TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

Bachelor's Thesis in Informatics

# 3D Scene Reconstruction for 2D Object Recognition

Kanstantsin Tkachuk

TECHNICAL UNIVERSITY OF MUNICH

DEPARTMENT OF INFORMATICS

Bachelor's Thesis in Informatics

# 3D Scene Reconstruction for 2D Object Recognition

# Rekonstruktion von 3D-Szenen für 2D-Objekterkennung

| | |
|---|---|
| Author: | Kanstantsin Tkachuk |
| Supervisor: | Prof. Dr. Daniel Cremers |
| Advisors: | Nikolaus Demmel |
| | Dr. Ulrich Klank |
| Submission Date: | 16. September 2019 |

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Munich, 16. September 2019 Kanstantsin Tkachuk

# Abstract

In this thesis, I consider a part of the operational cycle of the logistics robot TORU 5 by Magazino GmbH, during which the robot's manipulator moves within a vertical plane in front of a shelf with multiple layers containing rectangular boxes which must be detected and recognized by the robot. The motion constraints allow to reduce the object recognition problem to a two-dimensional object recognition task performed on a 2D-map of the front side of the shelf containing distances from the front side to the objects in the scene.

I focus specifically on the problem of reconstructing a three-dimensional surface model of the shelf using data from a depth sensor for its subsequent reduction to a two-dimensional representation which can be used for object recognition. I implement two solution approaches to this problem and compare their performances with regard to the quality of the resulting two-dimensional representations.

The first approach called the Assisted approach uses external estimations of the sensor's position to update the surface model. The second approach called the KinectFusion approach uses the OpenCV 4.0.1 implementation of the KinectFusion algorithm [1].

To measure the quality of the reconstruction, I use pointwise matching of the sensor's trajectories reconstructed by the algorithms with ground truth trajectories as well as pixelwise matching of the resulting 2D images with ground truth images. I execute the implementations for real-world data recorded on the robots as well as for error-free synthetic data to eliminate the influence of the noise in the input data.

The results show that the Assisted approach performs better in the given setting than the KinectFusion approach which demonstrates particularly poor performance with only about 7% of the reconstructions having high enough quality to be used for object recognition. This holds both the real-world data and for the error-free synthetic data, which allows to rule out the sensor's calibration and noise in depth images as the causes for its poor performance and indicates the unsuitability of the scene's surface geometry for the reconstruction by the KinectFusion algorithm.

# Contents


**1 Introduction** **2**
- 1.1 General model of the robot's motion . . . 3
- 1.2 Real-world setting: robot TORU 5 . . . 5
- 1.3 Problem statement . . . 8

**2 Experimental setup** **11**
- 2.1 Software infrastructure of TORU 5 . . . 11
- 2.2 Surface reconstruction pipeline . . . 12
  - 2.2.1 Point cloud reconstruction . . . 14
  - 2.2.2 Transformation estimation . . . 16
  - 2.2.3 Update of the surface model . . . 17
- 2.3 Generation of two-dimensional projection . . . 19
  - 2.3.1 Formal definition . . . 20
  - 2.3.2 Implementations . . . 21

**3 Evaluation methodology** **23**
- 3.1 Test scenarios . . . 23
- 3.2 Recording of data . . . 24
- 3.3 Execution of tests . . . 25
- 3.4 Evaluation of results . . . 26

**4 Results** **29**

**5 Discussion and conclusion** **38**

# Chapter 1

# Introduction

One of the core features of any non-trivial mobile robot is the ability to reconstruct a model of its physical environment. Such a model helps the robot to localize itself and plan its movement within the environment as well as detect and classify objects available for physical interaction.

The choice of a solution approach for the problem of environment reconstruction depends greatly on the constraints imposed on the robot's movement. Those constraints may limit the robot's ability to obtain information about the environment (e.g. sensors' view limited to only one side of an object), but they may also reduce the complexity of the resulting model and, as a consequence, the computational load needed to perform its analysis (e.g. navigation of a wheeled robot in a three-dimensional environment is reduced to a two-dimensional problem).

In this thesis, I consider a particular real-world setting in which a mobile robot has to perform spatial segmentation and object recognition in a three-dimensional environment, but the constraints of the robot's motion allow to reduce this problem to an equivalent two-dimensional object recognition task. This setting is a part of the operational cycle of the robot TORU 5 by Magazino GmbH.

I focus specifically on the problem of three-dimensional environment reconstruction performed exclusively for its subsequent reduction to a two-dimensional representation. I examine two different solution approaches for this problem, implement the corresponding software systems and compare their performances with regard to the quality of the resulting two-dimensional representations and to the robustness of the implementations against the negative effects of the motion constraints such as limited sensors' view.

## 1.1 General model of the robot's motion

The robot's main motion dimensions during the examined operational phase can be generally described by a model in which a robotic manipulator with a mounted depth sensor moves in two dimensions within a vertical plane in front of a bounded three-dimensional scene containing objects to be detected by the robot (Fig. 1.1). The objects are then to be removed from the scene by the manipulator (Fig. 1.2) which extends perpendicularly to the movement plane, grasps the object with help of a suction cup and then contracts, pulling the grasped object onto the robot's storage tray (Fig. 1.3).

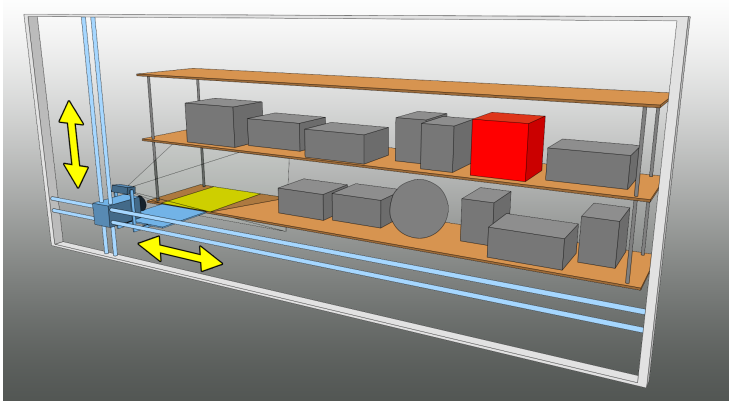


Figure 1.1: An instance of the general setting: a robotic manipulator moving within a plane defined by the support framework.

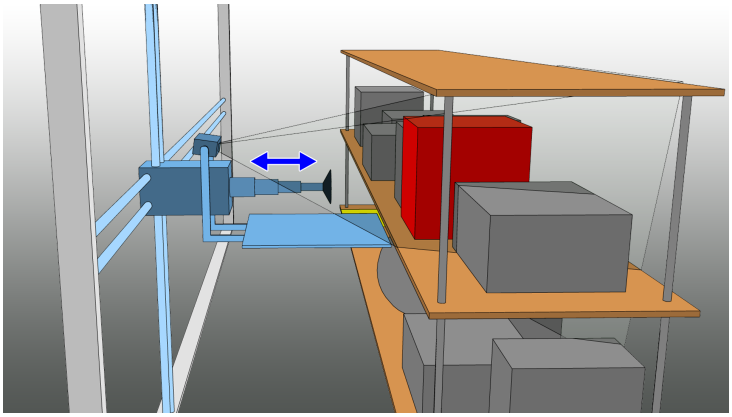


Figure 1.2: The exemplary robot has a storage tray, a depth sensor on top and a telescopic manipulator with a suction cup.

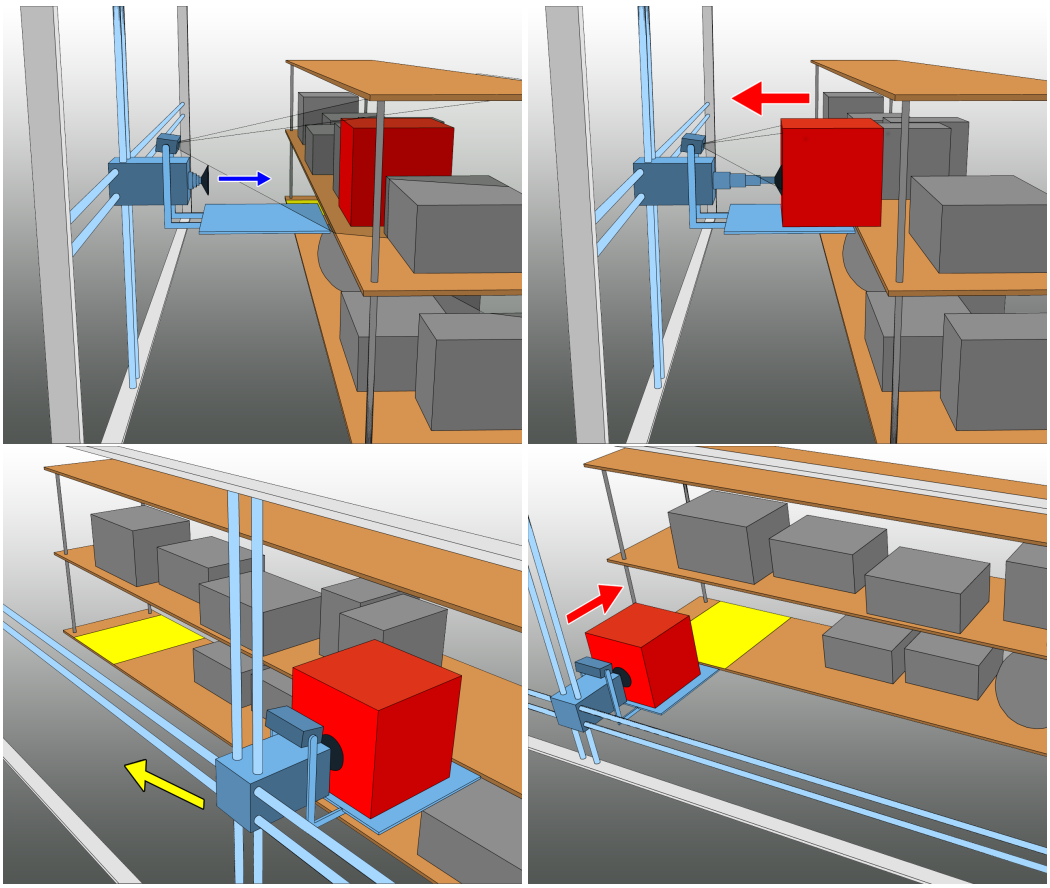
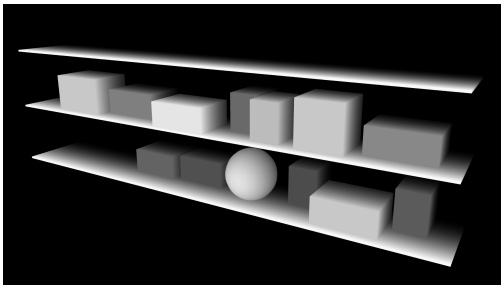
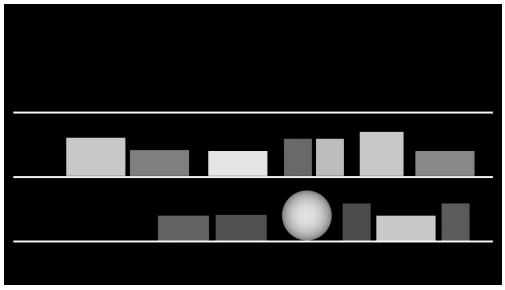


Figure 1.3: The exemplary robot transporting a box.

The positive effect of the motion constraints with regard to the problem's complexity is the possibility to reduce the three-dimensional environment to its two-dimensional projection on the plane of the robot's movement containing the perpendicular distances from the manipulator plane to the closest points of the objects within the scene (Fig. 1.4). The negative effect is however the restriction of the depth sensor's view of the scene to a single-sided view with a fixed camera angle. Most real depth sensors would only detect the frontal surfaces of the objects due to strong reflections on the side surfaces which would mean loss of information about the true shape of the objects.



(a) The surface of the objects is colored according to the distance from the robot's movement plane to the corresponding points on the surface.



(b) A 2D map of distances is built which contains all relevant information for manipulation planning and can be used for 2D object recognition.

Figure 1.4: Reduction of a 3D scene to its 2D representation

## 1.2 Real-world setting: robot TORU 5

The real-world setting examined in this thesis, which is a part of the operation cycle of the logistics robot TORU 5 by Magazino GmbH (Fig. 1.5), can be considered as an instance of the described above general setting.

TORU 5 is a wheeled mobile robot designed to locate and transport medium-sized objects, primarily shoe boxes, between different shelves in a warehouse. The process of transportation is divided into two stages: the navigation stage, during which the robot plans and executes its movement between different shelves within the warehouse, and the manipulation stage, during which the robot locates and manipulates the boxes on the shelves.

The movement pattern of the robot during the manipulation stage corresponds exactly to the described above general setting. The robot can move the whole body horizontally parallel to the shelf and can also move the manipulator vertically, resulting in a two-dimensional planar movement pattern (Fig. 1.5). To transport a box, the manipulator is extended to grasp it with help of an array of suction cups and is then contracted to pull the box onto a storage tray. The sensor used to scan the shelf is a depth sensor mounted at a fixed angle directly above the manipulator (Fig. 1.6).

The robot maintains a separate three-dimensional environment model for each shelf compartment involved in the transportation process (Fig: 1.7). These models represent the surfaces of the objects contained within their corresponding compartments.


Figure 1.5: Robot TORU 5 taking a box from the upper layer of the shelf, operating together with human workers at a Zalando warehouse. The manipulator (on top) can move vertically inside the black tower while the robot moves parallel to the shelf.

(a) Front view, gripper retracted. (b) Side view, gripper extended.

Figure 1.6: The manipulator of TORU 5, much like the exemplary robot, has a storage tray, a depth sensor (black on top) and an extending box gripper with an array of six suction cups (white in the middle).

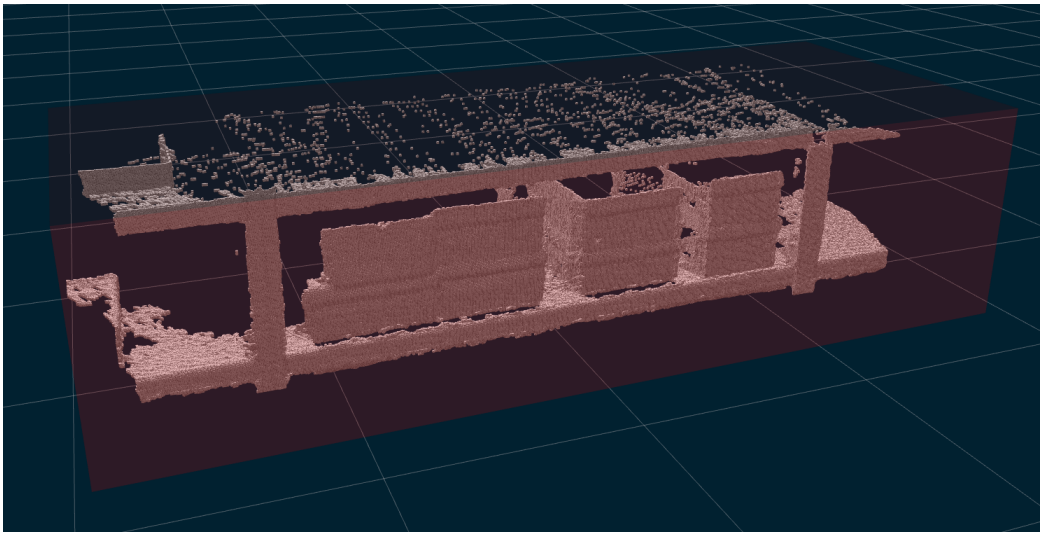


Figure 1.7: A 3D representation of a shelf's layer maintained by TORU 5.

The three-dimensional models are used to create two-dimensional projections of the contents of the compartments onto the front side of the shelf similarly to the general setting (Fig. 1.8). The resulting two-dimensional images are propagated to the object detection and manipulation planning systems which then analyze the images using fast two-dimensional object recognition algorithms.

Figure 1.8: A 2D representation of a shelf's layer generated from the 3D model.

## 1.3 Problem statement

The problem in the focus of this thesis is finding a suitable three-dimensional surface reconstruction algorithm for the described setting.

I examine two different approaches to the reconstruction of a three-dimensional scene for its subsequent reduction to a two-dimensional representation, one of which is implemented in the current version of scene reconstruction software for TORU 5 and the other one is a new experimental approach. I evaluate and compare their performances with regard to the quality of the resulting two-dimensional projections.

Both approaches discretize the volume of the scene into a three-dimensional voxel grid to accumulate information about the surfaces of the contained objects. As input, they use point clouds generated from depth images recorded by the manipulator's depth sensor. To integrate the surface information, they require estimations of the sensor's position and orientation relative to the center of the reconstructed compartment at the moment of registration of each depth image.

The differences between the approaches lie in the way those estimations are obtained and also in the way the surface data is represented using a discrete voxel grid.

The implementation of the first naive algorithm (further refered to as the **Assisted approach**) receives estimations of the sensor's pose from the robot's localization system which uses wheel odometry and lazer distance sensors to keep track of its movement. I designed and implemented this approach in the first versions of the scene reconstruction software for TORU 5, which I have been developing as a working student for Magazino GmbH.

The advantage of this approach is the simplicity of the calculations needed to update the reconstructed surface model with new depth data. When the position of the sensor is known from an external source, the update algorithm degenerates to a series of simple linear transformations of vectors representing the input point cloud. These transformations can also be calculated independently in parallel, so the algorithm works fast and adds little additional overhead to the overall computational load of the robot.

The disadvantage of this approach is obviously its complete dependence on the external sensor position estimations. The localization systems of real robots always produce noisy data, and the noise in the position estimation may actually exceed the resolution of the depth sensor resulting in the total error of the reconstruction being much greater then the error caused by the

noise in the depth data. This creates a bottleneck meaning that increasing the quality of the sensor has no effect on the quality of the surface reconstruction.

The localization system of TORU 5 uses wheel odometry which is prone to cumulative errors. The error of localization gradually grows over time until the pose estimation is refined by the data from the laser sensors and depth cameras and is replaced with a new estimation. Such sudden jumps of the robot's position, the cumulative error of the odometry and the gaussian noise in the estimation refinements all contribute to the overall error of reconstruction, since the point clouds are being inserted in wrong locations.

Another source of error is the movement of the robot which is not captured by the localization systems such as oscillation of the sensor. The sensor is mounted on a framework which moves vertically inside a hollow cylindrical tower. Due to the inertia of this framework, the tower has a tendency to oscillate when the robot's acceleration changes, with the amplitude growing larger the closer the manipulator moves to its top. Such oscillation is not registered by the robots localization systems but it leads to motion blur in the reconstructed surface and distorts the shapes of the objects.

I have observed all these effects on the robots where I have implemented this approach. This has motivated me to search for an algorithm which performs the estimation of the sensor's trajectory based only on the received depth data.

The implementation of the second approach uses the KinectFusion algorithm [1] (hence further refered to as the **KinectFusion approach**) implemented in the OpenCV Computer Vision Library of version 4.0.1 released in November 2018 to reconstruct the trajectory of the sensor using only the depth data from the sensor and from its own model. This allows to eliminate the need for an external trajectory estimation and potentially make full use of the quality and resolution of the depth sensor.

However, the performance of KinectFusion in comparison to that of the Assisted approach depends entirely on the suitability of the implementation for the particular scene scanning patterns and noise in the depth images. Scanning a scene like a rectangular shelf containing rectangular boxes only from one side with from a fixed camera angle yields depth images containing a set of disconnected mutually parallel planar shapes. It is not clear, whether the point clouds generated from such depth images can be matched to one another with a sufficient accuracy and how significant is the impact of the noise in the depth images on the reconstruction quality.

In order to evaluate and compare the performance of the examined surface

reconstruction approaches, I create working implementations of both algorithms and integrate them into the scene reconstruction software of TORU 5. I use the adapted implementation from the current version of the software as the Assisted approach implementation and also implement its extended version where the assisted surface reconstruction pipeline is replaced with the OpenCV implementation of the KinectFusion algorithm. In addition, I implement a testing framework which allows to automatically execute the algorithms on recorded datasets for multiple combinations of customizable parameters and quantitatively evaluate the results of the execution.

# Chapter 2

# Experimental setup

## 2.1 Software infrastructure of TORU 5

The robot TORU 5 is operated by the Robot Operating System (ROS). ROS regulates the interaction between separate software processes called **nodes**. The data exchange between the nodes is performed through channels called **topics**. Each node can communicate data to other nodes by publishing messages on a number of topics and can receive data from other nodes by subscribing to topics they are publishing on. Nodes can also communicate data on demand by offering a **service** that other nodes can call. By calling a service, a node sends a request to another node to perform some actions or to send some data back to the calling node. A service call can also be used to communicate input data to the node being called.

On TORU 5, the surface reconstruction and aggregation of surface data for multiple shelf compartments is performed by the node called `scene_aggregator` (Fig. 2.1). This node receives the calibrated depth data from the sensor by subscribing to the topics `/camera_info` and `/image_rect` published by the sensor's driver node.

The implementation of the Assisted approach also requires information about the transformations between different coordinate systems corresponding to various joints of the robot and objects in the environment such as shelves and compartments. The transformation data is published by the localization nodes on the topics `/tf` and `/tf_static`, which the `scene_aggregator` node reads to obtain the transformation matrices necessary for the sensor's position estimation.

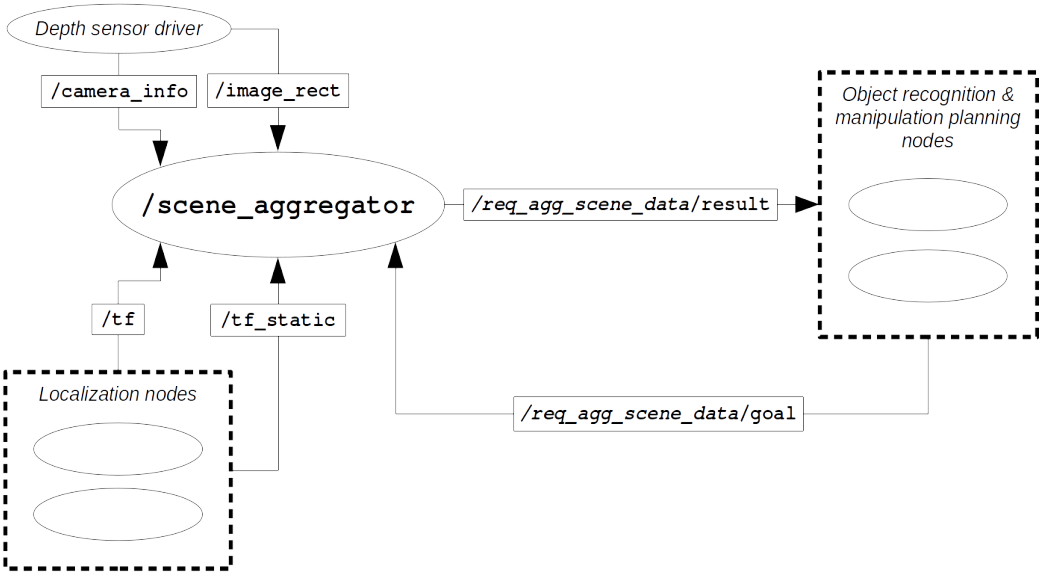


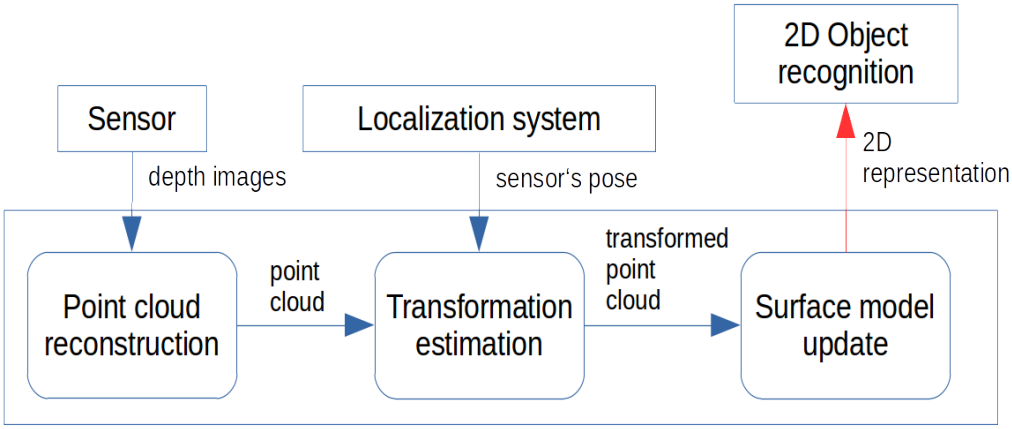

Figure 2.1: Graph of the ROS-infrastructure of the scene reconstruction pipeline.

The `scene_aggregator` node offers a service called `req_agg_scene_data`, which upon request generates the two-dimensional projection of the surface and sends it back to the calling node. This service is called by the box detection nodes when manipulation planning has to be performed. The box detection nodes receive the image from `scene_aggregator` as a response to the request, perform scene segmentation and object detection and then propagate the results to further nodes responsible for manipulation planning.

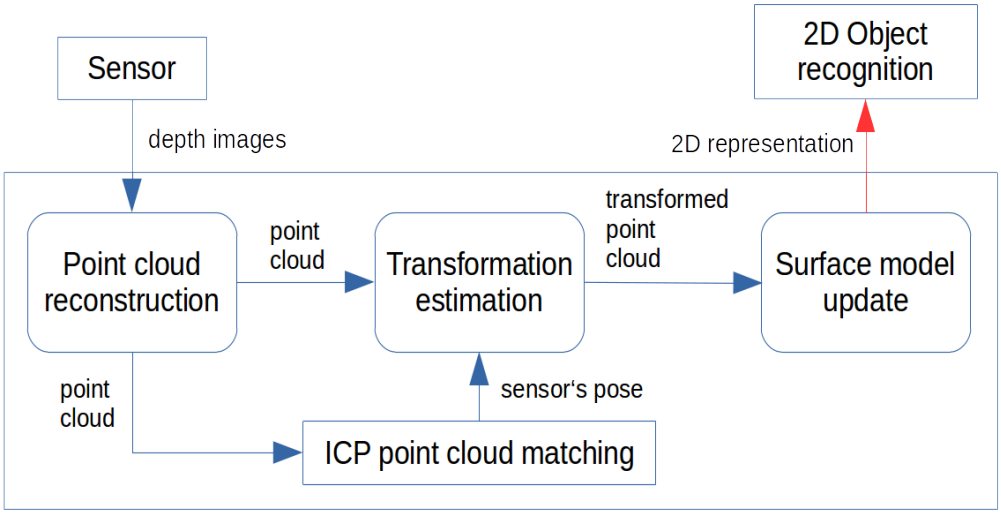
## 2.2 Surface reconstruction pipeline

The surface reconstruction process is performed in cycles with each new iteration triggered by the incoming depth images. An iteration is divided into three steps: reconstruction of the point cloud from the depth image in its own coordinate system, calculation of the transformation from the coordinate system of the point cloud to the coordinate system of the reconstructed compartment and the integration of the point cloud into the surface model of the compartment (Fig. 2.2). These steps are common for both reconstruction approaches, but the implementations of the last two steps are different and

require different data.




(a) Assisted approach.




(b) KinectFusion approach.

Figure 2.2: Surface reconstruction pipeline.

In case of the KinectFusion approach, the whole reconstruction pipeline is implemented internally in the module `"rgbd"` of the OpenCV library (more specifically, in the `opencv_contrib` repository for OpenCV's extra modules). This module contains the class `KinFu` which offers a single method to perform all the reconstruction steps above at once.

The `scene_aggregator` node maintains a single object of the class `KinFu` for each active compartment and only needs to call the method `"update"` of this object with a depth image as argument in order to update its internal surface model according to the described pipeline. The resulting reconstructed surface can be extracted from the `KinFu` object in form of a point cloud with normals which then can be used for the three-dimensional visualization of the surface or for the generation of its two-dimensional projection.

The `KinFu` object is intentionally considered as a black box implementation of the KinectFusion algorithm for compatibility reasons. In order to be able to use future binary releases of the OpenCV library for surface reconstruction on the robots, the code of the OpenCV implementation must not be altered in any way and should only be viewed in terms of input-output relations. For this reason, the particular implementation of the KinectFusion algorithm in the chosen version of the OpenCV library is not examined in this thesis, and the relevant parts of the algorithm are described in a general way as in the original paper [1]. For the same reason, the KinectFusion approach uses the point cloud representation of the reconstructed scene, which can be obtained by calling a method of the `KinFu` object, for the two-dimensional projection generation, although this process could be implemented more efficiently if the direct access to the internal surface representation of the `KinFu` object was possible.

### 2.2.1 Point cloud reconstruction

The surface reconstruction pipeline begins with the two depth data topics. The topic `/image_rect` is used to communicate the actual depth images and the topic `/camera_info` communicates information about the used depth sensor such as intrinsic calibration matrix and distortion parameters. The data is communicated in form of standard ROS messages which are generated and published in real time by the sensor's driver node.

Both message types begin with a standard ROS header which contains a sequence number, a timestamp and the name of the coordinate system the data is to be interpreted in. The timestamps are used by the `scene_aggregator` node to synchronize the messages on both topics and, along with the coordinate system names, to match each depth image to the sensor's position in space at the time the depth image was recorded.

The depth images are stored in the messages from the `/image_rect` topic as raw binary data which, using some auxiliary information also stored in the

message, can be converted to two-dimensional images representing pinhole-camera-model projections of the scene within the field of view of the sensor (Fig. 2.3). These projections can in turn be converted to three-dimensional point clouds approximating the source scene surface as follows.

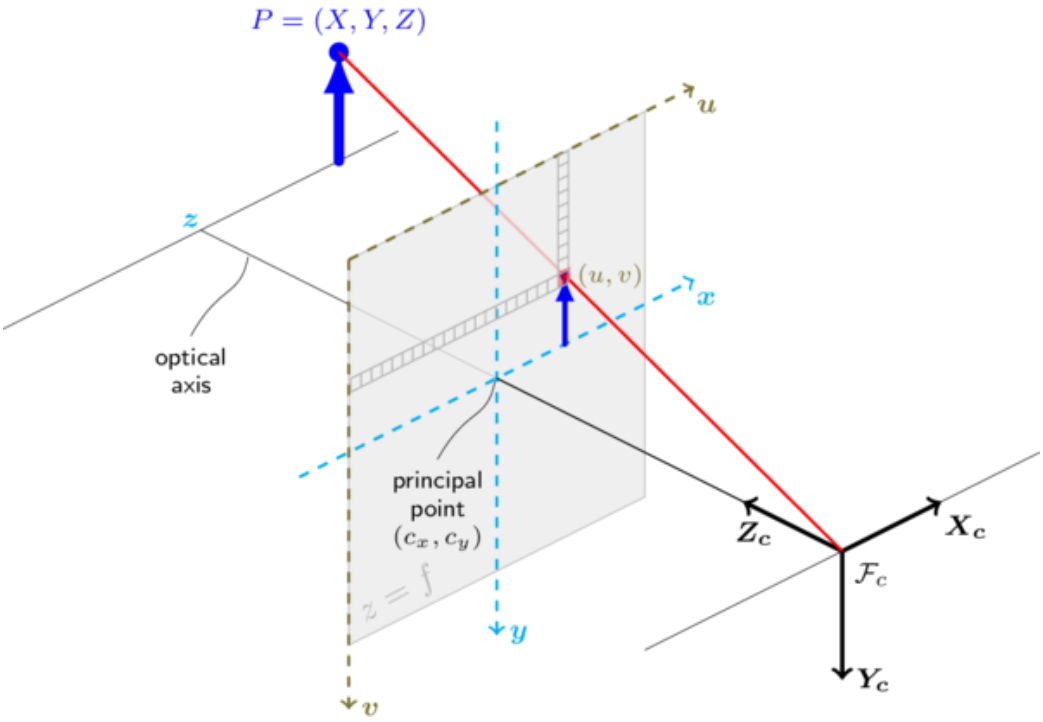



Figure 2.3: Projection of a point in the scene onto the image plane in the pinhole camera model (source: OpenCV documentation, docs.opencv.org).

The coordinates of the points relative to the focal point can be calculated using the focal lengths and the principal point of the model. This information is stored within the intrinsic calibration matrix $K$ in the messages from the `/camera_info` topic which has the following form:

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

where $f_x$ and $f_y$ are the focal lengths and $(c_x, c_y)^T$ is the principal point. The coordinates of the point $p(u, v) = (p_x, p_y, p_z)^T$ represented by the pixel

in the row $u$ and column $v$ of the depth image $D$ with value $d(u, v)$ can then be calculated as follows:

$$p(u,v) = d(u,v) \cdot K^{-1} \cdot \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} \frac{d(u,v)}{f_x}(u - c_x) \\ \frac{d(u,v)}{f_y}(v - c_y) \\ d(u,v) \end{pmatrix} \tag{2.2}$$

The resulting set of vectors:

$$P = \{p(u,v) \mid (u,v) \in D\} \tag{2.3}$$

represents the reconstructed point cloud with coordinates of the points given in the sensor's coordinate system or **frame** which is identified by the name or **frame ID** stored in the headers of the messages. The sensor's focal point is located at the origin of this system and the direction of the $z$-axis is the direction of view of the sensor. The $x$ and $y$ axes match with the corresponding axes of the image (Fig. 2.3).

### 2.2.2 Transformation estimation

In the second step of the reconstruction pipeline, the coordinates of the points must be transformed from the sensor's frame to the frame of each active compartment to make the integration of the surface information into the models possible.

The frames of the compartments are defined to have origin at the centers of the compartments with axis $z$ pointing upwards and axes $y$ and $x$ pointing respectively towards the front and the right side of the compartment relative to the robot standing in front of the shelf.

The borders of the compartment can be defined in its coordinate system as planes $x = \pm x_{end}$, $y = \pm y_{end}$ and $z = \pm z_{end}$ where $x_{end}$, $y_{end}$ and $z_{end}$ are half the width, depth and height of the compartment. The front side of the compartment is then described by the plane $y = +y_{end}$.

At this point, the two approaches diverge and follow different paths.

In the Assisted approach, the `scene_aggregator` node reads the information about the transformations from the topics `/tf` and `/tf_static` using a specialized buffer object which stores the received messages and provides a method to calculate the transformation between any two existing frames for a given point in time in present or recent past. The point in time for which

a given transformation is valid is specified by the timestamp in the header of a transformation message.

The `scene_aggregator` node reads the timestamps and the sensor's frame ID from the headers of the depth image messages and calls the transformation lookup method provided by the buffer, passing the timestamp, the sensor's frame ID and the frame ID of each compartment as arguments. The data structures returned by the transformation buffer desribe the transformations from the sensor's frame to each compartment's frame at the moment of depth image registration. Each structure is then multiplied with the vectors of the point cloud to obtain one new point cloud for each compartment with the coordinates of the points specified in the compartment's coordinate system.

The KinectFusion approach also uses the transformation data published on `/tf` and `/tf_static` to calculate the starting position of the sensor which will be used as the reference point for the sensor's trajectory estimation by the KinectFusion algorithm. Since the direction of the $z$-axis in the sensor's frame coincides with the sensor's direction of view, the tranformation from the sensor's frame to the compartment's frame, which can be described by the translation vector and the rotation quaternion, can also be interpreted as the pose of the sensor in the compartment's frame. The translation vector is then equivalent to the sensor's position and the rotation quaternion is equivalent to the sensor's orientation in the compartment's frame.

Upon receiving the first input point cloud, the KinectFusion algorithm calculates the corresponding transformation using the data from the transformation topics, stores it as the current pose of the sensor and updates its surface model with the received point cloud. The following sensor poses for each incoming point cloud will be calculated by matching the point clouds with the reconstructed surface and calculating the estimation of the sensor's movement in the time between the two last depth image registrations. This will reqiure no further information from the transformation topics. To integrate the point clouds into the surface model, the sensor poses can simply be interpreted as the coordinate system transformations and multiplied with the vectors of the point clouds.

### 2.2.3 Update of the surface model

The last step of the reconstruction process is to update the surface models using the obtained point clouds.

For both approaches, the volume inside the compartment is discretized

into cubes of fixed size forming a uniform voxel grid $V$, where each voxel is assigned a single numerical value describing the way it intersects with the reconstructed surface:

$$V := (v_{i,j,k}) \in \mathbb{R}^{[i_{min}\dots i_{max}]\times[j_{min}\dots j_{max}]\times[k_{min}\dots k_{max}]}, \tag{2.4}$$

$$i_{min} = \lfloor \frac{-x_{end}}{v} \rfloor, i_{max} = \lfloor \frac{x_{end}}{v} \rfloor, \tag{2.5}$$

$$j_{min} = \lfloor \frac{-y_{end}}{v} \rfloor, j_{max} = \lfloor \frac{y_{end}}{v} \rfloor, \tag{2.6}$$

$$k_{min} = \lfloor \frac{-z_{end}}{v} \rfloor, k_{max} = \lfloor \frac{z_{end}}{v} \rfloor, \tag{2.7}$$

where $v$ is the size of voxel, $x_{end}$, $y_{end}$ and $z_{end}$ are half the width, depth and height of the compartment. The voxels are identified by three integer indices $i$, $j$ and $k$ corresponding to each dimension starting at the origin of the compartment's frame which lies at the center of the compartment (so negative indices are also used).

In case of the Assisted approach, the voxel's value is the probability that the surface intersects with this specific voxel, i.e. that the voxel contains some points which lie on the surface of some object. The algorithm to update such voxel grid with a point cloud is straightforward (Algorithm 1): the coordinates of each point are divided by the size of a voxel and are rounded downwards to obtain the integer indices of the corresponding voxel in three dimensions. The value of this voxel is then modified using a certain update function $f(v_{i,j,k}, \boldsymbol{\theta})$ depending on the current value of the voxel and on a vector of parameters $\boldsymbol{\theta}$.

**Data:** voxel grid $V$ (eq. 2.4), voxel size $v$, point cloud $P$ (eq. 2.3)
**Result:** voxel grid $V$ is updated with new data from $P$
**foreach** $p = (p_x, p_y, p_z) \in P$ **do**
  $(i, j, k) \leftarrow \left(\lfloor \frac{p_x}{v} \rfloor, \lfloor \frac{p_y}{v} \rfloor, \lfloor \frac{p_z}{v} \rfloor\right)$;
  $v_{i,j,k} \leftarrow f(v_{i,j,k}, \boldsymbol{\theta})$;
**end**

**Algorithm 1:** Update algorithm for the Assisted approach.

The update function $f(v_{i,j,k}, \boldsymbol{\theta})$ and the parameter vector $\boldsymbol{\theta}$ can be chosen depending on the properties of the input data and the scene being reconstructed. For instance, assuming a certain probability distribution for the

coordinates of the points, the neighboring voxels can be updated along with the target voxel accoring to the probability density function of the given distribution. In the current implementation, the update function for the voxel values is trivial: if any point is found to lie within the given voxel, its intersection probability is set to one right away, and the voxel is considered "occupied" ($f(v_{i,j,k}, \boldsymbol{\theta}) = const = 1$).

The surface of all the occupied voxels approximates the surface of the objects inside the compartment for the purposes of three-dimensional visualization as well as generation of the two-dimensional projection of the scene.

In the KinectFusion algorithm, the values of the voxels are calculated using the Truncated Signed Distance Functions (TSDF) approach as described in the KinectFusion paper [1]. Such representation can potentially allow a more accurate approximation of the reconstructed surface while using a voxel grid of the same resolution as in the Assisted approach.

The reconstructed surface can be visualized using the point cloud generated by the KinectFusion algorithm from the voxel values using raycasting. It can be obtained from the `KinFu` object by calling the corresponding method. This point cloud is also used to generate the two-dimensional projection of the scene.

## 2.3 Generation of two-dimensional projection

The generation of two-dimensional projections is done by the `scene_aggregator` node on demand when the manipulation planning has to be performed.

To plan the next manipulation action, the manipulation nodes require the information about the detected objects in the scene, which is provided by the box detection node, which in turn requires a two-dimensional representation of the inspected scene to perform object recognition. To obtain this representation, the box detection node calls the service `req_agg_scene_data` provided by `scene_aggregator`. Upon receiving a request, the `scene_aggregator` node temporarily suspends the reconstruction process to avoid concurrent modifications of the reconstructed surface and generates the requested representation to then send it back to the calling node.

### 2.3.1 Formal definition

The two-dimensional representation of the compartment is constructed taking into account the robot's movement constraints. Since the manipulator can only move within a vertical plane in front of the compartment and extend and contract perpendicularly to this plane, the only information about the objects inside the compartment that is relevant for manipulation planning is the sideways shift and the elevation of the object relative to the center of the compartment (to know where to position the manipulator inside the movement plane) and the distance from the manipulator's default unextended position to the frontal surface of the object to be grasped (to calculate how far the manipulator must be extended to perform grasping).

The latter distance can be calculated using the distance between the object and the front side of the compartment by just adding a constant, since the robot always stays at the same distance from the shelf. For this reason, the two-dimensional representation of the compartment has to be a projection of the contents of the compartment onto its front side. Since only the frontal surfaces of the objects are relevant, the projection has to contain information about the distances from the compartment's front side to the surfaces of the objects inside the compartment.

In our setting, this projection can be formally defined by using an ideal continuous representation of the three-dimensional volume inside the compartment. The continuous volume itself is represented by the following set:

$$V^{\mathcal{C}} := [-x_{end}, x_{end}] \times [-y_{end}, y_{end}] \times [-z_{end}, z_{end}] \subset \mathbb{R}^3 \tag{2.8}$$

which is a subset of $\mathbb{R}^3$ containing its zero vector and bounded by the planes $x = \pm x_{end}$, $y = \pm y_{end}$ and $z = \pm z_{end}$. The front side is defined by the plane $y = +y_{end}$.

Each object within the compartment can now be defined as a distinct set of points $O \in V$. The contents of the compartment can then be represented by the following binary function:

$$occ : V^{\mathcal{C}} \to \{0, 1\} \tag{2.9}$$

$$occ : (x, y, z) \mapsto \begin{cases} 1, & \text{if } \exists O \mid (x, y, z) \in O; \\ 0, & \text{otherwise.} \end{cases} \tag{2.10}$$

Using this representation, the projection $d(x', y')$ of the scene onto the front side of the compartment can be described by the following function:

$$d : [-x_{end}, x_{end}] \times [-z_{end}, z_{end}] \to \mathbb{R} \tag{2.11}$$

$$d : (x', y') \mapsto \min_{\forall y \in [-y_{end}, y_{end}] : occ(x', y, y') = 1} (y_{end} - y) \tag{2.12}$$

This projection represents the distances from each point on the front side of the compartment to the closest point in the y-direction which lies on the surface of some object.

### 2.3.2 Implementations

In the real-world setting, a rectangular compartment is discretized into a uniform rectangular voxel grid $V$ (2.4). In this case of a discrete space, the front side of the volume is also a discrete two-dimensional structure, namely a layer of voxels $V' = (v_{i,j_{max},k}) = (v'_{i,k})$, so the projection will have form of a two-dimensional array $D = (d_{i',j'})$ with each element representing the distances from each individual voxel on the front side (e.g. from a single fixed point or a surface within the voxel) to the part of the reconstructed surface within the same voxel row in $y$-direction:

$$D = (d_{i',j'}) \in \mathbb{R}^{[i'_{min} \dots i'_{max}] \times [j'_{min} \dots j'_{max}]}, \tag{2.13}$$

$$\begin{aligned} i'_{min} &= 0, \;\; i'_{max} = i_{max} - i_{min} \;\; \text{(from 2.5)}, \\ j'_{min} &= 0, \;\; j'_{max} = k_{max} - k_{min} \;\; \text{(from 2.7)}. \end{aligned}$$

Since the voxel grid representations of the surface are different in both approaches, those distances are also calculated in a different way.

In the Assited approach, the calculation algorithm iterates over all voxels of the front side of the compartment (Algorithm 2). Starting at each voxel, the algorithm then iterates over the voxels within the same $y$-row until it reaches a voxel marked as occupied. The resulting distance is then defined as the distance between the front sides of the front voxel and the occupied voxel or, equivalently, the length of a voxel multiplied by the number of

inspected unoccupied voxels in the current $y$-row. The distances are written into the corresponding pixels of a two-dimensional image representing the resulting projection.

**Input:** voxel grid $V$ (2.4), voxel size $v$
**Output:** projection $D$ (2.13)
**for** $i \leftarrow i_{min}$ **to** $i_{max}$ **do**
  **for** $k \leftarrow k_{min}$ **to** $k_{max}$ **do**
    $j \leftarrow j_{max}$;
    **while** $v_{i,j,k} = 0$ **and** $j \geq j_{min}$ **do**
      $j \leftarrow j - 1$;
    **end**
    $(i', j') \leftarrow (i - i_{min}, k - k_{min})$;
    $d_{i',j'} \leftarrow v \cdot (j_{max} - j)$;
  **end**
**end**

**Algorithm 2:** Generation of a 2D projection in the Assisted approach.

In the KinectFusion approach, the algorithm iterates over the points of the point cloud $P$ returned by the corresponding method of the `KinFu` object (Algorithm 3). It starts with an image filled with the distances to the back side of the compartment, representing an empty scene with no objects. Then, for each point $(p_x, p_y, p_z)$, it calculates the coordinates of the pixel this point corresponds to using integer division and stores the difference $j_{max} \cdot v - p_y$ as the value of this pixel if this value is less than the current value of the pixel. When the calculation is done, each pixel will contain the smallest of the distances to all surface points registered within the corresponding voxel row.

**Input:** voxel grid $V$ (2.4), voxel size $v$, point cloud $P$
**Output:** projection $D$ (2.13)
$d_{i',j'} \leftarrow v \cdot (j_{max} - j_{min})\ \forall (i', j')$;
**forall** $p = (p_x, p_y, p_z) \in P$ **do**
  $(i, j, k) \leftarrow (\lfloor \frac{p_x}{v} \rfloor, \lfloor \frac{p_y}{v} \rfloor, \lfloor \frac{p_z}{v} \rfloor)$;
  $(i', j') \leftarrow (i - i_{min}, k - k_{min})$;
  **if** $(j_{max} \cdot v - p_y) < d_{i',j'}$ **then** $d_{i',j'} \leftarrow (j_{max} \cdot v - p_y)$;
**end**

**Algorithm 3:** Generation of 2D projection in the KinectFusion approach.

# Chapter 3

# Evaluation methodology

To ensure accurate results and create a fully controlable testing environment, the software implementing the examined algorithms is not being run directly on the robots, but is instead executed in offline mode with the specifically recorded input data and on the hardware similar to that installed on the real robots. The data used for testing is recorded on the actual robots operating in simplified but close-to-reality test scenarios. Along with the real-world data, the algorithms are also executed on a set of data generated by the simulation of TORU 5 in the Gazebo robot simulator. The synthetic data is free of noise and thus is used to examine the effects of the noise on the surface reconstruction quality.

## 3.1 Test scenarios

In the test setting, I simulate the process of scanning an entire compartment with TORU 5 moving sideways parallel to the shelf from one edge of the compartment to another with a constant velocity. Such movement pattern often occurs in real-world scenarios and is well suitable for the surface reconstruction testing since it delivers the largest amount of depth data for a given compartment and also introduces the largest possible error to the reconstructed model.

This error is caused by the error of the sensor's trajectory estimation. In case of the Assisted approach, this includes the cumulative error of odometry, which is either accumulated over the whole length of the path or combined with interruptions and leaps in the trajectory caused by sudden changes

of the position estimation after refinement. In case of the KinectFusion approach, it is due to the fact that the estimation for each new depth image is calculated relatively to the previous pose estimations, so the point cloud matching error is accumulated over time increasing the total estimation error after each iteration.

The starting position of the robot is in front of one of the support pillars of the shelf adjacent to the target compartment. The tower of the robot is open and the manipulator is directed towards the shelf and positioned at the necessary height to fit the whole height of the compartment into the field of view of the sensor. The robot starts to move parallel to the shelf towards the opposite side of the compartment. Shortly after it reaches the target velocity, the recording of the data starts. The velocity stays constant during the whole scanning process. As soon as the robot reaches the opposite side of the compartment and the second support pillar passes through the center of the sensor's field of view, the recording of the data ends, and the robot stops.

For the described robot movement pattern, I consider 7 types of box arrangements within the test compartment:

- emtpy compartment,
- one box in the middle,
- stack of two boxes in the middle,
- two boxes placed uniformly,
- two stacks of two boxes placed uniformly,
- four boxes placed uniformly,
- four stacks of two boxes placed uniformly.

These types range by filling density, which is necessary to investigate the influence of the distances between the objects in the scene on the quality of the reconstruction. Larger distances may lead to more similar depth images in a row which, in case of the KinectFusion approach, may have negative effects on the accuracy of sensor's movement estimation between the frames.

## 3.2 Recording of data

The data being recorded during the scanning are all the messages published on the topics `/camera_info`, `/image_rect`, `/tf` and `/tf_static`. The

sequences of messages published on these four topics together with their exact timelines are sufficient to completely replicate the scanning process in terms of the input received by the surface reconstruction node.

The recording is performed using the ROS's "`rosbag`" utility for recording from and playing back to ROS topics. Using this utility to play back the recorded messages, the reconstruction process can be simulated on any system that has the necessary hardware and supports ROS. The `scene_-aggregator` node can be started in isolation and perform the reconstruction while reading the data from the simulated topics and providing the two-dimensional projections on demand.

The process of data recording performed in the real-world setting with the decribed robot movement pattern is also performed in exactly the same way in the simulated environment. The simulated warehouse including the scanned compartment exactly replicates the actual robot testing area used by Magazino GmbH and both the real TORU 5 and the simulated one are controlled by the same software. The depth sensor used in the real robot is simulated using the depth camera plugin for Gazebo and is set up to deliver depth images of the same resolution as the real sensor but without any distortion. The transformation information published on the topics `/tf` and `/tf_static` is taken directly from Gazebo and is therefore also error-free and represents the true geometric relationships between different parts of the robot.

## 3.3 Execution of tests

With the test scenarios and box arrangement types defined as described above, the complete dataset used to test the surface reconstruction algorithms consists of a single "`rosbag`" recording of the specified ROS topics for each combination of the following parameters:

- real/simulated robot,
- movement direction left/right,
- box arrangement type 1 to 7

with the total of 28 recordings. Each recording has to be processed by the implementations of the both algorithms coming in multiple variations.

The implementations can be customized using a number of parameters. The common parameter for both algorithms is the resolution of the internal voxel grid representation. The implementation of the KinectFusion algorithm also offers three more cusomizable parameters: the raycasting step, the number of levels in the multi-resolution-pyramid representation of the depth image used in ICP algorithm and the number of iterations of the ICP algorithm at each pyramid level. Apart from that, the depth data can be downsampled by various factors by discarding a certain fraction of the depth images to simulate lower frame rates of the sensor or higher velocities of the robot.

Each combination of these parameters results in different behavior, so I execute the reconstruction once for each combination of the following values:

- voxel grid resolution: 2mm/3mm/4mm/5mm,
- downsampling factor: 1/2/4/8 (each 1/2/4/8-th depth image processed, the rest discarded),
- (KinFu) raycasting step factor: 0.25 (single value),
- (KinFu) number of pyramid levels: 3 (single value),
- (KinFu) number of iterations at each pyramid level: 5-3-2, 10-6-4, 15-9-6, 20-12-8 (level order from fine to coarse),

for each of the 28 data recordings, resulting in 448 executions of the Assisted approach implementation and 1792 executions of the KinectFusion approach implementation.

## 3.4 Evaluation of results

In the test scenarios during the recording of the depth data the robot moves with a constant velocity in a straight line in front of the compartment. I use such a trajectory to provide an initial measure of quality of the reconstruction which is the deviation of the estimated sensor's trajectory reconstructed by the algorithms from a straight line path.

In the beginning and in the end of the scanning process the testing framework requests and stores additional estimations of the sensor's pose from the localization nodes. These estimations are calculated using wheel odometry and refined with measurements from laser distance sensors with an accuracy

of several centimeters, while the average total length of the robot's path is approximately 1.5 meters, which means that a straight line trajectory of the sensor can be approximated by a line spanned by the endpoints with a reasonable accuracy.

The ground truth trajectories for each test scenario are then defined as straight line paths between the initial and the final position of the sensor. The intermediate positions on the trajectory for any given moment between the starting time and the stopping time are calculated using linear interpolation with sensor's pose coordinates as functions of time.

The ground truth trajectories are compared with the trajectories reconstructed by the implementations of the surface reconstruction pipeline.

The reconstructed trajectories are represented by series of sensor poses with timestamps corresponding to each processed depth image. After each model update cycle, the elements of the position vector and the orientation quaternion of the newly calculated pose estimate are written to a file along with the timestamp with the time of the depth image registration.

I consider these entries separately and first calculate the corresponding ground-truth pose for each resonstructed pose using linear interpolation with the timestamp as argument. I then compute the differences between the positions of the sensor and use the absolute trajectory error (ATE) or the root-mean-square-error (RMSE) of the lengths of the difference vectors as the quantitative measure for the dissimilarity of the trajectories.

I calculate the ATE for different implementations of the surface reconstruction pipeline working in the same test scenarios and use the values to compare the performance of the implementations and the influence of different parameters on the quality of the reconstruction.

In case of the KinectFusion approach, the evaluation of the trajectories is performed both on the real-world data and on the synthetic data. The latter is not distorted by the noise as the data from real-world sensors and is therefore used to investigate the performance of the algorithm in a pure error-free setting.

However, in case of the Assisted approach, the absense of error both in the depth data and the transormation data means that the trajectory reconstructed by the implementation of the algorithm will always exactly correspond to the actual straight line trajectory of the robot in the simulation environment. As a result, the models reconstructed by the Assisted approach implementation cannot be evaluated using the trajectory estimation in the simulated setting, but can instead be used as a benchmark for

the KinectFusion approach.

The projections produced by the Assisted approach implementation from the error-free data actually represent the ideal approximations of the shapes of the objects inside the compartment for the given discretization resolution and can therefore be used as the ground truth data for the evaluation of the images produced by the KinectFusion implementation. The images are compared pixelwise with each pair of pixels being marked as 'divergent' if their values differ by more than ten centimeters. The quantitative measure of the reconstruction quality for the KinectFusion image is then defined as the percentage of divergent pixels between the reconstructed image and the corresponding ground truth image (obviously, with smaller numbers meaning better quality).

# Chapter 4

# Results

The tables 4.1 to 4.6 show the results of the trajectory evaluation for the Assisted approach (columns "AA") and for the KinectFusion approach ("KinFu") for different test scenarios ordered by different values of the customizable parameters of the KinectFusion implementation.

The tables 4.7 to 4.9 show the average pixel mismatch ratios between the images reconstructed using KinectFusion from the synthetic data and the ground truth images. Figure 4.1 shows their overall distribution with the average of 0.2245 and the standard deviation of 0.077. The figures 4.2 and 4.3 show the images with the lowest and highest mismatch ratios respectively.

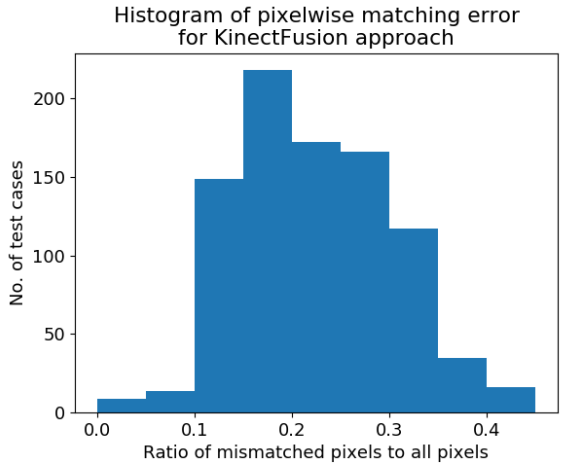



Figure 4.1: Results of pixelwise matching of KinectFusion images with ground truth images.

| | 2 mm | | 3 mm | | 4 mm | | 5 mm | |
|---|---|---|---|---|---|---|---|---|
| | AA | KinFu | AA | KinFu | AA | KinFu | AA | KinFu |
| Empty | 0.002 | 0.467 | 0.003 | 0.150 | 0.002 | 0.127 | 0.010 | 0.059 |
| 1 box | 0.002 | 0.489 | 0.003 | 0.117 | 0.002 | 0.127 | 0.010 | 0.089 |
| 1 stack | 0.002 | 0.467 | 0.003 | 0.095 | 0.003 | 0.097 | 0.009 | 0.125 |
| 2 boxes | 0.002 | 0.445 | 0.003 | 0.160 | 0.003 | 0.102 | 0.003 | 0.070 |
| 2 stacks | 0.003 | 0.075 | 0.003 | 0.231 | 0.003 | 0.118 | 0.003 | 0.072 |
| 4 boxes | 0.003 | 0.097 | 0.002 | 0.085 | 0.003 | 0.130 | 0.003 | 0.078 |
| 4 stacks | 0.003 | 0.118 | 0.002 | 0.080 | 0.010 | 0.048 | 0.003 | 0.100 |
| Average | 0.002 | 0.308 | 0.003 | 0.131 | 0.004 | 0.107 | 0.007 | 0.085 |

Table 4.1: Average ATE ordered by the resolution, real-world data.

| | 5-3-2 | | 10-6-4 | | 15-9-6 | | 20-12-8 | |
|---|---|---|---|---|---|---|---|---|
| | AA | KinFu | AA | KinFu | AA | KinFu | AA | KinFu |
| Empty | 0.001 | 0.627 | 0.001 | 0.554 | 0.001 | 0.381 | 0.012 | 0.383 |
| 1 box | 0.001 | 0.602 | 0.021 | 0.374 | 0.001 | 0.378 | 0.012 | 0.397 |
| 1 stack | 0.001 | 0.541 | 0.021 | 0.399 | 0.025 | 0.425 | 0.012 | 0.364 |
| 2 boxes | 0.001 | 0.617 | 0.019 | 0.407 | 0.023 | 0.383 | 0.022 | 0.357 |
| 2 stacks | 0.001 | 0.506 | 0.020 | 0.371 | 0.024 | 0.403 | 0.022 | 0.348 |
| 4 boxes | 0.001 | 0.528 | 0.001 | 0.432 | 0.025 | 0.360 | 0.021 | 0.350 |
| 4 stacks | 0.001 | 0.547 | 0.001 | 0.394 | 0.012 | 0.410 | 0.022 | 0.346 |
| Average | 0.001 | 0.567 | 0.012 | 0.419 | 0.016 | 0.391 | 0.017 | 0.364 |

Table 4.2: Average ATE ordered by the number of iterations, real-world data.

| | 1 in 1 | | 1 in 2 | | 1 in 4 | | 1 in 8 | |
|---|---|---|---|---|---|---|---|---|
| | AA | KinFu | AA | KinFu | AA | KinFu | AA | KinFu |
| Empty | 0.001 | 0.464 | 0.000 | 0.829 | 0.001 | 0.601 | 0.011 | 0.272 |
| 1 box | 0.001 | 0.475 | 0.020 | 0.167 | 0.000 | 0.711 | 0.004 | 0.566 |
| 1 stack | 0.000 | 0.747 | 0.019 | 0.246 | 0.024 | 0.125 | 0.001 | 0.627 |
| 2 boxes | 0.000 | 0.699 | 0.020 | 0.371 | 0.023 | 0.160 | 0.024 | 0.109 |
| 2 stacks | 0.001 | 0.380 | 0.022 | 0.767 | 0.023 | 0.616 | 0.024 | 0.265 |
| 4 boxes | 0.001 | 0.175 | 0.001 | 0.111 | 0.026 | 0.671 | 0.018 | 0.503 |
| 4 stacks | 0.001 | 0.752 | 0.001 | 0.162 | 0.030 | 0.089 | 0.021 | 0.523 |
| Average | 0.001 | 0.527 | 0.012 | 0.379 | 0.018 | 0.425 | 0.015 | 0.409 |

Table 4.3: Average ATE ordered by the downsampling rate, real-world data.

| | 2 mm | | 3 mm | | 4 mm | | 5 mm | |
|---|---|---|---|---|---|---|---|---|
| | AA | KinFu | AA | KinFu | AA | KinFu | AA | KinFu |
| Empty | 0.001 | 0.641 | 0.001 | 0.535 | 0.001 | 0.397 | 0.012 | 0.398 |
| 1 box | 0.001 | 0.610 | 0.021 | 0.355 | 0.001 | 0.378 | 0.012 | 0.346 |
| 1 stack | 0.001 | 0.608 | 0.020 | 0.391 | 0.024 | 0.394 | 0.012 | 0.418 |
| 2 boxes | 0.001 | 0.526 | 0.020 | 0.380 | 0.023 | 0.393 | 0.023 | 0.346 |
| 2 stacks | 0.001 | 0.528 | 0.020 | 0.425 | 0.024 | 0.385 | 0.019 | 0.353 |
| 4 boxes | 0.001 | 0.529 | 0.001 | 0.402 | 0.026 | 0.401 | 0.021 | 0.348 |
| 4 stacks | 0.001 | 0.544 | 0.001 | 0.408 | 0.012 | 0.391 | 0.024 | 0.354 |
| Average | 0.001 | 0.569 | 0.012 | 0.414 | 0.016 | 0.391 | 0.018 | 0.366 |

Table 4.4: Average ATE ordered by the resolution, synthetic data.

| | 5-3-2 | | 10-6-4 | | 15-9-6 | | 20-12-8 | |
|---|---|---|---|---|---|---|---|---|
| | AA | KinFu | AA | KinFu | AA | KinFu | AA | KinFu |
| Empty | 0.002 | 0.467 | 0.003 | 0.097 | 0.002 | 0.092 | 0.010 | 0.082 |
| 1 box | 0.002 | 0.472 | 0.003 | 0.157 | 0.002 | 0.105 | 0.010 | 0.082 |
| 1 stack | 0.002 | 0.464 | 0.003 | 0.142 | 0.003 | 0.132 | 0.010 | 0.071 |
| 2 boxes | 0.002 | 0.465 | 0.003 | 0.162 | 0.003 | 0.109 | 0.003 | 0.103 |
| 2 stacks | 0.003 | 0.126 | 0.003 | 0.142 | 0.003 | 0.105 | 0.003 | 0.069 |
| 4 boxes | 0.003 | 0.112 | 0.002 | 0.120 | 0.003 | 0.101 | 0.003 | 0.075 |
| 4 stacks | 0.003 | 0.104 | 0.002 | 0.102 | 0.010 | 0.087 | 0.003 | 0.073 |
| Average | 0.002 | 0.316 | 0.003 | 0.132 | 0.004 | 0.104 | 0.006 | 0.079 |

Table 4.5: Average ATE ordered by the number of iterations, synthetic data.

| | 1 in 1 | | 1 in 2 | | 1 in 4 | | 1 in 8 | |
|---|---|---|---|---|---|---|---|---|
| | AA | KinFu | AA | KinFu | AA | KinFu | AA | KinFu |
| Empty | 0.002 | 0.415 | 0.003 | 0.154 | 0.002 | 0.103 | 0.009 | 0.076 |
| 1 box | 0.002 | 0.407 | 0.003 | 0.127 | 0.002 | 0.176 | 0.010 | 0.078 |
| 1 stack | 0.002 | 0.478 | 0.003 | 0.086 | 0.003 | 0.085 | 0.010 | 0.065 |
| 2 boxes | 0.002 | 0.567 | 0.003 | 0.102 | 0.003 | 0.111 | 0.004 | 0.082 |
| 2 stacks | 0.003 | 0.105 | 0.002 | 0.289 | 0.003 | 0.110 | 0.003 | 0.086 |
| 4 boxes | 0.003 | 0.092 | 0.002 | 0.071 | 0.002 | 0.141 | 0.003 | 0.068 |
| 4 stacks | 0.002 | 0.088 | 0.002 | 0.070 | 0.010 | 0.102 | 0.004 | 0.083 |
| Average | 0.002 | 0.307 | 0.003 | 0.128 | 0.004 | 0.118 | 0.006 | 0.077 |

Table 4.6: Average ATE ordered by the downsampling rate, synthetic data.

| | 2 mm | 3 mm | 4 mm | 5 mm |
|---|---|---|---|---|
| Empty | 14.5% | 19.8% | 20.7% | 26.8% |
| 1 box | 14.5% | 21.1% | 21.6% | 26.5% |
| 1 stack | 15.0% | 23.2% | 23.0% | 27.2% |
| 2 boxes | 14.9% | 24.6% | 24.1% | 24.5% |
| 2 stacks | 17.7% | 25.6% | 24.9% | 29.6% |
| 4 boxes | 18.4% | 17.7% | 26.3% | 31.4% |
| 4 stacks | 19.2% | 19.9% | 23.7% | 30.8% |
| Average | 16.3% | 21.7% | 23.5% | 28.1% |

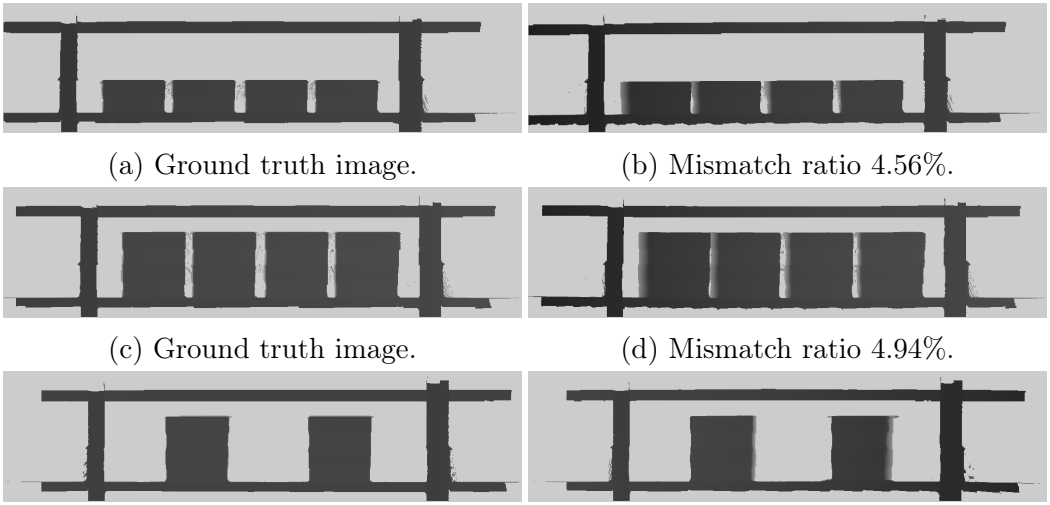
Table 4.7: Average pixel mismatch ratios ordered by the resolution.

| | 5-3-2 | 10-6-4 | 15-9-6 | 20-12-8 |
|---|---|---|---|---|
| Empty | 15.5% | 17.9% | 19.6% | 26.0% |
| 1 box | 14.7% | 23.6% | 19.5% | 26.0% |
| 1 stack | 14.1% | 23.6% | 26.3% | 25.9% |
| 2 boxes | 14.6% | 23.6% | 24.1% | 29.8% |
| 2 stacks | 19.7% | 23.5% | 23.9% | 28.7% |
| 4 boxes | 18.9% | 20.9% | 23.9% | 29.0% |
| 4 stacks | 18.6% | 19.9% | 26.3% | 28.8% |
| Average | 16.6% | 21.9% | 23.4% | 27.7% |

Table 4.8: Average pixel mismatch ratios ordered by the number of iterations.

| | 1 in 1 | 1 in 2 | 1 in 4 | 1 in 8 |
|---|---|---|---|---|
| Empty | 13.4% | 21.6% | 22.3% | 23.9% |
| 1 box | 12.7% | 19.5% | 23.1% | 33.3% |
| 1 stack | 16.2% | 22.9% | 16.7% | 30.8% |
| 2 boxes | 16.6% | 24.0% | 17.8% | 21.4% |
| 2 stacks | 16.6% | 28.1% | 31.7% | 26.0% |
| 4 boxes | 15.5% | 16.3% | 32.1% | 35.2% |
| 4 stacks | 21.4% | 18.3% | 16.2% | 33.8% |
| Average | 16.1% | 21.5% | 22.8% | 29.2% |

Table 4.9: Average pixel mismatch ratios ordered by the downsampling rate.

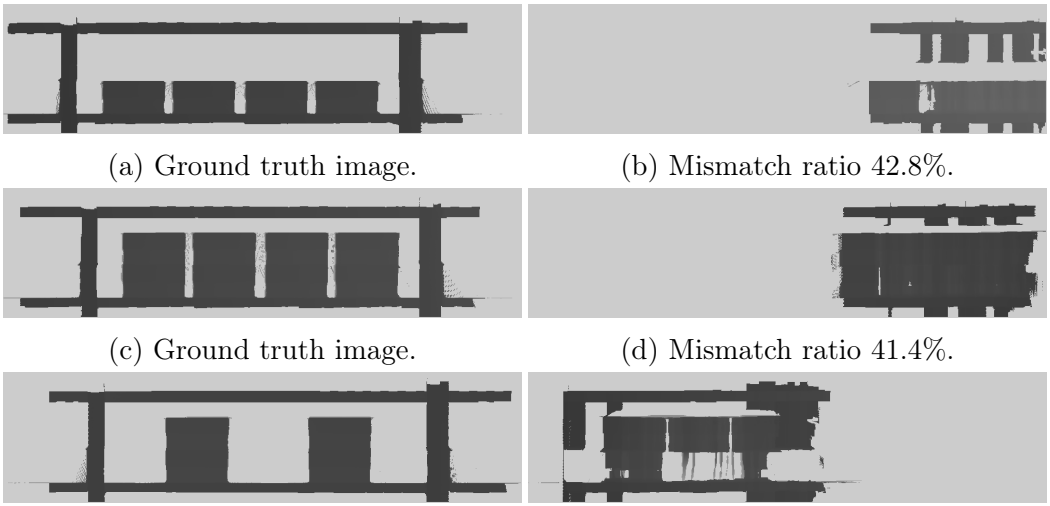


(a) Ground truth image. (b) Mismatch ratio 4.56%.

(c) Ground truth image. (d) Mismatch ratio 4.94%.

(e) Ground truth image. (f) Mismatch ratio 5.76%.

Figure 4.2: Some of the best results of KinectFusion reconstruction in terms of mismatch ratio.



(a) Ground truth image. (b) Mismatch ratio 42.8%.

(c) Ground truth image. (d) Mismatch ratio 41.4%.

(e) Ground truth image. (f) Mismatch ratio 34%.

Figure 4.3: Some of the worst results of KinectFusion reconstruction in terms of mismatch ratio.

The data shows that, both in real-world and in simulated data, the average ATE decreases with increasing voxel size and number of iterations. In case of the simulated data, it also quickly decreases with increasing downsampling rate, while for the real-world data no definitive trend can be observed.

For the pixel mismatch ratios the trends are inverse: the mismatch ratios grow with increasing voxel size, number of iterations and downsampling rates.

It can be clearly seen that the average ATE in the Assisted approach is always less than that of the KinectFusion approach by one to two orders of magnitude. In fact, the results of the trajectory estimation demonstrate an overall poor performance of the KinectFusion implementation (Fig. 4.4).

Out of 1792 test images produced by the KinectFusion implementation, only 127 images (7.1%) demonstrate satisfactory quality of reconstruction with ATE under 5 cm, 469 (26.2%) images with ATE between 5 and 10 cm contain significant errors which may hinder object recognition and lead to physical manipulation failure and the remaining 1196 (66.7%) show the ATE

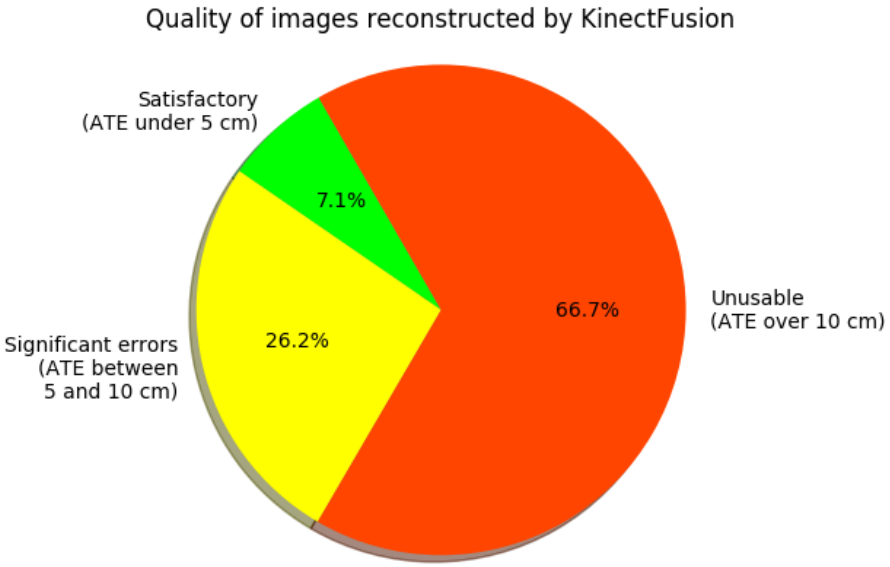


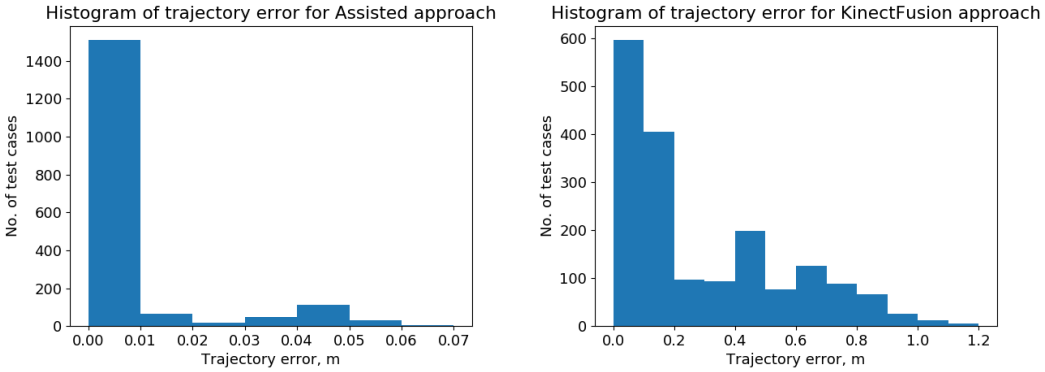

Figure 4.4: Proportion of the images of different quality reconstructed by the KinectFusion approach.

of more than 10 cm and are completely unusable for object recognition due to severe distortions.

The images produced by the Assisted approach, on the other hand, demonstrate decent reconstruction quality, since the transformation data used in the assisted reconstruction does not contain large amounts of noise.

Figure 4.5: Some of the worst cases of the Assisted approach reconstructions (ATE around 6 cm).

For most images, the trajectory error is less than 1 centimeter and even in the worst cases (Fig. 4.5) it does not exceed 7 centimeters, whereas more than half of the KinectFusion reconstructions show trajectory error greater than 10 centimeters, and in some cases it reaches up to 1.2 meters (Fig. 4.6).




(a) Assisted approach.

(b) KinectFusion approach.

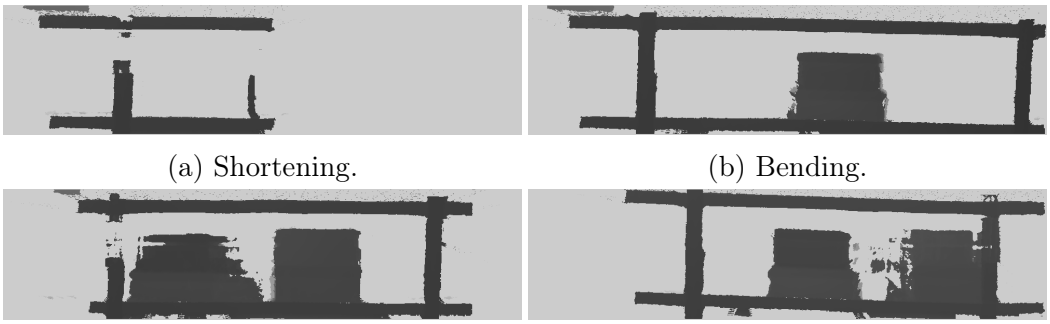
Figure 4.6: Histograms of trajectory errors registered for both approaches.

Nevertheless, by artificially introducing the error to the transformation data, the surface reconstruction error in the Assisted approach can be made arbitrarily large, while the quality of the reconstructions produced by the KinectFusion implementation remains unaffected (Fig. 4.7). This shows the advantage of the KinectFusion algorithm which is the independence of the reconstruction error from the error in the external sensor pose estimations.

(a) Assisted approach. (b) KinectFusion approach.

Figure 4.7: Reconstruction of the scene with artificially introduced transformation estimation error (constant drift in $z$-direction).

In general, every image generated by the KinectFusion implementation demonstrates one or more gross errors of reconstruction such as sudden interruptions, shortening and bending of surfaces, overlapping of objects and distorted box shapes (Fig. 4.8).



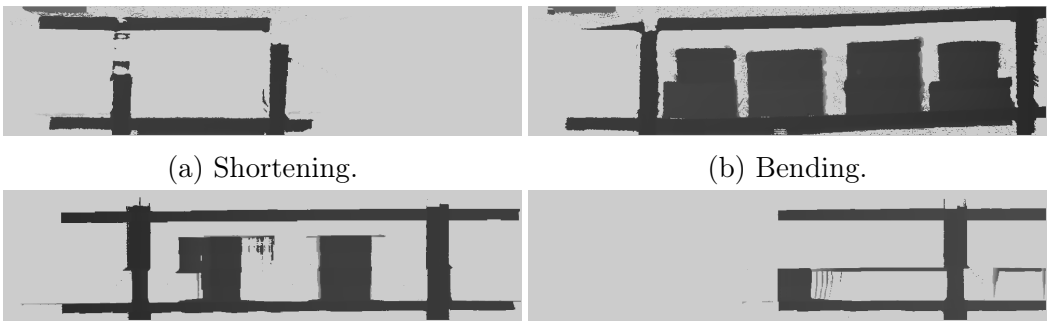
(a) Shortening. (b) Bending.

(c) Box shape distortion. (d) Multiple errors.

Figure 4.8: Observed types of reconstruction errors (real-world data).

The algorithm performs particularly poorly on box arrangements with low filling density. The lenghts of empty compartments are always compressed to approximately the width of the sensor's field of view, which is to be expected due to the similarity of depth images recorded in the middle regions of the compartments up to the point of practical identity (Fig. 4.8a). For non-empty compartments, such shortening is observed less frequently but is still present (compare 4.8b to 4.8c, 4.8d).

In all depth images, including those that correctly reflect the shapes of the objects (4.2), the reconstructed surface is skewed or bent towards the sensor at the side of the robot's final position (in the illustrations, the colors in the image gradually become darker from the initial to the final position which means decreasing distance to the robot). The bending is constant within



(a) Shortening. (b) Bending.

(c) Box shape distortion. (d) Multiple errors.

Figure 4.9: Observed types of reconstruction errors (synthetic data).

the given reconstructed scene (but not between different scenes) and always corresponds to the direction of the robot's movement. In many images, the scene is also bent in other directions (4.8b, 4.8d).

All kinds of bending appear both in the images reconstructed from real-world data (Fig. 4.8b) and from synthetic data (Fig. 4.9b) and, although the downward and upward bending is normally less significant in the synthetic data, the forward bending is present in all images in approximately equal amounts, which allows to generally rule out the errors in the sensor calibration and the noise in the depth images as its cause.

In about half of all images, regardless of the presence of other errors, the sizes and shapes of the boxes are distorted. Some reconstructed boxes have width from approximately 0.5 to 1.25 times their actual width and some have their side contours completely destroyed (Fig. 4.8c). Box shape distortions occur both in the real and the synthetic data equally often (Fig. 4.9c).

# Chapter 5

# Discussion and conclusion

The results of the testing show the superior performance of the Assisted approach over the current OpenCV implementation of the KinectFusion approach in the examined setting. Although the KinectFusion algorithm does in fact eliminate the influence of the error in the external sensor trajectory estimation on the reconstruction error, the distortion of the reconstructed models is shown to be too significant to be able to produce usable two-dimensional projections of the scene. The Assisted approach on the other hand is proven to be an acceptable reconstruction algorithm for the examined setting provided that the error of localization does not exceed certain levels.

This represents a significant, although negative in essence, result for Magazino GmbH who kindly provided the data and the testing environments for the evaluation of the approaches. Based on the results of this thesis, the research on the KinectFusion approach will be abandoned and the Assisted approach will remain the main productional implementation of the scene reconstruction software for TORU 5 until further research reveals a better solution for the surface reconstruction tasks in the given setting.

There are some possible research directions to consider while searching for an improved surface reconstruction algorithm.

One possibility would be to examine the OpenCV implementation of the KinectFusion algorithm in detail and investigate the practicability of its adaptation for the given setting. Since some of the obtained images demonstrate somewhat decent quality of reconstruction, there is probably a potential for improvement in the current implementation of the algorithm.

However, the modification of the code of the OpenCV library is still undesirable due to the compatibility reasons. Magazino GmbH would have

to maintain a separate version branch for the implementation of the 'rgbd' module of the OpenCV library, which would complicate software package management and integration of the improvements made to the module by its original author.

Probably the most promising possibility would be to investigate surface reconstrucion algorithms which use RGB images along with depth images for sensor trajectory estimation. The sensors used for shelf scanning on TORU 5 are RGB-D cameras capable of producing RGB and depth images simultaniously. There exist open-source implementations of various RGB-D simultanious localization and mapping (SLAM) algorithms such as BAD SLAM [3],[4] or Voxblox [5],[2] with readily available ROS integration. This approach has a lot of potential, since it allows to make full use of the sensor's capabilities and also solves the problem of segmented data in the depth images due to large amounts of disconnected flat surfaces in the scanned scenes.

# Bibliography

[1] Newcombe, R. A., Izadi, S., Hilliges, O., Molyneaux, D., Kim, D., Davison, A. J., Kohli, P., Shotton, J., Hodges, S., Fitzgibbon, A. (2011). *KinectFusion: Real-Time Dense Surface Mapping and Tracking.* Paper presented at The 10th IEEE International Symposium on Mixed and Augmented Reality (ISMAR 2011), Basel, Switzerland. DOI: 10.1109/ISMAR.2011.6092378

[2] Oleynikova, H., Taylor, Z., Fehr, M., Nieto, J., Siegwart, R. (2017). *Voxblox: Incremental 3D Euclidean Signed Distance Fields for On-Board MAV Planning.* Paper presented at IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, Canada. DOI: 10.1109/IROS.2017.8202315

[3] Schops, T., Sattler, T, Pollefeys, M. (2019). *BAD SLAM: Bundle Adjusted Direct RGB-D SLAM.* Paper presented at The IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 134-144, Long Beach, California, USA.

[4] https://github.com/ETH3D/badslam

[5] https://github.com/ethz-asl/voxblox