# Incorporating Semantic Structure in SLAM

Akash Sharma

CMU-RI-TR-YY-NN

May 2021


The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA

**Thesis Committee:**
Prof. Michael Kaess, *chair*
Prof. Katerina Fragkiadaki
Prof. Sebastian Scherer
Chaoyang Wang

*Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.*

*To my favorite robot.*

# Abstract

For robots to understand the environment they interact with, a combination of geometric information and semantic information is crucial. In this thesis, we propose a fast, and scalable Simultaneous Localization and Mapping (SLAM) system that represents indoor scenes as a graph of semantic objects. Leveraging the observation that artificial environments are structured and occupied by recognizable objects, we show that a combination of compositional rendering and sparse volumetric object graph as the map results in a SLAM system suitable for drift-free large-scale indoor reconstruction. While object based SLAM has been proposed in the past, we improve on both object reconstruction quality, trajectory accuracy, and online performance. We also propose a semantically assisted data association method that results in unambiguous and persistent object landmarks. We deliver an online implementation that can run at about 4-5Hz on a single commodity graphics card, and provide a comprehensive evaluation against state-of-the-art baselines.

# Acknowledgments

First, I am grateful to Prof. Michael Kaess, my advisor for his patience, guidance and faith in me over the past two years through thick and thin! Without his support and ideas through our discussions, notwithstanding funding, this work would not have been possible. I would like to thank Prof. Katerina Fragkiadaki, for her guidance through the third semester about deep learning, and impressing the need to stay up-to-date with the research literature, and providing avenues to hone my learning. I am thankful to Prof. Sebastian Scherer for serving on my committee on short notice, and for asking insightful and incisive questions! I acknowledge Chaoyang Wang for serving on my committee and for interesting discussions on future work.

I would like to thank Wei Dong for being foremost a friend, a supportive co-author and for the insightful research discussions, that significantly influenced this work. I am thankful to my colleagues and former labmates in the Robot Perception lab (RPL) for their encouragement, honest research feedback and comments. I have learned a lot from them, and they made work at CMU an enjoyable experience.

I would like to thank some of my friends and colleagues in the Master of Science in Robotics cohort of '21, in particular Tarasha Khurana for the support and riveting discussions that ensued about research. Last, but not least, I'd like to thank my parents and family for their faith and support.

## Funding

This work was partially supported by Army Research Lab (ARL) award W911NF-17-2-0181.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

## 1.1 SLAM

Simultaneous Localization and Mapping (SLAM) is the chicken-and-egg problem of estimating the model of the environment (the map), and simultaneously optimizing the robot trajectory associated with the map, given measurements.

Typically, when the map is modeled as a collection of landmarks $\mathcal{L} = \{\ell_m\}_{m=1}^{M}$, and the robot trajectory is modeled as a sequence of poses $\mathcal{X} = \{\mathbf{x}_t\}_{t=1}^{T}$, given a set of measurements $\mathcal{Z} = \{\mathbf{z}_n\}_{n=1}^{N}$, the SLAM problem can be summarized as a maximum-a-posteriori estimation [8] as follows:

$$\hat{\mathcal{X}}, \hat{\mathcal{L}} = \underset{\mathcal{X},\mathcal{L}}{\operatorname{argmax}}\, p(\mathcal{X}, \mathcal{L}|\mathcal{Z}) \tag{1.1}$$

$$\propto \underset{\mathcal{X},\mathcal{L}}{\operatorname{argmax}}\, p(\mathcal{Z}|\mathcal{X}, \mathcal{L})p(\mathcal{X}, \mathcal{L}) \tag{1.2}$$

In reality, however, a SLAM framework for a robot needs to deal with raw sensor measurements. Therefore, typically the system is divided into two parts: *front-end* and *back-end*. The front-end is dedicated to data pre-processing, data association and feeds into the back-end dedicated to the optimization that results in the best estimate of robot state and landmarks.

## 1.2 Semantics in SLAM

For autonomous robots in the near future, to work in the real world advanced interpretation of the environment is necessary. For workloads ranging from semantic 3D reconstruction and path planning to active interaction with the environment (see Figure 1.1) robotic systems require not only geometric perception including robot localization and map reconstruction, but also semantic and compositional understanding of scenes.


Figure 1.1: Boston Dynamics' Spot Mini robot picking up clothes in a living room. These types of complex interactions require the robot to have a semantic understanding of the objects in its environment.

In recent years, geometry-based SLAM has achieved high levels of performance in *experimental setups* for localization tasks. Many variants of SLAM algorithms, from ORB-SLAM [23] to Direct Sparse Odometry (DSO) [11], can now run in real-time with high trajectory accuracy. However, they are in general limited by the *static-world* assumption and low-level scene representation as sparse 3D feature points, and thus cannot distill high-level information also known as semantic understanding in scenes and adjust to structured environmental changes.

On the other hand, with progress in deep learning, near real-time semantic perception is achievable powered by efficient Deep Neural Networks (DNNs). There has been explosive progress in object detection and instance segmentation over the

past 5 years [13, 31, 32] which is underutilized in SLAM. Researchers have started to switch to semantic SLAM taking advantage of off-the-shelf solutions; pioneering research includes SLAM++ [35], Fusion++ [21], and MaskFusion [33]. These initial attempts take into consideration semantic segmentation, but typically simply attach DNN frontends to existing SLAM frameworks in an ad hoc fashion. Implementation-wise, they require high-end machines to achieve near real-time performance, or are not available to the community.

## 1.3 Contributions

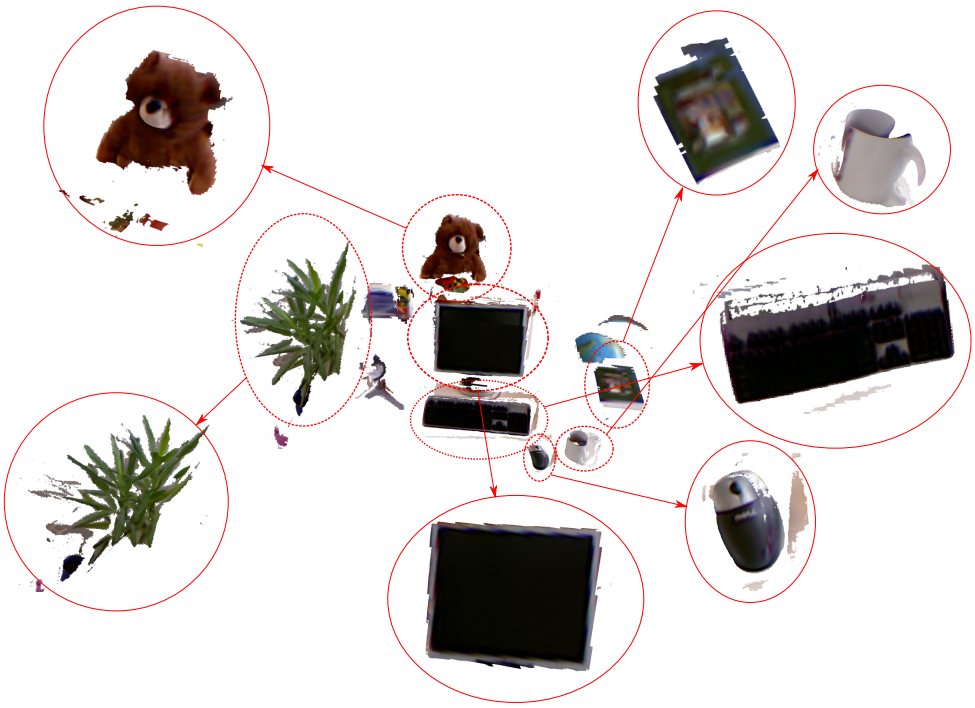


Figure 1.2: Reconstruction of *fr2_xyz* sequence from *tum rgbd* dataset. Our pipeline can reconstruct both camera trajectory and object models in the scene.

In this thesis, I present a Compositional and Scalable Object SLAM system that represents an environment as a pose graph of persistent objects. I describe a system, that incrementally improves each associated persistent object landmark through a RGBD Fusion approach, and optimizes the landmarks and the robot trajectory

simultaneously to accurately represent an environment. (see Figure 1.2).

In implementation, this work fully exploits the power of the modern GPU-based reconstruction pipeline [9] and object detection frameworks [17], to design an efficient architecture for data exchange without sacrificing the ease of system configuration and build.

This document presents the following core contributions:

1. A compositional volumetric rendering method that feeds the most up-to-date model render for RGBD camera tracking;
2. A hybrid object association method that combines geometric and semantic cues to enable drift-free tracking without an explicit relocalization module;
3. A scalable, modular, and easy-to-use open source system that runs nearly realtime.

## 1.4 Organization

In Chapter 2 we first review some requisite preliminaries and concepts in SLAM, and dense 3D reconstruction.
In Chapter 3 we introduce the core SLAM system that forms the major contribution of this thesis. This chapter presents some of the improvements made in the SLAM system with object landmarks, and delineates each component in the pipeline.
In Chapter 4, we present some qualitative and quantitative experiments and discussion by comparing against existing state-of-the-art geometric SLAM systems and a few dense semantic SLAM systems.
Finally, in Chapter 5, we conclude the thesis, with some limitations and future directions of research work to improve SLAM systems.

# Chapter 2

# Background

As eluded to in Chapter 1, SLAM systems that run on real robots can be crudely divided into *front-end* and *back-end*. This chapter is a primer on algorithms and methods used in the *front-end* and *back-end*.

## 2.1 The Front End

The robotic paradigm describes the relationship of an agent interacting with its environment through the *sense-plan-act* control loop. SLAM being in the sense paradigm, deals with processing the raw environment sensor readings to a structured representation that can be used in downstream tasks.

The front end deals with data preparation, modeling and data association such that it is amenable to the underlying optimization. More often than not, the choice of front-end is sensor dependent. For instance, many visual SLAM systems that utilize a monocular or stereo camera resort to *feature based* [20, 30] sparse representations, while RGBD sensor based systems resort to *direct methods* [24, 42] that utilize all the pixels in the input frame.

This distinction in the choice of front-end often influences the choice of map representation in a SLAM system. A popular choice with visual SLAM system is to use an *explicit* map representation as a set of 3D points [23] that is *triangulated* from multiple 2D image feature correspondences. On the other hand dense front-end systems rely on *surfel* representations or on volumetric representations such as occu-

pancy grids. More recently, after the seminal work [24], *implicit* map representations have gained popularity as a candidate representation for RGBD SLAM systems.

In this work, since the focus is on semantic high level object landmarks each represented with implicit representations such as Truncated Signed Distance Function (TSDF) grids, the next section provides a short introduction of this representation.

### 2.1.1 Implicit map representation through TSDFs

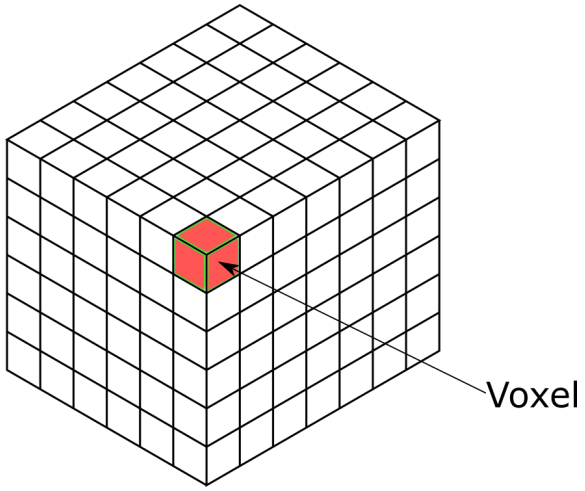



Figure 2.1: An example of metric space in 3D discretized on a computer as a voxel grid.

*Implicit* map representations are fundamentally different from *explicit* map representations. As opposed to point clouds or landmark collections, an implicit map refers to one where the environment is defined through a metric space $\mathbf{X}$ (see Figure 2.1) and structures in this space are described through an *implicit* function $f : \mathbf{X} \to \mathbb{R}$. One example of an implicit function is the log-odds occupancy, resulting directly in occupancy grids [39] and *octree* [14] representations. Another is the signed distance function (SDF) which is relevant for this thesis. Formally, it is defined as follows:

In a metric space $\mathbf{X}$ defined with distance $d$, if the surface defined by $\partial\Omega$ demarcates

this space into subsets $\mathbf{\Omega}$ and $\mathbf{\Omega}^c$, the signed distance function $f$ is defined by

$$f = \begin{cases} d(\mathbf{x}, \partial\mathbf{\Omega}), & \text{if } \mathbf{x} \in \mathbf{\Omega} \\ -d(\mathbf{x}, \partial\mathbf{\Omega}), & \text{if } \mathbf{x} \in \mathbf{\Omega}^c \end{cases} \tag{2.1}$$

In particular, for the 3D environment, we have that $\mathbf{X} = \mathbb{R}^3$ with $d = \|\cdot, \cdot\|_2$, and $\partial\Omega$ corresponds to the zero level set of the signed distance function and represents the surface of objects as a topological space. In simple words, the signed distance function returns the shortest distance of a query point in Euclidean space to its closest point on the surface boundary (see Figure 2.2).

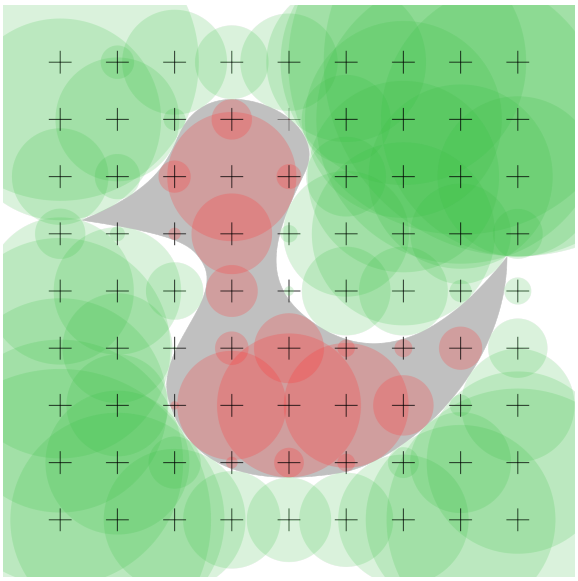


Figure 2.2: An illustration of Signed Distance Function for a 2D metric space discretized into pixels (query coordinates '+') (Courtesy: Wikipedia).

### TSDF integration

A TSDF representation requires a metric volume by design, which is implemented as a volumetric grid $V$, arbitrarily initialised in the world frame $\mathbf{T}_W = \mathbf{I} \in SE(3)$. Consider a single RGBD frame measurement $\langle \mathcal{I}, \mathcal{D} \rangle$ that consists of a color ($\mathcal{I}$) and depth ($\mathcal{D}$) image, at a relative camera pose $\mathbf{T}_C^W = \begin{bmatrix} \mathbf{R}_C^W & \mathbf{t}_C^W \\ 0 & 1 \end{bmatrix} \in SE(3)$ with camera

intrinsics $\mathbf{K}$. To integrate the measurement into the TSDF volume, we project all the query points $q \in \mathbb{R}^3$ of the volume $V$ into the current image as follows:

$$\dot{q}' = (\mathbf{T}_C^W)^{-1}\dot{q} \tag{2.2}$$

$$\dot{p} = \lfloor \mathbf{K}q' \rfloor \tag{2.3}$$

where $\dot{q} = \begin{bmatrix} q^\top & 1 \end{bmatrix}^\top$ and similarly $\dot{p}$ denote the homogeneous coordinates in $\mathbb{R}^2$ and $\mathbb{R}^3$ respectively. $p \in \mathbb{Z}^2$ is the associated 2D pixel coordinate in the image and $\lfloor \cdot \rfloor$ is the floor function and finds the greatest integer pixel coordinate less than the pixel value in the image. For every matched pixel coordinate, we backproject the associated measurement i.e., the depth value of the pixel $\mathcal{D}(p) \in \mathbb{R}$ into the volume to obtain a *noisy* 3D surface in the metric space (see Figure 2.3). This noisy surface is used as the proxy surface against which all query coordinates ($q \in \mathbb{Z}^3$) in the volumetric grid are evaluated against to obtain their corresponding SDF values as follows:

$$\text{SDF}(q) = \left\| \mathcal{D}(p) - \frac{1}{\lambda}\|\mathbf{t}_C^W - q\|_2 \right\|_2 \tag{2.4}$$

where $\|\mathbf{t}_C^W - q\|_2$ is the distance between the query point and the camera center along the camera optical axis, and $\lambda = K^{-1}p$ defines the ray direction for the pixel $p$. This effectively calculates the shortest distance of the query coordinate to the surface.

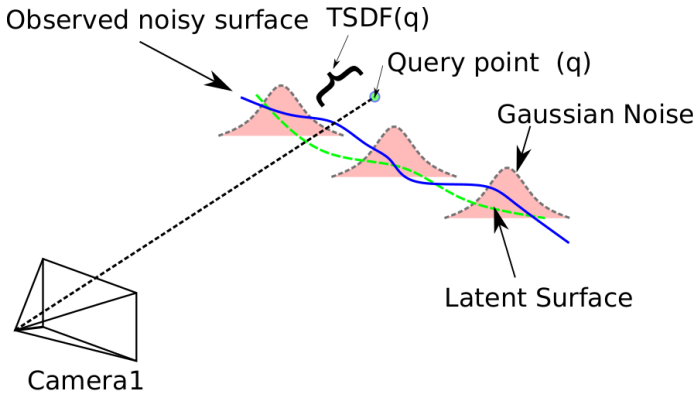



Figure 2.3: Back-projection of associated pixels into the 3D volumetric grid to obtain a noisy surface.

This SDF value is truncated such that only a band of SDF values ($+\mu$ to $-\mu$) around the measured surface are computed, to avoid computation of all the query coordinates in the volume. Similarly, colors from the image can be associated to the query coordinates by unprojection uniquely since a depth image is available.

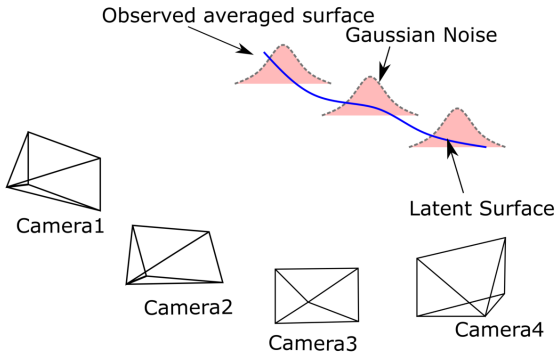



Figure 2.4: On weighted averaging of TSDF values obtained from multiple RGBD measurements, in the limit we converge to TSDF values of the underlying surface.

Now, when we have multiple RGBD frame measurements $\{\langle \mathcal{I}^{(i)}, \mathcal{D}^{(i)} \rangle\}_{i=1}^{N}$ with their respective poses $\{T^{W}_{C^{(i)}}\}_{i=1}^{N}$, we can compute TSDF measurements from each individual frame. The global fusion of all these TSDF measurements is then done through weighted average of the TSDF measurements for each query coordinate $q \in \mathbb{R}^3$ over the camera frame sequence as follows:

$$d = \phi(\text{SDF}(q)) \tag{2.5}$$

$$\text{TSDF}_k(q) = \frac{\text{TSDF}_{k-1}(q) + d}{W(q) + 1} \tag{2.6}$$

$$W(q) = W(q) + 1 \tag{2.7}$$

where $\phi$ truncates the SDF computed for the frame being integrated, and $W(q)$ corresponds to the maintained weight for every voxel coordinate in the volume grid. This integration process intuitively results in evaluation of the SDF values for the query coordinates with the expected surface (mean of the TSDFs) from all the incorporated RGBD measurements (see Figure 2.4).

In general, the integration process can also be applied to the color values, where the colors are averages elementwise, with the incoming stream of images.

#### TSDF Rendering

TSDF Rendering is the process of generating a virtual depth and color image given a camera pose $\mathbf{T}_C^W$ and TSDF volume grid $V$. The previous subsection assumed access to camera poses for global TSDF fusion, but in an actual SLAM setting camera poses are computed in the loop typically through iterative closest point (ICP) [34] registration. TSDF Rendering is the intermediate step that generates a virtual image that the incoming camera frame is registered against (also called *frame to model* registration [24]). (details in §3.2.3)

Rendering is accomplished by marching a ray per pixel of the virtual frame into the global TSDF volume which encodes surfaces as the zero level set (zero crossing). For a given pixel $p \in \mathbb{Z}^2$, once again the camera ray can be computed as

$$\mathbf{r} = \mathbf{T}_C^W \mathbf{K}^{-1} \dot{p}.$$

We then march along the ray from the minimum depth supported by the camera (usually assumed to be 0.1m) to a max depth value defined by the supported camera range (around 5m to 8m for Microsoft Kinect), or until we find a zero crossing. Typically this process is sped up by skipping marching along the ray, and using a step size $< \mu$, the truncation threshold. Once a zero crossing has been found, the depth value can be further refined by trilinear interpolation. (details in [24])

### 2.1.2 TSDF Limitations

Based on the discussion of TSDF for map representation in SLAM, the following limitations might be apparent to the keen reader.

1. **TSDF representation is memory intensive:** A naive TSDF representation, utilizes a 3D volumetric grid to represent the scene – typically a voxel grid of resolution $512^3$ covering a bounded volume of about $4m^3$. This makes the entire system very memory intensive. Later works such as Voxel Hashing [26] and OctoMap [14] have tried to ameliorate this issue. In our implementation, we utilize this spatial hashing based TSDF implementation based on [9] to achieve memory efficiency.
2. **TSDFs are computationally intensive:** During integration of an incoming

frame, all pixels that are projectively associated to the 3D query coordinates are updated requiring about $640 \times 480 = 307200$ operations for a VGA resolution image. Similarly, during TSDF rendering, each pixel to be rendered requires at most $\frac{\|d_{\max} - d_{\min}\|_2}{\mu}$ evaluations, where $\|d_{\max} - d_{\min}\|_2$ is the maximum ray length, and $\mu$ is the TSDF truncation width. Therefore, ray-casting is typically the most computationally intensive step in any dense RGBD SLAM system. Observing that each pixel is updated independent of others, a GPU can (and is) used to parallelize all TSDF operations, to obtain realtime operation.

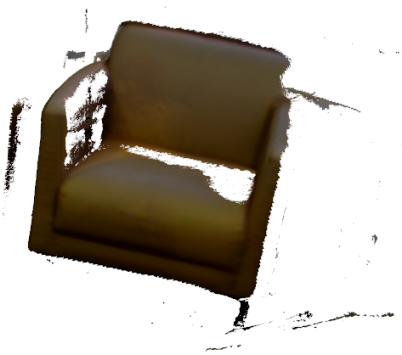


Figure 2.5: An example of dense high level landmark in our SLAM system.

3. **Cannot easily smooth noisy pose estimates:** It is not easy to correct noisy pose estimates with implicit models, and therefore most early systems employed only a filtering based integration [24]. This is because to correct for a noisy pose estimate in retrospect requires the expensive operation of subtracting the weighted TSDF values in the global volume from the appropriate coordinates and re-integrating the volume with corrected TSDF values [5]. [41] used submaps to subvert this problem, however with large TSDF volumes as submaps there are still locally noisy measurements that are not corrected for. In our method, since we use a TSDF representation only for landmark objects, we reap the benefits of explicit representation, where the whole landmark pose can be optimized, but each landmark provides a dense and appealing representation. (see Figure 2.5)

## 2.2 The Back End

The back end is responsible for generating a reasonable explanation in the state variables given the abstract sensor measurements and data associations from the front end. As opposed to filtering based methods such as Kalman Filter, more recently smoothing formulations based on matrix factorization have provided exact solutions in the linear case, and approximate but good solutions in non-linear case by adding the entire trajectory into the optimization problem while simplifying the solution [15]. In this section, I review the batch smoothing formulation that is quite common in graph based SLAM back-ends today [8, 12].

### 2.2.1 SLAM and Non-Linear Least Squares

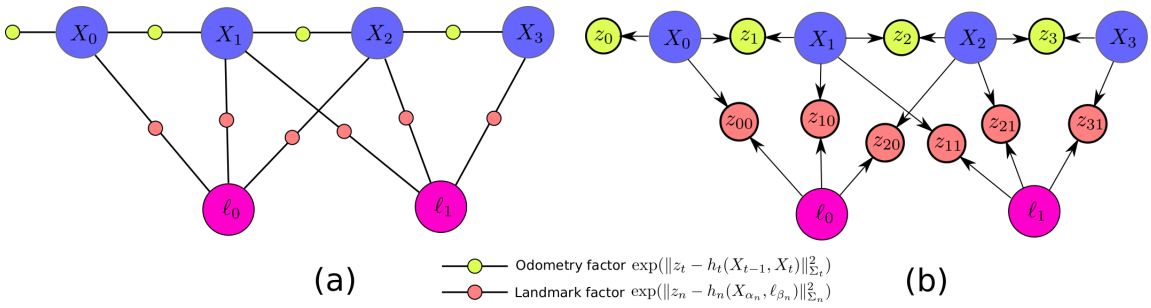



Figure 2.6: (a) A factor graph typical for SLAM and (b) the explicit Bayes net for the factor graph in (a). Note the Markovian conditional independences between measurements, given the state variables.

As introduced in Chapter 1, by treating the robot states and landmarks as random variables given measurements, the SLAM problem can be treated as the maximum a posteriori (MAP) estimation of robot and landmark states. Now, consider a collection of robot state variables $\mathcal{X} \triangleq \{\mathbf{x}_t\}_{t=1}^{T}$, and a collection of landmarks $\mathcal{L} \triangleq \{\ell_m\}_{m=1}^{M}$, and set of measurements $\mathcal{Z} \triangleq \{\{\mathbf{z}_n\}_{n=1}^{N}, \{\mathbf{z}_t\}_{t=1}^{T}\}$ including both landmark and odometry measurements respectively, then the MAP estimation is given as

$$\mathcal{X}^*, \mathcal{L}^* = \underset{\mathcal{X},\mathcal{L}}{\operatorname{argmax}}\, p(\mathcal{X}, \mathcal{L}|\mathcal{Z}) = \underset{\mathcal{X},\mathcal{L}}{\operatorname{argmax}}\, p(\mathcal{Z}|\mathcal{X}, \mathcal{L})p(\mathcal{X}, \mathcal{L}) \tag{2.8}$$

While MAP estimation is known to be NP-Hard in general, by assuming reasonable

conditional independences between the associated variables (see Figure 2.6), i.e., by making the markovian assumption on the state variables, and observing that measurements are local, we can write the joint probability in 2.8 as follows:

$$p(\mathcal{X}, \mathcal{L}, \mathcal{Z}) = p(\mathbf{x}_0) \prod_{n=1}^{N} p(\mathbf{z}_n | \mathbf{x}_{\alpha_n}, \ell_{\beta_n}) \prod_{t=1}^{T} p(\mathbf{z}_t | \mathbf{x}_{t-1}, \mathbf{x}_t)$$

$$\log p(\mathcal{X}, \mathcal{L}, \mathcal{Z}) = \log p(\mathbf{x}_0) + \sum_{n=1}^{N} \log p(\mathbf{z}_n | \mathbf{x}_{\alpha_n}, \ell_{\beta_n}) + \sum_{t=1}^{T} \log p(\mathbf{z}_t | \mathbf{x}_{t-1}, \mathbf{x}_t) \tag{2.9}$$

Note, that the above problem assumes known *data association* between measurements $\mathbf{z}_n$ of landmark $\ell_{\beta_n}$ at a robot pose $\mathbf{x}_{\alpha_n}$as $\mathcal{D} := \{(\alpha_n, \beta_n)\}_{n=1}^{N}$ [2]. Now as is typical in SLAM, we assume Gaussian measurement noise in each of the measurements i.e.,

$$\mathbf{z}_0 = \mathbf{x}_0 + \boldsymbol{\nu}_0 \tag{2.10}$$

$$\mathbf{z}_n = h_n(\mathbf{x}_{\alpha_n}, \ell_{\beta_n}) + \boldsymbol{\nu}_n \tag{2.11}$$

$$\mathbf{z}_t = h_t(\mathbf{x}_{t-1}, \mathbf{x}_t) \ + \boldsymbol{\nu}_t \tag{2.12}$$

Where $\boldsymbol{\nu}_n$ and $\boldsymbol{\nu}_t$ are the intrinsic noise in the respective measurements, each distributed as zero mean noise with respective covariances $\Sigma_n$ and $\Sigma_t$. Also note that we added a phantom measurement $\mathbf{z}_0$ to constrain the first pose with an empirically chosen small covariance $\Sigma_0$. On writing down the probability distributions for each of the Gaussian noise terms we have the following:

$$p(\mathbf{x}_0) \propto \exp\{\|\mathbf{z}_0 - \mathbf{x}_0\|^2_{\Sigma_0}\} \tag{2.13}$$

$$p(\mathbf{z}_n | \mathbf{x}_{\alpha_n}, \ell_{\beta_n}) \propto \exp\{\|\mathbf{z}_n - h_n(\mathbf{x}_{\alpha_n}, \ell_{\beta_n})\|^2_{\Sigma_n}\} \tag{2.14}$$

$$p(\mathbf{z}_t | \mathbf{x}_{t-1}, \ell_t) \propto \exp\{\|\mathbf{z}_t - h_n(\mathbf{x}_{t-1}, \ell_t)\|^2_{\Sigma_t}\} \tag{2.15}$$

Observe that

$$\mathcal{X}^*, \mathcal{L}^* = \underset{\mathcal{X}, \mathcal{L}}{\operatorname{argmax}}\, p(\mathcal{X}, \mathcal{L}, \mathcal{Z}) = \underset{\mathcal{X}, \mathcal{L}}{\operatorname{argmin}} - \log p(\mathcal{X}, \mathcal{L}, \mathcal{Z}) \tag{2.16}$$

Now writing the joint probability using the factorized measurement distributions we

have

$$\mathcal{X}^*, \mathcal{L}^* = \underset{\mathcal{X},\mathcal{L}}{\operatorname{argmin}} \left\{ \|\mathbf{z}_0 - \mathbf{x}_0\|^2_{\Sigma_0} + \sum_{n=1}^{N} \|\mathbf{z}_n - h_n(\mathbf{x}_{\alpha_n}, \ell_{\beta_n})\|^2_{\Sigma_t} + \sum_{t=1}^{T} \|\mathbf{z}_t - h_t(\mathbf{x}_{t-1}, \mathbf{x}_t)\|^2_{\Sigma_n} \right\}$$

If we generalize the different types of measurements in the system as $h_i(\mathbf{X}_i)$, and subsume the landmarks and state variables into a single state vector $\mathcal{X}$ then we may write the above as:

$$\mathcal{X}^* = \underset{\mathcal{X}}{\operatorname{argmin}} \left\{ \sum_{i=1}^{N} \|\mathbf{z}_i - h_i(\mathbf{X}_i)\|^2_{\Sigma_i} \right\} \tag{2.17}$$

The above minimization is a non-linear least squares optimization which is typically solved by iteratively linearizing the measurement models and solving a series of linear approximations to the problem to approach a local minima solution for landmark and robot latent state variables.

### 2.2.2 Iterative Linearization and Gauss Newton

Consider a single non-linear measurement $h_i(\cdot)$, then by Taylor's expansion, we can linearize the measurement at a linearization point $\mathbf{X}_i^{(0)}$ as follows (see Figure 2.7):

$$h_i(\mathbf{X}_i) \approx h_i(\mathbf{X}_i^{(0)}) + \left.\frac{\partial h_i}{\partial \mathbf{X}_i}\right|_{\mathbf{X}_i^{(0)}} (\mathbf{X}_i - \mathbf{X}_i^{(0)}) \tag{2.18}$$

If we define the jacobian as $\mathbf{J}_i \triangleq \left.\frac{\partial h_i}{\partial \mathbf{X}_i}\right|_{\mathbf{X}_i^{(0)}}$, and $\Delta_i \triangleq (\mathbf{X}_i - \mathbf{X}_i^{(0)})$ and substitute in the general non-linear least squares objective 2.17:

$$\Delta^* = \underset{\Delta}{\operatorname{argmin}} \sum_i \|\{\mathbf{z}_i - h_i(X_i^{(0)})\} - \mathbf{J}_i \Delta_i\|^2_{\Sigma_i} \tag{2.19}$$

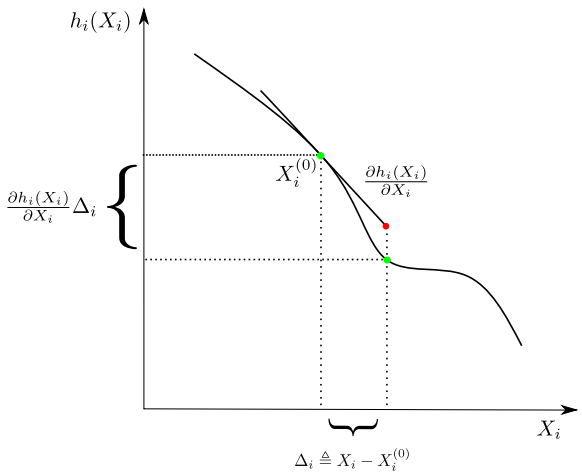



Figure 2.7: Illustration of linearization using Taylor's expansion.

where $\mathbf{q}_i \triangleq \mathbf{z}_i - h_i(\mathbf{X}_i^{(0)})$ is the prediction error at the linearization point, and $\Delta^*$ is the solution to the locally linearized problem. Now we may write the least squares objective as:

$$\Delta^* = \underset{\Delta}{\operatorname{argmin}} \sum_i \|\mathbf{q}_i - \mathbf{J}_i\Delta_i\|^2_{\Sigma_i} = \underset{\Delta}{\operatorname{argmin}} \sum_i (\mathbf{q}_i - \mathbf{J}_i\Delta_i)^\top \Sigma_i^{-1}(\mathbf{q}_i - \mathbf{J}_i\Delta_i) \tag{2.20}$$

$$= \underset{\Delta}{\operatorname{argmin}} \sum_i \left\{\Sigma_i^{-1/2}(\mathbf{q}_i - \mathbf{J}_i\Delta_i)\right\}^\top \left\{\Sigma_i^{-1/2}(\mathbf{q}_i - \mathbf{J}_i\Delta_i)\right\} \tag{2.21}$$

$$= \underset{\Delta}{\operatorname{argmin}} \sum_i \|\mathbf{b}_i - \mathbf{A}_i\Delta_i\|_2^2 = \underset{\Delta}{\operatorname{argmin}} \|\mathbf{b} - \mathbf{A}\Delta\|_2^2 \tag{2.22}$$

By writing $\mathbf{b}_i = \Sigma_i^{-1/2}\mathbf{q}_i$ and $\mathbf{A}_i = \Sigma_i^{-1/2}\mathbf{J}_i$ and then collecting them in the vector $\mathbf{b}$ and matrix $\mathbf{A}$ respectively, we obtain the standard least squares problem, that can then be solved using *normal equations*.

The above solution gives us the local update $\Delta^*$ for a linearization point $\mathbf{X}^{(0)}$. To solve a non-linear problem, we solve for local updates iteratively as in Algorithm 1.

A more detailed treatment of sophisticated non-linear optimization algorithms in the context of SLAM is provided in [8].

### 2.2.3 Discussion and Limitations

We noted earlier that MAP inference on the factor graph for the SLAM setting is tractable due to the inherent local structure of the factor graph. This structure

**Algorithm 1** Iterative Non-linear Optimization

**procedure** GAUSS NEWTON($g(\mathbf{X}), \mathbf{X}^{(0)}$)
  $d$ = threshold
  **for** t = 0 to N **do**
    $\mathbf{A}, \mathbf{b} \leftarrow$ Linearize($g(\mathbf{X})$) at $\mathbf{X}^{(t)}$
    $\Delta^* \leftarrow (\mathbf{A}^\top \mathbf{A})^{-1} \mathbf{A}^\top \mathbf{b}$ ▷ using QR or Cholesky factorization
    temp $\leftarrow \mathbf{X}^{(t)} + \Delta^*$
    **if** $g(\text{temp}) < g(\mathbf{X}^{(t)})$ **then**
      $\mathbf{X}^{(t+1)} \leftarrow$ temp
      **if** $g(\mathbf{X}^{(t+1)}) - g(\mathbf{X}^{(t)}) < d$ **then**
        **break**
      **end if**
    **end if**
  **end for**
  **return** $\mathbf{X}^{(t)}$
**end procedure**

appears in the sparsity of the measurement Jacobian $\mathbf{A} = \Sigma^{-1/2}\mathbf{J}$ with number of rows equal to the number of measurements in the graph (number of factors) and the number of columns equal to the number of state variables to be solved. The non-linear least squares optimization is solved efficiently using matrix factorization methods such as QR factorization and Cholesky factorization (equivalent to variable elimination on the factor graph [7]). Further improvement in performance can be gained by reordering the measurements using methods such as COLAMD [6] which consequently reduces fill in in the matrix during computation.

Nevertheless, for an online system with this formulation matrix factorization needs to be performed with a new graph when new measurements are added, resulting in wasted computation to resolve all the entries in the matrix. Later work has made progress on incremental solvers for SLAM, that are able to update the solution iteratively for non-linear systems and provide linearization point management, as well as careful updation of older variables based on the current estimate. As this thesis does not directly use incremental solvers, the reader is directed to Kaess et al. [16]

For implicit mapping representations, however, it is unclear how they are to be represented as landmark variables, and how updates to the landmark state variables are to be propagated in the map. In this thesis, we show that representing the map

as a collection of high level objects, each represented with implicit representation and corresponding to a landmark variable in the optimization is a good strategy, and benefits the SLAM in multiple facets. Each object is represented with a spatially hashed TSDF volumetric grid [29] [26] [9] that is not updated in the back end optimization process. Instead, since the object is rigid, optimizing the pose of the base frame attached to it suffices.

# Chapter 3

# SLAM with Object Landmarks

In this chapter I present a SLAM formulation with object landmarks, which is scalable to medium sized indoor scenes and is robust to data association errors through the use of semantic information. Further I also present a compositional rendering method to render an accurate model frame at any given time during operation to propagate updated information to the front end in turn providing better initialization for the back end. This work is similar in spirit to a line of work started by [35]. In this thesis however, the focus is on accurate trajectory estimation as well as object reconstruction. In essence, our work can be regarded as a bridge between feature based SLAM methods and dense SLAM methods.

## 3.1 Related Literature

In this section we review relevant literature in two aspects: classical geometry-based SLAM, and the application of deep semantic object detection in SLAM.

### 3.1.1 Geometry-based SLAM

#### Problem formulation and pose optimization

Modern geometry-based SLAM systems can be generally classified into *feature-based* and *direct* methods. Feature-based SLAM systems [18, 23] usually maintain a collection of sparse 3D *point landmarks* corresponding to hand-crafted feature

*keypoints* detected in 2D images. In order to correct accumulated *pose* error, *i.e.,* drift, these methods resort to bundle adjustment [40] that jointly minimizes reprojection error between *landmarks* and 2D *keypoints* via *pose* optimization. While being accurate in estimating trajectory, these approaches only come with sparse 3D maps that are less interpretable for visualization and recognition. Direct SLAM [10, 11] on the other hand, relies on pixel-wise projective data association between frames for odometry. Given relative poses between certain keyframes, pose graphs [8] are formulated and optimized to obtain globally consistent poses without landmark constraints.

Our approach can be regarded as a bridge between the two approaches. We replace point landmarks with objects in feature-based SLAM. As a result, since pose constraints attached to objects can naturally replace reprojection error, we may directly convert such a landmark-pose constraint optimization to pose graph optimization (PGO).

#### Map representation

For *dense* scene reconstruction, as introduced in Chapter 2 the volumetric Truncated Signed Distance Function (TSDF) [4] representation has been adopted and improved in several *direct SLAM* frameworks. KinectFusion [24] introduced a plain $512^3$ grid for small scenes and objects. VoxelHashing [26] designed spatial hashing to scale this data structure to larger scenes. Similar implementations are available in CPU/GPU in the modern Open3D framework [9, 44], and we adopt this representation due to its ease of use.

### 3.1.2 Object Instance Segmentation and Object based SLAM

#### Object instance segmentation

In recent years, *Region proposal* based Convolutional Neural Networks (R-CNN) [32] have established themselves as de-facto standards for object *instance segmentation* from images. Amongst the literature, *Mask-RCNN* [13] and *PointRend* [17] are the best off-the-shelf solutions. In this work, we use *PointRend* [17], which shows

significant improvement over [13] by reformulating the mask generation as a rendering problem. In essence, this formulation is consistent with our tracking via compositional rendering module.

#### Object based SLAM

Applying aforementioned DNNs on 2D images, several works for RGBD and monocular SLAM have attempted to incorporate object instance detection. CubeSLAM [43] and QuadricSLAM [25] fit cuboids and quadrics, respectively, to detected objects to generate parameterized object landmarks. While improving the localization accuracy compared to baselines, these methods fail to densely map objects. *MaskFusion* [33] adds labels to oriented point clouds and supports dense object visualization, but does not maintain persistent objects globally in a graph. *Fusion++* [21], on the other hand, supports persistent dense reconstruction from fixed size $64^3$ voxel grids, yet is sensitive to voxel size tuning and may fail to adapt to objects at varying scales. Our system utilizes scalable voxel grids that do not require much tuning to adjust to object scales. With a seamless CPU to GPU memory transfer implementation, larger environments can also be handled on-the-go.

## 3.2 Compositional and Scalable Object SLAM

### 3.2.1 System Overview

Our pipeline can be divided into typical SLAM components and a deep perception module, connected by an object-based semantic map. Figure 3.1 provides an overview.

Our pipeline consists of 5 modules each running in a separate thread: semantic segmentation, frame-to-model odometry, object data association and map update, PGO, and compositional rendering. Incoming RGBD frames are initially processed through *semantic segmentation* (§3.2.4) to obtain instance masks, labels, and semantic descriptors, from DNNs for keyframes. Then, odometry between the incoming live frame and the *compositional render* from the map (§3.2.6) is estimated via *frame-to-model odometry* (§3.2.3) to obtain relative poses. Maintained objects visible in the frame are rendered given the estimated camera pose, and objects are associated

with 2D instance detections to either integrate or initialize new objects in the global map (§3.2.4). Separately, a global factor-graph is updated to optimize the camera trajectory and object poses (§3.2.5). The optimized object poses are rendered to generate a compositional model of the scene for subsequent tracking (§3.2.6).

Before we discuss these modules in detail from §3.2.3 to §3.2.6, we introduce core concepts and notations in §3.2.2.

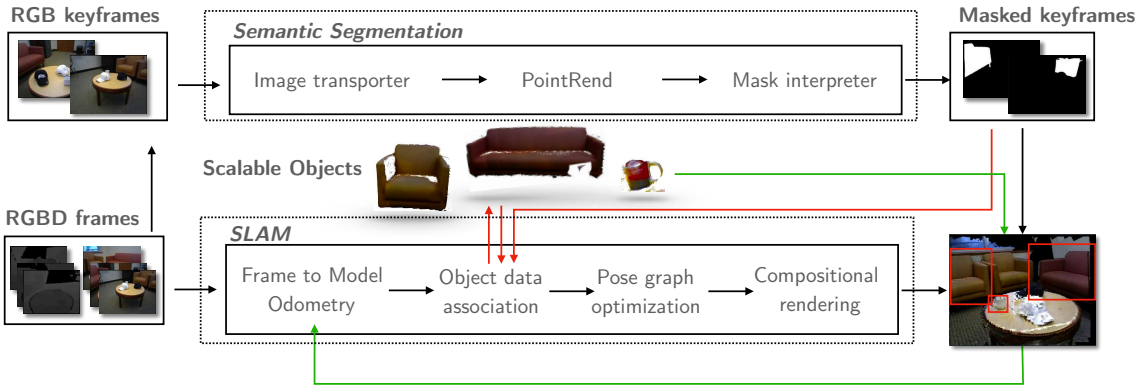



Figure 3.1: System overview: Top shows the deep object segmentation pipeline that runs asynchronously, Masked keyframes from the segmentation pipeline are used in Data association and Map update (shown with red lines). Bottom shows the major stages of the reconstruction system, specifically object models are used in tracking via compositional raycasting (shown with green lines).

### 3.2.2 Core concepts and notations

A background volume $V_B$ is a spatially-hashed voxel grid on GPU, where small $16^3$ subvolumes are allocated around observed 3D points. It is created and updated as a *temporary* instance for stable tracking. An object volume $V_{O_i}$ is akin to the background volume, but persistently maintains the object label, ID, and corresponding object descriptors.

A 3D volume $V$'s properties, including surface vertex positions, normals, and colors can be mapped to 2D images given a camera pose $\mathbf{T} \in SE(3)$ and camera intrinsics defined as $K$ with ray-casting. We denote such *rendered images* by $\langle \mathcal{N}, \mathcal{V}, \mathcal{C} \rangle$ for normal, vertex, and color maps respectively. They can be associated with input

RGBD images $\langle \mathcal{I}, \mathcal{D} \rangle$ that consist of color ($\mathcal{I}$) and depth ($\mathcal{D}$) images via projective closest points.

We use subscripts and superscripts to indicate multiple coordinate frames used in our pipeline, including $C_i$ for $i$th camera, $O_j$ for $j$th object, and $W$ for background or world coordinate frame. For instance $\langle \mathcal{N}_{C_s}, \mathcal{V}_{C_s}, \mathcal{C}_{C_s} \rangle$ represents 2D maps rendered from the volumes in the $C_s$ camera coordinate frame. $\mathbf{T}_{O_j}^{W} \in SE(3)$ encodes a rigid transformation from object $j$ to world. Finally, we denote the respective measurements between nodes with variable $\mathbf{Z}$.

### 3.2.3 Hybrid Frame to Model Odometry

In RGBD camera tracking, we seek to estimate the relative camera pose $\mathbf{T}_{C_s}^{C_t}$ given an incoming RGBD target frame $\langle \mathcal{I}_{C_t}, \mathcal{D}_{C_t} \rangle$ and a source model $\langle \mathcal{N}_{C_s}, \mathcal{V}_{C_s}, \mathcal{C}_{C_s} \rangle$ of the scene rendered by placing a virtual camera at the previous camera frame $C_s$.

We accomplish this by minimizing the joint weighted dense geometric error residual $r_D$ and the photometric error residual $r_I$. The general energy function is formulated as in [27] by accumulating residual at every point $p \in \mathbb{Z}^2$ with a valid data association:

$$E(\mathbf{T}_{C_s}^{C_t}) = \sum_{p} (1-\sigma) r_I^2(\mathbf{T}_{C_s}^{C_t}, p) + \sigma r_D^2(\mathbf{T}_{C_s}^{C_t}, p) \tag{3.1}$$

Here, we adapt the geometric ICP residual as the point-to-plane distance between the incoming depth map $\mathcal{D}$ and the rendered vertex and normal map $(\mathcal{V}_{C_s}, \mathcal{N}_{C_s})$ as follows, using the formulation in [24]:

$$r_D(T_{C_s}^{C_t}, p) = \left( T_{C_s}^{C_t} \mathcal{V}_{C_s}(\hat{p}) - \mathcal{V}_{C_t}(p) \right) \cdot \mathcal{N}_{C_t}(p) \tag{3.2}$$

where $\mathcal{V}_{C_t}$ is the vertex map from unprojecting the input depth image $\mathcal{D}_{C_t}$. Additionally, we use a photometric error residual to improve tracking robustness, which is defined as:

$$r_I(T_{C_s}^{C_t}, p) = \mathcal{C}_{C_s}(\hat{p}) - \mathcal{I}_{C_t}(p) \tag{3.3}$$

In equations (3.2) and (3.3), $\hat{p}$ is the correspondence of $p$ in the source frame, and is

computed via *warping*:

$$\hat{p} = K\mathbf{T}_{C_s}^{C_t\,-1}\mathcal{D}_{C_t}(p)K^{-1}[p^\top, 1]^\top \tag{3.4}$$

It must be noted that the points $p$ are a subset of pixels with valid object-level data associations detailed in §3.2.6.

The energy function in equation 3.1 is minimized using the Gauss-Newton algorithm. We implement the minimization in a coarse to fine scheme using an image pyramid, on the GPU in parallel since each pixel acts independently in the energy function using *reduction* with appropriate thread conflict handling as described in [9].

### 3.2.4 Object Instance Segmentation and Association

**2D instance segmentation:** Object detection and instance masks are generated every $n^{th}$ frame (we choose $n = 10$) in a separate thread from the *PointRend* backend. *PointRend* uses a Resnet-50-FPN backbone network to generate a convolutional feature map. In particular, after an empirical evaluation, we found that *PointRend* provided better masks over *Mask-RCNN*.

The semantic segmentation module maps incoming *RGB* frame $\mathcal{I}$ into a set of object labels $[l_1, \dots l_k]$, a set of binary object masks $M_n^i$ defined over $l \in \mathcal{L} \triangleq \{0, \dots, L_{max} - 1\}$ object classes ($L_{max} = 80$ in the MS-COCO dataset), bounding boxes $b \in \mathbb{N}^4$, and a probability distribution $p(l_i \mid \mathcal{I})$. We also extract the object feature map for the accepted object proposals, from the penultimate fully connected layer of the R-CNN from the object classifier head. We observe that these feature maps provide us with robust data association in ambiguous situations. To obtain instance segmentation for frames not sent to the DNN, we warp the binary mask images from the most recent frame with a segmentation and fill the holes in the warped masks using the *flood fill* algorithm.

Once the current camera pose and the semantic segmentation information are available, instance detections are associated with existing objects. Unmatched instance detections are used to initialize new object volumes.

**3D instance generation:** When an unmatched object is to be instantiated, the masked depth frame at $C_i$ is unprojected and transformed into the world frame to

obtain the object point cloud:

$$X_W = \mathbf{T}_{C_i}^{W} K^{-1} D_{C_i}(p)[p^\top, 1]^\top. \tag{3.5}$$

To obtain relatively high fidelity reconstruction, we adaptively calculate a conservative voxel length of

$$l = \gamma \| \max(X_W) - \min(X_W) \|_\infty, \tag{3.6}$$

where $\min, \max$ operators are applied to all dimensions of $X \in \mathbb{R}^3$ simultaneously. We empirically use $\gamma = 1/64\sqrt{2}$, but due to the scalability of the volume our model is less sensitive to $\gamma$. Finally, the object pose is simply chained by

$$\mathbf{T}_O^W = \mathbf{T}_{C_i}^{W} \left( \mathbf{T}_{C_i}^{O} \right)^{-1}, \tag{3.7}$$

where $\mathbf{T}_{C_i}^{O} = [I \mid t_{C_i}^{O}]$ with $t_{C_i}^{O} = \min(X_W) - t_{C_i}^{W}$. Each new object is also initialized with the object feature map from its corresponding instance mask.

**2D–3D semantic data association:** To associate existing object volumes to 2D instances, visible objects are rendered (in §3.2.6) in the current frame. The rendered color map $\mathcal{C}$ is thresholded to obtain a virtual binary mask. An intersection over union (IoU) between the virtual binary mask $\hat{\mathcal{M}}$ and the instance masks $\mathcal{M}_i$ in the current frame is used as a scoring metric as defined in [21].

As opposed to computing the $\text{argmax}_i \text{IoU}(\mathcal{M}_i, \hat{\mathcal{M}})$, we associate objects as given below:

$$i = \underset{i \in \mathcal{S}}{\text{argmin}} (\| f_i - \hat{f} \|_1), \tag{3.8}$$

where $\hat{f}$ and $f_i$ denote feature map of the object render (identical to the object in question), and the instance masks respectively and $\mathcal{S} \triangleq \{i : \text{IoU}(\mathcal{M}_i, \hat{M}) > 0.2\}$. Associating object renders to instance masks in this manner prevents incorrectly fusing object instances between nearby similar objects, in cases where there is large accumulated drift.

For subsequent fusion of a 2D instance detection to its associated 3D object,

the instance mask—containing the object foreground—and the bounding box mask—containing both the foreground and background are used. Similar to [21] we integrate the object in both the foreground and background through a weighted average of TSDF, color, and additionally maintain binomial foreground-background count variables for each voxel. This smoothes out artifacts from integration of 2D instances with spurious masks (see illustration in Figure 3.2).

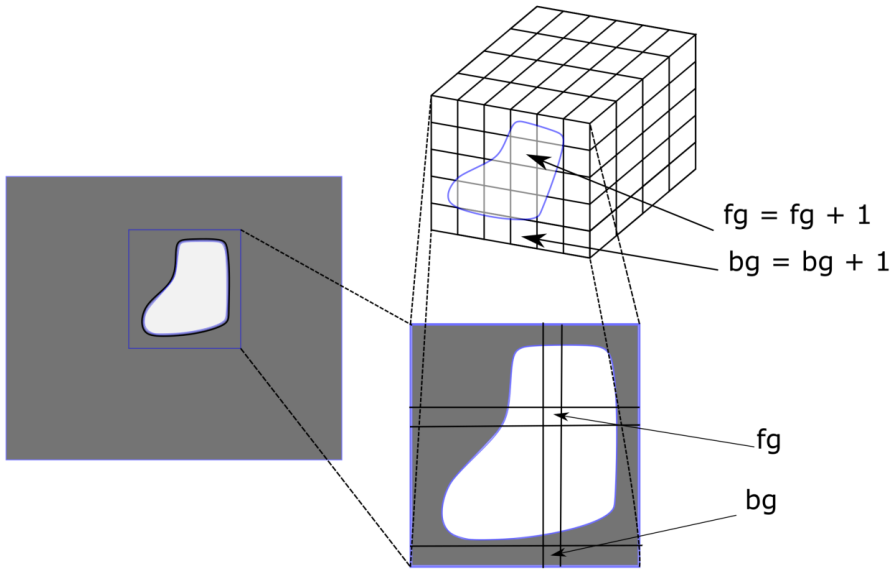



Figure 3.2: Each pixel is a binomial trial given a latent foreground probability of the associated voxel. By maintaining foreground and background counts, we estimate the expected latent probability required during rendering.

Finally, we update the object feature map by a gated weight average:

$$f_t = \frac{w_{t-1} \cdot f_{t-1} + \mathcal{H}(f_{t-1}, f_{in}) \cdot f_{in}}{w_{t-1} + \mathcal{H}(f_{t-1}, f_{in})} \tag{3.9}$$

$$\mathcal{H}(f_{t-1}, f_{in}) = \frac{\text{sgn}(\lambda - ||f_{t-1} - f_{in}||_1) + 1}{2} \tag{3.10}$$

where $\mathcal{H}$ is the Heaviside step function that hard-filters outlier input feature map $f_{in}$ compared to the maintained object feature map $f_{t-1}$ with weight $w_{t-1}$ controlled by the threshold $\lambda$.

### 3.2.5 Factor Graph Optimization

As we have mentioned before, a background volume is maintained for stable tracking and to handle *objectless* frames. The background volume additionally maintains the ratio ($r$) of visible volume units in the current camera frustum to the total number of allocated volume units in the volume. A low ratio implies that the camera may have moved away from a particular part of the scene. Pose graph optimization is conditionally triggered when the background volume is reset owing to low ratio of visible units ($r < 0.2$) and when there are new objects added into the graph.

Our object factor graph formulation is similar to [21, 35]. The variable nodes $\mathcal{X} = \{\mathbf{x}_1, \dots \mathbf{x}_N\}$ are partitioned into camera pose variables $\mathbf{T}_{C_i}^{W} \in SE(3)$ and object pose variables $\mathbf{T}_{O_j}^{W} \in SE(3)$. The first camera pose is initialized as the world frame $W$.

Assuming a Gaussian noise model, the *MAP* inference problem with the above variable nodes reduces to solving the following non-linear least squares optimization:

$$\mathcal{X}^* = \underset{\mathcal{X}}{\operatorname{argmin}} \Big( \sum_{k \in |C|} \| \mathbf{Z}_{C_{k-1}}^{C_k} \ominus \mathbf{T}_{C_{k-1}}^{C_k} \|^2_{\Sigma_{k,k-1}} + \sum_{j \in |\mathcal{O}|, k \in |\mathcal{C}|} \| \mathbf{Z}_{C_k}^{O_j} \ominus \mathbf{T}_{C_k}^{O_j} \|^2_{\Sigma_{o_j,k}} \Big) \tag{3.11}$$

where the operator $\mathcal{Y} \ominus \mathcal{X} = \operatorname{Log}(\mathcal{X}^{-1}\mathcal{Y})$ expresses the relative error in the local tangent vector space [36]. $\Sigma_{k,k-1}$ denotes the covariance between relative camera pose measurements, $\Sigma_{o_j,k}$ is the covariance in the camera to object measurement. They can be approximated by information matrices computed from *odometry*, however, empirically we found that a constant information matrix can achieve reasonable results. We obtain the relative camera measurements $\mathbf{Z}_{C_{k-1}}^{C_k}$ from *frame to model odometry* (§3.2.3), and obtain frame to object measurements $\mathbf{Z}_{C_k}^{O_j}$ by performing an additional Gauss Newton iteration with only the object pixels. Finally, the expected relative camera pose $\mathbf{T}_{C_{k-1}}^{C_k}$ and expected camera object pose $\mathbf{T}_{C_k}^{O_j}$ used in the factors are calculated as:

$$\mathbf{T}_{C_{k-1}}^{C_k} = \left( \mathbf{T}_{C_k}^{W} \right)^{-1} \mathbf{T}_{C_{k-1}}^{W}, \tag{3.12}$$

$$\mathbf{T}_{C_k}^{O_j} = \left( \mathbf{T}_{O_j}^{W} \right)^{-1} \mathbf{T}_{C_k}^{W}. \tag{3.13}$$

We solve the optimization in GTSAM [1] using Levenberg Marquardt. Since the entire object volume is transformed as a rigid body, the object volumes remain unchanged in memory after optimization. We note that this circumvents the time-consuming re-integration that usually takes place in volumetric methods after PGO [42].

### 3.2.6 Compositional Rendering

Compositional rendering is a serialized operation that generates normal, vertex, and color maps by ray-casting 3D objects in the viewing frustum into 2D object instances, and is illustrated in Figure 3.3.

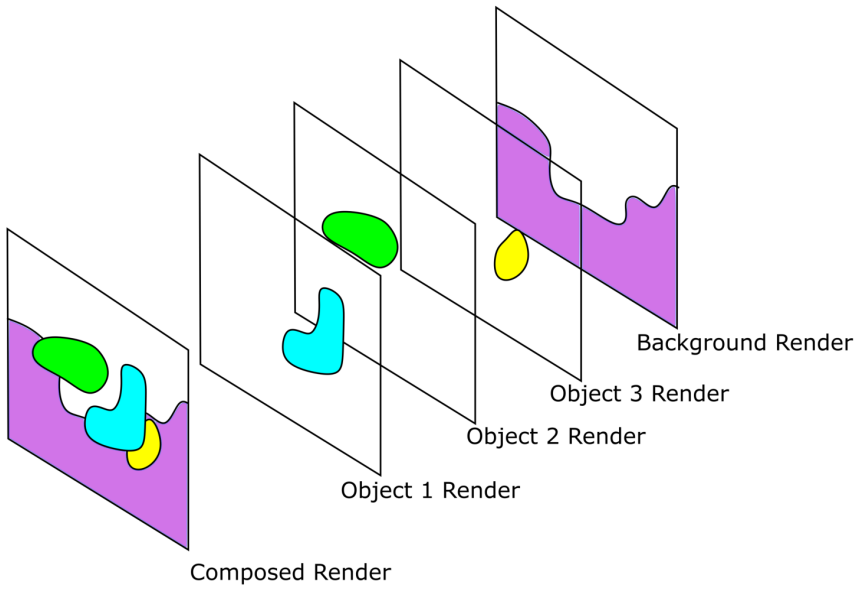



Figure 3.3: An illustration of compositional rendering.

$\langle \mathcal{N}_{C_i}, \mathcal{V}_{C_i}, \mathcal{C}_{C_i} \rangle$ is in fact an aggregation of separate renderings from object volumes $\langle \mathcal{N}_{C_i}^{V_{O_j}}, \mathcal{V}_{C_i}^{V_{O_j}}, \mathcal{C}_{C_i}^{V_{O_j}} \rangle$ and the background volume $\langle \mathcal{N}_{C_i}^{V_B}, \mathcal{V}_{C_i}^{V_B}, \mathcal{C}_{C_i}^{V_B} \rangle$, depending on the masks. In particular, we render the background volume, based on a background mask that is constructed from the union of existing virtual object masks in the current frame, and associated instance masks.

Then, the composed per-pixel map model render can be obtained as follows:

$$\hat{k} = \underset{k}{\operatorname{argmin}}\, \mathcal{V}_k(p)[z], \;\; k \in \{O_1, \cdots, O_n, B\} \tag{3.14}$$

$$\langle \mathcal{N}^*(p), \mathcal{V}^*(p), \mathcal{C}^*(p) \rangle = \langle \mathcal{N}_{\hat{k}}(p), \mathcal{V}_{\hat{k}}(p), \mathcal{C}_{\hat{k}}(p) \rangle, \tag{3.15}$$

where $\hat{k}$ is the volume index corresponding to the minimum distance to camera center for pixel $p$.

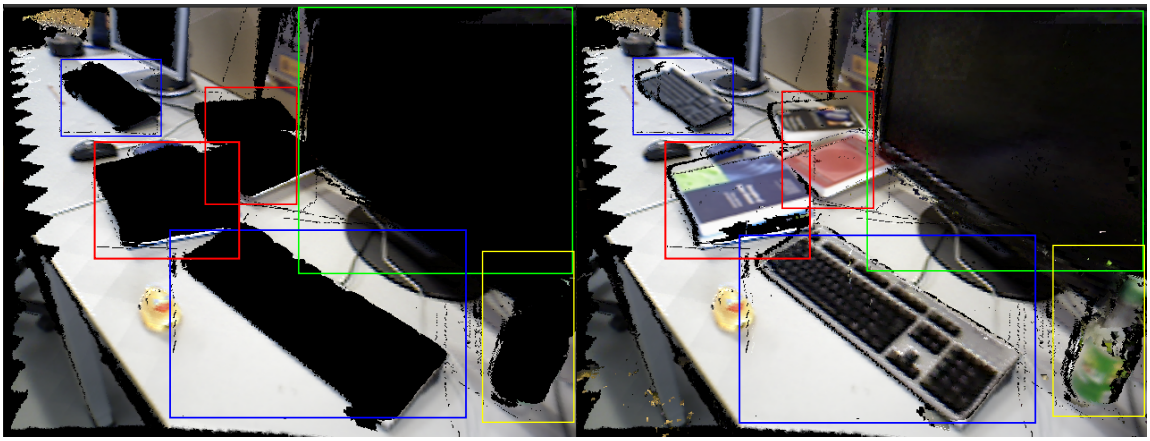


Figure 3.4: Compositional rendering during reconstruction on TUM *fr3_long_office_household* dataset. Left shows the background render, and right shows the composed render. Compositionally rendered objects are shown in bounding boxes.

Object volumes not currently visible are downloaded from GPU into CPU memory. Note that downloading the object volume does not affect the optimization problem, since the object volumes are required only for integration and raycasting.

# Chapter 4

# Experiments

In this chapter we demonstrate performance and comparable results of the proposed Object SLAM system with state of the art online and offline reconstruction systems in terms of trajectory accuracy and object reconstruction quality. We evaluate on the RGBD scenes V2 dataset [19] and the TUM RGBD dataset [37], both of which are established RGBD SLAM benchmarks and compare against baseline methods.

## 4.1 Architecture and Experimental Setup

To support relatively high frame rate operation in the presence of slow/non-realtime deep learning components our pipeline is highly parallelized. Our system adopts the *Actor* framework, where each component runs asynchronously, and communicates via thread-safe queues.

We implement the semantic segmentation pipeline as a separate python process which serializes the outputs using `protobuf` and communicates with the client thread via `zeromq` sockets in the Object SLAM pipeline. Since instance segmentation is carried out only for keyframes, the asynchronous python process exits early (typically) freeing GPU memory, which can be used for large reconstructions.

I implement the GPU code in CUDA, and leverage the Open3D GPU framework [9]. In particular, we build on the (in-house) spatially hashed TSDF implementation, implement an additional frame-to-model tracking module, and also build a back-end


Figure 4.1: Qualitative foreground object reconstruction results on *RGBD Scene 13* sequence.

module for pose graph optimization. We use the optimized spherical ray-casting method to improve performance.

Our experiments were run on a Linux system (Ubuntu 18.04) with Intel i7-6700 CPU at 4.00 GHz and 32GB of RAM and a NVIDIA GTX1080 with 8GB of GPU memory.

## 4.2 Qualitative Results

We first demonstrate qualitative reconstruction results on the RGBD scenes V2 dataset. Fig. 4.1 shows the object mesh extracted from our scalable volumes with a foreground count threshold. We can see that small objects are clearly reconstructed with details, and the background is correctly filtered.

At a larger scale, Fig. 1.2 segments teddy bear and computers from the cluttered scene and ensures a low-drift of the trajectory. Fig. 4.2 compares reconstructions from *MaskFusion* [33] and our system for a given sequence. It can be seen that the object-level reconstruction – specifically for the cap and sofa – is much cleaner in our system than in *MaskFusion*.

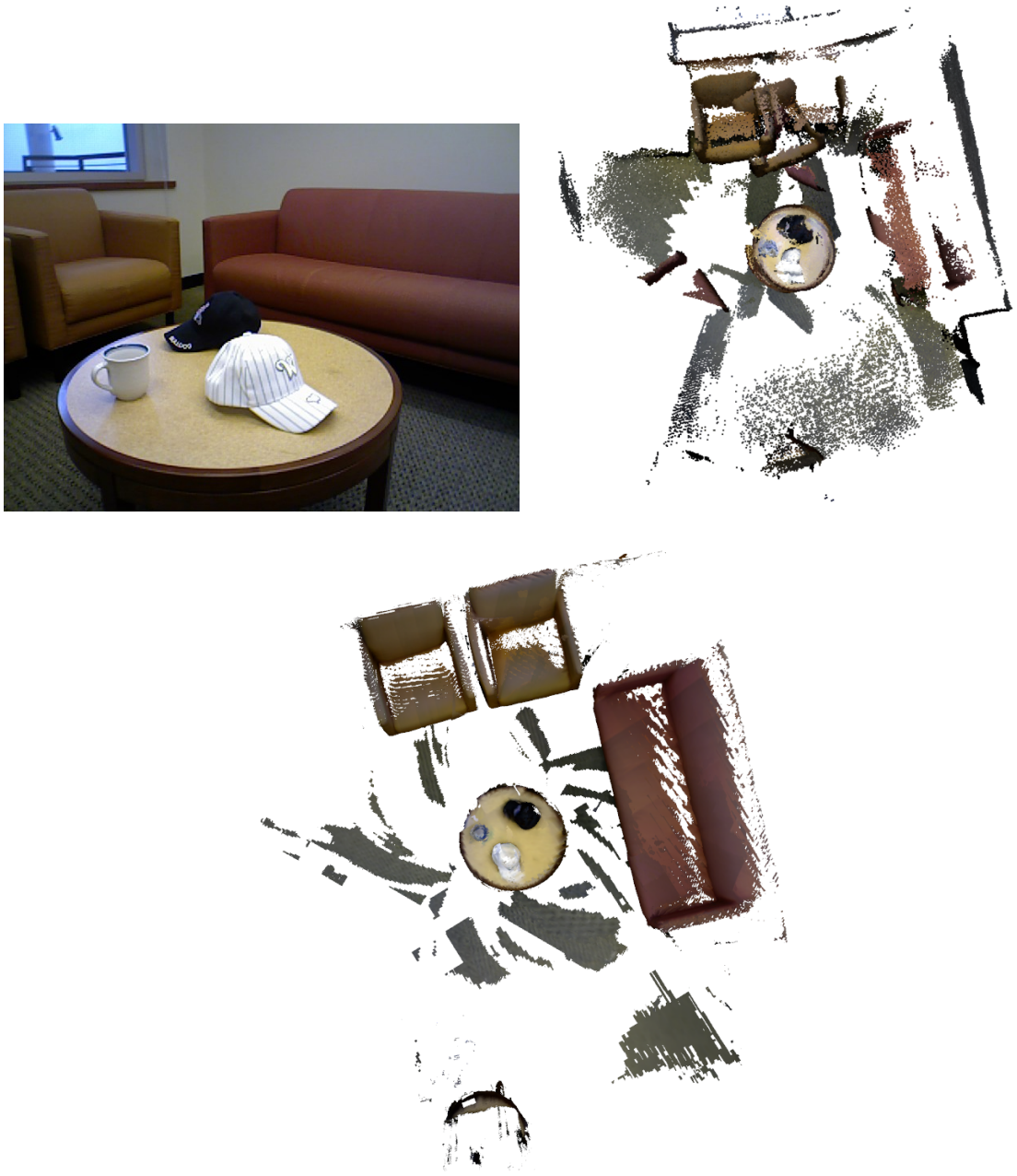


Figure 4.2: Reconstructed small indoor scene *RGBD Scene 12*. We first show an example input *RGB* frame (top-left) followed by a top-down view of the reconstruction from *MaskFusion* (top right). This is followed by result from our pipeline (bottom). Note that in our reconstruction background walls and floor are filtered out.

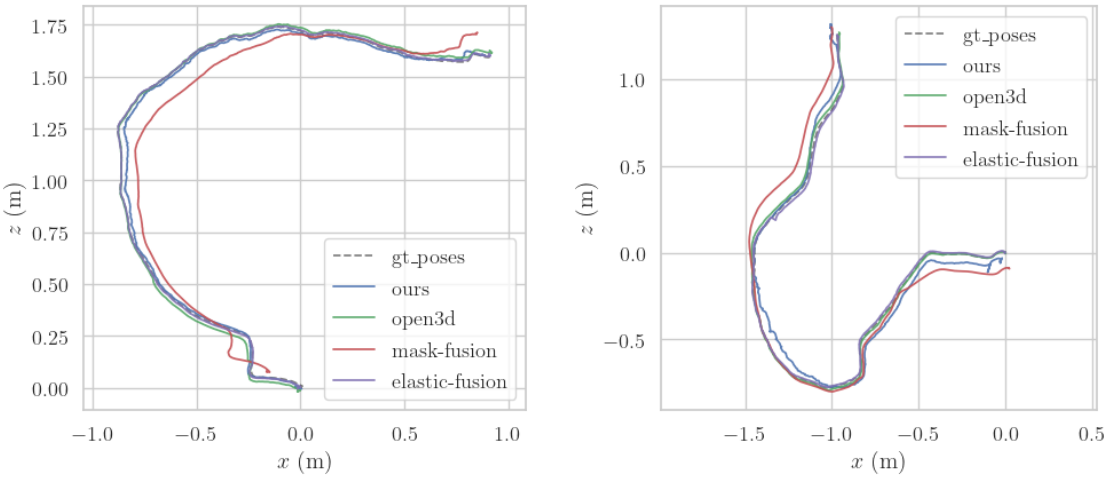
## 4.3 Quantitative Results

Table 4.1 presents Absolute Trajectory Error (ATE) of four different methods compared with our system. Note in the table, that we *bold* the best results of the object-based systems. We also provide results from geometric SLAM systems for comparison to present that our trajectory error is comparable to state of the art systems.

Table 4.1: Trajectory accuracy comparison on real world datasets (Absolute Trajectory Error in centimeters).

| Dataset | ElasticFusion | Open3D | MaskFusion | Fusion++ | Ours |
|---|---|---|---|---|---|
| Online | ✓ | | ✓ | ✓ | ✓ |
| Object Models | | | ✓ | ✓ | ✓ |
| RGBD Scenes - Scene 03 | 1.42 | 19.37 | 26.67 | - | **4.52** |
| RGBD Scenes - Scene 12 | 0.64 | 1.97 | 10.81 | - | **2.36** |
| RGBD Scenes - Scene 14 | 1.09 | 1.33 | 8.26 | - | **2.37** |
| fr1_xyz | 6.33 | 6.64 | 8.68 | - | **7.50** |
| fr1_desk | 2.70 | 5.73 | 24.05 | **4.9** | 5.82 |
| fr1_desk2 | 7.12 | 7.65 | 21.5 | 15.3 | **9.57** |
| fr1_room | 22.06 | 5.65 | 52.4 | 23.5 | **21.7** |
| fr2_xyz | 1.12 | 2.18 | 12.30 | **2.0** | 2.27 |
| fr2_desk | 7.61 | 4.72 | 163.6 | 11.4 | **9.94** |
| fr3_long_office | 2.23 | 3.54 | 140.8 | 10.8 | **9.68** |

In general, we achieve comparable results against the state-of-the-art surfel based online SLAM system *ElasticFusion* [42] and volumetric offline reconstruction system *Open3D* [44]. In the meantime, our method outperforms object-based SLAM systems *MaskFusion* [33] and *Fusion++* [21] by a large margin. This improvement can be attributed to the use of semantic data association and scalable voxel grids. It must be noted that *MaskFusion* requires 2 high end graphics cards to run it in online mode, therefore we ran it in offline mode, and stored all the detected object instances instead of manually selecting objects of interest for a fair comparison with our method. Additionally, we note that since Fusion++ is not open source, we obtain the results




(a) (b)

Figure 4.3: Comparison of trajectories between our pipeline and baselines with ground truth. (a) shows *rgbd-scenes-v12* and (b) shows *rgbd-scenes-v14* sequences.

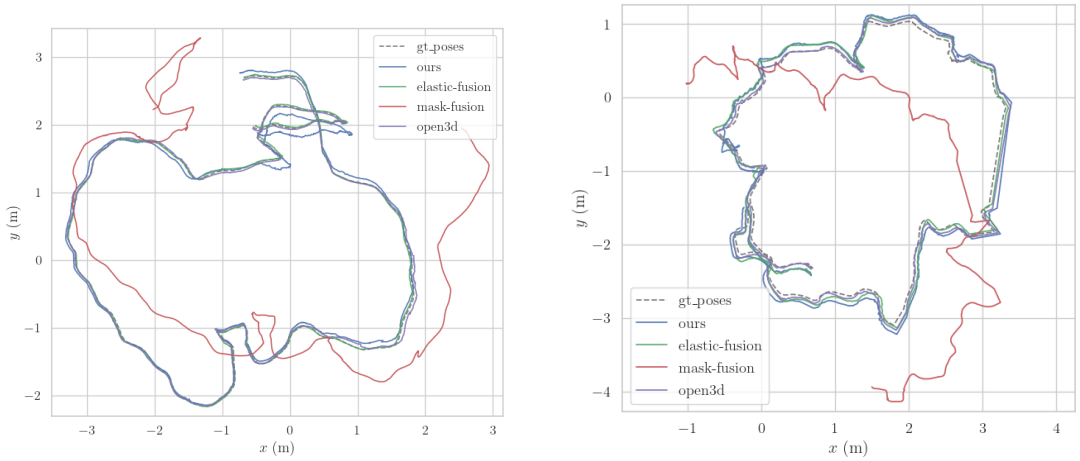
from the corresponding paper.

For small scenes in the *RGBD scenes V2 dataset*, we achieve consistently high accuracy with ATE below 5*cm* for all scenes. Figure 4.3 shows detailed trajectory visualizations.

For larger scenes, although noisy semantic segmentations affect masks and introduce noise for frame-to-model odometry, compositional rendering still ensures reliable tracking. Trajectory comparisons are provided in Figure 4.4.

## 4.4 Runtime Analysis

For runtime evaluation of our system we limit the number of initialized objects in the scene to 10 to ensure – close to – online performance on the aforementioned scenes. Processing each frame in the absence of any objects i.e., only background tracking takes about 200ms per frame. The largest computational bottleneck and time consuming operation is the rendering step, and while there are multiple rendering operations required (for instance, during object association), we render the objects and background only once per frame, and reuse the renders. We observe that each object takes on average about 45ms to render. In the presence of about 5-8 objects in




(a) (b)

Figure 4.4: Trajectory comparisons on the TUM-RGBD dataset (a) *fr3_long_office_household* and (b) *fr2_desk* sequences showing that even in the absence of explicit loop closures, our system maintains comparable accuracy.

the scene, the time taken per frame increases to about 450ms (200ms for background + 250ms for object renders) per frame. In comparison, we observed in our tests that *MaskFusion* runs at lower than 1FPS with one graphics card and suffers from random crashes in online mode. *Fusion++* reports its results with pre-computed segmentation masks. Our method runs seamlessly on a single GPU. A detailed runtime analysis is given in Table 4.2.

Table 4.2: Runtime breakdown component-wise for our pipeline

| Component | Tracking | Segmentation | Association | Rendering |
|---|---|---|---|---|
| Time (ms) | 13 | 250 | 15 | 45 per object |

# Chapter 5

# Conclusions and Future Work

## 5.1 Conclusion

In this thesis, I have investigated and presented a system for SLAM with higher level object landmarks, that provide both robust data association, and enable accurate trajectory estimation, and object reconstruction.

In this system, I develop a fast coarse-to-fine ICP to estimate initializations for the frame poses, I present an efficient method for composing renders as simple alpha matting between individual object renders, and I present a simple data association method that uses both geometric (the IoU metric or projective overlap) and semantic information (the object feature from the segmentation pipeline). Finally, combining these pieces in a factor graph optimization framework allows us to seamlessly correct for drift in the object poses – as rigid bodies. Finally, experimental results validated that our approach is both comparable to existing geometric SLAM methods and better than existing Object SLAM methods in terms of trajectory accuracy. We also presented qualitative results of object reconstruction.

## 5.2 Limitation and Future work

In this work, I have developed a system for scene reconstruction as a pose graph of objects.

To obtain the object masks and initializations we have resorted to existing semantic instance segmentation pipelines [17], and utilized geometric TSDF fusion as the method to integrate these measurements. Current deep learning based semantic segmentation pipelines introduce different types of errors:

1. Label inaccuracy: Since instance segmentation is trained only on single images, there exists no notion of temporal consistency of label predictions between frames. While we have alleviated this issue by comparing the object features and IoU between frames, this issue is far from solved. If we assume that the model label output to follows a categorical distribution, we may simply calculate the label of the fused data as the *mode* of the distribution. Note that in our method we use the object label only for pedagogical purposes and not for data association.
2. Mask inaccuracy: Our method quite heavily relies on the IoU metric for data association, which consequently assumes that the deep learning model generates similar masks for objects in temporally close frames. This assumption is violated in some cases, and results in multiple object landmark initializations. While we prune these objects by maintaining an existence probability for each object, we lose the information that was fused into these erroneous objects.

While our system loosely integrates a pre-trained semantic segmentation network, improvement on incorporating semantic information can be achieved through two orthogonal approaches:

1. Train networks explicitly for temporal consistency by training for object detection and instance segmentation on videos
2. Building an embodied agent with tightly coupled semantic pipeline based on self-supervised learning approaches. Some recent methods such as [3, 38] have shown promise is learning both motion and depth estimates via learning methods.

In this thesis, we update and integrate objects using geometric TSDF fusion, and do not utilize inductive biases from the object level inference to inpaint unseen parts of the object. TSDF fusion does not suffice for applications that lack dense views of the environment, which is largely the case. A potential direction of future work is to use a neural mapping representation that is capable of model completion for the objects. Recent improvements in implicit neural representations for scenes

such as [22, 28] provide rich avenues to explore. Since neural networks are excellent function approximators, using them to represent 3D surface manifolds is a reasonable approach that provides the additional benefit of imbuing object inductive biases.

Finally, we know that object landmarks are a rich description of the environment, and reduce the computational load in factor graph optimization significantly by reducing the number of factors in a pose graph. In addition to the proposed research directions, I am also excited to apply object based SLAM systems for multi-robot distributed inference, where distributed computation requires minimal factor sharing between robots.

# Bibliography


[1] Factor Graphs and GTSAM. http://gtsam.org/tutorials/intro.html, May 2019. 3.2.5

[2] Sean L. Bowman, Nikolay Atanasov, Kostas Daniilidis, and George J. Pappas. Probabilistic data association for semantic SLAM. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1722–1729, Singapore, Singapore, May 2017. IEEE. ISBN 978-1-5090-4633-1. doi: 10.1109/ICRA.2017.7989203. 2.2.1

[3] Ricson Cheng, Ziyan Wang, and Katerina Fragkiadaki. Geometry-Aware Recurrent Neural Networks for Active Visual Recognition. *arXiv:1811.01292 [cs]*, November 2018. 2

[4] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '96*, pages 303–312, Not Known, 1996. ACM Press. ISBN 978-0-89791-746-9. doi: 10.1145/237170.237269. 3.1.1

[5] Angela Dai, Matthias Nießner, Michael Zollöfer, Shahram Izadi, and Christian Theobalt. BundleFusion: Real-time globally consistent 3D reconstruction using on-the-fly surface re-integration. *ACM Transactions on Graphics 2017 (TOG)*, 2017. 3

[6] Timothy A Davis, Stefan I Larimore, John R Gilbert, and Esmond G Ng. Algorithm 8xx: COLAMD, a column approximate minimum degree ordering algorithm. page 5. 2.2.3

[7] Frank Dellaert and Michael Kaess. Square Root SAM: Simultaneous Localization and Mapping via Square Root Information Smoothing. *The International Journal of Robotics Research*, 25(12):1181–1203, December 2006. ISSN 0278-3649, 1741-3176. doi: 10.1177/0278364906072768. 2.2.3

[8] Frank Dellaert and Michael Kaess. Factor Graphs for Robot Perception. *Foundations and Trends in Robotics*, 6(1-2):1–139, 2017. ISSN 1935-8253, 1935-8261. doi: 10.1561/2300000043. 1.1, 2.2, 2.2.2, 3.1.1

[9] Wei Dong, Jaesik Park, Yi Yang, and Michael Kaess. GPU Accelerated Robust Scene Reconstruction. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7863–7870, Macau, China, November 2019. IEEE. ISBN 978-1-72814-004-9. doi: 10.1109/IROS40897.2019.8967693. 1.3, 1, 2.2.3, 3.1.1, 3.2.3, 4.1

[10] Jakob Engel, Thomas Schops, and Daniel Cremers. LSD-SLAM: Large-Scale Direct Monocular SLAM. page 16. 3.1.1

[11] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct Sparse Odometry. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 40(3):611–625, March 2018. ISSN 0162-8828, 2160-9292. doi: 10.1109/TPAMI.2017.2658577. 1.2, 3.1.1

[12] Giorgio Grisetti, Rainer Kummerle, Cyrill Stachniss, and Wolfram Burgard. A Tutorial on Graph-Based SLAM. page 11. 2.2

[13] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. *arXiv:1703.06870 [cs]*, January 2018. 1.2, 3.1.2

[14] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 34(3):189–206, April 2013. ISSN 0929-5593, 1573-7527. doi: 10.1007/s10514-012-9321-0. 2.1.1, 1

[15] M. Kaess. *Incremental Smoothing and Mapping*. Ph.D., Georgia Institute of Technology, December 2008. 2.2

[16] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. iSAM2: Incremental smoothing and mapping using the Bayes tree. *The International Journal of Robotics Research*, 31(2):216–235, February 2012. ISSN 0278-3649, 1741-3176. doi: 10.1177/0278364911430419. 2.2.3

[17] Alexander Kirillov, Yuxin Wu, Kaiming He, and Ross Girshick. PointRend: Image Segmentation as Rendering. *arXiv:1912.08193 [cs]*, February 2020. 1.3, 3.1.2, 5.2

[18] Georg Klein and David Murray. Parallel Tracking and Mapping for Small AR Workspaces. In *2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 1–10, Nara, Japan, November 2007. IEEE. ISBN 978-1-4244-1749-0. doi: 10.1109/ISMAR.2007.4538852. 3.1.1

[19] Kevin Lai, Liefeng Bo, and Dieter Fox. Unsupervised feature learning for 3D scene labeling. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3050–3057, May 2014. doi: 10.1109/ICRA.2014.6907298. 4

[20] Giuseppe Loianno, Michael Watterson, and Vijay Kumar. Visual inertial

odometry for quadrotors on SE(3). In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1544–1551, May 2016. doi: 10.1109/ICRA.2016.7487292. 2.1

[21] John McCormac, Ronald Clark, Michael Bloesch, Andrew J. Davison, and Stefan Leutenegger. Fusion++: Volumetric Object-Level SLAM. *arXiv:1808.08378 [cs]*, August 2018. 1.2, 3.1.2, 3.2.4, 3.2.4, 3.2.5, 4.3

[22] Ben Mildenhall, Pratul P. Srinivasan, Matthew Tancik, Jonathan T. Barron, Ravi Ramamoorthi, and Ren Ng. NeRF: Representing Scenes as Neural Radiance Fields for View Synthesis. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, volume 12346, pages 405–421. Springer International Publishing, Cham, 2020. ISBN 978-3-030-58451-1 978-3-030-58452-8. doi: 10.1007/978-3-030-58452-8_24. 5.2

[23] R. Mur-Artal and J. D. Tardós. ORB-SLAM2: An Open-Source SLAM System for Monocular, Stereo, and RGB-D Cameras. *IEEE Transactions on Robotics*, 33 (5):1255–1262, October 2017. ISSN 1941-0468. doi: 10.1109/TRO.2017.2705103. 1.2, 2.1, 3.1.1

[24] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. KinectFusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, October 2011. doi: 10.1109/ISMAR.2011.6092378. 2.1, 2.1.1, 3, 3.1.1, 3.2.3

[25] Lachlan Nicholson, Michael Milford, and Niko Sünderhauf. QuadricSLAM: Dual Quadrics From Object Detections as Landmarks in Object-Oriented SLAM. *IEEE Robotics and Automation Letters*, 4(1):1–8, January 2019. ISSN 2377-3766. doi: 10.1109/LRA.2018.2866205. 3.1.2

[26] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3D reconstruction at scale using voxel hashing. *ACM Transactions on Graphics*, 32(6):1–11, November 2013. ISSN 07300301. doi: 10.1145/2508363.2508374. 1, 2.2.3, 3.1.1

[27] Jaesik Park, Qian-Yi Zhou, and Vladlen Koltun. Colored Point Cloud Registration Revisited. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 143–152, Venice, October 2017. IEEE. ISBN 978-1-5386-1032-9. doi: 10.1109/ICCV.2017.25. 3.2.3

[28] Jeong Joon Park, Peter Florence, Julian Straub, Richard Newcombe, and Steven Lovegrove. DeepSDF: Learning Continuous Signed Distance Functions for Shape Representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174, 2019. 5.2

[29] Victor Adrian Prisacariu, Olaf Kähler, Stuart Golodetz, Michael Sapienza, Tom-

maso Cavallari, Philip H. S. Torr, and David W. Murray. InfiniTAM v3: A Framework for Large-Scale 3D Reconstruction with Loop Closure. *arXiv:1708.00783 [cs]*, August 2017. 2.2.3

[30] Tong Qin, Peiliang Li, and Shaojie Shen. VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator. *IEEE Transactions on Robotics*, 34 (4):1004–1020, August 2018. ISSN 1941-0468. doi: 10.1109/TRO.2018.2853729. 2.1

[31] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 779–788, 2016. 1.2

[32] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *arXiv:1506.01497 [cs]*, January 2016. 1.2, 3.1.2

[33] Martin Rünz, Maud Buffier, and Lourdes Agapito. MaskFusion: Real-Time Recognition, Tracking and Reconstruction of Multiple Moving Objects. April 2018. 1.2, 3.1.2, 4.2, 4.3

[34] S. Rusinkiewicz and M. Levoy. Efficient variants of the ICP algorithm. In *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pages 145–152, Quebec City, Que., Canada, 2001. IEEE Comput. Soc. ISBN 978-0-7695-0984-6. doi: 10.1109/IM.2001.924423. 2.1.1

[35] Renato F. Salas-Moreno, Richard A. Newcombe, Hauke Strasdat, Paul H.J. Kelly, and Andrew J. Davison. SLAM++: Simultaneous Localisation and Mapping at the Level of Objects. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1352–1359, Portland, OR, USA, June 2013. IEEE. ISBN 978-0-7695-4989-7. doi: 10.1109/CVPR.2013.178. 1.2, 3, 3.2.5

[36] Joan Solà, Jeremie Deray, and Dinesh Atchuthan. A micro Lie theory for state estimation in robotics. *arXiv:1812.01537 [cs]*, August 2020. 3.2.5

[37] Jrgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of RGB-D SLAM systems. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 573–580, Vilamoura-Algarve, Portugal, October 2012. IEEE. ISBN 978-1-4673-1736-8 978-1-4673-1737-5 978-1-4673-1735-1. doi: 10.1109/IROS.2012.6385773. 4

[38] Zachary Teed and Jia Deng. DeepV2D: Video to Depth with Differentiable Structure from Motion. *arXiv:1812.04605 [cs]*, April 2020. 2

[39] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005. ISBN

978-0-262-20162-9. 2.1.1

[40] Bill Triggs, Philip F. McLauchlan, Richard I. Hartley, and Andrew W. Fitzgibbon. Bundle Adjustment — A Modern Synthesis. In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Bill Triggs, Andrew Zisserman, and Richard Szeliski, editors, *Vision Algorithms: Theory and Practice*, volume 1883, pages 298–372. Springer Berlin Heidelberg, Berlin, Heidelberg, 2000. ISBN 978-3-540-67973-8 978-3-540-44480-0. doi: 10.1007/3-540-44480-7_21. 3.1.1

[41] Thomas Whelan, John McDonald, Michael Kaess, Maurice Fallon, Hordur Johannsson, and John J Leonard. Kintinuous: Spatially Extended KinectFusion. page 8. 3

[42] Thomas Whelan, Stefan Leutenegger, Renato Salas Moreno, Ben Glocker, and Andrew Davison. ElasticFusion: Dense SLAM Without A Pose Graph. In *Robotics: Science and Systems XI*, volume 11, July 2015. ISBN 978-0-9923747-1-6. 2.1, 3.2.5, 4.3

[43] Shichao Yang and Sebastian Scherer. Monocular Object and Plane SLAM in Structured Environments. *IEEE Robotics and Automation Letters*, 4(4): 3145–3152, October 2019. ISSN 2377-3766. doi: 10.1109/LRA.2019.2924848. 3.1.2

[44] Qian-Yi Zhou, Jaesik Park, and Vladlen Koltun. Open3D: A Modern Library for 3D Data Processing. *arXiv:1801.09847 [cs]*, January 2018. 3.1.1, 4.3