "Cosine similarity to determine similarity measure: study case in online essay assessment." In: *2016 4th international conference on cyber and it service management*. 2016, pp. 1–6.

[18] David Lowe. "Distinctive image features from scale-invariant keypoints." In: *International journal of computer vision* 60 (Nov. 2004), pp. 91–.

[19] Zhaoyang Lv, Frank Dellaert, James M. Rehg, and Andreas Geiger. "Taking a deeper look at the inverse compositional algorithm." In: *Corr* abs/1812.06861 (2018). arXiv: `1812.06861`. URL: `http://arxiv.org/abs/1812.06861`.

[20] Seonwook Park, Thomas Schoeps, and Marc Pollefeys. "Illumination change robustness in direct visual slam." In: May 2017, pp. 4523–4530.

[21] Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. "Pointnet: deep learning on point sets for 3d classification and segmentation." In: *Corr* abs/1612.00593 (2016). arXiv: `1612.00593`. URL: `http://arxiv.org/abs/1612.00593`.

[22] Jan Quenzel and Sven Behnke. "Real-time multi-adaptive-resolution-surfel 6d lidar odometry using continuous-time trajectory optimization." In: *Corr* abs/2105.02010 (2021). arXiv: `2105.02010`. URL: `https://arxiv.org/abs/2105.02010`.

[23] Jan Quenzel, Radu Alexandru Rosu, Thomas Läbe, Cyrill Stachniss, and Sven Behnke. "Beyond photometric consistency: gradient-based dissimilarity for improving visual odometry and stereo matching." In: (2020). arXiv: `2004.04090 [cs.CV]`.

[24] Mike Roberts and Nathan Paczan. *Hypersim: a photorealistic synthetic dataset for holistic indoor scene understanding*. 2020. arXiv: `2011.02523 [cs.CV]`.

[25] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. "Fast point feature histograms (fpfh) for 3d registration." In: *2009 ieee international conference on robotics and automation*. 2009, pp. 3212–3217.

[26] Radu Bogdan Rusu, Nico Blodow, Zoltan Csaba Marton, and Michael Beetz. "Aligning point cloud views using persistent feature histograms." In: *2008 ieee/rsj international conference on intelligent robots and systems*. 2008, pp. 3384–3391.

[27] et al. Sajad Saeedi. "Characterizing visual localization and mapping datasets." In: *International conference on robotics and automation (icra)*. IEEE. 2019, pp. 6699–6705.

[28] Daniel Schleich, Marius Beul, Jan Quenzel, and Sven Behnke. "Autonomous flight in unknown gnss-denied environments for disaster examination." In: *Corr* abs/2103.11742 (2021). arXiv: `2103.11742`. URL: `https://arxiv.org/abs/2103.11742`.

[29] Aleksandr Segal and Dirk Hähnel. "Generalized-icp." In: June 2009.

[30] Lukas von Stumberg, Patrick Wenzel, Qadeer Khan, and Daniel Cremers. "Gn-net: the gauss-newton loss for deep direct SLAM." In: *Corr* abs/1904.11932 (2019). arXiv: `1904.11932`. URL: `http://arxiv.org/abs/1904.11932`.

[31] Lukas von Stumberg, Patrick Wenzel, Nan Yang, and Daniel Cremers. "Lm-reloc: levenberg-marquardt based direct visual relocalization." In: *Corr* abs/2010.06323 (2020). arXiv: `2010.06323`. URL: `https://arxiv.org/abs/2010.06323`.

[32] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. "A benchmark for the evaluation of rgb-d slam systems." In: *Proc. of the international conference on intelligent robot systems (iros)*. Oct. 2012.

[33] Zachary Taylor, Juan Nieto, and David Johnson. "Multi-modal sensor calibration using a gradient orientation measure." In: *Journal of field robotics* 32.5 (2015), pp. 675–695. eprint: `https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21523`. URL: `https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21523`.

[34] Zhou Wang, Alan Bovik, Hamid Sheikh, and Eero Simoncelli. "Image quality assessment: from error visibility to structural similarity." In: *Ieee transactions on image processing* 13 (Sept. 2004).

[35] Binbin Xu, Andrew J. Davison, and Stefan Leutenegger. *Deep probabilistic feature-metric tracking*. 2020. arXiv: `2008.13504 [cs.CV]`.

[36] Fisher Yu and Vladlen Koltun. *Multi-scale context aggregation by dilated convolutions*. 2016. arXiv: `1511.07122 [cs.CV]`.

[37] Jiahui Yu, Zhe Lin, Jimei Yang, Xiaohui Shen, Xin Lu, and Thomas Huang. *Free-form image inpainting with gated convolution*. 2019. arXiv: `1806.03589 [cs.CV]`.

[38] Hang Zhao, Orazio Gallo, Iuri Frosio, and Jan Kautz. “Loss functions for neural networks for image processing.” In: *Corr* abs/1511.08861 (2015). arXiv: `1511.08861`. URL: `http://arxiv.org/abs/1511.08861`.