ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Computer Vision and Geometry Group
Prof. Marc Pollefeys

Master Thesis

Dense Object SLAM

Autumn Term 2018

Supervised by:
Marc Pollefeys
Peidong Liu

Author:
Kailin Huang

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

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

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

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

Title of work (in block letters):

Dense Object SLAM

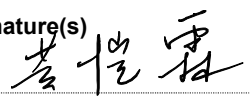
Authored by (in block letters):
For papers written by groups the names of all authors are required.

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

With my signature I confirm that

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

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

Place, date	**Signature(s)**
Zürich, 31.10.2018	

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

Contents

Abstract

Simultaneous Localization and Mapping (SLAM) is an active research flied with application in a wide range of areas.

Many visual SLAM methods are based on visual image features and descriptors in the images. Those features are hand-crafted and can suffer under different lighting conditions and viewpoints. The resulting map is only a dense or sparse geometrical map with no further semantic information.

Including the information of objects from a image into SLAM is one of the recent research fields. However, many of the previous researches rely on existing 3D object models or only perform geometrical segmentation of object. At the same time, recent researches in instance-aware semantic segmentation using deep neural networks has become very accurate and reliable. They can detect objects in an image and create a pixel level object mask for each detected object in the image.

In this work, we combine the output of an instance-aware semantic segmentation with the dense SLAM framework InfiniTAM and modified the mapping module of InfiniTAM to create a system that is able to track the ego-motion of camera reliably and build a map, where detected objects are standalone and separated from the background world. Observations of each object is fused by associating detected objects across different frames and the image to model alignment is refined before the new observation is fused into the reconstruction. A object cleanup method is developed to remove voxels from false segmentation, creating a 3D object model even if the segmentation mask is imperfect. From the background, objects are removed to create a background only environment which can be used even if the layout of the objects are changed.

We evaluate the resulting system in terms of the reconstructed objects and the accuracy of the estimated camera motion with ICL-NUIM and Microsoft 7 Scenes RGB-D datasets. The effect of the object and background voxel cleanup is can also be clearly demonstrated.

Due to the CPU-only implementation and the standalone segmentation pipeline, the system is currently not capable of running in real time. As the InfiniTAM framework the system based on is capable of using GPU acceleration, We will work on extending the system to GPU and improve its general performance to achieve real-time level performance.

Although it is currently mainly a mapping system, its output can be the trigger various further applications, including but not limited by place detection using objects, moving objects detection, reconstructing occluded part of objects, etc. These will be addressed in the further works.

Acknowledgements

I would like to thank my advisor Liu Peidong and the Computer Vision and Geometry Group for giving me the chance to work on this exciting project and guiding me through the technical problems.

I would also like to thank my parents and my girlfriend for supporting me during this time and my whole study.

Symbols

Acronyms and Abbreviations

ETH	Eidgenössische Technische Hochschule
SLAM	Simultaneous Localization And Mapping
TSDF	Truncated Signed Distance Function
ICP	Iterative Closest Point

Chapter 1

Introduction

1.1 Motivation

Visual Simultaneous Localization and Mapping (SLAM) is one of the active research fields in computer vision. It serves for two purposes: one is to localize and track the ego-movements of the camera, the other one is to build a global map of the environment, which can be used to localize a camera frame.

Many of the existing SLAM algorithms rely heavily on low-level hand crafted features in an image to track the camera movements and localize the camera[1]. For example, Montiel [2] and Mur-Artal and Tardos [3] rely on the ORB features, not only for the tracking, but also for the loop closure detection, by using the Bag of Binary Word technique presented in Gálvez-López and Tardós [4]. As pointed out by Lowry et al. [1], low level local feature descriptors are often not robust in case of lighting changes, while global features may suffer from large viewpoint changes.

We as human beings however, perceives our environment in a different way. Instead of focusing on edges, corners and other geometrical features, human can also recognize objects in an environment and use the information of objects to improve its understanding of the scene. Objects in the scene can be viewed as a higher level feature, which could be exploited for many other applications.

In 2013, Salas-Moreno et al. [5] was one of the first attempts to utilizing the information of objects to solve the problem of SLAM. However, the detection on objects relied on the pre-loaded 3D models of objects, which are represented by meshes. The objects are detected using point-pair features [6]. However, the need of a 3D model for each of the objects would be a limiting factor on using the system in different environments. Even objects of the same type can have very different shapes, such as different cars, different chairs, etc, and it is not feasible in many cases to have a known 3D model of the objects in the scene. Therefore, we choose to solve the problem of object detection without the prior knowledge of a 3D model.

Recent developments in object detection and segmentation by deep neural networks show that it can not only detect objects by drawing a bounding box around the objects, but also do a pixel-wise segmentation by differentiating which object a pixel in an image belongs to. [7] and [8] are attempts merge segmentation with SLAM systems and to identify moving objects and separate them from the rest

static world.

Lee et al. [9], Salas-Moreno et al. [5] and Finman et al. [10] proposed systems utilizing object information to enable place recognition on an object level, which could be more robust to lighting and viewpoint changes than hand crafted local and global descriptors.

Further, by getting prior knowledge of the objects, such as symmetric information of indoor objects like computers, cups and other information such as regularities in man-made environments, it is also possible to increase the quality and the global consistency of the map. By reconstructing each objects separately, it is also possible to create Virtual / Augmented Reality scenes with the objects stand alone, with out being a fixed part of the map.

1.2 Contribution

In this work, the output of a state-of-the-art object instance segmentation framework, Mask-RCNN [11], is used to provide the information of objects in images[1]. We focus on building a map with each object as a standalone part. Although not yet implemented, it enables the possibility to do place recognition and loop closure by matching the objects separately.

In order to create a 3D model suitable for further usage, a dense SLAM framework is preferred than a sparse one. [13] presented one of the first SLAM frameworks to use a RGB-D camera to create a dense 3D map. [13] uses a Truncated Signed Distance Function (TSDF) to represent the map, which divides the environment densely into 3D voxels. As the memory demand of such representation is linear to the volume whole 3D environment, Kähler et al. [14] extended [13] by using Voxel Hashing, Nießner et al. [15]. Instead of allocating memories for a pre-defined fixed volume, Kähler et al. [14] only allocate memory for a voxel if this is on or in the close distance of a surface, which makes the system capable of mapping larger environments.

By integrating the output of the instance aware object segmentation into the dense RGB-D SLAM framework in Prisacariu et al. [16], we present a system which is capable of tracking the camera pose and reconstruct each object instance in the environment separately. The output of the system is a 3D dense reconstruction of the environment without the detected objects, as well as each single object as a separate 3D model.

We implemented an algorithm to associate the objects in different frames in order to recognize if they are the same object in different frames or different objects. Further we implemented the ability to remove voxel blocks, which should belong to objects, from the background, even if it is not detected by each frame.

Further voting based system of cleaning up the voxels of objects is implemented. This is due to the fact that segmentation results are not perfect and this leads to the object containing voxels not belong to it.

A multi raycast algorithm is developed to efficiently doing raycast to multiple objects as well as the background in the scene. It also considers the possibility that

[1] The OpenSource implementation from Matterport [12] is used

certain part of objects can be occluded by others due to different view point. The point cloud can not only be used for visualization, but also for a frame-to-model based tracking, such as the ICP tracker implemented in [16].

The system should also be easily extendable to enable the use of other kind of tracking systems, as well as a parallel execution of tracking and mapping.

1.3 Outline

In the following part of this work, we first introduce previous works related to dense mapping and 3D reconstruction and the semantic segmentation in 2, followed by a more detailed introductions of the systems which this works is based on in 3. Then our system introduced in detail in 4 with some experimental evaluation performed on different datasets in 5. At the end, the limitation of the current system is discussed 6 with many known points to be improved on and possibilities to extend this for more advanced tasks in the future works.

Chapter 2

Related Works

In this chapter, related works in the field of dense reconstruction, instance aware semantic segmentation and SLAM.

2.1 Dense Map Representation

2.1.1 Surfel Based

Pfister et al. [17] first presented surface elements called surfels in a computer graphics context. They can be viewed similar as 3D points with extended information such as surface normal, textures etc. Each surfel has their own 3D position but do not have directly information about connectivity with other surfels.

Several dense SLAM systems make use of the surfel based reconstruction to represent the dense 3D map. One of the most influential SLAM systems build upon surfels is by Whelan et al. [18], which is able to perform a dense real time SLAM with the ability to recognize places and closing loops. Its further development by McCormac et al. [19] added semantic labels from a semantic segmentation into the surfel based map by using a Bayesian update of the surfel labels. In InfiniTAMv3[16], the latest version of InfiniTAM[20], the ability of reconstructing a surfel based scene is also implemented in addition to the voxel hashing based map representation.

2.1.2 Voxel Based

Unlike surfels, which directly represents surfaces of the environment in a unstructured way, the voxel based representation implies a structured discretization of the world with small cubes called voxels. Newcombe et al. [13] presented one of the first real time dense SLAM systems which utilizes the voxel based representation of the 3D map. Due to the structured grid, the system can however only represent a fixed, predefined volume, as the whole volume needs to be discretized at the beginning. Several systems based on Newcombe et al. [13] are aimed to make the system able to reconstruct to larger environments.

Moving Volume

Whelan et al. [21] presented Kintinuous, the first system to extend the KinectFusion [13] to a larger environment.In this approach, the actively mapped volume is not increased. However, the volume currently being mapped can be dynamically changed with the movement of camera, so the viewing range of the camera is always being mapped. The volume going out of the camera viewing range are retracted as a point cloud and a mesh is created based on this point cloud, which serves as the long term map.

Although this approach solved the problem of the limited mapping volume, the volume of the active map part is still a fixed size. This does not change the fact that the empty space between the camera and the surface is still being mapped as the grid has a regular shape. Therefore many voxels in the active region are wasted by just storing information of empty spaces. This leads to the following approaches, which make use of other data structures to only save the most related voxels near the surface and discard empty voxels.

Octree

First presented by Zeng et al. [22] to adress the problem of KinectFusion[13] mentioned in 2.1.2, its main idea it to not save the many empty voxels and hence reduce the memory usage significantly.

Voxel Hashing

A more efficient approach is presented by Nießner et al. [15] called voxel block hashing. It is able to build large-scale maps using voxel-based map representations. Instead of allocating memory for all visible voxels in the 3D environment, including many voxels in the empty space, only voxels near the actual object surface are allocated. The voxels are grouped in blocks and are only allocated within a small truncation distance of the actual surface. Theses voxel blocks are stored in a hash table for an efficient access where the hash value is calculated by using the 3D location of the voxel block in the world and hash collisions are handled by hash buckets with linked lists.

InfiniTAM by Kähler et al. [14] builds upon the voxel block hashing and is able to perform ICP based tracking and 3D reconstruction. InfiniTAM is divided such that the code can be compiled and run on either a CPU-only structure or with the help of nVidia CUDA, also on supported GPUs. The tracking can not only be performed using the RGB-D information, but also can utilize inertial information if a IMU is given. Prisacariu et al. [16] expanded the framework further with the ability to build smaller sub-maps in order to perform loop closure detection and optimization.

2.2 Instance Aware Semantic Segmentation

Since the AlexNet released by [23] achieve good results in the ILSVRC-2012, the use of Convolutionl Neural Network (CNNs) for scene understanding has been developed

rapidly. The scene understanding started with the classification, then developed to the detection and localization of objects by giving the bounding boxes of objects as the output. More advanced scene understanding continues to output a pixel-wise semantic segmentation with the most state-of-the art algorithms being able to not only label the pixels semantically, but also distinguish different objects in the same semantic class. Figure 2.1 illustrates the different stages of scene understanding.

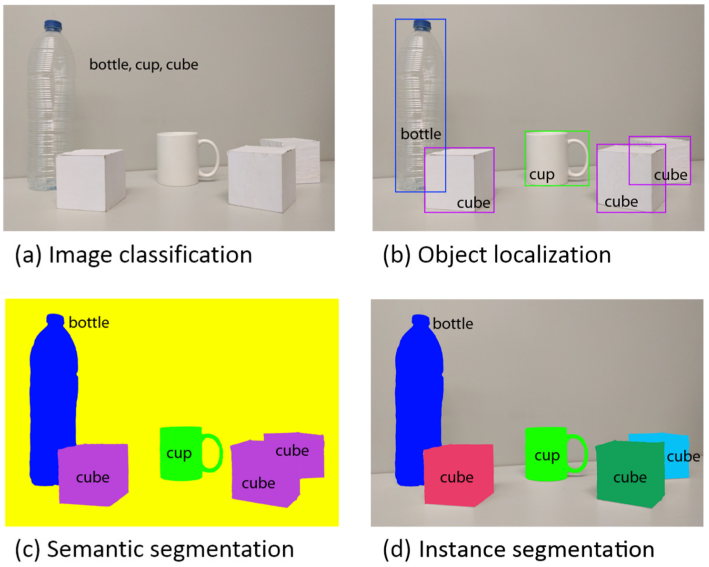


Figure 2.1: Different levels of scene understanding: Image classification, Object detection, Semantic segmentation, Instance segmentation. [24]

For the use of our object level SLAM system, it is required to have a pixel-wise segmentation of the semantic label and to correctly distinguish different objects of the same semantic label in the same image, which means we need the instance aware semantic segmentation illustrated in Figure 2.1 (d).

The early approaches such as AlexNet[23] mainly focused on the task of classification and object detected. They use deep convolution neural network to achieve the task. Typically it contains several convolutional layers with maximum pooling layers in between and the fully connected layers at the end to extract the features to perform the classification.

For the task of object detection, many algorithms including Fast-R-CNN[25] and Faster-R-CNN[26] are based on a regional proposal to have a first guess of regions possibly containing objects. Based on the region of interests it outputs a class label for the region and a bounding box of the object. By using the intermediate outputs of the convolution layers for the region proposal Faster-R-CNN solved the problem of the region proposal being the bottleneck of the inference speed.

For the pixel level scene understanding, FCN Long et al. [27] is one of the first attempts to replace the fully connected layers in CNNs like AlexNet[23] with de-convolution layers, upsampling the downsampled image back to the resolution of the original image and thus performing a segmentation on a per pixel basis, enabling the possibility of pixel-level scene understanding.

Combining the results from object detection in Faster-R-CNN [26] and FCN [27],

Mask-RCNN proposed by He et al. [11] basically adds a small FCN to the region proposal of the first stage of Faster-R-CNN, which creates a segmentation parallel to the original bounding box detection of the Faster-R-CNN, while still being able to run at approximately 5 FPS according to He et al. [11]. Although not able to run completely real time, due to the high accuracy of Mask-RCNN, we choose it as our segmentation provider.

We choose to use the Matterport implementation[12] of Mask-RCNN[11], because it provides a easy to use and easy to adapt framework which is pretrained on the Microsoft COCO dataset [28]. It is also able to run without GPU support, unlike the source code released by the original authors.

Figure 2.2 shows the segmentation mask of the Matterport implementation trained on the MS-COCO[28] data set, which contains 78 categories of object, including many indoor and outdoor object classes. The segmentation not only outputs the category which the pixel belongs to, but can also separate different instances of objects in the same category, even if they are located close to each other. In this example, there are multiple cars detected in the image, and they are labeled separately. This is crucial to make the object based mapping able to distinguish different objects of the same kind.

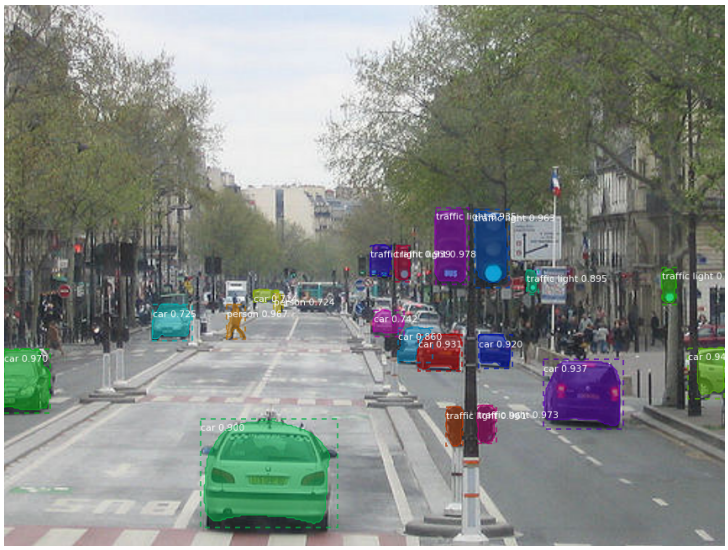


Figure 2.2: Example of the masks generated by Mask-RCNN[12]

2.3 Iterative Closest Point Tracking

KinectFusion [13] uses iterative closest point (ICP) to track the camera pose in the dense reconstructed world. ICP is first introduced by Besl and McKay [29] and used to align depth images with dense 3D maps in many dense SLAM algorithms including [13, 21, 18, 20, 14, 16]. Different from frame-to-frame matching methods used in visual odometry and many sparse SLAM algorithms such as [2, 30], it does not directly create a correspondence between two neighbouring frame, but rather matches the new frames into the reconstructed world.

As pointed out by Lee et al. [9], ICP has its limitation of being prone to local mini-

mum and only can track 6 DoF rigid body transformations, no similarity transforms with scale change. The latter is not a problem as RGB-D cameras directly provide scale information. However the first one means it can fall into wrong local minimum especially when tracking fast movements.

2.4 Simultaneous Localization And Mapping

In general, SLAM means the estimation of the ego-motion of the robot and the construction of the environment as a map [31]. Visual SLAM usually estimates the camera's ego-motion using techniques such as visual odometry and builds a map which contains sparse feature points or densely reconstructed surfaces. Usually a SLAM system comprises a front-end estimation pipeline to estimate the camera pose from the input image, and a back-end optimization module to ensure the global consistency of the map.

Depending on the different types of camera pose estimation, we can roughly divide SLAM systems into 3 categories: feature based methods, direct methods and RGB-D SLAM [32].

2.4.1 Feature based SLAM

Feature based SLAM usually uses hand crafted image feature point detectors and descriptors. A matching between the feature points across different images needs to be find in order to estimate the camera motion, which is done by minimizing the reprojection errors of matched feature points. This works well as long as the environment is textured. In a texture free area, often not enough feature points can be found to ensure accurate estimation of camera pose. Handcrafted features is also not suited for long term mapping, because seasonal changes can change appearance of certain areas so dramatically that the features can no longer be detected or matched.

One of the very first works in this area is MonoSLAM by Davison et al. [33]. It estimates the camera motion and the position of the feature points in 3D using an extended Kalman filter. Klein and Murray [34] proposed PTAM, the first SLAM algorithm which runs tracking and mapping in parallel threads. Therefore it is able to use more computation intensive bundle adjustment algorithms in mapping to ensure global consistency of the map, while the tracking runs in the separate thread in real time. A more complete feature based SLAM algorithm is the ORB-SLAM by Montiel [2]. It uses ORB features as key points. In addition to the same structure of parallel tracking and mapping as PTAM, ORB-SLAM uses a bag of word [4] model and a pre-loaded binary dictionary to detect loop closures, when the camera revisits a known location mapped before. When a loop closure is detected, a global pose-graph optimization is performed to reduce drift accumulated during the tracking.

2.4.2 Direct SLAM

Direct methods, unlike the feature based methods, do not use reprojection errors of matched feature points and thus do not require the resource consuming fea-

ture matching. They usually assume the photometric invariance of the same point and tries to minimize the differences in photometric intensity difference to achieve tracking.

DTAM by Newcombe et al. [35] is a dense direct method. It uses multi-baseline stereo to estimate the depth information for each pixel. For tracking, the image is aligned with synthetic views generated from the reconstructed map. As most of the dense methods, it uses GPU to parallelize the computation in order to handle the large computation need coming with the dense reconstruction. In stead of a dense reconstruction, LSD-SLAM Engel et al. [36] only reconstructs areas with high intensity gradients, leaving out the texture free areas because it is difficult estimate the depth for these areas. It also incorporates a loop closure detection and global pose graph optimization [32]. As a semi-dense method, it is also able to run in real-time with CPU only. DSO Engel [37] is a sparse visual odometry. It only selects key points with high intensity gradients across the image. DSO uses both the geometric and the photometric camera calibration to ensure the intensity measurement is consistent. Although the priginal DSO does not include the loop closure abilities, this is added in LDSO by Gao et al. [38].

2.4.3 RGB-D SLAM

RGB-D SLAM is a special category of SLAM algorithm which uses special depth sensors such as Microsoft Kinect[1] or Intel RealSense[2] depth cameras. They use structured light to gain depth information for each pixel. However, the maximum range usually do not exceed 10 meters[3], which mainly limits its use in indoor environments. Usually an ICP[29] based tracking is used to align the depth image with the reconstructed map.

First work in this area is KinectFusion by Newcombe et al. [13], which uses a structured 3D voxel grid to represent the 3D volume. Same as the dense methods mentions above, it also uses GPU to accelerate the processing. Kintinuous Whelan et al. [21] uses a sliding volume approach to make the fixed volume of KinectFusion extensible. Kähler et al. [14] proposed InfiniTAM in 2015, which uses voxel hashingNießner et al. [15] to reduce the computation cost and increase the scalability of KinectFusion.

2.4.4 Object SLAM

As pointed out briefly by Lowry et al. [1], using object level information can help SLAM in many ways, especially in terms of long term mapping.

Salas-Moreno et al. [5] proposed one of the first SLAM algorithms using objects, although the objects are limited to the ones with their 3D model pro-loaded. It also proposed to use object graph for the task of relocalization. SemanticFusion by McCormac et al. [19] uses deep neural networks for image segmentation to replace the fixed model database of SLAM++. It uses surfels instead of voxels for the dense 3D reconstruction. The results from the segmentation is used together with the class probabilities. The semantic mask is projected to the reconstruction and multiple

[1] https://developer.microsoft.com/en-us/windows/kinect
[2] https://realsense.intel.com/
[3] https://software.intel.com/en-us/realsense/d400

observations are fused in a Bayesian way. Bowman et al. [39] also fuses the semantic data across different frames in a similar way. Don et al. [40] proposed a localization using graph matching similar to SLAM++ to achieve localization, however, it is only using a semantic segmentation without exploiting the object instances. DynSLAM by Bârsan et al. [7] and MaskFusion by Agapito [8] are approaches which uses object level information to distinguish static and dynamic objects and reconstruct them differently.

Chapter 3

Background

In this chapter, the most important components and algorithms used in our object SLAM algorithm is reviewed in detail.

3.1 Dense Map Representation

3.1.1 Voxel-Based

Voxel-based map representation is based on a regular grid discretization of the 3D environment. Similar to pixels dividing a 2D image into discrete squares in 2D, voxels divide a 3D scene into discrete 3D cubes.

Curless and Levoy [41] introduced the signed distance function (SDF) for voxels to represent a 3D scene. SDF represents the distance from the object surface, where zeros represent the surface, positive values represent free space corresponding to the distance to the surface, and negative values represent occupied space behind the surface.

Newcombe et al. [13] used the truncated SDF to represent a 3D volume. As one is mainly interested in the location of the surface of the occupied space, only the voxels in a certain distance, the truncation distance, to the surface need to be evaluated. 3.1 shows a simple 2D illustration of a TSDF grid, where SDF values are saved in voxels in the truncation distance. Each voxel has the value representing the closest distance from the voxel to the nearby surface, with voxels in front of the surface, which are the free space between the surface and the camera, register positive values, and the voxels behind the surface and thus unseen by the camera, register negative values. As the SDF is the distance to the closest surface, the location of the surface is simply drawn as the location where the SDF has exactly the value of zero. This can be calculated by interpolating the SDF values of the neighbouring voxels.

Because TSDF representation is based on a regular grid discretization, it is able to be run on highly paralleled structures such as GPUs to accelerate the processing.

However, one significant limitation of the TSDF representation comes also with the nature of the regular grid discretization. The entire volume in the visible range

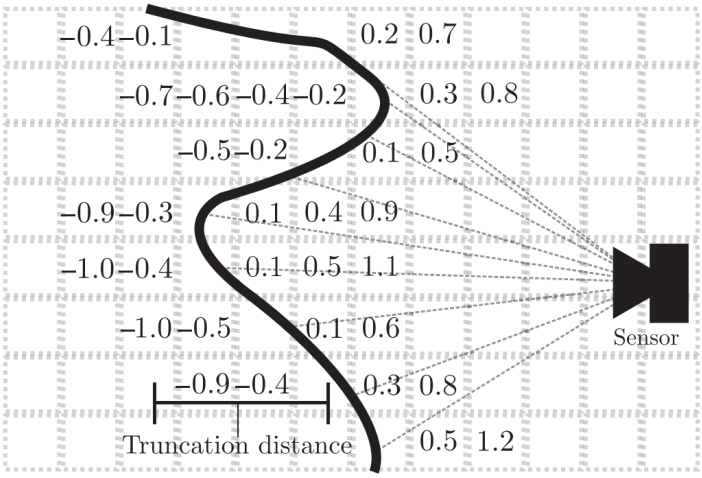


Figure 3.1: TSDF Example Illustration [42]. Voxels between the camera and the surface have positive values and voxels behind the surface have negative values.

needs to be discretized as voxel grid, although only the small part around object surfaces are actually needed and storing actual SDF values.

3.2 Voxel Hashing

To overcome the scalability problem, [15] presented the voxel block hashing method, which is used by [14] and [16] to expand KinectFusion[13]. Instead of preallocate all the voxels in a fixed volume, which means either a very large memory requirement or a limited spatial resolution of the grid, only voxels in close proximity to a surface will be allocated in the memory. This is achieved by storing the voxel data using a hash table. The basic operations for the voxel hashing is reviewed here.

In the practical implementation, voxels are grouped first into voxel blocks consisting $8 \times 8 \times 8$ voxels and the pointers to voxel blocks are then stored in a hash table and allocated as needed. This means only voxels containing useful SDF information in the truncation distance of the surface is allocated and processed, greatly reducing the memory requirement. The truncation distance can be defined manually at the start of the program.

The hash value of each voxel block is calculated by a hashing function using the position (X,Y,Z coordinates) of the voxel block as the key. The hash table is stored as an data array of hash buckets, each corresponding to a certain hash value. Each hash bucket is able to store multiple voxel blocks, in order to handle hash collisions. A further excess list in the form of a linked list is also created to store the address of voxel blocks that can no longer be stored in the regular hash bucket. The data structure of voxel block hashing is illustrated in figure 3.2.

Allocation of a voxel blocks starts with calculating the hash value using the position of the voxel block, which determines the corresponding hash bucket in the hash table. It will be iterated in the hash bucket and the excess list to try to find the voxel block with the desired position. If such is found, the reference of this voxel block is returned. Otherwise a new voxel block is inserted in the first empty position

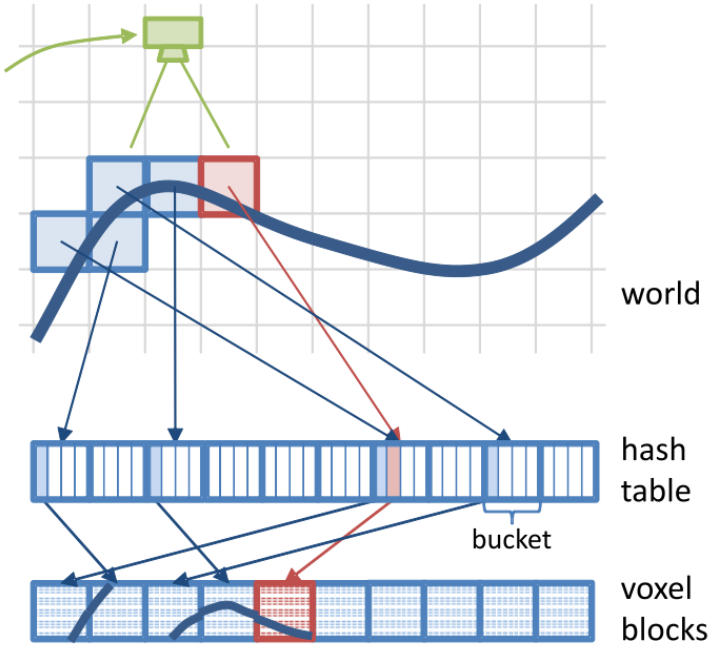


Figure 3.2: Data structure of voxel block hashing [15].

or, in case the has bucket is full, appended to the excess linked list.

The retrieval operation of a hash value is similar to the allocation. After the hash bucket is determined using the voxel block position and the hashing function, it will be searched inside the hash bucket and excess list to find the voxel block with the correct position.

In the implementation of InfiniTAM V3 Prisacariu et al. [16], the size of the voxel block, the hash table size and the excess list size are all pre-defined as macros, to suit the need of this work it is adapted. Details are in 4.3.1.

3.3 RGB-D Simultaneous Localization And Mapping

There exists many different algorithms for Simultaneous Localization And Mapping. Considering the way the map created, it can be generally divided into sparse methods and dense methods. Sparse methods usually only work with sparse feature points on images and only the feature points are registered in the map. On the contrary, dense methods, as the name suggest, create a map where the goal is to reconstruct a whole volume of the 3D environment using all the pixels available.

This work intends to build a object level SLAM system with dense reconstruction of the world, therefore the dense methods used is reviewed here.

In order to create a dense map, depth information needs to be available. This can be achieved through different ways. The most direct way is to apply a RGB-D camera, which records the color information and the depth information for each

pixel. Another common method is to generate a depth map through stereo image pair, which requires a calibrated pair of synchronized cameras to triangulate the images. Recent advanced in supervised and reinforcement learning leads to further depth generation methods from stereo or even monocular cameras.

In this work, we focus on the dense SLAM algorithms with the use of RGB-D cameras as sensor. It should be easily extended to work with other above mentioned methods which can generate depth images.

3.3.1 Tracking

Tracking in a dense map representation using a RGB-D camera can be achieved by aligning the incoming depth image with the known 3D world. Iterative Closest Point (ICP) first proposed by Besl and McKay [29] can be used to align different 3D point clouds.

The incoming depth image has pixel values representing the depth of world image point. Using the camera projection equation 3.1, one can project a point (x, y, z) in the 3D world into the camera image.

$$depth \cdot \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = K \cdot \begin{bmatrix} R & T \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{3.1}$$

Where *depth* is the vertical distance of the point to the image plane, u and v are the pixel location in the horizontal and vertical direction, K is the camera intrinsic matrix and R and T describe the rotation and transformation from the world to camera frame.

Inverting equation 3.1, we get the equation to project a depth image into the points in the world.

$$\begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} R & T \end{bmatrix}^{+} \cdot K^{+} \cdot \begin{bmatrix} u \cdot depth \\ v \cdot depth \\ depth \end{bmatrix} \tag{3.2}$$

Where $[R\ T]^{+}$ and K^{+} are pseudo-inverse matrices because they are not square matrices.

Using equation 3.2, with the known camera calibration and a initial pose, we can extract a point cloud from the depth image.

The point cloud of the known world can be then extracted using raycasting from the current camera perspective.

Having both the point cloud from the known world and the point cloud from the incoming depth image, ICP can find the camera pose aligning both point clouds by minimizing the distance norm of between points of both two point clouds iteratively.

In InfiniTAM, the tracker is implemented as the following:

First, a point map and a surface normal map is rendered using a initial guess of camera pose, this is typically the camera pose in the previous frame. Then, using a guesses camera pose for the incoming frame R, t, the points from the incoming depth image are projected using $Rp + t$. The error term to be minimized is the difference between the incoming points and the points rendered from the volume, projected onto the surface normal vector[16]:

$$d = (Rp + t - \mathcal{V}(\overline{p})) \times \mathcal{N}(\overline{p}) \tag{3.3}$$

InfiniTAM[16] further implement a resolution hierarchy to improve the convergence behaviour of minimizing the error term defined in 3.3. This incoming image is down-sampled into several levels of lower resolution images. The alignment procedure is started from the most coarse resolution, after it is converged, the resulting pose is used to initialize the alignment process of the next level finer resolution of image. This repeats until the original resolution of the image is reached.

Because the voxels have stored a weight value representing the number of observations they have, this is also used in the tracking. The weight is read and stored to the point during the raycasting process. During the tracking, the error term for different pixels are weighted according to their weight and their distance, as the points farther away is usually noisier and thus should be weighted less than the points closer to the camera.

The tracker from InfiniTAM also adds the ability to use the color information of the RGB image. By adding an weighted error term coming from RGB intensity difference to the error term from the depth value in equation 3.3, and the combined term is minimized using Levenberg-Marquardt optimization algorithm.

We currently only use the tracker with the depth information, but the use of the combined tracking can be easily adapted.

3.3.2 Dense Reconstruction

With the pose known from the tracker, the 3D world can be densely reconstructed using the depth image.

To save computing resources, not all voxels in the voxel scene are processed in the reconstruction. First, the visibility of the voxel blocks is determined by projecting its 8 corners using the pose and camera parameters, and check if they are still inside the image boundary. For all visible blocks, the range where the surface can possibly be located at is computed. Using this information, the voxel blocks are allocated to all the positions where it is close enough to the surface and added to the hash table according to 3.2.

A positive side effect of the visibility checking process is the additional knowledge about the visibility of different objects and voxels. This helps in limiting calculations only to visible objects and the visibility information also helped in removing unwanted artifacts in reconstructed objects.

The TSDF value in the voxels are then integrated as a weighted average over the last several frames, while the number of the averaged frames can be specified manually. For the weighted average, each voxel has a weight saved in form of a short integer.

When a new observation is integrated into the existing voxel, the new SDF value has the weight 1, while the previous SDF value in the voxel has an old weight, which is the number of observations it has. The weighted average is calculated in 3.4. The new weight is then updated as the sum of the old and the new weight and the SDF value and the new weight are write back into the voxel. If the voxel is new with no previous value, the new value will be stored in the voxel with the weight 1.

$$SDF_{new} = \frac{SDF_{old} \cdot Weight_{old} + SDF_{new} \cdot Weight_{new}}{Weight_{old} + Weight_{new}} \tag{3.4}$$

Chapter 4

Object SLAM Implementation Details

As the mapping and tracking system is mainly based on InfiniTAM's implementation, the libraries created and used in InfiniTAM are compiled and included into our system. However, many modifications are carried out in the code of InfiniTAM directly as many of its functions are adapted for the need of this project.

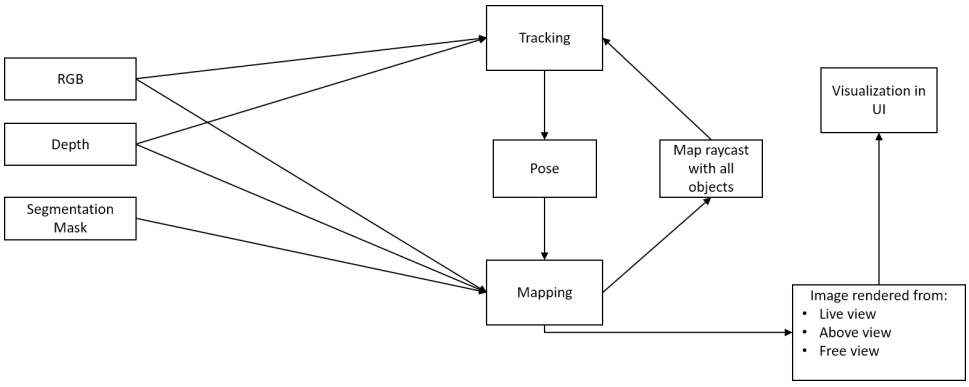


Figure 4.1: Overview of the whole algorithm

The whole algorithm has two core modules, the tracking module and the mapping module. An overview is provided in figure 4.1. The tracking module estimates the current camera pose. This is fed into the mapping module together with the RGB-D image and the segmentation. After the mapping module fused the observations into the map, it creates a point cloud using raycasting with all objects in the viewing field and the background. This point cloud is then used by the tracking module to track the following camera frame. Further, a input module is created to get the RGB-D images. As the system is currently not capable of running in real time, it is currently implemented as a image "reader" of RGB-D datasets. Lastly, a simple UI based on [43] is created to visualize the mapping results, this can be seen in figure 4.2. It renders the reconstructed scene and selected objects from the scene from different viewpoints in a window, providing a visualization while the system is tracking and mapping the environment. Obviously the system would also work without the visualization and UI.

Figure 4.2: Screenshot from the UI running the living room 1 sequence from ICL-NUIM dataset [44] with synthetic noise. Images in the first row are: 1. The reconstructed scene with all objects in it as a result of raycasting to multiple objects. 2. The input RGB-Image. 3. The whole Scene rendered from above for overview. Images in the second row are: 1. The background rendered alone. 2. The first object in the visible object list rendered. Here is a couch. 3. The background rendered from a camera viewpoint 2m behind the current camera view and with the same orientation.

To reduce the chance of misunderstanding, we define here how we use the following terms:

- Scene: The whole environment.
- Object scene: the scene which only contains the one object.
- Background scene: the scene only contains the background, with all other objects ignored.
- View: An RGB-D image with the corresponding camera pose.
- Object: An object in a scene, eg. a chair, a table etc.
- Label: The type of the object, eg. chairs, tables, background, etc.
- Object image: the segment of the RGB-D image which only contains pixels belong to the object.

4.1 Dataset Reader

The dataset reader has a base virtual class, which is implemented with different sub-classes to suit different dataset formats. The project mainly uses datasets in the format of Sturm et al. [45]. In addition to that, synthetic datasets from Handa et al. [46] are also used and they are provided in the same format as [45]. For the Microsoft RGB-D Dataset 7-Scenes [47] [48], we manually reorganize the file structure of the dataset so that it matches that of [45].

The dataset only needs the input directory where the RGB, depth and segmentation data is stored. A further association data according to Sturm et al. [45] should also be located in the same directory as the dataset reader takes the file names of the corresponding RGB and depth images from that files. Functions for reading RGB images are directly called from InfiniTAM and stored in InfiniTAM's own data types. For depth image, the InfiniTAM function only reads in the disparity values, which are then converted into the floating number real depth values, representing the distance vertical to the image plane of the camera.

For the segmentation information, the dataset reader stores all the segmentation masks from the same frame into a vector, which is then passed to the mapping module for further processing.

4.2 Segmentation Provider

The open source Matterport implementation [12] of Mask-RCNN is used as the segmentation provider. The current state of the system is not capable of running the system parallel to the SLAM, but has to be pre-computed. The model trained on the COCO dataset is used to provide detection and segmentation. The detectable object classes including person, car, airplane as well as indoor objects like couch, table, book, TV etc. The output of the system is a pixel-wise mask for each of the detected object. The mask reads 0 if it is not the object and reads a number from 1 to 78 corresponding to the respective class of object. A object mask for each object in each frame is stored as a text file.

The python script is derived from the demo provided by [12] to suit the dataset structures. Given the folder with the RGB images, the script will automatically output segmentation masks as text files with the name of the RGB-image and an increasing index of objects in the same image. The whole system is based on Keras [49] with a TensorFlow [50] backend. Depending on the system's hardware configuration it can be run on CPUs only or accelerated with GPUs using CUDA [51].

Note that the segmentation is done on the RGB image, so the segmentation mask is aligned with the RGB image. To segment the depth image, the segmentation mask needs to be aligned with the depth image first. This alignment and segmentation process is described in 4.5.3.

4.3 Map as Objects

Instead of a single map consisting the geometry of the whole environment, the map is now viewed as an accumulation of all objects. The background is also considered as a large object. Each object has its own reconstruction and the world is a sum of all of the objects:

$$World = \sum_{i=0}^{n} Object_i \tag{4.1}$$

In this way, each object is separate and has all information needed to perform tracking, reconstruction and raycasting on its own, without dependencies to other

objects. Therefore it can also be easily parallelized between different objects.

4.3.1 Modification of scene size of objects for memory efficiency

As already mentioned in 3.2, InfiniTAM [16] implements the voxel block hashing algorithm[15] with a fix-sized hash table.

In the original InfiniTAM, the number of pre-allocated voxel blocks is fixed as 0x40000, which is sufficient for most of the indoor scenes. However, for single object, this value is far too high and most of the allocated voxel blocks will remain unused as wasted memory space. Because the sizes of objects are typically much smaller than the complete volume of the whole sequence, it is needed to set up a smaller new default value for the objects, while the background "object" still uses the larger value. For the object, we fixed the number of voxel blocks to 0x4000, which is still sufficient to hold objects as large as sofas etc, but will allow the system to hold a larger number of objects without using up all the available memory.

By changing the size of the allocated memory, the system is able to process sequences with about 100 objects of different sizes on a PC with 24GB of RAM.

4.4 Tracking

The tracking of the camera's ego movement is based on InfiniTAM's tracker, which is basically a frame-to-model tracker using iterative closest point (ICP). In the current work, its ability to take the color information into account for the tracking is not exploited. The details of the implementation can be found in section 3.3.1 and [16].

Unlike InfiniTAM, where the tracking and the mapping is performed in one function, in our implementation, the tracking operation is separated from the mapping. It wraps the interface to the tracker from InfiniTAM. For tracking, it takes only the input of a RGB-D image from the reader and a point cloud using raycasting from the mapping module. This opens up the possibility of doing tracking and mapping at different frame rates, which is also effectively used in chapter 5.

Although the incoming RGB-D image with the segmentation masks are split into different object images containing only pixels belonging to the corresponding object, for tracking, the original whole depth image is used. Often the objects are the most important structures in the scene and the background without any objects is often only a plain surface. As the tracker is based on ICP, which matches the point clouds from the raycasting to the incoming frame, a simple plane surface would be prone to drift in the tracking as the plane surface can be matched even with a in-plane drift. One important notice is that this implementation implicitly assumes that the environment is static. To be able to work with dynamic objects, modification is needed to identify and remove moving objects from the tracking to prevent the dynamic objects distort the tracking results.

Because of the use of the whole depth image without segmentation, the point cloud from the reconstructed scene should also contains all objects instead of only the background. For this purpose, a raycasting needs to be done to all visible objects.

The raycasting as a preparation for the tracking of the next frame is done directly after all the voxel scenes of each object is processed. For this purpose, the preparation of tracking is modified to take the vector of visible objects as input and doing raycasting as described in section 4.7.

The prepare operation is done as the last step in the mapping engine. It saves a point cloud in the tracking state shared by the tracking engine and mapping engine. The point cloud is then used by the tracking module to calculate the pose of the next frame, which is then shared back with the mapping module.

4.4.1 Implementation of object level tracking

The previously described tracking method is still based on the whole scene, with the separated objects being added back to the point cloud of the background to enable the tracking, achieving a result which is in the best case the same as a tracking without segmentation. Here we further explore the possibilities of using the additional object information for improving the tracking. Using the pose from the tracking, we are able to associate the objects across the frames, as described in 4.6. This opens the possibility to do tracking for each objects. Here we simply assume the objects are all static, although this can also be used to identify dynamic objects.

Without the whole scene with possibly large areas having only very sparse geometric features, the tracking of single objects could provide additional information to increase the accuracy of the tracking results. By tracking each individual objects, we can distinguish the dynamic objects from the static objects. While the tracking results of the static object can be used to improve the estimation of the camera's ego-motion, removing dynamic objects can also leads to a better tracking accuracy. The dynamic objects' movement can also be tracked in this way providing useful information that can be used in certain scenarios.

After we identified an object in the new frame as already previously seen, a new raycast based on the pose of the new frame is created and the new object depth image is tracked against the point cloud of the existing object. As the initialization pose of the tracking is not the pose of the last frame, but the pose estimation of the new frame coming from the whole image tracking in 4.4. This means the ICP can start with a pose already close to the optimum.

If the tracking is successful, the updated pose will be saved in the tracking state of the object instance, and the new depth image will be integrated into the object's reconstruction using the updated pose from the object tracker. If the tracking is failed, the pose estimation from the whole image will be used to proceed the reconstruction.

We choose not to update the pose of the whole image. Simple tests have shown that the fusion of different camera poses from different objects, using an weighted average, may lead to additional inaccuracy if some object's tracking yield poses which is completely wrong, increasing the chance of a complete tracking failure. This can be addressed as a future extension of the project though.

4.5 Mapping Pipeline

An overview of the mapping module is shown in Figure 4.3. The input of RGB-D image, the segmentation masks and the pose is first pre-processed into object images saved in a vector of objects. For each of the objects, it is decided whether it is a new object or a object already in the map. If it is new, a initialization will be done and new voxel scene is allocated. If it is not new, the new depth image is integrated into the voxel scene already in the map. In the final step, a raycasting is done to create a point cloud to be used for the ICP based tracker. The raycast is also be used for the visualization. After this, the point cloud goes into the tracker and the whole loop starts again.

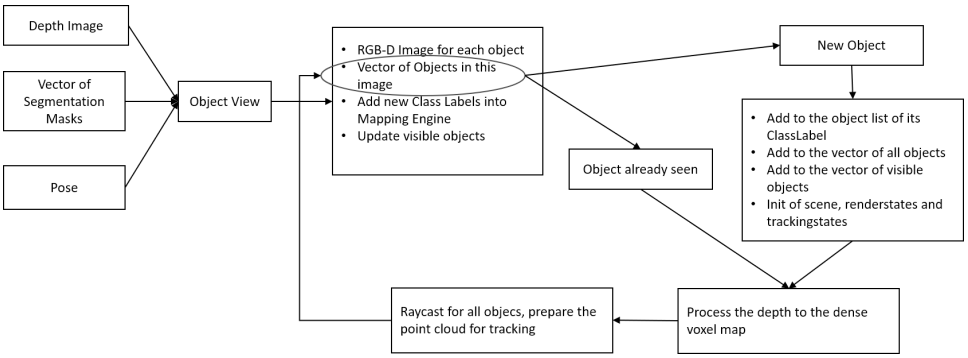


Figure 4.3: Overview of the mapping module

First, we introduce the structure of how the data is organized in the mapping module. Then the preparation of the input RGB-D data will be described followed by the processing from the RGB-D image to the reconstruction.

4.5.1 Data Structure

As the map is represented as a sum of all the objects, the core of the data structure is how to organize the objects. All objects are labeled with their respective categories and also grouped by the labels. All objects of the same label, can be accessed from the label. In this way, maintaining a list of all the labels being enough to access all objects in the environment. Figure 4.4 shows an example of how the objects are organized in the mapping module.

At the same time, in each object, the pointer to its corresponding label is also stored. Furthermore, the object instance stores all relevant information for the reconstruction of the object, including the voxel blocks and hash table, the render state with the rendering parameters and raycasted point cloud, as well as the tracking state with the current camera pose. With each object maintaining their own data, each object can be processed individually without affecting other objects. This enables the basis for a parallel execution of many operations among the objects.

An "anchor view", which is the first camera frame when the object is detected, is also stored in the object. This contains the object image of this frame as well as the pose information from that camera frame and can be used to relocate the object based on the camera pose.

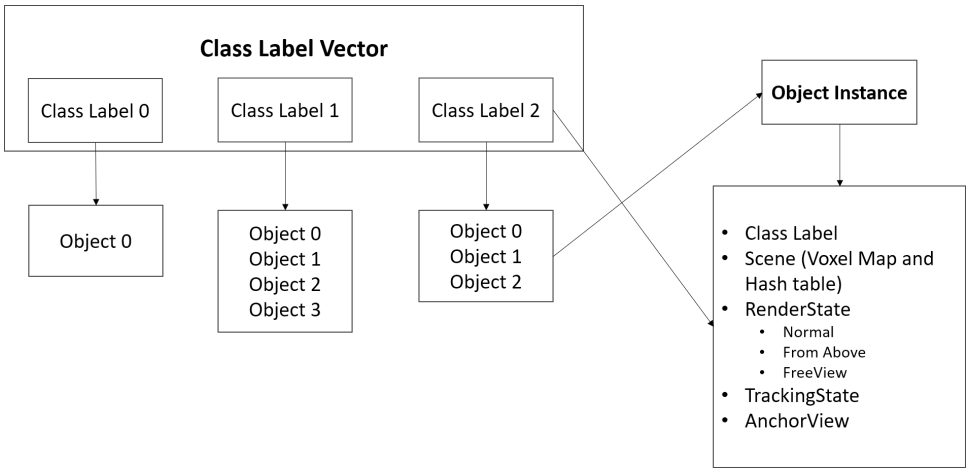


Figure 4.4: Object data structure of the mapping module

Although it would be sufficient to maintain a list of class labels to get all the objects in the current sequence. A separate list for all the objects is maintained for convenience and speed of using only one loop to get all the objects, provided it is not needed to access the objects by the order of labels. In order to make sure we only process objects which are in the visible viewing field of the camera, the objects' visibility is marked and a list of visible objects is maintained parallel to the list of all objects.

4.5.2 Input of tracking results

At the beginning of the mapping process of each frame, the mapping engine is updated with the current pose from the tracking. Using the new camera pose and the known camera internal calibration, the visibility of all the objects are updated. Only visible objects are stored will be processed in this frame. After the visibility of all objects is updated, a clean up of the voxels of the background is performed in order to make sure the background does not contain voxels which belong to objects. This can happen when the segmentation is inconsistent or the segmentation of certain object is missing on some frames. The details of the background clean up is described in 4.8.

4.5.3 Preparation of the input data

As the relative pose of the depth and RGB cameras is usually known as a given 6 DoF transformation, after reading the depth and RGB image, a pixel-to-pixel correspondence from the depth to the RGB image is established using the camera's calibration parameters and the depth values in the frame.

Assume we know the relative transformation between the RGB and the depth camera during from calibration. Using the camera projection equation 3.1, we do camera projection from a world point to both the RGB an the depth camera.

$$z_d \cdot \begin{bmatrix} u_d \\ v_d \\ 1 \end{bmatrix} = K_d \cdot \begin{bmatrix} R_d & T_d \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.2}$$

$$z_{rgb} \cdot \begin{bmatrix} u_{rgb} \\ v_{rgb} \\ 1 \end{bmatrix} = K_{rgb} \cdot \begin{bmatrix} R_{rgb} & T_{rgb} \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.3}$$

Where K_d, K_{rgb} are the intrinsic matrix of the depth and rgb camera, and $[R_d, T_d]$, $[R_{rgb}, T_{rgb}]$ are the rotation an translation matrices from the world coordinate frame to the depth and rgb camera. $[x, y, z]^T$ is the coordinate of the point in the world being projected to both cameras.

To set-up a correspondence of each depth pixel with the pixel of the object mask, which is in the same coordinate frame as the RGB camera, we need to know for each depth pixel, where is the corresponding pixel in the RGB camera. Therefore we reorganize equation 4.2:

$$\begin{bmatrix} R_d & T_d \end{bmatrix}^+ \cdot K_d^+ \cdot depth \cdot \begin{bmatrix} u_d \\ v_d \\ 1 \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{4.4}$$

Where K_d^+ and $[R_d T_d]^+$ are the pseudo inverse matrices.

Plug equation 4.4 into equation 4.3 and assume $z_{rgb} \approx z_d$ because both camera approximately lie on the same plane, we get:

$$\begin{bmatrix} u_{rgb} \\ v_{rgb} \\ 1 \end{bmatrix} = K_{rgb} \cdot \begin{bmatrix} R_{rgb} & T_{rgb} \end{bmatrix} \cdot \begin{bmatrix} R_d & T_d \end{bmatrix}^+ \cdot K_d^+ \cdot \begin{bmatrix} u_d \\ v_d \\ 1 \end{bmatrix} \tag{4.5}$$

Note $[R_{rgb} T_{rgb}]$ is the transformation from world to RGB camera and $[R_d T_d]$ is the transformation from the depth camera to the world coordinate frame. So the middle part of equation 4.5 becomes the transformation from the depth camera to the RGB camera, which we already know:

$$\begin{bmatrix} u_{rgb} \\ v_{rgb} \\ 1 \end{bmatrix} = K_{rgb} \cdot \begin{bmatrix} R_{d2rgb} & T_{d2rgb} \end{bmatrix} \cdot K_d^+ \cdot \begin{bmatrix} u_d \\ v_d \\ 1 \end{bmatrix} \tag{4.6}$$

We now established the association from the depth pixel $[u_d, v_d]$ to the RGB pixel $[u_{rgb} v_{rgb}]$ without needing the absolute camera pose.

We calculate the depth to RGB (and hence also segmentation mask) for each depth pixel at the beginning of the initialization of each frame. The correspondence is saved in a vector for later use when creating the object depth images from the segmentation mask.

This is needed because the segmentation mask, which is generated with the RGB images, has to be aligned with the depth images. For each object mask, the corresponding pixels in the already aligned RGB and depth image are saved as a separate image, referred as an object image. In the object image, pixels do not belong to the object have the value 0. After going over all object masks of the current image, pixels in the image which does not belong to any object are labeled as background and saved as an object image for the background. Simultaneously, it is checked whether the object class is already seen. If not, the object class will be created.

At this stage, although we already know the label of the object, we do not add the objects to the list of objects stored in the label, because it is still unknown if this is a new object or a object already seen in previous frames.

The pose of the current frame is determined by the tracker as described in 4.4. The pose is stored in the tracking state as a 4×4 transformation matrix M from world to camera. The pose stored in the mapping engine is updated with the results of tracking. After the pose is updated, the visibility of voxels in each object is updated using the camera parameters and the camera pose. The object which has zero visible voxel block will be removed from the further processing, with the exception of the free view visualization.

4.5.4 Process the reconstruction

In the preparation, a vector of objects in the current frame is created, but not added to the labels and do not have any voxel blocks allocated. For each of these objects, it will be determined whether the object is one of the objects already seen in the previous frames and hence a 3D reconstruction already exists, or it is a new object which is seen the first time. The detail of this is described in 4.6.

If a objects is new, the following initialization will be performed:

- The object is added to the object list of the label it belongs to.
- The object is added to the list of all objects. Because it is visible in the current frame, it is also added to the list of all currently visible objects.
- The scene for the object is created and the memory for the hash table as well as the pointers to the voxel blocks are allocated. The pointer of the scene is stored in the object.
- A tracking state for the object is created, which is used to store the camera poses from the tracking. Its pointer is stored in the object.
- A render state for the object is created, it is used mainly for visualize each objects independently by storing the point cloud from raycasting and also the final rendered image. The render state also keeps a list of the visible voxel blocks to avoid the need of going over all voxel blocks in each frame. It is also used to determine if a object is in the current viewing field or not. Its pointer is also stored in the object. A second render state is also created and stored in the object for the visualization in a different camera perspective than the current camera pose.

If the object is already seen before, the already existing object will be used for

further process using the object image from the current frame. The object created during the view preparation will be discarded as otherwise it will become a duplicate.

In the next step, voxel blocks are allocated according to the current depth imaged. A integration with the existing voxel blocks is also performed in case the object is already seen previously.

In the last step, a preparation is done to enable tracking for the next frame. As the tracker is based on a point cloud generated from the map, a point cloud from the current frame is needed. As there is now not only one scene containing all the voxels, but multiple scenes for multiple objects, the raycast operation needed to be modified such that the raycasting to all the different scenes create one single unified point cloud, containing all the objects seen in this frame. This is described in detail in 4.7.

4.5.5 Visualization

The visualization of the reconstruction is implemented as the last step of the mapping engine. A point cloud will be created using raycasting from the desired rendering point of view, which is not necessarily the current camera pose. If rendering multiple objects, the multi-object raycast in 4.7 is performed. From the created point cloud, an image of the desired resolution will be rendered and displayed on the GUI. Depending on the setup, images may also be saved to the hard disk.

The visualization is not a crucial part of the SLAM, but only needed for the GUI, as shown in figure 4.2, so we can see the map and the reconstructed objects while it is being built, it can also be switch off together with the GUI turned off.

4.6 Recognition of the same object

The recognition of the same object across different frames is of high importance, in order to fuse reconstructions of the same object over many frames to create a more complete reconstruction and not having duplicates of the same object. The current segmentation providers cannot create perfect segmentation, but only segmentation masks which do not match exactly the object boundary and can differ from frame to frame for the same object.

In this work, the recognition of the object is done using reprojection of the image and the current pose of the camera. Two different criteria are used to determine if a object labeled in the new frame is the same as one of the objects already in the map, if one of them is satisfied then the object is designated as the same.

For this purpose, for each object instance in the new frame, we compare them with all the objects with the same label. The process of checking if a object is new is shown in Figure 4.5.

If an object in the new frame has label A. All objects with the same label A will be taken into account. Because the pointer to the label is directly accessible within the object, there is no need to search through all the labels to find it. As each object stores the information whether it is currently visible, we skip all invisible objects and only consider the visible objects. We refer to the new object image as the new

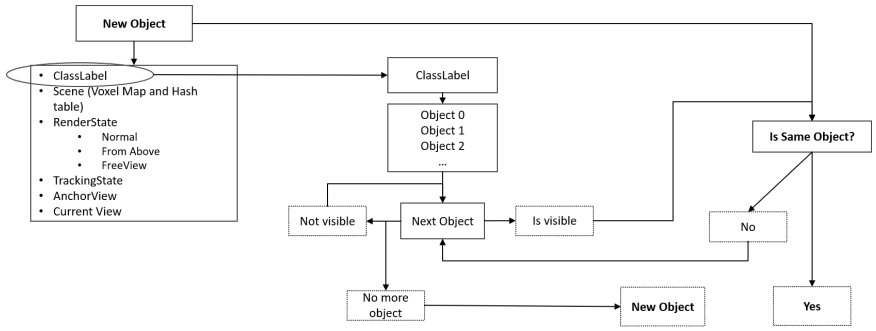


Figure 4.5: Process to determine whether a object is new or existing object

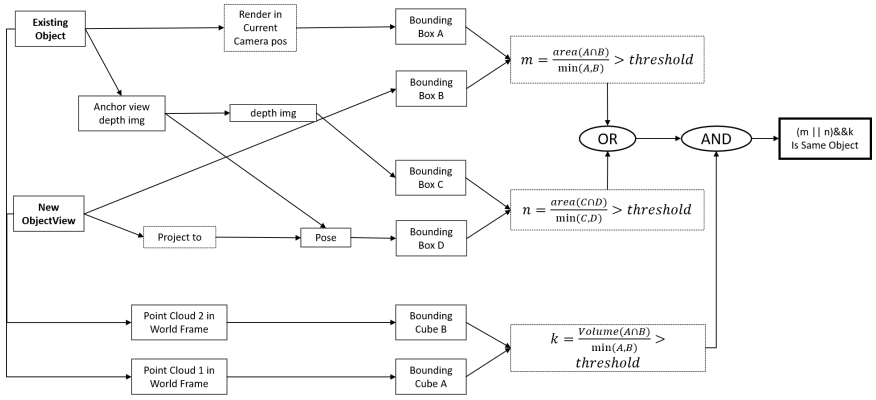


Figure 4.6: Process to check if two objects are the same

object and the object from the list of known object as the old object.

To find out if the new object and the old object is the same object, 3 different projections are carried out and then a overlap calculation is performed, as visualized in figure 4.6.

1. A image of the old object is projected in the current camera pose. This is basically raycasting the voxel blocks of the object using the current pose and render the resulting point cloud into an image. The bounding box the rendered object in the image is then compared with the bounding box of the new object in the new frame. The bounding box is represented as the largest and smallest horizontal and vertical pixel-coordinates which has a non-zero pixel value.

2. The new object image in the new frame will be projected to a 3D point cloud in the world coordinates using its current pose and the depth values with equation3.2. The point cloud is then projected into the anchor view of the old object, using the pose of the anchor view, with equation 3.1. The bounding box of the old object's anchor view compared with the bounding box of the projected new object.

3. From both the old object and the new object, a point cloud in the world coordinate frame is created, using equation 3.2 for the new object and raycasting for the old object with its reconstruction. For both point clouds, a bounding cube similar to bounding box in 2D is calculated.

For both above mentioned bounding box comparisons in 1 and 2, give bounding boxes A and B, the measure of intersection over minimum is used to determine is the objects are the same.

$$Overlap = \frac{Area(A \cap B)}{\min(Area(A), Area(B))} \tag{4.7}$$

For the bounding cube comparison in 3, it is similar to the bounding box comparison, only comparing volumes instead of areas.

$$Overlap = \frac{Volume(A \cap B)}{\min(Volume(A), Volume(B))} \tag{4.8}$$

The 3D bounding cube overlap check acts only as a confirmation. Only if the ratio of bounding box overlap in 2D is larger than the threshold in one of the two 2D comparisons, the 3D overlap will be carried out. If the 3D overlap check is also passed, the object in the new frame is considered as the same object in the map, and the depth information will be integrated to the old object in the map.

Another common measure for overlap if the intersection over union measure:

$$Overlap = \frac{Area(A \cap B)}{Area(A \cup B)} \tag{4.9}$$

Because it is possible that a segmentation mask may not cover the entire area of a object, or a object is only partially seen in the previous frame and more of the object moved into the frame as the camera moves, intersection over minimum is chosen over the intersection over union method. If partially seen object is fully revealed in the following frame, 4.9 will produce a smaller ratio than 4.7, increasing the chance that it falls below the threshold. Using 4.9, a perfect segmentation and perfect camera pose information would then always lead to 100% overlap ratio. We want to ensure that partially seen objects and none-complete segmentation masks do not result in multiple objects, because this can reduce the reconstruction quality and lead to an increased memory and resource usage with the duplicated object.

In order to achieve this, the ratio threshold is chosen to be a relatively low value of 0.25.

If none of the objects in map with the same label have enough overlap, the object is considered as a new object, initialized and then added into the map.

4.7 raycast into multiple objects and preparation for the ICP tracker

For the purpose of tracking as well as for the visualization and rendering the images, a point cloud needs to be created from the voxel scene. InfiniTAM implemented a raycast function which is able to do the raycast operation given the voxel scene and the pose of the camera view which is doing the raycast. However, this only works with one single voxel scene.

As opposed to the original InfiniTAM, our map contains multiple object which are multiple standalone scenes. Changes to the raycasting procedure need to be implemented to be able to raycast all objects to a single point cloud containing all objects in the current visible range.

When doing raycasting to only one voxel scene, occlusion of surfaces is not a problem, because the virtual ray from the camera stops at the first non-empty voxel it encounters anyway. However, in the multi-object world representation in this work, raycasting needs to be done to multiple object, where parts of one object can be blocked due to other objects between the object and the camera. There is no a priori knowledge about which object is blocking which object, hence it has to be determined which object surface is the closest to the camera viewpoint.

As described in 4.5.1, each object instance in the map has its own render state and tracking state to store the results of the raycasting. When doing raycasting operation with multiple objects, the raycasting is first done to each object scene separately, creating a separate point cloud for each object. As each object is independent at this stage, the process can be carried out in parallel.

The point clouds are in the form of an "image" of the same resolution as the depth images, where each pixel of the image is a 3D vector representing the point in the world coordinate frame. In a second step, the points from each object needs to be compared and from each pixel, the point closest to the camera viewpoint is saved in the final point cloud.

The points in the point cloud, which are in the world coordinate frame, are transformed to the camera's coordinate frame using the camera pose M.

$$pt_{camera} = M \cdot pt_{world} \tag{4.10}$$

The point pt_{camera} is in the camera frame with the origin in the center of the camera image plane. The euclidean norm of the point's coordinate is then the distance to the camera view point and for each ray. For each pixel, we compare the corresponding point of all objects the point with the smallest distance is chosen. This process can be run in parallel for each pixel, and in each pixel the loop over all the possible points are run serially to find the point with minimum distance.

The result of the multi-object raycasting is used for rendering the visualizations. For the tracking, the point cloud is further process it into a ICP map which is then used to track the camera's current pose. Further, the point clouds can also be used for further applications such as collision avoidance.

4.8 Object Voxel Removal from Background

The segmentation data in the 2D frames can be very inconsistent. Objects failed to be detected in some frames or only partially masked as the object. This leads to the part of the frame which actually belongs to objects being unlabeled and being interpreted as background. As we integrate the depth image into the scene frame by frame, voxels previously unseen parts of objects will be added and fused with the object. This is a desired behavior when processing objects, as it can make sure we can reconstruct the whole object even if some part of the object is not labeled

correctly in one of the frames. However, this also leads to the same behavior in the background object. Once a object is failed to be labeled, it will be added into the background and stays in the background. In a longer sequence, this means the background will eventually contain most objects, which is against the whole idea of reconstruct the background and each object separately. Therefore a background cleanup algorithm is needed to remove the voxels in the background, if it is find out that a object contains the same voxel. In the current implementation, the voxel level deletion is not implemented. The clean up is therefore performed on the level of the voxel blocks, which are cubes of $8 \times 8 \times 8$ voxels.

For this purpose, all objects in the visible list are considered. We register the positions of voxel blocks of each visible object in a vector. The visible list of voxel blocks in each object's live pose render state helps to only limit this operation to visible voxels in each object only. After the vector of all voxel block positions of all visible objects is created, the voxel block in the same position is searched in the background.

To search in the background, we first need the hash value of the position in the background scene. Note that due to the difference in the hash table size, for the same position, the hash value in the object scene is different to the hash value of the background scene. The hash value needs to be recalculated based on the position value. Using the calculated hash value we can restrict the search inside the hash bucket with the same hash value. We search through all voxel blocks stored in this hash bucket and compare the position of the voxel block with the position we want to delete.

If no match is found, we move to the next position in the vector. If a match is found, the voxel block is deleted. Each voxel in the voxel block is reset to zero and the voxel block is deleted. The pointer to the voxel block in the hash table entry is also set to -1, indicating it is not allocated. The result is a clean background scene with all objects completely removed from the background.

Because of the need of the visible list of voxel blocks for each object, the cleanup operation is performed directly after the pose is updated in the mapping module and the visibility of all objects and their voxel blocks is updated. If an object is newly detected, the corresponding voxel blocks will only be removed starting from the next frame.

4.9 Object Cleanup

A similar problem to 4.8 exists for the objects, only in the opposite direction. Sometimes the segmentation mask grows over the boundary of the object to include some area of the background. This can especially happen when the object has some fine structures like holes, where the background viewed through the hole is also masked as the object. Inspired by the voxel baggage collection in [52], a similar strategy is developed to find out unwanted voxels from the object.

As already implemented in InfiniTAM, each voxel has a weight parameter, which is used for the weighted average fusion of multiple observations. The weight is incremented by 1 every time the voxel is observed in the depth image and fused by the mapping module. It is basically the number of the depth images which has a corresponding observation of this voxel.

Wrong segmentation at the object boundary might only happen in a few frames and with the viewpoint changing, some areas not belonging to the objects may be no longer masked as the object. On the contrary, the true object area usually get the correct object mask in most of the frames regardless of the viewpoint change. This means the voxels which truly belongs to the object should have higher weight than voxels which should not be part of the object.

This leads to our approach that we can find out the highest possible weight a voxel can achieve and compare the voxel's actual weight to it. If the actual weight is much smaller than the possible maximum weight of the voxel, we remove the voxel from the object.

The highest possible weight that a voxel can get is basically the amount of frames in which the voxel is in the viewing range of the camera. This is however different for each voxel. Different objects can stay longer or shorter in the camera view depending on the camera moving speed and the size of the object. Even inside one object, the number can also vary due to the object being only seen partially in the camera, because it is very large or very close to the camera.

This means a individual maximum possible weight for each voxel needs to be recorded. We add a view count as a short integer value to each voxel in order to record the number of frames that can see it. Because the visibility of the object and its voxels is updated at the beginning of each mapping step anyway, each time a label is labeled as visible, the view count is also incremented by 1.

During the cleanup, we define the observation ratio as the ratio of the weight of the voxel and its view count.

$$ratio_{observation} = \frac{weight}{view\ count} \tag{4.11}$$

If the ratio is below a threshold, the voxel is deleted from the object.

Chapter 5

Experiments and Results

We evaluate the system on a computer running Ubuntu16.04 LTS with a quad core Intel i7 930 CPU and 24GB RAM.

We perform experiments using the ICL-NUIM dataset [44] and the Microsoft 7 Scenes dataset [47].

The ICL-NUIM dataset is a rendered synthetic dataset containing an office scene and a living room scene with 4 sequences each. The sequences are between 800 and 1500 frames in length. Because it is a synthetic dataset, for each frame, the depth image with perfect depth measurements is given. Additional noise is added to the rendered depth to provide a closer simulation of the real depth cameras, where the measurements are noisy. The groundtruth of the camera pose is also provided as quaternions and position vectors

The Microsoft 7 Scenes dataset is a real dataset recorded with Microsoft's Kinect camera. For each of the seven scenes, several sequences with 1000 frames each are provided. The camera pose groundtruth is provided as 4D rigid body transformation matrices. In our experiment we only use the office and chess scene.

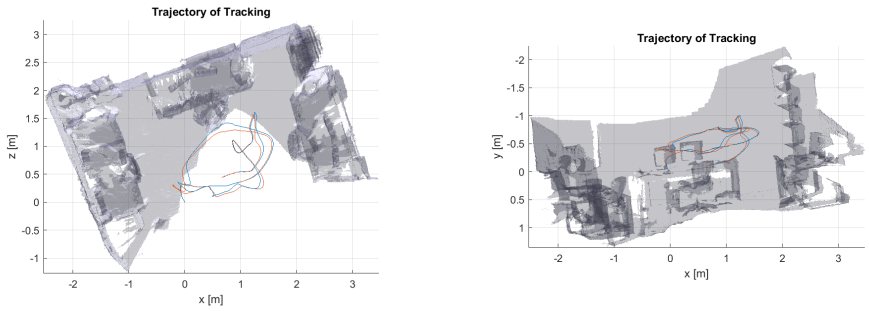
5.1 Tracking Accuracy

First we evaluate different setups with respect to tracking accuracy. The tracking results is evaluated and visualized. The absolute localization error with its mean, maximum and minimum is evaluated. Additionally the trajectory is visualized in 2D from the view above.

As the algorithm is running comparably slow and time/memory consuming, the algorithm's mapping part is run on every 3rd frame in the dataset, mimicking the effect of fixed frequency of key frame. The tracking is still performed on each frame. We run each experiment once with segmentation and once without segmentation. With segmentation off, the segmentation masks are not provided, the system therefore behaves exactly like the original InfiniTAM with the whole RGB-D image as input. This can provide a basic baseline to compare with.

In figure 5.1 and 5.3 the estimated camera trajectory (orange) and the groundtruth

trajectory (blue) are displayed. Although both have some drift compared to the groundtruth, the trajectories with label on or off do not change the estimated trajectory much. This is verified in the tracking error plots in figure 5.2 and 5.4. The largest error is seen in Z-direction, which corresponds to the direction of the optical axis at the first frame. This is an indication that the tracking has more difficulties with the the movements in the camera viewing direction. This appears however in both segmentation on and off, indicating this being an issue with the tracker, which is not caused by reconstructing objects separately.



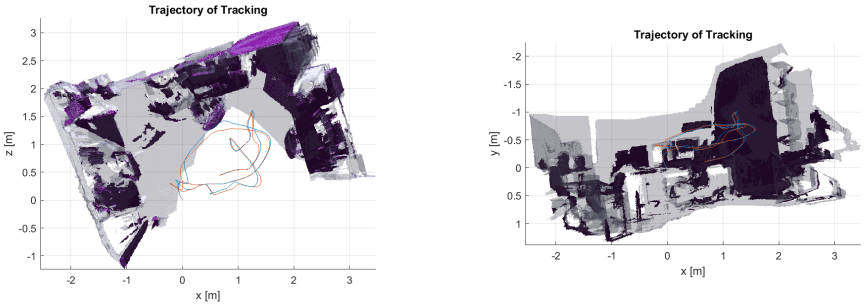
(a) XZ-Plane (b) XY-Plane

Figure 5.1: Trajectory of MS Dataset Office 1 with segmentation off from 2 views

Figure 5.2: Tracking error [m] of MS Dataset Office 1 with segmentation off

In table 5.1 and 5.2, the tracking error with segmentation on and off with different datasets are shown. The groundtruth trajectory and the estimated trajectory are aligned in position and orientation at the first frame, and the mean, standard deviation and the maximum of the absolute tracking error are evaluated.

In most datasets, the tracking errors of segmentation on and off stay in the same order of magnitude. In Microsoft Office dataset sequence 2, segmentation on provided a better accuracy of about 50%. In ICL-NUIM Living Room 2, segmentation on increased the error. In Living Room 3, regardless of segmentation on or off,



(a) XZ-Plane (b) XY-Plane

Figure 5.3: Trajectory of MS Dataset Office 1 with segmentation on from 2 views

Figure 5.4: Tracking error [m] of MS Dataset Office 1 with segmentation on

both tracking has failed in the middle of the sequence, because part of the sequence was recorded with the camera close to a flat wall. With no significant geometric features, the ICP based tracking algorithm cannot work. This however also shows the importance of a relocalization method in case of a failed tracking.

5.2 Background Cleanup

The result of the background voxel cleanup is discussed in this section. Figure 5.5(a) shows the reconstructed background of the living room 1 sequence of ICL-NUIM, while figure 5.5(b) shows the same with background cleanup turned on. It is obvious that the couch on the left hand side of the image and the table and the TV in the middle of the image are removed.

However, removing objects by whole voxel blocks has a significant drawback. Even if only 1 voxel belongs to an object, the whole voxel block with 512 voxels is removed

	Office 1		**Office 2**		**Chess 1**		**Chess 2**	
Segmentation on/off	Off	On	Off	On	Off	On	Off	On
Mean [m]	0.0484	0.0471	0.0612	0.0270	0.0280	0.0239	0.0159	0.0168
Std[m]	0.0446	0.0462	0.0543	0.0204	0.0107	0.0085	0.0072	0.0089
Max[m]	0.1795	0.1741	0.1732	0.0814	0.0754	0.0626	0.0391	0.0402

Table 5.1: Tracking Errors using Microsoft Dataset Office and Chess Sequences

	Living Room 0		Living Room 1		Living Room 2		Living Room 3	
Segmentation On / Off	Off	On	Off	On	Off	On	Off	On
Mean [m]	0.1601	0.0526	0.0122	0.0103	0.0449	0.0811	1.2149	1.0304
Std [m]	0.2792	0.1518	0.0044	0.0055	0.0294	0.0758	1.5569	1.4006
Max [m]	0.8585	0.8934	0.0239	0.0539	0.1721	0.3535	4.1388	3.9639

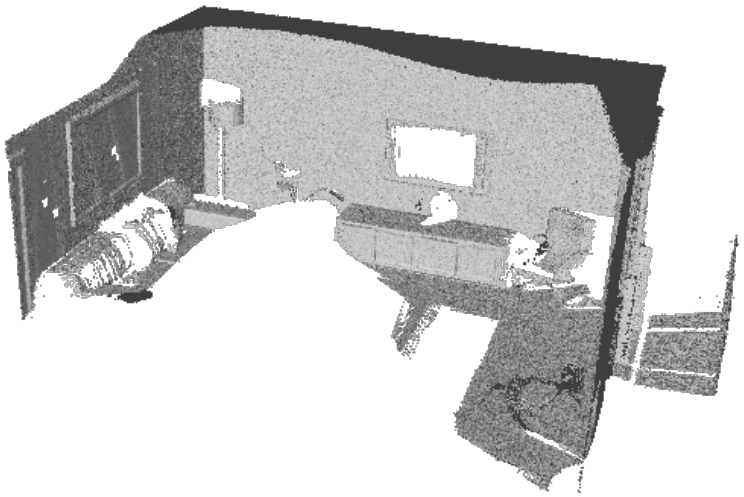
Table 5.2: Tracking Errors using ICL-NUIM Dataset Living Room Sequences

from the background, creating a zigzag shaped boundary and thus a gap between the background and the reconstructed object. This gap could reduce the accuracy of the tracking, as well as have a negative impact on path planning and other use cases. In figure 5.6 the gap is clearly visible in the fused visualization, around the TV and the couch in the two images. The regular cubic shape of voxel blocks can also be seen.

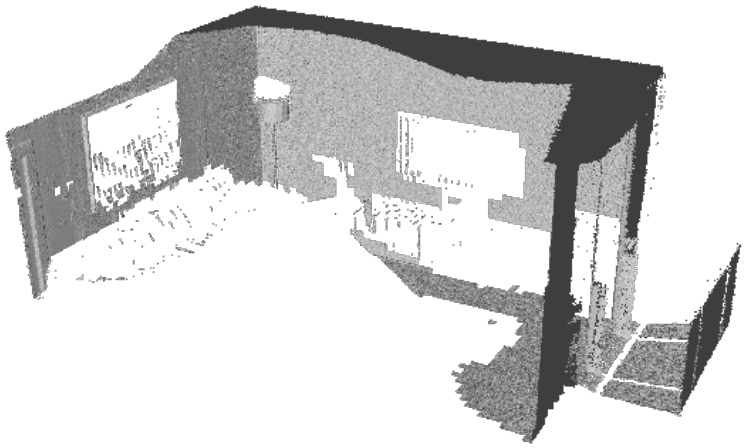
To overcome this problem, other methods of background object removal, including voxel location based removal of background voxels, is currently being evaluated. A background voxel removal similar to the object cleanup in section 5.3 might also help.

5.3 Object Cleanup

The removal of segmentation artifacts from the reconstructed objects, described in 5.3 is shown in this section. This is necessary because object segmentation is far from perfect and segmentation errors affect the reconstruction quality of objects significantly. As an example, figure 5.7 shows 3 consecutive frames with their segmentation masks. In the first frame 5.7(a), the segmentation clearly labels the monitors, the keyboard and the office chair with only small errors in the boundary. However, in the 2 following frames, the segmentation only detects one object, where its mask covers the office chair and part of the monitor and the keyboard as one object. These errors are projected to the depth image. Because the reconstruction will fuse all depth frames into the 3D model, this kind of segmentation error in even one single frame will result in the reconstructed object containing unwanted parts not belong to the object. In figure 5.8 the reconstructed chair is shown. It is very clear that it has contained parts of the ground, parts of the table as well as some more voxels in the left side.



(a) No Background cleanup

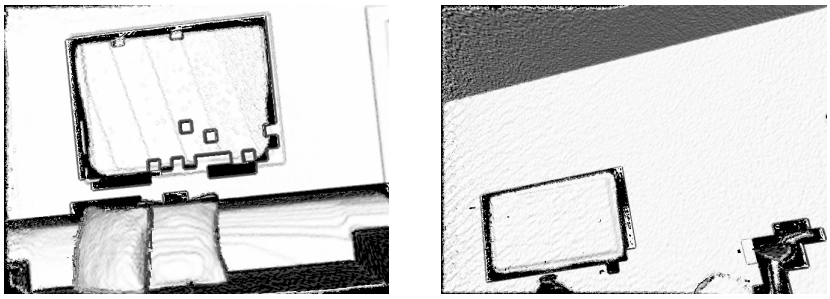


(b) With Background cleanup

Figure 5.5: Reconstruction results of the background object with and without background object removal.

The removal of these unwanted artifacts is described in section 4.9. An arbitrary choice of threshold 0.9 improved the reconstruction significantly, which can be seen in 5.9. The majority of the table and the ground is removed from the office chair. However, it is also visible that some part of the chair itself got removed, including the left arm rest and several voxels in the back of the chair.

The reason for the removal of correct parts of the chair goes back to how the visibility of the voxel is determined. The removed arm rest is partly occluded by the back of the chair in some of the frames. This results in the arm rest getting less depth observations. However the visibility evaluation does not take occlusions into account, which means in the frames, where the arm rest is covered and cannot be seen, it is still been classified as visible, as it is still in the camera's field of view. As the denominator in equation 4.11 is larger than it should be, the voxels are being removed because the ratio falls under the set threshold.



(a) Gap between couch and wall (b) Gap between TV and wall

Figure 5.6: Gaps between objects and the background caused by the background voxel removal

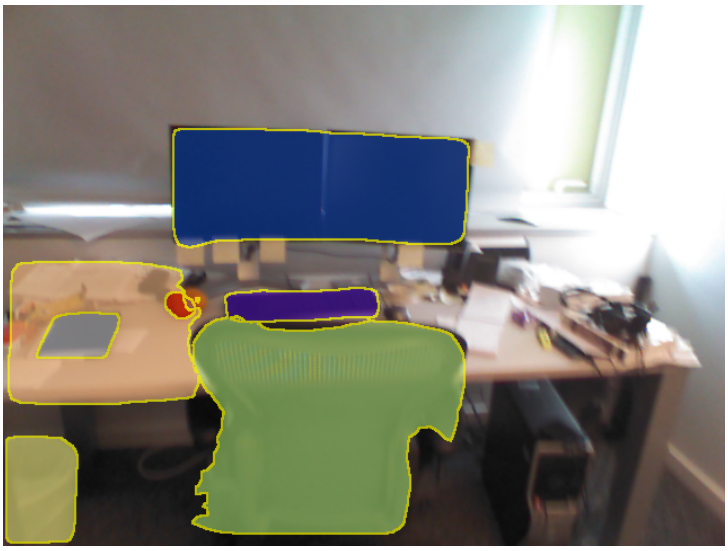
Different threshold are tested from 0.3 to 0.9, which is shown in figure 5.10. For the purpose of visualize the clean up, the dataset is only run up to the 300^{th} frame. Only chairs (purple) are shown over the reconstructed background. For the chair in the right, we can observe that the clean up has come into effect very well, while for the chair in the left, there is still many unwanted voxels remaining.

Looking at the orange colored trajectories in figure 5.11, we notice that the left chair is mostly viewed from the same direction, while for the right chair, the viewpoint moves from the right side of the chair over to the left side of the chair. Similar viewpoints may yield similar segmentation mistakes more often. When the same false segmentation happens too often, it may go over the threshold and therefore not be cleaned up by the algorithm. It can also be noticed that all the false segmentation of the left chair are coming from the same viewing direction, which is from camera looking in the top-left orientation of figure 5.11. A slight larger segmentation mask than the chair will result in the part of the table and ground behind the chair being classified as chair.

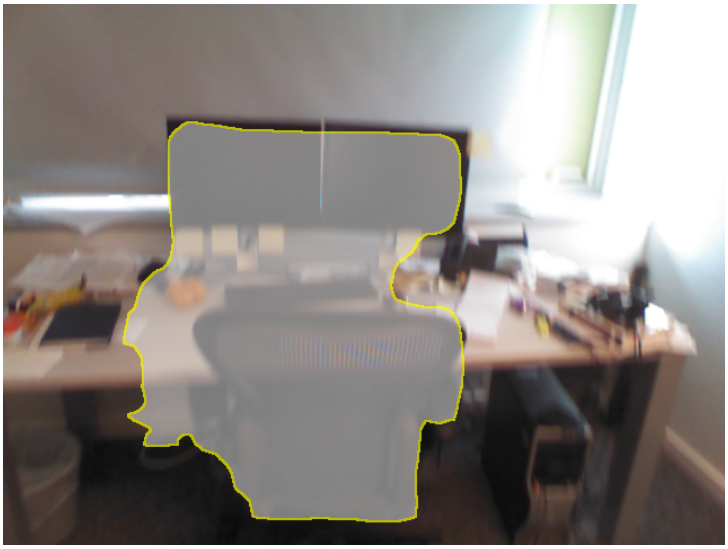
As the results above showed, the current object clean up method works fine for voxels which are occasionally wrongly labeled in a small number of frames, especially when a object is viewed from different camera viewpoints. If a object is only viewed from one single viewpoint, the chance of similar segmentation errors will increase and this results in false voxels not being correctly identified and removed. As segmentation errors in general cannot be avoided, further exploiting geometric clues, similar to [8], such as voxel connectivity or the gradient of the surface, can improve the recognition of the boundaries of objects and hence further improve the reconstruction of objects.

5.4 Multi-Object Raycast

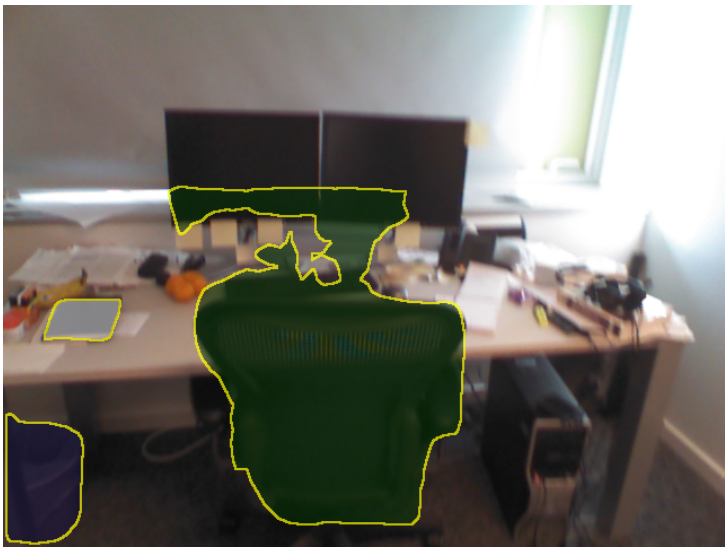
Figure 5.12 shows the result of a multi-object ray cast with all objects and the background. The couch and the TV on the wall are visible. Figure 5.13 is the same frame with only the background in the point cloud created by a normal ray cast. If the object is in front of another object or the background, the surface closest to the camera will be used to create the point. These point clouds are used for visualization and tracking the next frame.



(a) Frame 148



(b) Frame 149



(c) Frame 150

Figure 5.7: Segmentation errors in 3 consecutive frames with segmentation masks in MS Office 2

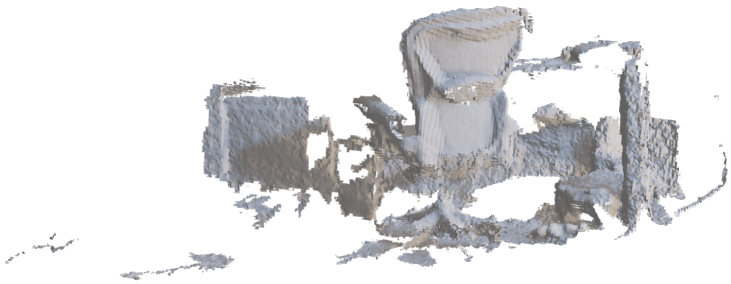


Figure 5.8: Reconstructed office chair with many parts not actually belong to the chair

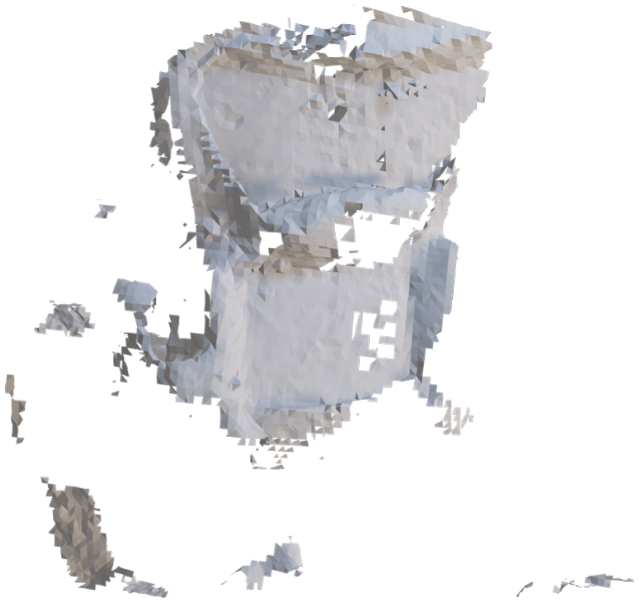
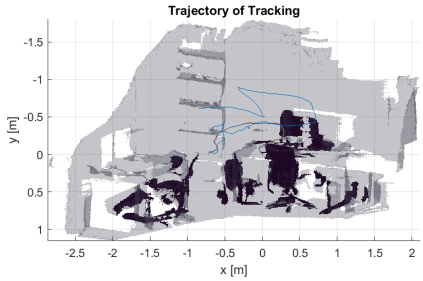
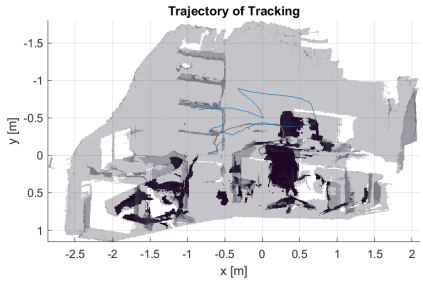


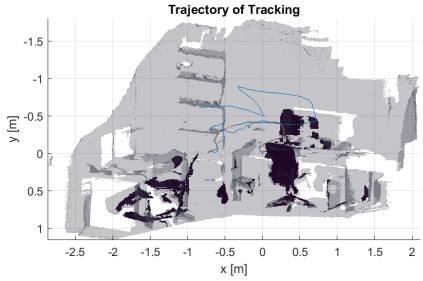
Figure 5.9: Reconstructed office chair with cleanup



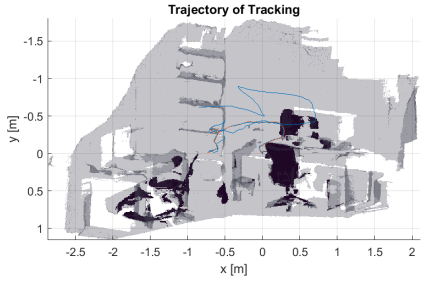
(a) threshold = 0.3



(b) threshold = 0.5

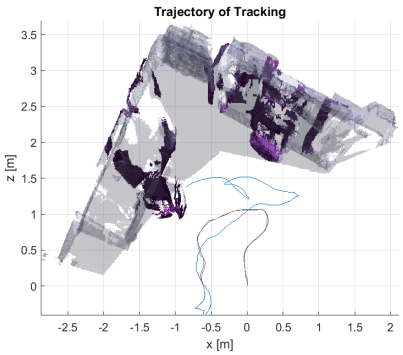


(c) threshold = 0.75

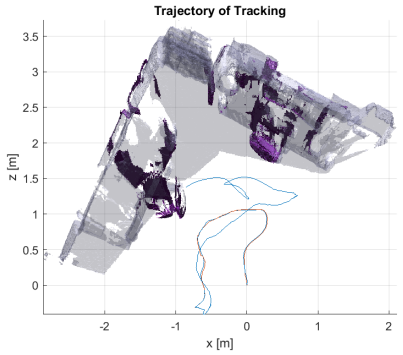


(d) threshold = 0.9

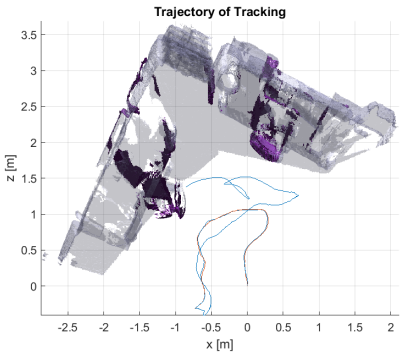
Figure 5.10: MS Office 2 reconstructed chairs with different object cleaning threshold



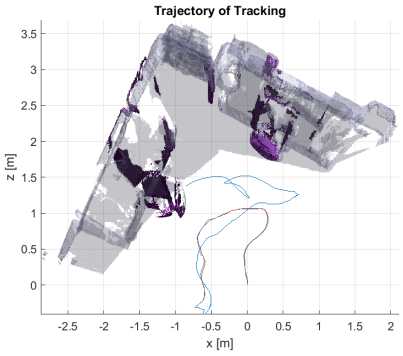
(a) threshold = 0.3



(b) threshold = 0.5



(c) threshold = 0.75



(d) threshold = 0.9

Figure 5.11: MS Office 2 reconstructed chairs with different object cleaning threshold, view from top

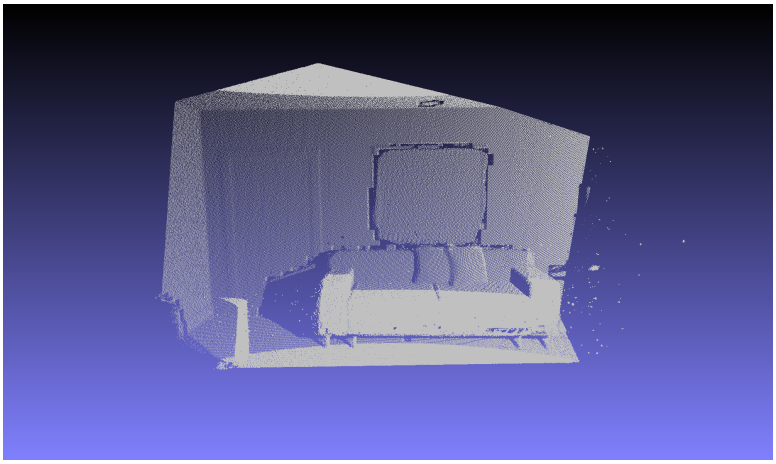


Figure 5.12: Point cloud with all objects from multi-object ray cast

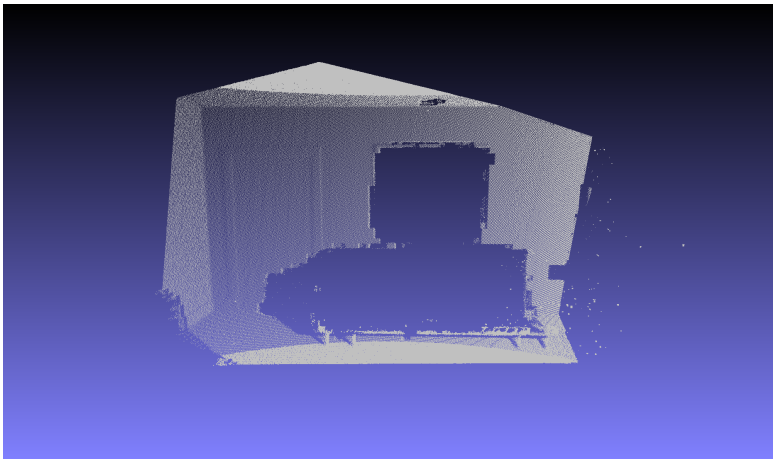


Figure 5.13: Point cloud of the background only. Note the space left behind by the couch and the TV

Chapter 6

Conclusion and Future Visions

6.1 Conclusion

In this work, we aimed at creating a dense mapping and localization system with RGB-D sensors on the level of objects. Although currently coming short at the real-time capabilities, the system could be the starting point of many future works. The system demonstrated a structure to build a standalone map for each object separately, creating a 3D reconstruction for each object. While maintaining the independence of different objects by creating independent render and tracking states as well as their own anchor view pose. Many resources are shared as long as they do not involve individual object properties. The objects as a central part of information can be easily extended with other additional information, such as the property of whether it is movable or deformable. Each object can be visualized and exported as a mesh or a point cloud, this also enables easy further processing of objects in the map.

We developed a simple and efficient way to associate objects across different frames. As long as the object is labeled as the correct class, even small objects can be associated. To make sure this works is essential for the mapping to work, as duplicate object would double the resources needed and could possibly corrupt the further processing such a object based place recognition.

We also created methods for removing unwanted artifacts created by imperfect or partially failed segmentation. By fusing multiple frames together, the probability that an object remains undetected or only partially detected is reduced. Even if some frames fails to detect the object, the object can still be successfully reconstructed. The voxel clean up based on the view count provides an efficient way of removing voxels that should not belong to the object. It makes use of the weight property of the voxel, which is needed for the weighted average already and only adds the view count property, which can be simply incremented during the obligatory visibility update, it doesn't add much extra computation, which is good for cleaning up while doing the tracking and mapping.

The background cleanup ensures the background to contain as little object as pos-

sible, creating a pure background which can be used as a basic map in case the objects in the scene is movable.

6.2 Future Visions

Based on the solid basic work provided by this system, many further applications can be built on it. With the already mentioned shortcomings in many areas, it also has much of room of improvements in many aspects.

6.2.1 Improvement of speed and memory consumption

One of the most important problem was the slow running speed and the relatively high memory consumption. One of the most promising gain would be to utilize the power of GPU for many pixel or voxel level operations, which can be executed largely in parallel. With the restricted parallel computing power of one single CPU, many operations able to be paralleled can only work in single thread.

Another issue is the amount of memory used. Although already decreased the size of the hash table and voxel block array drastically in 4.3.1, the memory usage still scales with the amount of the objects in the memory. Objects invisible for longer times and far away from current viewpoint could for example be moved out of the memory to the hard disk. Many objects detected are also so small that they may only have a very small reconstruction of several voxels. Such object could be discarded from the system easily with their voxels fused into the background. This would save a large amount of memory for storing the hash table and the pointers to the voxel blocks, and also save quite a lot processing power when doing operations for all objects.

The raycasting to multiple objects as well as rendering the visualization also cost a large amount of computation power. For operations which do not need rendering, removing these will speed up the system. As for the raycasting as a preparation for the tracking, 6.2.2 provides some further thoughts on improving the tracking and getting independent from the mapping module as well as the raycasting step.

One last point is also the way the segmentation mask is represented. The current masks are all plain text the same size as the image resolution, with huge amount of 0s representing the background that is only a waste of space and computation. Using a more compact way to represent the mask and integrating the python Keras based segmentation to the tracking and mapping operation would increase the performance of the system by reducing the file IO dramatically. The additional real time computing need for the segmentation should still be considered though.

6.2.2 Implementation of other tracking methods

The extended ICP based method from InfiniTAMv3[16] is although usable, it has many areas which leads to the desire of other tracking methods.

Being the nature of a frame-to-model tracker, it requires the frame to be close to the model in order to be able to match. This is not suitable for tracking higher speed

movements where the viewpoint can changed drastically from one frame to another. The requirement of the model to be up-to-date also means that the tracking cannot really be independent from the mapping. With our framework, it is possible to implement a frame-to-frame visual odometry to track the camera's movement.

Being independent from the building of the map, and thank to the separation of the mapping and the tracking module, both modules can also be executed in parallel. A key-frame based implementation would also be feasible, while the tracking module tracks each frame on a frame-to-frame basis in real time, the mapping module can only integrate the key-frames into the map, reducing computation cost and the need of segmentation for each of the frames.

6.2.3 Place detection and loop closure

Salas-Moreno et al. [5], Lee et al. [9], Finman et al. [10] and Don et al. [40] all demonstrated a system capable of detecting previously visited locations by using the information of relative position between objects in the form of a sparse connected graph. Objects are the vertices of the graph and the distance and orientation between objects form the edges of the graph. Lowry et al. [1] also pointed out that topological map with metric information, together with a location prior, could be useful for robust place recognition in changing environments. While Lee et al. [9] only uses graphs in single image frames, Salas-Moreno et al. [5] and Finman et al. [10] demonstrated a system to be able to use graph matching to recognize places and initialize loop closure optimization. As we have the reconstruction of each object as well as the pose information of their anchor views, it is straightforward to create an object graph for place recognition and optimizing a pose graph and the associated objects after a loop is detected. The single static background could lead to problems as it is rigid. However, like Prisacariu et al. [16] have demonstrated in InfiniTAMv3 with the loop clousure version, using multiple sub-maps instead of one single large static map as the background would be a feasible option to make loop closure optimization possible for the rigid background. In my opinion, this is one of the most important extension of the current system, to complete the place detection and loop closure capabilities in order to make the current system a full SLAM framework capable of building globally consistent dense object maps.

6.2.4 Dynamic Object Tracking

The current work completely based on the assumption that the mapped environment stays static. However this is seldom the case in the reality. With small modifications to the object tracking described in section 4.4.1, it is possible to identify and separate dynamic objects from statics object. One can also infer the object's label to help determine whether an object could be movable. For example trees or buildings are unlikely to move, while a car, even if it is static, could become dynamic at some time.

To identify dynamic object, it is only needed to identify objects which yields a pose largely different from the global pose. The difference between the global tracking result and the object tracking result is then the movement of the object during this frame.

Dynamic objects should then be removed from the global tracking step and being

tracked separately. The map's consistency is guaranteed and the moving object is also tracked and can be used for predicting its movement.

6.2.5 Extension to stereo with outdoor usage

The system is built with the use of RGB-D sensors like Microsoft's Kinect in mind. However, the information needed is a RGB image as well as a depth image and their relative transformation. Using stereo cameras, it is also possible to create a depth map using image from stereo cameras such as Geiger et al. [53], Mayer et al. [54]. Despite being noisier than the depth image from a RGB-D sensor, the algorithm for the rest would basically stay the same. Given enough distance between both cameras, stereo cameras can provide depth sensing for a longer range than RGB-D sensors, which can only read depth reliably up to the distance of several meters. The longer range would enable the use of the system in outdoor environment such as for autonomous driving and drones. Tateno et al. [55]'s approach of learning the depth map in an end-to-end fashion can also be an interesting research point.

6.2.6 Refinement of segmentation with geometric cues

Mask-RCNN[11] provides a good semantic instance segmentation of the objects. However, the boundary of the objects masks do not match exactly the real boundary of the objects. Objects often have a very different appearance from the background and are usually disconnected with the background or have some sharp edges at the location when they meet the background. Such appearances will lead to high gradient edges and corners in the RGB and in the depth image. Using such information to refine the semantic object mask before splitting and creating the object images could increase the accuracy of the boundaries of the reconstructed objects and create a more clear separation between object and background.

6.2.7 Post-Processing of reconstructed objects

One further possible extension focuses on the quality of the reconstructed objects. As we can see in figure 5.9, event the cleaned chair has a very rough surface and some parts of the chair could be missing due to missed segmentation or object cleanup. This can be improved in a post-processing step, which should ideally run in a separate thread. The surface of the reconstruction can be smoothed and small holed in the surface should be filled. Using symmetry information similar to Cohen et al. [56], it is also possible to complete the missing part of reconstructed object and remove failures due to drifts in the tracking. This could create more complete object reconstructions with higher quality, which could be used directly in AR / VR or processed for 3D printing, etc.

List of Figures

Bibliography

[1] S. Lowry, N. Sunderhauf, P. Newman, J. J. Leonard, D. Cox, P. Corke, and M. J. Milford, "Visual Place Recognition: A Survey," *IEEE Transactions on Robotics*, 2016.

[2] J. M. M. Montiel, "ORB-SLAM : A Versatile and Accurate Monocular," vol. 31, no. 5, pp. 1147–1163, 2015.

[3] R. Mur-Artal and J. D. Tardos, "ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras," *IEEE Transactions on Robotics*, vol. 33, no. 5, pp. 1255–1262, 2017.

[4] D. Gálvez-López and J. D. Tardós, "Bags of binary words for fast place recognition in image sequences," *IEEE Transactions on Robotics*, 2012.

[5] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. Kelly, and A. J. Davison, "SLAM++: Simultaneous localisation and mapping at the level of objects," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 1352–1359, 2013.

[6] B. Drost, M. Ulrich, N. Navab, and S. Ilic, "Model globally, match locally: Efficient and robust 3D object recognition," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2010, pp. 998–1005.

[7] I. A. Bârsan, P. Liu, M. Pollefeys, and A. Geiger, "Robust Dense Mapping for Large-Scale Dynamic Environments," Tech. Rep.

[8] L. Agapito, "MaskFusion: Real-Time Recognition, Tracking and Reconstruction of Multiple Moving Objects," Tech. Rep.

[9] B. Lee, J. Jeon, and J. Oh, "Place recognition for visual loop-closures using similarities of object graphs," *Electronics Letters*, 2015.

[10] R. Finman, L. Paull, and J. J. Leonard, "Toward Object-based Place Recognition in Dense RGB-D Maps," *Icra*, 2015.

[11] K. He, G. Gkioxari, P. Dollar, and R. Girshick, "Mask R-CNN," *Proceedings of the IEEE International Conference on Computer Vision*, vol. 2017-Octob, pp. 2980–2988, 2017.

[12] W. Abdulla, "Mask r-cnn for object detection and instance segmentation on keras and tensorflow," https://github.com/matterport/Mask_RCNN, 2017.

[13] R. A. Newcombe, 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 *," Tech. Rep.

[14] 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," *IEEE Transactions on Visualization and Computer Graphics*, 2015.

[15] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger, "Real-time 3D reconstruction at scale using voxel hashing," *ACM Transactions on Graphics*, 2013.

[16] V. A. Prisacariu, S. Golodetz, M. Sapienza, T. Cavallari, and D. W. Murray, "A Framework for Large-Scale 3D Reconstruction with Loop Closure," 2017.

[17] H. Pfister, M. Zwicker, J. Van Baar, and M. Gross, "Surfels: Surface Elements as Rendering Primitives," *Proceedings of the 27th*, vol. pp, no. May, pp. 335–342, 2000.

[18] T. Whelan, S. Leutenegger, R. Salas Moreno, B. Glocker, and A. Davison, "ElasticFusion: Dense SLAM Without A Pose Graph," in *Robotics: Science and Systems XI*, 2015.

[19] J. McCormac, A. Handa, A. Davison, and S. Leutenegger, "SemanticFusion: Dense 3D Semantic Mapping with Convolutional Neural Networks," 2016.

[20] V. A. Prisacariu, O. Kähler, M. M. Cheng, C. Y. Ren, J. Valentin, P. H. S. Torr, I. D. Reid, and D. W. Murray, "A Framework for the Volumetric Integration of Depth Images," 2014.

[21] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, J. Mcdonald, and J. J. Leonard, "Computer Science and Artificial Intelligence Laboratory Technical Report Kintinuous : Spatially Extended KinectFusion Kintinuous : Spatially Extended KinectFusion," 2012.

[22] M. Zeng, F. Zhao, J. Zheng, and X. Liu, "Octree-based fusion for realtime 3D reconstruction," *Graphical Models*, vol. 75, no. 3, pp. 126–136, 2013.

[23] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances In Neural Information Processing Systems*, 2012.

[24] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, P. Martinez-Gonzalez, and J. Garcia-Rodriguez, "A survey on deep learning techniques for image and video semantic segmentation," *Applied Soft Computing Journal*, vol. 70, pp. 41–65, 2018.

[25] R. Girshick, "Fast R-CNN," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[26] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[27] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 07-12-June, pp. 3431–3440, 2015.

[28] T.-y. Lin, C. L. Zitnick, and P. Doll, "Microsoft COCO : Common Objects in Context," pp. 1–15.

[29] P. Besl and N. McKay, "A Method for Registration of 3-D Shapes," pp. 239–256, 1992.

[30] P. Liu, M. Geppert, L. Heng, T. Sattler, A. Geiger, and M. Pollefeys, "Towards Robust Visual Odometry with a Multi-Camera System," *Iros*, 2018.

[31] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. Reid, and J. J. Leonard, "Past, Present, and Future of Simultaneous Localization And Mapping: Towards the Robust-Perception Age," *IEEE Trans. Robotics*, vol. 32, no. 6, pp. 1309–1332, 2016.

[32] T. Taketomi, H. Uchiyama, and S. Ikeda, "Visual SLAM algorithms: a survey from 2010 to 2016," *IPSJ Transactions on Computer Vision and Applications*, vol. 9, no. 1, p. 16, 2017.

[33] A. J. Davison, I. D. Reid, N. D. Molton, and O. Stasse, "MonoSLAM: Real-time single camera SLAM," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2007.

[34] G. Klein and D. Murray, "Parallel tracking and mapping for small AR workspaces," in *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality, ISMAR*, 2007.

[35] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "DTAM: Dense tracking and mapping in real-time," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2320–2327, 2011.

[36] J. Engel, T. Schöps, and D. Cremers, "LSD-SLAM: Large-Scale Direct monocular SLAM," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2014.

[37] J. Engel, "Direct Sparse Odometry," Tech. Rep.

[38] X. Gao, R. Wang, N. Demmel, D. Cremers, and C. V. Aug, "LDSO : Direct Sparse Odometry with Loop Closure."

[39] S. L. Bowman, N. Atanasov, K. Daniilidis, and G. J. Pappas, "Probabilistic data association for Semantic SLAM," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2017.

[40] D. Don, A. Gawel, C. Del Don, R. Siegwart, J. Nieto, and C. Cadena, "X-View: Graph-Based Semantic Multi-View Localization," Tech. Rep., 2018.

[41] B. Curless and M. Levoy, "A volumetric method for building complex models from range images," *Proceedings of the 23rd annual conference on ...*, pp. 303–312, 1996.

[42] 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," *International Journal of Robotics Research*, vol. 34, no. 4-5, pp. 598–626, 2015.

[43] S. Lovegrove, "Pangolin," https://github.com/stevenlovegrove/Pangolin, 2018.

[44] A. Handa, T. Whelan, J. McDonald, and A. J. Davison, "A benchmark for RGB-D visual odometry, 3D reconstruction and SLAM," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 1524–1531, 2014.

[45] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, "A benchmark for the evaluation of rgb-d slam systems," in *Proc. of the International Conference on Intelligent Robot Systems (IROS)*, Oct. 2012.

[46] A. Handa, R. A. Newcombe, A. Angeli, and A. J. Davison, "Real-time camera tracking: When is high frame-rate best?" in *Proc. of the European Conference on Computer Vision(ECCV)*, Oct. 2012.

[47] A. Criminisi, A. Fitzgibbon, and J. Shotton, "Rgb-d dataset 7-scenes," https://www.microsoft.com/en-us/research/project/rgb-d-dataset-7-scenes/, 2013.

[48] B. Glocker, S. Izadi, J. Shotton, and A. Criminisi, "Real-time RGB-D camera relocalization," *2013 IEEE International Symposium on Mixed and Augmented Reality, ISMAR 2013*, pp. 173–179, 2013.

[49] F. Chollet *et al.*, "Keras," https://keras.io, 2015.

[50] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org.

[51] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with CUDA," *ACM SIGGRAPH 2008 classes on - SIGGRAPH '08*, no. April, p. 1, 2008.

[52] I. A. Barsan, "Simultaneous Localization and Mapping in Dynamic Scenes," vol. 44, no. 23, 2017.

[53] A. Geiger, M. Roser, and R. Urtasun, "Efficient large-scale stereo matching," *Asian conference on computer vision*, pp. 25–38, 2010.

[54] N. Mayer, E. Ilg, P. Häusser, P. Fischer, D. Cremers, A. Dosovitskiy, and T. Brox, "A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation," 2015.

[55] K. Tateno, F. Tombari, I. Laina, and N. Navab, "CNN-SLAM : Real-time dense monocular SLAM with learned depth prediction."

[56] A. Cohen, C. Zach, S. N. Sinha, and M. Pollefeys, "Discovering and Exploiting 3D Symmetries in Structure from Motion."