3D Reconstruction of Dynamic Environments Using Egomotion Compensated Optical Flow

Connor Keevill

Bachelor of Science in Computer Science
The University of Bath
2022-2023

3D Reconstruction of Dynamic Environments Using Egomotion Compensated Optical Flow

Submitted by: Connor Keevill

Copyright

Declaration

This dissertation is submitted to the University of Bath in accordance with the requirements of the degree of Bachelor of Science in the Department of Computer Science. No portion of the work in this dissertation has been submitted in support of an application for any other degree or qualification of this or any other university or institution of learning. Except where specifically acknowledged, it is the work of the author.

Abstract

The field of 3D reconstruction has enjoyed something of a renaissance in recent years, with the commodification of RGB-D sensor technology and advancements in GPU capabilities fuelling a wave of research. Despite impressive advancements in the area, a persistent challenge remains in the reconstruction of so-called *dynamic* environments, where the contents of a scene change through time. Dynamic environments present a challenge to 3D reconstruction systems due to the entanglement of scene motion and camera motion in each observation of the scene.

This paper presents a novel system capable of reconstructing a wide range of dynamic environments through an approach termed *egomotion compensated optical flow*, wherein dynamic objects are detected and segmented from a reconstruction through the simultaneous estimation of an optical flow field and the camera's own egomtion. The novel system is evaluated against the state-of-the-art and is shown to perform as well as, and in some cases outperform, the state-of-the-art system used.

The accompanying code for this paper is available on GitHub: `https://github.com/connorkeevill/dynamic-3D-reconstruction`. The accompanying video is available on YouTube: `https://youtu.be/zrWnW1QZjHM`

Contents

Acknowledgements

I would like to thank Dr Yong-Liang Yang for his guidance and support throughout the course of this project, and Dr Tom Haines and Stuart Green for their provisioning of Hex. Thank you also to Anthony Escott-Lawrence for his role as a mentor to me and for sparking my interest in Computer Science.

Finally, thank you to my parents for their ongoing support throughout my degree.

Chapter 1

Introduction

Despite a dense body of research surrounding the topic, 3D scene reconstruction - the task of reconstructing a 3D model of a space from a set of images - remains an unsolved task within the field of Computer Vision, with modern systems struggling when scenes contain dynamic (i.e., moving) objects Figure 1.1 demonstrates accelerated research efforts as a result of recent advances in RGB-D sensor technology (ordinary RGB imaging combined with depth information), stemming from the 2010 release of the Microsoft Kinect (a technology licensed from PrimeSense [4]). Since then, various other devices have been showcased, from dedicated sensors like the Asus Xtion Live and Microsoft Kinect v2, to integrated sensors such as those seen in Apple's iPhones and iPads (the result of a $345 million acquisition of PrimeSense[1]), and Tesla's self-driving cars.

Accessible consumer grade sensors facilitate and promote subject research. One of the most notable papers in this space is KinectFusion [1, 2], published in the wake of Kinect's release. KinectFusion proposes a system capable of dense online 3D scene reconstruction from a handheld Kinect. Kintinuous [5] and ElasticFusion [6, 7] build on the work of KinectFusion to develop systems which can capture scenes of a larger scale (Kintinuous), and handle more complex camera pose trajectories (ElasticFusion). The results from these systems (and further works shown above in Figure 1.1) are very compelling, but share a common shortcoming: they struggle with dynamic scenes.

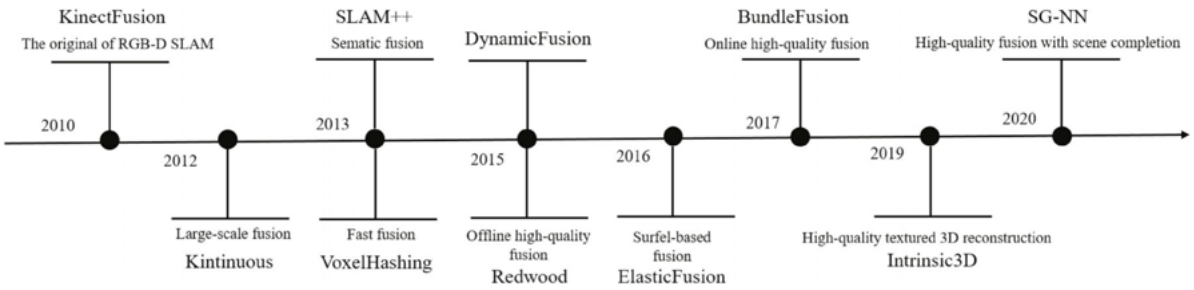


Figure 1.1: Timeline of research into 3D scene reconstruction since the influential KinectFusion paper [1, 2]. Figure from [3]

[1] https://www.reuters.com/article/us-primesense-apple-idUSBRE9AG03G20131117

1.1 Problem Description

Dynamic scenes are a common occurrence in the real world, and are a significant challenge for 3D scene reconstruction systems, which typically assume that the motion observed on the camera plane is solely the result of the camera moving through the scene. Dynamic scenes break this assumption, introducing the task of disambiguating the motion of the camera from the motion of the scene.

The task of dynamic scene reconstruction can be viewed as a dimensionality reduction problem, where the goal is to reduce the dimension of the scene from 4D (3D space + time) to a 3D space containing just the static components of the scene.

1.2 Objectives

Teaching a computer to see is the problem at the heart of Computer Vision. The ability to reconstruct a 3D model of a scene from a set of images is a key component of this problem, and is essential for many applications, such as Simultaneous Localisation and Mapping (SLAM), Augmented Reality (AR), Robotics, and Autonomous Vehicles. The requirement that the scenes in all of these applications be static is unrealistic, and so the ability to handle dynamic scenes is essential for the wider success of 3D scene reconstruction systems.

The system which is the subject of this dissertation aims to solve the problem of dynamic scene reconstruction by proposing an optical flow-based approach to the disambiguation of camera and scene motion. The primary objectives of the project are the following:

- perform a literature review of the state-of-the-art in 3D scene reconstruction;
- develop a system capable of detecting dynamic objects in a scene;
- use the developed system to perform dynamic 3D reconstruction;
- evaluate the developed system against the state-of-the-art;
- identify areas for future work.

1.3 Dissertation Structure

The remainder of this report is structured as follows:

- **Chapter 2: Related Work**
  A review of the literature is conducted, introducing the basic concepts of static and dynamic reconstruction, identifying key works which have contributed to the current state-of-the-art, and highlighting the works which this project sits at the intersection of.
- **Chapter 3: Methodology**
  The mathematics behind the proposed system are presented, and the implemented system is explained.
- **Chapter 4: Experimental Evaluation**
  This chapter performs a comparative evaluation of the developed system against the state-of-the-art and identifies key strengths and limitations of the project.

- **Chapter 5: Conclusion**
  The final chapter concludes the dissertation by summarising the work, identifying areas for future work, and discussing the personal journey of working on the project.

Chapter 2

Related Work

Recent years have seen researchers capitalize on the emergence of commodity RGB-D sensors and more capable GPUs to deliver impressive advancements in the field of 3D scene reconstruction. This chapter aims to condense around two and a half decades of research into the key achievements which have contributed most to the current state-of-the-art, and describe what the state-of-the-art is.

The review will begin with an overview of the static reconstruction problem, introducing the reconstruction pipeline and some notable works in the literature. Next, dynamic reconstruction and its challenges shall be explored. The areas where the pipeline deteriorates will be identified and works which attempt to solve these shortcomings also introduced. Finally, having formed a basis of the state-of-the-art, some avenues for future work will be proposed, laying the groundwork for the rest of this project.

2.1 Static Reconstruction

This section will introduce static reconstruction: a term used here to describe a 3D reconstruction system operating in a static environment. Much of the recent research observed in the field of static reconstruction can be traced back to Izadi et al.'s influential 2011 paper, KinectFusion [1, 2]. Therefore, in introducing static reconstruction, this section will also explore the key works which contributed to KinectFusion and those which drew inspiration from it.

2.1.1 The Problem

A sequence of RGB-D images (i.e., a video) of a scene is captured from a sequence of unknown positions and angles. The task of a 3D reconstruction system is to infer the location of the camera at each image captured and, using the inferred trajectory as a basis, construct a 3D

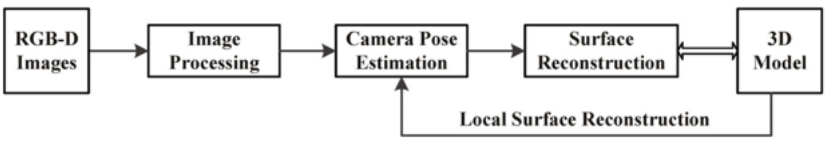


Figure 2.1: A typical 3D reconstruction pipeline as described by [3]

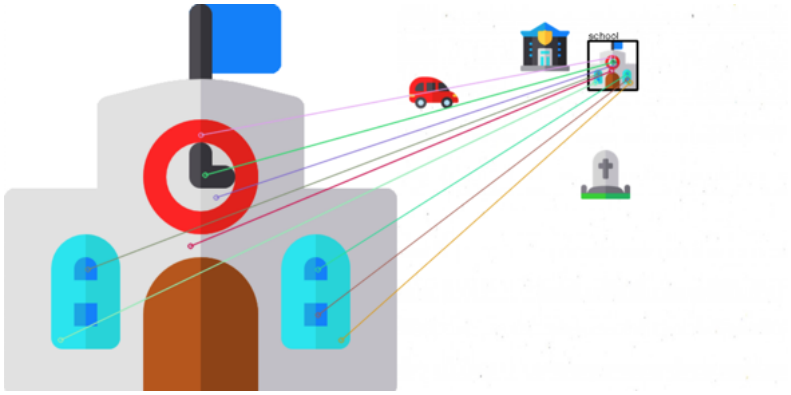


Figure 2.2: An illustration of SIFT [8] key point correspondences for image detection. Source: https://github.com/connorkeevill/emoji-detection-with-SIFT

model of the scene. Figure 2.1 shows a typical pipeline for 3D reconstruction as described by Li et al. [3], consisting of 3 main stages: Image Processing, Camera Pose Estimation, and Surface Reconstruction.

2.1.2 Image Processing

Image Processing consists of converting the raw RGB-D data captured by a sensor (e.g.: the Microsoft Kinect [4]) into a data structure which is amenable to the techniques employed by the rest of the reconstruction system. The Image Processing stage is, therefore, very sensitive to changes further down the reconstruction pipeline and, as a consequence, seldom has a rigid structure. Therefore, this section will aim to illustrate the range of strategies used by static reconstruction systems, rather than provide a comprehensive account of each individual approach.

Scale-Invariant Feature Transform

The Scale-Invariant Feature Transform (SIFT) [8] is an image key point extraction technique proposed by David Lowe in 2004. SIFT enables the identification and description of key points in an image. Key points are identified at various scales using a Gaussian pyramid to approximate a Laplacian operator using a Difference of Gaussians [9]. Key points are described with a 128-element feature vector describing the gradients of points surrounding the key point. Given a point from some query image, a match can be found in a point database by finding the closest feature vector in $\mathbb{R}^{128}$.

In 3D reconstruction, SIFT can be used to find correspondences between frames (see Figure 2.2), for example: for alignment when performing loop-closures [10]. While SIFT is accurate, it is too computationally complex for use in low-latency applications [11, 12], rendering it unfashionable for modern *online* systems. Moreover, an increase in the capability of GPUs has led to recent algorithms (most notably, KinectFusion's [1, 2] GPU based implementation of hierarchical ICP) which can handle *dense* point clouds. These are preferable to *sparse* features as the higher number of points can reduce the effect of noise and improve reconstruction accuracy.

Camera Space Projection

While each frame of RGB-D video is indeed an image, the inclusion of depth yields a 3D image, or so called *point cloud*. Therefore, as described by Palazzolo et al. [13], points from an RGB-D image can be projected into a 3 dimensional *camera space*, where each point becomes a point in $\mathbb{R}^3$ relative to the camera's basis vectors - much like casting a ray when ray tracing.

Given a point $p = [x\ y]^T \in \mathbb{R}^2$ and the function $D(p) : \mathbb{R}^2 \mapsto \mathbb{R}$ which maps the point onto its depth value, the point $p' = P(p) \in \mathbb{R}^3$ is the point in camera space, where $P(p) : \mathbb{R}^2 \mapsto \mathbb{R}^3$ is defined as:

$$P(p) = \begin{bmatrix} \frac{x - C_x}{f_x} D(p) \\ \frac{y - C_y}{f_y} D(p) \\ D(p) \end{bmatrix}, \tag{2.1}$$

where C_x, C_y, f_x, f_y are the intrinsic camera parameters.

Equation 2.1 is widely used and is a central component of many 3D reconstruction systems [1, 5, 6, 14, 13]. The main limitation of such an approach is the prerequisite of knowing the camera intrinsic parameters. Koch et al. [15] proposed a solution for calibrating (i.e. inferring camera intrinsics) uncalibrated images, providing a potential solution to this problem. However, for this project, the assumption that camera intrinsics are known will be made.

Bilateral Filtering

Point-based filter operations are some of the most fundamental image preprocessing techniques [9]. A common application of point operations is de-noising: the smoothing of a signal to reduce interference from erroneous sensor readings. However, an often faced problem with smoothing operators (e.g. a box filter or the Gaussian filter) is that, in removing high frequency noise, they remove high frequency detail from the image which denotes an edge. The Bilateral Filter [16] attempts to solve this problem by simultaneously smoothing out undesired noise while retaining the details which are needed for edge detection.

Bilateral filtering is used by many systems. KinectFusion [1, 2] and Or-El et al.'s RGBD-Fusion [17] apply the bilateral filter to the raw depth map, enhancing the quality of the normal maps produced. Meanwhile, Yang et al. [18] employ bilateral filtering on both the depth map *and* the normal map, here attempting to infer an unknown candidate datapoint from its surrounding data rather than filter a known depth value or normal.

Depth Map Hole Filling

Similar to the problem of de-noising is the problem of incomplete data. Data can be absent for a number of reasons, depending on sensor type, but can typically be blamed on reflections in the scene hindering the ability to detect a dot map (as with the Kinect [4]), objects being too far away, or occlusions blocking the background due to misalignment in the depth and RGB cameras.

Berdnikov and Vatolin [19] present a system capable of in-painting such holes in depth maps. Their approach involves classifying the type of hole as either the result of an occlusion due to misalignment (seen around the edges of objects) or a reflection in the scene. They then apply one of two different strategies to in-paint the missing region. Deep learning can also help to generate the missing data. SG-NN [20] is a static reconstruction system which aims to be robust to noisy or incomplete data by training a generative neural network - through

self supervised training - to predict what an incomplete mesh should look like. Both of these systems are limited in their ability to handle dynamic environments, but may be complementary to this project.

2.1.3 Camera Pose Estimation

The critical stage of the pipeline in Figure 2.1 (and the one most challenged by dynamic scenes) is Camera Pose Estimation (CPE): the inference of the camera's position and orientation relative to both the previous frame in the sequence (i.e. the rigid transformation which describes the camera motion between the two frames) and the global coordinate system. The aim of this stage is to produce a trajectory (a sequence of positions and orientations) which accurately describes the motion of the camera when capturing the video. This camera trajectory can be described as the "backbone" [21] of a 3D reconstruction and therefore has been the subject of considerable research (e.g. [11]).

Iterative Closest Point

A key algorithm for CPE is Iterative Closest Point (ICP) [22]. ICP is a popular algorithm to align 3D models, but in reconstruction systems it is used to track camera pose. ICP aims to find the rigid transformation which describes the motion between frames F_{i-1} and F_i, by starting with an initial guess for the transformation, and iteratively refining this by finding correspondences between *points* to minimize an error metric [23]. ICP yields a more accurate transformation as more points are introduced (as shown by Rusinkiewicz and Levoy [23]), but doing so also increases the computational cost of the algorithm; it is a trade-off. Some older systems (e.g. [10, 21, 24]) used *sparse* points, extracting points of interest during the Image Processing stage of the pipeline in Figure 2.1, e.g. with SIFT [8], and performing ICP with these features. Yet, as GPUs have grown more capable, researchers have moved towards *dense* point tracking. A notable example of this is Izadi et al.'s seminal KinectFusion [1, 2], presenting a novel GPU-based ICP algorithm, capable of treating the entire 640×480 resolution of the Microsoft Kinect [4] as a point cloud with which to perform ICP.

While Izadi et al. [1, 2] postulate that the dense point cloud tracking employed by KinectFusion allows ICP to be more robust to transient scene motion, dense ICP does not solve the problem of dynamic reconstruction, with the researchers finding that larger or long term scene motion leads to tracking failure.

Loop-Closures

Though ICP does a good job at tracking the local frame-to-frame transformations, ICP alone is not yet sufficient to perform end-to-end camera tracking, as cumulative error from ICP and noise in the data can introduce a degree of uncertainty. The second key component of reliable CPE is the application of *loop-closures*. A loop-closure occurs in a 3D reconstruction system when it encounters a region of the scene which is has already registered as part of the model it is constructing. The introduction of such a reference point allows the *loop* (the accumulated trajectory since either the last loop-closure or the start of the reconstruction) to be closed, in an event which corrects the "current" position of the camera and propagates the correction down the rest of the loop.

ElasticFusion [6, 7] makes significant use of loop-closures (performing frame-to-model tracking

instead of the frame-to-frame ICP described above). Their approach attempts to perform a loop-closure every frame, deforming the model if the loop-closure is successful.

2.1.4 Surface Reconstruction

Knowing the camera trajectory allows the point cloud each RGB-D frame represents to be projected into a global coordinate space, and 3D geometry can start to form.

Voxel Model

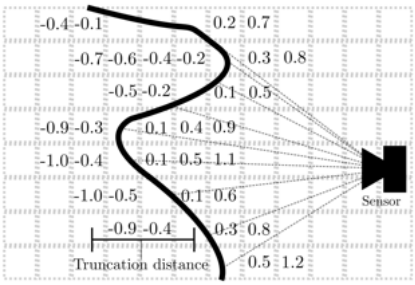


Figure 2.3: Cross sectional model showing the implicit surface representation of TSDF, from [25]

KinectFusion [1, 2] uses Curless and Levoy's approach [26] for volumetric model representation. A 3D grid of uniformly sized *voxels* of the scene is formed at a fixed resolution and a Truncated Signed-Distance Function (TSDF) is used: a value in each voxel specifying the distance to the nearest surface, with the sign indicating the side of the surface that the voxel is on (see Figure 2.3). Surfaces are found at zero-crossings of the TSDF, and the geometry is refined beyond the resolution of the voxel grid by linearly interpolating the values around the zero-crossing. This technique produces compelling results, but is memory-inefficient, leading to *moving volume* variants which stream "inactive" regions of space out of the voxel grid [5], and others which dynamically allocate memory for voxels as they are needed [27, 14, 13]. Another weakness of such a voxel model, identified by Whelan et al. [5], is its inability to properly model deformations (for example after loop-closure).

Surfels

Contrasting the volumetric approach described above is a *surfel*-based model, wherein the 3D model becomes a set of *surfels* [28] (**surf**ace **el**ements); discs in space defined by a position, surface normal, colour, and radius.

Surfels, whilst more conceptually abstract, aim to solve the key limitations identified in the voxel model. Essentially a point cloud, a surfel representation of a 3D space allows for extremely fine deformations in the model on a per-surfel basis (as described by [29] and used by [6, 7]). This means that model deformations (as a result of a loop-closure) are much easier: apply a transformation to the misaligned surfels. The other primary benefit of surfels is their memory efficiency. Henry et al. [30] post process a point cloud to produce a surfel model which incorporates all information into a memory-efficient representation of the scene. Surfels are a powerful alternative to traditional rendering primitives (i.e., polygons and quadrics), but fall short as a surface representation due to the point cloud structure that they follow. As [13] identifies, the lack of explicit connectivity between surfels complicates the mesh generation required as part of a reconstruction system.

2.2 Dynamic Scenes

This section will continue the review of the literature by discussing the problems faced when trying to reconstruct dynamic scenes and presenting some proposed solutions from the state-of-the-art. The difficulty with dynamic scenes stems from the fact that they change through time. While a static reconstruction system can assume that differences between frames (ignoring noise, white balance, AGC, etc.) are due to camera motion (egomotion), and therefore offer new information to fuse into the model, a dynamic reconstruction system has to contend with differences between frames caused by the *environment* changing. This challenges both CPE (requiring dynamic object detection) and Surface Reconstruction (both when surfaces are occluded by dynamic objects and when dynamic objects are mistakenly fused into the model).

2.2.1 CPE in Dynamic Scenes

Dynamic environments complicate the CPE problem with the additional task of estimating which parts of a scene are moving and which are static. In other words: to handle dynamic scenes, CPE must be able to determine whether the motion perceived is a result of camera motion or scene motion. Failure to do so will result in an inability to find correspondences and a collapse of the reconstruction pipeline. This presents a chicken-and-egg problem, where, to understand camera pose, the system must be able to segment dynamic objects, but, to segment the dynamic objects, the system must have some notion of camera movement. ICP alone is not sufficient, assuming that a single rigid transformation occurs in each frame due to camera motion.

Non-Rigid Warping of Canonical Scenes

An important feature of KinectFusion [1, 2] is the human interaction it offers. Users can move objects and the system can keep track of them, but it is assumed that camera tracking is performed with a *static* scene. Thus, an initial registration of the scene is required to reconstruct it, before transitioning to focus on interaction. During interaction, KinectFusion "locks" onto the background, which it uses for camera tracking, allowing independent tracking and reconstruction of dynamic foreground objects.

Newcombe et al.'s DynamicFusion [31] takes inspiration from KinectFusion's approach, but tries to eliminate the need for an initial registration. The researchers present a dense SLAM system which can reconstruct non-rigidly deforming scenes. Their approach hinges on the idea of capturing a *canonical* model of a dynamic object and, for each frame, computing the non-rigid *warp* which transforms the dynamic object into the canonical pose. This yields a static scene on which KinectFusion-style reconstruction can be performed, sidestepping the chicken-and-egg problem described above. Afterwards, dynamic motion can be extracted from the reconstructed scene by applying the inverse warp to each transformed frame. Though the results are very impressive[1], DynamicFusion struggles with large inter-frame differences and topology changes, both of which can lead to irrecoverable model corruption. Furthermore, the system is aimed at subject motion capture rather than larger scale scene capture.

This same approach is used by MixedFusion [14], but Zhang and Xu's work is aimed at larger scale scene capture by also using the work of [27] for efficient TSDF representation. The researchers acknowledge that their work neglects colour information which hinders CPE, and

[1] https://www.youtube.com/watch?v=i1eZekcc_lM

the system also suffers the same limitations of DynamicFusion. Despite this, MixedFusion remains a successful dynamic reconstruction system.

An important characteristic to note about this technique is that it does *not* remove dynamic objects, but rather seeks to model them, too. Although this project will be focusing on the removal of dynamic objects (instead reconstructing the static scene free of dynamics), these works are not rendered irrelevant; MixedFusion [14] in particular has relevance which will be discussed in the next section.

Joint CPE and Dynamic Segmentation

Instead of sidestepping the chicken-and-egg problem, an alternative approach is to perform CPE and segmentation together.

A critical observation by Izadi et al. [1, 2] is that an inability to find correspondences between points during ICP suggests scene motion. [32] leverages this observation in a system which attempts to identify dynamic objects. Having estimated a frame's pose using the hierarchical ICP method proposed by [1, 2], the system undergoes a "data association" step, during which frame points are either classified as *stable* and the model updated, or *unstable* and added to the model as new points. Unstable points are promoted to stable if subsequent registrations increase the confidence score beyond some threshold, or removed from the model if they remain unstable for too long. Keller et al. [32] extend this classification method to segment entire dynamic objects (not just the moving parts identified as unstable) from the scene. Using points for which ICP fails to find a correspondence as a starting point, the system employs a *hierarchical region growing* method to create a *dynamics map*, denoting the static and dynamic parts of the frame. The system's main limitations are its reliance on KinectFusion-style hierarchical ICP (which has already been discussed as inadequate for sustained dynamics), and the lack of any method to combat sensor drift (i.e., loop-closures).

In addition to applying DynamicFusion's [31] work for motion capture, MixedFusion [14] also proposes a Sigmoid-based ICP algorithm (S-ICP), capable of segmenting dynamic objects from the input sequence and performing CPE on the static parts of the scene only. Where traditional ICP aims to minimize the distance between corresponding points, S-ICP instead seeks to minimize the number of correspondences that do not fit well (according to some threshold). Having segmented the dynamic parts of the scene with S-ICP, MixedFusion will perform static model updates and dynamic motion estimation (as described above) independently, before recombining the static and dynamic parts. Minimising the number of unfitted points works well when dynamic objects occupy only a small part of the frame, but as dynamic objects grow larger this approach can misclassify static parts of a scene as dynamic in trying to minimise.

Scona et al.'s StaticFusion [33] clusters the RGB-D image into C clusters using K-Means, and seeks to assign each cluster a confidence score $B \in [0, 1]$ for each cluster. The camera pose is estimated by trying to minimise the geometric and photometric reprojection errors (or residuals) for each cluter between two consecutive frames, with the residuals being weighted by the confidence scores, so that only clusters associated with static parts of the scene have strong influence on the pose estimation. StaticFusion has a very competitive sample rate, able to process frames at 33Hz running on an Intel Core i7-3770 CPU @ 3.40GHz and an Nvidia GeForce GTX 1070 GPU.

Mask-Based CPE

Another popular approach for dealing with dynamic environments is to use a binary mask denoting the dynamic parts of each frame which is used to decouple the static background from any dynamic objects in the foreground. Much like the DynamicFusion [31] approach, this sidesteps the chicken-and-egg problem by allowing static reconstruction to be performed on the identified static parts of the RGB-D frame. A mask is conceptually simple, but its creation can introduce a computational overhead, and inaccurate masks can lead to "ghosting" artefacts in the final reconstruction, or tracking failure which corrupts the geometry of the model (see Figure 2.4).

Deep learning has been used to create masks for dynamic object removal in RGB-D reconstruction [34]. Zhang et al. propose a system which uses the OpenPose deep neural network [35] to estimate the pose of human subjects within a frame. Detected humans are removed from the frame, which is fed into ElasticFusion [6, 7] for reconstruction. This approach is successful (outperforming ElasticFusion by a significant margin), but it is limited to human subjects, constraining the scenes which it can reliably reconstruct (it would, for example, fail to reconstruct many of the tracking sequences in the Bonn dataset [13]).

ReFusion [13] creates masks using a purely geometric approach, removing the need for semantic object recognition. The paper also presents a dynamic RGB-D dataset (the Bonn dataset), which this project will use to evaluate its own results. Palazzolo et al. employ a volumetric model representation (using a TSDF) and directly use this for CPE, reasoning that the TSDF itself can be used as an error metric to minimize and find the transformation at each frame which produces a trajectory. ReFusion calculates an initial pose estimate with this technique, and uses geometric residuals to seed a mask. The mask undergoes a refinement process, before a second pass of CPE is performed using the refined mask to remove dynamic objects. Once the camera pose has been re-estimated, the scan is integrated into the voxel model, with the mask applied. Such a technique increases the runtime of the reconstruction considerably due to the second pass required. While this approach is quite successful, Figure 2.4 shows two failure cases.

ReFusion also published its source code on GitHub[2], which this project uses as both a baseline and a foundation for its own work.

Zhang et al.'s FlowFusion [36] is also object agnostic, using optical flow to identify dynamic objects based on the work of [37]. [37] proposes a method to learn a network which can estimate camera motion and scene flow [38] using 2D optical flow and an inferred rigidity mask. FlowFusion [36] segments an RGB-D frame into N clusters and uses an approach inspired by [37] to detect dynamic clusters, proposing an iterative pipeline which uses projected scene flow to converge towards a stable dynamic mask. The system, like ReFusion [13], can extract many types of dynamic objects (unlike PoseFusion [34]), but struggles in situations that challenge optical flow: very fast motion, or slight motion.

This project this dissertation presents is heavily motivated by the work of Zhang et al. in [36], the mathematical foundation described in [37], and to a lesser extent, the work of Palazzolo et al. in ReFusion [13].

[2] https://github.com/PRBonn/refusion

Figure 2.4: Two sequences from the Bonn dataset, reconstructed with ReFusion [13]. On the left: ghosting observed. On the right: model corruption as a result of tracking failure.

2.2.2 Dealing With Occlusions

Perfect camera tracking and dynamic object detection would allow the removal of dynamic objects from inputted sequences. While this solves many of the problems which arise from dynamic scenes, a concern with this approach is what to do with the void left by the removed objects. To avoid holes in the resultant mesh, researchers have developed systems capable of filling in holes in the RGB-D frames.

In-painting

Section 2.1.2 has already introduced the idea of depth map hole filling. Both systems outlined there [19, 20], despite targeting static scenes, offer potential for the completion of occluded surfaces. Another approach (this time aimed at dynamic scenes) is Trombley et al.'s Dynamic-GAN [39]. The paper presents a deep-learning framework for converting dynamic RGB-D frames to static ones, employing a generative adversarial network to detect dynamics and in-paint the voids. The results show that the system is able to outperform other state-of-the-art systems in many of the Bonn dynamic dataset's scenes [13], and therefore suggest a compelling direction for future work.

Although these systems deliver impressive results, the generative nature of the approach introduces a significant computational overhead, with the researchers of Dynamic-GAN [39] acknowledging that future work will need to focus on improving both the speed and memory efficiency of the system. Moreover, it is not clear that this generative step is needed, as other frames in the input sequence can fuse the missing data into the model, if a reliable camera pose can be estimated. This dissertation will attempt to show that such a generative step is unnecessary through the formulation of an optical-flow-based masking approach to dynamic scene reconstruction based on the work of [36], [37], and [13]

Chapter 3

Methodology

As mentioned in Section 2.2, this project is founded upon the system developed for ReFusion [13]. The source code for ReFusion[1] has been heavily modified for this research, altering Palazzolo et al.'s core reconstruction pipeline (specifically, the `Tracker()` class has been overridden) to yield a system encompassing three distinct reconstruction strategies: static, "traditional" ReFusion, and the novel optical flow based approach which is the subject of this dissertation. By building on an existing system, this project capitalizes on the already established reconstruction pipeline described in the paper [13], which includes the GPU-based implementation of CPE and the use of dynamic voxel allocation based on [27], allowing attention to be focused on the novel aspects of the project. Furthermore, this approach facilitates a comparative evaluation against the state-of-the-art, both because ReFusion is a subset of the implemented system, and because the system is compatible with the data format used by the TUM RGB-D benchmark [40] (a widely used benchmark seen in [11, 33, 34, 39]) and the Bonn dataset [13], introduced by Palazzolo et al.

The overarching concept of the proposed project involves estimating scene motion by analyzing the optical flow field between two frames, F_{i-1} and F_i, and compensating for perceived motion induced by camera movement by using the estimated egomotion. The resultant vector field (referred to as projected scene flow by [37]) describes the motion of moving objects in the scene from the perspective of F_i. Selecting a threshold on the magnitude of each pixel's scene flow vector yields a binary mask, which undergoes morphological opening and a depth-constrained flood fill for refinement. The refined mask is then used to re-estimate the camera pose (sans-dynamic objects) and integrate the scan into the model. Figure 3.1 illustrates the key steps in this pipeline.

It is important to acknowledge the influence of Zhang et al.'s FlowFusion [36] on this work, which similarly builds on the mathematical foundation of [37]. The core pipeline depicted in Figure 3.1 shares some similarities with FlowFusion [36], with the main novelty of this project being the removal of the iterative refinement observed in [36]. The system presented in this project only requires a single pass through this pipeline.

The rest of this chapter will explore the mathematical background underpinning this pipeline (Section 3.1), followed by the implementation details of the system (Section 3.2).

[1] `https://github.com/PRBonn/refusion`

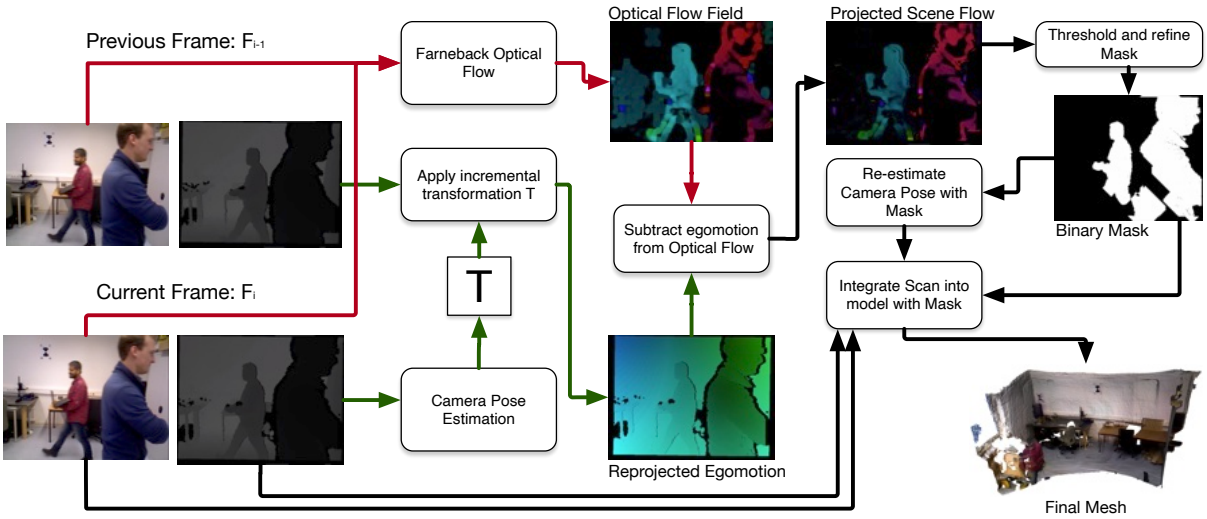


Figure 3.1: The core pipeline employed by this project for each frame. The RGB parts of F_{i-1} and F_i are used to create an optical flow field (red arrows). Meanwhile, the transformation between the two frames, T, is estimated using the depth image of F_i. T is subsequently utilized to warp the depth image of F_{i-1}, yielding a flow field representing the camera induced motion of each pixel (green arrows). Subtracting the two flow fields yields another vector field representing the motion of the scene (the projected scene flow), which is used to seed a mask. Finally, pose is re-estimated using the mask and the scan is integrated into the volume.

3.1 Mathematical Background

The central problem solved by this dissertation is concerned with the inference of a mask denoting the static / dynamic part of each frame in a video sequence. As described in Section 2.2, the main challenge which arises from this is that of disambiguating perceived motion as a result of camera movement from perceived motion as a result of scene movement. Unlike methods relying on per-point confidence scores [32, 33], deep learning [34, 39] or non-rigid warping [14, 31], the project proposed in this paper uses optical flow to detect dynamics.

3.1.1 Deriving the Projected Scene Flow

Let F_t be an RGB-D frame, P_t be the camera pose, W_t be the set of all points $\in \mathbb{R}^3$ observed in F_t and $w_t \in \mathbb{R}^3$ be the coordinates of a unique point in world space at time t. From P_t, the projective function $P_t(w_t) = u_t : \mathbb{R}^3 \mapsto \mathbb{R}^2$ projects a point in world space onto the camera plane F_t, and $P_t^{-1}(u_t) = c_t : \mathbb{R}^2 \mapsto \mathbb{R}^3$ projects the 2D point $u_t \in F_t$ into the camera coordinate space, as defined by Equation 2.1.

As described by [37], $\delta w_{t-1 \to t}$ denotes the 3D motion vector of w from time $t-1$ to t - this is also referred to as scene flow [38]. Critical to creating the dynamic segmentation mask is the projected scene flow field, Θ_t, the set containing $\delta w_{t-1 \to t}$ projected onto the image plane F_t for all $w_t \in W_t$. Θ_t is defined as:

$$\Theta_t = \Omega_t \ominus \Phi_t \tag{3.1}$$

where Ω_t is the optical flow field, Φ_t is the egomotion flow field, and $\ominus$ represents element-wise

subtraction performed between the two sets.

Optical Flow Field

Optical flow is a technique used in computer vision to estimate the motion of a pixel between two consecutive frames. It is applied in this project for the detection of dynamics in a scene.

Given a frame F_i observed at time $i > 0$, the 2D optical flow of $\delta u_{i-1 \to i}$, where $u_i = P_i(w_i)$ can be expressed as:

$$\omega(w_i) = \begin{bmatrix} \delta x \\ \delta y \end{bmatrix} = P_i(w_i) - P_{i-1}(w_{i-1}) \tag{3.2}$$

and the optical flow field Ω_i is defined as:

$$\Omega_i = \{\omega(w_i) \, \forall w_i \in W_i\} \tag{3.3}$$

In the absence of camera motion, P_i is equivalent to P_{i-1} and Equation 3.2 yields the projected scene flow necessary for the dynamic segmentation mask, making Equation 3.3 equivalent to Θ_i, in a special case of Equation 3.1 where all members of Φ_i are $[0 \; 0]^T$. However, the proposed system is aimed at dynamic scenes captured with a moving camera, tightly coupling the camera and scene motion observed in each frame. Camera pose estimation is used to decouple these motions by creating the egomotion flow field, Φ_i.

Egomotion Flow Field

The egomotion flow for a single point $u_i = P_i(w_i)$ is defined as:

$$\phi(u_i) = \begin{bmatrix} \delta x' \\ \delta y' \end{bmatrix} = P_i(\mathbf{T}_i^{-1} P_{i-1}^{-1}(u_{i-1})) - u_{i-1} \tag{3.4}$$

The above equation explains the warping process (presented in [36]) of projecting a 2D point $u_{i-1} \in F_{i-1}$ into the camera coordinate space, transforming it so that it is viewed from the perspective of P_i, reprojecting it onto the camera plane F_i and finally subtracting the image coordinates of the original point u_{i-1}. The resultant 2D vector encapsulates the camera induced perceived motion of the point $w_{i-1 \to i}$.

Section 2.2 described ReFusion's [13] use of the TSDF as an error metric for CPE. The pipeline employed by Palazzolo et al. (which is used by this project) estimates the camera pose P_i from the depth component of the F_i, but not the incremental transformation matrix $\mathbf{T}_i$. That is derived from the camera pose matrices P_{i-1} and P_i:

$$\mathbf{T}_i = P_{i-1}^{-1} P_i \tag{3.5}$$

In order to simulate the camera motion, $\mathbf{T}_i$ is inverted in Equation 3.4 to transform the point $c_i = P_{i-1}^{-1}(u_{i-1})$ relative to a fixed virtual camera.

Finally, the egomotion flow field Φ_i is found by applying Equation 3.4 to all pixels in frame F_i.

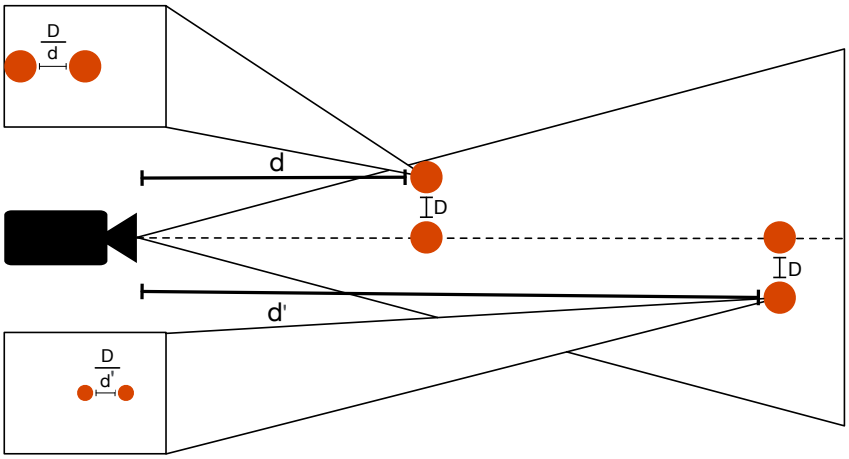


Figure 3.2: Illustration of the inverse relationship between an object's perceived size and its distance from the camera. The implication of this is that moving objects further away from the camera will have less perceived motion on the image plane, so the motion of objects *should* be compensated for by a factor of their distance from the camera, thus it is a conscious decision that this project does not do so.

$$\Phi_i = \{\phi(u_i) \, \forall u_i \in F_i\} \tag{3.6}$$

In a static environment, $u_i = P_i(w_i)$ and $u_{i-1} = P_{i-1}(w_{i-1})$ have the property that $w_i \equiv w_{i-1}$, such that the only difference between u_i and u_{i-1} results from the extrinsic matrices of P_i and P_{i-1}, i.e., any camera motion. Thus, Equation 3.4 has a symmetry with Equation 3.2 where a static camera yields $\Theta_i = \Omega_i$ and a static environment yields $\Omega_i \equiv \Phi_i \implies \Theta = \{[0\ 0]^T\}$. In other words, static points in the scene result in an egomotion flow Φ_i that is equivalent to the optical flow field Ω_i and the resultant projected scene flow Θ_i will be close to 0. Conversely, for dynamic points in the scene, Φ_i will not accurately represent the scene motion $\delta w_{i-1 \to i}$ resulting in a non-zero difference between Φ_i and Ω_i.

3.1.2 Mask Thresholding

Successfully finding the Θ_i for a frame F_i facilitates the creation of the dynamic segmentation mask. The mask is denoted by M_i and element access is defined by $M_i(u_i)$ for some pixel coordinate u_i. If Θ_i perfectly described the projected scene flow, the dynamic parts of the mask would simply be those where Θ_i has a non-zero vector: $M_i(u_i) = |\Theta_i(u_i)| > 0$. However, as the constituent vector fields (Ω_i and Φ_i) are estimated, the resultant Θ_i is corrupted by noise, so it is not the case that non-zero vectors in Θ_i imply dynamic parts of the scene. This problem is solved by thresholding a function of the magnitude of Θ_i and the pixel's depth:

$$M_i(u_i) = \frac{|\Theta_i(u_i)|}{d(u_i)} > \tau \tag{3.7}$$

where $d(u_i)$ is the depth of pixel u_i in frame F_i and τ is a threshold value. Empirical testing carried out during the development of this project found that $\tau = 3.5$ yielded compelling results on the Bonn [13] and TUM [40] data sets.

It should be noted that, despite the observation depicted in Figure 3.2, Equation 3.7 does not

Figure 3.3: Illustration of the stages of mask generation. From left to right: post-thresholding, post-opening and post-flood-fill (final mask).

compensate for pixel depth, as would have been the case if $d(u_i)$ were used as a multiplier. Instead, the depth of the pixels is inversely weighted, leading to a preference for foreground objects to be deemed dynamic. This approach is necessitated by the presence of noise in Θ_i, which results in non-zero projected scene flow vectors being assigned to static background pixels. It is a strategy that favours false negatives over false positives, which is desirable in this context as the mask refined further in the next section.

3.1.3 Mask Refinement

Thresholding Θ_i yields a binary mask where only the most dynamic parts of the scene are labelled as dynamic. Figure 3.3 shows the mask at various stages of refinement. Some noise is still present in the mask, so it is removed by applying a morphological opening operation $M_i = (M_i \ominus S) \oplus S$, with the structuring element S. The resultant mask represents the structure of the desired mask but is not necessarily contiguous. This is solved by applying a flood-fill algorithm (expounded in Algorithm 1) to the mask, using the already known dynamic pixels as seeds. The system employs a standard flood-fill algorithm, extended by two novel constraints: the algorithm performs a breadth-first search that grows the dynamic region of the mask by one pixel at a time where the depth of the pixel is within a threshold D of the seed pixel's depth, until the growth count c exceeds a threshold C. The depth threshold segments objects that are at different depths, and the growth count threshold prevents spurious filling due to a false positive seed pixel. The implemented system uses $D = 0.2$ (20cm) and $C = 75$.

3.2 Implementation

The novel system developed for this dissertation is published on GitHub: `https://github.com/connorkeevill/dynamic-3D-reconstruction`

The software presented by this dissertation is written in C++ [41] and CUDA [42], and is built on top of the publicly available source code for ReFusion[2] [13]. Such language choices are inherited from the ReFusion project, but are well-founded and similar choices would have been taken in the absence of a pre-existing codebase. C++ is very performant, facilitates low-level memory management, and enables direct use of OpenCV [43]; CUDA facilitates GPU acceleration, which is essential for real-time performance.

This section moves from the purely mathematical space of the previous section to explain how the underlying maths are efficiently implemented in code.

[2] `https://github.com/PRBonn/refusion`

Algorithm 1 Flood-fill algorithm

Require: M_i is a binary mask
  Queue $Q = \{ (u_i,\ 0)\ \forall u_i \in M_i\ \wedge\ M_i(u_i) = 1 \}$ ▷ Each element is a tple of coordinates, ▷ and a growth count
  while $Q \neq \emptyset$ **do**
    $(u_i,\ c) \leftarrow Q.dequeue()$
    if $c > C$ **then** continue
    end if
    for neighbour n_i around u_i **do**
      if $|d(n_i) - d(u_i)| < D$ and $M_i(n_i) = 0$ **then**
        $M_i(n_i) \leftarrow 1$
        $Q.enqueue((n_i,\ c + 1))$
      end if
    end for
  end while

3.2.1 High-level Code Structure

The approach for developing this project was to first simplify the ReFusion codebase to yield a system which can handle exclusively *static* scenes, and then to build on this to implement the novel system. As a result, ReFusion's [13] influence on the code structure is evident throughout the project, so in understanding how this system works, it is useful to understand the architecture of ReFusion. Palazzolo et al.'s system is built around a core loop of receiving a frame from its source, simultaneously estimating camera pose while generating the mask, and integrating the frame into the model. It uses three main classes to achieve this: `FrParser()`, `Tracker()` and `TsdfVolume()`. In each iteration of the core loop, the next frame in the sequence is read from FrParser() and passed into the `Tracker::AddScan()` method, which performs CPE and integrates the scan into the `Tracker()`'s internal `TsdfVolume()`. Once all frames from `FrParser()` have been exhausted, a mesh is extracted from the `TsdfVolume()` using the `Tracker::ExtractMesh()` method. Figure 3.4 depicts this process.

The work of this project is mostly concerned with the `Tracker()` class. The `Tracker()` class has been turned into an abstract base class, making way for three derived classes: `StaticTracker()` (removing the code pertaining to mask generation), `ReTracker()` (performing ReFusion [13] reconstruction as already described) and `OpticalFlowTracker()` (implementing mask creation as described in Section 3.1). Each class is characterised by the behaviour overridden in the `Tracker::AddScan()` method.

Other, more superficial, changes have been made to the codebase:

- the addition of a `Logger()` class to facilitate system-wide logging profiles;
- a new `Timer()` class for profiling the performance of the program between checkpoints;

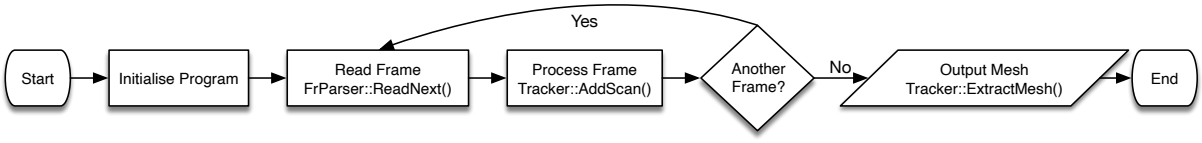


Figure 3.4: Flow of execution through the architecture of ReFusion [13].

- a new settings mechanism in which runtime configuration is read from a .toml file;
- a `TUMVideo()` class replacing the `FrParser()` in an implementation which loads all frames into memory instead of streaming them from the disk, so that performance is more representative of a system where frames are captured in real time from a sensor.

3.2.2 Calculating the Mask

The pipeline shown in Figure 3.1 depicts the multistage process involved in generating the mask M_i. Ideally, the pipeline would be implemented in a monolithic CUDA kernel keeping data on the GPU to reduce memory transfers between the host and GPU. However, due to incompatibility between some function interfaces, the envisioned monolith is fractured into four main stages within the `OpticalFlowTracker::calculateMask()` method, with data being transferred between the host and GPU between each stage:

1. optical flow - GPU;
2. camera pose estimation - GPU;
3. mask thresholding - GPU;
4. mask refinement - CPU.

The following sections will describe the implementation of each of these stages in detail.

Optical Flow

There are many approaches for estimating optical flow. This project is agnostic to the specific algorithm employed and tries to be robust to the style of the resultant optical flow field, but it is important to note that *dense* optical flow is expected (thus sparse methods like [44] are excluded). This project uses the OpenCV [43] GPU implementation of the Gunnar-Farnebäck optical flow algorithm [45], primarily because it is fast. The main deficiency of this algorithm is that optical flow is only detected on the edges of objects (as observed in Figure 3.5) due to the aperture problem. This *could* be mitigated by employing an algorithm which is able to fill in the missing flow information in the inner regions of homogenous objects, such as Horn-Schunck [46]. Ultimately this was deemed unnecessary as Horn-Schunck is more sensitive to noise, and the problem of missing flow is easily solved by the mask refinement stage described below. Another interesting avenue for future work would be to experiment with the use of more modern convolutional based approaches such as [47] and [48].

The optical flow field Ω_i is calculated in the `OpticalFlowTracker::GPUOpticalFlow()` method. This method begins by allocating GPU memory in form of two `cv::cuda::GpuMat` objects, representing frames F_{i-1} and F_i. The RGB frames are copied from the host to GPU, converted to greyscale and fed into the `FarnebackOpticalFlow::calc()` method. The resultant flow field is a `cv::cuda::GpuMat` object, which is copied from GPU to host to a GPU allocated `float *` for further processing. The copying back and forth between host and GPU is the result of no suitable interface for directly accessing elements of a `cv::cuda::GpuMat`. Therefore, in order to arrange the Ω_i into a contiguous block of GPU memory (as required by later stages of the pipeline), it must be copied to and from the host.

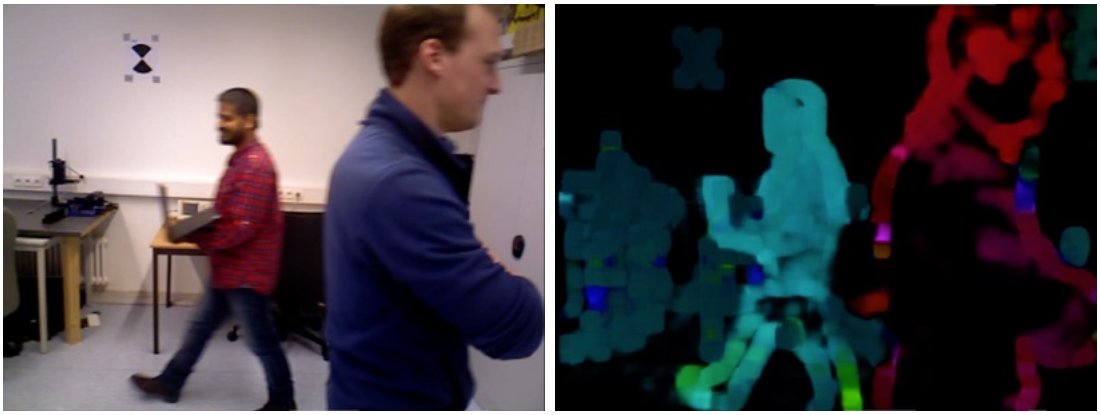


Figure 3.5: A frame F_i and its corresponding optical flow field, Ω_i, with each vector visualised as a point in HSV colour space. Observed in Ω_i are the *motion boundaries* between the static background and the moving foreground, with internal regions of dynamic objects failing to display any motion. This is the result of the *aperture problem*.

Camera Pose Estimation

The CPE stage is the least changed from the original ReFusion [13] implementation. The notable difference between `OpticalFlowTracker::TrackCamera(depth, mask)` and `ReTracker::TrackCamera(depth, mask)` is that the former uses the mask to ignore pixels in the depth frame, whereas the latter uses geometric residuals to ignore pixels and *write to* the mask.

Mask Thresholding

Although Figure 3.1 depicts multiple steps between the warping of the depth frame of F_{i-1}, and thresholding the scene flow to get the mask, the GPU-based implementation finds increased efficiency by consolidating these steps into a single stage (which, while split across by two CUDA kernels, is a single stage from the perspective of data movement between the host and GPU).

Equation 3.6 is optimised by the insight identified in Figure 3.5. Due to the aperture problem, the optical flow field Ω_i is unable to capture the motion of internal regions of dynamic objects. The system avoids introducing erroneous motion into the projected scene flow field Θ_i (which could arise if egomotion flow were subtracted from optical flow in these regions) by applying Equation 3.4 only to pixels u_i where $|\Omega_i(u_i)| > \tau$. The result of Equation 3.4 is subsequently subtracted from the optical flow field vector $\Omega_i(u_i)$ in place (i.e., no intermediary egomotion flow field is stored) to produce the projected scene flow field Θ_i in a single step. Parallelism is achieved by performing this to each pixel on a GPU thread.

Mask Refinement

Algorithm 1 does not parallelise well on a SIMD architecture; there is too much branching to efficiently utilise vectorisation, leading to many inhibited processors. Therefore, the algorithm is implemented sequentially on the CPU. Perhaps this algorithm could be parallelised across multiple (CPU) threads, but it is unclear whether this would yield significant speedup due to the overhead of thread creation and synchronisation. This is left as an area for future work.

Because the flood-fill algorithm necessitates CPU execution, the preceding morphological erosion

and dilation are (also) performed on CPU using the OpenCV [43] functions `cv::erode()` and `cv::dilate()`.

3.2.3 Performance

Despite the unsuccessful efforts to implement the mask generation pipeline as a monolithic CUDA kernel, the pipeline is still sufficiently parallelised to achieve real-time performance. Running in a Docker container on the university GPU cluster (configured in a VM equipped with an Intel Xeon Silver 4110 - Skylake @ 2.1GHz-3.0GHz and an Nvidia GeForce RTX 2080), the system achieves $\sim$ 10Hz on a 640$\times$480 resolution video stream.

3.2.4 Demonstration

An accompanying video for this dissertation is available at: `https://youtu.be/zrWnW1QZjHM`

Chapter 4

Experimental Evaluation

The previous chapter motivated the use of the ReFusion [13] codebase as foundation for this project with the comparative evaluation it enables. In this chapter, such a comparative evaluation is performed, with the objective of assessing the system's ability to reconstruct dynamic environments from the Bonn [13] and TUM [40] benchmarks.

The rest of this chapter will expand on the approach taken to evaluate the system, and provide a high level overview of the performance of the novel system, before diving into a more detailed analysis of the quantitative and qualitative aspects of the system's performance.

4.1 Experimental Design

The system provides two outputs (and some optional extras depending on the run configuration) given a reconstruction sequence: a mesh of the scene and a camera trajectory. These two formats lend themselves to a qualitative and quantitative evaluation respectively. The qualitative evaluation will appraise the quality of the reconstructed meshes, highlighting sequences in which this project has outperformed ReFusion in terms of dynamic object removal or background reconstruction, and sequences which have proven particularly challenging. The quantitative evaluation will make use of the ground truth trajectories provided by the Bonn and TUM datasets [13, 40] to calculate the absolute trajectory error (ATE) and relative pose error (RPE) for each sequence (using the tools provided by the TUM dataset [40]). Results will be reported for the three reconstruction strategies outlined in Chapter 3, with static reconstruction serving as a general reconstruction baseline, and ReFusion [13] serving as a baseline for dynamic reconstruction.

4.1.1 Testing Environment

This evaluation (and much of the developmental testing) was performed on the university GPU cluster, Hex.

Hardware

Hex has multiple nodes, with varying hardware configurations. The node used for this evaluation can be seen in Figure 4.1 and has the following specifications:

- CPUs: 2 × Intel Xeon Silver 4110 - Skylake - 8 cores (hyperthreaded) @ 2.1GHz - 3.0GHz
- GPUs: 6 × Nvidia RTX 2080, 8GB RAM, 2944 CUDA cores
- RAM: 128GB DDR4 @ 2666MHz
- OS: Ubuntu 20.04.4 LTS

Despite the seeming abundance of computational resource on offer, an individual run of the system does not make use of all available hardware, existing in a virtual machine that has access to only one GPU. It is therefore entirely feasible that this system could run on a desktop machine equipped with a similarly capable GPU.

Software

As Hex is a shared resource among members of the university, it is recommended to use Docker[1] for running code. The Dockerfile used to build the image used for this evaluation can be found in the published project GitHub repository: `https://github.com/connorkeevill/dynamic-3D-reconstruction`, and defines the following software stack:

- Ubuntu 16.04
- GCC 5.4.0
- CUDA 9.0
- Eigen 3.3.7
- OpenCV 3.3.1

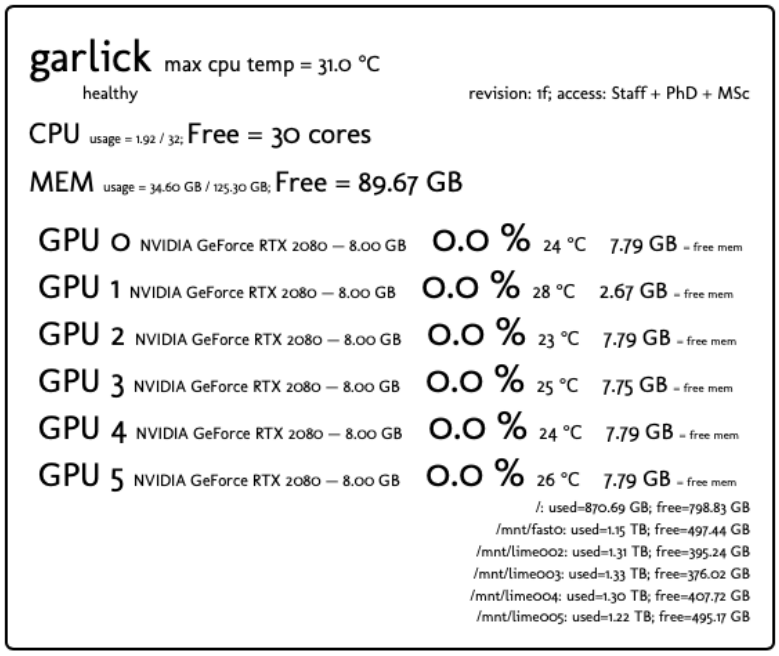


Figure 4.1: The Hex node used for this evaluation.

[1] `https://www.docker.com`

The Dockerfile is the simplest way to build and run the project, as it installs all necessary dependencies and builds OpenCV [43] from source with CUDA support.

4.1.2 Experimental Procedure and Data Gathering

The results which are the focus of this evaluation are produced by running the system on the Bonn [13] and TUM [40] datasets and performing numerical comparisons against ground truth data. From the Bonn dataset, all sequences are reconstructed except `rgbd_bonn_static`, which is omitted because it is extremely long and is too large to fit in the memory of a `TUMVideo()`. This could be fixed by streaming the frames from the disk, but as this is also a static sequence, is was deemed unnecessary. From the TUM dataset, all `freigburg3` sequences are used except for the two `halfsphere` sequences. These are omitted because they contain camera roll, which the novel system is not equipped to handle, so their inclusion would distort the results.

TSDF Options	
voxel_size	0.01
num_buckets	50000
bucket_size	10
num_blocks	500000
block_size	8
max_sdf_weight	64
truncation_distance	0.1
max_sensor_depth	5
min_sensor_depth	0.1

Table 4.1: Default TSDF configuration options as used by ReFusion [13].

Tracking Options	
max_iterations_per_level_0	6
max_iterations_per_level_1	3
max_iterations_per_level_2	2
downsample_0	4
downsample_1	2
downsample_2	1
min_increment	0.0001
huber_constant	0.02
regularization	0.002
reconstruction_strategy	[strategy]
output_mask_video	false
output_flow_video	false

Table 4.2: Default tracker options as used by ReFusion [13], with the additional selection of reconstruction strategy and output options.

Camera Intrinsics	
cx	319.5
cy	239.5
fx	525.0
fy	525.0
rows	480
cols	640
depth_factor	5000

Table 4.3: Camera Intrinsic parameters for the Microsoft Kinect [4], as described by [40].

Logging Profile	
verbose	false
debug	false
writeLogsToFile	false
filepath	""

Table 4.4: Logging profile used for the evaluation.

Generic Output	
outputMesh	true
outputResults	true
outputTimings	true
outputReprojectedVideo	false

Table 4.5: Generic settings for ouptut. For evaluation, the mesh, result (i.e. trajectory) and timings were all output.

Running the System

The reconstruction system has been run on the dataset three times - once for each reconstruction strategy - with the configurations depicted in tables 4.1, 4.2, 4.3, 4.4, and 4.5. The program is ran from the command line by providing the path to an `associated.txt` file (as described by [40]) which instructs the system where to find the RGB and depth images for the sequence. A Python script bootstraps the execution for every sequence in the dataset.

The same system is used for all three reconstruction strategies, and different strategies are selected by mounting a different `config.toml` file into the Docker container. The three strategies will be referred to with shorthand names: **S** (static), **RF** (ReFusion), and **N** (novel).

Post Processing

Meshes are output as `.obj` files, and are evaluated without any post-processing. MeshLab[2] is used to view the meshes.

The camera trajectory is output as a text file of camera poses with time stamps (corresponding to the time stamps of the frame's pose) in the format specified by the TUM dataset [40]. These are not evaluated directly, but compared against the ground truth trajectories provided by the dataset to calculate the ATE and RPE for each sequence. The ATE and RPE metrics are calculated using the tools provided by the TUM dataset [40], which have been slightly modified to write output to a `.csv` in addition to the charts they plot. The modified evaluation scripts are available in the published project GitHub repository, in the `./scripts/` directory. These data are subsequently analysed in Excel.

4.2 Evaluation

The project presented in this dissertation is a success. Most of the test sequences are reconstructed with total success; out of 32 sequences, there are 3 *total* failures, and 7 *partial* failures, 5 of which are sequences from the TUM dataset [40] (for a common reason relating to the depth maps of that benchmark). The term *total* failure is used here to describe a reconstruction which is corrupted by a tracking failure, and *partial* failure is used to describe a reconstruction which has failed to completely remove a dynamic object from the scene.

The *total* failure cases are generally sequences in which the camera is moving very quickly, resulting in blurry frames and large frame-to-frame motion. This hinders the system's ability to perform optical flow, estimate camera pose (thus the mask inference pipeline of Figure 3.1 deteriorates) and breaks the assumption of small frame-to-frame motion. *Partial* failures tend to occur as a result of a dynamic object initially being static (and thus being registered as part of the model before motion begins) or dynamic motion happening outside the camera frustum between two (distinct) visits of the same part of the scene, stopping the system from detecting the motion. ReFusion [13] does not suffer from this deficiency due to its use of geometric residuals - it does not rely purely on observed motion to detect dynamic objects.

The static reconstruction strategy (**S**) used as a baseline in the evaluation is already very competitive, illustrating Izadi et al.'s [1, 2] claim (see Section 2.1.3) that dense CPE provides robustness to transient scene motion. Therefore, after providing a summary of performance of the entire dataset, the evaluation focuses on the performance on sequences which the static

[2] `https://www.meshlab.net`

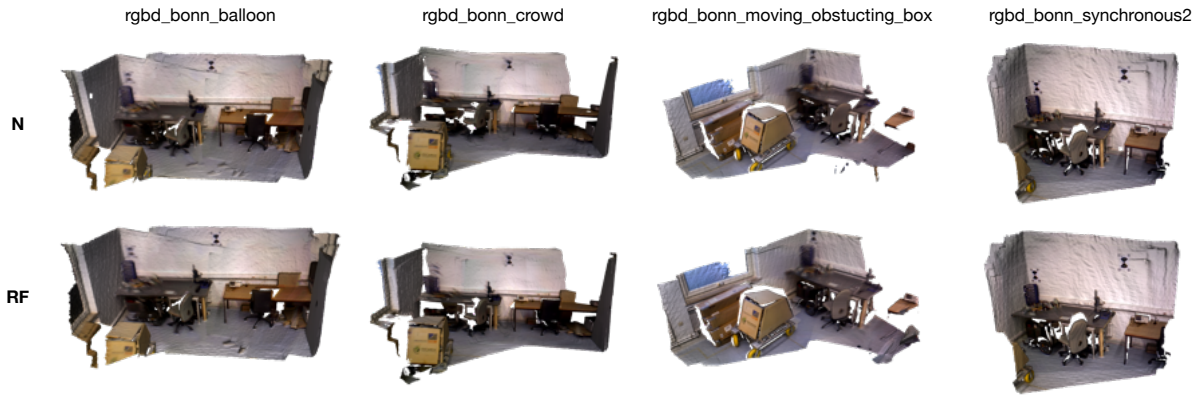


Figure 4.2: A direct comparison of the novel system and ReFusion [13] on the `rgbd_bonn_balloon`, `rgbd_bonn_crowd`, `rgbd_bonn_moving_obstructing_box`, and `rgbd_bonn_synchronous2` sequences. There *are* differences between the meshes, but they are **very** subtle.

system is unable to reconstruct well.

4.2.1 Performance Across Whole Dataset

The performance of the system can be described as *consistently good*. A quantitative evaluation of the system is performed by calculating two widely used metrics in SLAM and 3D reconstruction: the absolute trajectory error (ATE) and the relative pose error (RPE). Measures of central tendency for ATE and RPE are compared between the three reconstruction strategies, and the two metrics are also plotted graphically to show the error over time (e.g., Figure 4.3).

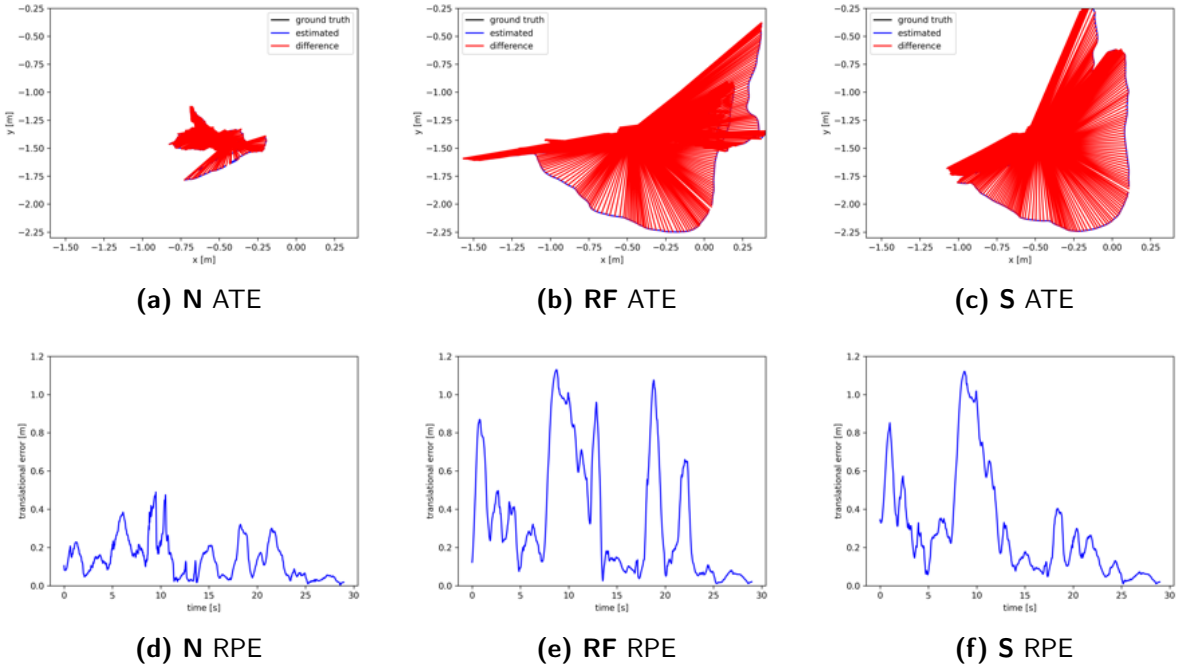
Tables A.1 and A.2 (Appendix A) show the root mean squared error (RMSE) for ATE and RPE respectively, across all reconstruction sequences and strategies. At face value, ReFusion [13] outperforms the novel system in terms of ATE and RPE in most sequences; in terms of ATE, the novel system is outperformed by ReFusion in 27 out of the 32 sequences, and in terms of RPE, the novel system is outperformed in 22 of the 32 sequences.

These data do not tell the whole story, however. Figure 4.2 and Table 4.6 show a direct comparison of the novel system and ReFusion on a small subset of the dataset, where, numerically, ReFusion outperforms the novel system. The qualitative comparison of the meshes shows that, despite the greater tracking accuracy shown in Table 4.6, the quality of the output meshes is still on par with those of ReFusion.

Moreover, a closer inspection of the data reveals that the margin by which ReFusion outperforms the novel system is very small in most sequences. On average, the difference in RMSE when ReFusion outperforms the novel system is 0.09m in ATE and 0.044m in RPE. Conversely, the average difference when the novel system outperforms ReFusion is 0.448m (+ **397.8%**) in

Sequence	N	RF	S
`rgbd_bonn_balloon`	0.1796	**0.1742**	0.1797
`rgbd_bonn_crowd`	0.1878	**0.1256**	0.173
`rgbd_bonn_moving_obstructing_box`	0.4413	0.3596	**0.2424**
`rgbd_bonn_synchronous2`	0.0264	**0.021**	0.0278

Table 4.6: A subset of Table A.1 showing the RMSE for ATE for the meshes in Figure 4.2.



(a) N ATE **(b) RF** ATE **(c) S** ATE

(d) N RPE **(e) RF** RPE **(f) S** RPE

Figure 4.3: The ATE and RPE for one of the most challenging (in terms of dynamic motion) sequences in the Bonn dataset: `rgbd_bonn_crowd2`. The novel optical flow solution outperforms both static reconstruction and ReFusion [13] by a considerable margin.

ATE and 0.071m (+ **61.4%**) in RPE[3]. The implication of this observation is that while the novel system sacrifices some accuracy, it is able to handle a much broader range of sequences than ReFusion, resulting in a more consistent baseline accuracy.

4.2.2 Performance on Challenging Sequences

There is little discussion to be had around the performance of the novel system on sequences which the static strategy also reconstructs well. This section will therefore highlight sequences where the novel system outperforms ReFusion (and the static system), or sequences which proved particularly challenging for all reconstruction strategies.

Bonn Crowd (2)

Figure 4.3 shows the ATE and RPE for one of the most challenging sequences in the Bonn dataset: `rgbd_bonn_crowd2`. The sequence consists of 895 RGB-D frames and contains a large number of people moving through the scene which the reconstruction system must detect and remove in order to produce an accurate reconstruction. The charts in Figure 4.3 depict a much more accurate trajectory estimated by the novel system when compared with both the static reconstruction baseline and ReFusion [13]. This is reflected in the comparative matrix for the sequence, shown in Figure 4.4. Notice how the mask from the novel system is much more accurate than ReFusion. The resultant reprojection therefore has dynamic objects removed, revealing the background which has already been fused into the model.

This highlights an important point which was discussed in Section 2.2.2. The **N** column in

[3]The difference here indicates the amount by which one system has outperformed the other: higher difference $\implies$ better comparative performance.

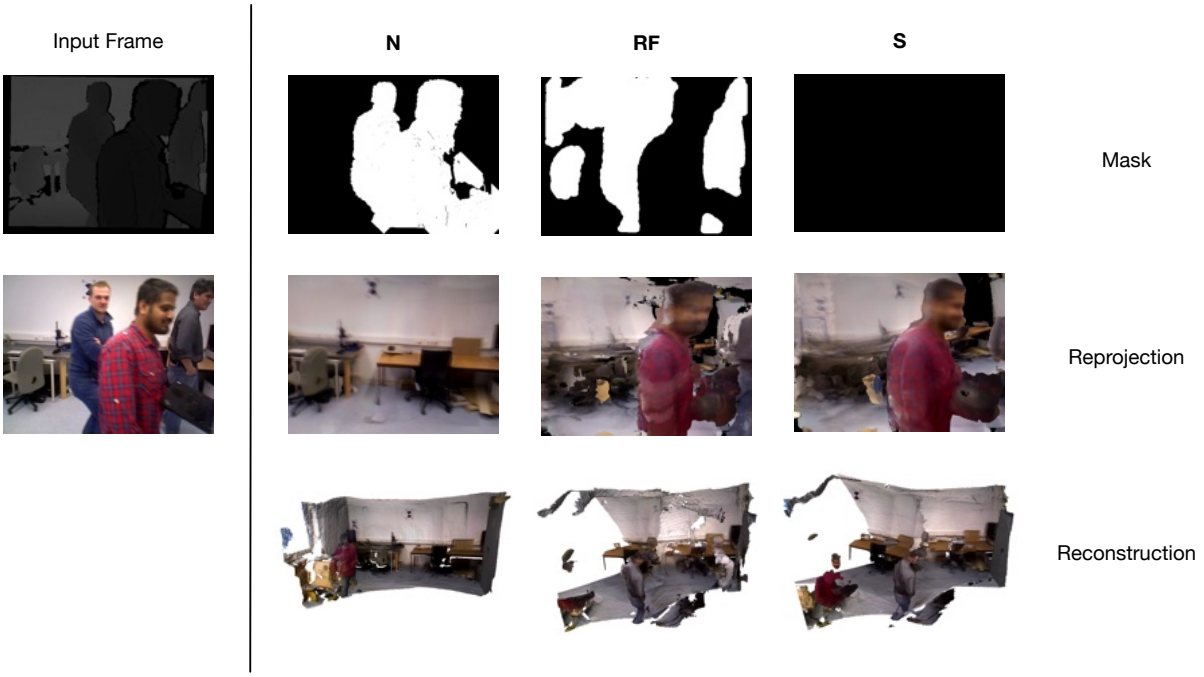


Figure 4.4: A comparative matrix of the `rgbd_bonn_crowd2` sequence, showing snapshots of the reconstruction at time=12s. To the left of the bar: the depth and RGB frames. To the right of the bar: the novel, ReFusion and static reconstruction strategies (left to right) and their mask, reprojection and output meshes (top to bottom). The reprojection (middle row) is a render of the mesh (at time=12s) from the estimated camera pose.

Figure 4.4 shows that, even with the excluded dynamic sections of the frame, the resultant void has *not* had an impact on the reconstruction. The system's ability to accurately track camera pose (shown in Figure 4.3) has enabled other frames in the sequence which contain the necessary background information to be fused into the model.

Bonn Balloon Tracking

The three reconstruction methods also struggle with the Bonn *balloon tracking* sequences: `rgbd_bonn_balloon_tracking` and `rgbd_bonn_balloon_tracking2`. These sequences show a dynamic subject throwing a balloon into the air which is left to fall onto the floor; the camera tracks the balloon. The difficulty in these sequences arises from the fast motion observed when the balloon is thrown. As depicted in Figure 4.5, the motion blur this causes results in a large spike (highlighted) in the RPE for all three reconstruction strategies. The frame shown in the figure is from the peak of the spike, and is taken just after the balloon is thrown. While none of the three reconstruction strategies have been able to successfully capture the scene's mesh, notice that **N** has the most accurate mesh (the floor is planar and ghosted cart is close to initial position) for `rgbd_bonn_balloon_tracking`, and **RF** has the most accurate mesh for `rgbd_bonn_balloon_tracking2`.

Even in the absence of the fast motion induced by tracking the balloon, the RPE plots are quite noisy for this sequence, displaying oscillations in the error. This is likely due to the ambiguous state of the subject: initially they are standing still, before transitioning to a dynamic state where the balloon is thrown. For **N**, the initial static state means that the subject is not segmented out of the frame and consequently is fused into the model for early tracking, before being removed once the dynamic state is detected.

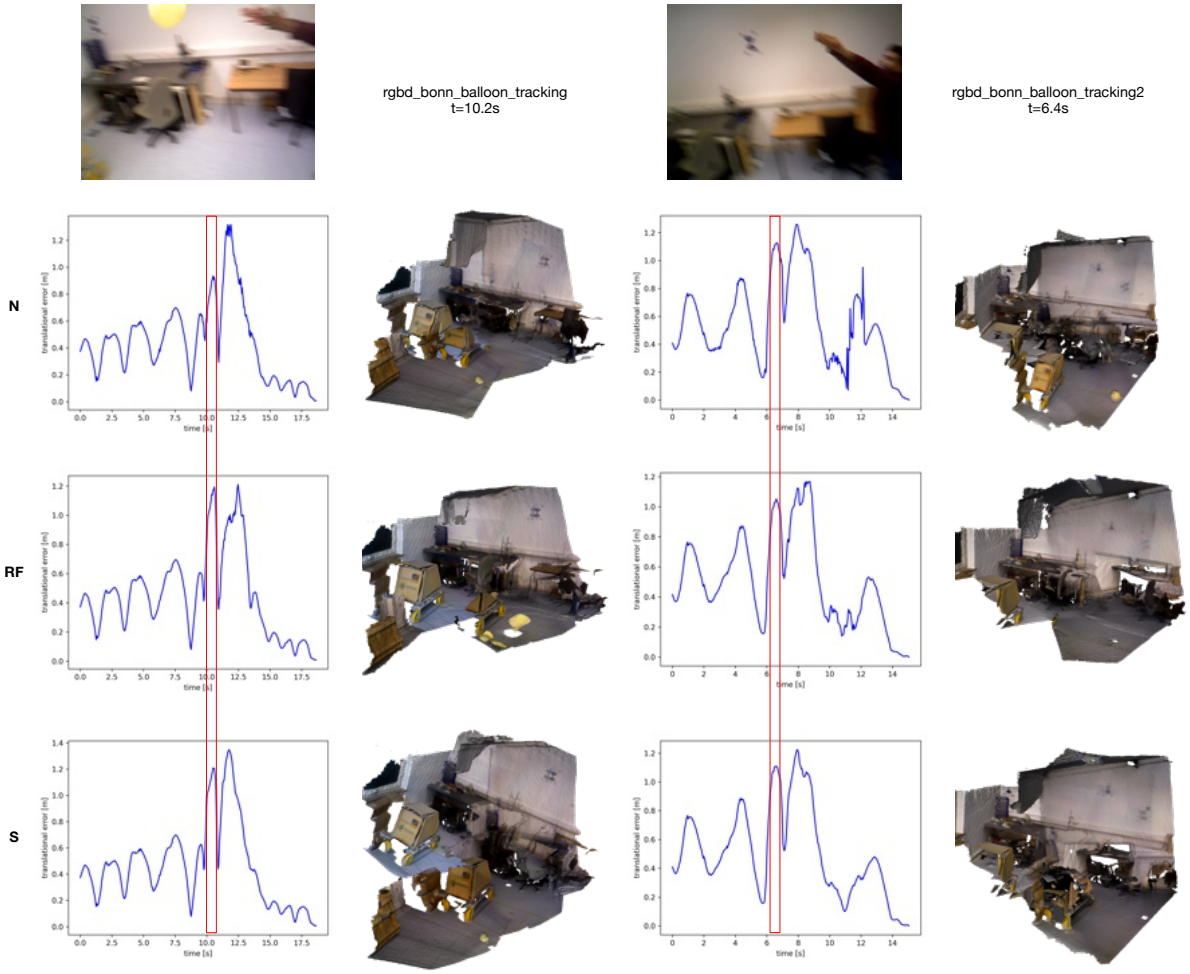
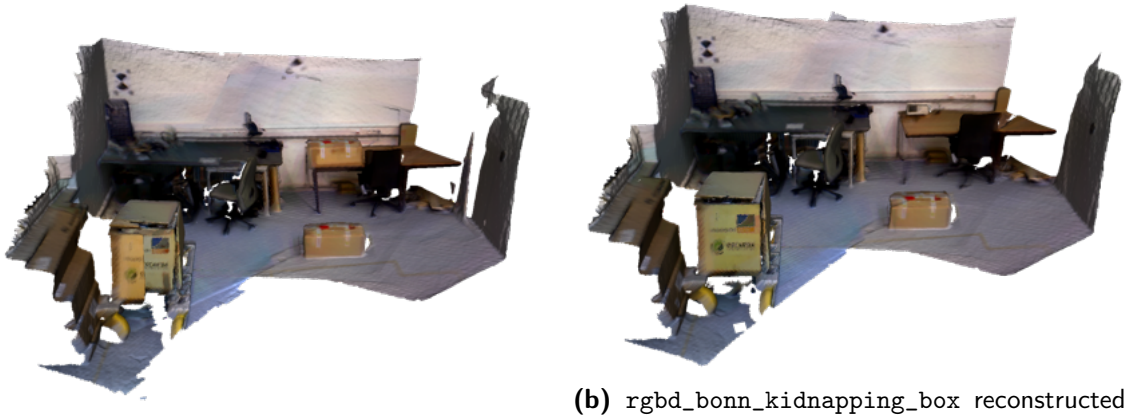


Figure 4.5: The `rgbd_bonn_balloon_tracking` (left) and `rgbd_bonn_balloon_tracking2` (right) sequences, reconstructed by all three strategies. The RGB frames shown at the top are from time=10.2s and time=6.4s respectively, and are frames from high motion parts of the sequence, corresponding to the spikes (highlighted in red) seen in their respective RPE charts.

Bonn Kidnapping Box

This weakness of the novel system (in which dynamic objects can be fused into the model before being detected as dynamic) is also seen in the `rgbd_bonn_kidnapping_box` sequence, where dynamic motion happens outside the camera's field of view between two encounters of the same region. This sequence consists of an initial sweep of the scene where a box is laid on the floor, before the camera moves away while the box is moved, and then returns to the same location. This type of *unobserved* dynamic motion is not handled well by the novel system, which relies on *observed* dynamic motion to create the mask. ReFusion's geometry-aware approach performs better in these sequences as it is able to detect the change in scene geometry between the two encounters of the box's location. Figure 4.6 exemplifies this difference in behaviour, showing the meshes produced by **N** and **RF**.

Similar behaviour is observed in the `rgbd_bonn_person_tracking` sequences, where an initially static subject is registered as part of the model before beginning to move (see Figure 4.7).



(a) `rgbd_bonn_kidnapping_box` reconstructed by **N**

(b) `rgbd_bonn_kidnapping_box` reconstructed by **RF**

Figure 4.6: Reconstructions of the `rgbd_bonn_kidnapping_box` sequence by **N** and **RF**. Notice the key difference that the mesh reconstructed by **RF** does not include the box on the table.

Figure 4.7: The `rgbd_bonn_person_tracking` sequence, reconstructed by **N**. The subject in this sequence is initially static, before becoming dynamic (walking in an arc through the room), resulting in their registration to the model.

Balloon (2)

The quantitative results fail to fully describe the quality of the generated mesh. A more accurate trajectory does generally lead to a better mesh, but the ATE and RPE do not encompass any information about the removal of dynamic objects from the mesh. `rgbd_bonn_balloon2` is a sequence which highlights this point. The sequence contains 469 RGB-D frames and shows a subject bouncing a balloon on their hand.

In terms of RPE and ATE, ReFusion performs best, followed by the baseline static system, and then the novel system (see Table 4.7 and Figure 4.8). Despite having the least accurate

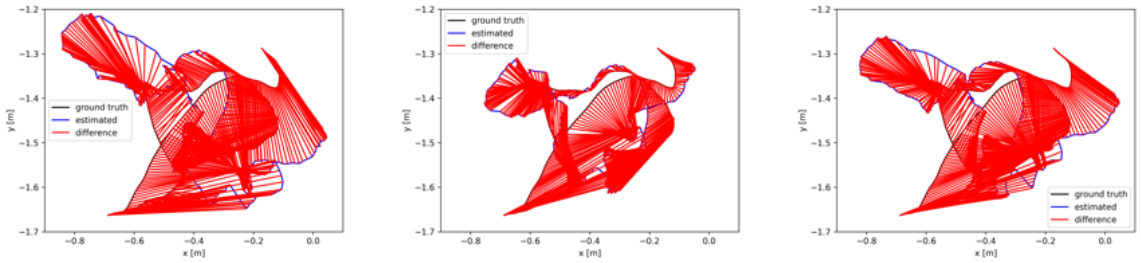
Metric	N	RF	S
ATE	0.2811	**0.189**	0.2562
RPE	0.3081	**0.2891**	0.3006

Table 4.7: RPE and ATE RMSE for the `rgbd_bonn_balloon2` sequence, reconstructed by **N**, **RF** and **S**.

(a) `rgbd_bonn_balloon2` reconstructed by **N**

(b) `rgbd_bonn_balloon2` reconstructed by **RF**

(c) `rgbd_bonn_balloon2` reconstructed by **S**



(d) `rgbd_bonn_balloon2` ATE from **N**

(e) `rgbd_bonn_balloon2` ATE from **RF**

(f) `rgbd_bonn_balloon2` ATE from **S**

Figure 4.8: Reconstructions and ATE plots of the `rgbd_bonn_kidnapping` sequence by **N**, **RF** and **S**. Notice the absence of the balloon (the yellow blur visible in **RF** and, faintly, **S**) in in the mesh of **N**, despite the less accurate trajectory.

trajectory, the novel system most successfully removes the dynamic objects from the scene, as shown in Figure 4.8. This difference in mesh accuracy can be attributed to the fact that ReFusion does not use observed motion to directly detect dynamic objects (instead looking to find the geometric and photometric residuals caused by dynamic objects). This sequence sees the balloon move quickly through a space which the camera does not return to; ReFusion fails to detect the change in geometry by not revisiting that part of the balloon's path. The novel system, however, is able to directly detect that the balloon is moving and segment it from the scene.

This does not invalidate the use of RPE and ATE as evaluation metrics; indeed, the novel system has the least accurate trajectory for this sequence. However, it is worth noting that these metrics do not fully capture the quality of the mesh. In instances where the differences in trajectory are relatively small (in this case, approximately 9cm in ATE and 2cm in RPE), the quality of the mesh can be more significant than these metrics suggest.

4.3 Limitations

Despite its success, the novel system developed for this dissertation is far from perfect. This section will identify some of the weaknesses of the system.

Section 4.2.2 has already touched on the system's inability to detect dynamics which have happened outside the camera frustum. This is an inherent problem of the approach (relying on observed motion to detect dynamics), with no simple solution.

Another weakness of the project is the absence of any technique to recover from tracking failures (i.e., loop-closures). This is a result of the project's use of ReFusion [13], which does not perform loop-closures, and a general focus of the project on the dynamics detection

problem (rather than the SLAM problem). The visual front end developed for this dissertation is not inherently tied to the ReFusion backend, and the approach could be adapted for an alternative SLAM system which does perform loop-closures. But it is important to consider the impact that loop closures would have on the mask inference pipeline (Figure 3.1). Egomotion compensation uses estimated camera pose to create the projected scene flow field, and a loop-closure would propagate a pose correction through the loop, which could necessitate a re-computation of the masks.

Motion blur presents another problem for the system. Hindering the system's ability to calculate optical flow and estimate camera pose, sustained motion blur can lead to tracking failure (see Figure 4.5). The solution to this would involve a more robust optical flow algorithm and camera pose estimation approach, both of which present large challenges as research problems in their own right.

Further, there is the system's performance on the TUM RGB-D dataset [40], which is consistently worse than on the Bonn dataset (Figure 4.9 shows some samples). This can be attributed to the sequences containing invalid depth measurements, which the developed system has not been designed to handle. This is not a complex issue to solve; one could employ the work of Berdinkov and Vatolin [19] to detect and interpolate invalid depth measurements as a preprocessing step.

(a) `fr3/sitting_rpy` reconstructed by **N** **(b)** `fr3/walking_xyz` reconstructed by **N**

Figure 4.9: Reconstructions from the TUM benchmark, demonstrating the poor performance due to the invalid depth measurements present in that dataset.

Chapter 5

Conclusion

This dissertation has presented a novel approach to the problem of reconstructing dynamic environments, using what has been termed *egomotion compensated optical flow* to detect and segment dynamic objects. The mathematical foundation for this approach was introduced, along with a detailed description of the system's implementation. The supporting code for this dissertation is published on GitHub: `https://github.com/connorkeevill/dynamic-3D-reconstruction`. The video demonstration of the system is available on YouTube: `https://youtu.be/zrWnW1QZjHM`

The paper performs a detailed evaluation of the system, comparing it to the state-of-the-art [13] in dynamic scene reconstruction, finding that the novel system performs as well as, or better than, the state-of-the-art in most cases. The evaluation examines the performance of the system on the Bonn [13] and TUM [40] benchmarks, and evaluates the quantitative performance of the system by computing the absolute trajectory error (ATE) and relative pose error (RPE) of the estimated trajectories, and correlates these numerical results with the visual quality of the reconstructed meshes. Many of the system's strengths and weaknesses are identified and discussed. The proposed system is shown to be capable of reconstructing dynamic environments containing multiple moving objects in real time, satisfying the initial goals of the dissertation.

5.1 Future Work

The work presented leaves many avenues for future research, some of which have already been identified in the paper. One such avenue is that of further GPU acceleration. Section 3.2 describes the failed efforts to implement Figure 2.1's pipeline as a monolithic CUDA kernel. A compelling direction for future work could see this pipeline more tightly integrated with the GPU, perhaps by reimplementing some of the OpenCV [43] functions for the GPU to eliminate the interface incompatibilities which are the source of the performance bottleneck, or by implementing the flood-fill of Algorithm 1 in CUDA. Such an implementation would reduce the movement of data between the CPU and GPU, and would likely result in a significant performance improvement.

Another avenue to explore in further research is an investigation into different optical flow algorithms. The optical flow algorithm used in this dissertation is the Farnebäck algorithm [45], motivated mainly by its speed. However, as more modern optical flow techniques based on

deep learning [47, 48] are beginning to outperform traditional approaches in both performance and speed, it would be interesting to see how the proposed system compares when using these more modern algorithms.

The main area of difficulty encountered by the proposed system was in the sequences from the TUM benchmark [40], due to the invalid depth measurements in many of the sequences. Future work could see this deficiency remedied through the implementation of a depth map inpainting algorithm, like Berdinkov and Vatolin's work in [19], or Trombley et al.'s Dynamic-GAN [39], or perhaps the use of a bilateral filter [16] to smooth the depth map.

Finally, the proposed system could be extended to perform trajectory refinement through the implementation of loop-closures. This would require significant rearchitecting of the core pipeline, as the current implementation does not retain any information about the camera poses, so cannot perform loop-closure detection. Moreover, changes to the estimated trajectory would alter the egomotion compensation used in generating the mask; the effects of loop closure would propagate through the pipeline, and would need to be accounted for. However, it is a promising direction for future work as would allow the system to recover from tracking failures and sensor drift.

5.2 Personal Reflection

For a personal reflection, I will switch to first person. I have found this dissertation to be an extremely challenging but rewarding introduction to the world of research. I have never aspired to perform research. I consider myself a poor writer, and expressing my thoughts in written English is not something that comes naturally to me. The prospect of being let loose on any problem with no clear solution, and no clear path forward, was daunting and overwhelming. And yet, as I am proofreading my final draft, I have an immense feeling of pride of what I have accomplished. This work represents *months* of hard work and frustration and I feel that it is a fair reflection of what I am a capable of as a Computer Scientist.

I have learnt a lot from this project, both about 3D reconstruction and the world of research as a whole. For instance: I have learnt how to read an academic paper; before this project I had never read one in its entirety. I now have an appreciation for the incredible amount of effort and sacrifice that every published paper is emblematic of, and a feeling of reverence towards the academic staff at the University of Bath's department of Computer Science who are able to produce research at a remarkable cadence.

In my opinion, the project has been a success. The 3D reconstruction system I have developed performs much better than I had envisioned at the start of the project. My work is able to consistently perform as well as, and in some cases outperform, the state-of-the-art that is ReFusion [13]. I am very grateful to have been afforded the opportunity to conduct this research.

5.3 Word Count

This dissertation has 10773 words (excluding these), as counted by Overleaf [1] and shown on the next page.

[1] urlhttps://www.overleaf.com/

Figure 5.1: Word count as counted by Overleaf

Bibliography

[1] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, and Andrew Fitzgibbon. Kinectfusion: Real-time 3d reconstruction and interaction using a moving depth camera. In *UIST '11 Proceedings of the 24th annual ACM symposium on User interface software and technology*, pages 559–568. ACM, October 2011.

[2] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, Oct 2011.

[3] Jianwei Li, Wei Gao, Yihong Wu, Yangdong Liu, and Yanfei Shen. High-quality indoor scene 3d reconstruction with rgb-d cameras: A brief review. *Computational Visual Media*, 8(3):369–393, 2022.

[4] Zhengyou Zhang. Microsoft kinect sensor and its effect. *IEEE MultiMedia*, 19:4–12, April 2012.

[5] T. Whelan, J. McDonald, Michael Kaess, Maurice Fallon, Hordur Johannsson, and John J. Leonard. Kintinuous: Spatially extended kinectfusion. In *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, July 2012.

[6] Thomas Whelan, Renato F Salas-Moreno, Ben Glocker, Andrew J Davison, and Stefan Leutenegger. Elasticfusion: Real-time dense slam and light source estimation. *The International Journal of Robotics Research*, 35(14):1697–1716, 2016.

[7] Thomas Whelan, Stefan Leutenegger, Renato F. Salas-Moreno, Ben Glocker, and Andrew J. Davison. Elasticfusion: Dense slam without a pose graph. In *Robotics: Science and Systems*, 2015.

[8] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004.

[9] Alberto Aguado Mark Nixon. *Feature Extraction Image Processing*. Elsevier, 4 edition, 2023.

[10] Peter Henry, Michael Krainin, Evan V. Herbst, Xiaofeng Ren, and Dieter Fox. Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments. In *International Symposium on Experimental Robotics*, 2010.

[11] Christian Kerl, Jürgen Sturm, and Daniel Cremers. Robust odometry estimation for rgb-d cameras. *2013 IEEE International Conference on Robotics and Automation*, pages 3748–3754, 2013.

[12] Felix Endres, Jurgen Hess, Nikolas Engelhard, Jurgen Sturm, Daniel Cremers, and Wolfram Burgard. An evaluation of the rgb-d slam system. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 1691–1696, 05 2012.

[13] Emanuele Palazzolo, Jens Behley, Philipp Lottes, Philippe Giguère, and Cyrill Stachniss. Refusion: 3d reconstruction in dynamic environments for rgb-d cameras exploiting residuals. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7855–7862, 2019.

[14] H. Zhang and F. Xu. Mixedfusion: Real-time reconstruction of an indoor scene with dynamic objects. *IEEE Transactions on Visualization and Computer Graphics*, 24(12):3137–3146, 2018.

[15] Reinhard Koch, Marc Pollefeys, and Luc Van Gool. Robust calibration and 3d geometric modeling from large collections of uncalibrated images. In Wolfgang Förstner, Joachim M. Buhmann, Annett Faber, and Petko Faber, editors, *Mustererkennung 1999*, pages 413–420, Berlin, Heidelberg, 1999. Springer Berlin Heidelberg.

[16] C. Tomasi and R. Manduchi. Bilateral filtering for gray and color images. In *Sixth International Conference on Computer Vision (IEEE Cat. No.98CH36271)*, pages 839–846, Jan 1998.

[17] Roy Or El, Guy Rosman, Aaron Wetzler, Ron Kimmel, and Alfred M. Bruckstein. Rgbd-fusion: Real-time high precision depth recovery. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5407–5416, June 2015.

[18] Long Yang, Qingan Yan, and Chunxia Xiao. Shape-controllable geometry completion for point cloud models. *The Visual Computer*, 33(3):385–398, 2017.

[19] Yuriy Berdnikov and Dmitriy Vatolin. Real-time depth map occlusion filling and scene background restoration for projected-pattern-based depth cameras. In *'2011*, pages 200–203, 2011.

[20] Angela Dai, Christian Diller, and Matthias Niessner. Sg-nn: Sparse generative neural networks for self-supervised scene completion of rgb-d scans. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 846–855, June 2020.

[21] Andrew W. Fitzgibbon and Andrew Zisserman. Automatic camera recovery for closed or open image sequences. In Hans Burkhardt and Bernd Neumann, editors, *Computer Vision — ECCV'98*, pages 311–326, Berlin, Heidelberg, 1998. Springer Berlin Heidelberg.

[22] Y. Chen and G. Medioni. Object modeling by registration of multiple range images. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, pages 2724–2729 vol.3, 1991.

[23] 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, 2001.

[24] 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 225–234, 2007.

[25] T. Whelan, M. Kaess, H. Johannsson, M.F. Fallon, J.J. Leonard, and J.B. McDonald.

Real-time large scale dense RGB-D SLAM with volumetric fusion. *Intl. J. of Robotics Research, IJRR*, 2014.

[26] 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, New York, NY, USA, 1996. Association for Computing Machinery.

[27] M. Nießner, M. Zollhöfer, S. Izadi, and M. Stamminger. Real-time 3d reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (TOG)*, 2013.

[28] Hanspeter Pfister, Matthias Zwicker, Jeroen van Baar, and Markus Gross. Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, pages 335–342, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[29] Thomas Schöps, Torsten Sattler, and Marc Pollefeys. Surfelmeshing: Online surfel-based mesh reconstruction. *CoRR*, abs/1810.00729, 2018.

[30] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgb-d mapping: Using kinect-style depth cameras for dense 3d modeling of indoor environments. *The International Journal of Robotics Research*, 31(5):647–663, 2012.

[31] Richard A. Newcombe, Dieter Fox, and Steven M. Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 343–352, 2015.

[32] *Real-Time 3D Reconstruction in Dynamic Scenes Using Point-Based Fusion*, 2013.

[33] Raluca Scona, Mariano Jaimez, Yvan R. Petillot, Maurice Fallon, and Daniel Cremers. Staticfusion: Background reconstruction for dense rgb-d slam in dynamic environments. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3849–3856, 2018.

[34] Tianwei Zhang and Yoshihiko Nakamura. Posefusion: Dense rgb-d slam in dynamic human environments. In Jing Xiao, Torsten Kröger, and Oussama Khatib, editors, *Proceedings of the 2018 International Symposium on Experimental Robotics*, pages 772–780, Cham, 2020. Springer International Publishing.

[35] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2d pose estimation using part affinity fields. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1302–1310, 2017.

[36] Tianwei Zhang, Huayan Zhang, Yang Li, Yoshihiko Nakamura, and Lei Zhang. Flowfusion: Dynamic dense rgb-d slam based on optical flow. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 7322–7328, 2020.

[37] Zhaoyang Lv, Kihwan Kim, Alejandro Troccoli, Deqing Sun, James M. Rehg, and Jan Kautz. Learning rigidity in dynamic scenes with a moving camera for 3d motion field estimation. In *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[38] S. Vedula, S. Baker, P. Rander, R. Collins, and T. Kanade. Three-dimensional scene

flow. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, volume 2, pages 722–729 vol.2, 1999.

[39] Christopher M Trombley, Sumit Kumar Das, and Dan O Popa. Dynamic-gan: Learning spatial-temporal attention for dynamic object removal in feature dense environments. In *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 12189–12195, 2022.

[40] Jürgen 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, 2012.

[41] ISO. *ISO/IEC 14882:1998: Programming languages — C++*. September 1998. Available in electronic form for online purchase at `http://webstore.ansi.org/` and `http://www.cssinfo.com/`.

[42] NVIDIA, Péter Vingelmann, and Frank H.P. Fitzek. Cuda, release: 10.2.89, 2020.

[43] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[44] Bruce D Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *IJCAI'81: 7th international joint conference on Artificial intelligence*, volume 2, pages 674–679, 1981.

[45] Gunnar Farnebäck. Two-frame motion estimation based on polynomial expansion. In Josef Bigun and Tomas Gustavsson, editors, *Image Analysis*, pages 363–370, Berlin, Heidelberg, 2003. Springer Berlin Heidelberg.

[46] Berthold KP Horn and Brian G Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

[47] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume. In *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8934–8943, 2018.

[48] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2462–2470, 2017.

Appendix A

Raw Results

This section presents full tables of results from the quantitative evaluation.

Sequence	N	RF	S
rgbd_bonn_balloon	0.1796	**0.1742**	0.1797
rgbd_bonn_balloon_tracking	**0.2869**	0.4665	0.4483
rgbd_bonn_balloon_tracking2	0.4472	**0.2697**	0.4443
rgbd_bonn_balloon2	0.2811	**0.189**	0.2562
rgbd_bonn_crowd	0.1878	**0.1256**	0.173
rgbd_bonn_crowd2	**0.1749**	1.4281	1.3671
rgbd_bonn_crowd3	0.1476	**0.133**	0.1358
rgbd_bonn_kidnapping_box	0.1744	**0.1515**	0.1908
rgbd_bonn_kidnapping_box2	0.1676	**0.1589**	0.1658
rgbd_bonn_moving_nonobstructing_box	0.073	**0.0706**	0.075
rgbd_bonn_moving_nonobstructing_box2	0.196	**0.1791**	0.1968
rgbd_bonn_moving_obstructing_box	0.4413	0.3596	**0.2424**
rgbd_bonn_moving_obstructing_box2	0.574	0.8563	**0.184**
rgbd_bonn_person_tracking	0.3288	**0.2847**	0.3527
rgbd_bonn_person_tracking2	0.4698	**0.4673**	0.4796
rgbd_bonn_placing_nonobstructing_box	0.134	**0.105**	0.1071
rgbd_bonn_placing_nonobstructing_box2	0.1455	**0.1393**	0.1456
rgbd_bonn_placing_nonobstructing_box3	0.1839	**0.1771**	0.2008
rgbd_bonn_placing_obstructing_box	1.1256	**0.7599**	0.7873
rgbd_bonn_removing_nonobstructing_box	0.0428	**0.04**	0.0436
rgbd_bonn_removing_nonobstructing_box2	0.1184	**0.1088**	0.1169
rgbd_bonn_removing_obstructing_box	0.2051	**0.1931**	0.2347
rgbd_bonn_static_close_far	**1.4411**	1.5472	3678.6443
rgbd_bonn_synchronous	**0.2262**	0.6466	0.4655
rgbd_bonn_synchronous2	0.0264	**0.021**	0.0278
rgbd_dataset_freiburg2_desk_with_person	0.1381	**0.0506**	0.0983
rgbd_dataset_freiburg3_sitting_rpy	1.0594	0.1453	**0.0873**
rgbd_dataset_freiburg3_sitting_static	0.0111	**0.0105**	0.0112
rgbd_dataset_freiburg3_sitting_xyz	0.0483	**0.0386**	0.0423
rgbd_dataset_freiburg3_walking_rpy	0.8439	**0.4197**	0.5219
rgbd_dataset_freiburg3_walking_static	0.0273	**0.0162**	0.0289
rgbd_dataset_freiburg3_walking_xyz	0.1239	**0.0875**	0.4676

Table A.1: The raw results of the root mean squared error (RMSE) for ATE across the different reconstruction strategies. The minimum (i.e., best) RMSE for each reconstruciton sequence is highlighted in bold.

Sequence	N	RF	S
rgbd_bonn_balloon	**0.2895**	0.2907	0.2901
rgbd_bonn_balloon_tracking	**0.5277**	0.5363	0.5595
rgbd_bonn_balloon_tracking2	0.6534	**0.6145**	0.6227
rgbd_bonn_balloon2	0.3081	**0.2891**	0.3006
rgbd_bonn_crowd	0.1419	**0.124**	0.1503
rgbd_bonn_crowd2	**0.1718**	0.4784	0.4024
rgbd_bonn_crowd3	0.1328	**0.1203**	0.1265
rgbd_bonn_kidnapping_box	0.3248	**0.3218**	0.3258
rgbd_bonn_kidnapping_box2	**0.3162**	0.3163	0.3164
rgbd_bonn_moving_nonobstructing_box	0.2609	0.2618	**0.2607**
rgbd_bonn_moving_nonobstructing_box2	0.3259	**0.3254**	0.3262
rgbd_bonn_moving_obstructing_box	0.344	0.3038	**0.2675**
rgbd_bonn_moving_obstructing_box2	0.2836	0.4086	**0.1891**
rgbd_bonn_person_tracking	0.4368	**0.4247**	0.453
rgbd_bonn_person_tracking2	0.3697	**0.3676**	0.3711
rgbd_bonn_placing_nonobstructing_box	0.1562	**0.1503**	0.1533
rgbd_bonn_placing_nonobstructing_box2	0.1864	**0.1837**	0.1861
rgbd_bonn_placing_nonobstructing_box3	0.2782	**0.2754**	0.2771
rgbd_bonn_placing_obstructing_box	0.3366	0.2678	**0.2339**
rgbd_bonn_removing_nonobstructing_box	**0.1703**	0.1704	0.1707
rgbd_bonn_removing_nonobstructing_box2	0.2324	**0.2313**	0.232
rgbd_bonn_removing_obstructing_box	**0.1585**	0.1589	0.1689
rgbd_bonn_static_close_far	**0.5223**	0.596	1479.4257
rgbd_bonn_synchronous	**0.1543**	0.3493	0.2958
rgbd_bonn_synchronous2	0.0434	**0.0418**	0.0438
rgbd_dataset_freiburg2_desk_with_person	0.0341	**0.0193**	0.0213
rgbd_dataset_freiburg3_sitting_rpy	0.3807	0.147	**0.0479**
rgbd_dataset_freiburg3_sitting_static	0.0084	**0.008**	0.0083
rgbd_dataset_freiburg3_sitting_xyz	0.0217	**0.0178**	0.0191
rgbd_dataset_freiburg3_walking_rpy	0.7693	**0.3129**	0.3389
rgbd_dataset_freiburg3_walking_static	0.0207	**0.014**	0.0274
rgbd_dataset_freiburg3_walking_xyz	0.1017	**0.0833**	0.2619

Table A.2: The raw results of the root mean square error (RMSE) values for RPE metric across the different reconstruction strategies. The minimum (i.e., best) RMSE for each reconstruction is highlighted in bold.