State of the Art in Real-time Registration of RGB-D Images

Patrick Stotko*
Supervised by: Tim Golla†

Institute of Computer Science II - Computer Graphics
University of Bonn
Bonn / Germany

Abstract

3D reconstruction gained more and more interest in the last years due to low-budget depth scanning devices and high-performance graphics hardware. Several approaches were developed in order to get accurate reconstructions in real-time at high frame rates and opened up new applications in the field of augmented reality (AR) and 3D scanning by providing immediate feedback to the user. But since measurements are corrupted with noise, registration of the captured RGB-D images needs to be very accurate to get high quality results. In this state of the art report, we will review several registration algorithms that were developed in the last years and compare their performance.

Keywords: State of the Art, Real-time, Registration, RGB-D, GPU, Kinect, 3D Reconstruction, Comparison

1 Introduction

During the last years, increasing effort was spent in the field of 3D reconstruction using depth cameras. The origin of this rising interest lies in the availability of low-budget, high-resolution depth sensors like the Microsoft Kinect. One of the first and most prominent systems in this field was KinectFusion, a 3D reconstruction framework developed by Izadi et al., (2011) and Newcombe et al., (2011). This system is capable of producing highly detailed 3D models from RGB-D images in real-time by integrating the captured data into the volumetric data structure of Curless and Levoy, (1996) in parallel on the GPU.

However, it has several limitations. In KinectFusion, Izadi et al., (2011) and Newcombe et al., (2011) used a full 3D volumetric grid to store the reconstruction. Since GPU memory is quite limited, creating large scale reconstructions was not possible. To overcome this issue, several techniques to reduce memory consumption were developed including Moving Volume approaches (Roth and Vona, 2012; Whelan et al., 2012), fast data structures (Laine and Karras, 2010; Zeng et al., 2012; Zeng et al., 2013) and streaming algorithms (Chen et al., 2013). Recent work done by Nießner et al., (2013) came to the result that by using hash tables only relevant regions of the volume needed to be stored leading to great advances in terms of performance and efficiency.

The other major drawback of the initial KinectFusion system is its registration algorithm. Before the captured data can be integrated into the volume, it must be transformed into the global coordinates system in which the volume is specified. Therefore, an estimate of the six-dimensional camera pose is needed. Unfortunately, even small errors in the estimation cause significant drift in the final reconstruction since those errors accumulate rapidly. The origin of this problem lies in the uncertainty of the data that is captured by the sensor. Usually, these data contain noise whose exact nature is still unknown. So a perfect registration is not possible and much effort has been spent to obtain good reconstructions in the end.

The purpose of this state of the art report is to review current approaches that tried to solve this problem. We also investigate how well the solutions fit into our reconstruction pipeline (Stotko and Golla, 2015) that is based on the VoxelHashing framework of Nießner et al., (2013). In particular, we will focus on:

1. Reviewing several approaches and explaining their strengths and weaknesses
2. Analyzing how efficient they can be implemented on the GPU
3. Comparing the different algorithms in terms of accuracy and efficiency

2 Problem Statement

Before we start to present and compare the different algorithms, we first need to formally state the problem that should be solved.

2.1 Preliminaries

In our settings, we want to reconstruct 3D geometry automatically and in real-time using a low-cost depth sensor. Such sensors have been becoming popular in the past years because they offer moderate reconstruction quality

*stotko@cs.uni-bonn.de
†golla@cs.uni-bonn.de

Figure 1: Example of an RGB image (left) and a depth image (right) captured by a RGB-D camera, taken from Henry et al., (2012).

at a cheap price. The probably most prominent one is the Kinect from Microsoft. Besides a depth image $\boldsymbol{I}_d$, it also captures an RGB image $\boldsymbol{I}_{RGB}$ at a particular resolution, e.g. 640×480 in case of the Kinect:

$$\boldsymbol{I}_{RGB} \colon \Omega \subset \mathbb{N}^2 \to \mathbb{N}^3 \tag{1}$$

$$\boldsymbol{I}_d \colon \Omega \subset \mathbb{N}^2 \to \mathbb{R} \tag{2}$$

We require these two maps to be time-synchronized and aligned. Since the cameras have a small offset, some sensor-specific transformations for aligning them may be performed in a preprocessing step. Figure 1 shows an example of such an RGB-D image. The 3D position $\boldsymbol{v} \in \mathbb{R}^3$ of a pixel $\boldsymbol{p} \in \Omega$ can then be computed by back-projection as

$$\boldsymbol{v}(\boldsymbol{p}) = \left(\frac{p_x - c_x}{f_x} \boldsymbol{I}_d(\boldsymbol{p}), \frac{p_y - c_y}{f_y} \boldsymbol{I}_d(\boldsymbol{p}), \boldsymbol{I}_d(\boldsymbol{p}) \right)^\top \tag{3}$$

using the focal length $\boldsymbol{f} = (f_x, f_y)$ and the principal point $\boldsymbol{c} = (c_x, c_y)$ of the camera. On the other hand, the pixel $\boldsymbol{p}$ into which the point $\boldsymbol{v}$ falls is computed as the projection

$$\boldsymbol{p}(\boldsymbol{v}) = \left(\frac{f_x v_x}{v_z} + c_x, \frac{f_y v_y}{v_z} + c_y \right)^\top \tag{4}$$

These two transformations can also be written compactly by considering the intrinsic camera calibration matrix of the sensor

$$\boldsymbol{K} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \tag{5}$$

and the projection operator

$$\boldsymbol{\Pi}\left((x, y, z)^\top \right) = (x/z, y/z)^\top \tag{6}$$

Then, we can express the two projections by

$$\boldsymbol{v}(\boldsymbol{p}) = \boldsymbol{K}^{-1} \cdot \boldsymbol{I}_d(\boldsymbol{p}) \cdot (\boldsymbol{p}, 1)^\top \tag{7}$$

$$\boldsymbol{p}(\boldsymbol{v}) = \boldsymbol{\Pi}(\boldsymbol{K} \cdot \boldsymbol{v}) \tag{8}$$

2.2 The Registration Problem

Our reconstruction pipeline is based on the system developed by Nießner et al., (2013). Like in their work, we use an efficient hash table on top of a set of voxel blocks to reconstruct the captured scene in a sparse volumetric grid. To perform registration, we use ray-casting to extract the implicitly stored isosurface from the volume and use the resulting RGB-D image as an estimate of the model. This image can further be processed for rendering by some shading kernels.

Now, the only missing part is the registration algorithm. Given new measurements $\left[\boldsymbol{I}_{RGB}^{(t+1)}, \boldsymbol{I}_d^{(t+1)} \right]$ at time step $t+1$, we wish to find the new camera transformation matrix

$$\boldsymbol{T}^{(t+1)} = \begin{bmatrix} \boldsymbol{R}^{(t+1)} & \boldsymbol{t}^{(t+1)} \\ \boldsymbol{0}^\top & 1 \end{bmatrix} \in \mathbb{R}^{4 \times 4} \tag{9}$$

consisting of a rotation $\boldsymbol{R}^{(t+1)} \in \mathbb{R}^{3 \times 3}$ and a translation $\boldsymbol{t}^{(t+1)} \in \mathbb{R}^3$. Since we also know this transformation from the previous time step t, namely the current model $\left[\hat{\boldsymbol{I}}_{RGB}^{(t)}, \hat{\boldsymbol{I}}_d^{(t)} \right]$, this problem can be reduced. We only need to align the two images to get our desired result. A successful alignment will give us the incremental transformation $\boldsymbol{\Delta T}$. This optimization might be performed in global coordinates which is usually done when dealing with points clouds. But since we are trying to align measurements from a camera, the local camera coordinates system at time step t is the better choice and also used in some of the approaches. Notice that the local system at time step $t+1$ is moving and thus not very appropriate as a reference. In this context, we have the following relation:

$$\boldsymbol{T}^{(t+1)} = \boldsymbol{T}^{(t)} \cdot \boldsymbol{\Delta T} \tag{10}$$

So we first move the new measurements by $\boldsymbol{\Delta T}$ such that they are aligned with the previous data in their respective coordinate system and then transform them to global coordinates using $\boldsymbol{T}^{(t)}$. Defining the transformation in this

way, we get an intuitive meaning of the incremental transformation by rearranging the equation:

$$\Delta \boldsymbol{T} = \left(\boldsymbol{T}^{(t)}\right)^{-1} \cdot \boldsymbol{T}^{(t+1)} \tag{11}$$

For any newly measured point given in the local camera coordinate system at time $t+1$, we can infer its coordinates in the local system at time t by applying the incremental transformation.

In the following, we will discuss three families of registration algorithms: Iterative Closest Point, Normal Distribution Transform and Sparse Methods.

3 Iterative Closest Point

We start our discussing with some popular approaches from the family of the Iterative Closest Point algorithms.

3.1 KinectFusion ICP

The original registration algorithm used in KinectFusion is a variant of the popular Iterative Closest Point (ICP) algorithm (Besl and McKay, 1992; Chen and Medioni, 1992). In our setting, the vertex maps obtained by back-projection of the depth maps describe the scene in the local camera coordinate system at time t:

$$\hat{\boldsymbol{v}}_i^{(t)} := \hat{\boldsymbol{v}}^{(t)}(\boldsymbol{p}_i) \tag{12}$$

$$\boldsymbol{v}_j^{(t+1)} := \Delta \boldsymbol{T} \cdot \boldsymbol{v}^{(t+1)}(\boldsymbol{p}_j) \tag{13}$$

For simplicity, we omit the conversion to homogeneous coordinates.

In KinectFusion, the newly captured RGB-D image at time step $t+1$ is now aligned to the current model at time step t. It is rather important to distinguish between frame-to-frame and frame-to-model tracking. The latter is much more robust against noise since the model is very smooth and contains much less noise than an input image. Therefore, frame-to-model tracking is used in the ICP error function:

$$E_{icp} = \sum_i \sum_j w_{ij} \left\langle \boldsymbol{T} \cdot \boldsymbol{v}_j^{(t+1)} - \hat{\boldsymbol{v}}_i^{(t)} \,\middle|\, \hat{\boldsymbol{n}}_i^{(t)} \right\rangle^2 \tag{14}$$

This formulation is the general point-to-plane error function which describes the quadratic distance of the point $\boldsymbol{T} \cdot \boldsymbol{v}_j^{(t+1)}$ to the tangent plane of $\hat{\boldsymbol{v}}_i^{(t)}$ that is defined by its normal $\hat{\boldsymbol{n}}_i^{(t)}$. The coefficients w_{ij} describe the level of correspondence between each pair. To find the global minimum of this error function, the weights w_{ij} and the transformation $\boldsymbol{T}$ need to be optimized jointly. Unfortunately this is an NP-hard problem, so in the ICP algorithm the weights and the transformation are optimized separately. The cost of this simplification is the loss of the global optimum, only a local one can be guaranteed.

3.1.1 Finding Correspondences

Optimizing the weights w_{ij} is further simplified using the heuristic ICP performs. It is assumed that there exists a set of correspondence pairs between the two points sets. From this assumption, it follows that the weights are binary and the error function can be simplified by using only one sum over the correspondence set:

$$E_{icp} = \sum_i \left\langle \boldsymbol{T} \cdot \boldsymbol{v}_i^{(t+1)} - \hat{\boldsymbol{v}}_{c(i)}^{(t)} \,\middle|\, \hat{\boldsymbol{n}}_{c(i)}^{(t)} \right\rangle^2 \tag{15}$$

This has the great advantage that the complexity is drastically reduced of being only linear instead of quadratic.

The initial objective was to optimize the general error function with respect to the weights. For our simplified formulation, this problem has an intuitive solution. Each term of the error is minimal if we choose for each point $\boldsymbol{v}_i^{(t+1)}$ the closest point $\hat{\boldsymbol{v}}_{c(i)}^{(t)}$ as the corresponding point.

KinectFusion further exploits the fact that these points are generated from a camera with known intrinsic parameters. Computing the correspondences is therefore done by the projective data association algorithm (Blais and Levine, 1995). Here, the definition of closest point is not related to being close in the three-dimensional space. Instead, for each vertex $\boldsymbol{v}^{(t+1)}(\boldsymbol{p}_i)$, which is defined in the local coordinate system at time $t+1$, we compute the vertex $\hat{\boldsymbol{v}}_{c(i)}^{(t)}$ at time t that is at the same line of sight in the local system at time t. This means, we first need to know the coordinates of vertex $\boldsymbol{v}^{(t+1)}(\boldsymbol{p}_i)$ in the other coordinate system by applying $\Delta \boldsymbol{T}$, then project it to pixel coordinates and use this pixel $\boldsymbol{p}_{c(i)}$ as a look up for the corresponding vertex:

$$\hat{\boldsymbol{v}}_{c(i)}^{(t)} = \boldsymbol{v}^{(t)}(\boldsymbol{p}(\Delta \boldsymbol{T} \cdot \boldsymbol{v}^{(t+1)}(\boldsymbol{p}_i))) \tag{16}$$

An example of this procedure is shown in Figure 2. If the two point sets are already perfectly aligned, then the

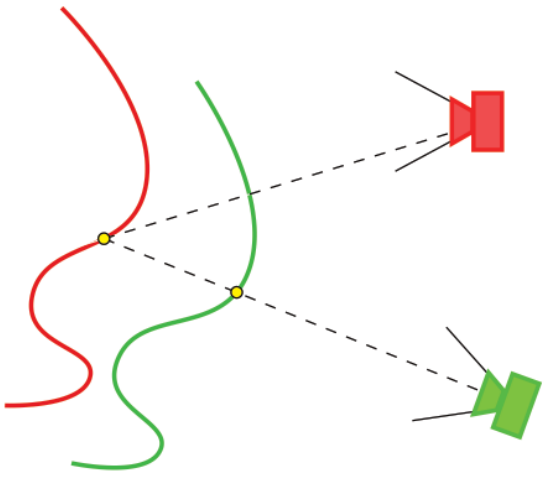


Figure 2: The Projective Data Association Operator. Correspondence pairs are found by projection to the previous camera coordinate system.

correspondence coincides with the input vertex and thus it is also the closest point in space.

3.1.2 Optimizing the error function

After having calculated the correspondences, the actual error function can be optimized. So we wish to find a rigid transformation $\boldsymbol{T}$ that minimizes the error between the correspondences. However, the rotation $\boldsymbol{R}$ of $\boldsymbol{T}$ has three degrees of freedom but nine unknown values to optimize for. In order to have a minimal representation, the following one is commonly used:

$$\boldsymbol{\xi} = \left(\boldsymbol{\omega}^\top, \boldsymbol{t}^\top\right)^\top \in \mathbb{R}^6 \tag{17}$$

This vector can be transformed to the original matrix by

$$\boldsymbol{T} = \tilde{\boldsymbol{\xi}} \quad , \quad \tilde{\boldsymbol{\xi}} = \begin{bmatrix} \exp[\boldsymbol{\omega}]_\times & \boldsymbol{t} \\ \boldsymbol{0}^\top & 0 \end{bmatrix} \tag{18}$$

For consistency with further approaches, we only consider the exponential mapping of the rotation and directly model the translation. This will help us to compare the approaches and will not affect the accuracy.

Since only small rotations are assumed, the exponential mapping can be approximated linearly:

$$\exp[\boldsymbol{\omega}]_\times = \sum_{n=0}^{\infty} \frac{([\boldsymbol{\omega}]_\times)^n}{n!} \approx \boldsymbol{I} + [\boldsymbol{\omega}]_\times \tag{19}$$

By plugging this approximation into the error function and rearranging the terms, one gets similarly to the derivation of Low, (2004) the following result:

$$E_{icp} = \sum_i \left(\begin{bmatrix} \boldsymbol{v}_i^{(t+1)} \times \hat{\boldsymbol{n}}_{c(i)}^{(t)} \\ \hat{\boldsymbol{n}}_{c(i)}^{(t)} \end{bmatrix}^\top \boldsymbol{\xi} + \left\langle \boldsymbol{v}_i^{(t+1)} - \hat{\boldsymbol{v}}_{c(i)}^{(t)} \,\middle|\, \hat{\boldsymbol{n}}_{c(i)}^{(t)} \right\rangle \right)^2 \tag{20}$$

$$= \left\| \boldsymbol{J}_{icp}\, \boldsymbol{\xi} + \boldsymbol{r}_{icp} \right\|_2^2 \tag{21}$$

This is a standard equation system that can be solved by the normal equation. The main advantage of such an equation system here is the independence between the pairs of vertices. Each row of the Jacobian and the residual can thus be calculated separately in parallel which makes it very suitable for a GPU implementation. In such an implementation, one would directly compute the components of the normal equation:

$$\boldsymbol{J}_{icp}^\top \boldsymbol{J}_{icp} = \sum_i \boldsymbol{j}_i^\top \cdot \boldsymbol{j}_i \quad , \quad \boldsymbol{J}_{icp}^\top \boldsymbol{r}_{icp} = \sum_i \boldsymbol{j}_i^\top \cdot r_i \tag{22}$$

$$\boldsymbol{j}_i = \begin{bmatrix} \boldsymbol{v}_i^{(t+1)} \times \hat{\boldsymbol{n}}_{c(i)}^{(t)} \\ \hat{\boldsymbol{n}}_{c(i)}^{(t)} \end{bmatrix}^\top \quad , \quad r_i = \left\langle \boldsymbol{v}_i^{(t+1)} - \hat{\boldsymbol{v}}_{c(i)}^{(t)} \,\middle|\, \hat{\boldsymbol{n}}_{c(i)}^{(t)} \right\rangle \tag{23}$$

Each summand is thus be computed independently in parallel and the total sum is then be obtained by a tree reduction. Since the memory cost is only linear in the number of correspondences, the total memory consumption is manageable. This property is very important since GPU's have only a very limited amount of memory compared to CPU's. Minimizing the footprint of such an algorithm is therefore a very important task and is also indirectly a criterion for being able to run in real-time.

In order to prepare for the next steps, we multiply both the Jacobian and residual by -1. This does not change the result but results in a nice notation:

$$\boldsymbol{j}_i = -\begin{bmatrix} \boldsymbol{v}_i^{(t+1)} \times \hat{\boldsymbol{n}}_{c(i)}^{(t)} \\ \hat{\boldsymbol{n}}_{c(i)}^{(t)} \end{bmatrix}^\top = \left(-\hat{\boldsymbol{n}}_{c(i)}^{(t)}\right)^\top \cdot \left[[-\boldsymbol{v}_i^{(t+1)}]_\times \quad \boldsymbol{I} \right] \tag{24}$$

$$r_i = -\left\langle \boldsymbol{v}_i^{(t+1)} - \hat{\boldsymbol{v}}_{c(i)}^{(t)} \,\middle|\, \hat{\boldsymbol{n}}_{c(i)}^{(t)} \right\rangle = \left\langle \hat{\boldsymbol{v}}_{c(i)}^{(t)} - \boldsymbol{v}_i^{(t+1)} \,\middle|\, \hat{\boldsymbol{n}}_{c(i)}^{(t)} \right\rangle \tag{25}$$

Later we will come back to this point and show what this reformulation actually means how it can be used.

The final step is to solve the equation system. Both components of the normal equation are only six-dimensional so this step is performed very efficiently on the CPU by a Cholesky decomposition. Finally, the estimated parameters $\boldsymbol{\xi}$ can be used to update the incremental transformation:

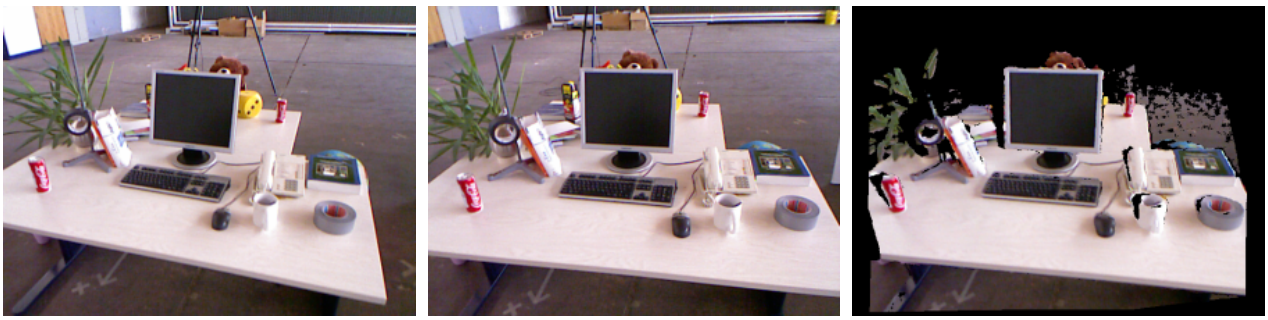
$$\boldsymbol{\Delta T} \leftarrow \begin{bmatrix} \exp[\boldsymbol{\omega}]_\times & \boldsymbol{t} \\ \boldsymbol{0}^\top & 0 \end{bmatrix} \cdot \boldsymbol{\Delta T} \tag{26}$$

As mentioned at the beginning, these steps must be performed iteratively until either the error falls below a threshold or a maximum number of iterations is reached. To improve the result, typically a three-level coarse-to-fine scheme is used where the images are step-wise filtered and down-sampled to lower resolutions. During the optimization, one starts at the coarsest level and then refines the solution until reaching the finest level. This helps to avoid local minima and guides the optimization towards the global optimum. However, there is still no guarantee to reach it.

3.2 Photometric ICP

The KinectFusion system was tested in several scenarios and while the accuracy is quite high in smaller scenes, it drops significantly on larger ones. Especially if the amount of characteristic features is low, e.g. on walls, only using the depth information is not sufficient to compute the camera pose. Steinbrücker et al., (2011) and Kerl et al., (2013b) used the color information to overcome this.

The main idea here is that if the camera has only moved a little bit like it is shown in Figure 3, the appearance of scene is the same on both images. Physically, this means the color of an object is the same and therefore its reflectance behavior does not change. This results in the assumption the whole scene mainly reflects perfectly diffuse and that specular parts can be neglected. For mirror-like objects, this assumption would of course fail, but usually walls, tables and most other objects can be approximately modeled



(a) First input image (b) Second input image (c) Warped first image

Figure 3: Example of warping, taken from Steinbrücker et al., (2011).

in this way. So based on this assumption, the so called photometric error can be formulated as

$$E_{rgbd} = \sum_i \left(\boldsymbol{I}_g^{(t)}(\boldsymbol{w}(\boldsymbol{\xi}, \boldsymbol{p}_i)) - \boldsymbol{I}_g^{(t+1)}(\boldsymbol{p}_i) \right)^2 \tag{27}$$

where $\boldsymbol{I}_g^{(t)}, \boldsymbol{I}_g^{(t+1)}$ are the intensity images computed from the RGB images at time steps t and $t+1$. Special care about the choice of the coordinate system has to be taken here. As we will see later, we need to use the same system as in the KinectFusion ICP. So unlike in the original paper, we use this formulation to describe the error. The vector $\boldsymbol{w}(\boldsymbol{\xi}, \boldsymbol{p}_i)$ is the warped pixel and defined according to the incremental transformation $\boldsymbol{\xi}$:

$$\boldsymbol{w}(\boldsymbol{\xi}, \boldsymbol{p}_i) = \boldsymbol{\Pi}(\boldsymbol{K} \cdot \tilde{\boldsymbol{\xi}} \cdot \boldsymbol{\Delta T} \cdot \boldsymbol{v}^{(t+1)}(\boldsymbol{p}_i)) \tag{28}$$

Intuitively, the warp takes a vertex from time step $t+1$ and transforms it into the local camera coordinate system at time t. Then, it is moved according to the optimization parameters and finally projected to pixel coordinates. A correct alignment would move the vertex in such a way that after projection the intensity value at the corresponding pixel is the same as the one of the starting pixel.

Figures 3 and 4 show an example of this idea. Note that the RGB image of the model is not used here since it is not as precise as the captured one and therefore frame-to-frame tracking is performed for the color values. To prepare for the next step, the warped point is written as a mapping:

$$\boldsymbol{v}(\boldsymbol{\xi}, \boldsymbol{p}_i) = \tilde{\boldsymbol{\xi}} \cdot \left(\boldsymbol{\Delta T} \cdot \boldsymbol{v}^{(t+1)}(\boldsymbol{p}_i) \right) = \tilde{\boldsymbol{\xi}} \cdot \boldsymbol{v}_i^{(t+1)} \tag{29}$$

One important note has to be mentioned at this point. $\boldsymbol{v}_i^{(t+1)}$ is the same point as in the KinectFusion ICP error and the warping function w is essentially the mapping used to compute the correspondences. Knowing this, the Photometric ICP is in fact an ICP that operates on intensity images. The error can now be rewritten as

$$E_{rgbd} = \sum_i (d(\boldsymbol{\xi}, \boldsymbol{p}_i))^2 \tag{30}$$

$$d(\boldsymbol{\xi}, \boldsymbol{p}_i) = \boldsymbol{I}_g^{(t)}(\boldsymbol{\Pi}(\boldsymbol{K} \cdot \boldsymbol{v}(\boldsymbol{\xi}, \boldsymbol{p}_i)) - \boldsymbol{I}_g^{(t+1)}(\boldsymbol{p}_i) \tag{31}$$

Like in KinectFusion, one wants to apply the Gauss-Newton method to solve the problem iteratively. This requires that the error function needs to be linearized which is done by a Taylor series of the parameters $\boldsymbol{\xi}$:

$$d(\boldsymbol{\xi}, \boldsymbol{p}_i) \approx d(\boldsymbol{0}, \boldsymbol{p}_i) + \frac{\partial d}{\partial \boldsymbol{\xi}}(\boldsymbol{0}, \boldsymbol{p}_i) \cdot \boldsymbol{\xi} \tag{32}$$

$$= \left(\boldsymbol{I}_g^{(t)}(\boldsymbol{\Pi}(\boldsymbol{K} \cdot \boldsymbol{v}_i^{(t+1)}) - \boldsymbol{I}_g^{(t+1)}(\boldsymbol{p}_i) \right) + \frac{\partial d}{\partial \boldsymbol{\xi}}(\boldsymbol{0}, \boldsymbol{p}_i) \cdot \boldsymbol{\xi} \tag{33}$$

The first term is the current residual r_i and states the difference in the intensity values under the current alignment. The other term is the derivative of the energy with respect to $\boldsymbol{\xi}$, an 1×6 matrix and the i-th Jacobian $\boldsymbol{j}_i$. By using the chain rule the Jacobian can be is given as

$$\frac{\partial d}{\partial \boldsymbol{\xi}}(\boldsymbol{0}, \boldsymbol{p}_i) = \frac{\partial \boldsymbol{I}_g^{(t)}}{\partial \boldsymbol{\Pi}} \cdot \frac{\partial \boldsymbol{\Pi}}{\partial \boldsymbol{K}} \cdot \frac{\partial \boldsymbol{K}}{\partial \boldsymbol{v}} \cdot \frac{\partial \boldsymbol{v}}{\partial \boldsymbol{\xi}}(\boldsymbol{0}, \boldsymbol{p}_i) \tag{34}$$

$$= \nabla \boldsymbol{I}_g^{(t)} \cdot \frac{\partial \boldsymbol{\Pi}}{\partial (x, y, z)^\top} \cdot \boldsymbol{K} \cdot \frac{\partial \boldsymbol{v}}{\partial \boldsymbol{\xi}}(\boldsymbol{0}, \boldsymbol{p}_i) \tag{35}$$

Thus, one needs to compute the image gradient

$$\nabla \boldsymbol{I}_g^{(t)} = \left(\frac{\partial \boldsymbol{I}_g^{(t)}}{\partial x}, \frac{\partial \boldsymbol{I}_g^{(t)}}{\partial y} \right) \in \mathbb{R}^{1 \times 2} \tag{36}$$

the derivative of the dehomogenization

$$\frac{\partial \boldsymbol{\Pi}}{\partial (x, y, z)^\top} = \begin{pmatrix} \frac{1}{z} & 0 & -\frac{x}{z^2} \\ 0 & \frac{1}{z} & -\frac{y}{z^2} \end{pmatrix} \in \mathbb{R}^{2 \times 3} \tag{37}$$

and the derivative of the moving point $\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{\xi}} \in \mathbb{R}^{3 \times 6}$ we are optimizing for. We found that there exists a compact formula for computing the last derivative. Gallego and Yezzi, (2015) showed a nice derivation of this formula and also considered the important case with $\boldsymbol{\xi} = \boldsymbol{0}$:

$$\frac{\partial \boldsymbol{v}}{\partial \omega_i}(\boldsymbol{0}, \boldsymbol{p}_i) = \frac{\partial \tilde{\boldsymbol{\xi}}}{\partial \omega_i}(\boldsymbol{0}) \cdot \boldsymbol{v}_i^{(t+1)} \tag{38}$$

$$= \frac{\partial \exp[\boldsymbol{\omega}]_\times}{\partial \omega_i}(\boldsymbol{0}) \cdot \boldsymbol{v}_i^{(t+1)} \tag{39}$$

$$= [\boldsymbol{e}_i]_\times \cdot \boldsymbol{v}_i^{(t+1)} \tag{40}$$

$$= [-\boldsymbol{v}_i^{(t+1)}]_\times \cdot \boldsymbol{e}_i \tag{41}$$

$$\frac{\partial \boldsymbol{v}}{\partial t_i}(\mathbf{0}, \boldsymbol{p}_i) = \frac{\partial \tilde{\boldsymbol{\xi}}}{\partial t_i}(\mathbf{0}) \cdot \boldsymbol{v}_i^{(t+1)} \tag{42}$$

$$= \frac{\partial \boldsymbol{t}}{\partial t_i} \tag{43}$$

$$= \boldsymbol{e}_i \tag{44}$$

Here, $\boldsymbol{e}_i$ is the i-th basis vector of the standard basis of $\mathbb{R}^3$. More compactly, these derivatives can be written in matrix form by

$$\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{\xi}}(\mathbf{0}, \boldsymbol{p}_i) = \begin{bmatrix} [-\boldsymbol{v}_i^{(t+1)}]_\times & \boldsymbol{I} \end{bmatrix} \in \mathbb{R}^{3\times 6} \tag{45}$$

And here we see the great similarity to the final formula for the normal equation components of the KinectFusion ICP given in equation (24). Also the residual term coincide with the one given in equation (25). In both cases, we were able to reformulate the problem such that it can be solved by the Gauss-Newton method and thus calculate a Jacobian. Here, the Jacobian is obtained directly by using the chain rule while in the KinectFusion ICP setting it is derived by using the small angles assumption. Now, we can also state what the last reformulation means. The derivative $\frac{\partial \boldsymbol{v}}{\partial \boldsymbol{\xi}}(\mathbf{0}, \boldsymbol{p}_i)$ of the moving point is also be present there and the term before it, namely the negative transposed normal $\left(-\hat{\boldsymbol{n}}_{c(i)}^{(t)}\right)^\top$, is the derivative of the point-to-plane error.

To sum this up, the photometric energy can now also be written like the point-to-plane energy:

$$E_{rgbd} = \left\| \boldsymbol{J}_{rgbd}\, \boldsymbol{\xi} + \boldsymbol{r}_{rgbd} \right\|_2^2 \tag{46}$$

So we can solve it by the Gauss-Newton method in exactly the same way as before. As mentioned before, Kerl et al., (2013b) use a slightly different approach in the first iteration of their DVO SLAM system. While the Jacobian matrix is the same as here, they additionally weight it using some statistical distributions. The weights come from the Maximum-A-Posteriori formulation and help to improve the results. In their approach, this is a t-distribution. Its covariance matrix is used for weighting but must be computed beforehand. In addition, a motion prior is used guiding the optimization to prefer small motions. So while this variant can also be implemented efficiently, the authors already state that approximately twice the run-time is needed.

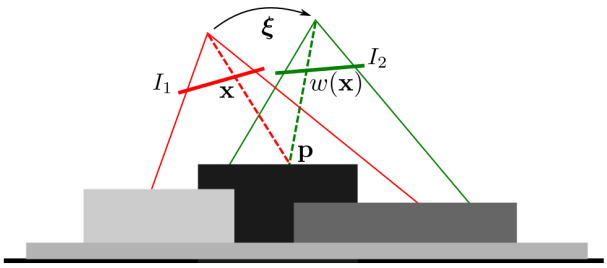


Figure 4: The warping function w shown here for a pixel $\boldsymbol{x}$ is essentially a correspondence mapping, taken from Kerl et al., (2013b). Note that in our setting the roles of two images are changed meaning that red indicates the new and green the previous measurements.

3.3 Combined ICP

So far, we showed two different approaches and their similarities. While the photometric variant of the ICP avoids the use of noisy depth data, it suffers from the assumption that the surfaces reflect ideally diffuse. Fortunately, both error functions use the same correspondences, are solved in the same way, defined in the same coordinate system and their limitations do not affect each other. Whelan et al., (2013) used these observations and combined the two error function so that all given information is used, the depth and the RGB image. Basically, the two error functions are weighted by a factor and then summarized to a combined Jacobian matrix and residual vector:

$$E_{combined} = E_{icp} + \lambda\, E_{rgbd} \tag{47}$$

$$\left(\boldsymbol{J}_{icp}^\top \boldsymbol{J}_{icp} + \lambda \boldsymbol{J}_{rgbd}^\top \boldsymbol{J}_{rgbd}\right)\boldsymbol{\xi} = \boldsymbol{J}_{icp}^\top \boldsymbol{r}_{icp} + \sqrt{\lambda}\boldsymbol{J}_{rgbd}^\top \boldsymbol{r}_{rgbd} \tag{48}$$

The computation costs are of course higher than using only one of the error terms. However, both have linear complexity in time and memory consumption which means that the total cost increases only by a constant factor. On the other hand, one can exploit the similarities between the two errors and speed-up the computation of the Jacobians.

This idea was further used in Kintinuous (Whelan et al., 2015b) and ElasticFusion (Whelan et al., 2015a). Kintinuous is an extension of the KinectFusion system that use a moving volume to allow large scenes to be reconstructed. ElasticFusion drops the volumetric structure and uses surfels for reconstruction. However, it still fuses measurements and therefore combines the fusion principle from the KinectFusion world with the surfel structure that is typically used in the family of the Normal Distribution Transform. Kerl et al., (2013a) also tried to combine both error metrics. In the second and current iteration of their DVO SLAM system, the Maximum-A-Posteriori approach is again used to solve the problem. In a post-processing step, all three system perform loop closure to achieve global consistency of the reconstructed model.

3.4 Generalized ICP

We continue our review of registration algorithms in the family of the ICP approaches by some probabilistic variants. As already introduced, Kerl et al., (2013b) converted the ICP energy into a Maximum-A-Posteriori problem. Similarly, Segal et al., (2009) presented a probabilistic variant which they called Generalized ICP. The basic idea is that the two sets of measurements $\mathcal{A}, \mathcal{B}$, which we want

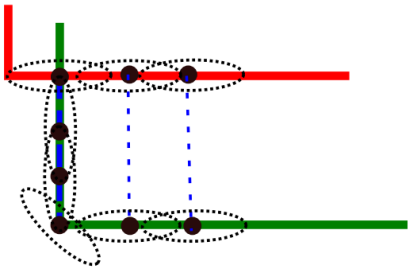


Figure 5: Influence of the covariance matrices on the plane-to-plane error, taken from Segal et al., (2009).

to align, are drawn from underlying sets $\hat{\mathcal{A}}, \hat{\mathcal{B}}$ probabilistically according to the normal distributions

$$\boldsymbol{v}_i^{(t+1)} \sim N(\boldsymbol{v}_{i\,opt}^{(t+1)}, \boldsymbol{\Sigma}_i^{(t+1)}) \tag{49}$$

$$\hat{\boldsymbol{v}}_j^{(t)} \sim N(\hat{\boldsymbol{v}}_{j\,opt}^{(t)}, \boldsymbol{\Sigma}_j^{(t)}) \tag{50}$$

This allows to model uncertainty in the measurements. Therefore instead of minimizing an energy, a Maximum-Likelihood estimate is computed:

$$\boldsymbol{T}^* = \arg\max_{\boldsymbol{T}} \prod_i p(d_i^{(\boldsymbol{T})}) = \arg\max_{\boldsymbol{T}} \sum_i \log\left(p(d_i^{(\boldsymbol{T})})\right) \tag{51}$$

with distances

$$d_i^{(\boldsymbol{T})} = \hat{\boldsymbol{v}}_{c(i)}^{(t)} - \boldsymbol{T} \cdot \boldsymbol{v}_i^{(t+1)} \tag{52}$$

Now it is assumed that both points are drawn from independent normal distributions, it follows that

$$d_i^{(\boldsymbol{T})} \sim N(0,\, \boldsymbol{\Sigma}_{c(i)}^{(t)} + \boldsymbol{T}\, \boldsymbol{\Sigma}_i^{(t+1)}\, \boldsymbol{T}) \tag{53}$$

Using this observation, the optimization problem can be formulated as

$$\boldsymbol{T}^* = \arg\min_{\boldsymbol{T}} \sum_i (d_i^{(\boldsymbol{T})})^\top \cdot (\boldsymbol{\Sigma}_{c(i)}^{(t)} + \boldsymbol{T}\, \boldsymbol{\Sigma}_i^{(t+1)}\, \boldsymbol{T})^{-1} \cdot d_i^{(\boldsymbol{T})} \tag{54}$$

To solve this, one first needs to know what the covariance matrices $\boldsymbol{\Sigma}_{c(i)}^{(t)}, \boldsymbol{\Sigma}_i^{(t+1)}$ are. If we use for example the configuration $\boldsymbol{\Sigma}_{c(i)}^{(t)} = \boldsymbol{I}, \boldsymbol{\Sigma}_i^{(t+1)} = \boldsymbol{0}$, the error functions reduces to the point-to-point distance where the square of the Euclidean distances between all point pairs is considered. In a similar way, also the point-to-plane distance can be constructed from this formulation. Therefore, this approach can be seen as a generalization of the standard ICP.

In order to define the so called plane-to-plane distance, the covariance matrices are chosen in a way such that the variance in the tangent plane is constant in every direction but small along the normal direction. For a normal that points along the x-axis, the covariance matrix would thus be

$$\boldsymbol{\Sigma}_i = \begin{pmatrix} \epsilon & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad \epsilon \in [0, 1] \tag{55}$$

Other covariances can be created from this one by using rotation matrices that map the normal to the x-axis. This is also shown in Figure 5. Here, the covariance matrices are visualized using ellipses and indicate uncertainty that is high in the tangent plane but quite low in normal direction.

Generalized ICP only changes the energy formulation and keeps the other steps unchanged. This means that the correspondences can be computed in the same way as for the other variants . However, the optimal transformation must now be computed using the Conjugate Gradient algorithm. Fortunately, there exist GPU implementations of this algorithm and those only need to operate on the correspondence set. Therefore Generalized ICP should also be quite efficient and fast enough to be included in our reconstruction pipeline. Since its optimization method is different from the other ones, the performance in terms of accuracy will be interesting, so we will compare this later more carefully.

3.5 Direct Volume Matching

Another quite interesting variant of doing frame-to-model tracking is to use the volumetric data structure of our 3d reconstruction framework directly. So instead of ray-casting the model from the volume and matching the new RGB-D image to it, one directly matches the incoming image to the volume. Bylow et al., (2013) and Canelhas et al., (2013) investigated this possibility and adjusted the Gauss-Newton solver to operate on the implicitly stored surface.

This surface is stored in the volumetric data structure and represented by a function

$$D_t : \mathbb{Z}^3 \to [-1,\, 1] \tag{56}$$

that maps voxels to truncated signed distance values. The three-dimensional space is therefore discretized and a signed distance field is constructed by this function. The surface itself is encoded as the zero-set of this function. Every time a new depth map gets integrated into the volume, the signed distance function D_t is updated:

$$D_{t+1}(\boldsymbol{x}) = \frac{D_t(\boldsymbol{x})\, W_t(\boldsymbol{x}) + d_{t+1}(\boldsymbol{x})\, w_{t+1}(\boldsymbol{x})}{W_t(\boldsymbol{x}) + w_{t+1}(\boldsymbol{x})} \tag{57}$$

$$W_{t+1}(\boldsymbol{x}) = W_t(\boldsymbol{x}) + w_{t+1}(\boldsymbol{x}) \tag{58}$$

To align the new data, an energy based on the signed distance can now be formulated:

$$E_{volume} = \sum_i \left(D_t(\tilde{\boldsymbol{\xi}} \cdot \boldsymbol{v}_i^{(t+1)})\right)^2 \tag{59}$$

Here, the fact is exploited that the signed distance not only gives a distance to the surface but also a direction. Thus, one is able to distinguish whether the current voxel is inside or outside the object. Like in the Photometric ICP this distance can now be linearized by a Taylor expansion

$$D_t(\tilde{\boldsymbol{\xi}} \cdot \boldsymbol{v}_i^{(t+1)}) \approx D_t(\boldsymbol{v}_i^{(t+1)}) + \nabla D_t(\boldsymbol{v}_i^{(t+1)}) \cdot \boldsymbol{\xi} \tag{60}$$

So we again can express the error in terms of a Jacobian and a residual:

$$E_{volume} = \|\boldsymbol{J}_{volume}\,\boldsymbol{\xi} + \boldsymbol{r}_{volume}\|_2^2 \tag{61}$$

Unfortunately, this formulation has several drawbacks in contrast to the previous ones. It is of course directly designed for our pipeline and exploits it but from the computational point-of-view it is not very efficient.

First, we consider the formulation more carefully. The signed distance is defined on discrete coordinate while in the error function, the actual 3D position $\boldsymbol{v}_i^{(t+1)}$ of a point is inserted. This means that the signed distance must be estimated using trilinear interpolation which requires eight look-ups in the volume. In a full volume, this operation would be very fast at the cost of the infeasible memory requirements. To do this in our scenario, we need eight hash table look-ups followed by eight access to the volume. One could sacrifice accuracy by speed when only the distance of the nearest voxel is used. However since drift is very critical in this application, this is also not an option. Even worse, the gradient of the signed distance function has to be computed. Usually, this is done by finite differences. But here, these are differences of trilinear interpolated values so the cost is even higher.

We also see a general limitation of this approach. It can only operate perfectly on full volumes. However, our pipeline exploits the facts the empty space is not needed for reconstruction and thus only stores a small region round the actual surface, called the truncation region. Only if the gradient and the residual are evaluated inside this region, they are correct. Otherwise, we would have to prune them. If the camera however moves quite fast, some important parts may get out of this region and the performance of the tracker would decreases rapidly. Therefore, we believe that this approach will not give the best performance neither in accuracy nor in run time.

4 Normal Distribution Transform

Up to now, we showed several approaches in the family of the ICP algorithm. In the following, we will consider another family of algorithms, the Normal Distribution Transform (NDT) invented by Biber and Straßer, (2003). While ICP algorithms directly align the two point sets, NDT changes the representation of one point set to a mixture of normal distributions and then tries aligns the other set to this model.

4.1 3D-NDT

Whereas the original NDT algorithm only operated on 2D data, Magnusson et al., (2007) extended the framework to 3D. Like in the 2D version, the data is subdivided into smaller sets by a regular volumetric grid. In each cell C_i, the points falling in it are modeled by a normal distribution $N(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)$:

$$\boldsymbol{\mu}_i = \frac{1}{K}\sum_{k=1}^{K} \hat{\boldsymbol{v}}_k^{(t)} \tag{62}$$

$$\boldsymbol{\Sigma}_i = \frac{1}{K-1}\sum_{k=1}^{K}\left(\hat{\boldsymbol{v}}_k^{(t)} - \boldsymbol{\mu}_i\right)\left(\hat{\boldsymbol{v}}_k^{(t)} - \boldsymbol{\mu}_i\right)^\top \tag{63}$$

where K is the number of points in the current cell C_i. For a given point, thew likelihood that it is observed in the model can be expressed by the probability density function

$$p_{c(i)}(\boldsymbol{x}_i) \propto \exp\left(-\frac{(\boldsymbol{x}_i - \boldsymbol{\mu}_{c(i)})^\top\,\boldsymbol{\Sigma}_{c(i)}^{-1}\,(\boldsymbol{x}_i - \boldsymbol{\mu}_{c(i)})}{2}\right) \tag{64}$$

Since the task is to align the two point sets, the likelihood of all points of the new measurement should be high. Thus a score function can be defined in the following way:

$$s_{p2d}(\boldsymbol{\xi}) = -\sum_{i=1}^{n} p_{c(i)}\left(\tilde{\boldsymbol{\xi}} \cdot \boldsymbol{v}_i^{(t+1)}\right) \tag{65}$$

The advantage of this formulation is that analytic derivatives of the score function exist. Finding the optimal pose $\boldsymbol{\xi}$ that minimizes this function is actually finding a pose where the first derivative of the score function is zero. Like in the original paper, Magnusson et al., (2007) also suggest to use the Newton method and solve the following equation system:

$$\boldsymbol{H}\,\boldsymbol{\xi} = -\boldsymbol{g} \tag{66}$$

Here, $\boldsymbol{H} \in \mathbb{R}^{6\times6}$ denotes the Hessian and $\boldsymbol{g} \in \mathbb{R}^6$ the gradient of s_{p2d}. In their paper, Magnusson et al., (2007) used the axis-angle representation of a rotation where the angle is considered separately and the axis is always normalized. In this case, they optimized seven instead of six parameters. However, this only leads to a slightly different Hessian and gradient. To be consistent with the other approaches, we use this formulation.

Now, we discuss the strengths and weakness of the 3D-NDT. As a preprocessing step, the normal distributions have to be constructed. Thus, all points need to be associated to their corresponding cells in the regular grid and the mean and covariance of each cell must be calculated. Computing nearest neighbors might be problematic since building search structures such as KD-Trees on the GPU is complicated. Furthermore, the projective behavior of the points is not further considered or exploited which makes the model more general. On the other hand, some optimization potential is lost making the approach possibly slower than others.

As stated in their paper, this representation is mush more compact than storing and using all points. So from the computational point of view, the change in the representation introduces additional complexity but fortunately this only

has to be performed on the fixed data set that is not moving. Nevertheless, we have to take this preprocessing into account. Most computation steps including preprocessing and matching can be performed in parallel on the GPU. Like in the ICP approaches, a tree reduction can be used to compute the Hessian and the gradient.

4.2 Color 3D-NDT

One big advantage of using normal distributions is that they can be modeled in any dimension. That was the reason for the extension to 3D and it is also the reason to extend it to incorporate colors. Huhle et al., (2008) developed this extension. A straightforward extension would be to convert colors to the $L^*a^*b^*$ color space, where perceptual differences coincide to the L2-distance, and use a 6D feature vector consisting of the 3D position and the color. Then, only the Hessian and the gradient have to be adjusted and the solution is again found by the Newton method. However, Huhle et al., (2008) observed that a single normal distribution per cell is not able to model the color distribution accurately.

To construct a better model, the authors suggest to use a Gaussian Mixture Model (GMM):

$$p_i(\boldsymbol{x}^*) = \sum_{j=1}^{M} \alpha_j \, N(\boldsymbol{\mu}^*_{i,j}, \, \boldsymbol{\Sigma}^*_{i,j}) \tag{67}$$

Here, $\boldsymbol{x}^*$ refers to the color and $\boldsymbol{x}$ to the 3D position of a point. So for each cell C_i, M normal distributions are computed using Expectation Maximization (EM) with an initial guess calculated from k-means. These color distributions are then used as weighting functions to the 3D-NDT algorithm:

$$w_{ij}(\boldsymbol{x}^*_i) = \exp\left(-\frac{(\boldsymbol{x}^*_i - \boldsymbol{\mu}^*_{c(i),j})^\top \, \boldsymbol{\Sigma}^{*-1}_{c(i),j} \, (\boldsymbol{x}^*_i - \boldsymbol{\mu}_{c(i),j})}{2}\right) \tag{68}$$

Having these weights, the mean and the covariance of the 3D positions of each color distribution are computed. Note that in contrast to the standard 3D-NDT algorithm M normal distributions $N(\boldsymbol{q}_{ij}, \boldsymbol{\Sigma}_{ij})$ are computed for each cell C_i. The score function is then given as

$$s_{p2d,color}(\boldsymbol{\xi}) = -\sum_{i=1}^{n} \sum_{j=1}^{M} w_{ij} \, p_{c(i),j}\left(\tilde{\boldsymbol{\xi}} \cdot \boldsymbol{v}_i^{(t+1)}\right) \tag{69}$$

where the probability density function is now defined as

$$p_{c(i),j}(\boldsymbol{x}_i) \propto \exp\left(-\frac{(\boldsymbol{x}_i - \boldsymbol{\mu}_{c(i),j})^\top \, \boldsymbol{\Sigma}^{-1}_{c(i),j} \, (\boldsymbol{x}_i - \boldsymbol{\mu}_{c(i),j})}{2}\right) \tag{70}$$

As in 3D-NDT, the score can be optimized by the Newton method.

This approach inherits most of the properties of the underlying 3D-NDT algorithm. However, the amount of work per cell is now much higher than before. Huhle et al., (2008) suggest to use $M = 3$ mixture models to guarantee unique poses which requires $2 \cdot 3 = 6$ normal distributions per cell. However, these normal distributions have to be computed by Expectation Maximization and k-means which both can run in parallel on subsets of the data on the GPU. Since the number of iterations varies for each subset, some performance of the GPU might be lost. In addition, the preprocessing step is much more costly and the compactness of the model is also lower than before because now 6 normal distributions per cell are used.

4.3 D2D 3D-NDT

Some years later, Stoyanov et al., (2012) came up with an idea that is similar to the Generalized ICP. In the standard ICP, the point-to-plane distance between two point clouds is minimized and this was generalized to what Segal et al., (2009) called the plane-to-plane distance. Consequently, the question arose whether it was possible to extend the 3D-NDT in the same way meaning that two sets of normal distributions should be registered. The task is therefore to find a transformation that aligns the two models

$$\boldsymbol{T}(M^{(t+1)}_{NDT}) = \{N(\boldsymbol{T}(\boldsymbol{\mu}_i), \boldsymbol{T}(\boldsymbol{\Sigma}_i))\} \tag{71}$$

$$M^{(t)}_{NDT} = \{N(\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j)\} \tag{72}$$

Distances between two probability density functions can be defined in different ways. However, the standard L2-distance between all possible points is a simple but reasonable choice:

$$d_{L2} = \int \left(p(\boldsymbol{x} \mid M^{(t)}_{NDT}) - p(\boldsymbol{x} \mid \boldsymbol{T}(M^{(t+1)}_{NDT}))\right)^2 d\boldsymbol{x} \tag{73}$$

After rearranging the terms and applying some identities of normal distributions, the L2-distance simplifies to

$$d_{L2} \sim \sum_i \sum_j p(\boldsymbol{0} \mid \boldsymbol{T}(\boldsymbol{\mu}_i) - \boldsymbol{\mu}_j, \boldsymbol{T}(\boldsymbol{\Sigma}_i) + \boldsymbol{\Sigma}_j) \tag{74}$$

A new score function can now be defined:

$$s_{d2d}(\boldsymbol{\xi}) = -\sum_i \sum_j p(\boldsymbol{0} \mid \boldsymbol{T}(\boldsymbol{\mu}_i) - \boldsymbol{\mu}_j, \boldsymbol{T}(\boldsymbol{\Sigma}_i) + \boldsymbol{\Sigma}_j) \tag{75}$$

As a consequence of this formulation, the optimization can be performed exactly in the same way as for the original 3D-NDT algorithm. Only the mean and the covariance differ. Like Magnusson et al., (2007), the score function is approximated by only considering the nearest normal distribution for each of the transposed ones. Thus, the matching here is faster than the standard 3D-NDT if the size of the model is significantly smaller than the size of the scan data. On the other hand, two models have to be constructed in a preprocessing step resulting in some

overhead. Similar to the ICP optimization, a coarse-to-fine scheme improves the results by optimizing in different levels of details to hopefully avoid local minima. While in the ICP setting this can be done cheaply by calculating the first levels of the Gaussian pyramid of the RGB-D images, here all points have to be considered in each level making the construction slower. Stoyanov et al., (2012) already recognized that this step is more costly than the actual registration so we have to consider this in our comparison.

4.4 Multi-Resolution Surfel Maps

More work was done to develop a model that allows cheap matching like the D2D 3D-NDT and also incorporate color information as in Color 3D-NDT. The result of this work were Multi-Resolution Surfel Maps invented by Stückler and Behnke, (2014). In their work, the authors extend the probabilistic framework of the Normal Distribution Transform by adding additional information.

First, a mean $\boldsymbol{\mu} \in \mathbb{R}^6$ and a covariance matrix $\boldsymbol{\Sigma} \in \mathbb{R}^{6\times6}$ on the shape and color are calculated for each cell. Inspired by the Fast Point Feature Histograms (FPFH) by Rusu et al., (2009), a descriptor $\boldsymbol{h} \in \mathbb{R}^{12}$ is constructed and used as an additional source of information for correspondence finding. To achieve high frame-rates, the authors state that the neighborhood relations between the cells are stored explicitly, so the 26 neighbors can be found in constant time. The surfel maps also provide a multiple resolution structure by using octrees to implicitly support a coarse-to-fine scheme.

In the registration stage, a score function is optimized by first using the Levenberg-Marquardt method to get an initial coarse pose estimate and then refining this estimate by the Newton method. In particular, Stückler and Behnke, (2014) used the same score function as in D2D 3D-NDT but only considered correspondences with sufficiently small L2-distance between the shape-texture descriptors. In a post-processing step, they detect loop closures to further reduce accumulated drift and improve results. We will later discuss this concept more carefully later.

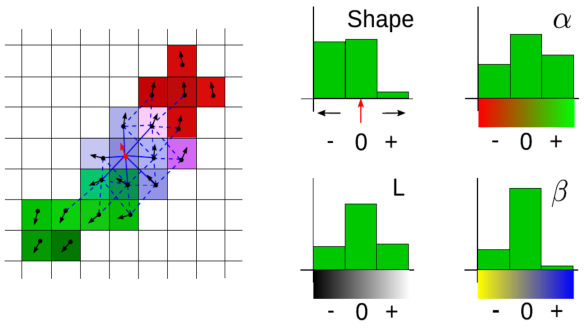


Figure 6: 2D illustration of the shape-texture descriptor, taken from Stückler and Behnke, (2014).

So in principle, Multi-Resolution Surfel Maps are a consequent evolution in the family of the Normal Distribution Transform. Since they inherit most properties from their predecessors, we only need to evaluate how efficient their construction on the GPU can be. Originally, this approach was developed and implemented on the CPU and the authors already argue that they gained significant speed-ups by using all cores of the CPU. However, to use the full potential of GPU's the whole process must run fully in parallel. The only challenging part is the octree structure. In the literature, there are many papers on GPU octrees but the explicitly stored neighbors here are somehow problematic. While they are allowing us to get a constant time lookup, they introduce a randomness in the memory lookups and additional memory costs. And since GPU memory is still quite valuable and GPU's typically operate on coherent data, it is not directly clear how well a GPU implementation of Multi-Resolution Surfel Maps will perform. However, we highlight again that this approach introduces a lot of complexity. That is probably because this family of algorithms can be easily extended to a full reconstruction algorithm and is therefore more than a simple registration algorithm.

5 Sparse Methods

Algorithms that are more oriented in the field of Computer Vision use features to estimate the camera pose. So instead of using all the data and minimizing an error function, one only considers parts of the scene which are informative and characteristic. This is reasonable since only very few points in an image are characteristic enough to describe the transformation like border points or points with a very strong gradient. More generally, these points can be seen as features and the task is to detect enough of them to robustly estimate the camera pose. An example of detected features is shown in Figure 7.

In the following, we will consider the approaches of Endres et al., (2012) and Huang et al., (2011). While the basic idea is the same in both systems, they made different decisions on the kind of features and how they are matched. In particular, the RGB-D SLAM system by Endres et al., (2012) use the combination of SIFT, SURF and ORB features. Since they argue that especially the SIFT features are computationally demanding, a GPU-accelerated version is used. After the features are matched, RANSAC is used to compute the best rigid transformation. To reduce accidental accumulation of drift, the registration is performed on the last three recent frame and seventeen randomly sampled earlier frames. Finally, they detect loop closures and optimize the pose graph to further reduce drift.

A similar technique is used by the fovis algorithm of Huang et al., (2011). But they only rely on FAST features and use a different inlier detection method. Instead of RANSAC, the problem is formulated in a graph of consistent features. To get the best transformation, a

Figure 7: Extracted features shown on the RGB image and the 3D point cloud, taken from Henry et al., (2012).

Maximal-Clique-Problem needs to be solved. Unfortunately, Maximal-Clique is NP-complete, so the exact solution can not be determined efficiently. The authors therefore suggest to use a greedy approximation to solve the problem in less time. To reduce drift, the registration is performed against a reference key-frame instead of the previous frame. It is a common technique and also used in other approaches like in Multi-Resolution Surfel Maps. For global consistency, loop closures are detected.

Unlike all previously discussed algorithms, sparse methods can not so easily be implemented on the GPU. Basically, that class of algorithms can be split into three steps. First, features are computed. This can be done on the CPU if they are fast to evaluate. For computationally more demanding descriptors, like the SIFT descriptor, a GPU version has to be used. In case of the SIFT descriptor, there is such an implementation publicly available (Wu, 2007). Second, the features have to be matched to get a list of pairs. Since the features are independent of each other, this can be parallelized and speeded-up by only considering a reasonably small search window. Third, the best transformation from the pairs has to be found. While RANSAC search the solution by randomly testing candidates, the greedy approximation of Maximal-Clique iteratively improve the found solution. Furthermore, the solver for the transformation parameters $\boldsymbol{\xi}$ must be implemented on the GPU in this case. This is non-trivial and special care has to be taken here to avoid numerical instabilities. But since this method is sparse, CPU versions can also be very fast.

6 Loop Closure

The last point we want to discuss are loop closures. In a few previously considered algorithms, the authors try to achieve global consistency by explicitly modeling this as a loop closure detection problem. This is rather important to prevent implausible reconstructions like it is shown in Figure 8. The basic idea is to build a graph where the nodes represent the camera poses $\boldsymbol{\xi}_i$ of some selected key-frames and edges between them the incremental transformations $\boldsymbol{\Delta T}_{ij}$. In order to optimize the graph, the corresponding reconstruction representations of the key-frames are attached to them. The optimization itself is usually performed by the generic graph optimization framework of Kümmerle et al., (2011) by minimizing a cost function based on the key-frames.

In the DVO SLAM system of Kerl et al., (2013a) this framework is used. Stückler and Behnke, (2014) also considered loop closures to improve their results. Fusion systems using loop closures that are closer to our pipeline are the Kintinuous (Whelan et al., 2015b) and ElasticFusion (Whelan et al., 2015a) system. While in Kintinuous a moving volume approach is used and the pose graph consisting of cloud slices of this volume is optimized after detecting a loop closure, ElasticFusion uses a surfel-based map representation and also overcomes the need of a pose graph.

However, our VoxelHashing pipeline strongly exploits the fact that the scene is static and thus culls all empty parts of the volume away. This assumption is very restrictive and even the loop closure system of Kintinuous can not be easily used since it would require to deform at least a part of the volume. Therefore, large parts of the hash table have to be changed since voxel blocks are moving. This is a drawback of our pipeline and nevertheless we include this strategy into our evaluation to see how much results improve when considering global consistency.

7 Comparison

In this part, we compare the discussed algorithms. Fortunately, most of them use the RGB-D SLAM benchmark by Sturm et al., (2012) to show their performance. The measurements of the scenes contained in this benchmark are real-world data and also ground truth trajectories are available. In addition, they also define useful error metrics like the absolute and relative pose error and provide a tool for evaluation.

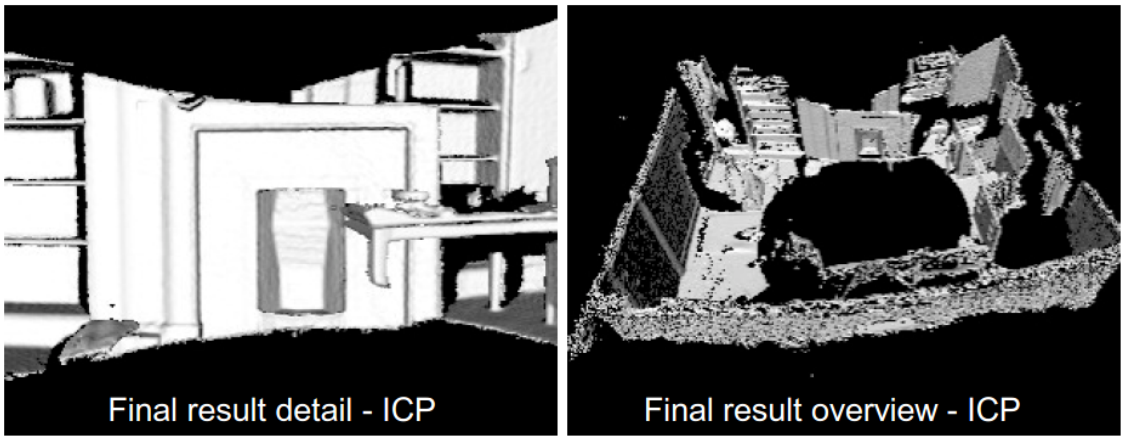


Figure 8: Drift accumulates and gets visible if a loop is closed, taken from Kähler et al., (2015).

However since the benchmark contains many different test scenes, each author of the presented papers have chosen different subsets of the provided scenes making a direct comparison impossible. Thus, we decided to compress the collected results. So instead of simply listing and comparing the error values measured by the authors directly, we use Won-Lost-Tables which are commonly used in the field of Machine Learning to compare the performance of learning algorithms. The main advantage is that they are abstracting from the absolute results and only measure the relative performance between two algorithms for the whole dataset. In particular, for each paper we construct a table that summarizes the results in a way that we can compare each pair of algorithms and decide which one performs better on the test set. Note that we have only done this for papers where the respective authors have tested their approach against more than one of the other approaches we have mentioned in this report and also used enough test scenes.

Constructing these tables is done in the following way. First, we compute the mean error of each algorithm L:

$$\mathrm{ME}(L) = \frac{1}{n} \sum_{i=1}^{n} e_i(L) \tag{76}$$

From these means, one can now compute the quotient $\mathrm{ME}(L_1) \,/\, \mathrm{ME}(L_2)$ which states the relative mean error of algorithm L_1 compared to L_2. This is our first measurement and it is built for all pairs of algorithms. It compresses the performance across all test cases and tells us how much L_1 is better or worse than L_2.

The second measurement is constructed by evaluating which algorithms was better at which test case. This means that for each scene one determines if L_1 has won, lost or was equally good against L_2 and accumulates this information. As a result, one gets the number of wins, losses and ties of L_1 against L_2 in the complete test set. Using that information helps to interpret the relative mean error since one large deviation can influence this value quite significantly. On the other hand, the won/lost/ties statistic is not affected from such deviations and thus helps us to give better interpretations. The number of scenes on which the approaches are evaluated is also implicitly encoded in these tables by summing the wins, losses and ties.

In Table 1 we show the results of our comparison. Here, the ICP variants KinectFusion ICP (Izadi et al., 2011; Newcombe et al., 2011; Nießner et al., 2013), Photometric ICP (Steinbrücker et al., 2011), Combined ICP (Whelan et al., 2013), Kintinuous (Whelan et al., (2015b)), ElasticFusion (Whelan et al., (2015a)), Generalized ICP (Segal et al., 2009) and DVO SLAM (Kerl et al., 2013a) are compared. The NDT approaches are D2D 3D-NDT (Stoyanov et al., 2012) and Multi-Resolution Surfel Maps (Stückler and Behnke, 2014) and from the sparse methods fovis (Huang et al., 2011) and RGB-D SLAM (Endres et al., 2012) are chosen. To ensure an objective comparison, all results are weighted equally.

First, we focus on the results of Whelan et al., (2013). They introduced the Combined ICP and compared its performance against the geometric and the photometric variant. In addition, they also evaluated the fovis system. From the table, one can infer that the Combined ICP is much stronger than the KinectFusion ICP and Photometric ICP. This result is quite intuitive since the motivation of it was that the strengths and the weaknesses of the two approaches were different and a combination would help both parts to perform better. Furthermore, Photometric ICP seems to be also much better than the standard KinectFusion ICP approach. Here, the much lower noise in the RGB images allows the algorithm to be more precise. fovis performs also very well and is on par with the Combined ICP while having a slightly lower mean error. Experiments of Whelan et al., (2013) also showed that especially in cases where only few or no visual and geometric features are available, like in corridor scenes shown in Figure 9, the combination of depth and color data stabilizes the camera pose estimation process and makes it very robust in such scenarios.

Next, we analyze the results of Multi-Resolution Surfel Maps. Interestingly, this approach is considered to be

Table 1: Won-Lost-Tables. If L_c and L_r indicate the column and row algorithms, then the lower triangle shows the number of win-loss-ties of L_c versus L_r whereas the upper triangle indicates the quotient of the average errors ME(L_r) / ME(L_c).

(a) Results built from Stückler and Behnke, (2014) using relative pose error (RPE) median

	Multi-Resolution Surfel Maps	Photometric ICP	Generalized ICP	D2D 3D-NDT	fovis
Multi-Resolution Surfel Maps		0.667	0.646	0.699	0.859
Photometric ICP	18-3-1		0.968	1.049	1.288
Generalized ICP	19-3-0	12-10-0		1.083	1.331
D2D 3D-NDT	20-2-0	15-7-0	9-13-0		1.229
fovis	18-4-0	17-4-1	9-13-0	5-17-0	

(b) Results built from Kerl et al., (2013a) using absolute trajectory error (ATE) root-mean-square error (RMSE)

	DVO SLAM	RGB-D SLAM	Multi-Resolution Surfel Maps	KinectFusion ICP
DVO SLAM		0.629	0.791	0.114
RGB-D SLAM	8-2-0		1.256	0.182
Multi-Resolution Surfel Maps	8-2-0	3-7-0		0.145
KinectFusion ICP	10-0-0	9-1-0	9-1-0	

(c) Results built from Whelan et al., (2013) using relative pose error (RPE) root-mean-square error (RMSE)

	Combined ICP	KinectFusion ICP	Photometric ICP	fovis
Combined ICP		0.233	0.726	1.289
KinectFusion ICP	4-0-0		3.117	5.537
Photometric ICP	4-0-0	0-4-0		1.224
fovis	2-2-0	0-4-0	1-3-0	

(d) Results built from Whelan et al., (2015b) using absolute trajectory error (ATE) root-mean-square error (RMSE)

	Kintinuous	DVO SLAM	RGB-D SLAM	Multi-Resolution Surfel Maps
Kintinuous		1.494	1.010	1.024
DVO SLAM	0-10-0		0.676	0.685
RGB-D SLAM	4-6-0	6-4-0		1.014
Multi-Resolution Surfel Maps	5-5-0	10-0-0	6-4-0	

(e) Results built from Whelan et al., (2015a) using absolute trajectory error (ATE) root-mean-square error (RMSE)

	ElasticFusion	ElasticFusion (no deformation)	Kintinuous	Multi-Resolution Surfel Maps	RGB-D SLAM	DVO SLAM
ElasticFusion		0.236	0.355	0.053	0.358	0.356
ElasticFusion (no deformation)	7-0-1		1.506	0.224	1.517	1.509
Kintinuous	6-2-0	5-3-0		0.148	1.007	1.002
Multi-Resolution Surfel Maps	8-0-0	7-1-0	7-1-0		6.786	6.750
RGB-D SLAM	6-2-0	3-5-0	4-4-0	0-8-0		0.995
DVO SLAM	8-0-0	4-4-0	4-4-0	1-7-0	6-2-0	

a reference algorithm in the literature since most of the papers evaluated their approaches against it. Here, this has the great advantage that if we assume that its performance is the same across the papers, we are able to compare the algorithms across this boundary. Stückler and Behnke, (2014), the inventors of the Multi-Resolution Surfel Maps, compared it to the Generalized ICP, the Photometric ICP, fovis and D2D 3D-NDT on which it is based on. It is

also not very surprising that Multi-Resolution Surfel Maps outperform its competitors quite significantly in the number of wins and also in relative accuracy. The main reason for this is that none of the other algorithms explicitly model a loop closure. So indeed the performance is better but the algorithm is not a pure registration algorithm anymore. We can also see that the Photometric ICP achieves very good results and is most of the time better than the other algorithms. This is again due to its design, it is the only one except fovis and Multi-Resolution Surfel Mapsthat uses color information and is therefore better than the non-color registration algorithms.

If we now compare Multi-Resolution Surfel Maps against other algorithms implementing a loop closure system, we see that its dominance is gone. A bit surprising are the results from Whelan et al., (2015a). Here, the approach of Stückler and Behnke, (2014) totally fails against all other algorithms by at least one order of magnitude. This is simply caused by two of the eight tests where Multi-Resolution Surfel Maps probably computed totally wrong poses resulting in a very high error. But even if we only count the other six tests, the results are still not very good. On the other hand, DVO SLAM seems to be very robust and is either on par with or better than the other approaches. Its main advantage is the usage of both color and depth data combined in a robust Maximum-A-Posteriori formulation. However Kintinuous, which also used all the available data, is accurate enough to be on par with it. Here, we see another interesting effect. Even though the Maximum-A-Posteriori model is much more general and allows to weight each Jacobian and residual to be weighted individually, this advantage can somehow be compensated if a fused model that is much smoother and contain less noise is used. While in KinectFusion, this seems to have not a great impact, Kintinuous uses the advanced Combined ICP and also models loop closures by deforming the reconstruction. In that way, its performance is quite well. However, further tests by Whelan et al., (2015b) showed that DVO SLAM is in fact better and achieves better results. Nevertheless, it is interesting to see how a fused model helps to find better poses.

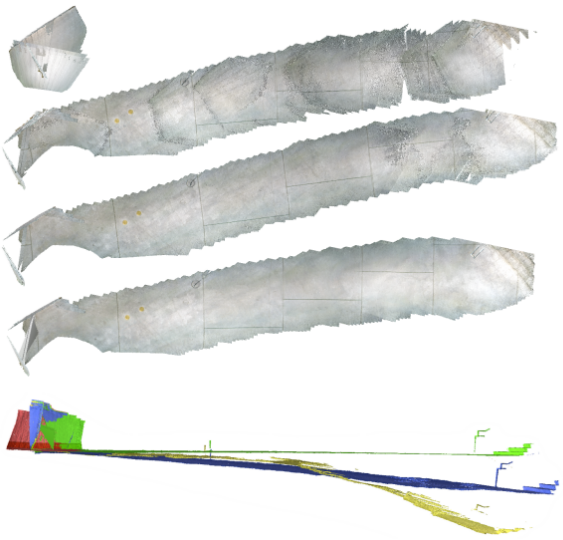


Figure 9: Reconstruction results of another data set, taken from Whelan et al., (2013). From top to bottom and in colors red, yellow, blue, green: KinectFusion ICP, Photometric ICP, fovis, Combined ICP.

Across all papers, KinectFusion ICP performs very bad and is outperformed by all other algorithms. This is not very surprising because most of its competitors have models that are more complex and use the data in a better way. The sparse methods RGB-D SLAM and fovis also achieve very good results and especially fovis is more accurate than Generalized ICP, D2D 3D-NDT and the Photometric ICP. ElasticFusion, probably the most advanced approach, beats every other algorithm quite significantly and even with disabled loop closure mechanism its performance is still very good. This is in fact very surprising. Disabling loop closure detection removes the possibility to correct drift that was accumulated during registration. Only if the estimation of the camera pose is very accurate, loop closures get less important. But in our scenario, the noise produced by depth sensors is really problematic and thus we don not expected such impressive results. The hybrid model that is somehow in-between the classic volumetric approach and the probabilistic NDT model seems to perform very well and combine the strengths of both worlds.

7.1 Conclusion

Summarizing all results we can conclude that the most powerful pure registration algorithm for our pipeline is the Combined ICP approach. It is fast, accurate and achieve very good results. If we consider all approaches, ElasticFusion achieves the overall best performance and is still good even if deformations are disabled. Its model also seem to be better and more flexible than ours.

The main drawback of our pipeline is the dependence on static scenes which makes it inflexible against deformations. This includes the mentioned loop closures and also dynamic scenes. Recently, Newcombe et al., (2015) extended KinectFusion in this way to reconstruct moving and deformable objects. ElasticFusion is also an example of a more flexible system. All in all, we have discovered interesting registration and reconstruction algorithms and we hope that the report will help researchers to find new ways to solve this hard problem.

References

[1] P. J. Besl and N. D. McKay. “A Method for Registration of 3-D Shapes”. In: *IEEE Trans. Pattern Anal. Mach. Intell.* 14.2 (1992), pp. 239–256.

[2] P. Biber and W. Straßer. "The normal distributions transform: A new approach to laser scan matching". In: *Intelligent Robots and Systems, 2003.(IROS 2003). Proceedings. 2003 IEEE/RSJ International Conference on*. Vol. 3. IEEE. 2003, pp. 2743–2748.

[3] G. Blais and M. D. Levine. "Registering multiview range data to create 3D computer objects". In: *Pattern Analysis and Machine Intelligence, IEEE Transactions on* 17.8 (1995), pp. 820–824.

[4] E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers. "Real-time camera tracking and 3d reconstruction using signed distance functions". In: *Robotics: Science and Systems (RSS) Conference 2013*. Vol. 9. 2013.

[5] D. R. Canelhas, T. Stoyanov, and A. J. Lilienthal. "SDF Tracker: A parallel algorithm for on-line pose estimation and scene reconstruction from depth images". In: *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE. 2013, pp. 3671–3676.

[6] J. Chen, D. Bautembach, and S. Izadi. "Scalable Real-time Volumetric Surface Reconstruction". In: *ACM Trans. Graph.* 32 (2013), 113:1–113:16.

[7] Y. Chen and G. Medioni. "Object Modelling by Registration of Multiple Range Images". In: *Image Vision Computing (IVC)* 10.3 (1992), pp. 145–155.

[8] B. Curless and M. Levoy. "A Volumetric Method for Building Complex Models from Range Images". In: *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. 1996, pp. 303–312.

[9] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard. "An evaluation of the RGB-D SLAM system". In: *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE. 2012, pp. 1691–1696.

[10] G. Gallego and A. Yezzi. "A compact formula for the derivative of a 3-D rotation in exponential coordinates". In: *Journal of Mathematical Imaging and Vision* 51.3 (2015), pp. 378–384.

[11] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox. "RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments". In: *The International Journal of Robotics Research* 31.5 (2012), pp. 647–663.

[12] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy. "Visual odometry and mapping for autonomous flight using an RGB-D camera". In: *International Symposium on Robotics Research (ISRR)*. 2011, pp. 1–16.

[13] B. Huhle, M. Magnusson, W. Straßer, and A. J. Lilienthal. "Registration of colored 3d point clouds with a kernel-based extension to the normal distributions transform". In: *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*. IEEE. 2008, pp. 4025–4030.

[14] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison, and A. Fitzgibbon. "KinectFusion: Real-time 3D Reconstruction and Interaction Using a Moving Depth Camera". In: *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*. UIST '11. ACM, 2011, pp. 559–568.

[15] O. Kähler, V. A. Prisacariu, C. Y. Ren, X. Sun, P. Torr, and D. Murray. "Very high frame rate volumetric integration of depth images on mobile devices". In: *Visualization and Computer Graphics, IEEE Transactions on* 21.11 (2015), pp. 1241–1250.

[16] C. Kerl, J. Sturm, and D. Cremers. "Dense visual slam for rgb-d cameras". In: *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE. 2013, pp. 2100–2106.

[17] C. Kerl, J. Sturm, and D. Cremers. "Robust odometry estimation for RGB-D cameras". In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE. 2013, pp. 3748–3754.

[18] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. "g2o: A general framework for graph optimization". In: *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE. 2011, pp. 3607–3613.

[19] S. Laine and T. Karras. "Efficient Sparse Voxel Octrees". In: *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*. I3D '10. 2010, pp. 55–63.

[20] K.-L. Low. *Linear Least-Squares Optimization for Point-to-Plane ICP Surface Registration*. Tech. rep. Chapel Hill, University of North Carolina, 2004.

[21] M. Magnusson, A. Lilienthal, and T. Duckett. "Scan registration for autonomous mining vehicles using 3D-NDT". In: *Journal of Field Robotics* 24.10 (2007), pp. 803–827.

[22] R. A. Newcombe, D. Fox, and S. M. Seitz. "DynamicFusion: Reconstruction and Tracking of Non-rigid Scenes in Real-Time". In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2015, pp. 343–352.

[23] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohli, J. Shotton, S. Hodges, and A. Fitzgibbon. "KinectFusion: Real-Time Dense Surface Mapping and Tracking". In: *IEEE ISMAR*. IEEE, 2011.

[24] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. "Real-time 3D Reconstruction at Scale Using Voxel Hashing". In: *ACM Trans. Graph.* 32.6 (2013), 169:1–169:11.

[25] H. Roth and M. Vona. "Moving Volume KinectFusion". In: *Proceedings of the British Machine Vision Conference*. BMVA Press, 2012, pp. 112.1–112.11.

[26] R. B. Rusu, N. Blodow, and M. Beetz. "Fast point feature histograms (FPFH) for 3D registration". In: *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*. IEEE. 2009, pp. 3212–3217.

[27] A. Segal, D. Haehnel, and S. Thrun. "Generalized-ICP." In: *Robotics: Science and Systems*. Vol. 2. 4. 2009.

[28] F. Steinbrücker, J. Sturm, and D. Cremers. "Real-time visual odometry from dense RGB-D images". In: *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*. IEEE. 2011, pp. 719–722.

[29] P. Stotko and T. Golla. "Improved 3D Reconstruction using Combined Weighting Strategies". In: *Proceedings of CESCG 2015: The 19th Central European Seminar on Computer Graphics*. 2015, pp. 135–142.

[30] T. D. Stoyanov, M. Magnusson, H. Andreasson, and A. Lilienthal. "Fast and accurate scan registration through minimization of the distance between compact 3D NDT representations". In: *The International Journal of Robotics Research* (2012), p. 0278364912460895.

[31] J. Stückler and S. Behnke. "Multi-resolution surfel maps for efficient dense 3D modeling and tracking". In: *Journal of Visual Communication and Image Representation* 25.1 (2014), pp. 137–147.

[32] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers. "A benchmark for the evaluation of RGB-D SLAM systems". In: *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE. 2012, pp. 573–580.

[33] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald. "Kintinuous: Spatially Extended KinectFusion". In: *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*. Sydney, Australia, 2012.

[34] T. Whelan, S. Leutenegger, R. F. Salas-Moreno, B. Glocker, and A. J. Davison. "ElasticFusion: Dense SLAM without a pose graph". In: *Proceedings of Robotics: Science and Systems (RSS)*. 2015.

[35] T. Whelan, M. Kaess, H. Johannsson, M. Fallon, J. J. Leonard, and J. McDonald. "Real-time large-scale dense RGB-D SLAM with volumetric fusion". In: *The International Journal of Robotics Research* 34.4-5 (2015), pp. 598–626.

[36] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. McDonald. "Robust real-time visual odometry for dense RGB-D mapping". In: *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE. 2013, pp. 5724–5731.

[37] C. Wu. "SiftGPU: A GPU implementation of scale invariant feature transform (SIFT)". In: (2007). URL: `http://cs.unc.edu/~ccwu/siftgpu/`.

[38] M. Zeng, F. Zhao, J. Zheng, and X. Liu. "A Memory-Efficient KinectFusion Using Octree". In: *Computational Visual Media*. Springer, 2012, pp. 234–241.

[39] M. Zeng, F. Zhao, J. Zheng, and X. Liu. "Octree-based Fusion for Realtime 3D Reconstruction". In: *Graph. Models* 75.3 (2013), pp. 126–136.