



Non-Iterative Three-Dimensional Reconstruction and Representation

By Luke Lincoln

A thesis submitted in partial fulfilment of the requirements of
the Degree of Doctorate of Philosophy

School of Information and Communication Technology
Griffith University, Australia

August, 2017

Statement of Originality

To the best of my knowledge and belief, the thesis contains no material previously published or written by another person except where due reference is made in the thesis itself.



31-OCT-2017

Signature

Date

Abstract

Three-dimensional reconstruction algorithms generate 3D data from two-dimensional image or video data. The current focus of this research area is on iterative algorithms such as: feature matching/RANSAC, Iterative Closest Point, and other non-linear optimization strategies. These strategies tend to fail in scenes with few features or scenes which contain feature confusion.

In 2D image registration research, feature matching is dominant but closed form solution based Fourier registration techniques have been proven to outperform them with increased robustness to under textured scenes and image noise. This thesis investigates the application of Fourier Volume Registration to 3D reconstruction. Results are compared between Fourier Volume Registration, and several current techniques both quantitatively and qualitatively. Results show that the Fourier Volume Registration Technique often outperforms other methods in terms of minimizing registration error prior to optimization. Furthermore it is a closed form solution which works well with parallel processing architectures. 3D data representations for 3D reconstruc-

tion data are also explored to improve storage and transmission of such data. Many current methods make use of Signed Distance Functions, volumetric occupancy grids or octrees. In the work presented here, lossy octree compression is analysed to pave the way for new storage and transmission rates of efficiency. A novel method, called the Plane-Tree, is proposed based on the octree compression method. This Plane-Tree data structure was inspired by work completed in the author's honours thesis. When compared to the original octree data structure, the Plane-Tree is optimal in terms of compression performance. The findings presented on both the Fourier Volume Registration method and the Plane-Tree indicate improvements over existing methods.

Acknowledgements

I would like to thank Dr. Ruben Gonzalez my supervisor.

Contents

Statement of Originality	iii
Abstract	v
Acknowledgements	vii
List of Figures	xiii
List of Tables	xix
1 Introduction	1
1.1 Introduction	1
1.2 Research Aims & Contributions	4
1.3 Overview	7
2 Literature Review	9
2.1 Background	9

2.1.1	Feature Matching	9
2.1.2	Phase Correlation	15
2.1.3	Depth Data Generation	21
2.1.4	Data Representations	32
2.2	3D Reconstruction Techniques	40
2.2.1	The Fundamental Matrix and its Properties	42
2.2.2	Feature Matching and RANSAC	49
2.2.3	ICP	55
2.2.4	Direct Optimization Methods	61
2.2.5	3D Feature Matching	65
2.2.6	Principal Components Analysis	70
2.2.7	Further Optimization	70
2.3	Summary	72
3	Methodology	75
3.1	Fourier Volume Reconstruction	75
3.1.1	Basic Volume Registration	79
3.1.2	Reconstruction from Stereo or 3D Data	101
3.1.3	Reconstruction from Stereo or 2D Data	110
3.1.4	FVR: Efficiency	125
3.1.5	3D Rotation Recovery	148

3.2	Plane-Tree Data Compression	162
3.2.1	Octree Overview	163
3.2.2	3D ShadeTree Coding	170
3.2.3	Plane-Tree Coding	172
3.2.4	Octree Subdivision	174
3.2.5	Leaf Node Computation and Representation	175
3.2.6	Compression and Decompression	177
3.2.7	The Plane-Tree vs. the SOT	178
4	Experiments	181
4.1	Experiments	181
4.2	Tools	183
4.3	Error metrics	184
4.4	Data Sources	189
4.4.1	Algorithms	189
4.4.2	Sensor Types	193
4.4.3	Stereo Camera	196
4.4.4	Active Camera	205
4.4.5	Monocular Camera	215
4.5	Camera Tracking & Noise Robustness	218
4.5.1	Camera Rotation Tracking	221

4.5.2	Reconstructed Scenes	224
4.5.3	Comparison	227
4.6	Plane-Tree Experiments	229
4.7	Plane-Tree vs. Octree	231
4.8	Plane-Tree vs. Existing Techniques	232
4.9	Plane-Tree: Qualitative Results	235
4.10	Plane-Tree: Reconstruction Compression	239
5	Conclusion	249
5.1	Fourier Volume Registration	250
5.2	Plane-Tree 3D Data Compression	251
5.3	Research Aims Revisited	252
5.4	Future Work	253
Reference List		255
Appendix A		285
.1	Stereo Results	285
.2	Active Camera Results	311

List of Figures

2.1	The polar representation of the Discrete Fourier transform on an input image.	18
2.2	Phase correlation used to align two images seperated by a translation.	19
2.3	The effects of the log polar transform.	20
2.4	Left: Polygonal Mesh, Right: Triangulated Mesh	33
2.5	A densely sampled point cloud of the Stanford Bunny.	35
2.6	Left: Bunny sampled into a 32^3 volume, Right: Bunny sam- pled into a 128^3 volume. Both images credit: [80]	36
2.7	Distance Field Slices of the Stanford Bunny [84].	37
2.8	A Height Map of the Stanford Bunny.	39
2.9	A pair of cameras viewing two points. This figure is a recon- struction of the figure on page 492 in Computer and machine vision: theory, algorithms, practicalities [115].	45

2.10	RGB-D SLAM Pipeline used by Endres et al. [111]	53
2.11	Kinect Fusion Algorithm Pipeline [158]	59
3.1	Left: Captured 3D frame, Right: Magnitude values in the frequency domain of the captured 3D frame	86
3.2	Left: Captured 3D frame, Right: Log-Polar Transform of the captured 3D frame	96
3.3	System Diagram for Registration Process	104
3.4	A Projected Frame.	117
3.5	Comparison Between Block Matching Estimated Depth (Top) and RGBD Device Depth (Bottom).	125
3.6	System Diagram for Fast Volume Registration	129
3.7	The Spherical Map Transform.	132
3.8	The Projection-Map Transform.	135
3.9	Comparison of performance between FVR and FFVR.	141
3.10	Theoretical Complexity Comparison	146
3.11	Theoretical Complexity Comparison (Including FM3D and ICP)	147
3.12	Early stages of FVR-3D	152
3.13	Final stages of FVR-3D	158
3.14	System Diagram for Registration Process	161
3.15	Visualization of cubic space being split [188].	164

3.16 An Example of an octree used to represent an object at different levels of accuracy [188]. Left: Original model, Middle: few node splits used to represent the model. Right: many node splits used to more accurately represent the model.	165
3.17 Quadtree Image Representation, left to right: Original Image, 1st Level of Decomposition, 2nd Level of Decomposition, 3rd Level of Decomposition, QT Codec Image.	166
3.18 A visualization of the QT hierarchy	167
3.19 Visualization of OT Decomposition	168
3.20 Tree Traversals, Left: Depth-First Traversal, Right Breadth-First Traversal	169
3.21 A possible child ordering for the QT (left), and the OT (right).	170
3.22 A Linear Tree Code Representation Example	171
4.1 Kitti 0001 Sync Data Set Sample	199
4.2 Kitti 0002 Sync Data Set Sample	202
4.3 Kitti 0005 Sync Data Set Sample	203
4.4 Kitti 0091 Sync Data Set Sample	204
4.5 Kitti 0095 Sync Data Set Sample	205
4.6 Four Sample Frames from the Apartment Texture Rotate Data Set	208

4.7 Four Sample Frames from the Apartment Texture X-Axis Rotation Data Set.	209
4.8 Four Sample Frames from the Desk Texture Translation Data Set.	211
4.9 Four Sample Frames from the Office Textured Blind-spot Rotation Data Set.	212
4.10 Four Sample Frames from the Office Textured Items Translation Data Set.	213
4.11 Four Sample Frames from the Park Plants and Table Data Set.	214
4.12 Four Sample Frames from the Office Texture Rotation Data Set.	215
4.13 Left: Ground Truth Depth Map Computed Using a LIDAR System, Right: Depth Map Computed Using a Monocular Method	217
4.20 The Bunny Model Compressed Using the Plane-Tree, Valence and Spectral Methods.	237
4.21 The Bunny Model Compressed Using the Plane-Tree and the Wavelet Compression System Khodakovsky et al.	239
4.14 Reconstructed Scenes.	241
4.15 Reconstructions of the Parks, Plants and Table Scene Using FVR, ICP and FM2D.	242
4.16 Models used to assess the Plane-Tree compression algorithm.	243

4.17 Rate-distortion graphs comparing the Plane-Tree with the Oc-tree.	244
4.18 Rate-Distortion graphs comparing the Plane-Tree to different state of the art codecs.	245
4.19 Models coded with our coder at thresholds of 8.0 and 1.0 with a maximum tree depth of 6 along with the original models. . .	246
4.22 PSNR vs Bit-rate comparing the OT and Plane-Tree compression methods.	247

List of Tables

3.1	Complexities for given Procedures	138
3.2	Complexities for given Procedures	140
3.3	Algorithm Complexities	145
4.1	Reconstruction Errors for the Kitti Data 0001 Sync Data Set .	201
4.2	Reconstruction Errors for the Kitti Data 0002 Sync Data Set .	201
4.3	Reconstruction Errors for the Kitti Data 0005 Sync Data Set .	203
4.4	Reconstruction Errors for the Kitti Data 0091 Sync Data Set .	204
4.5	Reconstruction Errors for the Kitti Data 0095 Sync Data Set .	205
4.6	Reconstruction Errors for the Apartment Texture Rotate Data Set	208
4.7	Reconstruction Errors for the Apartment Texture X-Axis Ro- tation Data Set	209
4.8	Reconstruction Errors for the Desk Texture Translation Data Set	210

4.9	Reconstruction Errors for the Office Texture Blind-spot Rotation Data Set	212
4.10	Reconstruction Errors for the Office Textured Items Translation Data Set	213
4.11	Reconstruction Errors for the Park, Plants and Tables Rotation Data Set	214
4.12	Reconstruction Errors for the Office Texture Rotation Data Set	215
4.13	Reconstruction Errors for the Kitti Data 0001 Sync Data Set Using Monocular (RGB) Input Only	218
4.14	Translation Tracking	220
4.15	Rotation Tracking	222
4.16	FVR Comparison Table	229
4.17	Seconds Per Frame Registration	230
1	Raw Registration Results for the Kitti Data 0001 Sync Data Set	285
2	Raw Registration Results for the Kitti Data 0002 Sync Data Set	288
3	Raw Registration Results for the Kitti Data 0005 Sync Data Set	290
4	Raw Registration Results for the Kitti Data 0091 Sync Data Set	295

5	Raw Registration Results for the Kitti Data 0095 Sync Data Set	304
6	Raw Registration Results for the Apartment Texture Rotate Data Set	311
7	Raw Registration Results for the Apartment Texture X-Axis Rotation Data Set	312
8	Raw Registration Results for the Desk Texture Translation Data Set	313
9	Raw Registration Results for the Office Textured Blindspot Rotation Data Set	314
10	Raw Registration Results for the Office Textured Items Translation Data Set	315
11	Raw Registration Results for the Park Plants and Tables Rotation Data Set	316
12	Raw Registration Results for the Office Texture Rotation Data Set	317

1

Introduction

1.1 Introduction

In recent years, there has been increasing demand for research and applications within the areas of human-computer interaction, including: virtual reality, augmented reality and 3D reconstruction. This thesis presents research on 3D reconstruction. 3D reconstruction is concerned with building dense 3D models out of various sensor input, such as video recorded with a stereo camera set-up, active sensor video data and monocular camera video data. This area is very similar to Simultaneous Localization and Mapping or SLAM. However, the topics differ in that SLAM research has not histori-

cally focused solely on producing dense 3D reconstructions. SLAM also has a focus on accurate camera localization. In 3D reconstruction, as long as pleasant, dense and useful 3D reconstructions are computed, accurate scans and camera localization do not matter.

3D reconstruction is important in a wide variety of fields including business, engineering and architecture, virtual reality and augmented reality. For example, an architect may want to record 3D structural data for later study. An engineer may want to study the under area of a bridge to assess the presence of possible faults while a software engineer may want to create an augmented reality application where possible home buyers take virtual tours through an existing property. Of course, the specifications and 3D structures of these areas may be reverse engineered and recorded. In this case, 3D models may be manually built by artists or engineers. This however, costs both time and money, furthermore some structures have no blueprint. For example, we may also want to provide virtual walking tours through a rainforest. This definitely has no blueprint.

Currently, there exist many methods capable of extracting 3D reconstructions and computing camera pose information from image and video data. Methods include: the Fundamental matrix method, Feature Matching

(FM) (both 2D and 3D) with the Random Sample Consensus (RANSAC) algorithm, Principal Components Analysis (PCA) and Iterative Closest Point (ICP). There also exist other forms of non-linear optimization, however these algorithms are used most often in state-of-the-art methods. These algorithms work well in simplistic environments where features are plentiful and data noise/corruption is of no concern. The Fundamental matrix method is used in the case of monocular video data, it requires camera calibration and currently cannot be used to generate accurate dense 3D data. Feature Matching methods (both 2D and 3D) used with RANSAC may be used in conjunction with an active or stereo camera set-up to generate dense 3D reconstructions. ICP may also be used in the same way. The problem with these methods is that they are not robust to noise. They also tend to fail in low-textured scenes and cannot register wide-baselines. Feature Matching based methods also fail when texture confusion occurs. Texture confusion occurs in scenes with many features which look identical locally but appear in different locations spatially, throwing off registration calculations. PCA may also be used in conjunction with an active camera or stereo camera set-up to register 3D frames to create 3D reconstructions. Whilst PCA can handle wider base-lines, it is more susceptible to noise. These iterative methods have complexity which depends on the input data, furthermore they may also get stuck in a local minimum prior to finding an optimal registration.

During 3D reconstruction research, it became apparent that 3D reconstruction data uses large amounts of storage. Since 3D reconstruction algorithms may be storing multiple 3D frames, and may also use some global representation for output, a compression could improve data processing, transmission and storage. Whilst lossless compression research for such data has begun, lossy compression remains unexplored. Therefore, research on a lossy compression system for 3D data is also presented in this thesis.

1.2 Research Aims & Contributions

In the current state of 3D reconstruction research, Fourier based methods have not been investigated. Research has proven 2D Fourier based registration techniques able to register wide base-lines and to be reliable, accurate and robust to noise, but in the context of 3D registration, such research does not exist. It is beneficial to use Fourier techniques as they are robust to noise, accurate and work well on wide base-lines. Fourier based approaches are also closed form solutions, in that their complexity does not depend on their input data only upon the dimension sizes of the data. In the case of compression systems for 3D reconstruction purposes, lossy compression methods based on the hierarchical octree data structure and modifications [1] were investigated

for compression of 3D data including 3D reconstruction frames and outputs.

The primary aim of this research is to improve the accuracy, noise robustness, speed and storage, quantitative quality and perceptual quality of 3D models generated from image data. To this end, Fourier based registration schemes were investigated as well as hierarchical data compression systems. Investigation into Fourier methods motivated the research questions, “Can Fourier based registration techniques improve accuracy and noise robustness in 3D reconstruction applications?” and “Can hierarchical techniques improve compression, storage and processing of 3D reconstruction data?”

Efforts to answer these questions have led to several novel algorithms and data structures being proposed. These are presented and discussed in Chapter 3. Firstly, a novel method applying Fourier based registration to 3D reconstruction named Fourier Volume Reconstruction (FVR) is proposed [2, 3]. This algorithm is able to generate accurate and dense 3D reconstructions whilst being robust to noise (see sections 4.5 and 4.4). The FVR method works well when provided with dense 3D frames from which to reconstruct a global model. The FVR method may be used to reconstruct data input via a RGB-D active camera or stereo camera set-up. To provide support for monocular camera sensors, a novel algorithm based on FVR was proposed

named Monocular View Volume Reconstruction (MVVR) [4]. Our research suggests, the higher the quality of the input depth data (produced via active, stereo or software based methods), the better the registration produced by FVR 4.5.

Since, the FVR method is only able to register against a single axis of camera rotation, a novel method which incorporates PCA into FVR is proposed. This method named FVR-3D outperforms several pose registration methods used in state-of-the-art 3D reconstruction algorithms using different input sensors (stereo & active) in terms of accuracy and robustness (see section 4.4 for results). A novel speed-up for the FVR algorithm is also proposed (section 3.1.4). This method, named Fast Fourier Volume Reconstruction (FFVR) improves efficiency by reducing the data to be processed by an entire dimension. Our results suggest this method improves upon the FVR method at the expense of registration accuracy.

In terms of compression research, a novel data representation based on octree 3D data compression is proposed named the Plane-Tree data representation [5]. This data representation and subsequent compression and decompression algorithm outperform the generic octree as well as several state-of-the-art 3D data compression algorithms.

1.3 Overview

This thesis is laid out as follows: Chapter 2 presents a review of techniques which are critical to different 3D reconstruction techniques. Following this, Chapter 3 introduces the proposed techniques used in this research project. These techniques are used to answer the research questions and accomplish the primary aim of this project. Chapter 4 details the experiments performed, and presents quantitative results showing the effectiveness of the techniques. Chapter 5 concludes the thesis with an analysis and discussion of the results and by discussing the results in terms of the primary aim and research questions. The results of this study bring Fourier based reconstruction methods into frame and show them to be capable of state-of-the-art 3D reconstruction and registration. These results also provide a platform for future Hierarchical compression methods and show that such methods outperform state-of-the-art compression systems.

2

Literature Review

2.1 Background

2.1.1 Feature Matching

Feature Matching is of major importance not just in SLAM and 3D Reconstruction but in much of the research and applied applications within Computer Vision. Therefore in this section we provide a brief survey of popular feature detection, description and matching algorithms.

Harris and Stephens [6] invented the Harris corner detector. This detector uses a variable sized window around each pixel, in which a Harris matrix

is formed from the x, y and xy gradients. The Harris response is calculated using the determinant and trace of this matrix. Several years later, Smith and Brady [7, 8] presented a feature detector called SUSAN. SUSAN is an alternative to second order methods for corner detection and uses a non-linear filter to find corners and edges. SUSAN also naturally provides feature vectors. It works by surrounding each pixel with a circular non-linear kernel for filtering. The kernel’s response is defined as the area within the kernel having the same or similar value to the nucleus (centre of the kernel). Walker et al. [9] used a classification (machine learning) method to find salient places in images.

Trajkovic & Hedley [10] presented a fast yet simple corner detection algorithm. This method computes the minimum intensity changes in all directions. It is fast as it only uses a 3x3 window for the corner response function. This method is compared to the Harris and Susan corner detectors. It is faster than both these methods whilst also being more robust than the Susan corner detector. Harris is the more accurate corner detector in terms of repeatability and detection.

Lowe’s [11, 12] method, SIFT (Scale Invariant Feature Transform) is a popular method for feature extraction and description. The method com-

putes features at different scales using an image pyramid and difference of Gaussian to approximate the Laplacian of Gaussian. Features are represented using a vector of weighted gradients surrounding the feature. The descriptor is invariant to scale because each descriptor is found at some scale within the multi-resolution pyramid. SIFT is rotationally invariant if the window is chosen so that its angle of origin is based on the angles surrounding the feature. It is invariant to luminance because of the use of gradients, and since features are described by the surrounding window of the feature, they are invariant to translation. This method is said to be robust to 3D viewing transforms and affine transforms. SIFT was also developed to run faster on GPUs [13].

Tuytelaars & Van Gool [14] developed a method for detecting and describing affine invariant regions. These regions are computed directly from intensity values in the image using rays extending from the centre of regions. Feature vectors made up from statistical moments within the regions, then nearest neighbour matching is used to match features. Boykov & Jolly [15] presented a method for region based feature extraction. This method uses graph cuts to find which regions are adequate features. This method requires some soft constraints performed by humans so is not good for automatic detection of features, also because features are not very localized their size and

shape is not viewpoint invariant.

Itti & Koch [16] presented a biologically inspired bottom up image saliency detector. Schaffalizky and Zisserman [17] developed a texture based region descriptor. It is invariant to photometric and affine transformations. It is also insensitive to the shape of the region and can be used to compute epipolar geometry. This method makes use of the second order matrix. Mikolajczyk and Schmid [18] presented a new feature point detector and descriptor. Their detector is based on multi-scale Harris corner detection which is used to filter the points by the value of the surrounding Laplace. For feature description they use Gaussian derivatives.

Carson et al. [19] developed a method for image feature classification and matching called Blobworld. This method segments an image before using region vectors for image querying and feature matching. This technique begins by defining a pixel neighbourhood size, it then groups pixels together based on the texture and colour data within one of these neighbourhoods. Finally, vectors describing colour and texture are formed for each region and these are used in image queries. Sebe et al. [20] compared local based feature detectors, their proposed method uses a wavelet saliency extractor, making use of textures and colour in order to obtain invariance in its descriptor.

Kadir et al. [21] developed a saliency based method for feature detection. This method is scale, viewpoint and perturbation invariant. Carbonetto et al. [22] presented a method which segments images, labelling them with feature vectors made up of descriptive words. Image region mapping can be performed by statistically comparing feature description vectors. Matas et al. [23] developed a new feature detection and description method called MSER. This algorithm uses small regions as features instead of a single point and surrounding window. These regions are calculated by taking the foreground blobs of an image at every possible binary threshold. MSER detection and representation is invariant to scale (3.5 x), illumination, out-of-plane rotation, occlusion, locally anisotropic scale change and 3D translation of viewpoint.

Mikolajczyk and Schmid [24] performed an evaluation of local feature descriptors. They performed a comparison between shape-context, PCA-SIFT, differential invariants, spin images, SIFT, complex filters, moment invariants and cross-correlation. They also present their method called GLOH, which is an extension of the SIFT descriptor. Results indicate that GLOH and SIFT perform the best. Rosten & Drummond [25, 26] aimed to improve the speed of feature detection over SIFT and SUSAN. They developed a technique called FAST (Features from Accelerated Segment Test), this method

tests the difference between the centre pixel and its surrounding pixels within the surrounding circle. They also improved this approach by first extracting FAST corners, then classifying these corners using a decision tree to extract better feature points, whilst retaining speed. Both FAST and FAST-ML (FAST with Machine Learning) methods are shown to be faster than SIFT and SUSAN whilst FAST-ML is also shown to be more reliable at classifying the same features from different viewpoints. Another method, ORB [27] is based on FAST and BRIEF [28]. ORB detects features similarly to FAST and computes rotation and feature descriptors similarly to BRIEF. It is shown to be robust to viewpoint changes and is faster than SIFT and SURF.

Bay et al. [29, 30] improved the speed and accuracy of feature matching compared with SIFT using their method named SURF (Speeded Up Robust Features). In SURF, a Hessian matrix is used for the detection of features whilst Haar wavelet components are used as descriptors. A non rotation invariant version was also analysed and proved to be faster. This version is suitable if rotation invariance is not required for a particular application. Lepetit and Fua [31] turn the wide stereo baseline matching problem into a classification problem. This method is stated to be robust, accurate and real-time. The training phase attempts to classify repeatable corners. The

training set is built from many different rasterizations (which affect illumination) of only a few images. Key-points are extracted at multiple octaves and scales at the surrounding image patch. The authors report randomized trees as being the optimal machine learning method for their technique.

Cabani & MacLean [32] presented a feature detector based on the Harris point operator. This method detects affine invariant features using the speed of an FPGA. It can process images of 640 x 480 at up to 30 frames per second. This technique is compared to C and Matlab versions of the same process. Finally, Tuytelaars and Mikolajczyk [33] presented a survey on feature detectors. They provided a detailed introduction to the subject and categorize different feature types and techniques.

2.1.2 Phase Correlation

Phase correlation is a technique used in digital signal processing to register two signals, images or 3D volumes with translation differences. It is robust to noise and view-point changes. Techniques in phase correlation can also be used to find additional parameters (such as rotation, scale and sheer). Fourier methods such as phase correlation can therefore be used to compute camera pose without the need for non-linear methods such as ICP, which

cannot recover scaling parameters and has several other inadequacies (see section 2.2.3).

In this section we describe 2D phase correlation. The 2D version of phase correlation is important to discuss as techniques are applied directly to the 3D version (described in chapter 3.1) which is important for this research.

Phase correlation is part of an area of research and applications named Digital Signal Processing (DSP). DSP is the scientific area concerned with the acquisition, filtering, processing and understanding of digital signals using computer technology. Digital signals are simply an array of numbers, digital images are 2D signals and digital volumes are 3D signals. One of the fundamental ideas behind this research area is that digital signals can be processed using a set of common techniques regardless of where or how they were generated. There are two fundamental techniques used in digital signal processing. One is the use of correlation and convolution, the other is the use of alternative representations ie the Fourier transform [34].

Convolution is the process by which one signal is used to filter another whilst correlation is used to measure the similarity between one signal and another at different phases. The Fourier transform is used to transform sig-

nals (in the time domain) into the frequency domain. The frequency domain represents the original signal using a set of sine and cosine waves at different frequencies. The time domain is the name given to the original domain of the signal. This transform is performed by correlating the signal with sine and cosine waves at varying frequencies. A transform called the inverse Fourier transform can be used to transform a signal from the frequency domain back to the time domain.

The inputs and outputs of the Fourier transform are digital signals made up of complex numbers. Most of the time, the input signal is not complex (as in the case of a photographed scene). In this case imaginary value scalars of zero are augmented to the signal to make it complex. The frequency domain can be visualized easily using the polar representation of these complex numbered signals. The polar representation represents the signal using the magnitude and phase of sine waves at different frequencies and directions (in the case of 2D). This can be visualized in figure 2.1, here the magnitude represents the height of the sine waves, the phase represents the translation from the origin in which these waves occur and the 2D position within the image represents the direction the waves travel in 2D. The most useful aspect of the Fourier transform is that when signals are represented in the frequency domain, their point-wise multiplication is equivalent to convolution in the

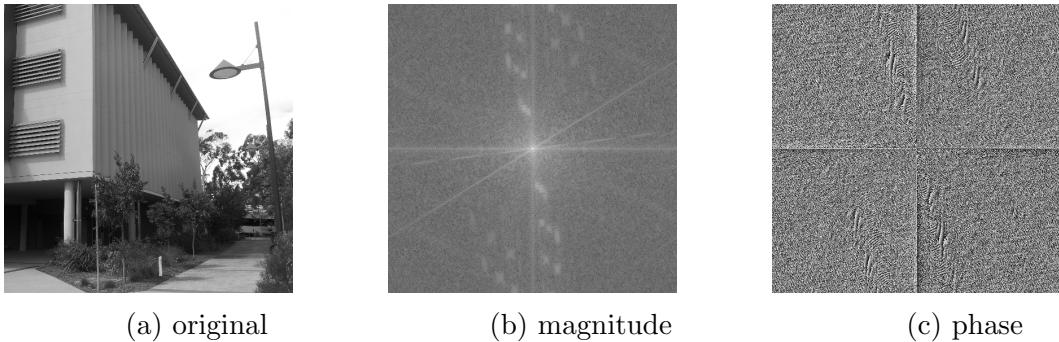


Figure 2.1: The polar representation of the Discrete Fourier transform on an input image.

time domain. This is important because convolution is a computationally intensive process, especially in image processing where there is a lot of data to process.

Phase correlation is the process of using the frequency domain representation to perform image registration using correlation. In image processing, this allows us to find the translation parameters between two images (it does not work if rotation or scaling are introduced) efficiently. This can be performed by flipping one image before transforming both into the frequency domain. Since point-wise multiplication in the frequency domain causes convolution in the time domain, flipping one image causes correlation in the frequency domain which is what is required. Once this correlation is performed, the inverse Fourier transform is performed, the output image contains a peak which is used to compute the translation difference between the two images.

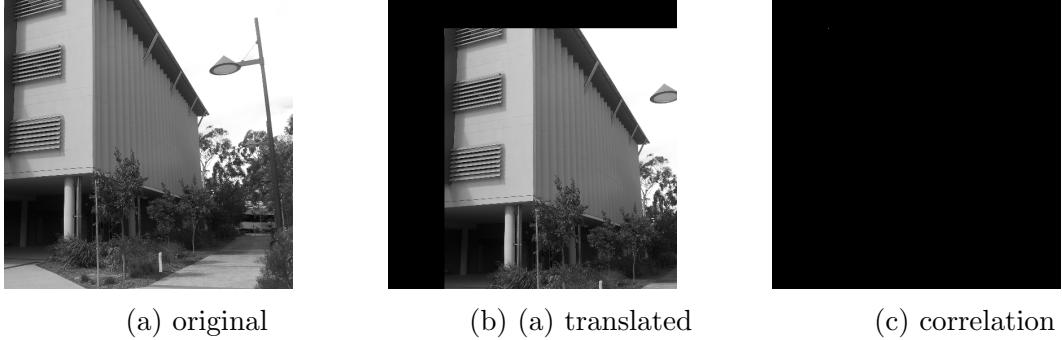


Figure 2.2: Phase correlation used to align two images separated by a translation.

This can be visualized in figure 2.2. Here, the original image is translated from figure 2.2a to figure 2.2b and the two are phase correlated producing the peak in figure 2.2c which represents the translation between the two.

The other parameters we wish to estimate for registration are the scale and rotation parameters, however another type of transform is introduced first. This image transform is called the log-polar transform. This transform re-arranges the pixels from euclidean 2D space $[x, y]$ to the polar space $[\log(\sqrt{x^2 + y^2}), \arctan(y/x)]$ where rotation about the centre is turned into y-axis translation and scaling about the centre is turned into x-axis translation. This representation changes any rotation and scaling performed on the image into translation. Translation parameters can be easily found using phase correlation. This process can be visualized with the help of figure 2.3. The problem is, the rotation and scaling has to be about the centre of the

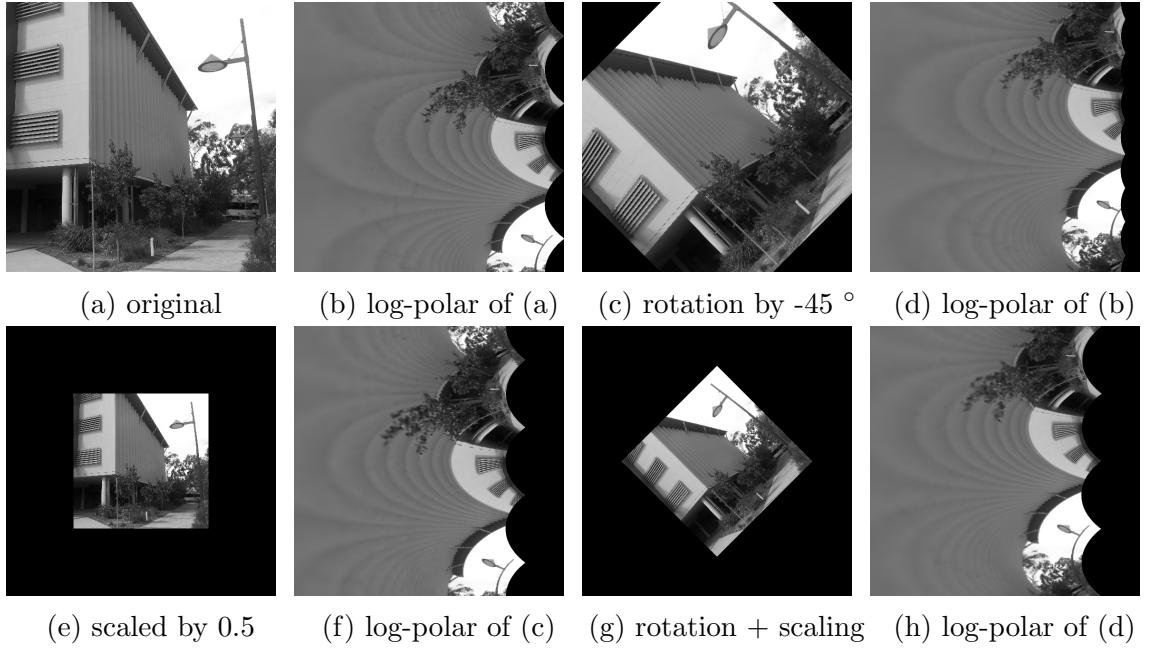


Figure 2.3: The effects of the log polar transform.

image, which is an issue if the images contain a translation effects. Luckily, in the magnitude of the polar representation of the frequency domain the effects of translation are not present. On top of this, any rotation or scaling (whether translation is present or not) occurs about the centre of the image.

This allows the phase correlation method to recover the translation, scaling and rotational information between two images. First, the magnitude of the polar representation of both the images is computed. Then both magnitudes are log-polar transformed and phase correlated. This finds the scaling and rotational parameters. Both of the original images have their

rotation and scaling reversed, then both images are phase correlated to undo the effects of translation, which is the final parameter to compute. In this way, translation, scaling and rotational parameters can be computed directly without the need for feature matching.

2.1.3 Depth Data Generation

3D reconstruction algorithms require either an explicit or implicit means of obtaining dense 3D data per frame. Without this dense 3D data, registering points would only lead to sparse maps rather than dense 3D reconstructions. This section introduces some techniques and research on different methods (both hardware and software) of depth data generation. In the first section, sensors which are capable of generating dense depth data are introduced. Next, software methods using stereo camera pairs are discussed. Finally, monocular techniques which are also software based techniques, are discussed.

Sensors

In 3D reconstruction, it is often ideal to use specialized sensors which capture reliable and dense depth data on a per pixel basis. One such camera is the

RGB-D (Red, Green, Blue & Depth) camera. These sensors are becoming more accurate and less expensive and are now found in mobile technologies.

Research by Zhang et al. [35] used an RGB-D camera to generate smooth, continuously updating dense 3D reconstructions using only depth data. Here, using the data, 6 degrees of freedom were tracked. This technique uses depth information only, and as such it works in the absence of visual light (it works in the dark) unlike passive camera approaches [36, 37, 38] and techniques which use colour data along with depth data [39].

The Kinect is one such rgb-d camera. It uses a structured light based depth sensor along with an application specific integrated circuit to generate an 11-bit 640×480 depth map at 30 times per second (real-time).

There is no doubt that these sensors are the easiest way to compute ready dense depth data for 3D reconstruction in real-time, but there are several drawbacks. For example, depth images generated by such sensors contain holes. This is caused by a lack of structured light on a captured surface. Some materials simply do not reflect infra-red light (surfaces at steep angles or very thin objects). Also, when moving fast, the device easily experiences motion blur which leads to missing and incorrect data.

Some 3D reconstruction techniques use the approach of dense mapping and tracking via depth sensors and lasers. These approaches usually compute feature matches to align the frames or minimize an error function.

Stereo Cameras

Stereo camera set-ups employ two cameras capturing a singular scene. Using a variety of techniques, depth data is computed from the stereo pair by measuring parallax. Stereo cameras may be calibrated or un-calibrated. Un-Calibrated stereo pairs require calibration by computing using the Essential matrix, whilst pre-calibrated cameras require no further action before parallax may be computed. Using a variety of pose estimation techniques, the dense depth data may be integrated forming a 3D reconstruction.

In this section, some important techniques proposed within the area of depth data generation via stereo algorithms. Readers wanting to research techniques prior to 2005 are encouraged to see a survey by Scharstein and Szeliski [40].

Sun et al. [41] presented an occlusion handling stereo matching algorithm. Their method incorporates a visibility constraint into the energy function for

the belief propagation global optimisation method. Klaus et al. [42] devised a stereo correspondence algorithm which uses colour segmentation combined with a self adapting matching score which minimizes the number of reliable occurrences. This method also uses belief propagation to assign a single disparity to each segmented region. Hirschmuller [43] proposed a semi-global matching method based on mutual information and approximation of a global smoothness constraint.

Yoon [44] presented work on a local method for computing disparity. In this method, the window is weighted based on geometric proximity and colour similarity. Darabiha et al. [45] presented an FPGA based local window stereo algorithm. This method obtains sub-pixel accuracy for 256×360 images at 30 frames per second. Their window matching method uses correlation. Klaus et al. [42] devised a stereo method which segments the image before fitting regions with disparities. This fitting is based on interpolating depth values along each region. A self adapting matching score for segments is also used to maximize correspondences. Belief propagation is then used to optimize depth among the regions.

Yang et al. [46] came up with a method which uses super pixel resolution to improve disparity images. First a depth map is computed at a lower

resolution, it is then up-sampled to the same resolution as the corresponding colour image. The depth map is then used as a hypothesis to compute a cost volume, which is bilateral filtered. Then, a winner take all and a sub pixel estimation procedure are performed to estimate the full resolution depth map. This method is shown to improve sub-pixel resolution by up to 100 times compared to previous methods.

Sarkis et al. [47] improved the efficiency of the graph cut global optimisation based disparity algorithm whilst maintaining similar accuracy. Their method involves splitting the global space up using a quad-tree. Each space has an adapted energy function which is minimized using the graph cuts algorithm. Results show this method improves efficiency by up to 3 times over similar works. Wang and Zheng [48] came up with an inter-regional cooperative optimization based stereo correspondence technique. This method uses a local adaptive window based approach to compute an initial depth map. The original image is also segmented using a colour based mean shift technique. The segmented image and the initial depth map are then input into a region based optimization method.

Yang et al. [49] attempted to solve the problem of stereo mapping for texture-less image regions. These regions are known to be difficult because

there is less texture to correlate with. They use colour segmentation and plane-fitting with loopy belief propagation for error correction. Wang and Zheng [48] used a stereo method which uses image regions, plane fitting and segmentation as well as a novel region stereo optimization method. Ernst and Hirshmuller [50] presented a GPU implementation of the semi-global matching technique (scan-line optimization) for stereo correspondence. Bleyer et al. [51] devised a method which extracts disparities and alpha matting information at the same time. Alpha matting is used to generate artificial views for viewpoint effects. This method divides the image up into segments and computes the depth and alpha value for each of these segments.

Bleyer and Gelautz [52] presented a method for generating stereo disparity information from an un-calibrated stereo video. First, the video is segmented into scenes. Next, the two camera shots are calibrated before dynamic programming is used to optimize the disparity calculation. Then the disparity estimates are smoothed temporally in order to achieve robustness to disparity flickering. Hirschmuller and Sharstein [53] presented a survey on stereo depth mapping with respect to radiometric differences. They investigated different metrics used in local matching methods as well as their relationship with radiometric variations. Also investigated were the effects of different filters: Laplacian of Gaussian (LOG), bilateral background sub-

traction, rank, SoftRank, census and ordinal.

In his masters thesis, Olofsson [54] presented a survey and evaluation of various global and local stereo vision methods. He also presented a novel and efficient local method. This technique is shown to achieve state-of-the-art results with some data sets. Bleyer et al. [55] introduced a method which models stereo images using smooth surfaces. This technique is based on the assumption that the entire scene is composed of a few smooth surfaces, and each pixel is a part of a surface. Colour segmentation is used to estimate different surfaces. Bleyer et al. [56] presented a variation on the Patchmatch algorithm (PMA). PMA is a global optimisation method which models each pixel as a plane in 3D space. Each pixel is set to a random value within a larger region, as long as there is at least one close guess for the planes of one of the pixels, this information can be propagated. The default PMA performs spatial propagation, and uses adaptive support weighting to improve correspondence around the border. The method by Bleyer et al. introduced view and temporal propagation to the original PMA.

Bleyer et al. [57] later presented a joint stereo matching and segmentation algorithm. This method models segmented regions as objects having colour and a disparity distribution, they also use a novel 3D connectivity property

for each object region. Lu et al. [58] presented a method for increasing depth map quality given a colour original of the same scene at higher resolution. This allows disparity maps to be computed quickly by computing a lower resolution depth map before scaling-back the resolution. This method makes use of markov random fields and is unique in applying the technique to super pixel resolution in terms of depth mapping. This method is also uses state-of-the-art disparity computation algorithms and so improves efficiency whilst retaining accuracy.

De-Maeztu et al. [59] presented a novel cost aggregation step for computing disparity images from a stereo pair. Their method works similarly to weighted pixel and scalable window routines but has complexity independent of window size. Unlike other cost aggregation methods, it can be used with colour and includes a novel disparity refinement pipeline. This method effectively filters the image so the pixels are weighted prior to performing some local disparity cost computation. Mei et al. [60] presented a GPU based stereo correspondence algorithm which makes use of cost aggregation followed by scan-line optimization. Results show this GPU based algorithm is among the state-of-the-art.

Mizukami et al. [61] described a novel method to reliably compute dis-

parity cost volumes for sub-pixel depth mapping. It uses a combination of interpolation and an edge preserving filter. First a sub-pixel cost volume is computed, then this volume is filtered. Finally a two step sub-pixel disparity search is performed. Later, Zhu et al. [62] presented a novel regularization method for stereo matching. This regularization method overcomes noise and captures general disparity at higher octaves and between regions. Lee et al. [63] introduced a non-iterative one pass method for improving local stereo methods. To this end, a novel three mode cross census transform with a noise buffer is introduced. This method is used for both stereo image and video calculation. Stereo Video computation also makes use of optical flow.

Chen et al. [64] presented a local windowing based method which uses an adaptive support weight to achieve state-of-the-art local method results. This method uses a novel trilateral filter as a weighting function. The trilateral filter extends the bilateral filter by adding in a component measuring boundary strength. Lu et al. [65] presented a super pixel variation of the Patchmatch stereo correspondence technique. Mei et al. [66] proposed a tree based approach for optimizing cost volumes. Their method first segments the image based on colour, instead of forming a graph between segments, and calculating the minimal spanning tree. A tree graph is created for each segment, then these graphs are linked with the optimization algorithm.

Tan et al. [67] made an improvement to segmentation for use in disparity mapping. Since under-segmented regions contain disparity discontinuities (many methods assume regions have a global or planar based disparity value) and over-segmented regions contain noise, Tan et al. proposed a new segmentation based stereo algorithm. This method makes use of a cost volume watershed algorithm and a new region merging strategy. It detects when regions are under-segmented and fixes the situation accordingly. Their method first computes information from an arbitrary segmentation method as well as an arbitrary local windowing disparity algorithm. This information is fed forward into their cost volume watershed. Using discontinuities in the cost volume, they further segment the regions, then a novel region merging method is performed, this final segmentation information is used for the global belief propagation disparity mapping method.

Yang [68] first developed a non-local disparity calculation algorithm based on using the minimal spanning tree to find an optimal solution using the cost volume. This method is supposedly improved upon by Vu et al. [69]. This other method was designed for robustness to texture-less regions. It formulates the cost volume as a minimal spanning tree search problem. Yang et al. also contributed some software for interactive depth of field effects called

scribble2focus. Tan et al. [70] devised a multi-resolution based approach to disparity selection using cost aggregation over a cost volume. Results show this method performs close to global methods for reduced complexity. Lie et al. [71] posed the stereo correspondence problem in terms of interacting 3D entities. Their solution is aimed at curved feature depth estimation, and so they formulate their cost functions according to this constraint.

Monocular Approaches

Monocular depth generation includes any and all systems which use a single colour or grey-scale camera to generate dense depth data. Dense depth data is often computed using consecutive frames of a video sequence. If a relationship between frames is known, then the depth data may be computed as in other stereo methods by calibrating image pairs first. The most popular technique is the computation of the fundamental or essential matrix in order to perform stereo calibration. Once stereo calibration is performed, generic disparity computation methods (reviewed in 2.1.3) may be used to generate dense depth data.

An overview of computing the essential and fundamental matrix and the role they play in not only computing dense depth but computing camera

pose is discussed in section 2.2.1.

2.1.4 Data Representations

In this section, we describe several 3D data representations which are not only commonly used in 3D graphics and computer vision, but in 3D reconstruction as well. First, the popular mesh representation is described and a few popular methods for compression are surveyed. We also surveyed 3D volume and point cloud data as some other popular alternatives. Some newer representations are then discussed (image based methods and Signed Distance Functions). Finally, the octree, a common data structure used for 3D compression is discussed.

Mesh

Mesh data is popular due to its simplicity and integration into GPU technology. The 3D data is made of connected polygons which in turn are made of vertices. Edge information is typically defined implicitly. An example can be seen in figure 2.4, vertices are represented using black dots, edges by lines, and polygons are labelled F_0, F_1, F_2, F_3, F_4 . Here, vertex data define geometric information whilst edge and polygon data forms topological information.

For processing purposes, polygons are usually triangulated, which means all polygons are sub-divided into triangles. Any mesh may be triangulated, an example of this is provided on the right hand side in figure 2.4. In a typical data representation, vertices are stored in a list and triangles are stored using three references to this list. The number of bits per vertex (bpv) is typically used to measure storage requirements for mesh data.

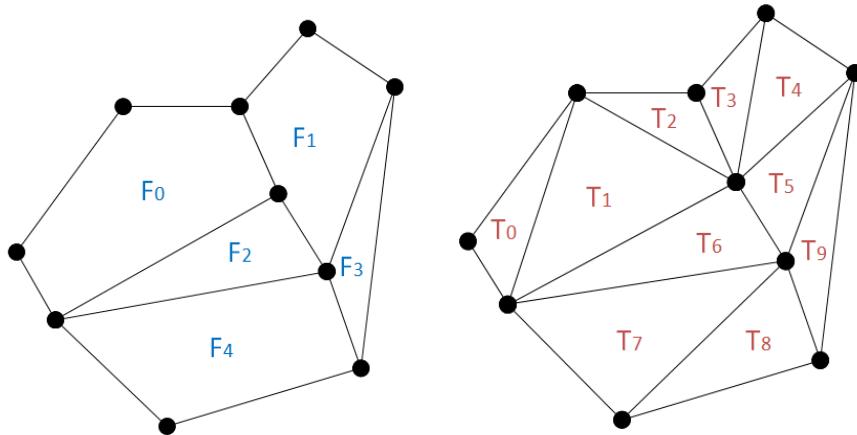


Figure 2.4: Left: Polygonal Mesh, Right: Triangulated Mesh

The Feature-Oriented Geometric Progressive Lossless Mesh coder (FOL-ProM) [72] is a state-of-the-art codec which is progressive. It also aims to be an effective low-bitrate codec. It classifies segments of the mesh as being visually salient or not. Salient segments are preserved more during compression compared to non-salient ones.

Karni and Gotsman [73] proposed a lossy method which compresses a spectral representation of a mesh. This algorithm generally partitions the mesh and compresses each partition separately since it does not work on large meshes. Encoding a basis function for each partition, coefficients are quantized, truncated and entropy coded. Results show this method outperforms the previous state-of-the-art valence method [74] at coarse quantization levels. Bayazit et al. [75] also developed a progressive method based on spectral compression. This method is based on the region adaptive transform in the spectral domain and is advertised as a current state-of-the-art lossy 3D data compression method.

A lossy wavelet based compression system was proposed by Khodakovskiy et al. [76]. This technique samples the mesh, and uses the wavelet transform to decorrelate the data. Coefficients are quantized and stored in a structure called a zero tree which increases compression performance. This method is also shown to outperform the valence method. Other wavelet approaches [77, 78] also sample the mesh and use a multi-resolution representation in which the data is described using local normal directions on the mesh surface.

Point Cloud

The point cloud structure stores a list of 3D points. This representation can be thought of as discrete samples of the surface of a real 3D object.

Figure 2.5 shows an example. This structure can be sampled using a variety of methods. These methods include both dense and sparse sampling, and sample steps can be either regular or irregular. Along with each vertex, a variety of attribute information can be stored. Point cloud data may be obtained via a 3D scanner or RGB-D camera.

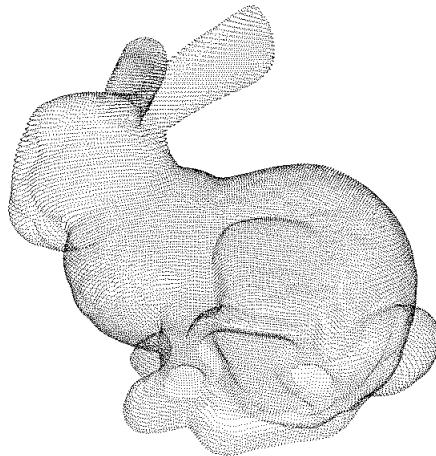


Figure 2.5: A densely sampled point cloud of the Stanford Bunny.

3D Volume

In this representation, points are sampled into a 3D cubic space of sub-cubes called voxels. Such a space may have separate dimensions for width, height and depth or the space may be a true cube. In the context of 3D reconstruc-

tion, this data-representation is common and is used to store boolean values describing the occupancy of a space [79]. A visualization of a 3D volume showing a reconstruction is shown in figure 2.6. This data type is useful because it allows for quick updates. Downsides include, large storage space and the fact that 3D space represented cannot be dynamically changed without significant cost.

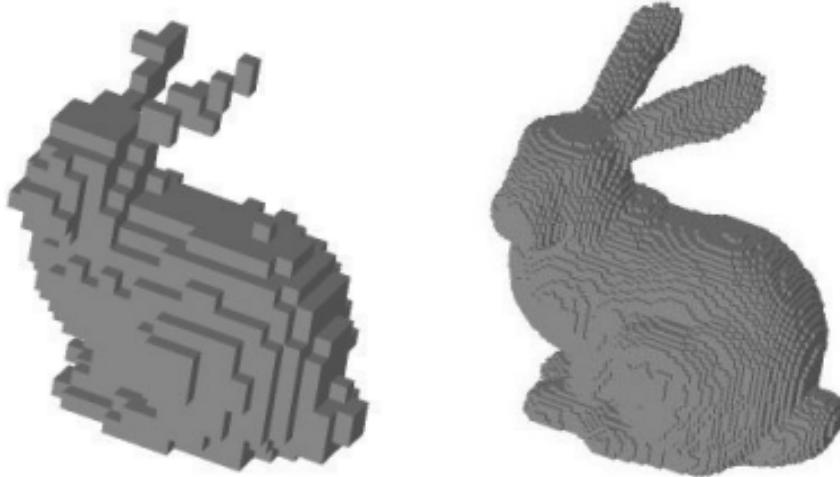


Figure 2.6: Left: Bunny sampled into a 32^3 volume, Right: Bunny sampled into a 128^3 volume. Both images credit: [80]

Signed Distance Functions

A Signed Distance Function [81] is a function which describes a 3D geometric space. Such a function takes a 3D location as x, y, z coordinates as input and returns a single number value. The value describes the geometric detail of a

particular object. Zero values are surface interfaces, positive values increase relative to the distance to the nearest surface and negative values represent the interior of the object.

SDFs may take the form of an equation or may be made discrete by means of discretization. In the context of 3D reconstruction we refer to the discrete SDF, an example of which is show in figure 2.7. The SDF may be visualized by converting it to a mesh and rendering (by means of the marching cubes algorithm [82]) or by directly ray-casting the structure [83]. In the case of ray-casting, there is a significant advantage over the volumetric representation, that is the step size may be adjusted dynamically reducing render time.

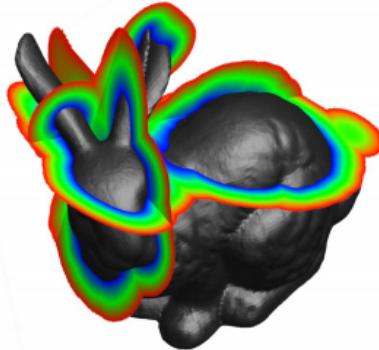


Figure 2.7: Distance Field Slices of the Stanford Bunny [84].

Canelhas [85] did a masters thesis on an approach for camera tracking which makes use of an SDF. Similar to the work by Bylow et al. [86] the

project concentrates on object detection and recognition in an SDF although little evaluation was performed. Additionally storage was not considered. Kubacki [87] proved the SDF is useful in estimating camera pose but only showed proof using synthetic data performing no comparative evaluation. Ren and Reid [88] demonstrated SDF based object tracking based on prior known models.

Elfes et al. [89] use a Bayesian probability of occupancy measure to decide if a point should be added to the grid and showed that SDFs may be used to fuse partial depth scans whilst impeding problems with mesh based reconstruction algorithms. The SDF representation was modified by Zach et al. [90] to be more robust to noise. Bylow et al. noted that the SDF may be used to produce globally satisfied reconstructions in real time.

Image Based Representations

3D reconstructions may also be represented by means of elevation maps [91] and multi-level surface maps [92] however these methods cannot store known and unknown areas of occupancy in a volumetric way. Additionally compression is often not considered and these methods do not have the search capabilities which the octree and volumetric representations have, nor the

fine detail available in a mesh or point cloud representation. An example of an elevation map is shown in figure 2.8.



Figure 2.8: A Height Map of the Stanford Bunny.

Gu et al. [93] devised a solution for representing 3D models as 2D images which are then compressed using state-of-the-art image compression methods (based on wavelets). To form this representation, the mesh is cut along a network of edge paths, opening the mesh into a topological disk, which is then sampled onto a 2D grid. Each pixel in the image has a corresponding coordinate in the model, with pixel neighbourhoods describing connectivity. Comparisons with the method by Khodakovsky et al. reveal the geometry image codec does not have as high compression performance.

Octree

The octree is a hierarchical data representation ideal for storage, search and processing of 3D data. Other hierarchical structures exist (such as K-D tree and BSP-tree) [94] but these are not as useful for compression as the octree, which is the aim of this research. The octree is described in detail below in section 3.2.1. An important technique based on the octree and used by many 3D reconstruction methods is the Octomap [95]. It essentially records a volume occupancy grid using an octree. This method models data probabilistically whilst simultaneously reducing memory size. The Octomap representation is lossless and can reduce the file size of the reconstruction by up to 50%.

Other methods also explore the octree for 3D reconstruction and SLAM [96, 97, 98] however these methods do not really address storage advantages.

2.2 3D Reconstruction Techniques

This section introduces several techniques used in SLAM and 3D Reconstruction for estimating pose. As mentioned, both of these research areas employ techniques to compute camera pose. Once camera pose is known, in the case of RGB-D methods, depth data may be integrated to generate dense 3D re-

constructions. In the case of monocular or stereo approaches, dense depth information may be computed implicitly to varying degrees of accuracy. Once computed it may be integrated in a similar manner to RGB-D based methods.

There are many examples of camera pose being computed by robots [99, 100, 101, 102, 103, 104, 105], in these cases, pose may be estimated by taking input from sensors (measuring wheel spins or thrust) [106]. It can also be computed via more complex methods based primarily on visual data [107, 36, 108, 109, 110].

If only monocular video data is available, the true scale of the map cannot be determined so true measurement based reconstructions are not possible [111, 112, 113]. Furthermore many monocular algorithms only produce sparse reconstructions [107, 36].

The first monocular slam system was presented by Davison in 2003 [107]. Davison’s method uses a hand-held camera in real time to produce globally consistent sparse maps. It makes use of probabilistic filtering in its estimates of both camera pose (translation and rotation) as well as triangulation of sparse features. It was successful but is limited to in-door office environments because it requires large state vectors which grow with scene size.

Another known limitation is that the use of sparse feature maps leads to poor accuracy.

Later, systems which split tracking and mapping (global optimization) came along such as the Parallel Tracking and Mapping Algorithm (PTAM) [36]. PTAM also performs camera tracking in real-time in small work spaces. It is essentially a bundle adjustment (a least squares solution to camera and feature optimization see section 2.2.7) based pose estimation procedure. The tracking system runs in parallel at frame-rate speeds, performing robust n-point pose estimation with feature matching. In comparison to filter based methods, much more features can be packed into the map [108]. Approaches such as these typically accumulate drift (such as work by Beardsley et al. [114]) or perform off-line loop-closure optimization.

2.2.1 The Fundamental Matrix and its Properties

The Fundamental matrix method is especially useful in systems where monocular cameras are used for 3D reconstruction. From 2D point correspondences, the fundamental matrix may be computed. This leads to both the direct estimation of camera pose as well as stereo calibration parameters (which are required to compute dense depth information from monocular views). Here

the Fundamental and Essential matrices, their estimation, properties and use in pose estimation are described.

To describe this technique, linear algebra is used. Note, all of the transforms required to represent imaging systems can be represented using 4×4 matrices and homogeneous 4×1 vectors. These important transforms include: 3D rotation, scaling, shearing/skewing, mirroring, translation and perspective transforms. Rather than describing camera capture using projecting rays, linear algebra provides further capabilities and clearly defines the estimation process of the fundamental matrix.

In figure 2.9 there are two camera systems. Next an explanation as to how point Q is projected from 3D onto the 2D point Q_1 is discussed. First, C_1 is defined as the origin (later this allows rotation and perspective transforms to more easily be performed). In order to do this, C_1 can be subtracted from itself (to become the origin) and so it is also subtracted from Q as well. To this end, a 4×4 translation matrix, T is used. Next, rotation is performed in order to align the camera's axes with the x, y and z axes. The camera's axes can be defined using three vectors. One pointing directly ahead where the camera is facing (piercing the centre of the projection frame), another pointing directly to the right perpendicularly (orthogonal to the first), the final points above the camera, aligned orthogonally with the previous two vectors.

These three axes can be placed into the columns of a 4×4 homogeneous matrix. This forms a matrix which rotates an aligned camera's axis to face the direction where C_1 currently points to. In other words, it performs the exact opposite of what is needed. Since this matrix is orthogonal (known because the column vectors are all perpendicular) the inverse transform is simply the transpose of this matrix. This rotation matrix is defined as R . Next, there may be a lateral alignment, this is a translation which further aligns the points to the centre of the frame. This is performed using another 4×4 matrix, L . Finally, another 4×4 matrix, P is used to project the point Q to the 2D point, Q_1 . The entire projection can be performed by multiplying these matrices together, $\text{Projection} = P * L * R * T$. Here, $P * L$ are known as the intrinsic parameters whilst $R * T$ are called the extrinsic parameters. Due to the imperfect nature of the camera lens, the intrinsic camera matrix often has distortion, this becomes important later when we define the difference between the fundamental matrix and another matrix introduced, the essential matrix.

Using the steps described and looking at figure 2.9, it has been identified how 3D point P is mapped to both of the frames with cameras C_1 and C_2 . Projecting point P onto the two frames gives points P_1 and P_2 respectively. V_1 is defined as a vector relating P_1 and P_2 . It is formed by normalising the vector from C_1 to P , and V_2 (normalizing $(P - C_1)$). The relation be-

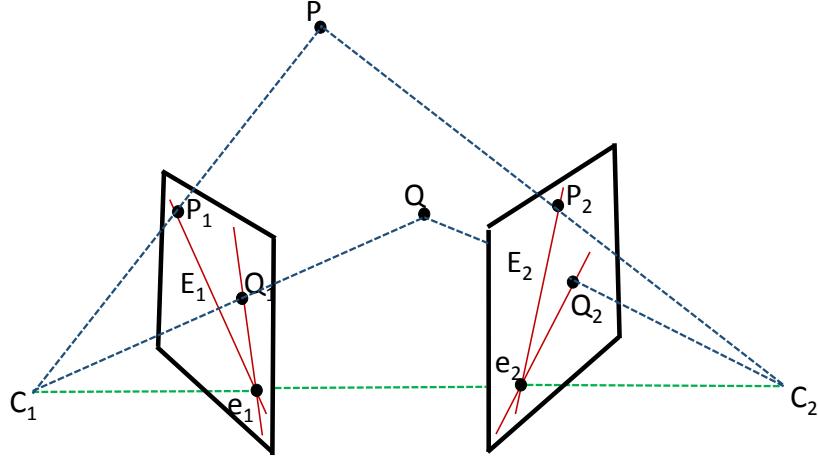


Figure 2.9: A pair of cameras viewing two points. This figure is a reconstruction of the figure on page 492 in Computer and machine vision: theory, algorithms, practicalities [115].

tween these vectors is the essential matrix [115], E . The specific relation is $V_2^T * E * V_1 = 0$. The depth of P is unknown and cannot be estimated directly unless an investigation is performed on the particular camera system in use. The precise perspective transform and associated distortion must be known. Fortunately, both the depth and the perspective are cancelled in this matrix formulation, and so we can specify the relation using only the 2D points, $P_1^T * E * P_2 = 0$.

This formula is based on the assumption that no distortion is present in the camera system. In this case, both of the cameras may have distor-

tion. We represent this distortion here using matrices G_1 and G_2 . If both shots are taken with the same camera, it is assumed that G_1 is equal to G_2 . We therefore relate the theoretically precise P_1 and P_2 with their real-world equivalents (which have distortion) D_1 and D_2 by $P_1 = Q_1^{-1} * D_1$ and $P_2 = Q_2^{-1} * D_2$. Inserting this into the essential matrix formulation we have, $(Q_1^{-1} * D_1)^T * E * Q_2^{-1} * D_2 = D_1^T * Q_1^{-1T} * E * Q_2^{-1} * D_2 = 0$. The presence of this distortion is the difference between the essential and fundamental matrices with the fundamental matrix being $F = Q_1^{-1T} * E * Q_2^{-1}$. If no distortion is present, the fundamental matrix is equivalent to the essential matrix.

Using the Essential or Fundamental matrices, we may compute both camera pose, and stereo calibration. The camera pose is computed by using singular value decomposition (SVD) on the essential matrix. This breaks it down =into 3 matrices (W , U and V). Then, both the translation part (equation 2.1) and rotational part (equation 2.2) of the camera pose may be computed.

$$TranslationMatrix = W \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} W^T \quad (2.1)$$

$$\text{RotationMatrix} = W \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} V^T \quad (2.2)$$

Looking at figure 2.9 again, line E_1 represents an epipolar line, this line lies on the plane in which both cameras and point P lie on. Line E_1 is the epipolar line belonging to the left frame and E_2 is the epipolar line belonging to the right frame. Point e_1 is the epipole of the left frame and point e_2 is the epipole of the right frame. Epipoles are the projection of one camera's location onto the frame of another camera. The projection of C_2 onto the left frame by camera C_1 is the epipole e_1 . If the fundamental matrix is multiplied by a particular point it produces a vector in R^3 which represents the epipolar line as $[abc]^T$ in which a, b and c represent the line equation $ax^2 + bx + c$. The epipolar line corresponding to Q and Q_1 passes through a common point with the other epipolar line E_1 . All epipolar lines intersect at the epipole.

The fundamental matrix may be computed using point matches between images. As mentioned the fundamental matrix can be used to perform image rectification and estimate camera pose. As shown, the fundamental matrix, epipolar lines and epipoles can be estimated from point correspondences. If the epipoles are known, then the direction in which the other camera resides is also known.

The basic pipeline for the fundamental matrix method begins with feature matching. Using matches, the fundamental matrix is estimated usually using an outlier filtering strategy such as Random Sample Consensus RANSAC. Then extrinsic camera information is estimated and images are rectified and so that disparity information can be computed. This may be used to compute dense depth data required for 3D reconstruction. One downside to this method is that the scale of the translation computed is not to scale.

Monocular Feature based SLAM systems use feature matches to estimate camera pose and location changes across frames [116]. Variations of this method use different features including: corners and lines [117], image patches [118] and exemplar feature matching [119]. SIFT features are used most often in SLAM [120, 121, 122, 123], in addition FAST features have been explored [124, 125, 126, 127]. Beall et al. [122] made use of both SIFT and SURF features in their underwater SLAM system. Real-time monocular SLAM systems based on this approach have also been proposed [119, 121]. RANSAC is often used in monocular SLAM [123, 124, 126, 127, 128] to remove outliers. Without RANSAC, incorrect camera parameter estimates would prevent any accurate pose estimation in all but synthetic data. Bundle adjustment is also used as an additional step to refine camera parameter

estimation [123] (section 2.2.7).

2.2.2 Feature Matching and RANSAC

Feature matching together with RANSAC may be used to compute the Fundamental and Essential matrices which leads to camera pose estimates. The process of estimating the Fundamental matrix simple, but in practice noise often makes accurate estimation difficult. Furthermore, the intrinsic camera parameters must be known prior in order to apply such a technique, else noise decimates the ability to accurately estimate camera pose and calibrate frames for stereo correspondence.

Another, more common way to use feature matching with RANSAC [129, 130] is to compute the camera pose directly using multi-frame RGB-D data. Alternatively (as mentioned in section 2.1.3), dense depth information may be recovered using monocular and stereo camera set-ups as well.

This method first computes feature matches between two given frames. These matches along with the 3D location of each corresponding match (projected using the dense depth data) are used as input for the popular RANSAC algorithm.

RANSAC is an iterative algorithm which works by repeatedly selecting a subset of input data, computing a model on the subset and testing that model on the global superset. During this repeated process, the subset with the best classification (lowest error or greatest strength model) is chosen for the final model. RANSAC is useful because it is an effective and capable algorithm for filtering out outliers.

In the case of camera pose estimation, the subset is used to compute the camera pose (using singular value decomposition) and the camera pose which minimizes the global error is chosen as the best camera pose. Here we describe how to compute the camera pose given a set of 3D points matched from one frame to another. Given two identical length arrays of 3D points, P and Q , for each point in P , P_i matches best with point Q_i . The covariance matrix X is then computed based on P , Q and their mean values. The calculation for X is shown in figure 2.3.

$$X = \sum_{i=1}^N (P_i - P_{mean}) \times (Q_i - Q_{mean})^T \quad (2.3)$$

Given the covariance matrix X we compute its singular value decomposition matrices W , U and V . Then the rotation part of the pose may be

computed as $R = V^T \times U^T$. The camera movement can be computed by $T = -R \times P_{mean} + Q_{mean}$. If scale must be computed, it can be computed as $S = (Q_i - Q_{mean})/(R \times (P_i - P_{mean}))$ for any i .

Computing 3D reconstructions using feature matching is often preferred as close competitor ICP is often unnecessary and expensive [111]. However, the computation of pose without ICP with feature matching only is still non-trivial because of the following issues:

- Synchronization problems between the RGB camera and infra-red camera shutters cause inaccurate pose estimates.
- Feature matches often occur at depth jumps (the interpolation of computed depth at object boundaries). This also leads to inaccurate pose estimates.

RGB-D SLAM

Endres et al. [111] presented a dense 3D reconstruction technique using RGB-D data from the Kinect sensor based on feature matching and RANSAC. They also evaluated their technique under different illumination and camera movement speed conditions. It is able to operate at near real time speeds in small in-door environments. This method begins by feature matching RGB images across frames. Using the projected 3D points available for each pixel

in the depth map, RANSAC [129, 131] is used to compute the camera pose across frames. These poses are optimized globally using the g^2o graph optimizer (see section 2.2.7), the Octomap representation [95] is used to voxelize the 3d points before storing them in a volumetric occupancy map. Since this system is based on feature matching, Endres et al. evaluated three different feature matching techniques: SIFT [11], SURF [29, 30] and ORB [27]. ORB feature matching is used because it is faster than SIFT and SURF with slightly less accuracy, the authors also evaluate a GPU implementation of SIFT [13]. Endres et al. evaluate their algorithm using their own benchmark [132]. Results show it can handle up to 50 degrees of rotation per second, and speeds of up to 43 centimetres per second.

This system is divided into a front-end and a back-end. The front-end performs feature detection, matching and sensor pose estimation via the RANSAC method whereas the back-end performs non-linear pose optimization using the g^2o optimization procedure and integrates these results into an occupancy grid based on the Octomap. A diagram of the combined front-end and back-end systems is given in figure 2.10.

The front-end uses OpenCV [133] for feature matching. During the feature detection process, the Hessian threshold is used to keep the total number of features constant. This is required because pose may not be able to be

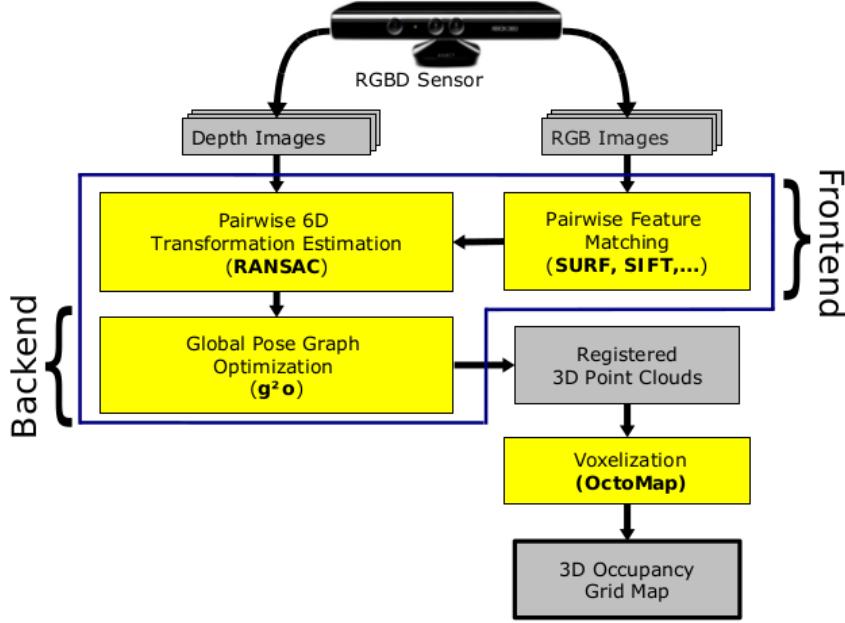


Figure 2.10: RGB-D SLAM Pipeline used by Endres et al. [111]

computed if there aren't enough features, on the other hand, too many features typically leads to many false positives in the feature matching schema. This results in sub-par performance due to a likely lengthier RANSAC process.

As mentioned, these matched features (and thus matched 3D points) are used with RANSAC to compute pose [134]. During the RANSAC process, corresponding points with distances below 3 cm are considered inliers. The inliers are then used exclusively to compute a finer pose. This method of pose estimation is fast but the computational complexity depends on the

number of features computed. For each frame, the pose relationship with 20 previous frames (including the most recent 3) is computed in parallel on the GPU. Then the final computed pose is given to the back-end. If accurate pose cannot be computed, constant motion is assumed.

Fast Odometry from Vision: FOVIS

Another technique which is based on feature matching is the FOVIS (Fast Odometry from Vision) technique [135]. FOVIS works in 6 stages. In the first two stages, the RGB-D image data is collected and put through a low-pass filter, Gaussian scale space is computed and then FAST features [25, 26] are computed. In the next stage, initial camera rotation estimation is computed by minimizing the sum of squared errors between down sampled versions of consecutive image frames. In the fourth stage, feature matching is performed, using the SAD (sum of absolute differences) metric and sub-pixel resolution. In the fifth stage, outliers are removed using a feature match graph and a greedy algorithm to find maximal cliques [136, 137].

In the final stage, the full camera pose is estimated by first computing it using Horns method [138] This technique minimizes an error function in computing the camera pose. The FOVIS method then optimizes this by min-

imizing the re-projection error using a non-linear least squares method.

Whelan et al. [139] extended Kinectfusion allowing it to map larger areas. They compared ICP with FOVIS and found that FOVIS contributed less drift but lacklustre models compared with ICP. Whelan et al. also proposed a technique for incrementally creating a triangle mesh as the reconstruction was built. This method uses multi-threading, and uses pose graphs for further optimization (see section 2.2.7).

2.2.3 ICP

Iterative Closest Point (ICP) [140, 141, 142] is an iterative optimization method used to register 3D data. It works by iteratively minimizing registration error given two sets of point clouds and is a popular method for estimating 6-dof (6 degrees of freedom) camera pose. The algorithm works by computing the closest point using some metric (usually Euclidean distance) for each of the points in the first point cloud. Then computing a transform which minimizes either the Euclidean distance error or some other error metric ([143, 144, 145, 146, 147].

The points are then transformed by the computed transform and this process is repeated which is supposed to iteratively minimize registration er-

ror. The 6-dof (3 axes of rotation plus 3 axes of translation) are commonly computed using the method described in section 2.2.2. Various distance metrics have been researched including the point-plane metric [146]. This metric improves convergence rates and is used for surface reconstructions which contain additional information about the normals of each points. This algorithm is highly successful in generating accurate 3D reconstructions but it does have a few issues.

The first issue is that whilst computing the best transform using ‘feature matches’ (which are nearest points between consecutive frames) is a simple concept, computing these closest points for each input point is expensive. This can be improved using the projective data association algorithm [148] usually used by exploiting a projected version of the 3D data. The efficiency issue can also be improved using a coarse to fine scheme.

Another issue is that ICP is limited to small rotation, translation and scale. If larger transforms are present (especially scale in the case of general registration) ICP will usually get stuck in a local minimum and fail to register [149]. The final issue is that of slippage [150]. ICP tends to fail in environments with little texture. Here we present some important algorithms which use ICP in 3D reconstruction.

Feature Matching, RANSAC and ICP Refinement

Some systems make use of feature matching with RANSAC (see section 2.2.2) for pose estimation and ICP for further alignment [151, 39]. These methods are very similar and make use of the advantages of both the faster and more robust feature matching method and the more accurate ICP optimization inclined method.

Henry et al. [39] presented work on a 3D mapping method which combines Feature Matching and RANSAC (similar to [111]) with ICP. This method makes use of an RGB-D camera, this sensor input for sparse feature matching. Features are used with RANSAC and the technique described in section 2.2.2. ICP is used for refinement of the initial prediction. If loop-closure is required, a constraint is added to the 3D pose graph [152] and is used to close the loop. A Surfel [153] volumetric fusion method is used to represent and store the 3D reconstruction. This technique may be used for robot localization and path planning [154]. Endres et al. [111] compared this method with theirs using a standard benchmark [155].

Non-Rigid Alignment

In addition to rigid alignment ICP is used by Pauly et al. [156, 157] to align depth maps using non-rigid transforms. In the method by Brown and Rusinkiewicz [157], ICP is used to first align the point clouds using a rigid alignment, then global feature positions are found using a relaxation method. Finally, 3D point sets are warped to final positions using thin plate splines. Brown and Rusinkiewicz state that this outperforms rigid based alignments.

Kinect Fusion

Newcombe et al. [158, 159] proposed an accurate, real-time dense 3D reconstruction algorithm which works well on complex indoor environments. This algorithm computes relationships between depth map frames generated using the Microsoft Kinect [35] sensor. By aligning depth maps this method is capable of tracking both camera pose and location as well as generating dense 3D reconstructions. Only depth data is used in alignment computations and because the Kinect is a structured-light based depth sensor Kinect Fusion works under any lighting condition, including complete darkness. Since this method uses the Kinect and GPU (both considered commodity) the technique may be considered inexpensive.

The method works by computing camera pose, transforming depth data

by a computing pose (frame-by-frame) and fusing this data into a global surface volume. It uses a coarse to fine grain ICP algorithm to compute camera pose. During the ICP phase, the target points come from the entire globally matched previous frames. Therefore, this method is considered global rather than frame-to-frame based. Such a method has direct advantages over frame-by-frame feature matching, since all data is used to compute pose. The downside, is that this method fails to find global solutions due to the nature of ICP. This may occur when some frames must be skipped due to motion blur, or the camera passing over surfaces which do not reflect infra-red light.

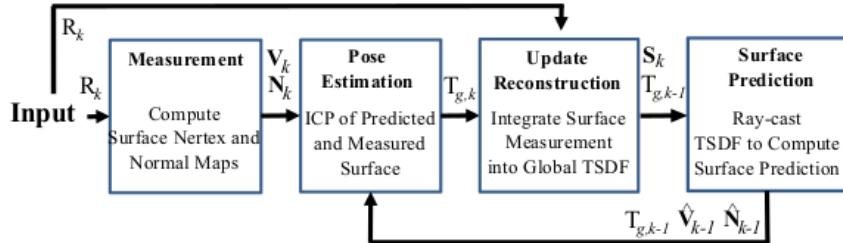


Figure 2.11: Kinect Fusion Algorithm Pipeline [158]

Kinect Fusion has four main steps as illustrated in figure 2.11. The first step performs pre-processing on the depth data as well as generation of additional information for use by Kinect Fusion. Each depth map frame is first passed through a bilateral filter. From this, a dense vertex map (map of 3D points projected using a known projection matrix associated with the Kinect)

is generated, as well as a normal map. For both the vertex and normal maps, a 3-level image pyramid is constructed. This makes the coarse to fine grain ICP technique possible.

Next, the dense coarse to fine grain ICP is used to compute pose between the fused frames and the current frame. The authors exploit the fact that the transformation between frames is small because camera motion is slow when computing against every frame. With coarse-to-fine grain ICP they use projective data association [148] and the point-plane metric for pose optimization [79]. The ICP based pose estimation computes pose given both a predicted and measured depth map.

After estimating camera pose relating to the globally fused model, each frame must be integrated into that model. The Kinect Fusion algorithm uses a truncated SDF (TSDF) representation. This is a SDF volume where the distances for each voxel are capped by some value. The TSDF uses a volume resolution of $512 \times 512 \times 512$. They use the TSDF, rather than relying on a linear but accurate discrete SDF transform [160] because of the computation required in calculating the discrete SDF for volumetric data.

As mentioned, Kinect fusion uses pose prediction and fuses each depth

map into the TSDF representation. In this way, they align and fuse each depth map to the global 3D reconstruction. Therefore, global loop closure is not required. This may have a negative side effect as some frames which have larger resolutions may be heavily quantized in order to be fused with the SDF, especially thin surfaces. These features may also negatively impact ICP in estimating pose.

FOVIS and ICP Integration

Whelan et al. improved [150] upon their previous method [139] by integrating FOVIS and ICP together on the GPU, and adding an advanced colour fusion model to the technique. The primary improvement lies in switching between FOVIS and ICP whenever the error using FOVIS is too high, ICP is used. They also integrate using a global model rather than a local one.

2.2.4 Direct Optimization Methods

Optimization over a Signed Distance Function

In 2013, Bylow et al. [86] presented a novel method which reconstructs static indoor environments in real time using RGB-D data captured using the Asus Xtion Pro Live sensor. Their system is able to generate accurate 3D RGB

coloured models of the environment in real-time by optimizing for 6-dof in terms of accurate projection of new depth map frames into an existing global SDF model. Their method uses Gauss Newton optimization over a SDF representation on a laptop with an NVIDIA GPU. Unlike Kinect Fusion [158], this method optimizes directly in the SDF. Camera pose is computed by finding a rotation and translation which minimizes the error in projecting depth images into the SDF. Compared to Kinect Fusion, this technique is more robust and accurate. It compares favourably to bundle adjustment but is much faster for small to medium sized scenes. Results are generated using the TUM RGB-D benchmark and SDF volume sizes 256^3 and 512^3 are used in evaluations. The authors note the algorithm may be able to handle large scale scenes if used in conjunction with other techniques [161, 152].

This technique is efficient because computing the error is performed via fast look-ups in the SDF. This is possible because the SDF explicitly captures distances from each voxel to the global model's surface. Because of this, the algorithm can easily work within a global space rather than frame-by-frame. Using the SDF to lookup depth map projection error, the camera pose is iteratively estimated and then the depth map is integrated into the SDF and colour information is stored in another volume. The pose estimation procedure begins by storing the first frame as a volumetric SDF. Then for

each new depth frame, camera pose is computed, and based on this pose the frame is projected into the scene. Using a lie algebra based 6-dof model [162] envisioned as a vector in R^6 representing camera pose, the error for a given pose may be computed as the squared error of the depth map transformed by the pose and projected into the signed distance function. Due to noise or missing data within the depth frame, this error may never be reduced completely, instead the best pose is iteratively computed using this model and the Gauss Newton non-linear optimization algorithm.

The SDF representation uses two volumes as in [81], one volume stores the average distances, the other stores the cumulative weights for each voxel. Bylow et al. use these weights to handle occlusion and sensor uncertainty. When integrating a point into the SDF, tri-linear interpolation is used between eight neighbours to handle point coordinates made up of floating point numbers. During integration, each voxel is projected onto the image plane rather than ray cast from the centre of projection as in [158]. This ensures that each voxel is visited once when updating the SDF, whereas in the ray casting approach, this may not necessarily be the case.

In computing the SDF for a given depth map, the exhaustive marching cubes algorithm is too slow, even the fast marching algorithm [163] is not

suited for real time discrete SDF generation. Instead, the SDF is approximated with either the point to point distance or point to plane distance functions. For final visualization, marching cubes is used [164] on the final SDF. Colour is computed from the colour volume using a technique used by Whelan et al. [150]. Since the method by Bylow et al. is based on optimizing the projection error using the SDF and only uses locations in its pose estimation procedure, it is independent to illumination. Given this, it will also fail in cases where only co-planar surfaces are visible. The authors mention that using colour information during tracking [145] may mitigate these concerns.

ICP + SDF Optimization

Rusinkiewicz et al. [79] developed a 3D reconstruction technique based on frame-by-frame ICP and integration using an occupancy grid. Users scan small objects by rotating them by hand. Output models are optimized using a SDF [81].

Warp Function Optimization

Kerl et al. [165] proposed a dense RGB-D SLAM system which uses a probabilistic camera parameter estimation procedure. Rather than strictly using feature matching, RANSAC and a linear pose estimator are used. This technique formulates the projection error as an image warping function (between

two frames) and optimizes it using Taylor’s expansion.

Event Feature Optimization

Weikersdorfer et al. [166] presented a novel sensor system named D-eDVS along with an event based SLAM algorithm. The D-eDVS sensor combines depth and event driven contrast detection. The event emitting sensor can be thought of as emitting points which may be used like features. After aligning pixels generated from the event transmitting device with depth information from the Asus Xtion Pro Live RGB-D camera, it computes pose by optimizing the re-projection error using least squares optimization.

2.2.5 3D Feature Matching

3D feature matching works much in the same way as the 2D feature matching and RANSAC approach (section 2.2.2) but uses 3D features rather than 2D only, so it can work independently to perspective changes which 2D feature matching methods are only robust to. Many techniques have been proposed [167, 168, 169]. Technically, registering against rigid transformations may be achieved with as little as 3 feature matches. These 3 matches can be found by 3D feature matching or geometric hashing [170].

4-Point Congruent Sets

Aiger et al. [171] developed a method for 3D registration (and pose estimation) named 4-Point Congruent Sets (4-PCS). This method is fast and robust to wide baselines in the registration of 3D data. It is resilient to noise and outliers and there is no pre-filtering or de-noising required prior to registration. It improves the typical feature matching approach by significantly reducing the number of points required to compare as features for use in a RANSAC approach to pose estimation.

This method works by extracting all co-planar 4 point sets from a 3D point cloud which are congruent (under a rigid transformation) to a given set of co-planar 4-points. The complexity of this method is $O(N^{2k})$ where N is the number of points and k is the number of 4-point sets. When there is not too much noise, this method may use feature matching only which brings the complexity down to $O(N + k)$. Aiger et al. also presented an extension to register against affine transforms. This method was tested with varying levels of noise, outliers and overlap.

4-PCS works based on the principal of large numbers, which means solving the largest common point-set (LCP) problem. This problem states that given two point-sets P and Q , LCP under -congruency solves for a subset

P_i of P having the largest cardinality such that the distance between $T(P_i)$ and Q is less than ϵ given an appropriate distance measure (T is a rigid body transformation).

The basic concept is to optimize the selection of the 3 points used as a base for alignment from one point-set to another. By choosing 3 matches, then measuring the distance between $T(P_i)$ and Q_i . This test is of complexity $O(m^3 \times n^3)$ but this can be reduced to $O(mn^3\log(n))$ [172] and even $O(n^3\log(n))$. Despite these improvements, these complexities are still too large for practically sized point-sets. This novel 3D alignment scheme is capable of aligning surfaces with limited overlap and may be refined using ICP [141].

Base Shape Matching

Gelfand et al. [173] presented a method which segments input models into base shapes, simple geometric parts which may be modelled and matched. Using these base shapes, slippable components are found by computing local slippage signatures for a set of points in the input and iteratively clustering regions with matching slippable motions. This method is stable but appears to be limited to mechanical human made parts and shapes.

Multi-scale Features

Extensions to both SIFT and SURF have been extended to 3D [167, 168]. These methods follow the same overall technique as the 2D counterparts however some procedures and practices are altered to better suit 3D data input. First, a 3D-volume scale pyramid is produced for matching at different scales. Then features are detected using the corresponding feature detection method (by SIFT or SURF) and then a feature vector for each feature is produced. Features can be matched between 3D frames and a rigid or non-rigid alignment may be computed.

Li and Guskov [169] developed a 3D feature matching technique based on 3D SIFT for SURFEL (3D point and normal) data. It computes a scale space pyramid using a per point scheme in which the next scale up for each point and corresponding normal is computed as the point which minimizes an error function. Salient features are then computed using the scale space. Features are those points which maximize (or minimize) a measurement based on dot product of the normal and the difference between points at adjacent scales.

Li and Guskov use their technique to find approximate transformations, then ICP is used for further alignment [140, 146]. This technique is slow for data sets with large amounts of input data, yet it does not work well with

sparse data sets.

Mori et al. [174] proposed a matching method based on two levels: a high level matching method which is used to compute a short-list for candidates and a low level method which is more computationally complex. This technique is simply used for matching and uses larger areas rather than local features like SIFT 3D.

Other 3D feature matching techniques which have been developed are based on images [170, 175]. In these techniques features are typically described using some sort of 2D image function rather than a single dimensional vector. Typically, a normal is computed for a feature point and some projection is used to compute an image which is used as a descriptor. Then these features may be matched between 3D volumes using normalized cross-correlation or another image comparison method.

Some feature matching techniques are based on hashing or voting [176, 177, 149, 178]. These techniques are out of the scope of this research and are still underdeveloped.

2.2.6 Principal Components Analysis

Principal Components Analysis (PCA) is an algorithm which divides a set of observed vector data into their principal component vectors and a centroid. The vectors along with the principal axes may be used to align 3D data. An issue with PCA is that in the case of partial overlap, the accuracy may break down [171]. PCA and it's use in the proposed technique is discussed in section 3.1.5.

Pottmann et al. [179] showed that PCA may also be performed on local neighbourhoods. This technique can compute principal directions and curvatures at a given scale which may be used for feature matching. This would be an interesting area of research to pursue but it outside the scope of this work.

2.2.7 Further Optimization

Further optimization of camera pose estimates has also been a popular area of research. Methods which work in global space do not technically require the use of such methods, but others, especially those working on a frame by frame basis require some sort of global optimization procedure. Here we briefly mention two of the most popular camera pose optimization methods

mentioned in the literature.

$$G^2\mathbf{o}$$

The Graph SLAM algorithm defines relationships between camera poses and locations as nodes in a graph. Rather than defining camera pose estimations as truth, they are represented using a Gaussian in which the peak probability is the estimated value. Then using the visible landmarks (visible from multiple views) and the functions for the camera poses the joint graph is optimized to improve pose estimates. Once this is complete, a 3D reconstruction map is produced.

Typically an occupancy grid or octree is used to represent the map [95]. Further detail about the Graph SLAM method is outside the scope of this work, more detail can be found in the original paper on Graph SLAM [152].

Bundle Adjustment

Bundle Adjustment is another optimization technique used in SLAM and 3D reconstruction [180]. In this context, a bundle refers to the light leaving each camera projected into world coordinates. This technique refines both camera pose and visual reconstruction. Bundle Adjustment requires some estimate

for camera pose, 3D world point positions and image point locations for each view as input.

Using these data and any non-linear optimization method (such as Gauss-Newton or the LevenbergMarquardt algorithm) the rigid motion based camera movement and the 3D world-coordinates of the points are optimized. In the literature, Fioraio [181] presented a system which uses bundle adjustment to align RGB-D data. It has also been used in conjunction with feature matching to produce sparse 3D reconstructions [36, 182].

2.3 Summary

As is evident from the current literature that SLAM and 3D reconstruction have a focus on feature matching and RANSAC or ICP. However, these approaches fail when there are too few features, when feature confusion occurs or, when features are non-stationary due to object motion. As the extent of random feature displacement becomes more global the effectiveness of these approaches diminishes. Feature matching also dominates in image registration. Some more current methods are based on optimization over an SDF [86, 79]. These methods promise more accurate results since they find accurate poses using the entire data. In 2D image processing, Fourier based methods have also been shown to register large rotations and scales using all

of the input data [183]. Fourier approaches are also closed form solutions, insensitive to noise (object motion) and scale easily to GPU based algorithms. Accordingly, in this thesis, we propose a novel, closed form Fourier based 3D reconstruction method.

3

Methodology

3.1 Fourier Volume Reconstruction

It is evident from the literature that many algorithms compute satisfactory 3D reconstructions in simple small scale man-made environments. These environments are ideal because there are often suitable levels of texture and features. This works well for both feature matching approaches and ICP. In some environments however, textures are few and far between. These environments often do not produce enough accurate textures for 3D reconstruction or SLAM.

Other environments may have an abundance of similar textures (grassy meadows, out-door environments which include trees, gardens and plants). These environments contain plenty of texture and thus generate many features. Feature matching is difficult though, as features tend to look identical within a local frame.

Some algorithms, such as the method by Bylow et al. (see section 2.2.4), work on finding the optimal solution using all the data. Finding the best solution using all the data mitigates issues with feature confusion and incomplete feature sets. However, this algorithm, as well as other iterative methods for SLAM and 3D reconstruction (Feature Matching + RANSAC, ICP, using the Fundamental Matrix), may still get stuck in local extrema, especially when registering against wide baselines. This is the nature of iterative methods.

In this section I propose using Fourier and phase correlation approaches to compute camera pose as an alternative to iterative and feature based methods. This strategy is novel on its own, but empirical results presented in Chapter 4 suggest it is also highly competitive, outperforming the previously proposed iterative solutions to the camera pose alignment problem given input from various sensor types (stereo and active RGB-D).

In the first section (3.1.1), a description of a basic variation of Fourier-based 3D registration is discussed. This method is capable of registering 3D data with respect to x/y/z axis translation as well as y-axis rotation. In cases where data is scaled to fit volumes or in cases where scale is not known, such as when depth data is estimated from monocular data, scale may also be recovered for registration.

Section 3.1.2 describes the implementation of a SLAM/3D reconstruction method for computing pose via the proposed Fourier Volume Registration (FVR) approach based on the basic Fourier-based registration method in section 3.1.1. This approach makes use of any depth data produced by stereo camera set-ups or active RGB-D sensors such as the Microsoft Kinect or Asus Xtion Pro Live sensor. A pipeline for registration, camera pose tracking and 3D reconstruction is proposed. In Chapter (4), this approach is quantitatively and qualitatively evaluated against other algorithms using stereo and active camera sensors (sections 4.4.3 and 4.4.4), as well as robustness to noise and input captured at wider baselines (section 4.5). Results show this method is capable of robust and accurate registration in the face of noise and that it is highly competitive compared to current algorithms presented in the literature. They also show it produces high quality models

fit for visualization and analytical purposes and can be used with different sensor types.

Section 3.1.3 presents an extension of the FVR SLAM/3D reconstruction method to monocular sensors and data. Such a technique would be useful in situations where depth information is not directly available. Since most portable devices (music players, mobile phones and tablets) contain passive monocular RGB camera sensors, extending FVR to monocular data makes the technique much more accessible. This method's usability is highly dependent on the accuracy of the depth data computed via monocular depth estimation procedures. Results in section 4.4.5 show that this method is capable of registering camera translations near the accuracy of the FVR method given dense depth data input approximating quality associated with either active or stereo camera set-up.

A novel phase correlation technique is proposed in section 3.1.4 as a trade-off between efficiency and accuracy/noise robustness. This technique, named FFVR (Fast FVR), exploits data projection (reduction) in a similar yet simpler way than other techniques which reduce the amount of data to process (such as Principal Components Analysis).

Finally, a novel technique to extend the FVR approach to register full 3D rotation is presented. This method uses the PCA algorithm and is capable of registering 3D data against seven types of transforms including: x-axis rotation, y-axis rotation, z-axis rotation, x-axis translation, y-axis translation, z-axis translation as well as scale. This technique may also be applied to 3D reconstruction and SLAM as well as monocular sensor FVR. Results in sections 4.4 show that the FVR/FVR-3D method outperforms several current iterative techniques in terms of registration error using both stereo and active camera set-ups.

3.1.1 Basic Volume Registration

Fourier Volume Registration within the context of a 3D reconstruction procedure requires two frames of 3D data. These two frames may be input as any 3D representation: mesh, volume, point-cloud, SDF or in a compressed format (e.g. octree). However, the input must be converted to a 3D signal or volume representation prior to phase correlation. Moreover, the size of the volume should be cubic and the data should be scaled to fit. If the two frames are scaled un-evenly then the method must also be registered against scale. This section details some of these issues and others within the context of video sensor input from sensors.

Camera Translation

To describe camera translation, let us say a camera produces 3D scans of a scene from a particular pose. This may be performed implicitly (in the case of stereo or monocular depth estimation procedures) or explicitly (through the use of sensors, i.e. Microsoft Kinect, Asus Xtion Pro Live) and the scans may be normalized into 3D volume frames. If the camera captured frame 1, then moved 20 units to the left, before capturing frame 2, visualizing these frames in conjunction with each-other, it would appear the data was shifted to the right by 20 units (20θ if the resolution of the 3D volume differs by a scale of θ). If the volumes were to be registered (assuming no error), it would be found that the data from frame 1 to frame 2 had been translated by $T_{frame-1,frame-2} = [20, 0, 0]^T$, which can be performed using the translation matrix in equation 3.1.

$$\begin{bmatrix} 1 & 0 & 0 & 20 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.1)$$

The volume data has shifted in the opposite direction. Therefore to register the data, frame 1 may be transformed by the matrix in equation 3.1 to align it with frame 2. Conversely, the camera pose may be updated by

adding $T_{frame-1,frame-2} \times -1$ to the camera's location.

Generalizing this, we define a volume V_1 captured as the first frame and a second volume V_2 which is captured after moving the camera in a particular direction (up, down, left, right, forward or backward) scaled by a magnitude (a vector). The translation from V_1 to V_2 , (x, y, z) can easily be recovered via phase correlation. As described, the camera pose may be updated by $-1 \times (x, y, z) = (-x, -y, -z)$.

Here we step into the detail of phase correlation with deeper insight than the superficial introduction in section 2.1.2 and with a focus on 3D data and pose estimation. Firstly, we define the correlation measure process. Correlation in the context of signal processing measures signal shape similarity between two signals.

In this measure, large correlation values signify a greater similarity in terms of shape, while a smaller value signifies the signals have very different shapes. Note, large negative values signify the shapes may be similar but mirrored. Following the doctrine of DSP, that all signals, no matter what dimension may be processed similarly with similar operations and measures, this technique can be applied to measure similarities between volumes.

The measure of correlation between V_1 and V_2 can be found by shifting V_1 and V_2 's mean values to zero. By doing this, the mean value cannot affect the correlation measure. For example, if the first frame was captured in low light and the second in a brighter light. The resulting values for frame 2 may look like those from frame 1 in terms of values fluctuating about a mean, however the change in lighting may affect the voxel values by increasing them by a uniform value or scaling them by some scalar. Therefore, by subtracting the mean we place the signals in states where by their shape, rather than their raw values may be compared. The procedure can then be completed by summing the element-wise multiplication of V_1 by V_2 . When V_1 has a positive voxel value (a voxel value above the mean) as does V_2 's corresponding voxel in the same position, the multiplication value is positive, and this is summed into the correlation measure. This is the same situation if both voxel values were negative (both below the mean). On the other hand, if one voxel value was positive and the other negative, the correlation measure would be rectified accordingly. Equation 3.2 computes this correlation measure.

$$\sum_{z=0}^N \sum_{y=0}^N \sum_{x=0}^N (V_1(x, y, z) - avg(V_1)) \times (V_2(x, y, z) - avg(V_2)) \quad (3.2)$$

As mentioned, correlation may be used to measure the similarity between

two volumes' shapes. In the context of registration for 3D reconstruction algorithms, it can be used to measure the accuracy of a registration. This again uses V_1 as frame 1 and V_2 as frame 2, given a supposed transform T_{est} to register V_1 to V_2 . The measure of registration may be defined as $correlate(T_{est}(V_1), V_2)$. The measure may be used to compare two registrations, possibly to select the better registration in terms of correlation measure. The two frames may be captured under different lighting conditions and the correlation measure can still be used to select the best transform.

Within the framework of measuring camera pose, the cross-correlation algorithm may be used to measure the correlation values for each possible camera movement, and define the camera location as the location with the largest correlation value. This algorithm is shown in listing 3.1.

Cross correlation is typically used in signal processing to compute the best alignment for two signals. In this context it has been used to compute the best camera location change, notice in the algorithm the *camera – location* variable is set to be the inverse of the translation amount being tested ($[-x, -y, -z]$). Again this is because when the optimal translation value aligning frame 1 to frame 2 is defined as vector v , then the camera location change is equivalent to the inverse. The cross-correlation function in this context may be thought of as an exhaustive optimization procedure which

Listing 3.1: Cross-Correlation based camera location estimation

```
V1 = CaptureFrame();  
//shift camera  
V2 = CaptureFrame();  
highest - correlation = infy;  
camera - location = [0, 0, 0];  
for(z in [0, N]) {  
    for(y in [0, N]) {  
        for(x in [0, N]) {  
            Vtemp = translate(V1, x, y, z);  
            tempMeasure = correlate(Vtemp, V2);  
            if(tempMeasure > highest - correlation) {  
                highest - correlation = tempMeasure;  
                camera - location = [-x, -y, -z];  
            }  
        }  
    }  
}
```

optimizes the camera location change (equation 3.3) in terms of the correlation function.

$$\text{correlate}(\text{translate}(V_1, x, y, z), V_2) \quad (3.3)$$

The range of camera location vectors to test is dependent on the dimensions of the raw 3D input data. If the data is scaled to fit an N^3 space, as is a requirement when using Fourier based techniques, the camera location values along the x, y and z axis would range between 0 and N . To compute the optimal camera location change using the phase correlation method would therefore have complexity N^3 . There are N^3 values to test, and the correlation function must be called for each iteration with a complexity also equivalent to N^3 , $N^3 \times N^3 = N^6$. This is significantly complex, even for smaller volume sizes. Note that the smaller the volume size, the faster the algorithm, however the more quantized the original data is. The more the input data is quantized, the more the camera location change estimates will be restricted in terms of precision. Accuracy will also be affected as quantization introduces unwanted effects of its own. The answer is to use the properties of the Fourier transform to speed up the process.

The 3D Fourier transform transforms a volume from the 3D spatial do-

main into the 3D frequency domain (see Figure 3.1). The frequency domain is a complex valued volume made up of sinusoids. Each voxel in the frequency domain represents the magnitude, phase and direction of a wave. The importance of these properties for computing camera pose will be discussed upcoming sections. An efficient software implementation of the Fourier transform, named the Fast Fourier Transform (FFT), can compute the frequency domain given N-dimensional data.

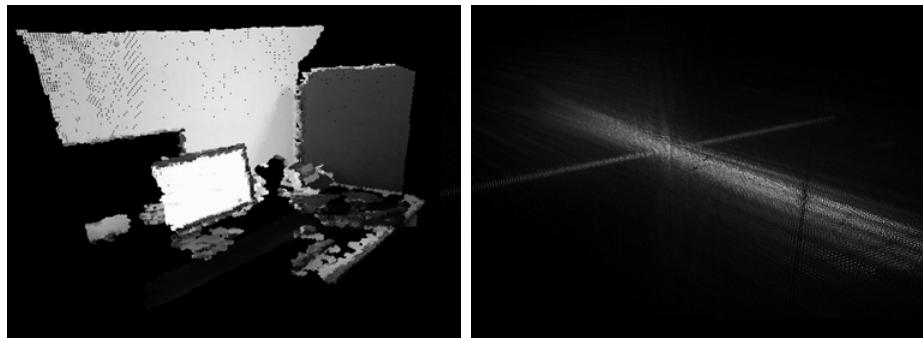


Figure 3.1: Left: Captured 3D frame, Right: Magnitude values in the frequency domain of the captured 3D frame

Exploiting the properties of the FFT and the frequency domain, cross-correlation may be carried out efficiently to compute the optimal camera location change between frames. In the 2D approach, this algorithm is named phase correlation and is a popular approach in 2D image processing to align two images. Applied to two 3D frames captured by sensors (or partially generated via software) it may also be used to register the frames for 3D

reconstruction as well as compute the translation part of the camera pose change for SLAM algorithms. This procedure is defined here as a function named *PhaseCorrelation* (Eq. 3.4). This function takes two frames (3D volumes) as input and returns the best translation alignment between them in terms of maximizing correlation.

$$T(x, y, z) = \text{PhaseCorrelation}(V_x, V_y) \quad (3.4)$$

The *PhaseCorrelation* consists of four steps. In the first step the two captured 3D frames V_1 and V_2 are transformed into the frequency domain using the Fast Fourier Transform. This computes two new volumes $F_{1_{x,y,z}} = FFT(V_1)$ and $F_{2_{x,y,z}} = FFT(V_2)$. Applying the Fourier Shift Theorem to the context of camera pose estimation, if frames V_1 and V_2 are separated by some camera translation, then $F_{1_{x,y,z}}$ and $F_{2_{x,y,z}}$ will have phase values shifted relative to each-other. The normalized cross power spectrum (equation 3.5) of the two complex valued Fourier spaces may then be used to find the camera translation change. The normalized cross power spectrum of $F_{1_{x,y,z}}$ and $F_{2_{x,y,z}}$ is another complex valued volume of the same size.

$$F_{3x,y,z} = \frac{F_{1x,y,z} \circ F_{2x,y,z}^*}{|F_{1x,y,z} \circ F_{2x,y,z}^*|} \quad (3.5)$$

In equation 3.5, \circ is an element-wise multiplication and $|x|$ is the magnitude function or absolute value function. The Inverse Fast Fourier Transform ($IFFT$, FFT^{-1}) may then be used on the output of the normalized cross power spectrum volume to produce a phase correlation surface. The peak value on the phase correlation surface represents the optimal value of the correlation function applied to the two original real valued 3D frames. Therefore, the *PhaseCorrelation* procedure is equivalent to the cross-correlation procedure, but rather than being complexity N^6 , phase correlation reduces the complexity to approximately $6N^3 \times \text{Log}(N)$.

Due to the nature of camera capture, if the camera is translated by some vector before the capture of the next frame then the first frame contains data which the second frame does not. Conversely, the second frame will capture some other data which is not present in the first frame (unless the camera was not moved). The parts of the 3D frames which do not overlap cause noise on the phase correlation surface. This makes it more difficult to decipher the location of the peak. There may be several peaks to choose from, or no clear peak. The nature of computing the Discrete Fourier Trans-

form affects peak estimation. The Discrete Fourier Transform assumes the output Fourier space is an infinitely repeating signal. The cross-correlation procedure on the other hand does not assume this, so therefore the phase correlation surface may be affected. The solution adopted by many is to filter the volumes prior to transforming into the frequency domain. This can be done using a Hanning window function (equation 3.6). This function can be used to filter edge effects prior to computing the Fourier transform which helps to reduce noise on the phase correlation surface.

$$Hanning(x) = \frac{1}{2} \left(1 - \cos \left(\frac{2\pi x}{\frac{N}{2} - 1} \right) \right) \quad (3.6)$$

Noise on the phase correlation surface may also be present if other types of transforms are present. In this case, the true camera translation may be lost. Therefore, camera pose/rotation must be computed prior to computing the camera translation. Other artefacts which may produce noise on the phase correlation surface include moving objects. If an object is present in one scene and not the other, it will cause some noise to be present in the phase correlation surface, making finding the peak more difficult but not impossible. Alternatively if there is an object whose location is changed between frames, the same thing may happen. The phase correlation procedure

should already be robust to these artefacts but filtering approaches may also prove useful in selecting the correct peak which optimizes the translation used to register the 3D camera frames.

Camera Rotation and Zooming

SLAM and 3D reconstruction algorithms must compute both the camera location changes and camera pose. SLAM must compute the pose to track the camera, 3D reconstruction methods may simply register the data for integration or compute the camera pose prior to alignment and integration. Computing the pose involves estimating the camera's post change (rotation) from one frame to another. This may take the form of a 4×4 rotation matrix or the explicit magnitudes of rotation for each axis in a pre-determined order. These two formats may be converted between each other. The pose may also be represented by three axes (three orthogonal vectors) which represent the coordinate space of the camera.

Another transform often not covered in the 3D reconstruction literature is that of scale. Given a single camera moving in a 3D space, scale is of no concern. However, if frames are zoomed or data is computed across sensors, scale factors may be unknown. For example, if an RGB-D image is captured

as frame 1 and a second is captured as frame 2, but frame 2 is zoomed (scaled) prior to projection, then the data may still be registered if scale can also be recovered. Alternatively, if frame 2 is captured by another sensor or software system and is projected at a different scale, then this scale should also be recovered. The recovery of scale could be considered most important in the case of monocular 3D reconstruction. Here, the fundamental matrix may be used to calibrate camera frames. Once calibrated, stereo methods may be used to extract dense depth information for each frame. The problem with this technique is that depth is relative, but not to scale, across multiple frames. If the scale can also be recovered during the camera pose estimation procedure it would eliminate the need for more advanced methods of optimization.

Phase correlation alone cannot register against camera rotation. An important part of SLAM is the ability to recover 3D camera pose, and a fundamental part of 3D reconstruction is to align dense depth data against as many camera movements as possible. Therefore, I propose using an extension of the popular rotation and scale recovery procedure from 2D registration research to perform camera rotation change estimation as well as align frames which have been zoomed or scaled to fit the volumetric space.

Using the rotation and scale recovery technique, which is comprised of a

few spatial transformations and a phase correlation, a single axis of rotation may be recovered. The axis chosen is integrated into the spatial transformation part of the procedure. Since most scenes would be captured by moving around a room and rotating the camera to look at the areas within the scene directly around the user and camera, we have designated the y-axis of rotation as the most important axis to recover. Additionally, vehicles (with dash mounted cameras) and robots typically keep the camera steady and perform movements and rotations about the y-axis. The application of this rotation and scale recovery technique to 3D reconstruction is novel in itself, but a completely novel technique named FVR-3D is presented in this work. The FVR-3D recovers all 3 axes of rotation using Fourier techniques, this is discussed in section 3.1.5. FVR-3D makes use of the rotation and scale estimation procedure as discussed below.

Two frames are captured as 3D volumes, V_1 and V_2 , where V_2 is captured by a second camera at a different location, pose (about the y-axis) and has data projected differently prior to both systems submitting data for registration and integration into a single global 3D reconstruction. From the perspective of a registration system, the data in frame 2 has been transformed by translation vector $t = [t_x, t_y, t_z]$, rotation factor θ and scale φ . The procedure to recover these values makes use of several properties of the Fourier

transform and the frequency domain. One important property is that, the magnitude of the frequency domains of both V_1 and V_2 are related differently to the raw data from the frames alone. The magnitudes are related by the same rotation factor θ and scale factor φ , but these transforms occur about the origin at the centre of the magnitude volumes, rather than at locations which depend on the translation factor t . In this way, the translation factor can be ignored and the rotation and scale factors can be recovered separately.

This procedure recovers the camera rotation as well as any scale factor which may be present independent of any translation effects. Therefore, once the rotation and scale factors have been recovered, the translation parameters may be estimated using the 3D phase correlation procedure described in section 3.1.1 as an additional step. This procedure starts by applying a Hanning windowing function to both volume frames. The Hanning window function is used to counter the noise generated by the Fourier transform. This noise occurs because the discrete frequency domain is assumed to be infinitely repeating when in practical computer algorithms the space is treated as singular occurring structure surrounded by finite empty space. This unavoidable assumption makes some operations on the frequency domain erroneous. The function for the Hanning window was given in equation 3.6, the scalar for each voxel in frames may be computed for each point $[x, y, z]^T$ via

equation 3.7. The use of this filtering is especially important in computing the rotation and scale parameters.

$$HW_{x,y,z} = \frac{1}{2} \left(1 - \cos \left(\frac{2\pi \left(\sqrt{\left(\frac{N}{2}\right)^3} - \sqrt{\left(x-\frac{N}{2}\right)^2 + \left(y-\frac{N}{2}\right)^2 + \left(z-\frac{N}{2}\right)^2} \right)}{2\sqrt{\left(\frac{N}{2}\right)^3 - 1}} \right) \right) \quad (3.7)$$

The translation separating the camera frames must be removed from the equation prior to estimating the rotation and scale. As mentioned previously, the magnitude of the frequency domain is independent of the effects of translation. So, the magnitude of the frequency domains of both frames are taken, $M_1 = |FFT(V_1)|$, $M_2 = |FFT(V_2)|$. The zero frequency, which is at the corners of the volume (because the discrete frequency domain is assumed to be circularly repeating infinitely) must be transformed to the centre of the volume. In this way, any rotation or scaling occurs about the centre of the volume. Then, both M_1 and M_2 are filtered using an element-wise log function. The frequency domain contains larger values around the zero mean and low frequency areas and smaller values around the higher frequency areas, but for the purposes of computing the camera rotation and any possible scale factor, all frequencies should be treated equally. The element-wise log function therefore suppresses the lower frequency values bringing the higher frequencies up to a similar level.

Given these new volumes, $M'_1 = \text{Log}(M_1)$, $M'_2 = \text{Log}(M_2)$, a geometric transform named the log-spherical transform is used on both M'_1 and M'_2 . The log-spherical transform function re-arranges the input data in such a way, that any y-axis rotation becomes x-axis translation and any scaling becomes z-axis translation. Figure 3.2 shows a projected 3D frame and the log-spherical transformation of it. Equation 3.8 shows the log-spherical space coordinate $(X_{\text{log-spherical}}, Y_{\text{log-spherical}}, Z_{\text{log-spherical}})$ for a given (x, y, z) euclidean space coordinate. This encompasses the effects of the log-spherical transform.

The output x coordinate, $X_{\text{log-spherical}}$ in the log-spherical transform centres the input x and y coordinates about the centre of the volume and uses the atan trigonometric function to obtain the angle in degrees the point is around the y-axis. This is scaled to the full size of the volume via scalar $\frac{N}{360}$. The result is that data at different angles about the y-axis are linearly grouped by their particular angle along the output x-axis. The output y-coordinate $Y_{\text{log-spherical}}$ is also centred about the volume centre and is put through the cosine function. This gives the angle in degrees about the secondary axis (x rotational axis). The degree between 0 and 180 is then mapped to the range $[0 - N]$ using the scalar $\frac{N}{180}$. The y coordinate can be directly passed

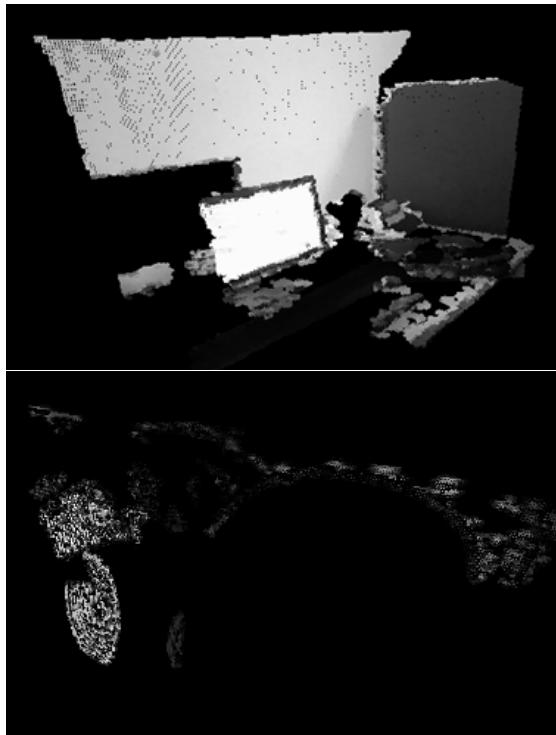


Figure 3.2: Left: Captured 3D frame, Right: Log-Polar Transform of the captured 3D frame

to the cosine function because it is known that the x rotational angle can be computed as the cosine function of the dot product between the input point, $[x, y, z]^T$ and the vertical axis $[0, 1, 0]^T$ which results in $x \times 0 + y \times 1 + z \times 0 = y$.

The output z coordinate, $Z_{log-spherical}$ encompasses the scale information of the input coordinate. First, the distance between the input coordinate and the centre of the volume is computed. This is placed through a log function, the log function compresses the distance such that any scaling relationship

between distances becomes a relationship of addition, or in this case volume translation along the z-axis. This is then mapped using the scalar $\frac{N}{\log(\frac{N}{2.56})}$. This mapping is used so that scales within the range $[2.56^{-1}, 2.56]$ may be registered. Increasing this range too much results in decreased precision and possibly a decrease in accuracy or an inconclusive registration result.

$$\begin{aligned} X_{log-spherical} &= \frac{\arctan\left((x - \frac{N}{2}) \times (y - \frac{N}{2})^{-1}\right) N}{360} \\ Y_{log-spherical} &= \frac{\arccos(y - \frac{N}{2}) N}{180} \\ Z_{log-spherical} &= \frac{\log\left(\sqrt{(x - \frac{N}{2})^2 + (y - \frac{N}{2})^2 + (z - \frac{N}{2})^2}\right) N}{\log(\frac{N}{2.56})} \end{aligned} \quad (3.8)$$

The log-spherical transforms of M'_1 and M'_2 are then related by translation. The translation relationship describes the camera rotation and the scale factor separating both of the original frames. The exact translational differences can easily be recovered using phase correlation. The resulting peak location from the phase-correlation procedure, $(x_{M'}, y_{M'}, z_{M'}) = \text{PhaseCorrelation}(M'_1, M'_2)$ is used to estimate the rotation and the scale. The phase correlation surface of these two volumes is often very noisy. It is usually noisier than phase correlating data from the spatial domain. Filtering techniques may be used

to alleviate this issue. If multiple peaks are detected, the rotation and scale may also be tested for each candidate peak. The rotation θ and scale factor φ separating the two original frames, V_1 and V_2 can then be computed using the peak values on the phase correlation surface, $(x_{M'}, y_{M'}, z_{M'})$. The functions to compute θ and φ are given in equation 3.9.

$$\begin{aligned}\theta &= \frac{-360x_{M'}}{N} \\ \varphi &= e^{-(2.56^{-1}N)z_{M'}N^{-1}}\end{aligned}\tag{3.9}$$

The $x_{M'}$ coordinate is mapped to a unit number in the range $[0, 1]$ and is scaled by -360 resulting in θ , the angle of rotation which may be used to align the frames. The $z_{M'}$ coordinate is also mapped to a unit number, it is then scaled by $\frac{N}{256}$ to align the result prior to inverting the log function effects. The value is negated also, this gives the relationship relative to frame 1 rather than frame 2. Finally, the log function is reversed using the exponential function. The resulting rotation matrix may be formed as in equation 3.10. This conjuncted matrix first transforms the centre of the volume to the origin and then rotates it by θ , then transforms it back to the centre of the volume.

$$\begin{bmatrix} 1 & 0 & 0 & \frac{N}{2} \\ 0 & 1 & 0 & \frac{N}{2} \\ 0 & 0 & 1 & \frac{N}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & -\frac{N}{2} \\ 0 & 1 & 0 & -\frac{N}{2} \\ 0 & 0 & 1 & -\frac{N}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.10)$$

The resulting scaling matrix is formed similarly (equation 3.11) by transforming the volume centre to the origin, scaling the geometry and then transforming the data back to the centre of the volume. The resulting matrix to un-do both the scale and rotational effects of the camera movement can therefore be computed as the scale matrix multiplied by the rotational matrix, $SR_{matrix} = S_{matrix} \times R_{matrix}$.

$$\begin{bmatrix} 1 & 0 & 0 & \frac{N}{2} \\ 0 & 1 & 0 & \frac{N}{2} \\ 0 & 0 & 1 & \frac{N}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \varphi & 0 & 0 & 0 \\ 0 & \varphi & 0 & 0 \\ 0 & 0 & \varphi & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 & -\frac{N}{2} \\ 0 & 1 & 0 & -\frac{N}{2} \\ 0 & 0 & 1 & -\frac{N}{2} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.11)$$

If frame 1, V_1 is then transformed by the scale-rotation matrix SR_{matrix} , the resulting volumes are separated only by translation. The translation matrix separating them may then be computed using phase correlation. The result of this procedure is that the rotation factor θ , the scale factor φ and the translation factors x , y and z have all been recovered. Given the translation matrix, T , formed by the translation factors x , y & z , the full registration

matrix aligning frames V_1 and V_2 may be computed as, $M_{T,\varphi,\theta} = T \times SR_{matrix} \cdot M_{T,\varphi,\theta}$. This matrix can be used to align frame 1 to frame 2 prior to integration. Alternatively frame 2 may be aligned with frame 1 by transforming it by $M_{T,\varphi,\theta}^{-1}$.

The camera pose relationship between frame 1 and frame 2 is also now known. Given the camera coordinate axes in frame 1, $Forward \in R^3$, $Right \in R^3$, $Up \in R^3$, the coordinate space for the camera in frame 2 may be computed by adjusting these axes by the registration matrix $M_{T,\varphi,\theta}$. Once the camera pose is known, the camera may be tracked by repeating the process between successive frames. The resulting dense 3D data may then be integrated to create the complete 3D reconstruction. Alternatively, the dense 3D data may be projected and integrated directly resulting in the output 3D reconstruction.

Conclusion

In this section, the volume registration method was applied to 3D pose estimation. Experiment results for this method are presented in sections 4.4 and 4.5, these results show that this method is capable of accurately reconstructing scenes using different sensors whilst being robust to noise. In the next

section, the method is extended into a full 3D reconstruction algorithm.

3.1.2 Reconstruction from Stereo or 3D Data

Section 3.1.1 described techniques to register camera frames which have been projected to three dimensions. As mentioned, these frames may be registered with respect to the most common type of camera pose transform, y-axis rotation, as well as camera movement. In addition, scale may be registered. For frames which have been projected differently or frames in which depth is computed relative to each frame, such as per frame depth computed in monocular view situations, scale is an important transformation to register against. This is especially true since most existing methods perform simple rigid movement camera pose registration.

This section integrates the phase correlation method into a full 3D reconstruction algorithm named Fourier Volume Registration (FVR). This algorithm may also be used to perform SLAM and it may be applied to input data in which depth is not projected at correct scale between frames. The input required for this method is a list of colour RGB images, each with a depth component. The data may be generated via an RGB-D active camera or computed implicitly using a stereo camera set-up or calibrated monocular set-up.

Results for the FVR method are presented in sections 4.4.3, 4.4.4, 4.4.5 and 4.5 and show the FVR method is capable of accurate 3D reconstructions whilst being robust to noise. In some experiments, it also outperforms the current state-of-the-art methods from the literature. Details on how the Fourier Registration methods described in section 3.1 are integrated to create the FVR algorithm are presented in the following sections.

Pipeline

In Figure 3.3 a pipeline for the FVR registration method is shown. This is a high-level pipeline for the recovery of scale, y-axis rotation and translation. Many of the operations may be performed using GPU processing techniques. Signal Processing methods are very common, many hardware devices exist which can execute these algorithms which are used by the FVR method.

Two volumes, $Volume_1$ and $Volume_2$ are both first put through a Hanning window function, these operations may be performed in parallel. This process multiples each voxel by a scalar which may be computed entirely based on location or a 3D lookup table. Each Hanning window operation may also be performed on the GPU or using another parallel programming

technique such as multi-threading.

Next, using the FFT the magnitude of the Discrete Fourier Transform is computed. The 3D FFT is a common operation which is implemented on most General Purpose Graphics Processing Units (GPGPUs). The 3D FFT operations on the GPGPUs produce the real and imaginary components of the frequency domain, but an additional GPU operation may be used to compute the magnitude values based on the real and imaginary values. Again this operation is easily extended to parallel processing platforms and both volumes may be transformed in parallel.

The $\text{Log}(x)$ function is also a per voxel operation which may be computed in a parallel processing platform and both volumes may be transformed in parallel. The log-spherical transform may be computed for each volume at the same time and is also inherently parallel. Additional memory requirements must also be met as both input and output volumes must be simultaneously kept in memory.

The phase correlation operation is made up of multiple FFTs and their inverse, as well as computing the cross power spectrum between two volumes. All these operations may be performed on GPGPUs. Searching for

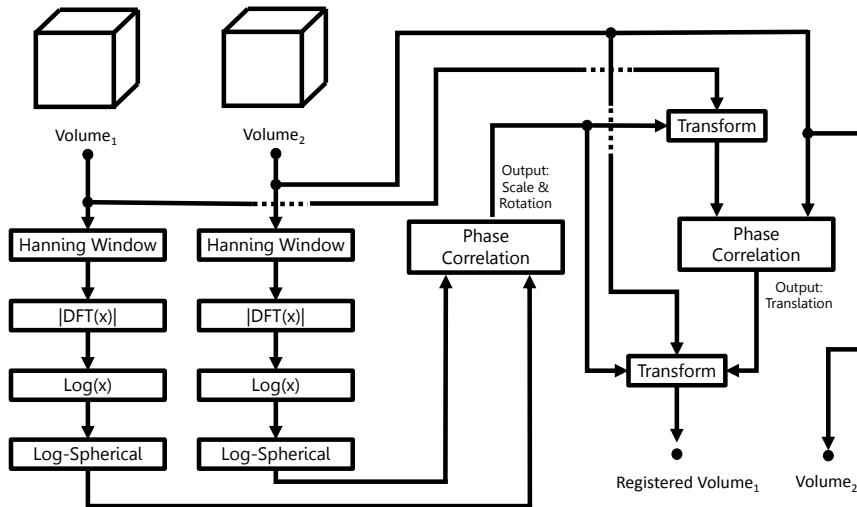


Figure 3.3: System Diagram for Registration Process

the output peak is, however, a sequential process. Parallel techniques, such as reduction, may be used but the performance gain is minimal for volumes of size 256^3 .

Transforming the input volume is another process which may be performed using parallel techniques. The final phase correlation procedure is also parallel in nature apart from the peak value search on the phase correlation surface which is a sequential process.

Algorithm

Using the Registration pipeline to compute the relationship between two frames, a SLAM and 3D reconstruction algorithm may be formulated. This algorithm can construct dense 3D reconstructions whilst simultaneously computing relative camera pose estimates.

The overall algorithm is presented in Listings 3.2. In the first line, the first frame is read as an RGB-D image into variable f_1 . This RGB-D image is projected into a 3D point cloud representation using a projection function. The point cloud data for this first frame is then integrated into an automatically expanding occupancy grid representing the global 3D reconstruction output variable named *GlobalReconstruction*.

An accumulation matrix, M , is used to formulate transforms for each frame read in. It is used by the algorithm to integrate frames directly into the output global reconstruction. Two variables, $Camera_{location}$ and $Camera_{pose}$, are used to represent the camera location and pose information. The pose contains three vectors representing the camera space within the world space. A list of the camera locations and poses is referred to by the *Cameras* variable. Each element in this list is a camera location and pose pair.

Listing 3.2: Phase Correlation Based SLAM Algorithm

```

f1 = ReadFrame();
PointCloud = project(f1);
GlobalReconstruction.integrate(PointCloud)
M = IdentityMatrix();
CameraLocation = [0, 0, 0]T;
CameraPose = [[1, 0, 0]T, [0, 1, 0]T, [0, 0, 1]T];
Cameras = [[CameraLocation, CameraPose]];
while(more frames){
    f2 = ReadFrame();
    points1 = project(f2);
    points2 = project(f1);
    V1 = Voxelize(points1);
    V2 = Voxelize(points2);
    ( $\theta, \varphi, t_x, t_y, t_z$ ) = VR $\theta, \varphi, t_x, t_y, t_z$ (V1, V2);
    Temp = TransformMat(( $\theta, \varphi, t_x, t_y, t_z$ ));
    M = M × Temp;
    points1 = Transform(points1, M);
    GlobalReconstruction.integrate(points1);
    CameraLocation = Temp-1 × CameraLocation;
    CameraPose = Temp-1 × (CameraPose + CameraLocation);
    CameraPose =  $\frac{CameraPose - CameraLocation}{CameraPose - CameraLocation}$ ;
    Cameras.add([CameraLocation, CameraPose]);
    f1 = f2;
}

```

The algorithm iterates over each frame pair which must be registered and integrated. Each iteration begins by reading in a new frame. This is pointed to by variable f_2 . Next, both f_1 and f_2 are projected into point clouds named $points_1$ and $points_2$ respectively. The Fourier Volume Reconstruction method requires the frames be integrated into 3D volumes, so the Voxelize function is used to integrate the points and colour information into volumes.

The FVR method works with either integrated greyscale information based on the RGB data, or with raw binary occupancy values. The use of RGB data may have slight performance implications. Many computer vision algorithms prefer to work in greyscale, especially feature matching methods. This is often due to a combination of complexity and performance reasons. Whilst colour information does improve accuracy, it is considered to be minimal relative to savings made in terms of computational complexity by processing greyscale data alone.

Integrating the colour information for use in the FVR method does not incur any performance penalty. The only shortcoming of using the extra colour information to increase accuracy is the loss of the ability to work in total darkness using RGB-D sensors.

As mentioned in the literature, the Kinect Fusion technique only makes use of depth information which is obtained via an active camera (the Kinect). In this way it can effectively work in the dark. If the FVR method uses colour information integrated directly, and the colour components do not pick up any light, the method will fail. Whether or not colour information is present can be easily detected and because FVR also works by simply registering occupancy volumes, it can switch over to this method upon detecting that no colour (visible light) information is available.

Once the volumes are reconstructed, the rotation, scale and translation factors may be computed using the pipeline discussed in section 3.1.2. A transformation matrix, $Temp$, may then be created from these values. Such a matrix was presented in section 3.1. M is then replaced by itself multiplied by the computed transformation matrix $Temp$. In this way, M can be used to accumulate previous registrations.

The $points_1$ point cloud is then transformed by the newly computed M . The point cloud can then be integrated into the global reconstruction. Next, the camera location and pose list must be updated. The camera location is adjusted by the inverse of the newly computed $Temp$ matrix. As previously

mentioned, the camera transforms will be inversely related to the registration transforms.

The camera pose is also updated, by adjusting the axes relative to the camera location using the *Temp* matrix. The camera pose vectors are then normalized and added into the list of camera locations and poses. Finally, f_1 is set to f_2 . In this way, the algorithm can repeat with f_1 as the previous input frame.

Integration

The integration procedure used with this technique is the common global voxel integration method. An automatically expanding volume is used to store the global 3D reconstruction. Given the algorithm described above, the output global reconstructed volume must take in point cloud data, and integrate it into the volume.

The algorithm proposed does not make use of loop closure. Many recent techniques also forgo the use of this procedure, especially if they integrate globally. The FVR method can also be easily adjusted by changing the last line of the algorithm. Rather than simply setting f_1 to f_2 , f_1 may be set to

the volume extracted by sampling the global reconstruction into a volume based on the most recent camera location and pose computed. This way, loops may be detected by registering upcoming frames globally.

Limitations

The FVR method is capable of registering 3D pose data and in turn can generate accurate, high quality 3D reconstructions. However, it is limited to a single axis of rotation (the y-axis). The FVR method was tested and remains robust to up to 10 degrees of rotation along the other axes, but it is still not able to register all 3 axes of rotation. To move past this limitation whilst retaining the robustness and accuracy of the technique, a novel extension named FVR-3D is proposed in section 3.1.5. Both the FVR method and the FVR-3D algorithm are tested empirically in Chapter 4 and results show they outperform other algorithms from the literature in terms of 3D pose estimation and 3D reconstruction.

3.1.3 Reconstruction from Stereo or 2D Data

The Fourier Volume Reconstruction (FVR) method described in sections 3.1.1 and 3.1.2 was designed to be independent of RGB-D information input

and sensors. As long as greyscale data with a depth component, RGB data with a depth component or depth data on their own are input, the FVR method should be able to compute accurate relatively dense 3D reconstructions and relative camera pose data.

In section 3.1.3 both the advantages and limitations of 3D reconstruction via active depth sensors are discussed. It was discovered that better camera pose registration using the FVR method occurred when the computed depth is closer to the true depth (either via a stereo camera pair, active camera or monocular camera). Section 3.1.3 also covers stereo input. Stereo pair computed depth is potentially the most accurate in terms of depth accuracy and resolution. Since the resolution only depends on the camera pair used, it can be high in contrast to active cameras. Budget active cameras such as the Microsoft Kinect and Asus Xtion Pro Live camera, typically have a lower resolution than budget HD video cameras.

In section 3.1.3 the FVR is applied to monocular data. This technique is named Monocular View Volume Reconstruction (MVVR). The MVVR is discussed in terms of process and limitations in this section. Experiments in section 4.4.5 show that the MVVR method is capable of camera translation tracking using noisy depth data generated by using only monocular meth-

ods. However, its accuracy greatly depends on the quality of the 3D frames generated.

Depth Sensor Based Reconstruction

Depth sensor input is advantageous for several reasons. It is faster to compute depth compared with both stereo and monocular methods, and in practice is more robust and reliable. In most cases it is more accurate than monocular based methods. The major disadvantage in using such sensors has historically been one of accessibility. Thanks to budget active cameras such as the Microsoft Kinect and the Asus Xtion Pro Live, being available to the general public at low cost, this is changing. However, there is still a mass of legacy video data which does not have actively generated depth data provided. Moreover, there is still a wealth of content still being produced by simple RGB cameras.

Another drawback of this sensor is that of accuracy. The accuracy of both the Kinect and Asus Xtion Pro Live sensors is limited to the resolution of the device. To alleviate limited resolution, some sub-pixel resolution generating algorithms may be used. These methods can be used to generate greater resolution for given depth data. Methods may work on frames in a

standalone fashion, although additional data such as consecutive frames and multi-camera techniques may also be used. Both the Microsoft Kinect and Asus Xtion Live Pro have maximum resolutions of 640×480 . Methods of depth generation based on stereo data input can generate dense depth information at resolutions limited only by the resolution of the cameras used. As of 2017, most cameras, even those found on mobile devices, can generate video data at resolutions of 1024×1080 .

Another possible drawback in using RGB-D sensors is that certain materials reflect the infra-red light used by such active cameras to compute the depth information. These sensors are also known to produce noisy depth data around the corners and edges of objects captured within the frame. Since most computer vision algorithms make use of such salient features such as edges and corners within the image data, this can lead to inaccurate 3D reconstructions, especially in cases where 2D feature matching is used with RANSAC to produce 3D camera pose information. These sensors are also limited to indoor environments. If their infra-red sensor components are saturated with sunlight, little to no useful depth data may be generated. Given that much of the world is made up of complex outdoor environments, this is an unfortunate major restriction, especially as these cameras are very useful in indoor and lower light environments.

Another major disadvantage is that these sensors cannot produce depth data over distances. This is due to the limited range of the active infra-red projection component of these sensors. Because the projection cannot reach far distances, the infra-red sensor cannot detect depth. This limited range is in contrast to stereo and monocular methods which are capable of distant depth estimates.

RGB-D camera sensors are still a popular choice for 3D reconstruction frameworks as the depth data produced by these methods is faster and more reliable than other sensors and techniques. For scenes and environments which can be accurately scanned by such depth sensors (e.g. environments with low sunlight, short ranges as in office environments and scenes with few objects which reflect infra-red light), RGB-D cameras are typically the preferred choice. The RGB-D data used to test the FVR method comes from an Asus Xtion Pro Live sensor. This sensor produces a colour and depth image input pair, which is processed to generate 3D volume frames which are the required input for the FVR method.

The colour and depth image pair are referred to by $f(u, v)$ and $g(u, v)$ where $f(u, v)$ refers to the colour image and $g(u, v)$ refers to the depth image. The colour image data contains a red, green and blue component each

between the range [0, 255]. Depth information is processed within the range [0, 10000]. In terms of resolution, u ranges [0, 639] and v ranges [0, 479] giving a resolution of 640×480 . Examples of images generated by the Asus Xtion Pro Live sensor are shown in Figures 3.5a and 3.5b. Given depth value for a given coordinate u, v , $Z_{u,v}$ is equal to the raw depth data value $g(u, v)$. Given coordinate u, v and depth value $Z_{u,v}$ obtained via image g , $f(u, v)$ is projected into 3D space using equation 3.12 to obtain the x-axis coordinate $X_{u,v}$ and y-axis coordinate $Y_{u,v}$. For each pixel value where depth information may be accurately recovered, the 3D point $[X_{u,v}, Y_{u,v}, Z_{u,v}]^T$ may be recovered.

From equation 3.12, the $X_{u,v}$ and $Y_{u,v}$ values are first translated relative to the centre of the screen and then multiplied by the depth and attenuated by the focal length parameter. This is a typical projection function. In the equation, c_x and c_y represent the point where the optical axis intersects the projection plane (which occurs right at the centre of the image, $c_x = 319.5$, $c_y = 239.5$). Parameters f_x and f_y represent the focal length which is defined as $f_x, f_y = 525.0$.

$$\begin{aligned} X_{u,v} &= \frac{(u - c_x)Z_{u,v}}{f_x} \\ Y_{u,v} &= \frac{(v - c_y)Z_{u,v}}{f_y} \end{aligned} \tag{3.12}$$

The FVR method requires that the point cloud generated by 3D points $[X_{u,v}, Y_{u,v}, Z_{u,v}]^T$ and colour data $f(u, v)$ be quantized for integration within a 3D volume. The volume sizes used for testing were 128^3 , 256^3 and 384^3 . Primarily, sizes 256^3 and 384^3 were used. It is common to use sizes which are an exponent of 2 for various computational reasons. However, 512^3 was too large for the GPU used for testing (an NVIDIA GeForce 840 M GPU) so the maximum size used was 384^3 . The 3D point clouds were scaled to fit the volume, then their coordinates were rounded and integrated into the volume prior to registration. Examples of a colour and depth image pair as well as the corresponding projection into different volume sizes (128^3 and 256^3) are shown in Figure 3.4.

Once the data have been projected into a volume, they may be used as input to the FVR method described in section 3.1.2. The algorithm may then compute the registration matrix required to merge two frames, the matrix is accumulated and used to integrate each frame in succession. Upon registration and integration of each frame, the final output 3D reconstruction

may be obtained.

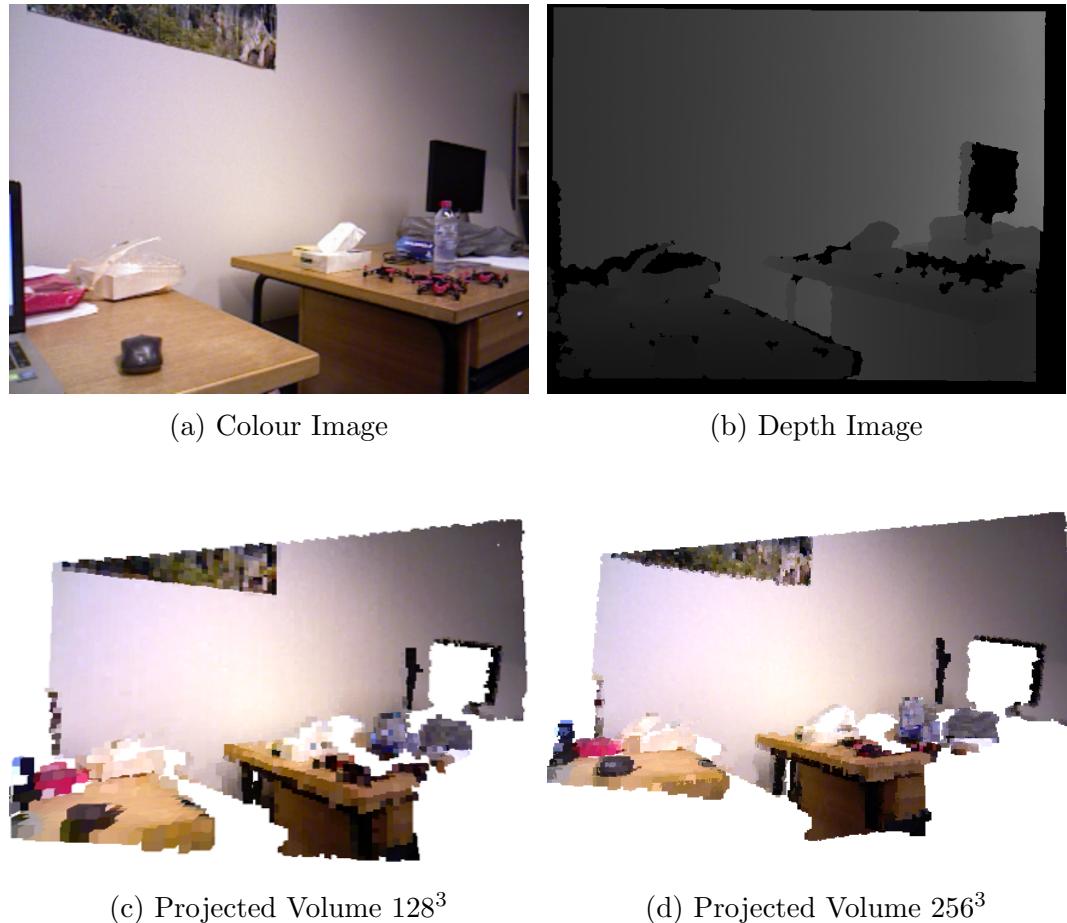


Figure 3.4: A Projected Frame.

Stereo Camera Based Reconstruction

As mentioned previously, the FVR method may take dense depth data from any type of input sensor. As long as dense 3D point clouds can be generated

per frame or near per frame, 3D reconstructions may be computed. Stereo methods can produce depth data at much higher resolutions and accuracies but are not as fast reliable as active sensors such as RGB-D cameras. They can, however, produce depth maps under many environmental conditions where active cameras cannot. Outdoor scenes with a lot of sunlight may be captured with a stereo pair and used to compute dense depth data per frame. These data can then be integrated into the FVR algorithm. Additionally, stereo sensors can generate depth data over larger ranges than active camera approaches. In this way, stereo sensors can be used to scan objects in far off distances or produce depth data from overhead aerial data.

A major disadvantage of practical stereo algorithms is that, depth maps can be noisy. Computing 3D reconstructions from depth data which have been produced via stereo methods is a challenging research project in itself. The noise, however, is much lower in comparison to monocular methods of depth estimation. In generating depth data using a stereo set-up, the stereo camera pair must first be calibrated. The camera pair may be calibrated using software techniques to compute intrinsic camera parameters. Alternatively, the stereo data may be computed without first calibrating the stereo pair, resulting in less accurate depth data. Section 2.1.3 described some of the available techniques in generating stereo data.

Monocular View Volume Reconstruction

Generating dense depth information from monocular view is challenging. There are methods however, which take advantage of the properties of the Fundamental and Essential matrices to pre-calibrate consecutive frames. This allows stereo methods (see section 2.1.3) to be used to compute dense depth data per frame. These stereo methods are typically split into three categories: global, semi-global and local. Global methods are the most complex, and typically make use of some optimization method which jointly optimizes the depth map's smoothness, disparity and contouring of edges. Semi-global methods also optimize the depth map with respect to these requirements, however they are restricted to optimizing scan lines. This makes them much faster than the global optimization methods, whilst still providing better results than the local methods. Local methods are fast and ideal for real time applications, they make use of block matching to compute the disparity per pixel.

These methods all require frame by frame rectification. To do this, the Fundamental matrix must be estimated accurately. The computation of the Fundamental matrix requires the use of feature matching and RANSAC

which often are unable to accurately compute the Fundamental matrix. To generate dense depth data per frame, the use of optical flow based on 2D block matching was explored. By using 2D block matching, dense depth approximations were computed whilst circumventing the use of the fundamental matrix and frame rectification. The depth maps computed are very noisy and inaccurate. The use of 2D spatial filtering as well as temporal filtering reduces such noise.

The spatial filtering technique used is basic Gaussian convolution. Temporal filtering was performed on a per pixel basis, where the output depth map is computed based on both the current and previous depth maps. Each depth map value is set to a value of $\delta \times prior-depth + (1-\delta) \times current-depth$. The best value for δ was judged based on qualitative analysis of experiment data sets. This method, although simpler and less accurate than other stereo methods, was used to test the FVR method's robustness to noisy and inaccurate 3D frame input.

For best performance, RGB-D information from alternative sources, such as depth computed via active camera (Kinect, Asus Xtion Pro Live) or disparity computed via stereo device, may be used in the FVR 3D reconstruction algorithm. An alternative method is proposed here, named Monocular

View Volume Registration (*MVVR*). This algorithm takes consecutive RGB frames as input from a monocular camera sensor and computes depth data from them using the simple block matching method. The depth data are not aligned and because of camera speeds, projection may not be consistent across frames. The depth maps are computed under the assumption of temporal camera translation. By performing block matching on frames taken from different camera locations, depth maps can be computed but only if the camera displacement is large enough. Therefore, a *skip* variable is used to block match frames with a certain distance from each other.

Using the computed depth maps, the algorithm projects consecutively computed depth maps into frames, which are then voxelized. Once the two frames are voxelized, the FVR technique proposed in sections 3.1 and 3.1.1 is used. A registration matrix may then be used to update camera parameters and to register and integrate the 3D frames.

The algorithm is detailed in Listing 3.3. A queue is used, *frameQueue*, to keep track of input RGB frames to generate depth data using frames which have a predefined temporal separation. A stack, *depthMapStack* is used to keep track of the next two consecutive depth maps to register. *GlobalReconstruction* is an automatic dynamically resizeable occupancy grid.

The matrix M is used to keep track of the accumulation of transforms used to register and integrate each depth map into the global reconstruction output. Both $Camera_{location}$ and $Camera_{pose}$ keep track of the camera's location and pose information on a per frame basis. The list, $Cameras$ keeps track of the list of camera location and pose information for each frame.

A loop then processes the frames whilst there are still frames available for input. For each new frame f , f is added to the queue of frames, $frameQueue$. If there are more frames in the queue than variable $skip$ (typically set to 4 to ensure enough camera translation is present to obtain measurable depth), then block matching is performed between the current frame and the oldest item in the queue which is then removed. The corresponding RGB-D image obtained from the $BlockMatch$ function is inserted into the $depthMapStack$ stack.

Also during the loop, if there are two or more RGB-D images within the stack, they are popped out and projected into the $points_1$ and $points_2$ 3D point clouds. Both point clouds are then voxelized into two volumes for input into the FVR method. A matrix representing the transform between the two volumes is then computed.

Listing 3.3: Monocular View Volume Reconstruction

```

frameQueue = newQueue();
depthMapStack = newStack();
GlobalReconstruction.clear();
M = IdentityMatrix();
Camera_location = [0, 0, 0]T;
Camera_pose = [[1, 0, 0]T, [0, 1, 0]T, [0, 0, 1]T];
Cameras = [[Camera_location, Camera_pose]];
while(more frames){
    f = ReadRGBFrame();
    frameQueue.push(f);
    if(frameQueue.length >= skip)
        depthMapStack.push(BlockMatch(f, frameQueue.pop()));
    if(depthMapStack.length >= 2){
        f2 = depthMapStack.pop();
        f1 = depthMapStack.pop();
        points1 = project(f2);
        points2 = project(f1);
        V1 = Voxelize(points1);
        V2 = Voxelize(points2);
        ( $\theta, \varphi, t_x, t_y, t_z$ ) = FVR $\theta, \varphi, t_x, t_y, t_z$ (V1, V2);
        Temp = TransformMat(( $\theta, \varphi, t_x, t_y, t_z$ ));
        M = M × Temp;
        points1 = Transform(points1, M);
        GlobalReconstruction.integrate(points1);
        Camera_location = Temp-1 × Camera_location;
        Camera_pose = Temp-1 × (Camera_pose + Camera_location);
        Camera_pose =  $\frac{Camera\_pose - Camera\_location}{Camera\_pose - Camera\_location}$ ;
        Cameras.add([Camera_location, Camera_pose]);
        depthMapStack.push(f2);
    }
}

```

The accumulation matrix M is then updated using this information and $points_1$ is aligned with $points_2$ using the matrix. The transformed $points_1$ may then be integrated into the global reconstruction occupancy grid. Next the camera pose and location are updated. Finally, the most recently computed depth map, f_2 is inserted back into the stack for comparison with the next depth map computed.

Using the concept of parallax, we know that the magnitude of the displacement vector computed using block matching is an approximation of the actual depth for each corresponding location in both images. Due to noise, however, the resulting displacement vectors can be very noisy. Figure 3.5 shows how noisy this estimated depth information can be compared with input from an RGB-D device. As mentioned previously, the noise can be largely eliminated using spatiotemporal filtering.

Limitations of MVVR

During experiments, it was discovered that the MVVR method was limited to registration of translation information and small scale rotation only due to the high levels of noise. These high noise levels were caused by the difficulty in

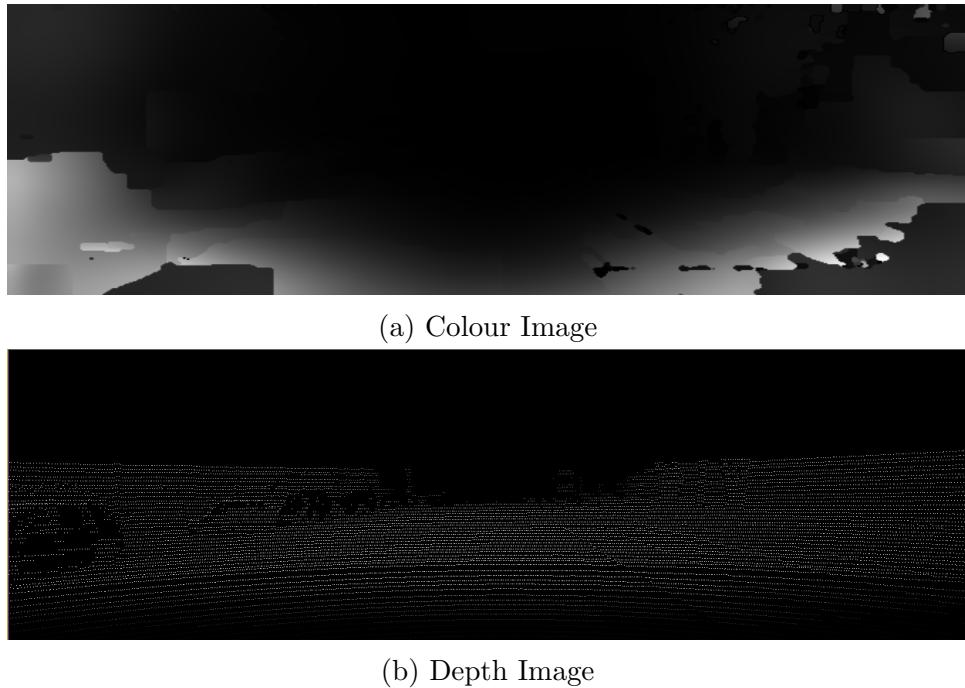


Figure 3.5: Comparison Between Block Matching Estimated Depth (Top) and RGBD Device Depth (Bottom).

computing depth data using basic methods, such as stereo methods, between frames and optical flow. Despite this limitation, the MVVR method was able to register camera translation (see experiments in section 4.4.5).

3.1.4 FVR: Efficiency

The proposed FVR method is interesting in comparison with other major techniques, such as Feature Matching + RANSAC, ICP and other optimization methods, in that it is a closed form solution and not an iterative solution. That is, its computational complexity is fixed and does not depend on the

input data. Despite this, some parts of the pipeline (see Figure 3.3) remain intensive, even for GPGPU and other parallel processing devices.

To reduce complexity, the different parts of the pipeline were examined to reveal any possible improvements. It is understood that each Hanning window function in Figure 3.3 is required to reduce noise on the phase correlation surface. The technique is already highly parallelized and is not very computationally intensive. Much effort has also already been given to improving efficiency in computing the Fourier transform.

The element-wise log function is similar to the Hanning window function, it is required to correlate the data to find the rotation and scale factors separating both volumes. Furthermore it is already highly parallelized and cannot be simplified. The 3D phase correlation technique is by far the most computationally intensive operation within the pipeline. It requires $2 \times$ FFTs, $1 \times$ element-wise operation, $1 \times$ inverse FFT and $1 \times$ peak search operation. Moreover, there are two required 3D phase correlation operations during the pipeline. The other transform operations are also element-wise operations introducing minimal computation expense into the pipeline.

To reduce the computational complexity of the 3D phase correlation, sev-

eral projection operations are used to retain as much information as possible whilst reducing the data to 2 dimensions in such a way that 2D phase correlation (a much faster operation) may be used in place of 3D phase correlation to retrieve transformation factors. Two transforms are proposed to achieve this. The Spherical-Map Transform (section 3.1.4) reduces the original 3D frames to 2D. One useful property of this transformation is that correlation between two Spherical-Map domains retrieved from two 3D frames yields the y-axis rotation and scale factor parameters between the original 3D frames. Moreover, because the spherical-map space is a 2D space, phase correlation may be used in place of manual correlation.

The other transformation is proposed to efficiently compute translation factors separating two 3D volumes. This transform is simply named a projection transform (see section 3.1.4). It reduces the 3D input frames to 2D images whilst retaining the translational information along two remaining axes. Correlating two projection transform domain images yields two translation parameters (depending on the type of projection transform) which separate the two original 3D frames.

A block diagram integrating these speed improvements into the FVR method introduced in section 3.1 is shown in Figure 3.6. This procedure

is referred to as the Fast Fourier Volume Reconstruction method (FFVR). As shown in the block diagram, this procedure takes two 3D volume frames as input, $Volume_1$ and $Volume_2$. The second frame may be taken after the camera has changed pose about the y-axis and/or has moved locations. Both inputs are then put through a 3D FFT function to produce the magnitude values of the 3D frequency domain of both volumes. These operations may be performed on a GPGPU and may be performed in parallel with each other.

The magnitude of the frequency domain is independent of translation and any rotation and scale occurs about the centre of both volumes. Both frequency domain volumes are then transformed into 2D spherical space using the *Spherical2DMap* function. This function produces an image in which 3D y-axis rotation from the original volume is interpreted as 2D translation. To recover the rotation parameter, phase correlation is used to measure the translational component separating the spherical-map domain images. This translational component is then processed to compute the y-axis rotational factor separating the original input volumes. The rotation factor can be directly output as a parameter if required.

Next, the first 3D frame, $Volume_1$ is transformed by the computed rotation factor. This leaves only a 3D translation transform separating both

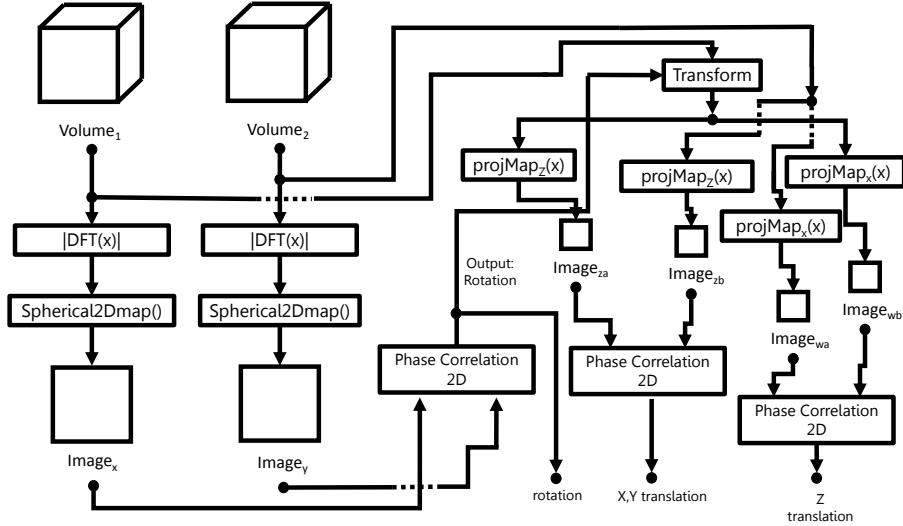


Figure 3.6: System Diagram for Fast Volume Registration

inputs $Volume_1$ and $Volume_2$. Two Projection-map transforms of both $Volume_1$ and $Volume_2$ (equivalent to four transforms in total) are then used to efficiently find this translation factor. The first Projection-map transform is along the z-axis. The z-axis Projection-map transform of $Volume_1$ produces 2D image, $Image_{za}$ the z-axis Projection-map transform of $Volume_2$ produces 2D image, $Image_{zb}$. Both $Image_{za}$ and $Image_{zb}$ may be phase correlated producing the x and y axis components of the translation separating $Volume_1$ and $Volume_2$.

The other two Projection-map transforms are along the x-axis. The Projection-map transform of $Volume_1$ produces $Image_{wa}$, whilst the Projection-

map transform of $Volume_2$ produces $Image_{wb}$. These two images ($Image_{wa}$ and $Image_{wb}$) may be phase correlated producing the z-axis component of the translation. A composite registration matrix aligning $Volume_1$ to $Volume_2$ may be formed by translating each volume's centre to the origin, rotating by the computed rotation factor, translating the origin to the volume's centre and finally translating by the $[X, Y, Z]^T$ translation vector computed.

Noticeably, the Hanning window function and post $|DFT|(x)$ $\text{Log}(x)$ function from the original pipeline (figure 3.3) are missing. The Hanning window function may be incorporated in the first phase correlation procedure, which processes 2D images, therefore the Hanning window function would be in 2D rather than 3D which also improves efficiency as an entire dimension is removed from the process. The log function may also be performed as part of the phase correlation procedure, however it was found that the FFVR procedure estimates rotation more reliably using the spherical-transform if the log function is not performed. Consequentially, this saves computational cost.

It is advantageous to use 2D phase correlation over 3D phase correlation from a computational complexity perspective. The 3D Fourier transform has complexity of $N^3 \times \text{Log}(N^3)$ whilst the 2D has complexity $N^2 \times \text{Log}(N^2)$.

Essentially the amount of data to process has been reduced by an entire dimension. The phase correlation method requires $2 \times$ FFTs, $2 \times$ element-wise computations and $1 \times$ inverse FFT. The corresponding 3D phase correlation complexity equates to $3N^3 \text{Log}(N^3) + 2N^3$ whilst the 2D equivalent is only $3N^2 \text{Log}(N^2) + 2N^2$.

Spherical-map transform

As noted, the Spherical-Map transform reduces the 3D volume to a 2D image whilst retaining information about y-axis rotation. In the new domain, and rotation about the y-axis becomes x-axis translation within the output image. The transform requires a single iteration over the input volume, so it has identical complexity to the 3D Log-Polar transform whilst reducing computational complexity further down the pipeline by compacting the data to process from three dimensions to two dimensions.

An example of an input model (Figure 3.7a) and the Spherical-Map domain of the model (figure 3.7b) is shown in Figure 3.7. The relationship between the input 3D volume Vol and the output 2D image Im is defined using equations 3.13, 3.14 and 3.15. The value of pixel $Im_{x,y}$ located at coordinate x, y is computed by summing the values along a given ray within the

volume. The ray is defined by the vector valued function $Ray(x, y, r)$ found in equation 3.14. This function takes the x and y coordinates within the image as well as a radius value (the index from the summation in equation 3.15) and returns a vector. The vector is used to index the volume and allows the output pixel to include all the values along the ray.

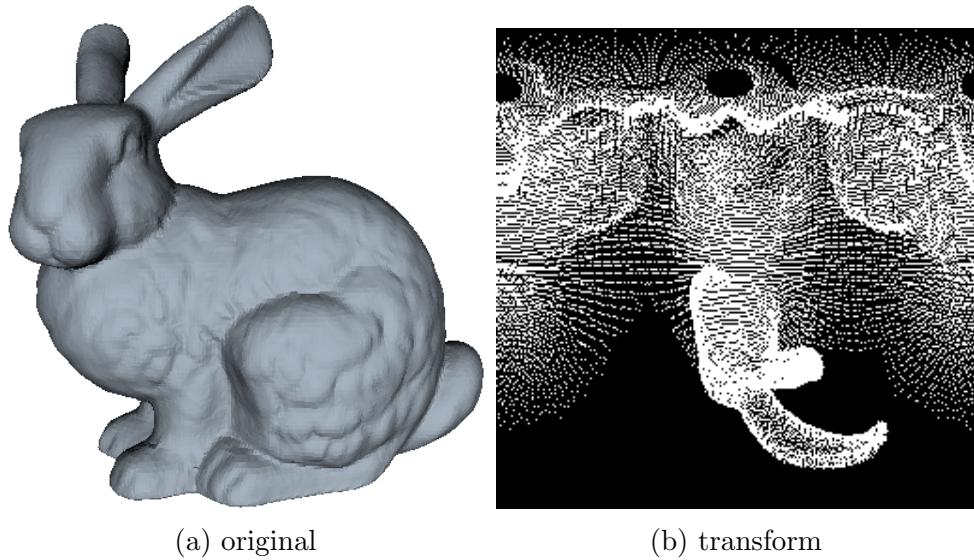


Figure 3.7: The Spherical Map Transform.

Equation 3.14 is a vector valued function with three separate functions for the x, y and z axis components. These functions are found in equation 3.13. The x-axis function $Ray_x(x, y, r)$ is used together with the $Ray_y(x, y, r)$ and $Ray_z(x, y, r)$ functions to form a spherical transform. The reduction in dimension is achieved by summing the components which share x and y co-

ordinates but have different r (radius) values. In these equations the 3D volume Vol has a width, height and depth of N whilst the output image Im has a width and height of M .

$$\begin{aligned} Ray_x(x, y, r) &= r \times \cos\left(\frac{360x}{M}\right) \sin\left(\frac{180y}{M}\right) + \frac{N}{2} \\ Ray_y(x, y, r) &= r \times \cos\left(\frac{180y}{M}\right) + \frac{N}{2} \\ Ray_z(x, y, r) &= r \times \sin\left(\frac{360x}{M}\right) \sin\left(\frac{180y}{M}\right) + \frac{N}{2} \end{aligned} \quad (3.13)$$

$$Ray(x, y, r) = [Ray_x(x, y, r), Ray_y(x, y, r), Ray_z(x, y, r)]^T \quad (3.14)$$

$$Im_{x,y} = \sum_{r=1}^{(2^{-1}N)^{1.5}} Vol(Ray(x, y, r)) \quad (3.15)$$

In the Log-Spherical transform described in section 3.1.1 re-arranges the rotation and scale transforms along two axis, the third axis retains information and increases the accuracy of the solution. By reducing the other axis from size N to the average value (size 1) information is lost, however computational complexity is reduced by an entire dimension. This is a complexity/performance trade off. Computational complexity may be afforded

given input data contains relatively little noise. The output image maps 3D y-axis rotation to 2D x-axis translation.

This process essentially sums up the values along a given ray defined by scaling spherical coordinates and adding up the values intersecting the ray. The resulting image, maps 3D y-axis rotation to 2D x-axis translation. Once two 3D frame volumes V_a and V_b are transformed into their corresponding spherical-map domain SM_a and SM_b respectively, the two may be phase correlated to measure the x-axis translation separating them. This x-axis translation may then be mapped directly to the rotational angle separating the two input 3D camera frames. The relationship may be computed as $\frac{360x}{M}$ where x is the x-axis translation and M is the width and height of the spherical map domain image.

Projection-map transform

The Projection-map transform is similar to an orthogonal projection of the volume along a particular axis. Instead of simply projecting the closest point as done in projections for visualization purposes, the sum of values computed along the ray is used for representation. Figure 3.8 shows an example model (Figure 3.8a) and three Projection-map transformations of the model along

three different axes (Figures 3.8b, 3.8c and 3.8d). The Projection-map transform is described here in terms of an output Projection-map image Im and an input 3D volume frame Vol , which may be input from a sensor or software system which is able to generate 3D frames of the environment. Each pixel in Im has its value defined mathematically as the summation of values along a particular axis given the x,y coordinates. The Projection-map x-axis transform and the Projection-map z-axis transform are defined in equations 3.16 and 3.17 respectively.

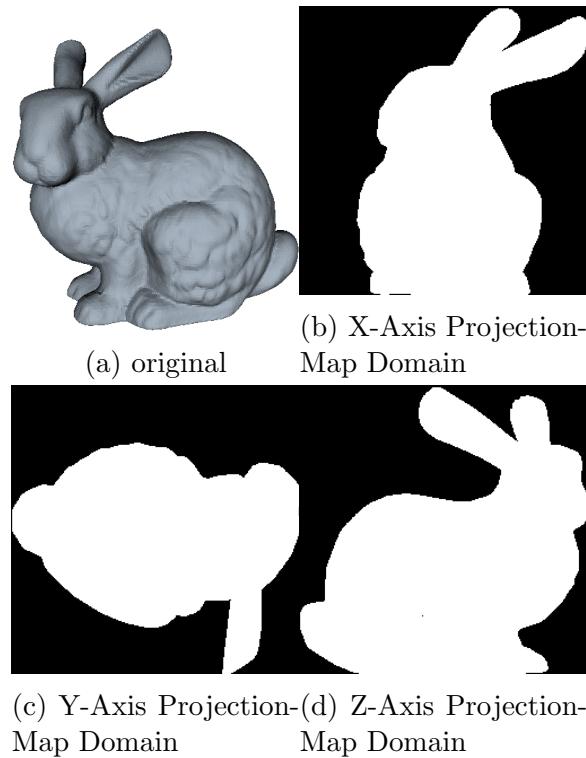


Figure 3.8: The Projection-Map Transform.

$$Im(z, y) = \sum_{x=0}^N Vol(x, y, z) \quad (3.16)$$

$$Im(x, y) = \sum_{z=0}^N Vol(x, y, z) \quad (3.17)$$

In these equations the Projection-map transform along the x-axis is computed via summing up values along the x-axis in the input 3D frame. In this way, any 3D z-axis translation in Vol corresponds to 2D x-axis translation in the output image Im . Additionally any y-axis translation in the input 3D frame Vol is mapped to a corresponding y-axis translation in the output 2D Projection-map domain Im . If both the width and height of the output image Im is equal to the width, height and depth of the input 3D frame then the relationship is a 1 : 1 mapping between 3D z-axis translation and 2D x-axis translation as well as 3D y-axis translation to 2D y-axis translation.

The corresponding z-axis Projection-map transform sums up values along the z-axis. Here, 3D y-axis translations within the original 3D frame also produce 2D y-axis translations in the output Projection-map domain. However, all x-axis translations occurring within the 3D frame domain are mapped to 2D x-axis translations in the output Projection-map domain image Im .

Given both of these Projection-map transforms, the 3D translation separating two 3D frames may be registered with respect to 3D translation. This 3D translation also has an inverse relationship to the camera location separation between the two frames. Both 3D frames are transformed into the z-axis Projection-map domain. By phase correlating these domains, both the x and y axis translation may be retrieved which map directly to the 3D translation x and y coordinates separating the input frames. The 3D z-axis translation may be computed by transforming both frames into the x-axis Projection-map domain. Once the two Projection-map domains are phase correlated, the x-axis translation separating the domains can be mapped directly to the corresponding z-axis translation separating the two input 3D frames. In total, using $2 \times$ Projection-map transforms and $1 \times$ 2D phase correlation method, the 3D translation parameters separating two 3D frames may be computed.

Performance Analysis

To assess the efficiency gain between the Fast Fourier Volume Registration method over the original FVR method, the computational complexity of the FVR is assessed. The computational complexity is dependent on two factors: the size of the input depth map $W \times H$ and the width/height/depth of the

3D volumes used in the volume registration procedure. Since the computational complexity is fixed no matter the input data (depending only on the sizes specified by the user), this method is considered a closed form solution.

The FVR method makes use of several sub-procedures including: $2 \times 3D$ projections (of the depth frame to a 3D volume frame), $2 \times$ Hanning window filters, $2 \times 3D$ FFTs, $2 \times$ element-wise Log filters, $2 \times$ Log-spherical transforms, $2 \times 3D$ phase correlation processes and $2 \times$ linear transformations. The 3D phase correlation method makes use of several sub-procedures including: $3 \times 3D$ FFTs, $1 \times$ spectra multiplication and $1 \times 3D$ peak finding operation. The number of operations required by each of these methods are listed in table 3.1.

Table 3.1: Complexities for given Procedures

Procedure	Complexity
3D Projection	$14WH$
Hanning window	26 (GPGPU)
3D FFT	$N^3 \log N^3$ (GPGPU)
Log	3 (GPGPU)
Log-Spherical	58 (GPGPU)
Multiplying Spectra	15 (GPGPU)
Geometric Transform	30 (GPGPU)
3D Peak Search	$2N^3$
3D Phase Correlation	$3N^3 \log N^3 + 2N^3 + 44$

Summing these operations whilst scaling them by the number of times they are used gives the total complexity for the FVR method. This com-

plexity is given in equation 3.18. The complexity of the 3D phase correlation procedure adds up to the complexity shown in equation 3.19.

$$8N^3 \log N^3 + 4N^3 + 28WH + 322 \quad (3.18)$$

$$3N^3 \log N^3 + 2N^3 + 15 \quad (3.19)$$

The full complexity of the FVR method (equation 3.18) is next compared to the performance of the FFVR method to assess performance gain. The FFVR methods makes use of several of the procedures found in Table 3.1 and several others. These include: $2 \times 3D$ FFTs, $2 \times$ Spherical-Map transforms, $3 \times 2D$ phase correlation procedures, $1 \times 3D$ geometric transform and $4 \times$ Projection-Map transforms. The 2D phase correlation method makes use of several sub-procedures including: $3 \times 2D$ FFTs, $1 \times$ spectra multiplication and $1 \times 2D$ peak finding operation. Table 3.2 cites the complexities for the additional operations including: Spherical-Map transform, Projection-Map transform, 2D Peak finding, 2D FFT and 2D phase correlation (made up of multiple sub-procedures).

From Figure 3.6 we can see that the FFVR method is made up of multiple sub-procedures. These include: $2 \times 3D$ projections, $2 \times 3D$ FFTs, $2 \times$

Table 3.2: Complexities for given Procedures

Procedure	Complexity
Spherical-Map Transform	45 (GPGPU)
Projection-Map Transform	1 (GPGPU)
2D Peak Search	$2N^2$
2D FFT	$N^2 \log N^2$ (GPGPU)
2D Phase Correlation	$3N^2 \log N^2 + 2N^2 + 15$ (GPGPU)

Spherical-Map transforms, $1 \times 3D$ geometrical transformation, $3 \times 2D$ phase correlations and $4 \times$ Projection-Map transforms. The total complexity is summed for all of these functions and given in equation 3.20.

$$2N^3 \log N^3 + 9N^2 \log N^2 + 6N^2 + 28WH + 169 \quad (3.20)$$

Figure 3.9 provides a visualization of the performance improvement which the FFVR method achieves over the original FVR approach. The graph displays the performance of both methods at different volume sizes increasing logarithmically. The FFVR method is around 1.8 times faster for 64^3 volumes, 3.4 times faster for 128^3 volumes, 4.1 times faster for 256^3 volumes and 4.2 times faster for 512^3 volumes. This is due to the reduction in the amount of data to process afforded by the novel spherical-map transform and Projection-map transform methods.

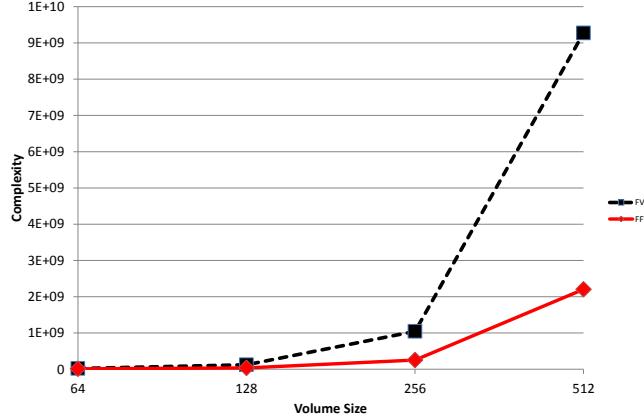


Figure 3.9: Comparison of performance between FVR and FFVR.

Performance Comparison

In this section, the computational complexities of the FVR related techniques are compared with common algorithms used for 3D reconstruction from the literature. These include: 2D/3D Feature Matching (SURF/SIFT) with RANSAC, PCA, and ICP. These algorithms are compared with the complexity of the FVR, FFVR and FVR-3D (Note: the complexity of MVVR is similar to FVR). Here we use N to represent volume size, for example the FVR based methods usually use a volume of size N^3 . W is used to describe the size of images processed (used to describe the number of pixels in a $W \times W$ image, projected into a $W \times W$ 3D point cloud).

First, the two feature matching methods SIFT and SURF are evaluated. The complexity evaluation of SIFT was taken from work by Vinukonda [184] whilst the complexity of SURF was computed based on a public implementation found at <https://github.com/lukes611/phdThesis>. The complexity of SIFT is based on the image size W , the Gaussian filter size w (typically 5), the number of octaves s (3 are used in experiments), the fraction of extrema in the image α (found to be approximately 6%), the fraction of extrema which are features β (found to be approximately 35%), the number of additional features found γ (found to be approximately 0.04%) and the size of the descriptor neighbourhood x (typically 8).

The complexity of finding SIFT features and computing a SIFT descriptor for an image was found to be,

$$W^2(s(4w^2 + 100\alpha + 156) + 1520x^2(\alpha\beta + \gamma)) \quad (3.21)$$

By plugging in the usual values (equation 3.22), we can form a computational cost as a function of W and variables α , β and γ , the variables representing the fraction of pixels which are features.

$$W^2(300\alpha + 768 + 97280(\alpha\beta + \gamma)) \quad (3.22)$$

The complexity of SURF is based on several similar factors. Again, W

is the image size, s is the number of octaves (3 is the default), α is a scalar (with a default value of 3), and f the fraction of image pixels which are detected as features. Functions which make up the SURF feature detection and descriptor extraction include integral image creation of complexity $W^2 + W$, computation of Hessian feature detection of complexity $\sum_{i=1}^s 668(\frac{N}{2^i\alpha})^2$ and the computation of the SURF descriptors of complexity $213434N^2f$. Plugging in default values (equation 3.23), the complexity may be formed as a function of the image size, W and the percentage of pixels found to be features f .

$$W^2 + W + N^2(1972.688 + 213434f) \quad (3.23)$$

The complexity of RANSAC was also analysed. Based the implementation used to compute camera pose from feature matches and corresponding 3D points, the complexity is dependent on W the image size, I the number of iterations (we use 500) and f the fraction of pixels which are matched features. The total complexity is given in equation 3.24.

$$134IN^2f \quad (3.24)$$

Therefore, the total complexities of SURF + RANSAC and SIFT + RANSAC are given in equations 3.25 and 3.26 respectively.

$$W^2 + W + N^2(1972.688 + 213434f) + 134IN^2f \quad (3.25)$$

$$W^2(300\alpha + 768 + 97280(\alpha\beta + \gamma)) + 134IN^2(\alpha\beta + \gamma) \quad (3.26)$$

The complexity of the 3D extension of the SIFT feature matching + RANSAC method is shown in equation 3.27, where N^3 is the volume size used to project the 3D data into.

$$N^3(1968 + 300\alpha + 41040(\alpha\beta + \gamma)) + 134IN^3(\alpha\beta + \gamma) \quad (3.27)$$

The complexity of the PCA method was taken from [185]. It is a function of W (again where the number of pixels in the image is W^2) and $p = 3$, the dimensionality of the 3D points. The complexity was found to be $p^2W^2 + p^3$, plugging in 3 for p gives the complexity as a function of W provided in equation 3.28.

$$9W^2 + 27 \quad (3.28)$$

Finally the complexity of ICP is described. The complexity of ICP in the context of pose estimation is based on the complexity of (i) computing the K Nearest Neighbours (KNN) algorithm between point clouds ($3W^2f$), (ii) computing the least squares optimal pose based on the matches ($16W^2f + 96$),

Table 3.3: Algorithm Complexities

Algorithm	Complexity
FM2D + RANSAC	$W^2 + W + W^2(1972.688 + 213434f) + 134IW^2f$
FM3D + RANSAC	$N^3(1968 + 41040f) + 134IN^3f$
PCA	$9W^2 + 27$
ICP	$1008W^8f^3 + 6048W^4f^2$
FVR	$8N^3 \log N^3 + 4N^3 + 322$
FFVR	$2N^3 \log N^3 + 9N^2 \log N^2 + 6N^2 + 169$
FVR-3D	$8N^3 \log N^3 + 4N^3 + 349 + 9W^2$

and (iii) testing the computed pose $21W^2f$. Here, W is the image size and f is the fraction of projected pixels used in the ICP algorithm. This is performed up to 500 times leading to the complexity in equation 3.29.

$$1008W^8f^3 + 6048W^4f^2 \quad (3.29)$$

The complexities for each of the algorithms mentioned are shown in Table 3.3. Figure 3.10 compares complexity as a function of percentage of pixels used as features. In this figure, both ICP and FM-3D were omitted as they had theoretical performances which were so large they skewed the comparison (for perspective, Figure 3.11 includes both ICP and FM3D). In these tests, we used a RANSAC iteration count of 500, image sizes of 1024^2 and volume sizes of 256^3 . The RANSAC iteration count was chosen as it worked well in experiments, the image sizes were chosen to be HD and the volume size was chosen based on the experiments performed using the FVR based al-

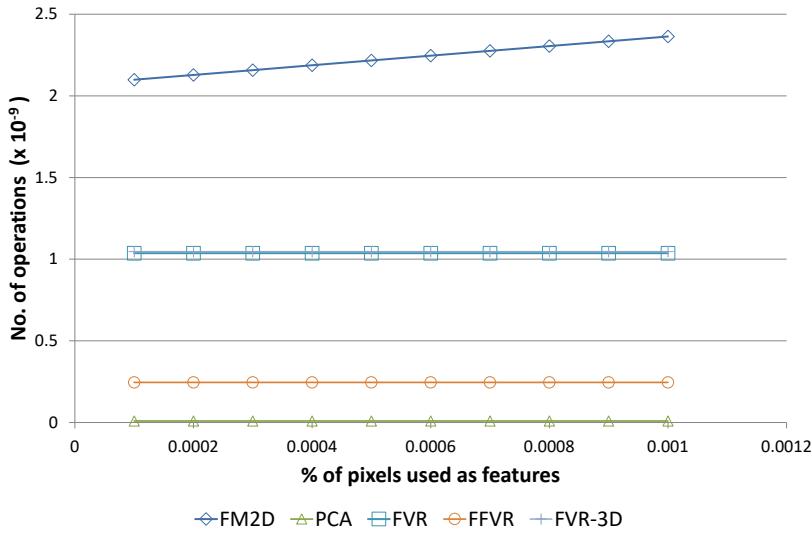


Figure 3.10: Theoretical Complexity Comparison

gorithms. Comparing these results, FM2D has the largest complexity. FVR and FVR-3D had essentially the same complexity and FFVR was found to be slightly faster than these, as it was designed to be. PCA is the simplest algorithm and the fastest here. Both ICP and FM-3D had complexities which increased exponentially as the percentage of pixels increased, whilst FM2D increased linearly. PCA and the FVR based methods have flat complexities which do not vary with the number of features used since they use the entire point cloud or volume.

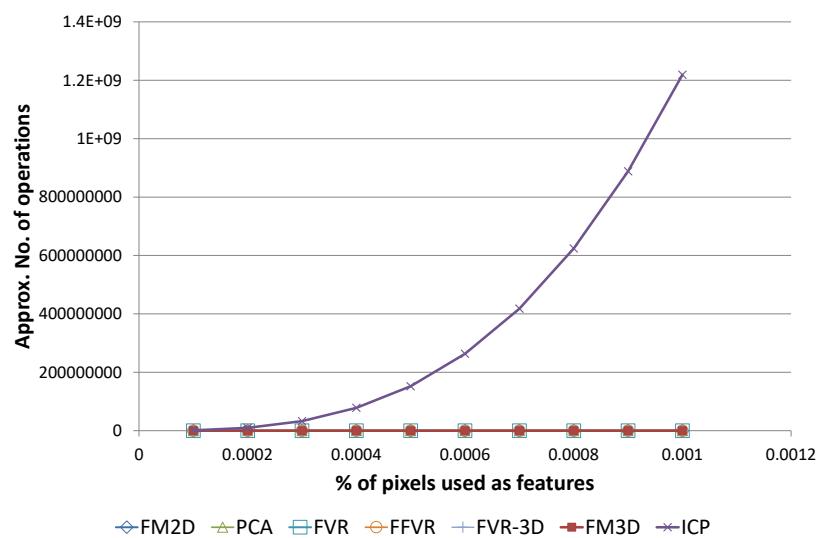


Figure 3.11: Theoretical Complexity Comparison (Including FM3D and ICP)

3.1.5 3D Rotation Recovery

This section explains how the FVR algorithm may be extended to recover full 3D camera pose rotation in addition to translation and scale differences between a pair of captured 3D frames. We propose a novel method which uses Principal Components Analysis (PCA) as a pre-processing step to achieve full 3D rotation estimation. This method is named FVR-3D. The FVR method, the FFVR method and the FVR-3D method all have unique advantages and disadvantages. FVR is designed to be an accurate and efficient method of estimating camera pose and location on wider baselines whilst remaining robust to the effects of noise. It is robust to 3D rotations (not just y-axis rotation) but cannot register them, if 3D rotation recovery is required, the FVR-3D method discussed in this section provides an efficient solution, but is less robust to noise and wider baselines. The FFVR method may also be used, it works similarly to the FVR method but is computationally more efficient at the expense of accuracy and noise robustness. FFVR also cannot handle wide baselines well, but is up to 4 times more efficient than the FVR for realistic volume sizes.

To integrate 3D rotation into the standard FVR method, a single axis relating the two 3D models must be known. If such an axis is known prior to executing the FVR method, both 3D frames may be aligned vertically (along

the y-axis) with respect to the found axis. Once aligned, the FVR method may be used to register both frames relative to the final y-axis rotational separation which can be used to recover rotation and scale factors so translation may also be recovered.

Several techniques were explored to compute this commonly rotated axis. These include computing the average normal value, the axis defined as the vector between the furthest two points as well as other techniques based on feature matching. The use of PCA was also tested. Based on empirically driven factors, PCA was chosen to compute the common axis for use in the FVR-3D method. PCA may be used to compute the full rotation separating two 3D frames but this typically fails when noise or non-overlapping parts of the frames are present. To be more robust to noise and non-overlapping features within the frame, only the primary axis is used to align both frames with their primary axes pointing directly up. This alignment is more robust because the primary axis has a stronger presence within the data. Once the y-axis rotation is solved, the full 3D rotation factors may be computed via the FVR method.

Experiments evaluating FVR-3D in section 4.4 show it outperforms other popular methods used for 3D camera pose estimation in terms of accuracy

on a wide range of scene types. Additional experiments prove its robustness in the face of noise when registering camera pose.

Computing a Principal Axis

Using PCA to compute the principal axis used in FVR-3D is discussed in this section. Both 3D frames have their axis computed separately before they are used to align both frames to a common axis. To compute the common axis using PCA, both the eigen-values and eigen-vectors of the covariance matrix of the 3D frames must be computed. The eigen-vector with the largest corresponding eigen-value is set as the primary axis. The procedure works as follows.

Given a 3D point cloud A of N points, which may be generated via monocular methods, stereo disparity estimation or active sensor (Kinect, Asus Xtion Pro Live), the co-variance of the points between two of the dimensions (x and y) is computed as the summation of each x component subtracted by the mean x value multiplied by the y component subtracted by the mean y component value (see equation 3.30, A_{xmean} and A_{ymean} refer to the mean values of these x and y dimensions of A).

$$Cov(A_x, A_y) = \sum_{i=0}^N (A_{x_i} - A_{x_{mean}})(A_{y_i} - A_{y_{mean}}) \quad (3.30)$$

Using the formula for covariance the covariance matrix may be computed for an N dimensional signal. In the case of 3D construction, this matrix is a 3×3 covariance matrix where each column/row index represents a covariance relationship. This covariance matrix is shown in equation 3.31. This matrix describes how each coordinate axis changes with respect to each other coordinate. The eigen-vectors of this matrix constitute the principal components of the 3D point cloud frame.

$$\begin{bmatrix} Cov(A_x, B_x) & Cov(A_x, A_y) & Cov(A_x, A_z) \\ Cov(A_y, B_x) & Cov(A_y, A_y) & Cov(A_y, A_z) \\ Cov(A_z, B_x) & Cov(A_z, A_y) & Cov(A_z, A_z) \end{bmatrix} \quad (3.31)$$

The three eigen-vectors of the covariance matrix describe the primary axis of the 3D frame A and the three corresponding eigen-values describe the dominance of these vectors. The eigen-vector/axis with the largest corresponding eigen-value is the principal component in PCA and is used as the aligning axis in the FVR-3D method. This principal axis is used to align data to a common axis defined as the vertical y-axis. For simplicity, the principal axis of a 3D frame A is written as A_{pa} . During the PCA process, the cen-

troid of the point cloud is also computed. This is simply the average point location within the input point cloud. For simplicity in later discussions, the centroid of a 3D frame A is written as $A_{mean_{location}}$. A visualization of the principal axis and centroid are shown in Figure 3.12. Here two input frames (Figures 3.12a and 3.12b) are corrected such that their mean is translated to the centre of the volume and their principal axis is aligned with the vertical axis (Figures 3.12c and 3.12d).

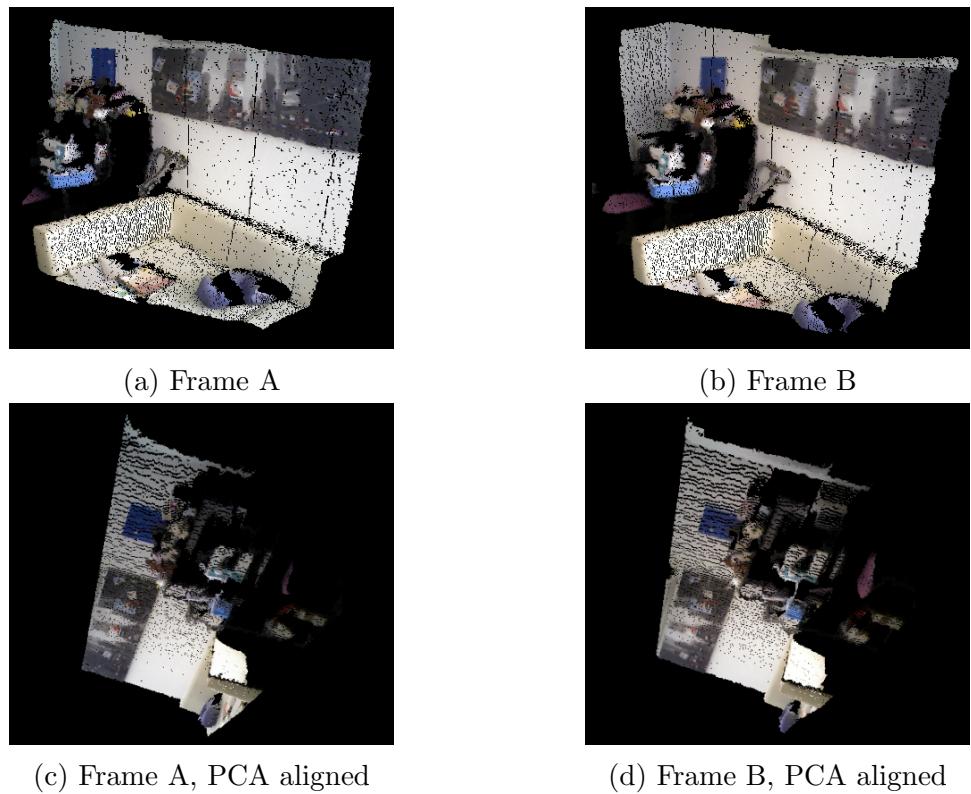


Figure 3.12: Early stages of FVR-3D

Alignment Pre-Process

Using the PCA method, for each input 3D frame V , the computed principal axis V_{pa} and the centroid $V_{mean_location}$ are used to align the input 3D frame using V_{pa} as the new vertical axis. To this end, a transformation matrix is formed and used to rotate the volume so the principal axis V_{pa} points up. This is useful because if a 3D frame pair are both rotated with respect to their principal axis, and both frames share enough overlap, then only 3D y-axis rotation, 3D translation (and possibly 3D scale) separate the two frames. As discussed in section 3.1.2, these parameters may be recovered using the FVR method. The matrix used to normalize the 3D frames in terms of their principal axis is discussed here.

If V_{pa} (the vector pointing along the principal axis of V) is to be set as the new y-axis for the volume, then the other axes must also be computed (x-axis and z-axis). The z-axis, named the forward (*Fwd*) axis, is computed based on the cross product between the then x-axis and the principal axis V_{pa} (taken as the y-axis) which gives a psuedo z-axis. The cross product between the principal axis and this pseudo z-axis gives an accurate x-axis. Lastly the cross product between this x-axis and the y-axis (the principal axis) gives the corresponding z-axis in the form of the variable *Fwd*. This is shown in equation 3.32.

$$Fwd = \left(\begin{bmatrix} V_{pa_x} \\ V_{pa_y} \\ V_{pa_z} \end{bmatrix} \times \left(\begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix} \times \begin{bmatrix} V_{pa_x} \\ V_{pa_y} \\ V_{pa_z} \end{bmatrix} \right) \right) \times \begin{bmatrix} V_{pa_x} \\ V_{pa_y} \\ V_{pa_z} \end{bmatrix} \quad (3.32)$$

The final axis completing the new space is the x-axis defined at the cross product between the principal axis (the new y-axis) and the z-axis (the *Fwd* vector) and is named *Rgt* (standing for right facing axis). Equation 3.33 shows this calculation. The new space is now defined by x-axis *Rgt*, y-axis V_{pa} and z-axis *Fwd*. A rotation matrix transforming original space to the new space defined by these axes may be computed using the *Rgt*, V_{pa} and *Fwd* axes as the column vectors of the rotation matrix. The inverse, that is the transformation from the new space to the original space may be computed using the transpose of the rotation matrix. The rotation of the point cloud from the new space defined by the principal axis to the original space is useful as it aligns the principal axis to the y-axis leaving only 3D y-axis rotation, 3D translation and possibly scale to be registered.

$$Rgt = \begin{bmatrix} V_{pa_x} \\ V_{pa_y} \\ V_{pa_z} \end{bmatrix} \times \begin{bmatrix} Fwd_x \\ Fwd_y \\ Fwd_z \end{bmatrix} \quad (3.33)$$

The input 3D frame V may be aligned by its principal axis as the new

y-axis by first transforming the centroid to the origin. Next the 3D frame may be rotated from the new space to the default identity space. This sets Rgt as the new x-axis, V_{pa} as the new y-axis and Fwd as the new z-axis. The rotation matrix may be computed as the row vectors of these axes. Finally, after rotation the 3D frame V may be transformed back to the centroid. This aligns the 3D frame so that V_{pa} points towards the y-axis. The compounded transformation to perform this alignment is fully defined in equation 3.34.

$$CorrectMat(V) = \begin{bmatrix} 1 & 0 & 0 & V_{mean_x} \\ 0 & 1 & 0 & V_{mean_y} \\ 0 & 0 & 1 & V_{mean_z} \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Rgt_x & Rgt_y & Rgt_z & 0 \\ V_{px} & V_{py} & V_{pz} & 0 \\ Fwd_x & Fwd_y & Fwd_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -V_{mean_x} \\ 0 & 1 & 0 & -V_{mean_y} \\ 0 & 0 & 1 & -V_{mean_z} \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (3.34)$$

3D Rotation Registration

To recover a matrix for full 3D rotation separating two input 3D frames A and B which have been taken from two different locations (translation separation) with two different poses (3D rotational separation) and possibly projected differently (scale separation), the first step is to align both frames by their principal axis. The alignment matrix for A can be computed as $C_A = CorrectMat(A)$ and the alignment matrix for B may be com-

puted as $C_B = \text{CorrectMat}(B)$. Transforming both A and B by their alignment matrix gives two volumes $A_{aligned} = \text{Transform}(A, C_A)$ and $B_{aligned} = \text{Transform}(B, C_B)$ which are now only separated by 3D y-axis rotation, 3D translation and 3D scaling factors. As discussed in sections 3.1.1 and 3.1.2 these factors may be computed and a registration matrix aligning $A_{aligned}$ and $B_{aligned}$ generated. This matrix is computed as $R_y ST = FVR(A_{aligned}, B_{aligned})$. Using this registration matrix and the alignment matrices C_A and C_B , the full 3D registration matrix $R_{x,y,z} ST$, which transforms A to B , may be computed. Matrix $R_{x,y,z} ST$ aligns A , registers it using $R_y ST$ (which aligns it with $B_{aligned}$) then inverse transforms it by C_B which un-aligns it according to alignment matrix C_B . This can be followed as in equation 3.35.

Figure 3.13 (along with Figure 3.12) shows the output at different stages of the FVR-3D technique. Once two input frames (Figures 3.12a and 3.12b) are aligned via PCA (Figures 3.12c and 3.12d), they must be registered using the FVR method. This is shown in Figure 3.13d, as without any further registration, the reconstruction would not be correct (see Figure 3.13c). By reversing Frame B's PCA alignment stage on the frame in Figure 3.13d the full FVR-3D registration output is revealed (Figure 3.13d). This registration is very accurate in comparison to zero registration being performed prior to integration, which is shown in Figure 3.13c.

$$R_{x,y,z}STMatrix = C_B^{-1} \times R_yST \times C_A \quad (3.35)$$

FVR-3D is a capable registration method which extends Fourier registration to take into account full 3D rotation. The complexity of the technique adds a single layer of complexity over the FVR method. The additional computational complexity (N^3), which is primarily made up of computing the covariance matrix, is minimal compared to the rest of the FVR technique.

Algorithm and Pipeline

The proposed SLAM/3D reconstruction routine for the FVR-3D is almost identical to the routine for the FVR method in Listing 3.2 and is shown in Listing 3.4. First, the 3D frame f_1 is read in and projected to construct a point cloud *PointCloud*. *PointCloud* is then integrated using volumetric integration into the global reconstruction. Matrix M is initialized to the identity matrix to accumulate the registration matrices so future frames may be efficiently integrated. $Camera_{location}$ and $Camera_{pose}$ accumulate camera location and 3D rotational pose, respectively, and are combined and listed in *Cameras* on a per frame basis. These variables may be omitted in the case of 3D reconstruction, or output in the case of 3D SLAM. Next, a loop is used

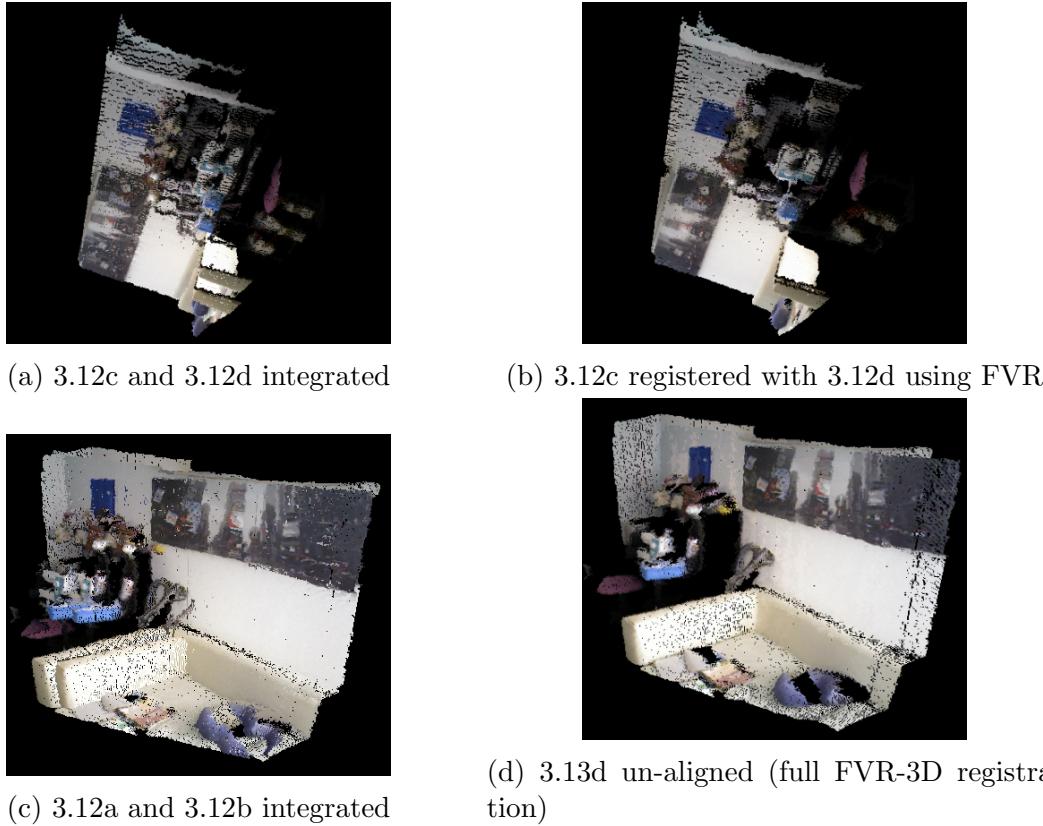


Figure 3.13: Final stages of FVR-3D

for each new frame input into the system. In this loop, 3D frame f_2 is read in. Both f_2 and f_1 are projected and voxelized into V_1 and V_2 , respectively.

Next, the FVR-3D method is used in place of the FVR method to compute the full six (plus 1 for scale) degrees of freedom required for 3D SLAM or 3D reconstruction. A temporary variable $Temp$ is used to hold this rigid transformation matrix. M is then updated to take into account the previ-

Listing 3.4: Phase Correlation Based SLAM Algorithm

```

f1 = ReadFrame();
PointCloud = project(f1);
GlobalReconstruction.integrate(PointCloud)
M = IdentityMatrix();
CameraLocation = [0, 0, 0]T;
CameraPose = [[1, 0, 0]T, [0, 1, 0]T, [0, 0, 1]T];
Cameras = [[CameraLocation, CameraPose]];
while(more frames){
    f2 = ReadFrame();
    points1 = project(f2);
    points2 = project(f1);
    V1 = Voxelize(points1);
    V2 = Voxelize(points2);
    Rx,y,zST = FVR - 3Dθφtxtytz(V1, V2);
    Temp = Rx,y,zST;
    M = M × Temp;
    points1 = Transform(points1, M);
    GlobalReconstruction.integrate(points1);
    CameraLocation = Temp-1 × CameraLocation;
    CameraPose = Temp-1 × (CameraPose + CameraLocation);
    CameraPose = CameraPose - CameraLocation
    CameraPose = CameraPose / CameraPose - CameraLocation;
    Cameras.add([CameraLocation, CameraPose]);
    f1 = f2;
}

```

ous registration transforms plus the current one. Using the updated transformation matrix, M is applied to $points_1$ to register it with the current global reconstruction, then integrated with the final reconstruction. Next, $Camera_{location}$ and $Camera_{pose}$ are updated and the new camera pose and location values are added to the $Cameraslist$.

Figure 3.14 shows a pipe-line for the FVR-3D technique. Here, two 3D frames are input as $Volume_1$ and $Volume_2$. The second frame may be taken after the camera has changed location and pose. The two frames may also be taken by different cameras with differing projection procedures projecting depth map values into frames. First the alignment matrices C_a and C_b based on the principal axis are computed by the PCA algorithm. These matrices are then used to align $Volume_1$ and $Volume_2$ to a common axis. Only then can the proposed FVR method be used to register the final axis of rotation along with scale and translation changes. Finally a full registration matrix may be formed by the alignments matrices and the output registration matrix via the FVR procedure.

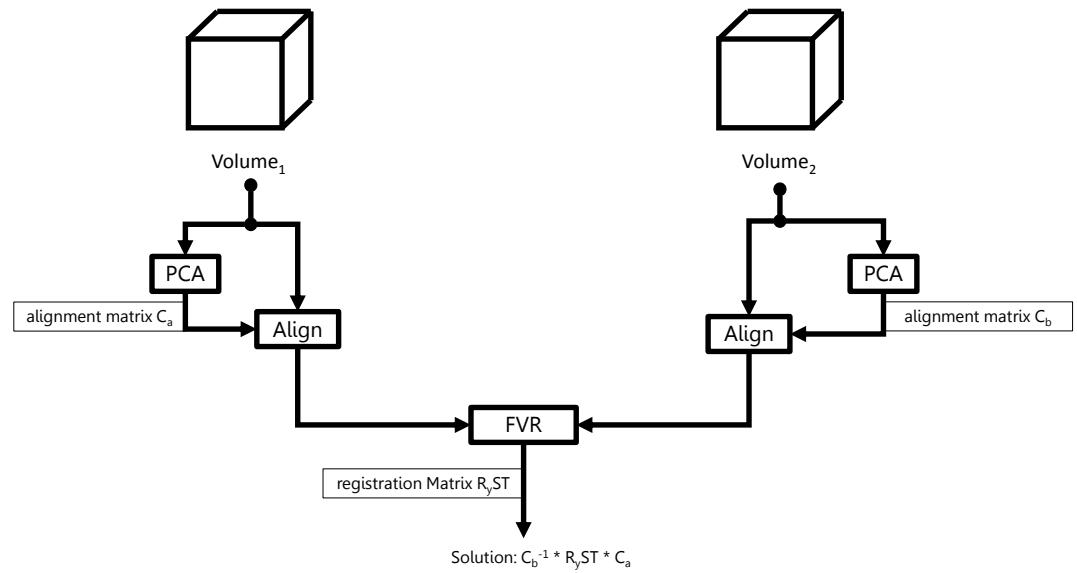


Figure 3.14: System Diagram for Registration Process

Limitations

Experiments (sections 4.4 and 4.5) show that the FVR-3D algorithm is a capable 3D reconstruction method in terms of accuracy, however, it is less robust to noise and wide-baselines than the FVR method. This is due to PCA being much more affected by noise than the FVR method on its own. Therefore, whilst the FVR method is limited to y-axis rotational registration, it is more robust under noisy conditions. To solve this noise issue associated with FVR-3D, the plain FVR method is also used whenever FVR-3D is used and the best registration is chosen on a per frame basis. Despite the reduced accuracy at wide baselines or when more noise is present, the FVR-3D

method outperforms several state-of-the-art algorithms used in registering 3D frames produced by different sensor types, including: stereo and active camera set-ups.

3.2 Plane-Tree Data Compression

In this section, a novel 3D frame data representation and compression method is described. It was designed to help reduce data size whilst facilitating data processing. This novel representation is named Plane-Tree and is an extension of the basic octree 3D data representation and was inspired by the Shade-Octree (SOT), a data structure developed during in the author's honours thesis [186]. More detail about the SOT can be read in [186].

The Plane-Tree compresses 3D data and it was also inspired by the Interpolating Leaf Quad-Tree and Shade-Tree representations [1, 187] which are used for image compression. These techniques make use of Quad-Tree decomposition and have been shown to outperform several transform based methods of compression.

3.2.1 Octree Overview

In this section octree decomposition is explained. This strategy forms a cubic shaped space around some 3D data. In the case of 3D reconstruction, it may be the 3D frame input from an active sensor, stereo method or monocular depth estimation procedure. A geometric representation is then computed for the data contained within the cubic space. This representation may be as simplistic as recording the cube's centre or using the cube to represent the space, or may be complex, employing splines and curvature representation techniques to represent the space.

Whatever the representation, a measurement system must be used to decide whether the computed geometric representation adequately fits the data within the space. If the geometric representation is accurate enough for application purposes it may be used in place of the data. This representation is typically designed such that the amount of data required for representation of a space is less than the amount which would be used otherwise. Alternatively, the representation may make use of known correlations which may be more easily compressed than the initial representation.

If, by the error generated via the measurement system, it is decided the representation is not suitable, the cubic space is broken down into 8 sub-

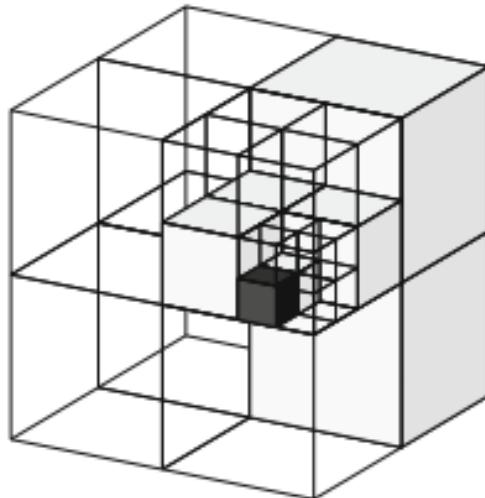


Figure 3.15: Visualization of cubic space being split [188].

cubes and the process is repeated. For an example of the space broken down into sub-cubes see Figure 3.15.

At each level of decomposition, the data representation achieves finer detail, but more nodes must be stored which means less compression. This is a classic trade-off between compression and accuracy in representing the underlying object. An example of the octree's use in representing an object at differing levels of accuracy is shown in Figure 3.16. In the next section, the decomposition procedure is discussed further. Section 3.2.2 describes the geometric representation employed by the Plane-Tree.

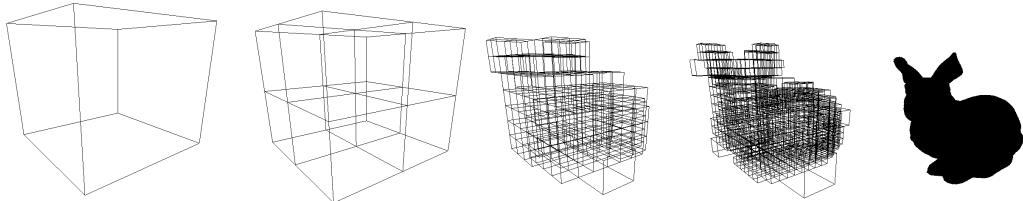


Figure 3.16: An Example of an octree used to represent an object at differing levels of accuracy [188]. Left: Original model, Middle: few node splits used to represent the model. Right: many node splits used to more accurately represent the model.

Octree Description

The octree is the three-dimensional extension of the two dimensional Quad-Tree (QT) technique. First we provide some information about the QT and the idea behind using planes or shading the cubic (or square in the case of the QT) nodes to improve compression and efficiency. This was the fundamental idea behind the Shade-Tree and ILQT algorithms [1, 187]. Both these techniques have been shown to improve compression in comparison to state-of-the-art transforms and form the basis of reasoning behind the Plane-Tree.

The QT is a hierarchical data structure used for processing and compressing 2D data. Figure 3.17 shows an example of a QT used to represent an image. In this figure, the original image is on the left. The QT first uses a single coloured square to represent the entire image, then using some error

metric it decides whether to, a) represent the image more accurately using more memory or b) stop decomposition and represent the image with a single colour. If option b) is chosen, the image will look like the second left picture. If option a) is chosen, the image is divided into four sub-images each with its own colour. From here the whole process begins again, with each sub-image given the same options. The final product of the QT is shown on the far right in Figure 3.17 and a visualization of a QT hierarchy is shown in Figure 3.18.

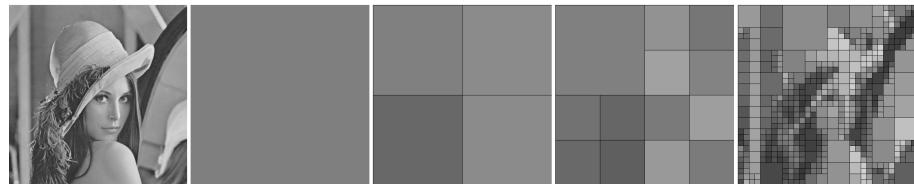


Figure 3.17: Quadtree Image Representation, left to right: Original Image, 1st Level of Decomposition, 2nd Level of Decomposition, 3rd Level of Decomposition, QT Codec Image.

The ILQT and Shade-Tree use small amounts of memory at the corners of the quadrants and interpolate to generate the underlying structure. The small data structures used for representation are much more compressed in their representation than the raw pixel data. For this reason, the ILQT and Shade-Tree do not need to decompose as much as other representations and allow for greater compression as proven by their ability to compete with

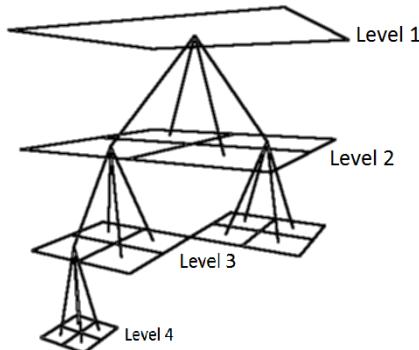


Figure 3.18: A visualization of the QT hierarchy

transform based methods.

As mentioned previously, the OT works identically to the QT but in 3D. Instead of surrounding the 2D image with a square encompassing the entire frame, a cube is used to enclose the 3D space. Similar to the QT method, an error metric is used to decide whether the current representation is adequate. If the octant must be split, it is divided into eight sub-octants. Each split operation divides each coordinate (x, y, z) by two, giving the 8 sub-octant spaces. Figure 3.19 shows an example of this decomposition process. The further each cube is split, the more the OT resembles the model being compressed. In the QT and OT, nodes which have not been split are called leaf nodes and the function which decides whether a node should be split is called a leaf criterion function.

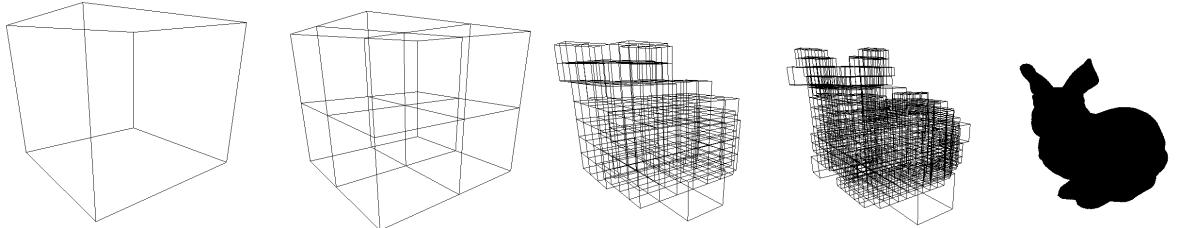


Figure 3.19: Visualization of OT Decomposition

The goal of the OT and the QT is to implicitly store the size and attributes of each node. There are two main methods of representation, a packed traversal of the tree and a linear tree. The packed traversal method stores the hierarchy using a pre-defined traversal of the tree, and typically follows one of two orderings. These are, breadth-first traversal and depth-first traversal, as seen in Figure 3.20. The depth-first traversal starts at the root, then works its way down, top to bottom, then left to right of the tree. In essence, it travels to a node's children before considering its neighbours. In contrast, the breadth-first traversal goes to all nodes which are at the same depth before continuing to lower levels.

Encoding the two packed tree traversal methods requires one nibble (4 bits) for a QT node, and one byte (8 bits) for an OT node. Here, the bits represent the structure of the sub-node. For example, the QT node bit sequence, 1001_2 means that the node has two children (corresponding

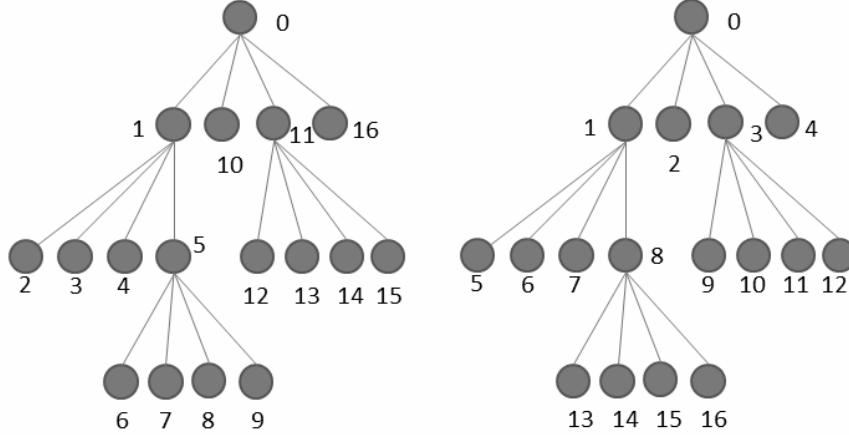


Figure 3.20: Tree Traversals, Left: Depth-First Traversal, Right Breadth-First Traversal

to the ones), the sequence 0000_2 means the node is a leaf node. Each bit position indexes a child node using a pre-determined ordering. An example of a possible ordering for a quadrant and an octant is shown in Figure 3.21, both breadth-first and depth-first traversals use the same index method, the difference lies only in the order in which nodes are visited.

The linear tree representation stores each leaf node individually. Each leaf is encoded as the pathway from the root to the leaf itself. This method was investigated in the 1980s [189, 190], but has not been used much in modern compression algorithms due to its inefficiency compared with the packed traversal method. An example of this structure is shown in figure 3.22, where each node: a, b, c, \dots, m is encoded as a variable length traversal path. The QT and OT are widely used in image [191] and 3D compression

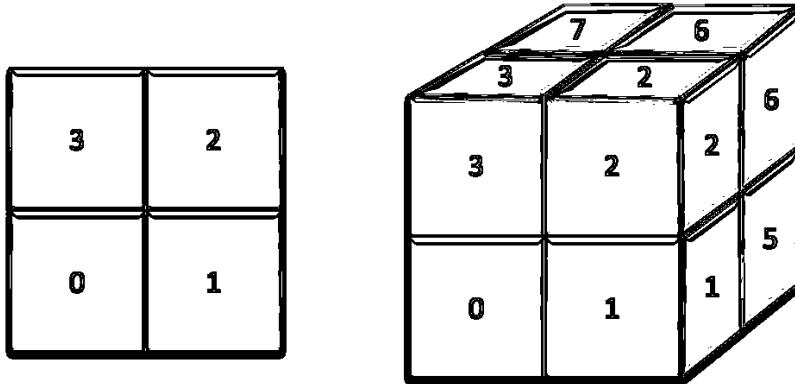


Figure 3.21: A possible child ordering for the QT (left), and the OT (right).

[192]. Hanan Samet [193] presents an introduction to both structures, he also describes both tree storage methods in depth.

3.2.2 3D ShadeTree Coding

The Shade-Tree compression system [187] was designed for the compression of 2D image data. However, this method is easily extended to 3D Volumetric data. In this system, octants are decomposed in the same manner as with a regular octree but the leaf node representation is different. In the Shade-Tree system, the corner values in the volume are sampled for each node.

To model the geometric information within each node using the corner values, interpolation is used. By interpolating between the 4 corner values, a model of the data within the geometric subspace within the node may

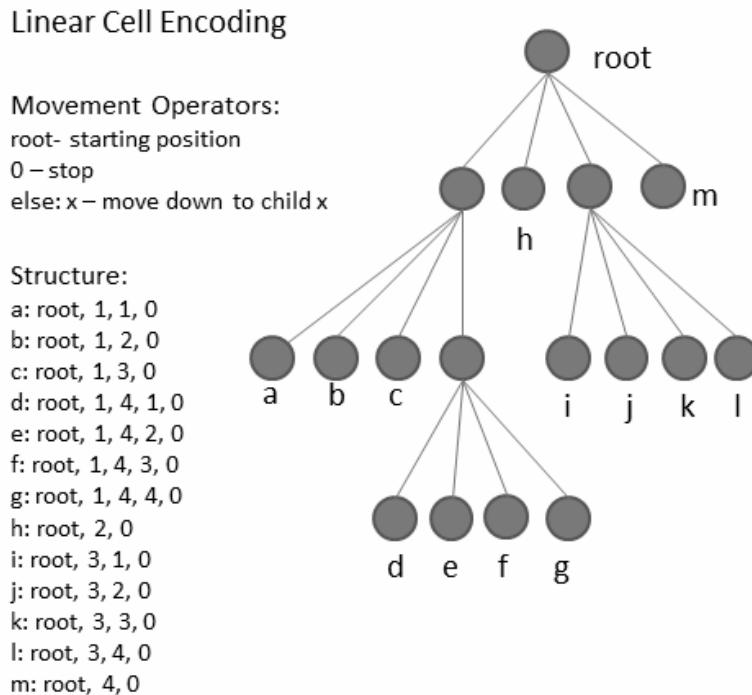


Figure 3.22: A Linear Tree Code Representation Example

be generated. This representation saves space by storing only 4 corner values rather than explicitly storing the dense volumetric data. As in the 2D Shade-Tree, this method allows nodes to share data and further compress the structure. For example, if two leaf nodes happen to share a corner, then the corner only needs to be encoded at the node which is visited first according to the ordering of the tree (breadth-first or depth-first). This constitutes no additional coding for the representation.

The Shade-Tree coding method can quantize the corner values and map them into values within the range [0, 255]. This way the Shade-Tree requires four or fewer bytes per leaf node. The Shade-Tree is ideal for 2D images or volumetric 3D data. In this research I aim to target sparse 3D volumetric and sparse 3D mesh data. Therefore, another data structure based on the octree, the Plane-Tree is proposed. This data representation is similar in idea to the Shade-Tree but was designed for low bit-rate compression of sparse 3D data as well as occupancy grids which are commonly generated by 3D reconstruction algorithms.

3.2.3 Plane-Tree Coding

The proposed Plane-Tree method is also based on octree subdivision. The technique places the input 3D frame/mesh or volume into a cubic space. It checks whether the representation corresponding to this single cube is at a desired level of quality. If it is not, the cube is decomposed into 8 sub-cubes. For each sub-cube associated with the 3D data, the process repeats. This process is further controlled using an error threshold and a maximum depth value. At each level of subdivision, the error between the sub-cube (or node) representation of the space is compared to the original geometric information within the sub-space of the cube. If the error is below the error

threshold, decomposition stops. This way, the desired level of accuracy can be more easily controlled rather than manually inspecting the quality prior to subdivision. Likewise, the level of subdivision may freely continue until the maximum tree depth is reached, this also assists in controlling the quality vs. compression ratio. To this end, the depth of each node is compared to the maximum depth. If the maximum depth is reached, decomposition stops and other nodes are visited until the procedure ends.

In the typical octree node representation, the raw cube is used to represent the space. Unless this cube is small (deep within the tree) the error is typically high. Trees which are very deep require more storage space. Our method stores arbitrary first order planes within nodes at 20 bits per leaf. This small bit cost per leaf node greatly improves compression performance compared to the octree and makes it competitive with state-of-the-art methods.

In the following sections, we present the details of the Plane-Tree in terms of subdivision, leaf node plane computation and representation as well as compression and decompression.

3.2.4 Octree Subdivision

Prior to compression, the input 3D model is normalized into a 512^3 space.

Starting with the cube which represents the entire space, a 3D Plane representation is computed based on the data within the cube. The mean squared error metric is computed between the sampled points of the plane and the sampled points of the 3D object (which lie within the cube). If this value is below a given threshold, or the maximum level of subdivision is reached, decomposition stops. Alternatively if both predicates (maximum depth and error threshold) are not met, the cube is divided into 8 sub-cubes. Each sub-cube is then tested to see whether part of the mesh lies within it. If so, the process is repeated for that cube, otherwise no further action is taken.

Each cube/sub-cube is referred to as a node. Each node which has no children is referred to as a leaf node. During compression/decompression, the plane based representation is only stored at leaf nodes, with non-leaf nodes serving only as paths giving the location of leaf nodes. Below, the computation and representation of our novel leaf node data model is discussed.

3.2.5 Leaf Node Computation and Representation

Our novel leaf node representation better represents the mesh which intersects it than the octree, it does so by using a first order plane. This representation requires only 20 bits, allowing it to encode higher quality models whilst saving on bits which would otherwise be used to form a deeper, and thus more costly, octree representation. This also gives the Plane-Tree an advantage at low bitrates.

To generate a plane for a given leaf node, we first sample the mesh within the node space. Using these points, the x, y and z axis variances are measured. These indicate how much variation lies across each axis within the node. We then find the plane using least squares, solving for the axis with the lowest variance. Once we have coefficients describing the plane, we use a single point on the plane, and the plane normal to describe it.

Using a point on the plane and a plane normal, we can find a set of triangles which represent the plane within the node. We first find all points in which the cube's edges intersect with the computed plane. Using the average point as the origin, and the plane's normal as the y-axis, we order the points based on their x/z angle. This gives us an ordered set of points which corresponds to a polygon defining the plane within the node. This polygon

is then triangulated forming the final representation. This process is used at both the compression and decompression stages.

The triangles representing the plane within the node are then sampled along with the parts of the mesh lying within the node. The mean squared error (mse) is then taken between these samples for comparison with the error threshold value. Within the summation for the mse we use the closest point within the second point set, this is shown in equation 3.36.

$$MSE(pts_1, pts_2) = \frac{1}{N} \sum_{i=0}^N (pts_{1i} - \text{closest}(pts_{1i}, pts_2))^2 \quad (3.36)$$

Using this equation, the sampled points from the plane triangles p , and the sampled points from the mesh m (which lie within the cube), we take the average of both mean squared errors as a measurement of total error, $\text{error} = \frac{1}{2}MSE(p, m) + \frac{1}{2}MSE(m, p)$. This is the value which is compared with the error threshold to decide whether decomposition should stop.

To compress the data for our leaf node representation, we store the plane using its normal vector and a single point lying on the plane. We store the normal vector using 12 bits (4 bits per coordinate). The point which lies on the plane is represented using two pieces of information, an edge number

and a distance variable. It must be mentioned that for a given plane which intersects a cube, a minimum of 3 of the cube's edges pass through the plane. We therefore record one of these edges (the edge number) and the distance from on of its end points (the distance variable). Each edge has a predefined number which identifies it, all edges also have a predefined start and end point.

Each cube has 12 edges in total, so 4 bits are used for the edge number, another 4 are used for the distance variable. Adding these to the normal vector results in 20 bits per leaf node representation.

3.2.6 Compression and Decompression

To compress our data structure we iterate through the tree in depth-first order. If we encounter a non-leaf node, we first store a single bit of 1_2 . Then the configuration of the sub-nodes (since not every sub-node intersects the object) is stored as a single byte. Each bit is labelled 1_2 if a particular sub-node exists and 0_2 if it does not. This is possible since we order our sub-nodes in a predefined order. If a leaf node is encountered, we store a 0_2 , then record our 20 bit leaf node representation in three other files: the normal data, the distance variables and edge numbers, totalling 4 files (tree,

normals, distances and edge numbers). After the entire structure is stored we employ entropy encoding on each of these files.

To decompress our structure, we read the first bit of the output file. This step checks if the current node is a leaf or not. If it is, we read out a 20 bit plane representation (from the three separate files) and generate a list of triangles representing where the plane intersects the node (explained in section 3.2.5). These triangles are then added to a final database which represents the decoded model. If we reach a non-leaf node, we read out the 8 bit sub-node configuration and repeat the process for each existing sub-node in the predefined order.

3.2.7 The Plane-Tree vs. the SOT

The Plane-Tree differs from the SOT in many ways. In both representations, the leaf nodes are represented as flat planes, but the computation and storage of the leaf nodes differs. The SOT computes planes using the average normal and location within and around a node whereas the Plane-Tree uses least-squares decomposition to compute the plane. The SOT stores a directional code and four distance codes per node, but nodes may share distance codes. The Plane-Tree uses a quantized distance vector, a single distance code and

a coded edge. Compared to the SOT, the Plane-Tree has better compression and representation of leaf nodes.

4

Experiments

4.1 Experiments

Several experiments were designed to evaluate the FVR, FFVR and FVR-3D methods as well as the Plane-Tree and 3D Shade-Tree methods. These experiments as well as their conditions and results are presented in this section. In the next section (section 4.2), the tools used to run the experiments are introduced along with various error metrics (section 4.3) which provide means for comparisons between algorithms.

In experiments on reconstruction algorithms, results from three differ-

ent sensor types are presented. In section 4.4.3 FVR based techniques are compared to algorithms widely used in the literature in terms of stereo camera based 3D reconstruction. Section 4.4.4 presents results using active RGB sensors based 3D reconstruction and section 4.4.5 presents results for monocular camera based 3D reconstruction. To test stereo sensor data, the KITTI Vision Benchmark Suite was used [194]. For active sensor and monocular camera tests, data were specifically generated for these tests using an Asus Xtion Pro Live camera.

These experiments compare the performance of the FVR based techniques with the presently used algorithms in registering different types of sensor data. Results show that the FVR and FVR-3D outperform several state-of-the-art algorithms in terms of accuracy and noise robustness. Section 4.5 presents results comparing the FVR based methods in terms of registering specific wide baseline camera movements, translations of 5-15 cm and rotations of 10-30 degrees. Results show that the FVR method can outperform others in terms of reducing registration error.

After these experiments, the Plane-Tree is compared to the octree. These experiments compare both methods' efficacy in compressing 3D mesh data. The Plane-Tree is also compared to various state-of-the-art compression tech-

niques from the literature. Qualitative compression results are also presented comparing the Plane-Tree to several other methods. Finally, the Plane-Tree is compared to both the octree and the 3D-ShadeTree in compressing 3D reconstructions.

4.2 Tools

In the experiments, various machines, devices and software were used to generate and perform tests. This equipment is discussed here. All experiments were performed on two machines. One is an Asus laptop running both Windows 10 and Ubuntu 16. This laptop has an Intel i7 CPU and an NVIDIA GeForce 840 M GPU with 4 GB of RAM. The other is a Dell desktop computer running Windows 7. This machine has an Intel i5 CPU and 4 GB of RAM.

Both the Visual Studio C/C++ compiler on Windows and the GCC C/C++ compiler on Ubuntu were used to write programs for testing purposes. C++ version 17 was primarily used to write programs. Libraries used include: the OPEN-CV 3 computer vision library used for capturing, writing and processing image data, and CUDA used for writing General Purpose GPU (GPGPU) programs. Both Visual Studio (12 and 15) and Code

Blocks 16 were used to write programs using the libraries and compilers. The METRO 3D object comparison tool was used for comparing 3D objects in Plane-Tree experiments.

To capture depth (RGB-D) video frames, the Asus Xtion Pro Live active camera was used to capture 640×480 video frames at 30 frames per second. This camera can capture depth between 0.8 and 3.5 m and was used to capture most of the test-data used in FVR experiments.

All source code and experiments are available online. The link for the FVR method is <https://github.com/lukes611/phdThesis> FVR and the link for the Plane-Tree source code is <https://github.com/lukes611/PlaneTree> Plane-Tree. Further discussion about the metrics used for testing these algorithms is given in section 4.3.

4.3 Error metrics

Several metrics are used to assess the set of proposed 3D reconstruction algorithms as well as the proposed Plane-Tree and other compression methods. These metrics are presented and defined mathematically here. Assessing both the 3D reconstruction algorithms as well as the lossy 3D data compression

techniques requires comparing output 3D models. In the case of 3D reconstructions, we can compare the error between the registered frame and the ground truth, the larger the error, the worse the registration. Alternatively, a measurement may be used to measure the difference between two frames before and after they are registered. If the error is reduced or almost zero after registration, then the registration method may be deemed correct. If the error is larger or almost the same post registration, the registration maybe considered incorrect or there possibly was little camera movement.

In both cases, a robust way to measure the similarity/difference between two 3D objects must be used. To compute this error, we use nearest neighbour functions to measure the closest point in one model to the closest point in the next model. For example, given model A and model B, the closest point B_j in B for point A_i in A , is used in summation of the total error. The closest point B_j to A_i is simply the nearest neighbour of A_i . This can be formally described as a function of the 3D point list V and a given point p . This is described mathematically in equation 4.1. Here, the result is a point q in V , in which the distance between q and p , according to function $Dist(x, y)$, is shorter for q given p than any other point k in V .

$$NN(p, V) = \{q \in V | (Dist(q, p) < Dist(k, p)) \forall k \in V\} \quad (4.1)$$

For all purposes within this research, the function $Dist(x, y)$ is simply the Euclidean distance between the two input points. Using this definition of a nearest neighbour, several metrics may be used to compute the distance between two models. The one-way distance between two models is defined as the summation of distances for each point in one model to its nearest neighbour from the other model. The one-way mean error between two 3D models, P and Q is given in equation 4.2. The full mean error between two objects is then computed as the average of the $Mean - Error_{one-way}$ function from model P to Q and model Q to P . This is defined in equation 4.3.

$$ME_{1-way}(P, Q) = \frac{1}{N} \sum_{k=0}^N Dist(P_k, NN(P_k, Q)) \quad (4.2)$$

$$ME(P, Q) = \frac{ME_{1-way}(P, Q) + ME_{1-way}(Q, P)}{2} \quad (4.3)$$

Other metrics may also be used, for example the mean squared error can be used instead of mean error. The mean squared error (MSE) may then replace the mean error used in equation 4.2. The one-way and full error functions based on the MSE are provided in equations 4.4 and 4.5.

$$MSE_{1-way}(P, Q) = \frac{1}{N} \sum_{k=0}^N Dist(P_k, NN(P_k, Q))^2 \quad (4.4)$$

$$MSE(P, Q) = \frac{MSE_{1-way}(P, Q) + MSE_{1-way}(Q, P)}{2} \quad (4.5)$$

The mean error based on the nearest neighbour is often referred to as the Hausdorff error. In this research, we used the Hausdorff error and the Mean Squared Error (MSE) based on the nearest neighbour technique, as well as a percentage of total matches. The one-way percentage of total matches is the computation of the percent of points from one model which have a nearest neighbour with a distance below a given threshold. This metric is defined in equation 4.6.

$$PM_{1-way}(P, Q) = \frac{100}{N} \sum_{k=0}^N x, \text{ where } \begin{cases} x = 1 & \text{if } Dist(P_k, NN(P_k, Q)) < \text{threshold} \\ x = 0 & \text{otherwise} \end{cases} \quad (4.6)$$

Following this, the full Percent-Match function may be defined as the average of the PM_{1-way} function in both directions. These three metrics are used along with several others in 3D reconstruction experiments. Addition-

ally, the camera tracking error is computed as the euclidean distance between the ground truth camera movement from one frame to another and the estimated camera movement computed via the FVR method. The voxel error is also measured. Since the camera distance metric measures the real positions moved by the camera, it theoretically has the accuracy of the real number system. Conversely, FVR uses voxel spaces (3D volumes) to find the translation and rotation (location and pose) between frames. Therefore, the result must be quantized. By quantizing the ground truth prior to comparison, the accuracy up to the resolution of the voxel grid may be computed. This is important since it measures the error of the FVR 3D reconstruction method without penalizing it based on the effects of quantization which reduce the larger the volume.

In experiments comparing the Plane-Tree with state-of-the-art algorithms, the root mean squared error is also used. The root mean squared error is simply the square root of the mean squared error function defined in equation 4.5, that is $RMS(P, Q) = \sqrt{MSE(P, Q)}$. The Plane-Tree is also assessed in terms of 3D reconstruction occupancy grid compression. To assess the difference between the original 3D occupancy grid and the lossy compressed version, the Peak Signal to Noise Ratio metric (PSNR) is used. This is a function of the voxel-wise mean squared error function. Given the entire

3D volume, the mean squared error is computed between the original 3D reconstruction and the compressed 3D reconstruction. The mean squared error over two 3D volumes is defined in equation 4.7 as the summation of the voxel-wise squared error divided by the number of voxels.

$$MSE_{volume}(V_1, V_2) = \frac{1}{N^3} \sum_{z=0}^N \sum_{y=0}^N \sum_{x=0}^N (V_1(x, y, z) - V_2(x, y, z))^2 \quad (4.7)$$

The PSNR metric is then defined as in equation 4.8. This metric is commonly used in image compression comparisons and since 3D images are being compared in this test, it is used here also. The PSNR metric is inversely related to error, so the larger the PSNR the lower the error between the two models.

$$PSNR(V_1, V_2) = 10 \times \log 10 \frac{255^2}{MSE_{volume}(V_1, V_2)} \quad (4.8)$$

4.4 Data Sources

4.4.1 Algorithms

To further assess the set of proposed FVR based registration techniques, other frame registration techniques used in the literature were implemented for comparative purposes. It is important to compare the proposed algo-

rithms with these methods because they represent the current standard in 3D reconstruction techniques. The techniques which are most competitive with the FVR based methods are the 2D Feature Matching + RANSAC method and ICP. The 2D Feature Matching + RANSAC method, discussed in section 2.2.2, performs 2D feature matching to compare frames. Using these feature matches along with their correspondences to the 3D points within the projected depth maps, RANSAC is used to filter outliers and iteratively estimate camera location and pose. It is important to compare this method with the FVR based techniques because it is dominant within 3D reconstruction as well as in other areas of image processing and computer vision. In these tests, SURF was used with default values using OpenCV, this included using 3 octaves and selecting the top 200 feature matches. In RANSAC, a maximum of 500 iterations were used, and 4 random points were used to compute a camera pose model at each iteration. Our experiments indicate that the FVR techniques outperform this 2D Feature Matching + RANSAC method whilst remaining closed form solutions which are often more robust in certain situations (such as reconstructing scenes with little texture or scenes where texture confusion is prevalent). Through empirical tests, SURF was chosen as the optimal 2D feature matching algorithm to compare with the FVR techniques.

In addition to comparisons with 2D Feature Matching + RANSAC, 3D Feature Matching + RANSAC (discussed in section 2.2.5) was also implemented and compared. In 2D feature matching, the features are found and matched between a pair of 2D image frames and then RANSAC is used to compute camera pose. The camera pose is then used to reconstruct the scene. The 3D Feature Matching + RANSAC technique uses the same concept, but features are matched between the projected 3D frames rather than the 2D frames captured with standard cameras. The distinction between these approaches is that 2D feature matching is considerably faster, whilst 3D feature matching can handle wider baselines and takes into account certain spatial information which may give it an edge over the 2D feature matching approach. This advantage is greatest in scenes with less texture or repeated local textures (texture confusion). SIFT-3D was chosen as the 3D feature matching method to compare with the FVR techniques. This decision was made based on empirical testing. Due to computational power limitations, only a single octave is used, with Gaussian window sizes of 5.

The Iterative Closest Point algorithm (discussed in section 2.2.3) is another technique which has dominated 3D reconstruction and SLAM research. The ICP method works by iteratively refining a registration between two models based on each point's nearest neighbour. ICP is popular within both

3D reconstruction and SLAM research because of its accuracy and consistency. It also works well on most scene types. The major disadvantage of ICP is that it gets stuck in local extrema if there is too much noise, or too wide a base-line. In either of these cases, ICP fails to register frames. ICP is also known to fail in cases where too few features are present. In these experiments, ICP is used without pre-filtering the points and all points are used in each iteration with a maximum of 500 iterations.

Another algorithm present in the experiments is the Principal Components Analysis (PCA) method. PCA is used to find the mean and principal components of multi-dimensional data. The PCA method, used for registration purposes, uses the computed principal axes and mean to register two 3D frames. This method is useful for registration purposes as it works on wide baselines, is fast and provides additional information about the scene (in the form of principal components and a point representing the origin, the mean point). The downside to this method is that it is very susceptible to noise and misaligned data resulting in inaccurate and useless mean and secondary axes. As the proposed FVR-3D method makes use of the primary axis information from PCA, it is important to compare the two methods to determine what improvements are made by FVR-3D over a standard PCA approach.

These algorithms from the literature were compared with the set of FVR based techniques: FVR, presented in section 3.1.2, FVR-3D presented in section 3.1.5 and FFVR presented in section 3.1.4. These three algorithms are used to compute the registration between two frames. The FVR method is most robust to noise and has the highest accuracy in terms of registration performance at wider baselines. However, it is only capable of handling a single axis of rotation at a time. Conversely, the FVR-3D method is capable of handling full 3D rotation, translation and scale but is less accurate and robust. It is also not able to handle as wide a baseline as the FVR method. The FFVR method is also only able to handle a single axis of rotation but is faster than the FVR method at the expense of accuracy and robustness to noise.

4.4.2 Sensor Types

Experiments show that the FVR based techniques are capable of registration across the three primary types of sensors used in 3D reconstruction and SLAM research. These sensors include stereo sensors, which are capable of generating dense depth data with the highest accuracy and resolution; active cameras, which are efficient hardware based solutions to depth generation;

and monocular cameras, which produce the least accurate and most noisy depth maps. As mentioned in the literature review these sensor types have associated advantages and disadvantages.

The results from testing the stereo Kitti data set are presented in section 4.4.3. These results show that FVR based methods, particularly FVR-3D are suitable for dense 3D reconstruction and are highly robust to noise, object movement and outdoor scenes. In particular, it is shown that the FVR-3D method is capable of outperforming other algorithms used in the literature in terms of registration accuracy. Below is a list of data included in the Kitti Benchmark data set, data used in experiments are documented in the list.

- Grayscale stereo images
 - Synced and Rectified
 - resolution: 1390 x 512
 - captured with a 0.5 mega-pixel camera
 - used in experiments
- RGB-Colour stereo images
 - Synced and Rectified
 - resolution: 1390 x 512
 - captured with a 0.5 mega-pixel camera
 - used in experiments
- 3D point clouds
 - generated using a Velodyne laser scanner
 - max depth: 120m
 - 100,000 points per frame
 - stored as a list of float based vectors in binary format
 - used in experiments

- 3D GPS / IMU data
 - location, speed, acceleration and other meta-data information
 - text format
- Calibration Data
 - Camera
 - Camera to GPU/IMU
 - Camera to Velodyne [used in experiments]
 - text file
- 3D object tracklet labels
 - cars, trucks, trams, pedestrians and cyclists
 - xml file

Section 4.4.4 presents the results of testing scenes captured using the Asus Xtion Pro Live active camera. Various types of scenes and camera movements were recorded and results show the FVR based methods perform robust and accurate 3D reconstruction. In particular, the FVR-3D can outperform other techniques from the literature, even when faced with a reduction in depth resolution when using the active camera as opposed to the stereo set-up tested with the Kitti Vision Benchmark data set. A list of data within this data set is shown below.

- RGB-Colour images
 - Synced & Calibrated
 - resolution: 640 x 480 (when calibrated), (1240 x 1080 max)
 - 30-60 frames per second
 - used in experiments
- 3D point clouds
 - generated using a structured light technique
 - depth range: 0.8m - 3.5m
 - low resolution: 640 x 480

- 30 frames per second
- used in experiments
- 3D GPS / IMU data

Lastly, section 4.4.5 presents the results of the MVVR method, the FVR extension for monocular sensor input data sets. Experiments revealed that the MVVR is not able to robustly handle rotation due to the noise generated when building the depth map, however, translation is possible. It can be said based on empirical results that the FVR based technique's accuracy and robustness increases with the resolution and accuracy of the depth maps used as input. The Kitti Benchmark data set was also used in these experiments.

4.4.3 Stereo Camera

In these experiments, several current techniques (2D Feature Matching (FM-2D), 3D Feature Matching (FM-3D), ICP and PCA) are compared to the FVR technique in terms of stereo camera registration accuracy. To this end, five data sets from the Kitti Vision Benchmark data set [194] were used. Each data set scene is a complicated outdoor environment filmed using an autonomous driving platform. The majority of frames contain moving objects which interfere with the registration process of several algorithms.

Each Kitti Vision Benchmark data set contains accurate laser scans,

stereo grey-scale images, stereo RGB-images and GPS and IMU data. To simulate the theoretical situation in which stereo cameras generate the most accurate depth maps, the laser scans are used in place of depth data computed by a stereo disparity algorithm. Therefore, in these experiments, only the laser scans and stereo colour and greyscale images are used.

The five stereo data sets used in the experiments are shown in Figures 4.1, 4.2, 4.3, 4.4 and 4.5. The first scene is the Kitti 0001 Sync Data Set, this scene has 107 frames. This data set contains 12 cars, two cyclists and a tram. The tram and cyclists are non-static objects within the scene, making it more difficult for the current set of reconstruction algorithms which rely on scenes being primarily static in nature. To the right in this data set, there is a tram-line and some trees and gardens, to the left is a residential street. The second scene is the Kitti 0002 Sync data set, this set is 76 frames long. It contains one car and two cyclists which are non-static objects. Within this scene there is a long brick wall hiding some tall trees, several cyclists ride along next to the wall. To the left, there are some grassy areas, some buildings and some parked cars.

At 153 frames, the Kitti 0005 Sync data set is also used. This scene contains nine cars, three vans, two pedestrians and one cyclist. Because of

the size of the van and the other non-static objects within the scene, this appears to be one of the more difficult scenes to register. The camera winds through several different streets as opposed to the previous data sets which are mostly of one long road. The fourth data set used, the Kitti 0091 Sync data set is composed of 339 frames making it the largest data set used in these experiments. This data set contains two cars, a van, 42 pedestrians, 14 sitters, eight cyclists and one miscellaneous object. As it contains many non-static objects this scene is also considered difficult like the Kitti 0005 data set.

The last Kitti Vision Benchmark data set used is the Kitti 0095 data set. Unlike the previous data sets, all 267 frames only contain static objects, and the scenes are made up primarily of small winding streets.

Experiments are tabled in full at per frame intervals in Appendix .1. Here, registration errors are presented where frame n is registered against frame $n + 1$ and the registration error is reported for each algorithm. The error function used is the Mean Squared Error $MSE(P, Q)$. This error is computed between the consecutive frames after registration. This error value is computed as in equation 4.9. Here, the function $Register(x)$ is replaced by the registration method being tested.

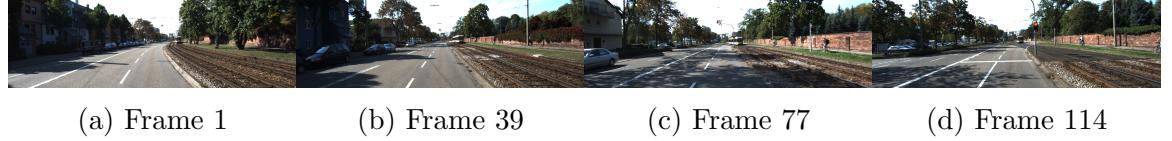


Figure 4.1: Kitti 0001 Sync Data Set Sample

$$\text{Error}(\text{frame}_1, \text{frame}_2) = \frac{\text{Register}(\text{frame}_1), \text{frame}_2}{\text{MSE}(\text{frame}_1, \text{frame}_2)} \quad (4.9)$$

The summary of these results is also tabled here for convenience. For each algorithm, the median registration error is provided. This is computed by listing and sorting the registration error values for a particular algorithm and selecting the value in the middle. Also included is the percent of best results metric. This measures in percentage, the frequency of times a particular algorithm achieved the best (lowest error) registration result compared to the other algorithms tested. An algorithm with a lower median error than another algorithm has performed better overall. Additionally, if an algorithm has a higher percentage of best results, it outperformed the other algorithms most of the time. If an algorithm achieved an average percentage of best results but a higher median error, this could be explained by outliers. If an algorithm achieved a lower median error but did not achieve the highest percentage of best results, it may be due to having a very competitive and consistent registration error.

Table 4.1 presents results for the Kitti 0001 Sync data set, some example colour frames are shown in Figure 4.1. The road in which this scene was filmed contains a tram-line and moving tram as well as a garden area to the right and a line of parked cars and houses under cover of shadows to the left. Registration statistics were taken over the full length of this data set, which is 107 frames. Results show that FVR-3D achieved the lowest median registration error, ICP achieved the next lowest followed by FM2D and the FVR method. FVR-3D also achieved the highest percentage of best frame registration results at 34.91 % compared to ICP with 27.36 %. If the FVR methods were combined as a single hybrid method, they would have computed the best registration result 55.67 % of the time. The FVR method alone outperformed FFVR, PCA and FM-3D. The FM-3D method (which had several frame registration failures, as is evident in its statistics) and the PCA method were the worst performers on this data set. The FFVR, whilst faster than the FVR method is slightly worse off in terms of performance.

Table 4.2 presents median error and percent best result statistics for the Kitti 0002 Sync Data Set, example colour frames are shown in Figure 4.2. The road in which this scene contains some parked cars to the left as well as a building, some grass and some large trees. To the right there is an orange brick wall and some trees and gardens as well as two moving cyclists. Results

Table 4.1: Reconstruction Errors for the Kitti Data 0001 Sync Data Set

Algorithm	Median Error × 1000	% best results
FM2D	5.28	13.21%
FM3D	9235.71	0.94%
ICP	5.15	27.36%
PCA	5.66	2.83%
FVR	5.5	13.21%
FFVR	5.59	7.55%
FVR-3D	5.1	34.91%

Table 4.2: Reconstruction Errors for the Kitti Data 0002 Sync Data Set

Algorithm	Median Error × 1000	% best results
FM2D	4.78	5.33%
FM3D	4.85	6.67%
ICP	4.43	30.67%
PCA	4.86	6.67%
FVR	4.67	6.67%
FFVR	5.23	5.33%
FVR-3D	4.27	38.67%

for the full 74 frames of the data set show that the FVR-3D method achieved the lowest median registration error at 4.27. The ICP algorithm achieved the next best result at 4.43 and the non-rotation-invariant FVR method achieved the third best result. Combined, the FVR based methods achieved the best result 50.67 % of the time compared to ICP at 30.67 %. In this scene, the FM-3D method did not have as many registration failures and achieved a better result than the FFVR method and a comparable result to the PCA method.

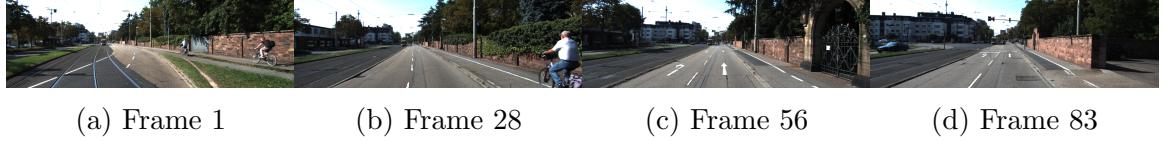


Figure 4.2: Kitti 0002 Sync Data Set Sample

Statistics for the Kitti 0005 Sync Data Set are presented in Table 4.3, example colour frames are shown in Figure 4.3. The scene captured in this data set was more difficult than previous scenes as it contains two moving cyclists and one large van which are all moving around in the scene without any relation to camera movement. In other words, these non-static objects cause major difficulties in most registration algorithms. In the full 152 frames of the data set, FM2D performed best with the lowest median error and highest percentage of best results. Next, FVR-3D also performed well with the second best median error and percentage of best results measurement. ICP achieved the third best median error and percentage of best results. In the results for this data set, FVR outperformed the FFVR method, as well as PCA and FM3D. Combined, the FVR algorithms achieved the best registration result 35.29 % of the time, which is still below the performance of FM2D on this data set.

Table 4.4 presents results for the Kitti 0091 Data Set, example colour frames from the data set are shown in Figure 4.4. This is the largest data

Table 4.3: Reconstruction Errors for the Kitti Data 0005 Sync Data Set

Algorithm	Median Error $\times 1000$	% best results
FM2D	3.39	39.22%
FM3D	3.83	0%
ICP	3.49	25.49%
PCA	4.06	0%
FVR	3.7	5.88%
FFVR	4.25	1.96%
FVR-3D	3.42	27.45%



Figure 4.3: Kitti 0005 Sync Data Set Sample

set tested at 339 frames. The scene filmed in this data set is that of an outdoors inner city. It contains many moving agents making it a scene which is difficult to register for most registrations algorithms. Specifically, it contains two cars, a van, 42 pedestrians and eight cyclists. Registration results show FVR-3D outperformed all the other algorithms in terms of both median error and percentage of best results. The FVR algorithm achieved the next best results. The FVR-3D algorithm achieves the best result more than twice as often as ICP and FM2D. The FVR based methods performed well on this data set, a hybrid approach would have achieved the best results 65.19 % of the time.

Table 4.4: Reconstruction Errors for the Kitti Data 0091 Sync Data Set

Algorithm	Median Error $\times 1000$	% best results
FM2D	3.6	17.11%
FM3D	4.04	0.88%
ICP	3.61	15.63%
PCA	4.1	1.18%
FVR	3.57	20.94%
FFVR	3.78	5.9%
FVR-3D	3.43	38.35%



Figure 4.4: Kitti 0091 Sync Data Set Sample

Results for the Kitti 00095 Data Set are presented in Table 4.5 and Figure 4.5 presents four example colour images from this data set. This scene was much more static than the previous scenes. It contains primarily parked cars and buildings in an inner-city environment. There are a few pedestrians and cyclists which are moving agents within the scene. This data set is 266 frames and results show that FVR-3D achieves both the best (lowest) median error score and the highest percentage of best results score at 46.82 %. The FVR algorithm achieved the second best results in terms of both the median error and percentage of best results metrics. ICP and FM2D were third and fourth, respectively. The FVR-3D algorithm achieved the best result three times more often than both the ICP and FM2D methods.

Table 4.5: Reconstruction Errors for the Kitti Data 0095 Sync Data Set

Algorithm	Median Error $\times 1000$	% best results
FM2D	4.19	12.36%
FM3D	5.18	0%
ICP	4.4	13.48%
PCA	5.32	0.37%
FVR	4.12	22.85%
FFVR	4.73	4.12%
FVR-3D	3.96	46.82%



Figure 4.5: Kitti 0095 Sync Data Set Sample

4.4.4 Active Camera

Experiments evaluating the performance of the set of FVR related algorithms on active sensor camera input are presented here. These data sets are all 24–25 frames long and captured different environments using different camera movements. Due to the limitations of the Asus Xtion Pro Live active camera used, primarily indoor scenes were captured. Similar to the results presented in section 4.4.3, statistics are presented for algorithms from the literature (FM2D, FM3D, ICP and PCA) as well as FVR based algorithms (FVR, FFVR and FVR-3D) in the form of the median registration error and the percentage of best results. Raw frame registration errors are presented in

Appendix .2.

All test data were generated using the Asus Xtion Pro Live camera. Each scene was captured using a specific type of camera movement. Some scenes were captured by rotating the camera about the x or y-axis, others by translating the camera. By testing with different movements, future algorithms may be constructed by switching to different registration methods based on camera movement. Additionally, a difficult scene containing areas prone to texture confusion was captured.

The first scene, the Apartment Texture Rotate scene, was taken by rotating the camera around the y-axis across an apartment. This scene contains a lot of texture information. The Apartment Texture X Axis Rotation scene is similar in terms of texture but contains x-axis rotation rather than y axis rotation. These different scenes test each method's ability to register 3D pose. The Office Textured Blind-spot Rotation scene is a textured office scene where the camera is rotated about the y-axis. The scene is focused on a large divider which separates two desks. The divider may confuse registration methods which rely too heavily on minimization by aligning the large divider as a priority rather than taking into account the smaller details within the scene. An example of such an algorithm would be ICP and its deriva-

tives. An Office Textured Items Translation and an Office Texture Rotation data set are also captured. These office scenes are standard and contain fair amounts of texture. Another scene, Parks, Plants and Table, was captured by translating and rotating the camera across a scene with plenty of texture confusion.

Table 4.6 shows results for the Apartment Texture Rotation Data Set, Figure 4.6 shows some example RGB image frames. The scene filled in these data frames are of an apartment living room, where the camera is rotated approximately 90 degrees over 25 frames. The scene contains an abundance of texture information. Results show that the FVR-3D method achieved the lowest median error result, making it the top performer in terms of this metric. FM2D performed next best, followed by the FVR method. The FVR algorithm outperformed all other algorithms except FVR-3D and FM2D on this data set. In terms of the percentage of best results metric, FVR-3D achieved the highest percentage at 36 %, FM2D was second at 28 %. A hybrid FVR method would have produced a 60% record of achieving the best registration result on these frames.

Results for the Apartment Texture X-Axis Rotation Data Set are presented in Table 4.7. Figure 4.7 shows some example image frames of this

Table 4.6: Reconstruction Errors for the Apartment Texture Rotate Data Set

Algorithm	Median Error $\times 1000$	% best results
FM2D	2.13	28%
FM3D	5.14	0%
ICP	2.42	8%
PCA	8.61	4%
FVR	2.26	16%
FFVR	2.7	8%
FVR-3D	2	36%



Figure 4.6: Four Sample Frames from the Apartment Texture Rotate Data Set

data set. This scene was captured by rotating the Asus Xtion Pro Live active camera about the x-axis. The scene contains a Christmas tree as well as some bags on the floor in the centre of an apartment room and some chairs to the right of the frame. Out of the set of FVR based techniques, only FVR-3D is capable of full 3D rotation registration, FVR and FFVR are only capable of being robust to this type of registration (handling it in the best way possible using translation, scaling and y-axis rotational transforms). Results show that the ICP method achieved the lowest median registration error with

Table 4.7: Reconstruction Errors for the Apartment Texture X-Axis Rotation Data Set

Algorithm	Median Error $\times 1000$	% best results
FM2D	1.97	4%
FM3D	2.58	0%
ICP	1.78	28%
PCA	3.81	0%
FVR	1.99	4%
FFVR	2.01	0%
FVR-3D	1.82	64%



(a) Frame 1 (b) Frame 10 (c) Frame 15 (d) Frame 20

Figure 4.7: Four Sample Frames from the Apartment Texture X-Axis Rotation Data Set.

FVR-3D achieving the second best result. In terms of the percentage of best results metric, FVR-3D outperformed ICP and the others 64 % of the time compared to ICP at 28 %. Interestingly, both FVR and FFVR outperformed PCA and FM3D despite these methods' abilities to register full 3D rotation (including x-axis). In terms of the percentage of best match metric, the set of FVR algorithms outperformed the others around 68 % of the time.

Table 4.8 shows the results for the Desk Texture Translation scene, Figure

Table 4.8: Reconstruction Errors for the Desk Texture Translation Data Set

Algorithm	Median Error $\times 1000$	% best results
FM2D	1.24	8%
FM3D	2.48	12%
ICP	1.59	28%
PCA	1.51	4%
FVR	1.16	16%
FFVR	1.29	16%
FVR-3D	1.23	16%

4.8 shows some example image frames for this data set. This scene includes a desk, a computer and computer monitor as well as several items including books, a hat and pair of glasses. The scene was captured by moving the Asus Xtion Pro Live active camera directly down the x-axis to the right in a purely translational movement. Results show that the FVR-3D method achieved the lowest median registration error, with ICP performing second best. In this test, the FVR method outperformed the others in terms of median registration error. This is likely due to the fact that the camera was moved in a purely translational way compared to previous tests. The next lowest median error value was achieved by the FVR-3D algorithm, followed by FM2D. Interestingly in the percentage of best results metric, ICP, which achieved a higher median error than all algorithms except FM3D, had the highest individual percentage of best results value of 28 %. Combined, the FVR, FVR-3D and FFVR methods achieved the best frame registration around 48 % of the time.



(a) Frame 1 (b) Frame 10 (c) Frame 15 (d) Frame 20

Figure 4.8: Four Sample Frames from the Desk Texture Translation Data Set.

Table 4.9 shows the results for the Office Textured Blind-spot Rotation data set. Figure 4.9 shows some example frames from this data set. The scene is identical to the previous one (results presented in Table 4.8, but the camera was rotated about the y-axis rather than translated about the x-axis). The scene contains a large blind-spot in which a part of the scene is hidden by a divider, the effect of which was to intentionally reduce overlap in terms of frame registration making it more difficult for these algorithms to compute registration accurately. Results show that the FVR-3D method achieved the lowest median error, but tied in second place with FM2D in terms of percentage of best results. In this data set, ICP achieved the highest percentage of best results.

Table 4.10 presents results for the Office Textured Items Translation Data Set, whilst figure 4.10 shows some example captured frames. This scene was captured of a large room with large pieces of furniture, a large screen and a

Table 4.9: Reconstruction Errors for the Office Texture Blind-spot Rotation Data Set

Algorithm	Median Error $\times 1000$	% best results
FM2D	1.39	24%
FM3D	5.92	0%
ICP	1.2	44%
PCA	4.83	8%
FVR	2.07	0%
FFVR	2.92	0%
FVR-3D	1.1	24%

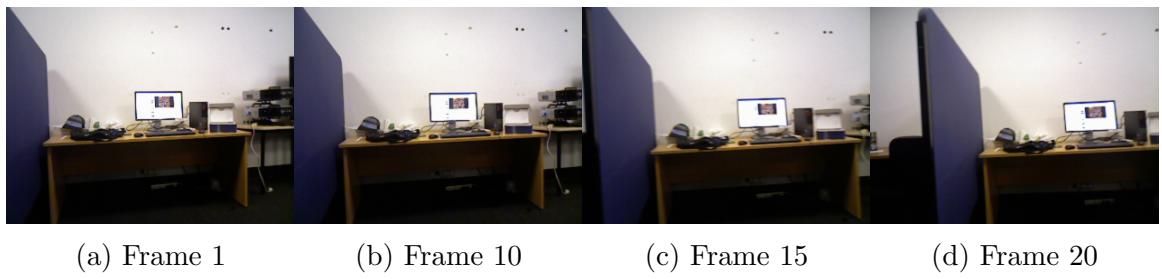


Figure 4.9: Four Sample Frames from the Office Textured Blind-spot Rotation Data Set.

set of desks and chairs. This is another scene in which x-axis translation was the primary camera movement. Results show that the FVR-3D algorithm achieved both the lowest median error and the highest percentage of best results. Combined, the FVR methods achieved the best result 72% of the time. After FVR-3D and FFVR, FM2D performed next best with the second lowest median error score and 3rd highest percentage of best result score.

Table 4.11 presents results for the Park Plants and Tables Rotation Data

Table 4.10: Reconstruction Errors for the Office Textured Items Translation Data Set

Algorithm	Median Error $\times 1000$	% best results
FM2D	2.89	24%
FM3D	5.45	0%
ICP	2.93	4%
PCA	3.79	0%
FVR	5.04	0%
FFVR	3.06	28%
FVR-3D	2.83	44%

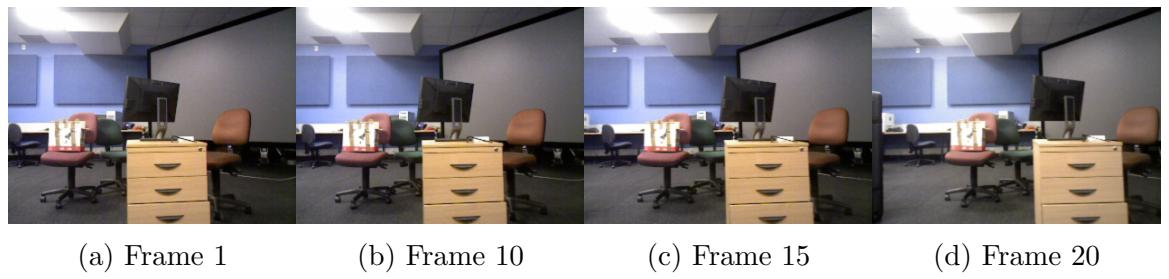


Figure 4.10: Four Sample Frames from the Office Textured Items Translation Data Set.

Set, Figure 4.11 shows some example frames from the data set. The scene captured was of an outdoors park area. In the scene, there is a metal table and metal fence surrounding a dense garden. This type of scene was expected to cause large amounts of texture confusion, as the metal fence and interspersed sections of garden appear to be similar locally. The primary camera movements were translation and y-axis rotation. Results are interesting and show that FM2D, FM3D and PCA were thrown off by this texture confusion. In terms of both median error and percentage of best result, the FVR method

Table 4.11: Reconstruction Errors for the Park, Plants and Tables Rotation Data Set

Algorithm	Median Error $\times 1000$	% best results
FM2D	10.2	4%
FM3D	19.7	0%
ICP	15.03	12%
PCA	24	0%
FVR	8.03	76%
FFVR	16.47	0%
FVR-3D	19.4	8%



Figure 4.11: Four Sample Frames from the Park Plants and Table Data Set.

beat all other algorithms 76 % of the time, and had the lowest median error score. ICP performed next best, followed by FVR-3D. A hybrid FVR based method would have produced the best result 84 % of the time.

In the final experiment using the Asus Xtion Pro Live Active sensor camera, another large office space was filmed with y-axis rotation being the primary camera movement, some example frames are shown in Figure 4.12. This scene was named the Office Texture Rotation Data Set and statistics for the registration results are presented in Table 4.12. These results reveal

Table 4.12: Reconstruction Errors for the Office Texture Rotation Data Set

Algorithm	Median Error $\times 1000$	% best results
FM2D	4.36	26.92%
FM3D	7.15	0%
ICP	4.76	34.62%
PCA	6.55	0%
FVR	5.3	7.69%
FFVR	4.74	7.69%
FVR-3D	4.35	23.08%

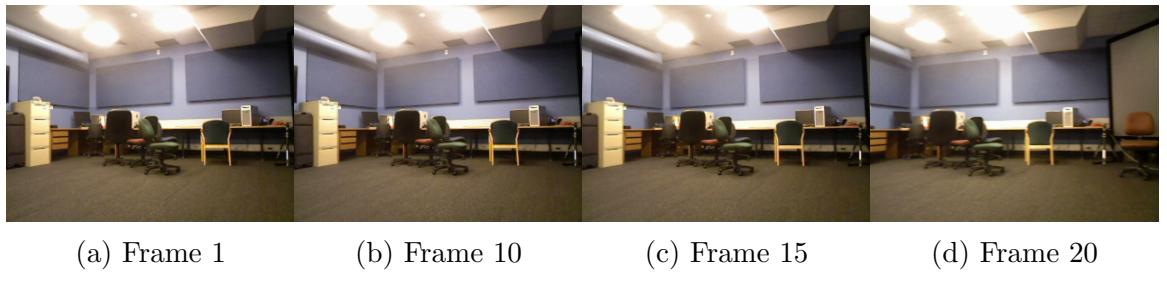


Figure 4.12: Four Sample Frames from the Office Texture Rotation Data Set.

the FVR-3D method achieved the lowest median error result but only the third highest percentage of best results score. A hybrid FVR based method would have achieved the best result around 38.46 % of the time. In terms of median error, FM2D was better than ICP but their positions were reversed in the percentage of best results metric.

4.4.5 Monocular Camera

During experiments, it was found that MVVR (FVR applied to monocular camera sensor data) had considerably less accuracy and robustness than the other FVR methods (FVR, FFVR and FVR-3D). Additionally, we found

that MVVR was less robust to rotational transforms. This is due to the fact that depth maps produced by monocular sensor methods, such as optical flow, do not produce sufficiently accurate and reliable depth maps. Results from stereo and active camera tests suggest that this is due to the reduction in accurate and precise depth data generation.

Nevertheless, to evaluate the performance of the MVVR method, quantitative experiments were performed on the Kitti Vision Benchmark Data Set. In these experiments, the MVVR method was compared with methods from the literature including FM2D, FM3D, ICP and PCA. Each depth map was projected into volume sizes of 256^3 for processing by the rest of the MVVR method (the FVR part). To generate the depth maps, a local 2D block matching method was used. Kernel sizes used in the correlation procedure were 3×3 in size with a search area size of 21×21 . The sizes of the kernel, search area and volume sizes were all chosen empirically.

Depth maps computed using the block matching method are only estimates of true scene depth. Therefore, resulting projections are not accurate. These projections, however, are still good enough to produce reconstructions. Registration of these inaccurate projections is difficult due to the amount of noise in the data. Figure 4.13 shows the noise contrast between depth maps

produced by a LIDAR and a monocular block-matching system.

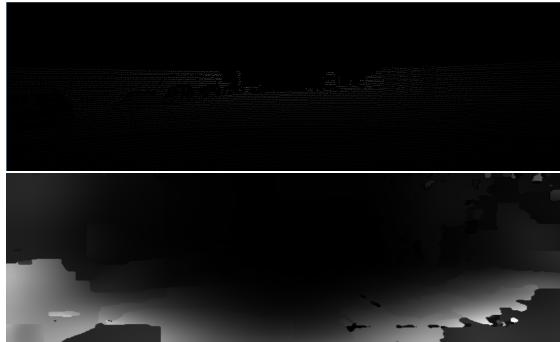


Figure 4.13: Left: Ground Truth Depth Map Computed Using a LIDAR System, Right: Depth Map Computed Using a Monocular Method

Results on the Kitti Benchmark 0001 Sync data set are shown in Table 4.13. In these results, ICP outperformed the other methods, with each algorithm having larger errors due to the inaccurate depth maps. Here, FVR achieved the best result 29.25 % of the time. Compared to results in Table 4.1, the MVVR method was less competitive than ICP. Due to the low-quality projections, FM2D and PCA failed were not able to register every frame. Most of the algorithms had incorrectly registered frames at some point.

These results, and those from the Stereo and Active camera experiments, indicate that, if the quality of the depth maps had been higher, the FVR based methods would have produced better results.

Table 4.13: Reconstruction Errors for the Kitti Data 0001 Sync Data Set Using Monocular (RGB) Input Only

Algorithm	Median Error $\times 1000$	% best results
FM2D	3742.4	2.83%
FM3D	918.05	14.15%
ICP	772.48	50.94%
PCA	2046.96	2.83%
MVVR	944.81	29.25%

4.5 Camera Tracking & Noise Robustness

Experiments measuring the error after moving the camera in specific increments using the FVR based methods and other methods from the literature are presented in this section. For these experiments, one camera frame of an indoor environment was captured using the Asus Xtion Pro Live active camera. The camera was then moved (translated) by different amounts, 5 cm, 10 cm and 15 cm. The 2D and 3D Feature matching RANSAC methods (FM2D, FM3D), ICP and PCA methods were all tested. The results for the FVR, FFVR and FVR-3D methods are also presented. FM3D, FVR and FVR-3D require the 3D frames be quantized into $256 \times 256 \times 256$ volumes.

Different levels of noise were added to both frames prior to 3D registration to measure each method's ability to register frames with large amounts of noise. The Signal to Noise Ratio (SNR) metric is used to describe the noise added to both captured frames, prior to any registration. This noise effects

any registration method's ability to accurately estimate the transformation separating two sets of data. In a given experiment, a noise range value of x means random noise was added in the range $[\frac{-x}{2}, \frac{x}{2}]$ to a signal within the range $[0, 1]$. Reconstruction error is measured in mean squared error (as presented in section 4.3).

Table 4.14 shows the results of these tests which, illustrate the FVR's robustness to noise whilst registering frames which are captured during different camera translation intervals. Each sub-table is labelled with a distance in centimetres in which frames were separated prior to registration. The first two columns represent the amount of noise added both in terms of noise-range and the subsequent Signal to Noise ratio computed from it. The remaining columns represent the registration error using the Mean Squared Error metric. Errors above 30 are capped, the final row of each table shows the average error for every noise level.

Results show that the basic FVR method is generally the most accurate in terms of registration across the range of camera translation magnitudes and noise levels. In comparison, the FFVR method reduces processing time at the expense of accuracy and robustness over a wide-baseline. Results generally show that FFVR is only capable of up to 5 cm of translation reg-

Table 4.14: Translation Tracking

5cm									
Noise	SNR	FM2D	FM3D	ICP	PCA	FVR	FFVR	FVR-3D	
0	∞	5.07	4.75	12.35	3.73	4.04	30	6.24	
0.1	20db	18.96	30	30	2.33	2.73	14.6	5.63	
0.25	12db	5.12	4.3	2.03	4.2	4.13	5.51	2.3	
0.5	6db	5.61	1.95	3.71	6.83	3.7	7.73	2.83	
0.75	2.5db	5.02	8.57	6.41	20.94	2.24	6.5	4.33	
Average	-	7.96	9.91	10.91	7.61	3.37	12.87	4.27	
10cm									
Noise	SNR	FM2D	FM3D	ICP	PCA	FVR	FFVR	FVR-3D	
0	∞	15.35	30	10.26	16.87	3.47	30	7.07	
0.1	20db	18.2	15.36	6.24	30	3.89	30	11.45	
0.25	12db	30	24.12	6.02	20.12	5.4	30	7.53	
0.5	6db	30	4.42	5	3.96	4.65	30	9.47	
0.7	2.5db	7.27	8.57	3.98	7.85	15.45	30	4.3	
Average	-	20.16	16.49	6.3	15.76	6.57	30	7.96	
15cm									
Noise	SNR	FM2D	FM3D	ICP	PCA	FVR	FFVR	FVR-3D	
0	∞	30	30	7.85	30	25.87	30	30	
0.1	20db	30	30	7.13	30	12.27	30	30	
0.25	12db	15.17	30	30	30	8.1	12.52	30	
0.5	6db	30	30	11.11	30	10.13	11.09	30	
0.7	2.5db	30	30	10.31	20.59	7.79	30	30	
Average	-	27.03	30	13.28	28.12	12.83	22.44	30	

istration. FVR-3D and the 2D Feature Matching method can register 5-10 cm with larger levels of noise. However, both methods fail to register the full 15cm of camera translation well. In both the 5cm and the 15cm experiments, the FVR method has the lowest average error, and the second lowest in the

10cm camera translation tests.

ICP performed next best, capable of performing well up to 15cm of translation, but failing to register twice during the tests. PCA could handle up to 10 cm of translation but often had larger registration errors. For the 15 cm translation tests, the FVR performed the best followed by ICP, which failed once but outperformed FVR within the two lowest noise brackets.

It can be shown that, despite being limited to a single axis of rotation, the FVR algorithm can consistently register camera movements up to 15 cm better than other algorithms from the literature. To put these camera translations into perspective at video frame rates, a displacement of 10 cm per frame equates to camera velocity of 3 m/s, this is twice the average person's walking speed making both the FVR method and FVR-3D suitable for many applications.

4.5.1 Camera Rotation Tracking

Table 4.15 shows the results for camera rotation experiments. These experiments were captured with the Asus Xtion Pro Live camera using the same scene as in section 4.5. 3D frames of these scenes are separated by 10, 20 and

30 degrees of camera rotation about the y-axis. These degrees were chosen because they are significantly large enough to be difficult for these algorithms to register against. Again, different levels of noise were added to each frame prior to registration. This experiment was designed to test the robustness of the FVR based methods in registering camera pose.

Table 4.15: Rotation Tracking

10deg									
Noise	SNR	FM2D	FM3D	ICP	PCA	FVR	FFVR	FVR-3D	
0	∞	0.17	9.77	0.19	0.3	0.18	0.35	4.05	
0.1	20db	0.21	0.41	0.17	0.67	0.15	0.21	10.54	
0.25	12	0.16	0.34	0.13	0.39	0.15	0.44	5.44	
Average	-	0.18	3.51	0.16	0.45	0.16	0.33	6.68	
20deg									
Noise	SNR	FM2D	FM3D	ICP	PCA	FVR	FFVR	FVR-3D	
0	∞	1.37	13.1	1.81	30	7.29	1.32	3.18	
0.1	20db	1.35	30	30	2.97	0.7	2.03	1.92	
0.25	12db	1.6	3.26	30	0.64	1.54	4.29	2.24	
Average	-	1.44	15.45	20.60	11.20	3.18	2.55	2.45	
30deg									
Noise	SNR	FM2D	FM3D	ICP	PCA	FVR	FFVR	FVR-3D	
0	∞	3.87	10.15	3.16	9.97	2.11	5.68	3.63	
0.1	20db	3.77	30	30	30	3.84	2.82	1.39	
0.25	12db	3.68	2.76	5.12	5.78	3.3	7.15	6.38	
Average	-	3.77	14.30	12.76	15.25	3.08	5.22	3.8	

In the first sub-table, registration errors for 10 degrees of rotation are pre-

sented. Results show that the FVR method outperforms the other methods. Here, ICP performs next best followed by the 2D Feature Matching method. This was expected as the FVR method was designed to be both robust to noise and to be able to handle larger rotations. In the 20 degree and 30 degree tests, the FVR method performs better than the other algorithms in terms of having lower registration error. FFVR and FVR-3D also worked well at different noise levels up to 30 degrees of rotation. FVR-3D was found to be as robust as ICP at registering rotation but not as accurate as the 2D Feature Matching method. In the 20 degree rotation experiments, the 2D feature matching method was shown to perform best in terms of average error, but in the 30 degree experiments, FVR was the top performer. Again, when processing low quality depth data such as those generated by an active camera, the FVR based methods have reduced performance compared to other algorithms compared to processing higher quality depth data.

It should be noted that twelve degrees of camera rotation per frame is almost a full rotation per second in video rates. This is so fast that most cameras would acquire too much motion blur for registration to be possible. Therefore, this test indicates the robustness of the FVR method in comparison with the other algorithms within the context of camera pose estimation.

4.5.2 Reconstructed Scenes

To assess qualitative reconstructions, three scenes were captured and registered using the FVR method. These scenes are the apartment scene, office scene and the garden scene. All three scenes were captured using the Asus Xtion Pro live active camera at 30 frames per second. For each scene, only one in every 30 frames was registered, constituting real time FVR performance. The apartment scene was generated by moving and rotating the camera around an apartment living room before moving towards the kitchen area. This scene contains an abundance of features and would be considered a basic test for most 3D reconstruction algorithms. The second scene is the office scene. This scene was generated by rotating the camera whilst zooming in and out on different objects within the room. Again, this scene was reconstructed by registering every 30th frame. Despite both scenes being trivial to reconstruct, most algorithms (especially feature matching based methods) would find registering large rotations (such as those present in this data) difficult. The garden scene is a difficult scene to reconstruct regardless of the reconstruction algorithm or technique used for registration. This scene was captured by rotating the camera around a garden outside the university. The scene contains many textures which are similar at the local level but are located in totally different locations. Therefore, the scene is difficult to reconstruct using local and feature based methods such as Feature Matching

+ RANSAC and ICP.

Reconstructions for the two indoor environments (apartment and office) as well as one outdoor environment (garden) are shown in Figures 4.14a, 4.14b and 4.14c, respectively.

The apartment reconstruction was recorded by moving the Asus Xtion Pro Live active camera through a room and rotating the camera. Each frame was registered using the FVR algorithm. Some of the frames in the apartment scene contain walls which have few features. Between frames, walls also had colour contrast shifts. These shifts are due to the Asus camera's automatic contrast feature which adjusts contrast based on colour histograms. Despite these setbacks, accurate 3D reconstruction was achieved by the FVR method as illustrated in Figure 4.14a.

The office reconstruction was also generated by rotating the Asus Xtion Pro Live active camera about the y-axis while moving the camera around the room. During rotation, the camera was focused on both foreground and background objects. Here, the entire video sequence was accurately registered. Despite the foreground and background focus, the global reconstruction is accurate. This scene, as in the apartment scene has usable texture which should

not cause large amounts of texture confusion. These qualitative experiments show that despite being a closed form solution, the FVR has reconstruction accuracy comparable to existing feature based SLAM methods.

Typical feature based methods work well with indoor environments where local features are readily distinguishable and easy to match. They do not tend to work as well in complex outdoor scenes where feature confusion is likely. To assess performance in such outdoor scenes, a garden scene containing bushes, plants and a ground covering of bark and rocks was captured for testing. Again, this scene was captured using the Asus Xtion Pro Live active camera moving around the outdoor garden. The proposed FVR method produced a good quality reconstruction of this garden scene, as shown in Figure 4.14c. This shows that reconstruction approaches which integrate or make use of the FVR registration method may have an advantage in performing 3D reconstructions in these types of scenes, which commonly disturb many existing feature matching methods, as expressed in the literature.

Reconstructions of the Parks, Plants and Table data set using the FVR, ICP and FM2D methods are shown in Figure 4.15. It can be seen that the FVR method could reconstruct the scene accurately. FM2D and ICP failed to generate a suitable reconstruction for this data set. This is likely due to the level of texture confusion present in the scene. The FVR based methods

are more robust to texture confusion, since they make use of all of the data independent of salient points within the scene.

4.5.3 Comparison

Here we present a summary table (Table 4.16) to assess the various abilities of each 3D reconstruction algorithm in terms of translation, rotation, scale, non-static object motion robustness and ability to handle various sensory types. In terms of translation, each of the presented algorithms can register with respect to translation. In experiments, it was shown that the FVR algorithm outperforms others when this type of camera movement is present, especially at wider baselines of 10 to 15 cm. In registering rotation, the FVR and FFVR methods are only capable of a single axis of rotation. FVR-3D, however, can register against all 3 axes of rotation. Results have shown that in the presence of rotation, the FVR-3D performs at or above the level of the more accurate algorithms from the literature, ICP and FM2D. It was found that MVVR was not able to handle rotation well, due to the high levels of noise found in the depth maps produced by monocular based techniques.

Only the FVR and FVR-3D methods were capable of registering against scale. From the literature, ICP and PCA are not capable of handling such transformations without some modification. Results have shown that these

algorithms are capable of handling non-static object motion well. Results on the Kitti Vision benchmark show that FVR-3D outperforms the state-of-the-art, even on data sets which have non-static moving agents. Results also showed that the FVR was the most accurate in the registration of wide baselines, that is translations above 5 cm and rotations above 5 degrees. The FVR algorithm outperformed the others, including the top methods used in the literature. Of these methods, MVVR was shown to be able to register monocular sensor data against translation when noise levels were low. The closer the depth data computed via monocular methods approaches perfectly accurate depth images, the closer the MVVR algorithm approaches FVR performance. In terms of both stereo and active sensor input, FVR, FFVR and FVR-3D are all capable of handling this type of input data well.

Given each algorithm, the number of seconds required to perform frame registration is recorded in table 4.17. Results show that the PCA algorithm is fastest. This was expected, as the PCA method had the least amount of processing operations per feature. Next, FM2D and ICP took around 0.3 seconds at best. During testing, SURF features were sorted by their strength of match so FM2D’s time per registration varied little. On the contrary, ICP was highly dependent on how many points were used. ICP varied from 0.3 seconds to 11 seconds. FFVR was next fastest, and took

0.4 seconds to register frames. The FVR method was slower than this, at 1 second per registration. FVR-3D was slowest, and took around 10 seconds per registration. Unlike ICP however, FVR-3D did not vary based on the number of points used in computing the camera transform. The GPU used in these experiments has just 384 cores. The time it takes to compute the FVR, FFVR and FVR-3D methods is reduced inversely relative to the number of cores available for computation. We suspect as parallel processing technology improves, methods such as FVR, FFVR and FVR-3D will become faster.

Table 4.16: FVR Comparison Table

<i>Algorithm</i>	<i>translation</i>	<i>rotation</i>	<i>scale</i>	<i>object motion</i>	<i>wide baselines</i>	<i>monocular</i>	<i>stereo</i>	<i>RGB-D</i>
FVR	yes	1 axis only	yes	yes	yes	no	yes	yes
FFVR	yes	1 axis only	no	yes	no	no	yes	yes
FVR-3D	yes	yes	yes	yes	no	no	yes	yes
MMVR	yes	no	no	no	no	yes	no	no

In summary, the stereo camera and active camera results, show that FVR-3D method outperforms all the other algorithms from the literature. For most of these data sets the FVR algorithm was a strong second place contender. When registering wider baselines, the FVR algorithm was the top performer while FVR-3D fell behind.

4.6 Plane-Tree Experiments

In this section, we present experiments in which the Plane-Tree 3D model and 3D reconstruction data compression system are examined. The Plane-Tree

Algorithm	Seconds per Frame Registration
PCA	0.014
FM2D	0.3
ICP	0.3 - 11
FFVR	0.4
FVR	1
FVR-3D	10
FM3D	15

Table 4.17: Seconds Per Frame Registration

is first compared to its similar neighbour method, the octree. It is compared with the octree using the mean error metric discussed in section 4.3. The two algorithms are compared when compressing three commonly available object models, Bunny, Fandisk and Horse. These data are shown in Figure 4.16.

Next, the Plane-Tree is compared with the current state-of-the-art methods from 3D object compression research. These experiments are presented in section 4.8. These experiments reveal that the Plane-Tree not only, outperforms the original octree method, but also some of the state-of-the-art transform methods at low-bit-rates. Several state-of-the-art algorithms are compared including some transform based methods [75, 76] and some compression systems which were designed to compress at low bit-rates [72, 186].

To compare the Plane-Tree with each of these algorithms quantitatively,

each algorithm is set-up to compress a model at different levels of compression vs quality. An algorithm's rate-distortion (RD) is plotted and compared to each of the other algorithms. Since lossy algorithms are compared, this indicates the amount of distortion (in the decoded model compared with the original) given a particular bit-rate. This measurement represents the number of bits each algorithm requires to store a particular model at a particular level of quality. The fewer bits required to represent a model of a given accuracy, the better the compression system performs. The mean error and root mean squared error discussed in section 4.3 are used to measure the quality of the compressed models whilst the bit-rate in bits per vertex metric is used to measure the number of bits required to store the compressed model. Both the mean error and the root mean squared error are scaled by the main diagonal of the input model to make the results invariant to model size. Qualitative results comparing the Plane-Tree with other state-of-the-art methods are also presented.

4.7 Plane-Tree vs. Octree

Figure 4.17 shows rate-distortion graphs comparing the Plane-Tree with the octree compression method. In these rate-distortion graphs we use the mean error between the decoded and input model as the distortion metric. Results

show that for the Bunny and Fandisk experiments (Figures 4.17a and 4.17b) the Plane-Tree has better quality for a given bitrate compared to the octree. It is also evident that the octree is unable to reach the level of quality the Plane-Tree reaches. In both cases, due to the Plane-Tree’s ability to prevent further tree decomposition via its more accurate representation method, the Plane-Tree has a much lower error rate for a given bit-rate compared with the octree method.

In the Horse model graph in Figure 4.17c there is some overlap in the model quality (error rates). This overlap ranges from around 0.0025 to 0.004 in which for these levels of quality, the octree requires around 8 times more storage space compared with the Plane-Tree. These qualitative results show how much of an improvement the Plane-Tree model codec is over the octree method.

4.8 Plane-Tree vs. Existing Techniques

In Figures 4.18a, 4.18b and 4.18c we compare the Plane-Tree with the state-of-the-art transform methods by Bayazit et al. [75], Khodakovskiy et al. [76] and the SOT [186].

Experiment results comparing the Plane-Tree compression system proposed in this work with the transform-based method by Khodakovsky et al. and SOT are shown in figure 4.18a. This Rate-Distortion graph shows that the Plane-Tree has similar performance with the method by Khodakovsky et al. At low bit-rates the Plane-Tree is highly competitive the with performance matching Khodakovsky's. The SOT method outperforms both of these data structures at lower bit-rates.

Figure 4.18b shows comparisons between the Plane-Tree as well as methods by Khodakovsky et al., Bayazit et al. and the SOT. Here, both the Plane-Tree and the method by Khodakovsky et al. perform similarly as in the bunny model experiment (Figure 4.18a). The Plane-Tree stays competitive with that state-of-the-art method, and improves upon the compression performance in comparison to the spectral compression method by Bayazit et al. and the SOT. It can be seen that at low to mid bit-rates, the Plane-Tree method compressed the Fandisk model at a higher level of quality for a given bit-rate.

The Plane-Tree was also compared with Bayazit et al. and the SOT using another model, the horse model. Figure 4.18c shows the result of this experiment. The Plane-Tree method outperforms the transform based method

of Bayazit et al. and the SOT whilst decreasing coding complexity compared to the complicated transform method. Overall, in this experiment, the proposed Plane-Tree method remains competitive at higher bitrates, and it outperforms the method by Bayazit et al. at lower bitrates.

Finally, the Plane-Tree is compared with the state-of-the-art low-bitrate compression system FOLProM presented by Peng et al. [72] (as well as the SOT). Unlike the other experiments, the mean-error metric is used in this comparison, as the results presented by Peng et al. used this metric. It can be seen that at low bit-rates (below 2 bits per vertex), the Plane-Tree method outperforms this state-of-the-art low-bitrate compression system. Moreover, the Plane-Tree easily outperforms the SOT.

In conclusion, these experiments show that the Plane-Tree is extremely competitive with the state-of-the-art compression systems at high bit-rates. At lower bit-rates it improves upon the results by these methods. In three out of the four experiments, the Plane-Tree outperformed the SOT. The Plane-Tree is also powerful in that it may be used to compress 3D volumetric data as well as point-cloud and mesh data. This makes it an interesting candidate for 3D reconstruction compression no matter the format output by a given 3D reconstruction method.

4.9 Plane-Tree: Qualitative Results

Figure 4.19 shows qualitative results for the Plane-Tree. The third row shows the bunny, rabbit and horse models along with the number of bytes and bpv required to store them uncompressed. The first row shows each model compressed by the Plane-Tree with a threshold of 8.0. Despite these being crude approximations of the originals, the models are still distinguishable with around $1000 \times$ less storage space required.

In the second row, each model was compressed with the Plane-Tree at a threshold of 1.0. In these experiments, there is little detail missing compared with the physical model. When looking at the bunny's legs, the same ripples are present. In the rabbit model, the outlines inside the ears and on the eyes are still present. On the horse model, the creases on the body and shoulder of the horse are still present. Here, the bunny is compressed to around $70 \times$ less storage space, the rabbit at around $265 \times$ less and the horse around 90 \times less storage space.

Figure 4.20 shows the Bunny model compressed using the Plane-Tree method and two state-of-the-art methods, the valence method [74] and the

spectral method [73]. The original model in Figure 4.20a requires 2,258,902 bytes for storage. The valence method and the spectral method require 17,852 bytes to store the model and the compression effects are shown in Figures 4.20c and 4.20d. Noticeable artefacts not present in the original model are produced by both codecs. The model coded by the valence method fairs worse than the one coded by the spectral method. The model compressed by the Plane-Tree requires just over half the amount of bytes than the other codecs. The Plane-Tree coded model is smoother than the other models (despite flat shading) too. This is likely due to the way the Plane-Tree compression system approximates the original model using planes. This process doesn't introduce these artefacts, allowing the Plane-Tree to represent the original model more accurately.

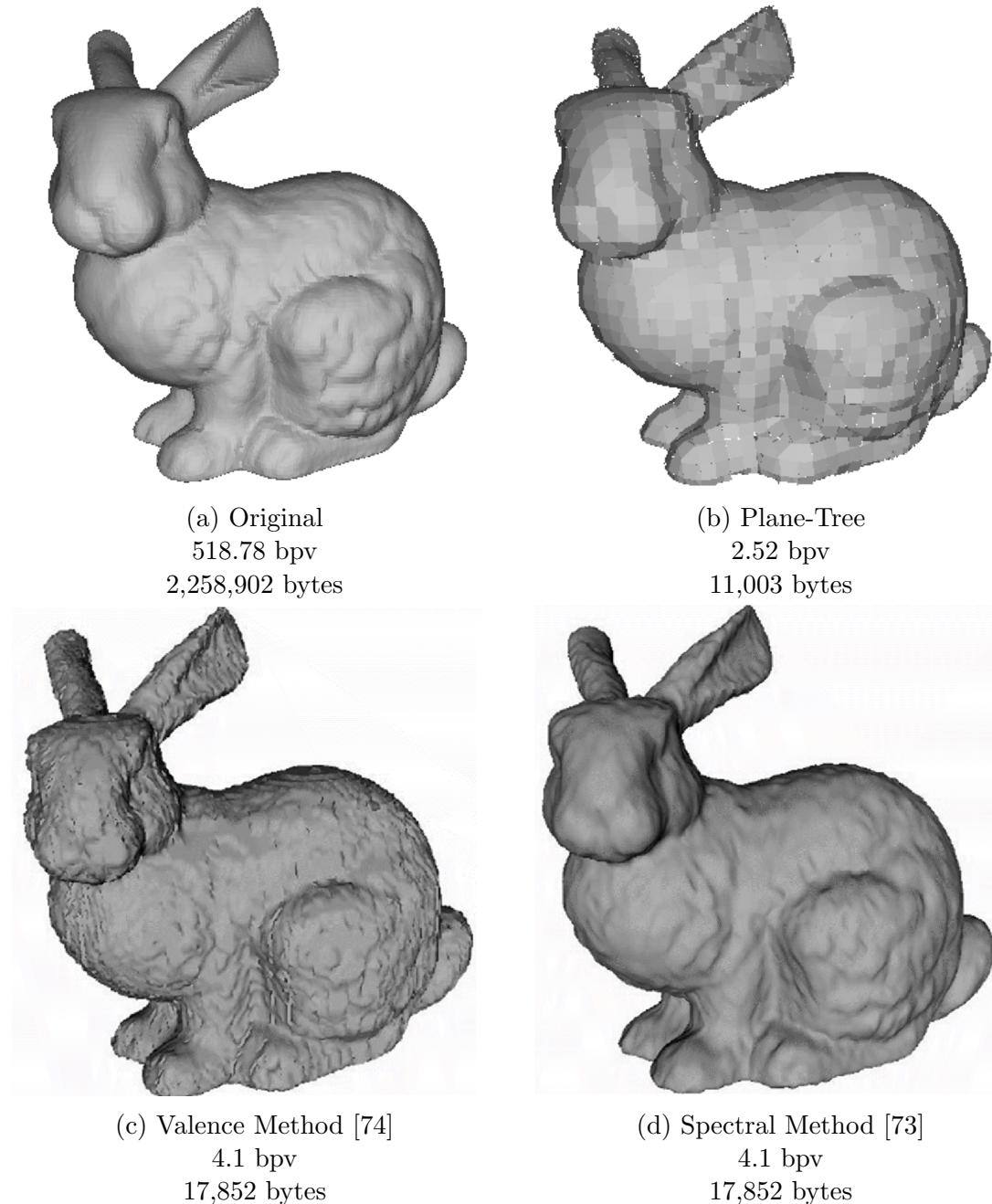


Figure 4.20: The Bunny Model Compressed Using the Plane-Tree, Valence and Spectral Methods.

Figure 4.21 shows the bunny model coded using the Plane-Tree and the codec by Khodakovsky et al. Both the Plane-Tree and method by Khodakovsky et al. require just 1349 bytes of storage to represent these low bit-rate models. Both methods however, use different strategies and introduce different styles of noise into the compressed model. The method by Khodakovsky et al. shrinks the models and warps the shape, simplifying it. The Plane-tree approximates the shape and positions of the vertices more closely but does not produce smooth surfaces. We conclude that both methods may be beneficial in different situations. The method by Khodakovsky et al. may be useful in situations where visual appeal are important. The Plane-Tree method may be more useful in applications where model shape and size are more important.

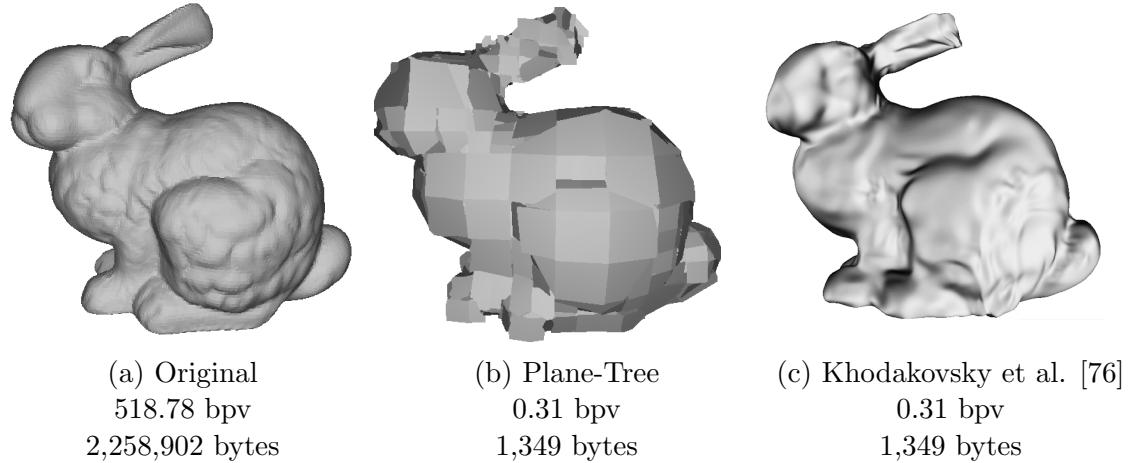


Figure 4.21: The Bunny Model Compressed Using the Plane-Tree and the Wavelet Compression System Khodakovsky et al.

4.10 Plane-Tree: Reconstruction Compression

In this section, experiments comparing the Plane-Tree with the octree in terms of 3D frame or 3D occupancy reconstruction grid compression. Essentially this experiments compares the performance of the Plane-Tree and octree in 3D occupancy grid compression.

The octree is used for comparison since it is the closest relative of the Plane-Tree and existing methods which use the octree are presently used in 3D reconstruction research.

In this experiment, Rate-Distortion is measured in PSNR (Peak Signal to

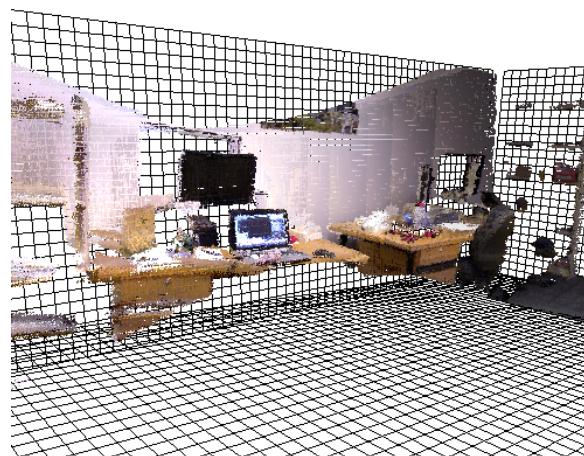
Noise Ratio) which is a measurement of the amount of accuracy between the original model and the compressed version for a given algorithm and level of compression. The larger the PSNR, the higher the quality in which the compression system produces for a given bitrate.

A Rate-Distortion graph comparing the Plane-Tree and octree is presented in Figure 4.22. Results show that for low and high bit-rates the Plane-Tree outperforms the octree method. As higher quality is required, the Plane-Tree’s quality rises exponentially. The Plane-Tree algorithm proposed in this work is shown to be dominant compared to the Octree. At any bit-rate, the Plane-Tree has a much larger Peak-Signal-To-Noise ratio.

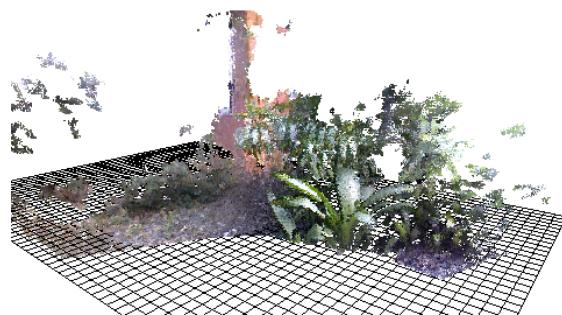
In conclusion, as in results comparing the Plane-Tree with state-of-the-art mesh compression algorithms, the Plane-Tree is shown to perform well and even outperform SOT and the generic octree data structures when compressing 3D data used in reconstructions.



(a) Apartment



(b) Office



(c) Garden

Figure 4.14: Reconstructed Scenes.



(a) FVR



(b) ICP



(c) FM2D

Figure 4.15: Reconstructions of the Parks, Plants and Table Scene Using FVR, ICP and FM2D.

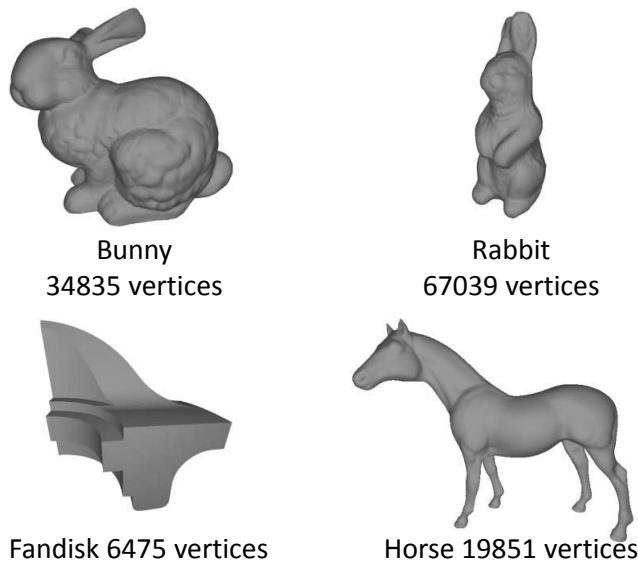


Figure 4.16: Models used to assess the Plane-Tree compression algorithm.

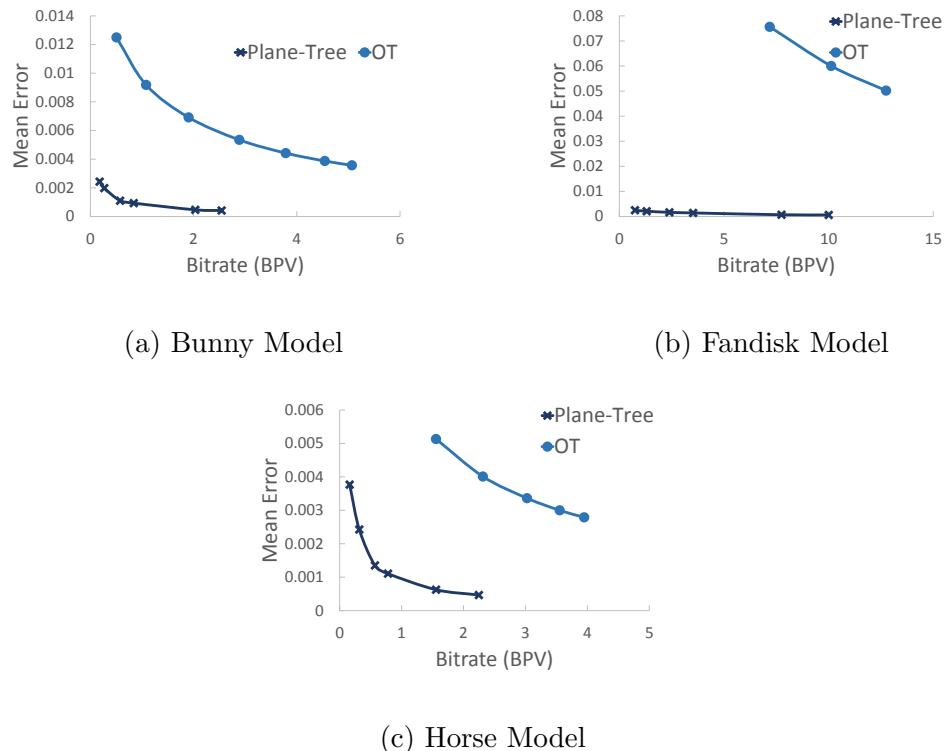


Figure 4.17: Rate-distortion graphs comparing the Plane-Tree with the Octree.

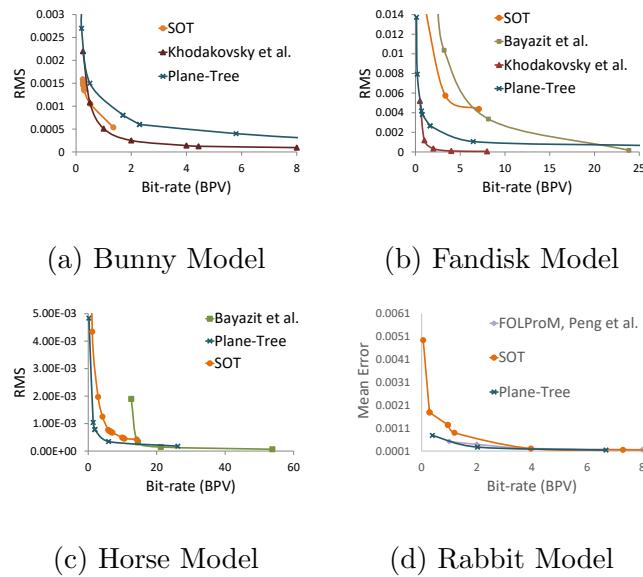


Figure 4.18: Rate-Distortion graphs comparing the Plane-Tree to different state-of-the-art codecs.

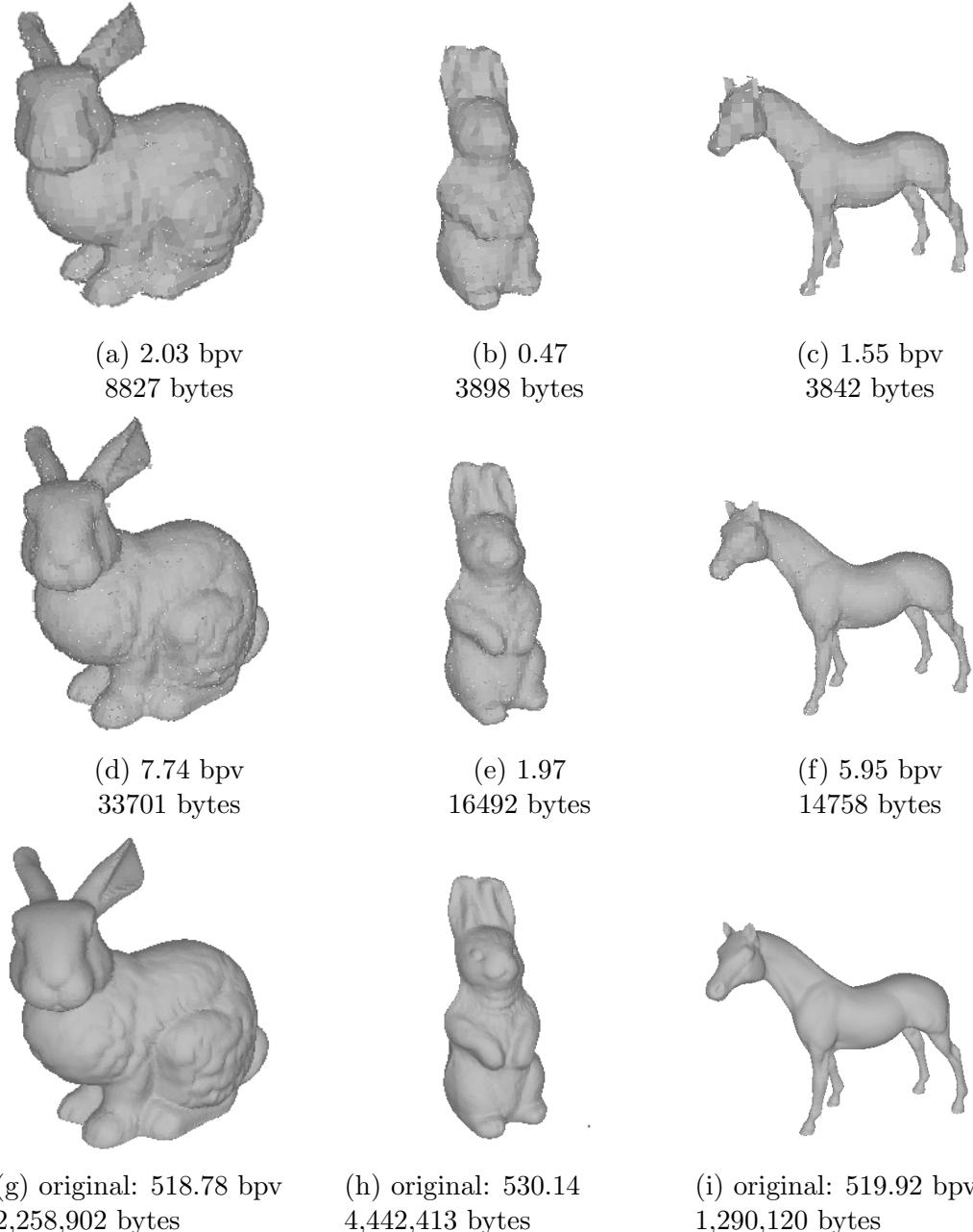


Figure 4.19: Models coded with our coder at thresholds of 8.0 and 1.0 with a maximum tree depth of 6 along with the original models.

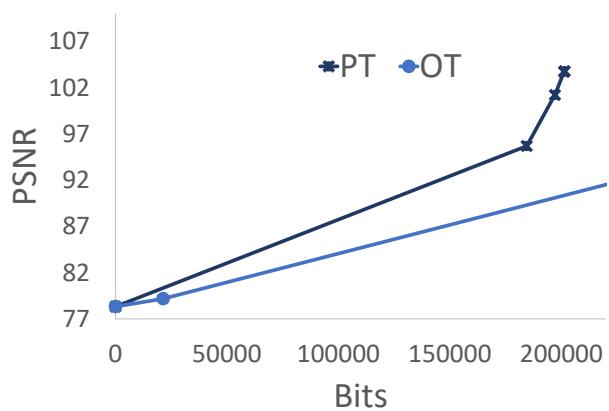


Figure 4.22: PSNR vs Bit-rate comparing the OT and Plane-Tree compression methods.

5

Conclusion

In this thesis, novel techniques were proposed which improve the accuracy, noise robustness, speed and storage, quantitative quality and perceptual quality of 3D reconstructions. The FVR and FVR-3D methods are accurate and robust to noise. Both quantitative and qualitative results show these methods outperform existing techniques from the literature 4. The FFVR (Fast FVR) method improves upon the computational performance of the FVR. Although the FFVR achieves great performance gains at the cost of accuracy and robustness, it was still shown to be competitive. MVVR was proposed, allowing the FVR algorithm to work with monocular data to mixed results.

To reduce storage requirements for 3D reconstructed models during processing and in general, a novel 3D data compression system was presented names, Plane-Tree. The Plane-Tree data representation is based on octree decomposition. The Plane-Tree is able to compress models at low bit-rates more effectively than several state-of-the-art algorithms. Further details on the findings made during this research are presented in forthcoming sections.

5.1 Fourier Volume Registration

FVR and FVR-3D were developed to be accurate and noise robust closed form solutions to the 3D frame registration problem. Quantitative and qualitative results show the FVR-3D and FVR methods are capable of producing accurate 3D reconstructions given stereo or active sensor data. In a majority of experiments on data captured using active and stereo sensors, FVR-3D achieved the best quantitative performance. When rectifying larger camera translations and rotations, the FVR method often performed best. Therefore, when camera displacement is large, FVR is the best choice for 3D reconstruction.

The FFVR performs similarly to the FVR but trades accuracy for speed. A theoretical speed analysis shows the FFVR method approaches the speed

of PCA and is three times faster than next quickest algorithm, FVR. Despite having reduced accuracy compared with the FVR, FFVR is shown to have outperformed PCA in terms of average registration error in three out of five stereo experiments and seven out of seven active sensor experiments. MVVR is used to register 3D projections computed from monocular data. Results suggest that the MVVR has reduced performance compared with the FVR method because of the low quality depth data produced by monocular depth extraction techniques. If computed depth map accuracy approaches the quality of laser measured depth, MVVR's performance approaches FVR's.

5.2 Plane-Tree 3D Data Compression

The Plane-Tree 3D data compression scheme, based on the octree data structure, reduces the storage and transmission requirements of 3D reconstructed models and general 3D data. Quantitative experiments show the Plane-Tree improves upon the octree data representation technique in terms of compression. It also outperforms several state-of-the-art 3D model compression algorithms from the literature. When compressing 3D reconstructed data, Plane-Tree has better compression ability compared with the octree data structure.

5.3 Research Aims Revisited

The primary aim of this research is to improve the accuracy, noise robustness, speed and storage, quantitative quality and perceptual quality of 3D models generated from image data. The FVR, FFVR, FVR-3D and MVVR methods, as well as the Plane-Tree 3D data representation method, were proposed to achieve this aim. The research questions were:

1. “Can Fourier based registration techniques improve accuracy and noise robustness in 3D reconstruction applications?”

and

2. “Can hierarchical techniques improve compression, storage and processing of 3D reconstruction data?”

Results in Chapter 4 show that FVR, FFVR, MVVR and FVR-3D are accurate, robust to noise and produce qualitatively pleasant 3D reconstructions. Quantitative experiments show that the FVR method improves upon existing solutions in terms of accuracy and noise robustness on especially when performing wide-baseline registration. The FVR-3D algorithm out-

performs ICP, FM2D, FM3D and PCA when registering frames from active sensors and stereo camera pairs. Therefore, the answer to Question 1. is yes, Fourier based registration techniques can and do improve accuracy and noise robustness in 3D reconstruction applications.

Results show, hierarchical method Plane-Tree improves upon existing state-of-the-art methods compressing 3D models and outperforms the octree in compressing 3D reconstructions. Therefore, the answer to the second research question is yes, hierarchical techniques improve compression, storage and processing of 3D models, including reconstruction data.

5.4 Future Work

There are still many topics in 3D Fourier reconstruction and hierarchical data storage, processing and transmission to be researched. Further development of the FVR technique is required as presently, the system does not have recovery functionality for when individual frame registration fails. One possible topic is the effects of filtering registration data using a Kalman filter to improve reconstruction performance. There are also many hierarchical based representation variants to investigate. Investigating hierarchically compressing frames prior to model integration, would be another interesting topic.

Reference List

- [1] L. Lincoln and R. Gonzalez, “Interpolating leaf quad tree image compression,” in *2013 Seventh International Conference on Signal Processing and Communication Systems (ICSPCS’2013)*, Gold Coast, Australia, Dec. 2013. 4, 162, 165
- [2] ——, “Fourier volume registration based dense 3d mapping,” *EAI Endorsed Transactions on Context-aware Systems and Applications*, vol. 16, no. 10, 9 2016. 5
- [3] ——, “Dense 3d mapping using volume registration,” in *Nature of Computation and Communication - Second International Conference, ICTCC 2016, Rach Gia, Vietnam, March 17-18, 2016, Revised Selected Papers*, ser. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, P. C. Vinh and L. Barolli, Eds., vol. 168. Springer, 2016, pp. 22–32. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-46909-6_3 5
- [4] ——, “Dense 3d mapping using volume reigstration from monocular view,” in *2016 10th International Conference on Complex, Intelligent, and Software Intensive Systems (CISIS)*, July 2016, pp. 137–141. 6

- [5] ——, “Plane-tree low-bitrate mesh compression,” in *2015 International Conference on Digital Image Computing: Techniques and Applications, DICTA 2015, Adelaide, Australia, November 23-25, 2015*. IEEE, 2015, pp. 1–5. [Online]. Available: <http://dx.doi.org/10.1109/DICTA.2015.7371295> 6
- [6] C. Harris and M. Stephens, “A combined corner and edge detector.” in *Alvey vision conference*, vol. 15. Manchester, UK, 1988, p. 50. 9
- [7] S. M. Smith and J. M. Brady, “Susana new approach to low level image processing,” *International journal of computer vision*, vol. 23, no. 1, pp. 45–78, 1997. 10
- [8] S. Smith, “A new class of corner finder,” in *BMVC92*. Springer, 1992, pp. 139–148. 10
- [9] K. N. Walker, T. F. Cootes, and C. J. Taylor, “Locating salient object features.” in *BMVC*, vol. 98. Citeseer, 1998, pp. 557–566. 10
- [10] M. Trajković and M. Hedley, “Fast corner detection,” *Image and vision computing*, vol. 16, no. 2, pp. 75–87, 1998. 10
- [11] D. G. Lowe, “Distinctive image features from scale-invariant keypoints,” *International journal of computer vision*, vol. 60, no. 2, pp. 91–110, 2004. 10, 52
- [12] ——, “Object recognition from local scale-invariant features,” in *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, vol. 2. Ieee, 1999, pp. 1150–1157. 10
- [13] W. Changchang, “Siftgpu: a gpu implementation of scale invariant feature transform (sift),” 2007. 11, 52

- [14] T. Tuytelaars and L. J. Van Gool, “Wide baseline stereo matching based on local, affinely invariant regions.” in *BMVC*, vol. 412, 2000. 11
- [15] Y. Y. Boykov and M.-P. Jolly, “Interactive graph cuts for optimal boundary & region segmentation of objects in nd images,” in *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, vol. 1. IEEE, 2001, pp. 105–112. 11
- [16] L. Itti and C. Koch, “Computational modelling of visual attention,” *Nature reviews neuroscience*, vol. 2, no. 3, pp. 194–203, 2001. 12
- [17] F. Schaffalitzky and A. Zisserman, “Viewpoint invariant texture matching and wide baseline stereo,” in *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, vol. 2. IEEE, 2001, pp. 636–643. 12
- [18] K. Mikolajczyk and C. Schmid, “Indexing based on scale invariant interest points,” in *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, vol. 1. IEEE, 2001, pp. 525–531. 12
- [19] C. Carson, S. Belongie, H. Greenspan, and J. Malik, “Blobworld: Image segmentation using expectation-maximization and its application to image querying,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 8, pp. 1026–1038, 2002. 12
- [20] N. Sebe, Q. Tian, E. Loupias, M. S. Lew, and T. S. Huang, “Evaluation of salient point techniques,” *Image and Vision Computing*, vol. 21, no. 13, pp. 1087–1095, 2003. 12

- [21] T. Kadir, A. Zisserman, and M. Brady, “An affine invariant salient region detector,” in *Computer Vision-ECCV 2004*. Springer, 2004, pp. 228–241. 13
- [22] P. Carbonetto, N. de Freitas, and K. Barnard, “A statistical model for general contextual object recognition,” in *Computer Vision-ECCV 2004*. Springer, 2004, pp. 350–362. 13
- [23] J. Matas, O. Chum, M. Urban, and T. Pajdla, “Robust wide-baseline stereo from maximally stable extremal regions,” *Image and vision computing*, vol. 22, no. 10, pp. 761–767, 2004. 13
- [24] K. Mikolajczyk and C. Schmid, “A performance evaluation of local descriptors,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 27, no. 10, pp. 1615–1630, 2005. 13
- [25] E. Rosten and T. Drummond, “Machine learning for high-speed corner detection,” in *Computer Vision-ECCV 2006*. Springer, 2006, pp. 430–443. 13, 54
- [26] ———, “Fusing points and lines for high performance tracking,” in *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, vol. 2. IEEE, 2005, pp. 1508–1515. 13, 54
- [27] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “Orb: an efficient alternative to sift or surf,” in *Computer Vision (ICCV), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2564–2571. 14, 52
- [28] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, “Brief: Binary robust independent elementary features,” in *European conference on computer vision*. Springer, 2010, pp. 778–792. 14

- [29] H. Bay, T. Tuytelaars, and L. Van Gool, “Surf: Speeded up robust features,” in *Computer Vision–ECCV 2006*. Springer, 2006, pp. 404–417. 14, 52
- [30] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, “Speeded-up robust features (surf),” *Computer vision and image understanding*, vol. 110, no. 3, pp. 346–359, 2008. 14, 52
- [31] V. Lepetit and P. Fua, “Keypoint recognition using randomized trees,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 28, no. 9, pp. 1465–1479, 2006. 14
- [32] M. W. J. Cabani C, “Implementation of an affine-covariant feature detector in field-programmable gate arrays,” *Proceedings of the International Conference on Computer Vision Systems*, 2007. 15
- [33] T. Tuytelaars and K. Mikolajczyk, “Local invariant feature detectors: a survey,” *Foundations and Trends® in Computer Graphics and Vision*, vol. 3, no. 3, pp. 177–280, 2008. 15
- [34] S. W. Smith *et al.*, *The scientist and engineer’s guide to digital signal processing*. California Technical Pub. San Diego, 1997. 16
- [35] Z. Zhang, “Microsoft kinect sensor and its effect,” *IEEE multimedia*, vol. 19, no. 2, pp. 4–10, 2012. 22, 58
- [36] G. Klein and D. Murray, “Parallel tracking and mapping for small ar workspaces,” in *Mixed and Augmented Reality, 2007. ISMAR 2007. 6th IEEE and ACM International Symposium on*. IEEE, 2007, pp. 225–234. 22, 41, 42, 72

- [37] R. A. Newcombe and A. J. Davison, “Live dense reconstruction with a single moving camera,” in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 1498–1505. 22
- [38] J. Stühmer, S. Gumhold, and D. Cremers, “Real-time dense geometry from a handheld camera,” in *Joint Pattern Recognition Symposium*. Springer, 2010, pp. 11–20. 22
- [39] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, “Rgb-d mapping: Using depth cameras for dense 3d modeling of indoor environments,” in *In the 12th International Symposium on Experimental Robotics (ISER)*. Citeseer, 2010. 22, 57
- [40] D. Scharstein and R. Szeliski, “A taxonomy and evaluation of dense two-frame stereo correspondence algorithms,” *International journal of computer vision*, vol. 47, no. 1-3, pp. 7–42, 2002. 23
- [41] J. Sun, Y. Li, S. B. Kang, and H.-Y. Shum, “Symmetric stereo matching for occlusion handling,” in *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, vol. 2. IEEE, 2005, pp. 399–406. 23
- [42] A. Klaus, M. Sormann, and K. Karner, “Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure,” in *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, vol. 3. IEEE, 2006, pp. 15–18. 24
- [43] H. Hirschmuller, “Accurate and efficient stereo processing by semi-global matching and mutual information,” in *2005 IEEE Computer*

Society Conference on Computer Vision and Pattern Recognition (CVPR'05), vol. 2. IEEE, 2005, pp. 807–814. 24

- [44] K.-J. Yoon and I. S. Kweon, “Adaptive support-weight approach for correspondence search,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 4, pp. 650–656, 2006. 24
- [45] A. Darabiha, W. J. MacLean, and J. Rose, “Reconfigurable hardware implementation of a phase-correlation stereoalgorithm,” *Machine Vision and Applications*, vol. 17, no. 2, pp. 116–132, 2006. 24
- [46] Q. Yang, R. Yang, J. Davis, and D. Nistér, “Spatial-depth super resolution for range images,” in *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on.* IEEE, 2007, pp. 1–8. 24
- [47] M. Sarkis, N. Dörfler, and K. Diepold, “Fast adaptive graph-cuts based stereo matching,” in *Advanced Concepts for Intelligent Vision Systems.* Springer, 2007, pp. 818–827. 25
- [48] Z.-F. Wang and Z.-G. Zheng, “A region based stereo matching algorithm using cooperative optimization,” in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on.* IEEE, 2008, pp. 1–8. 25, 26
- [49] Q. Yang, C. Engels, and A. Akbarzadeh, “Near real-time stereo for weakly-textured scenes.” in *BMVC*, 2008, pp. 1–10. 25
- [50] I. Ernst and H. Hirschmüller, “Mutual information based semi-global stereo matching on the gpu,” in *Advances in Visual Computing.* Springer, 2008, pp. 228–239. 26

- [51] M. Bleyer, M. Gelautz, C. Rother, and C. Rhemann, “A stereo approach that handles the matting problem via image warping,” in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on.* IEEE, 2009, pp. 501–508. 26
- [52] M. Bleyer and M. Gelautz, “Temporally consistent disparity maps from uncalibrated stereo videos,” in *Image and Signal Processing and Analysis, 2009. ISPA 2009. Proceedings of 6th International Symposium on.* IEEE, 2009, pp. 383–387. 26
- [53] H. Hirschmuller and D. Scharstein, “Evaluation of stereo matching costs on images with radiometric differences,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 31, no. 9, pp. 1582–1599, 2009. 26
- [54] A. Olofsson, “Modern stereo correspondence algorithms: investigation and evaluation,” Ph.D. dissertation, Master Thesis, Department of electrical engineering, Linköpings Universitet, 2010. 27
- [55] M. Bleyer, C. Rother, and P. Kohli, “Surface stereo with soft segmentation,” in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on.* IEEE, 2010, pp. 1570–1577. 27
- [56] M. Bleyer, C. Rhemann, and C. Rother, “Patchmatch stereo-stereo matching with slanted support windows.” in *BMVC*, vol. 11, 2011, pp. 1–11. 27
- [57] M. Bleyer, C. Rother, P. Kohli, D. Scharstein, and S. Sinha, “Object stereojoint stereo matching and object segmentation,” in *Computer*

- Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on.* IEEE, 2011, pp. 3081–3088. 27
- [58] J. Lu, D. Min, R. S. Pahwa, and M. N. Do, “A revisit to mrf-based depth map super-resolution and enhancement,” in *Acoustics, Speech and Signal Processing (ICASSP), 2011 IEEE International Conference on.* IEEE, 2011, pp. 985–988. 28
- [59] L. De-Maeztu, S. Mattoccia, A. Villanueva, and R. Cabeza, “Linear stereo matching,” in *Computer Vision (ICCV), 2011 IEEE International Conference on.* IEEE, 2011, pp. 1708–1715. 28
- [60] X. Mei, X. Sun, M. Zhou, H. Wang, X. Zhang *et al.*, “On building an accurate stereo matching system on graphics hardware,” in *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on.* IEEE, 2011, pp. 467–474. 28
- [61] Y. Mizukami, K. Okada, A. Nomura, S. Nakanishi, and K. Tadamura, “Sub-pixel disparity search for binocular stereo vision,” in *Pattern Recognition (ICPR), 2012 21st International Conference on.* IEEE, 2012, pp. 364–367. 28
- [62] S. Zhu, L. Zhang, and H. Jin, “A locally linear regression model for boundary preserving regularization in stereo matching,” in *Computer Vision–ECCV 2012.* Springer, 2012, pp. 101–115. 29
- [63] Z. Lee, J. Juang, and T. Nguyen, “Local disparity estimation with three-moded cross census and advanced support weight,” *Multimedia, IEEE Transactions on*, vol. 15, no. 8, pp. 1855–1864, Dec 2013. 29

- [64] D. Chen, M. Ardabilian, and L. Chen, “A novel trilateral filter based adaptive support weight method for stereo matching,” in *British Machine Vision Conference (BMVC)*, 2013. 29
- [65] J. Lu, H. Yang, D. Min, and M. N. Do, “Patch match filter: Efficient edge-aware filtering meets randomized search for fast correspondence field estimation,” in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on.* IEEE, 2013, pp. 1854–1861. 29
- [66] X. Mei, X. Sun, W. Dong, H. Wang, and X. Zhang, “Segment-tree based cost aggregation for stereo matching,” in *Computer Vision and Pattern Recognition (CVPR), 2013 IEEE Conference on.* IEEE, 2013, pp. 313–320. 29
- [67] X. Tan, C. Sun, X. Sirault, R. Furbank, and T. D. Pham, “Stereo matching using cost volume watershed and region merging,” *Signal Processing: Image Communication*, 2014. 30
- [68] Q. Yang, “Stereo matching using tree filtering,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PP, no. 99, pp. 1–1, 2014. 30
- [69] D. Vu, B. Chidester, H. Yang, M. Do, and J. Lu, “Efficient hybrid tree-based stereo matching with applications to postcapture image refocusing,” *Image Processing, IEEE Transactions on*, vol. 23, no. 8, pp. 3428–3442, Aug 2014. 30
- [70] X. Tan, C. Sun, D. Wang, Y. Guo, and T. D. Pham, “Soft cost aggregation with multi-resolution fusion,” in *Computer Vision–ECCV 2014.* Springer, 2014, pp. 17–32. 31

- [71] J. Liu, C. Li, F. Mei, and Z. Wang, “3d entity-based stereo matching with ground control points and joint second-order smoothness prior,” *The Visual Computer*, pp. 1–17, 2014. 31
- [72] J. Peng, Y. Huang, C.-C. J. Kuo, I. Eckstein, and M. Gopi, “Feature oriented progressive lossless mesh coding,” in *Computer Graphics Forum*, vol. 29. Wiley Online Library, 2010, pp. 2029–2038. 33, 230, 234
- [73] Z. Karni and C. Gotsman, “Spectral compression of mesh geometry,” in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH ’00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000, pp. 279–286. [Online]. Available: <http://dx.doi.org/10.1145/344779.344924> 34, 236, 237
- [74] C. Touma and C. Gotsman, “Triangle mesh compression,” in *Graphics interface*. CANADIAN INFORMATION PROCESSING SOCIETY, 1998, pp. 26–34. 34, 235, 237
- [75] U. Bayazit, U. Konur, and H. Ates, “3-d mesh geometry compression with set partitioning in the spectral domain,” *Circuits and Systems for Video Technology, IEEE Transactions on*, vol. 20, no. 2, pp. 179–188, 2010. 34, 230, 232
- [76] A. Khodakovskiy, P. Schröder, and W. Sweldens, “Progressive geometry compression,” in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, ser. SIGGRAPH ’00. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.,

- 2000, pp. 271–278. [Online]. Available: <http://dx.doi.org/10.1145/344779.344922> 34, 230, 232, 239
- [77] I. Guskov, K. Vidimče, W. Sweldens, and P. Schröder, “Normal meshes,” in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., 2000, pp. 95–102. 34
- [78] A. Khodakovsky and I. Guskov, “Compression of normal meshes,” in *Geometric modeling for scientific visualization*. Springer, 2004, pp. 189–206. 34
- [79] S. Rusinkiewicz, O. Hall-Holt, and M. Levoy, “Real-time 3d model acquisition,” *ACM Transactions on Graphics (TOG)*, vol. 21, no. 3, pp. 438–446, 2002. 36, 60, 64, 72
- [80] G. Passalis, T. Theoharis, G. Toderici, and I. A. Kakadiaris, “General voxelization algorithm with scalable gpu implementation,” *journal of graphics, gpu, and game tools*, vol. 12, no. 1, pp. 61–71, 2007. xiii, 36
- [81] B. Curless and M. Levoy, “A volumetric method for building complex models from range images,” in *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, 1996, pp. 303–312. 36, 63, 64
- [82] M. Cubes, “A high resolution 3d surface construction algorithm/william e,” *Lorensen, Harvey E. Cline-SIG 87*, 1987. 37
- [83] S. Parker, P. Shirley, Y. Livnat, C. Hansen, and P.-P. Sloan, “Interactive ray tracing for isosurface rendering,” in *Proceedings of the con-*

- ference on Visualization'98.* IEEE Computer Society Press, 1998, pp. 233–238. 37
- [84] C. Sigg, R. Peikert, and M. Gross, “Signed distance transform using graphics hardware,” in *Proceedings of the 14th IEEE Visualization 2003 (VIS'03)*. IEEE Computer Society, 2003, p. 12. xiii, 37
- [85] D. Canelhas, “Scene representation, registration and object detection in a truncated signed distance function representation of 3d space,” *Master's Thesis*, 2012. 37
- [86] E. Bylow, J. Sturm, C. Kerl, F. Kahl, and D. Cremers, “Real-time camera tracking and 3d reconstruction using signed distance functions,” in *Robotics: Science and Systems (RSS) Conference 2013*, vol. 9, 2013. 37, 61, 72
- [87] D. B. Kubacki, H. Q. Bui, S. D. Babacan, and M. N. Do, “Registration and integration of multiple depth images using signed distance function,” in *IS&T/SPIE Electronic Imaging*. International Society for Optics and Photonics, 2012, pp. 82960Z–82960Z. 38
- [88] C. Y. Ren and I. Reid, “A unified energy minimization framework for model fitting in depth,” in *European Conference on Computer Vision*. Springer, 2012, pp. 72–82. 38
- [89] A. Elfes and L. Matthies, “Sensor integration for robot navigation: combining sonar and stereo range data in a grid-based representataion,” in *Decision and Control, 1987. 26th IEEE Conference on*, vol. 26. IEEE, 1987, pp. 1802–1807. 38

- [90] C. Zach, T. Pock, and H. Bischof, “A globally optimal algorithm for robust tv-l 1 range image integration,” in *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on.* IEEE, 2007, pp. 1–8. 38
- [91] M. Herbert, C. Caillas, E. Krotkov, I.-S. Kweon, and T. Kanade, “Terrain mapping for a roving planetary explorer,” in *Robotics and Automation, 1989. Proceedings., 1989 IEEE International Conference on.* IEEE, 1989, pp. 997–1002. 38
- [92] R. Triebel, P. Pfaff, and W. Burgard, “Multi-level surface maps for outdoor terrain mapping and loop closing,” in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on.* IEEE, 2006, pp. 2276–2282. 38
- [93] X. Gu, S. J. Gortler, and H. Hoppe, “Geometry images,” *ACM Trans. Graph.*, vol. 21, no. 3, pp. 355–361, Jul. 2002. [Online]. Available: <http://doi.acm.org/10.1145/566654.566589> 39
- [94] H. Samet, *Foundations of Multidimensional And Metric Data Structures*, ser. Morgan Kaufmann. Elsevier/Morgan Kaufmann, 2006. [Online]. Available: <http://books.google.com.au/books?id=vO-NRRKHG84C> 40
- [95] K. M. Wurm, A. Hornung, M. Bennewitz, C. Stachniss, and W. Burgard, “Octomap: A probabilistic, flexible, and compact 3d map representation for robotic systems,” in *Proc. of the ICRA 2010 workshop on best practice in 3D perception and modeling for mobile manipulation*, vol. 2, 2010. 40, 52, 71

- [96] J. Fournier, B. Ricard, and D. Laurendeau, “Mapping and exploration of complex environments using persistent 3d model,” in *Computer and Robot Vision, 2007. CRV’07. Fourth Canadian Conference on.* IEEE, 2007, pp. 403–410. 40
- [97] D. Meagher, “Geometric modeling using octree encoding,” *Computer graphics and image processing*, vol. 19, no. 2, pp. 129–147, 1982. 40
- [98] N. Fairfield, G. Kantor, and D. Wettergreen, “Real-time slam with octree evidence grids for exploration in underwater tunnels,” *Journal of Field Robotics*, vol. 24, no. 1-2, pp. 03–21, 2007. 40
- [99] E. Olson, J. Leonard, and S. Teller, “Fast iterative alignment of pose graphs with poor initial estimates,” in *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.* IEEE, 2006, pp. 2262–2269. 41
- [100] U. Frese, P. Larsson, and T. Duckett, “A multilevel relaxation algorithm for simultaneous localization and mapping,” *IEEE Transactions on Robotics*, vol. 21, no. 2, pp. 196–207, 2005. 41
- [101] F. Dellaert and M. Kaess, “Square root sam: Simultaneous localization and mapping via square root information smoothing,” *The International Journal of Robotics Research*, vol. 25, no. 12, pp. 1181–1203, 2006. 41
- [102] S. Thrun *et al.*, “Robotic mapping: A survey,” *Exploring artificial intelligence in the new millennium*, pp. 1–35, 2002. 41
- [103] A. Nuchter, K. Lingemann, J. Hertzberg, and H. Surmann, “6d slam with approximate data association,” in *ICAR’05. Proceedings., 12th*

- International Conference on Advanced Robotics, 2005.* IEEE, 2005, pp. 242–249. 41
- [104] G. Grisetti, S. Grzonka, C. Stachniss, P. Pfaff, and W. Burgard, “Efficient estimation of accurate maximum likelihood maps in 3d,” in *2007 IEEE/RSJ International Conference on Intelligent Robots and Systems.* IEEE, 2007, pp. 3472–3478. 41
- [105] M. Kaess, A. Ranganathan, and F. Dellaert, “isam: Incremental smoothing and mapping,” *IEEE Transactions on Robotics*, vol. 24, no. 6, pp. 1365–1378, 2008. 41
- [106] M. Callieri, A. Fasano, G. Impoco, P. Cignoni, R. Scopigno, G. Parolini, and G. Biagini, “Roboscan: an automatic system for accurate and unattended 3d scanning,” in *3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004. Proceedings. 2nd International Symposium on.* IEEE, 2004, pp. 805–812. 41
- [107] A. J. Davison, “Real-time simultaneous localisation and mapping with a single camera,” in *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on.* IEEE, 2003, pp. 1403–1410. 41
- [108] H. Strasdat, J. Montiel, and A. J. Davison, “Real-time monocular slam: Why filter?” in *Robotics and Automation (ICRA), 2010 IEEE International Conference on.* IEEE, 2010, pp. 2657–2664. 41, 42
- [109] H. Jin, P. Favaro, and S. Soatto, “Real-time 3d motion and structure of point features: a front-end system for vision-based control and interaction,” in *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, vol. 2. IEEE, 2000, pp. 778–779. 41

- [110] D. Nistér, “Preemptive ransac for live structure and motion estimation,” *Machine Vision and Applications*, vol. 16, no. 5, pp. 321–329, 2005. 41
- [111] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burghard, “An evaluation of the rgb-d slam system,” in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*. IEEE, 2012, pp. 1691–1696. xiv, 41, 51, 53, 57
- [112] K. Konolige, M. Agrawal, R. C. Bolles, C. Cowan, M. Fischler, and B. Gerkey, “Outdoor mapping and navigation using stereo vision,” in *Experimental Robotics*. Springer, 2008, pp. 179–190. 41
- [113] L. M. Paz, P. Piniés, J. D. Tardós, and J. Neira, “Large-scale 6-dof slam with stereo-in-hand,” *IEEE transactions on robotics*, vol. 24, no. 5, pp. 946–957, 2008. 41
- [114] P. A. Beardsley, A. Zisserman, and D. W. Murray, “Sequential updating of projective and affine structure from motion,” *International journal of computer vision*, vol. 23, no. 3, pp. 235–259, 1997. 42
- [115] E. R. Davies, *Computer and machine vision: theory, algorithms, practicalities*. Academic Press, 2012. xiii, 45
- [116] A. J. Davison and D. W. Murray, “Simultaneous localization and map-building using active vision,” *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 24, no. 7, pp. 865–880, 2002. 48
- [117] W. Y. Jeong and K. M. Lee, “Visual slam with line and corner features,” in *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*. IEEE, 2006, pp. 2570–2575. 48

- [118] G. Silveira, E. Malis, and P. Rives, “An efficient direct approach to visual slam,” *Robotics, IEEE Transactions on*, vol. 24, no. 5, pp. 969–979, 2008. 48
- [119] D. Chekhlov, M. Pupilli, W. Mayol, and A. Calway, “Robust real-time visual slam using scale prediction and exemplar based feature description,” in *Computer Vision and Pattern Recognition, 2007. CVPR’07. IEEE Conference on.* IEEE, 2007, pp. 1–7. 48
- [120] P. Jensfelt, D. Kragic, J. Folkesson, and M. Bjorkman, “A framework for vision based bearing only 3d slam,” in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on.* IEEE, 2006, pp. 1944–1950. 48
- [121] M. Pollefeys, D. Nistér, J.-M. Frahm, A. Akbarzadeh, P. Mordohai, B. Clipp, C. Engels, D. Gallup, S.-J. Kim, P. Merrell *et al.*, “Detailed real-time urban 3d reconstruction from video,” *International Journal of Computer Vision*, vol. 78, no. 2-3, pp. 143–167, 2008. 48
- [122] C. Beall, F. Dellaert, I. Mahon, and S. B. Williams, “Bundle adjustment in large-scale 3d reconstructions based on underwater robotic surveys,” in *OCEANS, 2011 IEEE-Spain.* IEEE, 2011, pp. 1–6. 48
- [123] A. Eudes, M. Lhuillier, S. Naudet-Collette, and M. Dhome, “Fast odometry integration in local bundle adjustment-based visual slam,” in *Pattern Recognition (ICPR), 2010 20th International Conference on.* IEEE, 2010, pp. 290–293. 48, 49
- [124] A. Kundu, K. M. Krishna, and C. Jawahar, “Realtime motion segmentation based multibody visual slam,” in *Proceedings of the Seventh*

Indian Conference on Computer Vision, Graphics and Image Processing. ACM, 2010, pp. 251–258. 48

- [125] T. Leelasawassuk and W. W. Mayol-Cuevas, “3d from looking: using wearable gaze tracking for hands-free and feedback-free object modelling,” in *Proceedings of the 17th annual international symposium on International symposium on wearable computers.* ACM, 2013, pp. 105–112. 48
- [126] K. Konolige, J. Bowman, J. Chen, P. Mihelich, M. Calonder, V. Lepetit, and P. Fua, “View-based maps,” *The International Journal of Robotics Research*, 2010. 48
- [127] K. Konolige and M. Agrawal, “Frameslam: From bundle adjustment to real-time visual mapping,” *Robotics, IEEE Transactions on*, vol. 24, no. 5, pp. 1066–1077, 2008. 48
- [128] V. Pradeep, C. Rhemann, S. Izadi, C. Zach, M. Bleyer, and S. Bathiche, “Monofusion: Real-time 3d reconstruction of small scenes with a single web camera,” in *Mixed and Augmented Reality (ISMAR), 2013 IEEE International Symposium on.* IEEE, 2013, pp. 83–88. 48
- [129] M. A. Fischler and R. C. Bolles, “Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,” *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981. 49, 52
- [130] C.-S. Chen, Y.-P. Hung, and J.-B. Cheng, “Ransac-based darces: A new approach to fast automatic registration of partially overlapping

- range images,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 21, no. 11, pp. 1229–1234, 1999. 49
- [131] D. P. Huttenlocher, “Fast affine point matching: An output-sensitive method,” in *Computer Vision and Pattern Recognition, 1991. Proceedings CVPR’91., IEEE Computer Society Conference on.* IEEE, 1991, pp. 263–268. 52
- [132] J. Sturm, S. Magnenat, N. Engelhard, F. Pomerleau, F. Colas, D. Cremers, R. Siegwart, and W. Burgard, “Towards a benchmark for rgbd slam evaluation,” in *RGB-D Workshop on Advanced Reasoning with Depth Cameras at Robotics: Science and Systems Conf.(RSS)*, 2011. 52
- [133] G. Bradski and A. Kaehler, *Learning OpenCV: Computer vision with the OpenCV library.* ” O'Reilly Media, Inc.”, 2008. 52
- [134] S. Umeyama, “Least-squares estimation of transformation parameters between two point patterns,” *IEEE Transactions on pattern analysis and machine intelligence*, vol. 13, no. 4, pp. 376–380, 1991. 53
- [135] A. S. Huang, A. Bachrach, P. Henry, M. Krainin, D. Maturana, D. Fox, and N. Roy, “Visual odometry and mapping for autonomous flight using an rgbd camera,” in *Robotics Research.* Springer, 2017, pp. 235–252. 54
- [136] H. Hirschmuller, P. R. Innocent, and J. M. Garibaldi, “Fast, unconstrained camera motion estimation from stereo without tracking and robust statistics,” in *Control, Automation, Robotics and Vision, 2002.*

ICARCV 2002. 7th International Conference on, vol. 2. IEEE, 2002, pp. 1099–1104. 54

- [137] A. Howard, “Real-time stereo visual odometry for autonomous ground vehicles,” in *Intelligent Robots and Systems, 2008. IROS 2008. IEEE/RSJ International Conference on*. IEEE, 2008, pp. 3946–3952. 54
- [138] B. K. Horn, “Closed-form solution of absolute orientation using unit quaternions,” *JOSA A*, vol. 4, no. 4, pp. 629–642, 1987. 54
- [139] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald, “Kintinuous: Spatially extended kinectfusion,” *MIT-CSAIL*, 2012. 55, 61
- [140] P. J. Besl and N. D. McKay, “Method for registration of 3-d shapes,” in *Robotics-DL tentative*. International Society for Optics and Photonics, 1992, pp. 586–606. 55, 68
- [141] S. Rusinkiewicz and M. Levoy, “Efficient variants of the icp algorithm,” in *3-D Digital Imaging and Modeling, 2001. Proceedings. Third International Conference on*. IEEE, 2001, pp. 145–152. 55, 67
- [142] A. Segal, D. Haehnel, and S. Thrun, “Generalized-icp.” in *Robotics: Science and Systems*, vol. 2, 2009. 55
- [143] F. Steinbrücker, J. Sturm, and D. Cremers, “Real-time visual odometry from dense rgb-d images,” in *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*. IEEE, 2011, pp. 719–722. 55

- [144] T. Tykkälä, C. Audras, and A. I. Comport, “Direct iterative closest point for real-time visual odometry,” in *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2050–2056. 55
- [145] C. Kerl, J. Sturm, and D. Cremers, “Robust odometry estimation for rgb-d cameras,” in *Robotics and Automation (ICRA), 2013 IEEE International Conference on*. IEEE, 2013, pp. 3748–3754. 55, 64
- [146] Y. Chen and G. Medioni, “Object modelling by registration of multiple range images,” *Image and vision computing*, vol. 10, no. 3, pp. 145–155, 1992. 55, 56, 68
- [147] J. Stückler and S. Behnke, “Robust real-time registration of rgb-d images using multi-resolution surfel representations,” in *Robotics; Proceedings of ROBOTIK 2012; 7th German Conference on*. VDE, 2012, pp. 1–4. 55
- [148] G. Blais and M. D. Levine, “Registering multiview range data to create 3d computer objects,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 17, no. 8, pp. 820–824, 1995. 56, 60
- [149] N. J. Mitra, N. Gelfand, H. Pottmann, and L. Guibas, “Registration of point cloud data from a geometric optimization perspective,” in *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. ACM, 2004, pp. 22–31. 56, 69
- [150] T. Whelan, H. Johannsson, M. Kaess, J. J. Leonard, and J. McDonald, “Robust real-time visual odometry for dense rgb-d mapping,” in

- Robotics and Automation (ICRA), 2013 IEEE International Conference on.* IEEE, 2013, pp. 5724–5731. 56, 61, 64
- [151] N. Engelhard, F. Endres, J. Hess, J. Sturm, and W. Burgard, “Real-time 3d visual slam with a hand-held rgb-d camera,” in *Proc. of the RGB-D Workshop on 3D Perception in Robotics at the European Robotics Forum, Västerås, Sweden*, vol. 180, 2011. 57
 - [152] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, “g 2 o: A general framework for graph optimization,” in *Robotics and Automation (ICRA), 2011 IEEE International Conference on.* IEEE, 2011, pp. 3607–3613. 57, 62, 71
 - [153] H. Pfister, M. Zwicker, J. Van Baar, and M. Gross, “Surfels: Surface elements as rendering primitives,” in *Proceedings of the 27th annual conference on Computer graphics and interactive techniques.* ACM Press/Addison-Wesley Publishing Co., 2000, pp. 335–342. 57
 - [154] A. Hornung, K. M. Wurm, and M. Bennewitz, “Humanoid robot localization in complex indoor environments,” in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on.* IEEE, 2010, pp. 1690–1695. 57
 - [155] J. Sturm, N. Engelhard, F. Endres, W. Burgard, and D. Cremers, “A benchmark for the evaluation of rgb-d slam systems,” in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems.* IEEE, 2012, pp. 573–580. 57
 - [156] M. Pauly, N. J. Mitra, J. Giesen, M. Gross, and L. J. Guibas, “Example-based 3d scan completion,” in *Proceedings of the Third*

- Eurographics Symposium on Geometry Processing*, ser. SGP '05. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2005. [Online]. Available: <http://dl.acm.org/citation.cfm?id=1281920.1281925> 58
- [157] B. J. Brown and S. Rusinkiewicz, “Global non-rigid alignment of 3-d scans,” in *ACM Transactions on Graphics (TOG)*, vol. 26. ACM, 2007, p. 21. 58
- [158] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, “Kinectfusion: Real-time dense surface mapping and tracking,” in *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*. IEEE, 2011, pp. 127–136. xiv, 58, 59, 62, 63
- [159] S. Izadi, D. Kim, O. Hilliges, D. Molyneaux, R. Newcombe, P. Kohli, J. Shotton, S. Hodges, D. Freeman, A. Davison *et al.*, “Kinectfusion: real-time 3d reconstruction and interaction using a moving depth camera,” in *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 2011, pp. 559–568. 58
- [160] C. Rasch and T. Satzger, “Remarks on the $O(n)$ implementation of the fast marching method, preprint, 2007,” 2009. 60
- [161] M. Kaess, H. Johannsson, R. Roberts, V. Ila, J. Leonard, and F. Dellaert, “isam2: Incremental smoothing and mapping with fluid relinearization and incremental variable reordering,” in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 3281–3288. 62

- [162] Y. Ma, S. Soatto, J. Kosecka, and S. S. Sastry, *An invitation to 3-d vision: from images to geometric models.* Springer Science & Business Media, 2012, vol. 26. 63
- [163] J. A. Bærentzen, “On the implementation of fast marching methods for 3d lattices,” Department of Informatics and Mathematical Modeling, Technical University of Denmark, Tech. Rep., 2001. 63
- [164] W. E. Lorensen and H. E. Cline, “Marching cubes: A high resolution 3d surface construction algorithm,” in *ACM siggraph computer graphics*, vol. 21. ACM, 1987, pp. 163–169. 64
- [165] C. Kerl, J. Sturm, and D. Cremers, “Dense visual slam for rgb-d cameras,” in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on.* IEEE, 2013, pp. 2100–2106. 64
- [166] D. Weikersdorfer, D. B. Adrian, D. Cremers, and J. Conradt, “Event-based 3d slam with a depth-augmented dynamic vision sensor,” in *Robotics and Automation (ICRA), 2014 IEEE International Conference on.* IEEE, 2014, pp. 359–364. 65
- [167] P. Scovanner, S. Ali, and M. Shah, “A 3-dimensional sift descriptor and its application to action recognition,” in *Proceedings of the 15th ACM international conference on Multimedia.* ACM, 2007, pp. 357–360. 65, 68
- [168] G. T. Flitton, T. P. Breckon, and N. M. Bouallagu, “Object recognition using 3d sift in complex ct volumes.” in *BMVC*, 2010, pp. 1–12. 65, 68

- [169] X. Li and I. Guskov, “Multiscale features for approximate alignment of point-based surfaces.” in *Symposium on geometry processing*, vol. 255. Citeseer, 2005, pp. 217–226. 65, 68
- [170] H. J. Wolfson and I. Rigoutsos, “Geometric hashing: An overview,” *IEEE computational science and engineering*, vol. 4, no. 4, pp. 10–21, 1997. 65, 69
- [171] D. Aiger, N. J. Mitra, and D. Cohen-Or, “4-points congruent sets for robust pairwise surface registration,” *ACM Transactions on Graphics (TOG)*, vol. 27, no. 3, p. 85, 2008. 66, 70
- [172] S. Irani and P. Raghavan, “Combinatorial and experimental results for randomized point matching algorithms,” in *Proceedings of the twelfth annual symposium on Computational geometry*. ACM, 1996, pp. 68–77. 67
- [173] N. Gelfand and L. J. Guibas, “Shape segmentation using local slippage analysis,” in *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*. ACM, 2004, pp. 214–223. 67
- [174] G. Mori, S. Belongie, and J. Malik, “Efficient shape matching using shape contexts,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 27, no. 11, pp. 1832–1837, 2005. 69
- [175] A. E. Johnson, “Spin-images: a representation for 3-d surface matching,” Ph.D. dissertation, Citeseer, 1997. 69
- [176] R. S. Germain, A. Califano, S. Colville *et al.*, “Fingerprint matching using transformation parameter clustering,” *IEEE Computational Science and Engineering*, vol. 4, no. 4, pp. 42–49, 1997. 69

- [177] R. Gal and D. Cohen-Or, “Salient geometric features for partial shape matching and similarity,” *ACM Transactions on Graphics (TOG)*, vol. 25, no. 1, pp. 130–150, 2006. 69
- [178] D. H. Ballard, “Generalizing the hough transform to detect arbitrary shapes,” *Pattern Recognition*, vol. 13, no. 2, pp. 183–194, 1991. 69
- [179] H. Pottmann, J. Wallner, Y.-L. Yang, Y.-K. Lai, and S.-M. Hu, “Principal curvatures from the integral invariant viewpoint,” *Computer Aided Geometric Design*, vol. 24, no. 8, pp. 428–442, 2007. 70
- [180] M. I. Lourakis and A. A. Argyros, “Sba: A software package for generic sparse bundle adjustment,” *ACM Transactions on Mathematical Software (TOMS)*, vol. 36, no. 1, p. 2, 2009. 71
- [181] N. Fioraio and K. Konolige, “Realtime visual and point cloud slam,” in *Proc. of the RGB-D workshop on advanced reasoning with depth cameras at robotics: Science and Systems Conf.(RSS)*, vol. 27, 2011. 72
- [182] S. Agarwal, N. Snavely, I. Simon, S. M. Seitz, and R. Szeliski, “Building rome in a day,” in *2009 IEEE 12th international conference on computer vision.* IEEE, 2009, pp. 72–79. 72
- [183] R. Gonzalez, “Improving phase correlation for image registration,” in *Image And Vision Computing New Zealand 2011 (IVCNZ2011).* <http://ieeexplore.ieee.org/xpl/conhome.jsp?punumber=1002602>, 2011. 73

- [184] P. Vinukonda, “A study of the scale-invariant feature transform on a parallel pipeline,” Master’s thesis, Louisiana State University and Agricultural and Mechanical College, May 2011. 142
- [185] I. M. Johnstone and A. Y. Lu, “Sparse principal components analysis,” *Unpublished manuscript*, vol. 7, 2004. 144
- [186] L. Lincoln, “Hierarchical techniques in 3d model compression,” Griffith University Honors Thesis, 2013, undergraduate Honours Thesis. 162, 230, 232
- [187] R. Gonzalez, “Shadetree image compression for embedded computing,” in *10th International Symposium on DSP and Communication Systems, DSPCS*, Gold Coast, December 2007. 162, 165, 170
- [188] A. Hornung, K. M. Wurm, M. Bennewitz, C. Stachniss, and W. Burgard, “Octomap: An efficient probabilistic 3d mapping framework based on octrees,” *Autonomous Robots*, vol. 34, no. 3, pp. 189–206, 2013. xiv, xv, 164, 165
- [189] I. Gargantini, “An effective way to represent quadtrees,” *Commun. ACM*, vol. 25, no. 12, pp. 905–910, Dec. 1982. [Online]. Available: <http://doi.acm.org/10.1145/358728.358741> 169
- [190] L. Yufei, “Octcodes and their use in representing three-dimensional objects,” in *Systems, Man, and Cybernetics, 1988. Proceedings of the 1988 IEEE International Conference on*, vol. 2, 1988, pp. 1125–1127. 169
- [191] K. Varma, S. Parthasarathy, and P. Sankaran, “Application of quad tree for low bitrate compression,” in *Computational Intelligence and*

Signal Processing (CISP), 2012 2nd National Conference on, 2012, pp. 104–108. 169

- [192] R. Schnabel and R. Klein, “Octree-based point-cloud compression.” *Spbg*, vol. 6, pp. 111–120, 2006. 170
- [193] H. Samet and R. Webber, “Hierarchical data structures and algorithms for computer graphics. i. fundamentals,” *Computer Graphics and Applications, IEEE*, vol. 8, no. 3, pp. 48–68, 1988. 170
- [194] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun, “Vision meets robotics: The kitti dataset,” *The International Journal of Robotics Research*, vol. 32, no. 11, pp. 1231–1237, 2013. 182, 196

Appendix A

.1 Stereo Results

Table 1: Raw Registration Results for the Kitti Data 0001 Sync Data Set

frame	FM-2D	FM-3D	ICP	PCA	FVR	FFVR	FVR-3D
0	4.61767	5.32903	4.29533	5.43002	5.77859	4.48089	4.31634
1	5.58073	97838.7	5.02896	5.95525	5.46572	9.46393	5.28962
2	6.02244	105425	4.9742	5.67723	6.17976	6.27451	4.98074
3	4.18465	139097	3.69935	4.41445	4.53483	4.0578	3.97824
4	4.87622	5.57205	4.28291	6.00352	4.71944	5.5889	4.24961
5	5.76827	5.56533	5.2571	6.01189	5.76413	5.58867	4.62449
6	5.99727	25864.9	5.42174	6.48775	6.68527	6.5874	5.75757
7	5.45112	5.66693	5.631	5.96035	6.77246	5.49704	5.97797
8	5.59298	5.35189	4.63077	5.30576	4.72905	6.30678	4.234
9	4.86083	5.80337	4.73291	5.68578	5.98504	4.80517	4.46505
10	5.44836	9235.71	6.36579	6.5696	5.20186	8.37353	4.98271
11	5.23478	5.57038	6.74113	6.07219	6.47283	6.6327	4.90577
12	7.58893	19552.4	7.03406	7.71283	7.08542	8.35665	6.84138
13	4.63246	4.90545	4.65159	4.59741	4.86233	5.04928	5.00301
14	5.39655	5.42601	5.30633	5.8112	5.29114	5.17521	4.57732
15	5.27262	6.47485	5.09788	5.2152	6.40838	5.40483	4.72465
16	5.5436	3766.04	5.461	5.69188	6.91227	5.38372	5.81891
17	4.81508	130851	4.59424	5.62355	4.50471	5.4209	4.63469
18	4.00779	4214.9	3.92531	4.59796	3.94754	4.0559	4.0321
19	4.28421	5.10618	4.52619	4.72885	4.43962	3.98497	4.24734

20	4.48135	5.21103	4.8585	4.81297	4.86002	4.90852	4.46801
21	6.64	6.64684	4.59663	5.06546	5.23464	5.21956	4.25693
22	5.13697	5.17023	4.59618	5.1936	5.12897	5.74173	4.53822
23	4.64986	4.93649	4.52268	4.97689	4.70366	5.04662	4.40918
24	4.84677	6.083	5.40279	5.32683	4.58482	5.86823	4.75154
25	5.2719	15604	5.12423	6.43023	5.41725	5.99697	6.22611
26	4.46766	4.85459	5.03549	5.04746	5.14205	6.75146	4.35163
27	5.66482	99429.5	5.5768	5.65694	5.47218	5.95852	5.29211
28	5.92643	937.023	7.08007	6.11491	6.59816	6.13918	5.36414
29	6.16881	6.17455	6.07677	6.41914	5.31262	6.42176	5.90582
30	5.17311	5.10011	4.93152	6.16958	4.54295	5.49422	4.56949
31	5.31275	87463.4	4.3208	5.49572	5.66624	5.35319	4.76579
32	5.55481	5.36219	5.11154	5.23516	5.69667	5.64758	5.251
33	4.38553	5.24997	4.97161	4.72492	4.93188	12.8416	4.26143
34	4.95424	52413.3	4.65631	5.0355	4.85058	7.43292	4.42355
35	4.47359	83960	4.29008	4.51898	4.32231	4.25752	3.92223
36	4.70117	4.77106	4.31962	4.48925	4.45238	4.45086	4.35791
37	4.6463	4.4695	4.07065	4.47837	4.30519	7.15793	4.0775
38	5.76517	83892	5.14944	5.42855	5.67629	5.58788	5.60871
39	5.12901	5.15631	4.98215	5.69549	5.71745	5.89268	5.29639
40	5.16652	86254	4.46058	5.51603	6.94176	4.52279	5.03098
41	4.33504	491.951	4.31374	4.6333	4.24833	4.98012	4.10088
42	4.49201	54392.8	4.29026	4.69413	4.61577	5.32668	4.68923
43	4.97777	93890	5.23906	5.36553	4.83793	5.78882	4.4541
44	4.39895	4.47994	4.15503	4.6755	4.0051	4.79127	3.98608
45	4.6658	95361.7	4.49009	4.84021	4.61306	5.01277	4.64188
46	6.00234	41131.4	4.9628	6.16019	5.49597	5.66449	5.08847
47	7.04882	28355.6	10.9921	4.98743	5.96323	10.8524	5.80936
48	5.45541	72802.5	5.37284	5.98864	6.45666	5.24467	5.61791
49	6.72714	33528.1	7.19572	6.99925	7.75982	7.37557	7.26875
50	5.38314	96124.9	5.35687	6.05051	5.28716	5.65055	5.07307
51	5.9336	5.33317	6.8648	6.88309	5.61785	5.82357	6.37646
52	4.88867	64130.3	4.22148	4.87104	4.38427	4.4151	4.37996
53	5.29808	2887.02	5.12657	5.74233	4.89354	5.4175	5.02644
54	4.95216	108236	4.64082	5.40395	5.08319	6.53337	5.10334
55	4.60462	2815.28	5.158	5.10158	5.11529	4.64934	4.69914
56	6.01715	7.54336	6.68939	7.28164	6.64857	5.7791	6.39782

57	4.3483	16703.2	4.59122	5.03231	4.47041	4.47593	4.077
58	5.32076	82345.1	5.28358	5.40035	5.1975	5.13673	4.60226
59	5.01778	21587.7	4.64971	4.94375	5.25643	5.09237	4.18604
60	4.19817	103299	4.45638	4.71047	4.39606	4.57925	4.4956
61	4.92186	5.50674	6.10912	5.71122	5.19694	5.25735	5.29724
62	5.12582	85913.6	4.80604	5.25928	5.50357	5.08839	4.78205
63	4.21642	4315.31	4.32867	5.17689	4.29473	4.70596	4.35598
64	5.27832	79753	5.11255	9.40166	6.20885	5.52206	5.48644
65	8.50477	9858.51	8.75755	5.84533	7.87121	8.78859	5.97812
66	6.18098	93172.7	6.04108	6.49413	5.58389	5.88168	5.86359
67	7.51411	6.87623	6.88545	9.42718	6.83282	7.58698	6.47571
68	6.78666	7.15313	7.71408	7.83395	6.50161	6.75154	7.11524
69	5.20742	13205.1	5.83383	5.6281	5.62968	5.35961	5.2369
70	6.02765	107876	6.40966	6.32437	5.70185	6.10728	6.62557
71	6.71366	23425.1	9.7965	6.675	6.59859	6.36746	6.13763
72	6.88908	7.62445	6.66949	8.89236	6.53109	6.98776	7.07059
73	4.98563	34834.1	4.85086	5.5239	4.94735	5.05666	4.73225
74	5.80474	15148.4	5.85557	6.75817	6.18985	5.98803	5.91941
75	4.93381	37408.8	5.76066	5.43098	5.51797	5.19714	5.31495
76	6.08017	102820	6.27288	7.14073	5.69086	5.73698	5.8403
77	7.48623	6.84256	7.40882	9.29908	6.7625	7.05972	6.18532
78	6.04402	6.50972	6.82211	7.14161	6.2057	6.03227	6.29879
79	5.21598	18131.7	4.92335	5.00528	5.87098	5.27716	6.01284
80	6.21473	121485	7.12803	8.05202	6.21121	6.35647	6.27128
81	6.02357	6.01517	5.85607	6.20492	5.89878	6.78389	5.74828
82	6.17589	6.21922	5.85031	6.09165	6.17458	5.87716	6.82106
83	7.35138	15799	6.7332	6.80213	7.23032	7.2943	6.92823
84	6.48219	9.34383	6.95575	6.77574	6.48171	6.41718	6.43238
85	5.54082	8.47931	5.33056	5.42163	5.96968	5.10065	5.3478
86	5.18582	5.2871	4.91016	5.3645	5.50116	5.08923	5.33711
87	5.79225	7.65251	5.44758	6.08859	5.4975	5.4527	5.64501
88	6.4899	133086	6.14546	8.23461	6.56619	6.31743	7.978
89	6.26597	5.92558	5.39391	6.62096	5.83314	6.05755	5.92291
90	5.77945	5.72587	5.51713	6.27735	5.56167	5.97056	5.47656
91	5.60856	29660.8	5.75636	7.5598	5.71663	6.46676	5.7596
92	6.71433	129123	7.47842	8.25766	6.31632	6.77049	6.86332
93	4.86661	4.96833	4.33533	4.78395	4.54232	4.67157	4.70713

94	4.26698	78642.6	4.79331	4.66993	4.32696	4.38748	4.61306
95	4.67278	40305.6	4.95134	4.88113	4.51508	5.11602	4.16193
96	5.25928	141815	5.30732	5.30835	5.06861	5.20282	5.27435
97	5.83167	17499.7	5.15808	6.67218	5.04821	5.48233	5.13354
98	4.8974	5.27706	4.93718	4.73959	4.50308	4.91201	4.32168
99	4.50227	16800	4.59984	4.59646	4.73412	4.68926	4.77213
100	5.09285	68575.9	4.40759	5.12531	4.63031	5.03582	5.04469
101	5.70423	5.60461	6.13324	5.77986	6.3429	5.67327	5.34737
102	4.99854	66598.8	5.69001	5.75554	5.11098	4.99571	5.37832
103	6.17397	21481	6.10576	5.94686	6.24055	5.84899	5.67626
104	4.41834	4.62649	4.6936	4.60197	4.82708	4.57245	4.98001
105	4.75343	7.5081	4.96334	5.09375	4.99992	4.78582	5.12672

Table 2: Raw Registration Results for the Kitti Data 0002 Sync Data Set

frame	FM-2D	FM-3D	ICP	PCA	FVR	FFVR	FVR-3D
0	5.05086	4.94072	4.33802	4.93391	5.25059	5.55132	4.65052
1	6.16482	6.21841	6.2616	6.47462	6.35779	6.37448	5.78909
2	6.85857	6.81943	6.0864	6.43713	6.15429	7.30368	6.521
3	16.2066	17.0807	15.6458	16.4459	16.3367	16.5471	15.9051
4	3.50198	3.86445	3.70225	4.80994	3.30395	5.12324	3.25735
5	4.58719	5.11774	4.67632	5.25638	4.45767	4.54029	4.41654
6	5.89601	5.62223	5.62543	5.84164	5.33532	5.92547	5.29331
7	6.27123	6.85567	6.29416	6.66982	8.09376	6.64017	6.24907
8	4.2759	4.15746	3.68789	3.97525	3.72886	3.65254	3.88771
9	7.01757	6.11245	6.973	6.58888	6.68508	6.5802	6.53747
10	5.50507	5.61392	4.97994	5.87173	7.20071	6.16221	5.09187
11	5.53911	5.23828	5.09409	5.90251	5.70355	5.64843	5.36757
12	4.00992	4.48817	4.9641	4.86449	4.18521	4.46608	4.88013
13	4.79772	4.38202	4.54437	4.85413	5.88243	5.23215	5.23913
14	4.30203	3.80502	3.47925	3.95896	4.10798	3.7717	3.66328
15	3.29963	4.56142	3.7165	4.39814	3.44285	3.5378	3.42315
16	3.62452	3.4494	3.99358	3.41332	3.4495	3.69912	3.02328

17	3.92782	4.51608	3.24577	3.59198	3.99868	3.83213	3.27206
18	3.3216	3.87889	3.13008	3.65673	3.197	3.49437	3.26597
19	3.94273	3.65291	3.19511	3.48252	3.92316	4.86699	3.95291
20	4.78113	4.9655	2.92096	3.13085	4.40756	3.66269	2.86573
21	4.83947	4.94489	4.36998	4.71949	4.55902	5.88064	4.53641
22	4.19892	4.34779	4.68206	4.45334	4.28699	4.70866	3.8302
23	3.83934	4.30842	2.92345	3.43902	3.54279	3.47873	3.119
24	4.05986	3.73301	3.32662	3.76528	3.58132	3.64945	3.62562
25	4.21654	3.83556	7.18737	6.41837	6.33025	4.71703	4.26803
26	5.14722	5.02735	3.92507	3.91014	4.75773	5.98422	5.24384
27	5.11527	4.20863	4.34208	4.20102	4.56545	4.56774	4.27486
28	3.66161	3.59021	3.40311	3.46239	3.683	3.82016	3.63548
29	3.95408	4.31996	3.82975	3.91458	4.49089	5.5189	3.40329
30	4.25563	4.1929	4.34456	3.92101	4.34509	5.56239	4.05533
31	5.21524	5.17247	4.70982	5.22438	7.41897	9.95607	4.6776
32	5.17715	5.21374	4.94463	5.8739	5.83717	16.583	6.04891
33	4.98865	4.40008	4.26455	4.25459	4.79723	8.95572	4.22342
34	5.23954	7.267	4.75395	6.66518	8.46972	7.12266	5.58569
35	3.3877	3.97702	3.36821	5.03337	3.24704	3.37048	3.41907
36	3.48066	3.60367	3.89699	3.37704	3.08839	4.53352	3.13086
37	4.88354	5.38869	4.42807	5.08647	3.88551	5.21012	3.51268
38	3.6797	4.25999	3.8853	4.06428	4.67129	3.53285	3.46861
39	3.31598	3.74634	3.29381	3.45276	3.11655	4.41216	3.28512
40	3.14752	3.41351	3.23871	3.08988	2.90705	4.19889	2.61041
41	3.30042	3.34641	2.9433	3.30551	3.21495	2.69907	2.71868
42	4.39651	4.01816	4.10507	4.36244	4.39134	5.01702	4.17671
43	5.27246	5.90762	5.26648	6.51495	6.5638	10.0018	5.88721
44	3.86735	4.37691	4.17666	4.17462	4.9276	6.94471	4.3548
45	4.52446	4.42972	3.89604	4.41536	4.62521	4.76991	4.14215
46	3.91289	3.51127	3.58506	3.49559	3.6775	3.5834	2.95704
47	3.55755	3.80074	3.28902	4.56383	3.16235	6.48683	3.13065
48	3.46484	3.20014	3.50221	3.46344	3.26319	4.09852	2.95531
49	4.47809	5.39685	4.08759	6.35505	4.38846	9.3472	4.09343
50	5.22748	5.41338	5.58195	6.12177	7.00019	6.06171	4.47842
51	4.50363	4.87936	5.2199	4.00299	4.99197	5.44073	3.84931
52	5.22387	6.63806	5.07476	5.84862	4.97406	5.05099	5.07
53	7.20089	7.05688	8.76378	5.38629	5.5521	8.49669	6.07093

54	8.32904	4.62892	4.74131	4.92097	6.7537	10.6078	7.06365
55	4.00526	4.08617	3.88554	4.09079	3.60441	3.66006	3.17973
56	4.9308	4.39881	4.61228	5.60492	4.88328	4.34345	4.67856
57	5.2231	5.7598	5.25082	5.75703	4.39591	5.92268	4.69769
58	4.76417	6.24875	6.12133	5.47654	4.66734	6.49817	4.61793
59	5.53649	5.13132	4.45987	4.98501	4.88897	5.09016	4.59719
60	3.74444	4.82462	4.3045	4.30769	3.88941	4.6428	3.54799
61	4.93808	6.15033	5.00965	5.4074	5.62225	6.86312	4.71968
62	7.50075	7.36632	6.39438	6.63146	6.39203	7.21487	6.07086
63	4.21978	4.59894	4.11312	4.19989	3.93838	4.78377	3.6263
64	5.79908	5.38649	5.46508	5.34054	5.54917	5.10662	5.12554
65	5.32582	6.79445	6.11662	5.94658	5.59066	5.82704	6.85421
66	8.87589	12.9965	6.12499	9.67651	10.0102	22.8587	8.76393
67	5.35725	4.15746	3.56487	4.07682	3.81467	4.3177	3.84733
68	5.65776	5.76836	5.05224	5.33078	5.86478	5.95726	25.0134
69	4.6246	4.85377	4.36558	4.01962	5.30473	4.62669	4.2464
70	5.29665	5.24988	7.48755	8.38293	5.19282	6.10896	4.99155
71	5.70508	5.58911	5.98723	6.6354	5.30171	5.76734	4.52703
72	4.49075	5.18428	5.54762	5.63067	4.64932	6.02883	4.06937
73	7.43427	9.20047	5.24347	6.59261	7.14297	6.10619	5.3853
74	5.21926	6.75513	6.61152	7.87317	5.29144	6.04834	4.77176

Table 3: Raw Registration Results for the Kitti Data 0005 Sync Data Set

frame	FM-2D	FM-3D	ICP	PCA	FVR	FFVR	FVR-3D
0	3.01521	3.52263	3.28737	3.49361	3.318	3.13712	3.12523
1	3.57114	3.72409	3.31727	3.7894	3.64221	3.35629	3.40463
2	3.92783	4.10737	3.59305	4.07092	3.90862	3.82918	3.59984
3	2.74277	3.06327	2.91408	3.15097	3.11469	3.07926	2.75131
4	3.14555	3.29445	2.91743	3.50396	3.32775	3.1152	2.83389
5	2.89419	3.02132	2.78311	2.95295	3.18609	3.2536	2.86449
6	2.9336	3.29831	3.96992	3.30168	3.15388	2.86465	2.64936
7	2.86871	2.94432	2.94496	5.12024	2.99927	4.24502	3.04891
8	7.58831	6.8406	6.39557	12.7329	7.05438	7.553	5.92742

9	2.77308	2.99177	2.99868	3.46837	3.02356	2.78323	3.02682
10	3.36637	3.47523	3.41459	3.48821	3.43474	3.41993	3.50498
11	3.06112	3.21383	2.93645	3.17493	3.23893	3.31025	3.05444
12	3.17281	3.28726	3.10581	3.24875	3.38734	3.28989	3.22893
13	2.99857	3.2921	2.86313	3.07366	3.12676	3.37476	3.19259
14	3.00257	2.95538	2.97537	2.96125	3.08415	3.74015	2.73596
15	3.27749	3.49342	3.33959	3.37736	3.57231	3.95443	3.51558
16	3.19235	3.24906	3.04698	3.59355	3.13325	3.39099	3.01746
17	3.14417	3.29079	2.9055	4.08508	3.82186	3.29132	3.2176
18	3.20557	3.39152	2.95968	4.64762	4.00774	4.13448	2.98165
19	3.53935	3.55701	3.07224	7.61314	3.66126	4.15869	3.38343
20	3.0849	3.50013	3.2849	3.37222	3.35693	3.22141	2.91417
21	3.0076	3.28182	3.1106	3.13281	3.27259	3.07241	3.06769
22	3.35104	3.65094	3.36737	3.52491	3.81723	3.90998	3.41595
23	2.88802	2.97642	2.8945	3.17434	3.25697	3.65218	2.96846
24	2.84726	3.08842	2.78837	3.35794	3.15197	3.81851	2.76637
25	4.28961	4.71112	4.07833	4.37954	4.2762	4.8818	3.71509
26	3.32014	3.68873	3.40318	3.64995	3.53366	3.70707	3.40809
27	3.11979	3.04011	2.94186	3.50703	3.13189	3.35855	2.86392
28	2.87431	3.15311	2.98676	3.64002	2.97954	3.50013	3.05055
29	3.27525	3.49703	3.40537	3.65103	3.89574	3.61917	3.56784
30	2.90046	3.04836	3.37415	3.0893	3.10765	3.14179	2.98234
31	2.97396	3.03441	3.02428	2.93347	2.95975	3.18538	2.89575
32	3.63129	3.86399	3.62324	3.84481	3.70994	4.1906	3.633
33	3.22849	3.32426	3.23822	3.19825	3.40103	3.86017	2.93833
34	3.53437	3.76252	3.62065	3.61229	3.64149	4.29813	3.58348
35	3.44967	3.7738	3.61985	3.53375	3.78265	3.89186	3.77966
36	3.27948	3.30567	2.83872	3.28768	3.27404	4.31733	2.9267
37	3.66436	3.48778	4.28293	4.08703	3.2567	3.67967	3.3717
38	3.43523	3.47158	3.68181	3.8077	3.49086	4.62702	3.34778
39	3.27957	3.81696	3.43704	3.90491	3.53208	4.42103	3.36713
40	3.52797	4.09686	3.63702	4.4489	3.99384	3.7518	3.88431
41	3.57476	4.19392	3.82461	4.85827	3.7044	4.2689	3.84919
42	3.89752	4.1224	3.75439	4.91952	3.75651	4.54563	3.80131
43	3.72801	4.19014	3.97585	4.50795	3.8687	4.33092	3.66612
44	4.52779	4.88412	4.29604	4.90124	4.45247	7.17366	4.75791
45	3.77379	4.29421	3.71045	4.4718	4.07625	4.22571	3.88649

46	3.77265	4.33479	3.83428	5.0414	4.36924	6.35137	3.86415
47	4.00467	4.32626	3.83034	4.35027	3.87158	4.70666	3.52464
48	3.90607	4.58639	3.93763	4.40093	4.07872	5.15218	3.9971
49	3.67369	4.45756	3.83058	5.53484	4.15714	5.83958	3.96865
50	5.03909	5.71321	5.02428	6.05693	5.3117	7.37711	5.02133
51	4.0141	4.75024	5.09563	4.80902	4.03686	5.09224	4.07236
52	3.79117	4.48313	4.23038	4.55379	3.99099	4.69596	3.80516
53	4.49073	5.00348	4.32903	4.65774	4.32919	4.98683	4.26132
54	4.63569	5.20045	4.7002	5.06335	4.62451	5.22759	5.34774
55	5.025	5.4998	5.08957	5.83647	5.12149	5.68152	5.21875
56	5.2018	5.59787	5.33846	5.97746	5.23428	5.67746	5.45837
57	3.96512	4.46025	4.12809	5.0011	4.17374	4.22283	5.00331
58	4.4388	4.85791	4.29608	5.48866	4.79097	4.51658	4.35657
59	4.1987	4.5886	4.61908	6.35728	4.36265	4.54053	405.815
60	4.62914	4.86112	4.51634	4.78178	5.06852	5.15841	160.969
61	3.78297	4.14543	3.77081	5.79791	4.21527	4.32231	4.05534
62	3.96241	4.13345	4.3018	7.83431	4.55273	4.03182	121.927
63	4.22055	4.37966	4.08152	4.7894	4.75861	4.40234	351.493
64	3.55722	3.79773	3.74036	3.86239	3.79635	3.70882	3.9656
65	3.6866	3.90329	3.91626	3.97619	4.07117	3.73607	4.23404
66	3.34817	3.77281	4.37171	3.91806	4.02186	3.46948	3.65077
67	3.67093	4.04358	3.63059	11.0312	3.61985	3.70272	3.67525
68	3.47175	3.73026	3.4178	18.2746	3.786	3.8179	3.63201
69	3.51717	4.17223	3.50985	15.9672	3.65743	4.29107	3.43013
70	3.64637	3.91932	3.53688	11.7846	3.56353	3.95945	3.56894
71	3.50544	3.97878	3.67579	10.1108	3.60769	4.3021	3.31989
72	3.04143	3.57118	3.3265	6.85616	3.27807	3.67733	3.04919
73	3.15844	3.82681	3.02824	8.42026	3.61003	4.13346	3.0116
74	3.42206	3.77414	3.52355	8.95378	4.68475	4.22276	3.48283
75	3.36306	3.91314	3.13695	5.01824	3.84038	8.67633	3.32066
76	4.69042	4.28976	4.54131	4.82168	4.70616	7.94131	4.27887
77	3.4315	4.05694	3.33369	3.91407	3.5526	7.48413	3.39175
78	3.31	4.23901	3.63224	4.21332	3.91306	6.84614	3.42682
79	3.10065	3.76404	3.08263	3.88862	3.25863	5.48206	3.35705
80	3.52829	4.31701	3.35552	4.19334	3.79705	4.48398	3.68534
81	3.9997	4.55302	4.10937	4.52364	4.34079	6.57591	3.69989
82	3.59615	4.90967	3.63712	4.31002	4.13669	6.46121	3.48681

83	5.69112	6.20602	5.42189	8.05533	5.80706	7.48815	5.3408
84	3.11549	3.949	3.32895	3.90789	2.92678	3.42232	3.06489
85	3.12453	3.89576	2.93395	3.34432	3.26193	4.29945	3.16182
86	3.25865	4.27483	3.51339	3.80668	3.74745	4.39328	3.63647
87	2.82187	3.84772	2.92611	3.7494	3.53558	5.28764	2.83437
88	3.32213	4.56032	3.37233	4.91373	3.57979	4.48022	3.50253
89	3.29259	4.53233	3.38222	3.99738	3.96658	4.19493	3.34083
90	3.25906	4.3849	3.37921	3.98247	3.48613	4.52379	3.24365
91	3.78295	4.69623	3.64849	4.89991	4.30572	6.62766	3.32426
92	3.36566	4.67062	3.46244	4.14147	3.31384	4.45309	3.37204
93	3.88313	5.21013	4.18872	4.38375	3.77892	6.06928	4.17301
94	4.93308	5.32001	5.08288	6.83811	5.24961	4.94514	6.81935
95	5.06164	5.67362	4.46178	8.7201	5.42492	5.46792	5.43285
96	5.49215	6.36297	5.93951	8.87457	5.7159	6.9371	5.8186
97	3.18284	4.30239	4.34386	10.3481	3.42649	4.787	3.28386
98	4.09378	5.02856	4.299	8.21753	4.74604	5.70541	4.34451
99	7.94985	8.08457	8.78945	9.13465	7.70943	7.74769	7.35432
100	3.62677	4.61846	3.68439	4.48327	4.07555	11.0223	3.65997
101	3.07968	3.94569	3.39432	3.9085	3.27319	5.4659	3.44774
102	3.34877	4.21029	3.10616	4.13027	4.11897	3.94797	3.46448
103	3.71335	4.66511	3.6492	5.29498	4.30019	5.57718	3.91481
104	12.0699	12.1126	11.9469	12.0476	12.1969	17.4558	12.2148
105	3.08625	3.69504	3.74267	4.06186	3.47473	8.20444	2.94782
106	2.69143	3.40796	2.50017	3.26971	3.29948	9.4231	2.45046
107	3.06977	3.50827	3.05965	3.40135	3.48239	13.4421	3.14159
108	2.89733	3.63042	3.13873	4.17132	3.37134	4.6938	2.76059
109	2.95574	3.83425	3.27541	4.14078	4.10294	3.85152	2.95242
110	2.93739	4.15422	3.25374	4.78018	3.02755	11.9726	2.97787
111	3.32056	3.82637	2.94276	4.35848	3.11028	5.62991	3.06246
112	3.34154	3.76153	3.19845	4.46801	3.10728	4.83702	3.13737
113	3.75286	3.78516	3.49167	4.67832	3.6342	3.88358	3.37971
114	3.63186	3.79923	3.58327	3.97362	3.64076	3.89708	3.69372
115	3.63327	4.64468	4.12119	3.72369	4.00742	3.80739	3.83758
116	3.4341	3.87291	3.46219	3.77406	3.567	4.77121	3.24444
117	3.24069	3.61616	3.12451	3.12863	3.27325	3.5489	3.20224
118	2.51054	3.38647	2.43687	3.32544	2.93356	4.37175	2.50532
119	2.45591	3.1867	2.33744	2.69671	2.82826	3.43951	2.32143

120	2.77965	3.45351	2.89651	3.06736	3.16702	4.11042	2.75973
121	4.22461	4.69892	4.54594	4.27196	4.55347	4.63538	4.27591
122	2.56812	3.22094	2.81397	3.04105	2.95849	2.89091	2.5457
123	2.35691	3.04393	2.25833	2.69767	2.6377	3.23025	2.24564
124	2.73379	3.69968	3.18975	3.6425	3.2547	5.23885	3.07352
125	3.14647	3.63675	2.99641	3.41658	3.66164	4.61397	3.18817
126	3.49849	4.31472	4.12078	3.94123	3.95313	5.79653	3.27907
127	2.46172	2.8361	2.2849	2.65918	3.24134	3.25147	2.56846
128	3.09599	3.37476	2.8895	3.51833	4.73081	9.1352	2.9273
129	3.43155	4.57896	4.33662	4.53295	4.18784	4.62106	4.22433
130	3.30248	3.60905	3.77767	4.07459	3.21146	9.66695	3.11375
131	3.61223	3.91976	3.26428	4.01468	3.57513	3.4133	3.05207
132	3.47796	3.77053	3.53752	3.92569	3.83154	3.59159	3.49204
133	3.15442	3.23675	3.11282	2.8417	3.12154	2.67026	2.54329
134	2.75232	3.30784	3.24992	3.21513	2.90813	2.91669	2.91323
135	3.45001	3.83381	3.9622	3.68049	3.25965	3.43447	3.51961
136	2.98583	3.687	2.83778	3.33999	3.67163	3.36177	2.98975
137	2.75016	3.48169	3.00118	3.32104	3.66531	3.40522	3.07556
138	2.64064	3.5185	2.9376	3.45712	3.7745	2.95251	2.79488
139	3.49585	3.668	3.53522	3.91486	3.72961	3.55346	3.6723
140	3.31433	3.70866	3.40932	3.85582	4.12332	3.61222	3.56165
141	4.2912	4.60199	4.14186	4.77492	5.05876	4.83079	4.50827
142	3.41839	3.69863	4.35561	7.26236	4.08322	37.0217	3.73901
143	3.10894	3.39844	3.35629	3.28729	3.48802	2.92667	2.93279
144	2.88524	3.60138	3.32236	3.11308	3.21221	2.91057	2.91887
145	3.23331	3.77679	3.41608	4.04887	3.37743	3.46049	3.23017
146	4.23638	5.1245	5.21093	4.63953	4.42046	4.54427	4.51136
147	3.39369	3.83091	3.7959	3.70186	4.19867	3.22597	3.55876
148	3.19468	3.62304	3.27222	3.7031	3.917	3.74213	3.327
149	3.64621	4.02095	3.93643	5.05307	5.22403	6.96361	3.4993
150	4.76422	5.10873	4.83662	5.24993	4.66872	5.24938	4.80059
151	4.18231	4.59293	5.06787	5.03412	4.05595	4.00309	4.09136
152	3.51142	4.11196	3.96364	3.88179	3.95342	3.51928	4.39492

Table 4: Raw Registration Results for the Kitti Data 0091 Sync Data Set

frame	FM-2D	FM-3D	ICP	PCA	FVR	FFVR	FVR-3D
0	5.00485	5.0019	4.7115	5.67917	5.0019	5.12378	4.93267
1	5.43414	5.2679	4.97513	6.41151	5.22299	4.8758	4.90824
2	4.11429	4.61588	4.19271	4.31113	3.91882	4.15126	3.93248
3	3.41672	4.16803	3.28763	3.74408	3.22165	3.55394	3.41776
4	3.57411	3.85516	3.77114	3.94937	4.1954	3.47752	3.95665
5	3.50072	3.91967	3.31783	3.86038	3.65917	3.55733	3.43173
6	4.22011	4.25314	3.78114	4.25145	4.16365	3.91009	3.98266
7	3.16275	3.51208	3.04116	3.66412	3.92726	3.29864	3.08615
8	3.6745	3.836	3.69354	3.96416	3.28107	3.44354	3.74999
9	3.10039	3.65648	3.20345	3.49397	3.22242	3.25219	3.16125
10	3.21554	3.62774	3.32444	3.99828	3.23302	3.25254	3.46564
11	3.1637	3.58248	3.15202	3.49058	3.96584	3.05276	3.11556
12	3.82644	3.70922	3.00418	4.18151	3.75446	3.70238	3.61637
13	3.7737	4.00155	4.00977	4.32486	3.67016	3.97514	3.55307
14	3.41898	4.16933	3.59905	4.99004	3.21301	3.26922	3.1637
15	4.21394	4.08534	3.86088	4.602	3.94243	4.16067	3.83943
16	3.59799	4.26614	3.60301	4.17828	4.2784	3.60941	3.3247
17	5.55465	4.67646	3.39171	3.49739	3.41196	3.32131	3.1906
18	3.199	4.3482	3.56135	3.5951	3.1056	3.42987	3.13189
19	3.41229	4.23935	3.66608	3.58931	3.57503	3.4189	3.62463
20	3.29667	4.52287	4.01539	3.8291	3.73426	3.52737	3.11041
21	3.63469	4.03453	3.42704	4.04629	3.2982	3.68214	3.32141
22	3.58652	3.68818	3.37191	4.92031	3.63745	3.64552	3.3857
23	3.33348	3.23145	2.9232	4.80813	2.87845	3.16698	2.78884
24	3.63381	3.95112	3.55154	3.86047	3.56061	3.50138	3.48576
25	3.04259	3.98857	3.39676	3.56293	2.89832	3.19252	2.91249
26	3.66532	4.06465	3.57607	3.37759	3.40333	3.3923	3.35991
27	3.13249	4.27803	3.13465	3.54021	3.00941	3.15457	2.99665
28	3.49016	4.00296	3.48415	4.32892	3.401	3.45577	3.3405
29	4.81162	5.09406	4.86448	5.14262	4.82946	4.9528	4.64927
30	4.07117	4.15837	3.46046	4.33341	3.72362	3.76722	3.76784
31	3.98787	4.7729	4.05183	4.0949	4.00877	3.97841	3.8186
32	3.61878	4.0315	3.2177	3.4446	3.75006	4.68381	3.44776
33	3.70715	4.74913	3.9582	4.21686	4.09541	4.00058	3.74117

34	3.93615	3.49539	3.55481	3.38074	3.48926	3.44021	3.46345
35	3.53687	3.7141	3.58347	3.90895	3.47497	3.51544	3.29011
36	3.42068	3.75051	3.30938	4.12879	3.22901	3.57267	3.28817
37	5.83645	6.14702	5.44147	6.92611	4.85349	6.17084	4.9809
38	5.97929	5.76935	5.51019	6.12277	6.34525	5.775	5.96661
39	3.30627	3.55059	3.22719	3.39504	3.77624	3.56898	3.06612
40	2.85261	3.30699	2.67552	3.60765	2.96334	2.75407	2.69803
41	3.07106	3.24623	3.036	3.28596	3.13709	3.24824	3.1542
42	3.41867	3.7591	3.42789	4.03174	3.4145	3.42004	3.44523
43	3.25149	3.36185	2.98163	3.61123	3.25572	3.31206	3.02484
44	3.47156	3.75967	3.2571	3.82642	3.45765	3.47057	3.21448
45	3.56684	3.32035	2.78609	3.25807	3.01029	3.06439	2.6403
46	3.28977	3.8091	3.25446	3.67857	3.61686	3.45697	3.11564
47	2.95585	3.82098	2.97154	3.40421	3.11703	3.12486	3.02259
48	3.32319	3.73478	3.33926	3.57004	3.38566	3.433	3.15465
49	4.06014	4.20657	3.60617	4.02614	3.99041	4.15816	3.7211
50	4.24892	4.60565	4.1858	4.41896	4.33069	4.30088	4.22067
51	3.87027	3.88248	3.96642	4.05629	3.67598	3.94778	3.61165
52	3.9833	4.15397	4.11853	4.23352	4.36791	4.12002	4.03257
53	3.63164	4.18805	3.57383	3.93749	3.73714	3.52392	3.60361
54	3.59688	3.79759	3.23921	3.62051	3.50173	3.37519	3.16046
55	3.56988	3.99163	3.90729	3.95005	3.73068	3.81267	3.26742
56	3.32024	3.70862	3.20863	3.91373	3.51051	3.95489	3.10194
57	3.30608	3.68827	3.18539	3.52717	3.31937	3.4648	3.20308
58	4.43295	4.33804	4.10953	4.84792	4.72742	4.81876	4.56579
59	3.84915	3.81623	3.28933	4.37609	3.42044	3.78258	3.20593
60	3.62177	4.13181	3.35925	4.18595	3.53054	4.04458	3.43571
61	3.93039	4.24481	3.71202	4.03706	3.77032	4.00525	3.62836
62	3.52424	4.36617	3.64928	4.10592	3.59797	3.74677	3.57696
63	3.97304	4.79775	3.69482	4.46283	4.1193	4.44803	3.78986
64	3.67106	4.56452	3.90776	4.47058	3.77064	3.93114	3.77609
65	3.90923	5.47468	4.09997	4.94089	4.06466	4.61296	3.91143
66	4.21214	4.65057	4.31785	4.67156	4.2663	4.4421	4.16627
67	5.97567	6.24818	6.02717	7.72283	5.50309	6.12162	5.43962
68	5.35099	6.66497	5.97472	12.6216	5.93127	5.06945	5.26248
69	11.3486	7.89206	7.30847	5.63815	4.9	4.97491	5.6586
70	4.32197	4.54557	4.67562	5.05783	4.31694	4.78464	4.38863

71	4.07528	5.64063	4.43659	4.61352	4.14245	5.18008	4.16371
72	4.46456	4.81589	4.6511	4.81854	4.49851	4.97249	4.2424
73	4.55309	5.23675	4.63117	5.05418	4.51299	4.51007	4.64838
74	4.64536	4.87945	4.8696	5.02949	4.50609	4.60891	3.95742
75	4.2125	4.68397	4.49376	5.63427	4.29976	5.10048	4.33684
76	5.13828	5.38959	5.00782	5.36193	4.75412	5.44244	5.27098
77	4.87728	5.12707	5.04832	5.45543	5.02263	5.98622	4.93526
78	5.29542	5.8692	5.50524	5.99178	5.24544	5.56144	5.74063
79	4.5766	5.07956	4.63712	5.32482	5.04325	4.68561	4.19143
80	5.01329	6.74842	4.99537	5.07022	4.77313	5.12707	5.01229
81	5.28865	5.75351	5.76877	5.26948	4.98958	5.59046	5.4625
82	4.38869	4.84561	4.23967	4.43906	4.1987	4.53267	4.02243
83	4.51552	4.86702	4.75165	5.14164	5.2848	4.3319	4.76999
84	5.35175	6.13492	5.65859	6.2308	5.46026	5.26433	8.57848
85	5.46128	5.80174	4.98217	6.33628	5.3856	5.175	4.95607
86	11.8381	12.2272	14.3936	12.9736	11.7652	12.31	11.1287
87	4.23072	4.55648	4.1011	4.76402	3.97364	4.11016	4.05887
88	5.40225	5.56398	5.2794	5.49561	5.13789	6.29596	5.10947
89	4.48134	5.19613	4.83774	5.51046	4.49784	4.53805	4.79338
90	4.75319	5.15463	4.44643	6.03052	4.84239	4.95468	4.30265
91	5.43137	6.20325	4.93725	5.77274	5.07939	5.62692	5.12627
92	5.12457	5.94651	5.62709	5.80918	5.23543	5.39969	5.35464
93	4.55194	4.88241	4.16909	4.80912	4.85463	4.47087	3.9941
94	4.65256	4.93298	4.71351	5.77068	4.74903	5.29733	4.46333
95	4.29916	4.9056	4.47528	5.20634	4.35102	4.10004	4.73519
96	3.88647	4.49685	5.64178	4.70975	4.84652	4.12947	4.23418
97	4.5137	8.56977	4.18354	4.151	4.20178	4.07706	4.10164
98	5.81548	6.41842	4.5457	17.3444	5.39011	10.314	4.63296
99	4.68134	4.81182	4.5185	4.99461	4.88649	5.14563	4.37193
100	4.39974	4.87162	4.48913	5.14446	4.05512	4.4263	4.14827
101	3.68434	4.48658	4.67433	5.19124	3.82354	4.04148	4.02943
102	4.33244	5.13666	4.08049	4.27113	3.95576	4.11818	4.14634
103	3.6819	4.54711	4.27279	4.84084	3.43371	4.51991	3.31485
104	3.67856	4.22635	4.12451	4.16006	3.80074	3.91724	3.87168
105	4.76343	5.33013	5.02456	6.24464	4.81245	4.67057	4.66662
106	5.09598	5.10652	5.20235	6.45523	4.90331	5.28412	4.79577
107	5.5994	6.61944	5.61208	6.90856	5.60883	5.78452	5.17816

108	3.65435	3.95705	3.4734	3.68195	3.27279	3.24202	3.34023
109	4.01461	4.58573	3.99929	3.9138	3.84984	3.92378	3.9431
110	3.44107	4.137	3.53955	4.1491	3.48684	3.43984	3.41627
111	4.34412	5.18118	4.44113	5.13564	4.81751	4.51309	4.0564
112	3.67282	4.17322	3.58781	3.89034	3.35472	3.71554	3.42848
113	3.95152	4.8465	3.85401	4.80613	4.03846	4.13138	3.70406
114	3.48737	3.97311	3.27092	4.35857	3.41176	3.48563	3.34306
115	3.61207	4.53	4.07986	3.97555	3.73044	3.55054	3.23795
116	4.25195	4.66539	4.18417	4.58951	4.04022	4.2134	4.03346
117	5.02751	5.50293	5.22519	6.16467	4.99263	4.99823	5.05244
118	5.15802	5.57055	5.09106	5.56001	5.45234	5.44848	5.01232
119	3.26027	4.20079	3.32033	4.25038	3.29237	3.3642	3.1415
120	4.08024	4.96826	4.54112	5.46368	4.37315	4.43892	4.50055
121	3.49636	3.53684	3.23489	4.8924	3.21738	3.67025	3.66407
122	3.49673	4.30167	3.39888	3.98361	3.48054	3.84261	3.19796
123	6.02552	6.0733	4.50851	7.82859	5.06075	4.88333	4.72308
124	2.92195	3.62749	3.56226	3.65988	2.93225	2.97534	2.58748
125	3.81224	3.904	3.18668	4.21148	3.3457	4.00916	3.07481
126	3.26438	3.79369	3.29035	4.14111	3.40163	3.74713	2.98511
127	3.21694	3.55977	3.1124	3.63203	2.67585	3.2667	2.96573
128	2.76307	3.52888	3.09037	4.12875	2.90141	3.4076	2.91343
129	3.61597	4.05425	4.01011	4.33822	3.57033	3.62079	3.69826
130	3.04134	3.38815	2.77074	3.22236	3.31978	3.08039	2.76177
131	3.68362	4.48141	3.87773	4.05783	3.57066	4.18135	3.57159
132	3.39095	3.69829	3.60921	3.96997	3.73122	3.11248	3.05111
133	3.93119	4.61504	4.80534	4.52598	4.17286	4.25813	4.22637
134	3.19314	3.21038	2.85522	4.57174	3.12941	2.98222	2.56568
135	4.03095	4.45306	4.53956	6.15058	3.80869	4.31846	3.73751
136	2.71153	3.72963	7.529	3.05953	2.57743	2.82775	2.65446
137	3.17668	4.17957	6.16508	4.5555	2.97065	4.17351	3.15207
138	3.4003	3.50171	4.25859	3.25779	3.02936	3.16184	3.17109
139	3.18316	3.48452	2.79368	3.48946	2.84319	3.16067	2.82762
140	3.41693	3.92732	3.38226	4.00201	3.42637	3.42744	3.43847
141	3.63903	3.78302	3.44687	3.74366	3.30808	3.25896	3.02527
142	3.13078	3.74419	3.89702	3.50446	3.15566	3.34983	2.9088
143	2.7946	3.29223	3.15432	3.69788	2.72407	2.96257	2.61235
144	3.19416	4.2584	3.25266	6.2989	3.01724	3.50895	3.03053

145	4.05248	4.58277	4.69175	4.27496	3.19324	4.53743	3.92092
146	3.778	3.66764	3.60862	3.68949	3.30137	3.39919	3.30175
147	4.93919	5.73965	4.55936	5.07293	4.6543	4.75977	4.45238
148	2.94668	2.81154	2.93724	3.29136	2.60011	3.15337	2.81534
149	3.46908	3.404	3.37186	4.38939	2.99772	3.12187	3.11236
150	3.31873	3.79637	3.51926	3.59004	3.47383	3.66403	3.44484
151	4.04503	4.20442	3.6957	4.10209	3.27197	4.02253	3.3282
152	4.2743	4.88471	5.10473	4.58523	4.66767	4.51669	4.12834
153	3.2124	3.60024	3.52763	3.559	3.03506	3.20348	2.99027
154	3.19448	3.56587	3.09516	3.59614	2.96885	3.22436	3.10724
155	3.03967	3.18109	3.38956	3.95232	2.87069	3.07654	3.13816
156	3.57376	3.19542	3.39996	3.79833	3.0098	3.25869	3.16282
157	4.16835	3.80348	3.68172	3.85147	3.49546	3.92229	3.55351
158	3.50937	3.65418	3.8397	3.78822	3.4362	3.32043	3.44502
159	4.0056	3.47812	4.0072	3.80319	3.96724	3.43332	3.6665
160	3.41695	3.91107	2.7683	3.25139	3.21716	3.21297	2.98089
161	2.74978	3.18964	3.55155	3.70214	2.65502	2.6672	2.76392
162	3.40045	4.47937	4.42619	4.70061	3.86928	4.43775	3.20147
163	2.91382	3.05303	3.36885	3.29832	2.76441	2.75843	2.72756
164	3.25272	3.51249	3.60637	3.55439	2.7955	3.10984	2.88675
165	3.1667	3.27969	3.30567	3.6064	2.9722	2.98842	3.00083
166	2.44187	2.77777	2.68114	3.17332	2.4554	2.66272	2.48561
167	3.36958	3.58932	3.88814	4.03199	3.35836	4.04957	3.53843
168	2.43386	2.53247	2.46568	2.50179	2.35284	2.62386	2.49684
169	2.39675	2.82383	2.69321	2.52664	2.51917	2.549	2.45171
170	2.55153	2.63348	2.51184	2.64487	2.43192	2.74816	2.3944
171	3.4075	3.68883	4.1143	3.76002	3.04652	3.46202	3.24756
172	3.42696	3.42161	3.75924	3.54874	3.42489	3.60709	3.41621
173	2.86852	2.99808	3.14878	3.47176	2.69618	3.34078	2.53727
174	3.29772	2.64688	2.80215	2.76095	2.91618	3.05837	2.89904
175	2.91855	2.86007	2.92073	3.01438	2.816	3.70904	2.91072
176	2.41733	2.64962	2.60233	2.72105	2.71828	2.51926	2.26832
177	2.51694	2.52941	2.22604	2.81865	2.6224	2.90948	2.30395
178	2.90312	3.17959	2.63338	3.97718	2.77407	3.45727	2.75202
179	2.56972	2.61426	3.09411	3.38273	2.76044	3.28523	2.66273
180	2.3209	2.59401	2.17164	2.91125	2.45643	2.76678	2.12649
181	2.0543	2.14066	1.98413	3.11636	2.11144	2.21963	1.9259

182	2.55484	3.45233	3.19929	4.11423	3.56102	3.6378	2.69863
183	2.73085	3.01295	2.91322	3.95944	2.9305	3.16277	2.85715
184	3.68822	2.76544	2.84572	2.85387	3.08259	2.72271	2.56411
185	2.72021	2.71466	2.97801	2.7014	2.58609	2.60251	2.29521
186	2.68568	2.96428	2.86313	3.09982	2.84095	2.75617	2.86326
187	2.35353	2.72181	2.76985	2.72415	2.90217	2.56909	2.6199
188	2.69255	2.97399	2.3974	2.83536	2.90531	2.98741	2.52845
189	2.22891	2.57521	2.25318	2.20608	2.25167	2.58241	2.05802
190	2.16474	4.53749	4.65106	3.03929	2.41595	3.36285	3.03585
191	3.42629	4.57073	3.58697	5.62087	3.20606	3.53375	3.74279
192	4.01669	4.56327	3.5455	3.58147	3.86089	4.34683	3.47813
193	4.40219	4.76162	3.36713	4.62921	3.53089	6.30133	4.04392
194	2.30656	2.53202	2.58808	2.42491	2.50441	2.37961	2.28381
195	2.59245	2.86331	2.8492	2.6476	2.91981	3.63333	2.47556
196	4.44985	3.6373	3.06498	3.40647	2.95478	4.37973	2.90513
197	2.34762	2.73805	2.52774	2.74902	2.73357	3.15007	2.45526
198	4.40309	3.89146	5.12404	5.07439	4.07612	4.87231	3.92991
199	2.53012	3.23184	2.77117	3.49625	2.67713	2.97891	2.72025
200	2.52029	3.06417	2.5282	3.18345	2.79539	2.58375	2.48052
201	2.61092	3.29433	2.52057	3.23077	3.17532	3.08921	2.34072
202	2.71316	3.03325	2.88408	2.92279	2.88991	3.28295	2.62933
203	3.40917	3.72945	3.28093	3.6089	3.20374	5.49726	2.94929
204	3.77966	4.77166	4.07077	5.84204	5.16322	8.50978	3.90268
205	3.34705	4.87339	3.97089	4.20936	3.67728	3.53059	3.25088
206	3.62895	3.70581	3.55117	3.79483	3.50659	3.32417	3.06446
207	3.35016	3.3929	3.18148	3.60542	3.78113	3.5498	3.01844
208	3.0266	2.90523	3.18112	2.73354	2.65042	2.77415	2.6969
209	4.03811	4.43662	3.74183	4.03978	3.90723	4.12793	3.74195
210	3.59641	3.95516	4.0126	3.91077	3.30751	3.46288	3.25047
211	4.18802	5.29845	4.49584	4.38506	4.75585	3.79902	3.91319
212	3.8333	3.70455	3.35404	3.66214	3.07243	3.2142	3.21716
213	3.85978	4.61127	3.77473	4.43964	3.71473	4.23285	3.44428
214	3.24502	3.74545	4.18789	3.71319	3.15843	3.2928	3.41027
215	4.09587	3.75489	3.25114	3.2373	3.37009	3.87925	3.34393
216	3.72037	3.85715	3.78922	3.53886	3.22108	3.68122	3.11929
217	3.38603	3.67747	3.80521	3.84659	3.2421	3.99527	3.04778
218	3.1202	3.49775	2.95666	3.58397	2.98965	2.94339	2.88912

219	4.56381	4.50734	6.62011	4.726	4.25942	5.02578	6.17292
220	3.18821	3.77245	3.48348	3.90707	3.60925	3.2151	3.19145
221	3.99883	4.45927	3.80724	4.69186	3.57588	3.78187	3.51999
222	3.57519	3.77278	3.4203	4.29806	3.40939	3.74456	3.38404
223	3.32028	3.8528	4.09546	3.76497	3.7228	3.37395	3.5609
224	3.78145	4.11419	3.94574	5.01378	3.56038	6.26791	3.63541
225	3.03777	3.5778	3.24372	3.75736	2.96337	3.12851	3.23965
226	3.45558	3.75834	3.74403	3.66998	3.33091	3.33118	3.26774
227	3.69906	4.08656	4.34072	4.33719	3.77696	4.01167	4.10526
228	3.13793	3.24755	3.01138	3.37146	2.89137	3.25438	2.9392
229	7.09597	4.09981	4.10829	6.1315	3.8483	3.86742	3.56317
230	3.10914	3.58066	3.02155	3.89389	3.12483	3.56851	3.14823
231	3.09529	3.39884	3.53789	3.82868	3.57218	4.27537	3.24771
232	3.45516	3.78248	3.53533	4.21555	3.59056	4.395	3.02859
233	3.4815	3.65904	3.77968	3.73087	3.86446	3.88689	3.35505
234	4.1164	4.92667	3.36953	5.17984	4.19155	4.98661	3.41335
235	5.08837	4.73776	4.74497	4.19526	4.58725	4.60006	4.39948
236	4.05541	4.43294	4.85383	4.56597	3.8877	4.63452	3.90744
237	4.44443	4.98097	4.3012	5.74554	4.81394	8.00877	4.76555
238	2.72181	3.29646	2.71701	3.20724	3.54467	4.15654	2.81573
239	2.90485	3.13197	2.93776	3.11353	2.79684	4.94114	2.59874
240	2.40343	2.80439	2.41507	3.11714	2.42383	5.46066	2.6078
241	2.83476	3.55855	2.94847	3.09867	2.91874	3.14857	2.85298
242	6.45415	8.35568	3.50812	8.31764	4.79565	7.57638	3.97166
243	3.52674	3.34147	2.87755	3.45744	3.29137	5.31248	2.79781
244	3.15582	3.43671	3.19544	3.92996	3.75482	7.44558	3.56783
245	3.33359	3.42533	3.22651	3.59423	3.10485	4.68613	3.09844
246	2.4204	2.87594	2.59636	3.16674	2.47453	5.55251	2.32887
247	2.88454	3.39247	2.75366	3.37697	2.99418	3.46322	2.70922
248	2.87865	3.27035	2.83424	3.21156	2.82389	4.68904	2.82021
249	2.67525	3.47725	2.83176	3.48855	3.00441	5.85276	3.10927
250	3.82336	4.00367	3.57995	4.83173	3.91753	4.07383	4.12507
251	3.35434	3.93625	3.25724	3.52865	3.0705	3.26576	2.94634
252	2.89594	3.46911	3.46201	3.94753	2.90565	4.1527	3.06379
253	3.47442	3.68415	3.82222	3.87434	3.57488	4.26677	3.41697
254	2.39703	3.26071	2.27177	3.12181	2.47376	2.26369	2.37281
255	2.39994	2.97115	2.37564	2.91386	2.27335	2.48027	2.36184

256	3.92967	4.62058	3.80029	4.86475	3.51647	4.15804	3.72521
257	2.99326	3.42553	2.78857	3.14493	3.3442	2.6911	2.74549
258	4.15387	4.49118	3.68621	4.11567	3.70602	4.17288	3.57659
259	2.70666	3.70577	3.05072	3.496	2.75353	2.92383	2.78489
260	4.1134	5.00079	4.03666	4.47452	4.23152	4.02974	3.82437
261	4.1059	4.85277	3.95744	4.80219	4.46257	4.03752	3.81976
262	3.98705	4.70845	4.08335	4.69487	4.08226	4.90397	3.56226
263	3.48948	4.2017	3.86334	4.48353	3.91655	4.01391	3.47209
264	3.90678	3.90935	3.71135	4.2554	3.48823	3.06997	3.42608
265	3.15986	3.64264	2.795	3.62575	2.89413	2.83478	2.696
266	3.79554	4.57789	3.35559	4.92588	3.60667	3.8263	3.78942
267	3.74803	4.03749	3.75157	3.70229	3.47887	4.26427	3.44761
268	4.12202	4.46664	4.15187	4.37116	3.75606	4.19598	3.6268
269	3.2479	3.98234	3.09536	3.78317	3.44366	3.33481	3.40248
270	3.24415	3.94304	3.44029	3.83953	3.76078	3.59023	3.39186
271	4.38595	10.4526	4.64741	6.35722	5.29456	8.66745	6.38956
272	4.90648	5.11787	5.21003	5.12955	4.82614	4.97733	5.17351
273	3.09608	3.93091	3.57688	3.82466	3.47506	3.24243	3.25098
274	3.45556	4.73421	4.05561	5.34716	4.85468	5.18324	3.52815
275	3.52978	3.95499	3.34156	3.69626	3.57936	3.42646	3.2658
276	4.06521	5.0047	3.6617	4.40784	4.5393	4.27481	3.72354
277	4.42981	8.17662	4.66046	4.61002	4.81947	4.62574	4.60103
278	4.1519	4.67285	5.35102	5.02718	4.10984	4.10886	3.89142
279	2.9303	3.20511	2.98964	3.34709	3.56927	3.0207	2.72466
280	3.38173	3.67708	3.51528	7.50858	2.94607	2.97077	2.99228
281	5.58969	4.60253	5.42026	4.528	4.14218	4.20598	5.27312
282	4.47314	4.59039	4.22879	4.63908	4.58359	4.75619	4.01316
283	4.59858	5.58506	4.32344	5.04865	5.09792	4.98039	4.47696
284	4.49797	4.64162	4.10034	4.2621	5.03054	4.62273	3.73364
285	7.52465	3.96634	6.87778	4.38095	3.52435	3.21512	3.93421
286	3.32958	3.56772	3.72626	3.67714	3.12803	3.84644	3.33215
287	2.96149	3.77846	3.41174	3.33438	3.0518	2.91015	2.91359
288	3.03063	3.78128	3.37861	3.73515	3.10342	3.31365	3.12193
289	3.33854	4.35554	3.37583	3.96765	3.30343	3.57937	3.33144
290	8.67617	13.3958	12.0464	16.8895	10.018	17.8228	7.10392
291	4.33479	4.43694	4.0287	4.1965	3.88562	4.23623	4.09689
292	2.88436	3.35993	2.6729	3.79084	2.95591	2.89228	2.71611

293	5.07519	6.51603	3.50859	6.03251	6.71327	7.68495	4.7103
294	3.01345	3.66544	3.18313	3.91743	3.01781	3.93846	3.22194
295	2.82193	3.75171	3.06011	3.47713	3.12527	3.48288	3.12485
296	4.97797	5.45818	4.913	4.31655	4.81491	4.47147	4.06848
297	5.66592	5.95511	5.2399	5.68038	5.35648	5.86387	5.43884
298	5.19591	5.75064	5.04119	5.9895	4.83064	5.32445	5.13688
299	3.64014	3.91868	3.82493	6.58644	3.48385	3.50916	5.84269
300	3.60088	3.68564	3.35186	3.81485	3.7981	3.45893	3.02519
301	3.44476	3.53161	3.29235	3.80928	3.55558	3.20056	3.16757
302	5.40783	8.85599	3.37503	4.02039	3.48063	3.66774	4.03122
303	5.77693	5.03182	5.30613	5.46181	5.36098	5.69201	5.11314
304	3.00804	3.3848	2.73145	3.37497	2.74569	2.98305	2.96035
305	3.4368	4.25928	3.45734	4.37184	3.50409	3.82707	3.20681
306	3.51475	4.20326	4.33986	4.26525	3.18191	3.74994	3.46892
307	3.5457	4.10996	3.3034	4.36112	3.63258	3.87524	3.40321
308	4.03428	4.31934	5.58482	4.44628	3.73774	4.17437	3.22738
309	4.08614	4.82366	3.81736	4.97214	4.02442	4.91573	4.13805
310	4.15269	4.57812	3.66639	5.06945	3.63668	3.94205	3.76546
311	3.26882	4.60274	3.10103	3.9585	3.19926	3.77642	3.10025
312	3.62962	4.52396	4.013	4.74707	3.47596	3.5677	3.27388
313	4.78607	5.147	11.034	4.98928	4.987	5.13314	5.00444
314	4.64362	4.75487	5.55358	4.45426	4.50437	5.5976	4.94444
315	2.95974	3.78281	3.1263	3.3861	3.05839	3.23136	2.98083
316	3.04398	3.4873	2.86099	3.48879	3.15515	2.87627	2.94933
317	3.79495	4.2277	4.15264	4.5325	3.84528	3.75945	3.59402
318	3.24746	3.75143	3.37219	4.42923	3.24186	3.41706	3.45891
319	3.56262	4.24598	3.67705	4.45469	3.62749	3.73241	3.80568
320	3.20796	3.91119	3.32113	4.23792	3.25389	3.67544	3.25945
321	4.45568	4.60669	4.34205	4.51941	3.85947	4.10907	3.91189
322	3.88716	5.10256	4.42498	5.15154	3.86214	4.04929	3.8542
323	3.5096	4.0882	3.91287	4.36707	4.0452	4.02187	3.89357
324	5.50489	6.74309	7.14927	6.61417	6.16291	5.58876	5.38347
325	5.92599	15.612	5.38855	11.0733	5.07644	7.35732	4.96638
326	3.1742	3.41983	3.09838	3.69405	3.21261	3.05656	3.112
327	3.55076	4.07304	3.43161	3.61971	3.48372	3.56553	3.61311
328	4.08087	3.97832	5.64224	4.58088	3.65291	4.00745	3.76458
329	3.21716	3.72544	3.54944	3.68359	3.17223	3.43494	3.24356

330	3.72955	4.12957	3.96104	4.16091	4.20337	3.81993	3.74006
331	4.18397	5.64337	4.39652	4.76128	5.1053	4.38983	4.82011
332	3.88574	4.14366	3.7804	4.17304	3.97035	3.9115	3.74032
333	3.61279	4.06851	3.42065	4.05799	4.18872	3.7448	3.4911
334	3.94107	4.31843	3.8976	4.48988	4.34724	3.94595	3.9685
335	3.55229	4.16246	3.76144	4.8143	3.80401	3.64794	3.55371
336	3.844	4.6205	3.84051	4.42027	3.82096	4.0358	3.82765
337	3.17871	3.97574	3.17556	4.03232	3.69685	3.27607	3.40376
338	3.42166	4.17154	3.39644	3.97724	3.47646	3.52654	3.48778

Table 5: Raw Registration Results for the Kitti Data 0095 Sync Data Set

frame	FM-2D	FM-3D	ICP	PCA	FVR	FFVR	FVR-3D
0	5.43423	6.08793	5.38247	6.04365	5.35404	7.43814	4.9425
1	3.42981	4.19521	3.88897	5.02692	3.58138	3.54829	3.56374
2	3.81079	4.97923	4.74385	5.01112	3.59281	4.49448	3.71785
3	6.34189	6.58745	5.93999	6.38457	5.30856	5.65099	5.1105
4	4.85831	5.44152	6.04256	6.48229	4.5672	5.59023	5.14698
5	4.3046	4.9309	4.98855	5.2134	4.29284	3.94573	4.13845
6	4.18209	4.50911	4.64697	5.15108	3.66069	3.94232	3.71477
7	5.43972	5.64905	5.28751	7.14136	4.68342	4.99908	4.83119
8	5.84379	7.06821	5.19333	6.32078	5.64349	5.42695	5.36126
9	4.2061	5.10372	6.16801	5.11627	3.94753	4.2588	4.64448
10	5.55289	6.18685	4.73027	5.75389	4.25775	5.25418	4.22131
11	4.86052	5.07738	5.76558	5.29931	4.36001	5.36121	5.35898
12	4.43348	5.3348	4.92324	4.72439	5.57875	4.16821	4.19374
13	4.73666	5.05276	7.55872	4.82483	4.50281	5.07115	4.36414
14	4.93742	6.2474	5.45532	6.74771	4.41431	6.29736	4.7379
15	3.87831	5.28902	4.35009	6.98847	3.88023	3.98074	3.84478
16	3.90265	4.39125	4.07808	4.58719	3.64442	4.27102	3.60724
17	4.113	5.39636	4.80434	6.24898	3.91494	3.92091	3.96974
18	3.71108	4.6244	3.63047	4.63619	3.44639	3.75011	3.55899
19	4.24895	4.55291	4.4018	5.34046	3.91396	4.61569	3.93506
20	4.32863	6.58709	7.15534	5.73878	4.9673	9.25479	4.63591
21	4.58515	5.92412	4.54453	6.61746	4.69613	5.08814	4.85366

22	6.00347	6.50191	6.3624	6.01132	5.06207	5.32833	5.02467
23	6.78312	5.7679	5.76984	6.15646	4.18563	5.10059	4.22117
24	4.15093	4.83111	4.07263	4.37794	3.9183	4.14817	4.03804
25	4.08559	4.49268	4.44348	4.66311	4.22899	4.63474	4.26246
26	3.84831	4.98009	5.01961	6.03842	3.91331	4.1143	3.87702
27	5.20843	5.37292	4.86584	6.255	5.17509	4.73258	4.32658
28	5.9557	5.9413	5.89949	5.50881	5.22029	5.58884	5.7483
29	3.56741	4.40805	3.53222	4.11249	3.68761	3.84247	3.27754
30	5.22839	4.71303	4.54874	4.86471	4.86899	5.9955	4.56219
31	3.64898	4.57162	4.24013	5.08079	4.46756	3.79251	3.65457
32	3.86967	5.02876	4.23604	6.41181	3.44645	4.02318	3.68003
33	8.19383	9.3015	8.3409	11.7452	7.71351	8.13085	7.60566
34	3.37415	4.08446	4.28401	7.4956	3.63387	4.14514	4.14964
35	4.3791	11.7731	3.65775	4.77462	3.72174	5.43906	3.49714
36	4.0391	5.48134	4.33076	4.4859	4.44354	4.33344	3.82211
37	4.14969	4.80405	4.72306	5.25045	3.49153	4.07498	4.00373
38	4.21165	5.25967	4.34003	6.65921	4.93069	4.50026	4.217
39	4.53606	5.76098	4.37665	5.6477	4.5093	4.76602	4.05008
40	4.54774	5.79278	4.47947	5.41145	4.71564	6.5951	5.05008
41	5.01593	5.17575	5.32461	5.49426	4.07797	4.80611	4.24358
42	4.19132	5.51134	5.98473	5.26295	4.1883	4.6283	4.26842
43	3.72886	4.39368	3.72424	4.08057	3.45769	3.86987	3.41668
44	3.66258	5.41558	3.89837	4.74512	3.53969	4.13246	3.61362
45	4.94472	6.87994	5.09074	7.01708	5.02328	4.76433	4.88129
46	3.57015	4.64003	3.55801	5.09384	3.36847	4.89304	3.55226
47	3.63731	4.2307	3.68407	5.9735	3.42209	3.80628	3.21726
48	6.89189	7.711	5.46648	7.85652	5.66893	9.48523	4.27112
49	3.99795	4.11277	4.34696	3.6127	3.70881	7.03421	3.6714
50	3.65422	28.2928	3.92001	4.94188	3.18724	6.04074	3.54628
51	3.35515	4.10065	3.27378	6.42678	3.07084	4.4202	2.97947
52	5.51358	9.08614	4.52561	14.1184	8.49562	7.78181	4.34026
53	5.47332	6.44556	5.92445	7.93438	5.2458	7.42735	5.93493
54	5.31713	6.94586	5.57533	10.1753	5.37719	6.81852	5.19899
55	4.20763	6.18624	4.12094	4.7127	4.62417	7.51821	4.20916
56	4.91278	5.80023	4.2334	5.31235	4.42915	4.48252	4.42953
57	5.39292	5.60429	6.05146	5.68316	4.66152	6.87648	4.90619
58	3.31263	4.54896	4.4802	4.85352	3.28501	3.88984	3.74272

59	4.24767	11.0006	4.06722	4.8616	7.44517	4.9897	4.18897
60	3.36827	4.38556	5.09159	3.63405	3.41264	3.48433	3.24556
61	3.61415	3.829	3.55806	4.00282	3.20391	3.53743	3.48835
62	3.18958	4.35038	3.21917	4.93392	3.1717	3.37672	3.05948
63	3.25942	4.03282	30.7858	4.80478	4.04714	3.61183	3.43327
64	3.10209	4.43715	15.2799	4.08228	3.31324	3.63273	3.43573
65	22.7597	7.20144	8.41801	23.6205	5.52032	13.1187	4.95146
66	6.78954	7.13173	5.81654	6.88018	5.94889	7.13175	5.2353
67	3.33785	3.97712	3.77249	3.77282	3.31585	3.94538	3.04631
68	3.43266	4.33149	3.52935	3.88085	3.28305	3.67798	3.45716
69	3.94889	4.89438	4.36377	4.42781	3.6548	3.97562	3.96653
70	4.02169	4.83407	4.37993	6.2027	3.80786	4.04592	3.50129
71	4.53186	5.70034	4.64586	6.4874	4.71246	4.34389	4.28935
72	4.20961	4.86476	4.96212	6.276	4.03116	4.50234	3.88438
73	4.62943	6.34424	5.06506	5.78295	4.32506	4.68349	4.35909
74	4.56452	6.42254	4.41444	4.63236	5.18614	4.34915	4.46467
75	4.87482	6.13771	4.82962	5.39782	5.31767	5.69661	5.35215
76	3.77001	4.70573	3.46651	4.80242	3.96382	3.83812	3.74031
77	3.62495	4.21076	3.93766	4.09832	3.32383	3.33735	3.05219
78	3.41337	5.37304	3.47478	5.04455	3.67969	3.26322	3.16971
79	4.68038	5.53082	5.3209	6.10263	4.7937	4.82364	4.85973
80	5.27954	5.6655	5.64418	7.52064	5.3943	5.56888	4.82512
81	4.18658	6.31309	4.62009	6.27021	4.8192	4.04942	4.2544
82	4.46971	4.99676	4.86938	6.68389	3.96569	4.42851	3.87057
83	5.06497	6.27032	4.62224	5.91748	5.45686	4.7017	5.0764
84	5.04779	5.65724	5.80525	8.9618	4.55165	5.01618	4.77258
85	5.31099	6.23293	6.22143	7.23446	6.83254	5.11125	5.61759
86	8.43877	8.60727	8.39237	8.75244	7.71386	7.98915	8.06142
87	5.51831	7.14276	6.89643	5.61014	5.64508	5.49775	5.32141
88	5.36663	5.76337	4.36648	6.05069	4.45712	4.88517	4.02604
89	2.91783	3.99308	3.3793	3.70492	2.91275	2.93035	2.96436
90	3.25593	5.10503	3.78039	4.44722	3.62224	3.768	3.29729
91	4.00544	4.63035	3.81501	4.72873	3.48112	3.72181	3.08638
92	3.4649	4.62224	3.48129	4.69547	3.22049	3.6473	3.78626
93	4.97905	5.61332	5.07067	5.70691	4.49345	5.18197	4.6221
94	4.73003	5.63875	4.62316	6.1569	4.24276	5.036	4.79736
95	5.87791	6.99212	5.65806	7.95671	5.92723	6.4195	5.10043

96	5.28296	6.04321	4.47774	14.0549	4.52139	6.34547	4.77231
97	6.37912	7.29549	19.526	7.33911	5.82798	7.21774	5.33646
98	9.50808	6.22936	6.93986	8.32254	11.0804	6.74276	5.9436
99	50.6843	140.216	4.61824	151.692	8.99223	94.8515	4.33469
100	3.28505	4.25822	4.50613	3.70678	3.16058	3.33215	3.02149
101	4.58434	5.51398	4.14259	5.08152	3.76719	5.44075	3.69478
102	4.81577	5.85234	4.70787	6.01167	4.73437	5.27826	4.44074
103	3.89858	4.61075	5.13695	4.47959	5.33562	4.5223	3.75656
104	4.84411	5.53879	4.67248	4.91163	4.59301	5.78596	4.34511
105	6.34137	5.85071	6.66894	6.98114	5.96665	5.37365	4.86946
106	4.92105	5.94245	3.85048	9.06099	5.6718	5.45668	4.23577
107	8.96389	15.8032	5.23241	14.9455	5.58839	10.7824	4.89142
108	4.5153	4.9588	3.73468	4.34691	4.49439	4.82667	3.83463
109	5.52334	8.15747	5.14054	14.4577	5.18269	4.77753	5.44809
110	12.0597	10.0818	10.5603	10.1123	9.85544	12.1989	6.2646
111	3.91996	4.54242	3.86269	4.56925	4.1855	3.97771	3.51793
112	4.13102	5.27468	4.39362	4.40568	4.50162	4.10085	4.24279
113	3.31942	4.19157	3.24234	3.76594	3.36337	3.33705	3.11364
114	2.75232	3.20472	2.97433	3.20501	2.48317	2.83077	2.54944
115	3.46385	3.93964	2.79842	3.88611	2.75159	3.36274	2.95109
116	4.86582	4.94669	34.792	12.338	4.09804	10.792	6.53615
117	3.58496	4.53466	3.39768	4.49176	3.30818	3.96897	3.47402
118	3.7055	4.52562	5.26904	7.17647	3.51289	4.2316	3.69455
119	4.73334	5.98645	5.18981	6.82745	4.32605	4.82334	4.31281
120	4.77793	6.07063	5.68854	6.86167	4.95191	4.92315	4.71261
121	5.59141	5.3967	4.92783	7.16937	4.73442	5.7318	5.02882
122	5.39704	5.46511	5.02006	6.94961	4.44726	5.21385	5.30833
123	5.50918	6.27431	5.74176	6.04537	5.18576	6.20311	5.06744
124	5.15544	5.92612	5.41769	6.13495	5.46733	6.14163	5.13052
125	5.49904	7.07938	5.30704	5.69656	6.37686	6.04796	5.2311
126	5.8027	6.17614	6.3425	6.88505	6.20927	5.66171	5.94354
127	4.04889	5.49696	3.82297	8.47835	3.98527	4.79174	3.54556
128	4.89755	5.79829	4.66657	7.79551	4.42146	5.20648	4.80729
129	4.81029	6.09439	5.1011	5.59	4.97688	5.4182	4.31623
130	5.37581	6.36695	5.89398	5.7247	5.34741	5.27973	5.19191
131	11.2919	12.5987	9.24748	10.7515	10.1981	10.8801	9.26246
132	3.71152	4.73026	3.88105	3.76622	4.63661	4.24608	3.73982

133	4.41089	4.34477	3.97712	4.72269	3.60332	3.86494	3.48584
134	3.27107	4.05679	2.75852	4.38104	3.51114	2.95739	2.74249
135	3.80369	4.90617	3.54058	5.22196	3.58411	4.15817	3.48715
136	2.72574	3.49677	3.19889	3.37519	3.47866	2.81927	2.69796
137	3.81852	4.8033	3.88345	5.67611	3.53923	4.06996	3.55182
138	3.13873	4.09773	3.39448	4.09561	3.95076	3.58776	2.81968
139	3.65874	5.07716	4.56145	4.96297	3.80858	3.8706	3.76909
140	5.32732	5.09942	4.6804	6.25124	4.88363	5.22501	5.34364
141	3.53846	4.22292	3.32732	4.24826	3.51365	3.89563	3.35858
142	3.11985	3.89509	2.95606	3.72733	2.98626	3.36325	3.02323
143	8.61114	6.64598	6.98954	6.99344	6.41149	8.14974	5.71742
144	4.09889	5.36618	3.88588	5.39522	4.97582	4.93951	3.6762
145	3.06951	3.23145	2.80295	3.14172	3.19184	2.93143	2.63874
146	3.08685	4.3186	3.85212	4.54538	3.31812	3.47514	3.32691
147	3.20668	3.76939	2.8145	3.5326	3.00728	3.43392	3.11358
148	3.34564	3.84474	3.11756	4.03039	3.35727	3.3886	3.22534
149	4.33326	5.70974	5.17459	5.03115	4.36402	4.09076	4.26211
150	3.1652	3.40401	3.29483	3.53371	2.98431	3.4701	3.04253
151	3.59594	4.47633	3.37195	3.80939	3.41032	3.89995	3.46581
152	3.61115	4.42258	3.58598	4.13163	3.41448	3.76548	3.14034
153	5.11283	4.00782	3.78705	5.07065	3.84818	4.30203	4.00026
154	3.18019	3.74325	3.07677	4.387	3.13803	3.27266	2.76498
155	4.38422	4.59817	3.93128	5.02416	3.73487	3.98006	3.69745
156	3.08286	3.61557	3.40002	3.51277	3.12491	3.39014	3.27943
157	3.14916	3.99207	3.52765	3.96824	3.22188	3.21123	3.06629
158	3.37309	3.9998	3.91049	4.31711	3.24116	3.76654	3.45254
159	3.65323	4.72275	4.03871	5.32386	3.98509	3.79201	4.64699
160	3.83135	4.2981	3.41917	4.8009	3.5278	4.58467	3.19064
161	3.5581	4.91023	3.66662	4.86623	3.87828	5.04214	3.77273
162	5.381	7.79626	6.18394	7.98009	4.32359	4.31493	4.13293
163	4.6445	6.18853	6.22502	4.94261	4.36426	6.48314	4.47426
164	16.8333	13.3896	7.14738	19.1476	17.9148	11.3635	6.66942
165	3.04422	3.81517	3.09472	4.92302	3.08113	10.237	2.70174
166	3.59893	4.22579	3.2272	5.4293	3.3594	5.45064	3.20908
167	5.42652	5.54968	5.02884	6.22849	5.06191	5.94756	5.48748
168	3.17751	3.75976	2.95851	3.56329	3.03861	4.11478	2.76754
169	3.02897	4.2004	2.90964	3.89138	2.95267	3.35364	3.0636

170	3.69127	5.04237	4.12899	4.53297	3.67853	4.35054	3.55111
171	3.20105	4.87108	3.42439	4.32272	3.01719	3.3779	3.09677
172	4.06649	5.64661	4.50997	6.73729	4.00926	5.3552	4.24944
173	5.05848	6.0246	4.593	8.72291	5.13837	6.17892	4.36082
174	3.60941	5.40533	3.69393	7.26564	3.93116	4.63875	3.90789
175	4.28322	5.55452	4.75228	5.45525	4.69872	6.78983	4.60808
176	5.75665	6.17827	5.63688	4.91059	6.94766	4.77756	4.60091
177	3.85277	5.5804	4.35197	4.11714	4.34698	4.28546	3.99868
178	7.88542	7.89629	7.44016	8.75289	7.49365	8.59979	7.17433
179	7.86588	8.04772	7.03181	6.66365	6.12076	5.9505	6.5447
180	3.88512	5.3442	4.49447	6.75471	4.29161	4.27946	3.82277
181	3.44161	4.61082	3.7889	8.02415	3.56635	4.05224	3.11168
182	3.68398	5.48035	3.48266	8.56245	4.56977	5.08072	3.41306
183	3.51545	4.3007	2.94661	5.28807	3.14982	3.58663	2.99074
184	3.44783	4.6403	3.83059	4.24438	4.36245	5.50693	3.63895
185	3.82791	5.0664	3.79466	3.99499	3.74172	4.2022	3.46937
186	5.84241	6.51334	5.93765	6.0322	5.63589	6.93726	5.43066
187	3.94875	4.77695	3.55509	5.32994	4.06303	6.22337	3.58349
188	3.51406	4.79355	3.27766	5.71609	4.0513	3.41917	3.28288
189	3.49498	4.45445	4.36593	4.5327	3.42311	4.72387	3.28108
190	3.75298	4.58268	3.38346	5.74887	3.76623	4.82662	2.91565
191	3.24545	4.57357	3.26585	3.25451	3.59753	3.54121	3.40742
192	3.46573	4.3048	3.71713	4.25017	3.39356	4.81507	3.12015
193	3.83547	5.42741	3.76526	4.7283	5.14369	6.79374	3.72842
194	3.87204	4.36843	3.92786	4.80606	3.86017	5.29878	3.64682
195	3.77075	4.28061	4.15348	4.20689	3.97955	5.35164	3.79049
196	3.4165	4.51321	3.34101	3.94383	3.44044	4.31639	3.4262
197	3.22955	3.77774	3.51629	3.44141	3.34939	4.01221	3.16989
198	3.49237	3.98684	3.4511	3.91837	4.34916	4.49521	3.4018
199	4.22214	4.82045	5.00328	5.31959	4.37439	4.92748	4.48049
200	3.10789	3.38916	3.0564	3.89385	2.86385	4.02174	2.73496
201	3.75096	4.78247	3.00041	4.67121	4.12048	3.49287	3.09498
202	4.47014	5.24496	6.17467	6.95651	4.12132	4.52085	4.67997
203	4.88499	6.11041	4.31239	7.07012	4.6144	5.07955	4.17784
204	8.48296	8.20718	7.73286	8.06328	7.88874	9.63042	8.01448
205	3.64337	4.31303	3.65767	3.75077	3.4109	3.75718	3.27398
206	4.43781	4.97657	4.64269	4.98585	4.37864	4.41992	3.97353

207	3.67849	4.27072	3.79322	5.04933	3.51843	4.04013	3.46035
208	4.17175	4.48832	5.55289	6.84069	3.77149	4.87044	4.07615
209	4.9492	4.98177	3.9415	5.62996	3.53592	4.17509	3.53598
210	3.31935	4.03563	3.26209	4.41026	3.94075	3.61143	3.18381
211	3.31093	4.62679	2.84701	4.33979	3.97397	3.57357	3.05267
212	3.2813	4.41188	3.56946	4.21574	3.24477	3.48539	3.10832
213	3.10262	3.92945	2.75615	3.61434	3.07191	3.63834	2.94565
214	3.37872	4.18333	3.41126	4.01925	3.37479	3.50429	3.34179
215	4.22139	4.89127	3.43518	5.13878	3.98327	5.05531	3.41098
216	3.24872	4.48782	3.16615	3.98193	3.53008	3.3798	3.30259
217	5.66791	6.47581	5.17333	5.70885	5.88691	6.5054	4.93811
218	3.53419	4.32778	4.18116	5.11441	3.28179	4.0735	3.64736
219	4.38274	5.78876	4.42295	6.26046	4.53902	5.71346	4.39931
220	3.61202	4.57268	3.64502	5.35118	5.31097	4.97778	3.67675
221	3.77497	5.35705	4.13487	4.32541	4.11416	5.8068	3.6428
222	4.31038	6.081	4.2563	4.90829	3.93663	4.33204	3.96373
223	3.2611	4.42013	3.03161	4.19678	3.43708	3.83699	3.00048
224	5.20666	5.69002	4.35855	5.46194	4.66817	5.09264	3.6747
225	5.17108	7.4761	8.10348	9.39929	4.66567	8.2154	4.00886
226	4.32636	5.18346	4.56494	5.28657	4.20371	4.32452	4.2979
227	4.21172	4.68406	3.68712	4.62683	3.75947	4.94708	3.64986
228	3.84554	4.83872	3.82745	4.51953	4.4128	4.93826	3.41543
229	4.16184	4.92049	3.96081	5.47665	5.4015	4.83109	4.0894
230	4.53374	5.01963	3.96798	5.66743	4.05212	4.38051	3.76592
231	3.96669	4.86964	3.85298	5.07973	3.98703	7.75515	3.46579
232	4.97922	6.18938	5.53798	5.81715	5.9932	6.00872	5.11271
233	7.91914	13.5419	5.37523	12.4601	4.80783	5.8529	4.83708
234	7.61309	7.9726	4.91255	9.11744	5.62953	14.098	4.9676
235	4.90192	4.80445	4.12327	8.03817	3.79745	4.56842	4.29616
236	4.0959	5.45236	5.81027	7.28377	4.30579	4.09728	3.95822
237	3.73288	5.98745	4.40776	4.45196	5.57658	4.95201	3.62309
238	4.94311	8.03303	4.76994	7.33901	7.82623	7.33587	4.24371
239	5.21782	5.99986	5.26123	7.67865	4.82255	5.7818	4.61491
240	4.21032	5.76969	3.92185	6.53909	4.88504	4.09557	4.10373
241	4.23858	5.4016	4.76223	5.30883	3.81371	4.80885	3.98774
242	3.9485	5.31801	3.82784	4.67198	3.66782	4.06979	3.59003
243	8.02138	5.87105	5.49511	4.69246	7.35101	31.26	4.35261

244	4.54167	6.43049	5.03855	9.01523	6.44867	4.91828	4.8721
245	5.06591	6.41315	4.50743	5.55497	5.20945	5.85545	4.35699
246	3.50887	4.69139	4.05824	4.49429	3.36673	3.97965	3.79768
247	4.53189	8.03888	4.7423	8.83063	5.25229	26.1709	4.14579
248	3.63672	4.89897	3.68736	6.10109	3.69355	5.01788	3.98666
249	3.73763	4.88122	3.89718	6.37881	3.68743	3.97476	3.62878
250	4.5324	5.11474	5.34015	6.09643	4.01771	5.0122	4.38388
251	4.87734	7.10393	4.84461	6.73753	5.43706	5.48534	4.58332
252	4.38484	6.21899	4.90492	5.80828	4.70464	10.9632	4.8229
253	3.79338	4.99305	3.6319	5.43888	3.6913	3.87152	3.50853
254	5.67903	5.73183	3.64366	9.99468	3.23139	5.675	3.42811
255	4.03962	7.06895	4.51548	6.38459	5.70899	4.97964	4.3275
256	3.75333	5.46426	5.40434	4.43224	4.04674	5.86557	4.05999
257	3.83412	5.71943	3.85326	5.10096	4.02811	4.03321	3.76375
258	4.18509	5.88112	4.26993	6.3389	4.69024	5.85618	4.36716
259	4.15358	5.05834	4.44136	5.34718	3.99099	5.20403	3.70474
260	3.73139	4.05467	4.02765	3.81558	4.04806	3.96465	3.59973
261	4.36925	5.13793	4.61121	4.27338	4.28008	4.6902	3.82059
262	4.56951	6.16486	4.93888	4.89319	3.93856	4.91585	3.62944
263	4.7936	5.39727	4.6568	6.71037	4.19503	5.02618	4.41236
264	5.94059	6.28145	4.4251	8.51289	5.17011	5.55756	4.5951
265	4.34874	4.90842	4.70686	5.15476	3.81708	4.72544	3.85366
266	4.34397	6.4104	5.41199	4.4216	4.50919	4.33915	4.14657

.2 Active Camera Results

Table 6: Raw Registration Results for the Apartment Texture Rotate Data Set

frame	FM-2D	FM-3D	ICP	PCA	FVR	FFVR	FVR-3D
0	0.478417	0.506736	0.447183	0.445932	0.499573	0.505137	0.470504
1	0.582597	1.61258	0.595438	1.04106	0.754007	0.600421	0.761263
2	19.4351	19.1487	18.6912	20.7154	19.5778	17.7408	18.0525
3	21.7546	24.3136	22.6241	25.0549	22.087	23.671	22.4893

4	1.39929	1.46424	1.415	2.99145	1.46422	1.4817	1.29737
5	0.859244	1.51199	0.857898	2.49516	1.15092	1.06275	0.79598
6	1.06336	1.45169	1.03948	2.72662	1.08908	1.11591	0.957882
7	19.6122	23.3278	20.2606	23.4074	22.226	21.2543	19.4695
8	53.3025	65.6231	56.643	68.3139	58.6042	61.2692	55.1727
9	19.37	20.367	17.4562	22.9191	19.7775	21.9125	16.9486
10	8.25349	9.75559	9.25963	13.7863	9.43728	8.26443	8.50086
11	1.85978	2.19762	1.76417	6.84059	2.25501	2.70236	1.72725
12	1.44217	1.68831	1.48648	5.10666	1.40064	2.39001	1.3949
13	2.67979	3.27427	3.1218	8.25529	2.80969	3.2638	2.5564
14	29.4747	27.4894	21.3027	30.343	24.1091	25.1089	21.4795
15	1.77204	1.9622	2.42012	7.06448	1.559	1.73455	1.99627
16	1.48743	5.13644	1.67647	6.49642	1.3802	1.37318	1.46306
17	3.85845	4.17497	4.09397	22.9444	3.75495	4.14752	4.23414
18	1.69175	2.9192	2.0393	10.3053	1.76836	1.73564	2.0516
19	1.59776	6.71202	1.55585	6.34746	1.42057	1.7095	1.62563
20	3.75736	5.50361	3.99929	29.4137	4.55243	18.6852	3.76322
21	2.12519	3888.74	3.52166	12.418	1.90772	5.23758	1.88939
22	2.43943	2.40684	1.90791	8.61471	3.32825	2.62841	1.96005
23	5.17007	12.0863	5.98678	23.5096	5.02634	5.56165	6.20293
24	1.67374	7.61316	1.82444	4.09527	2.16772	2.16103	1.82006

Table 7: Raw Registration Results for the Apartment Texture X-Axis Rotation Data Set

frame	FM-2D	FM-3D	ICP	PCA	FVR	FFVR	FVR-3D
0	1.33481	7927.91	1.46722	2.777	1.68266	1.46117	1.38799
1	1.68483	2.77985	1.60828	2.4862	1.73615	1.92964	1.49758
2	1.34864	2.12382	1.34021	1.52616	1.81546	1.38809	1.26199
3	2.26964	291.922	1.88852	10.3672	1.99035	1.98026	1.95294
4	1.34885	1.75835	1.3159	12.31	1.38821	1.61666	1.23494
5	1.43367	1.65131	1.44732	3.80585	1.37873	1.46513	1.38593
6	1.71925	2.21381	1.69422	10.4574	1.93514	1.90556	1.82341
7	1.79216	2.43062	1.55475	7.7394	1.77011	2.01243	1.52445
8	2.97638	4.26347	2.60474	7.75538	2.95724	3.3101	2.25391

9	1.48528	1.914	1.30556	3.06841	1.68179	1.73592	1.29888
10	3.28894	3.50101	3.09294	3.863	3.58629	3.09713	2.91952
11	24.6335	24.3533	23.4071	25.1373	25.3556	26.0748	21.7875
12	1.54148	1.7395	1.53637	3.19431	1.7693	1.5894	1.48089
13	2.08955	2.42828	1.58474	3.45761	1.87269	1.74314	1.57237
14	2.12663	2.58139	2.0057	3.72699	2.17569	2.12096	2.05302
15	1.75532	1.84621	1.64236	3.58264	1.85394	2.29551	1.54714
16	17.3425	15.4415	12.9454	17.2937	15.457	16.0395	13.9853
17	30.0512	29.2805	29.1818	30.7484	30.1717	32.0976	27.6835
18	2.4129	2.98033	2.34206	3.14888	3.6444	2.5757	2.36149
19	1.55593	1.66131	1.52418	285.285	1.74957	1.6415	1.5145
20	1.97251	2.2396	1.77509	3.64156	2.2089	1.95566	1.84711
21	2.27393	4.06376	2.32114	3.77645	2.67621	2.41867	2.14902
22	9.86705	14.2648	9.13174	11.947	9.96653	10.6107	9.43974
23	9.23458	9.99594	8.41619	12.9221	9.47407	10.8921	8.11402
24	1.81147	2.23523	1.98875	2.22941	2.39562	2.11487	1.7818

Table 8: Raw Registration Results for the Desk Texture Translation Data Set

frame	FM-2D	FM-3D	ICP	PCA	FVR	FFVR	FVR-3D
0	0.427059	0.386444	0.425551	0.444711	0.79895	0.404669	0.886539
1	0.511405	0.600255	0.37038	0.443054	0.375919	0.398651	0.736221
2	0.505066	0.411181	0.689881	0.495027	0.520825	0.520784	0.696738
3	1.16111	1.14981	0.959396	1.22485	0.742059	0.936038	1.31274
4	1.18962	0.840738	0.747345	1.04372	0.931254	1.02042	1.08887
5	0.579869	0.578428	2.34602	0.699312	0.756597	0.534477	1.0598
6	2.23156	1.59657	1.21059	1.50673	1.0997	1.1579	1.09505
7	0.575454	0.770299	0.451198	0.675469	1.1625	0.463289	0.473196
8	1.51708	2.48019	1.58989	2.27208	2.02134	1.5529	1.55731
9	0.924191	2.46662	0.829318	1.1526	0.870573	0.820371	1.10438
10	26.0733	27.1609	24.6425	26.1605	31.8409	27.3771	26.6029
11	25.8039	16.9519	12.6847	14.0889	13.4272	14.7013	16.1955
12	84.6808	76.7683	75.187	73.985	72.2941	86.135	72.5103
13	14.8138	12.593	11.3068	14.9934	12.3519	12.8473	12.1939

14	8.04678	8.39369	5.96803	8.0337	7.46631	8.55085	8.74751
15	35.9532	33.1912	27.6466	33.1955	32.8383	32.8621	26.51
16	0.677605	0.425612	0.645189	0.618851	0.831358	0.441106	0.601424
17	0.631157	1.08224	0.542638	0.987994	0.531382	0.918999	0.551222
18	1.90329	3.42056	1.58637	2.29047	3.81843	1.57775	1.65624
19	3.32153	2.57921	2.29164	2.69458	6.54668	2.11716	2.9997
20	0.945952	3.52895	2.92841	2.79178	0.958754	1.7856	1.07923
21	2.98774	2.56914	2.37079	3.49873	2.52724	2.64389	1.7058
22	4.03651	31.7833	8.1486	3.23295	3.36118	6.5068	3.83573
23	0.651134	1076.79	0.859187	0.679688	0.513815	0.790372	1.23335
24	1.23536	1.11099	1.11946	1.45004	1.3036	1.28556	1.06952

Table 9: Raw Registration Results for the Office Textured Blindspot Rotation Data Set

frame	FM-2D	FM-3D	ICP	PCA	FVR	FFVR	FVR-3D
0	0.573025	0.538702	0.5302	0.615022	0.754954	0.590357	0.790187
1	0.602199	1.09951	0.643139	0.560507	0.61614	0.772099	0.899576
2	0.711823	0.893597	1.00898	0.812034	1.16404	4.11162	0.93797
3	0.715407	1.05478	0.750809	0.695175	0.705927	1.92263	0.725146
4	0.919503	1.03622	0.752635	0.887547	0.849316	0.88867	0.733311
5	1.08517	0.933163	0.857079	1.29486	0.899383	0.9249	0.910766
6	0.777341	0.931114	0.801236	1.02982	1.85375	1.08238	0.799723
7	0.862278	1.00702	0.747474	1.98783	1.70211	1.75381	0.913958
8	0.759912	1.0946	0.786065	1.50255	1.50606	1.59803	0.817704
9	1.39135	1.97857	1.37474	1.69629	2.06514	2.31021	1.08818
10	1.61388	1.76466	1.20281	5.25926	7.1593	4.8721	1.3515
11	1.65904	3.0659	1.5097	4.83305	4.41015	4.10436	1.3956
12	7.5934	7.23411	6.45561	11.6119	7.95341	15.0171	5.24665
13	10.5448	7.81134	5.84243	11.5236	8.03211	9.0443	5.48707
14	2.30907	6.58343	24.9983	12.1117	3.28068	73.3063	8.87859
15	3.21292	6.39236	4.06469	30.5271	7.24281	4.07405	4.6313
16	2.7655	7.95184	3.82615	26.2629	8.10414	2.92256	4.30693
17	2.34562	3088.49	1.34209	57.892	3.96389	4.45513	1.43469
18	0.940884	325.548	0.744895	2.55858	1.95551	1.47043	0.913393

19	3.25882	7.08096	2.57775	63.8641	25.6575	7.70519	31.9194
20	1.2229	5.91952	1.08254	4.98855	1.65308	1.7223	1.09824
21	35.2899	250.715	22.9173	62.1878	33.6411	35.9841	23.2537
22	6.82018	129.265	4.06837	21.6115	9.68393	9.61946	4.30237
23	0.993991	69.992	0.830335	4.4983	0.880916	0.880278	0.925099
24	3.2537	10.9157	3.95472	19.3076	6.06813	8.00289	2.80695

Table 10: Raw Registration Results for the Office Textured Items Translation Data Set

frame	FM-2D	FM-3D	ICP	PCA	FVR	FFVR	FVR-3D
0	2.62451	2.79327	2.69584	3.15954	7.27674	2.70418	2.83422
1	2.08671	35.8261	2.15925	2.56577	6.67926	2.17364	2.14083
2	2.89056	3.09043	2.92901	3.85315	7.70155	2.71159	2.77382
3	1.93159	4.98797	1.87836	2.11593	1.92978	1.93793	1.87813
4	2.34486	12679.8	2.38783	3.01349	2.45215	2.46212	2.27314
5	3.48885	3.91358	3.5011	3.78892	3.55513	3.48562	3.53985
6	1.77573	6.11207	2.05227	2.06358	1.88169	1.82086	1.81771
7	2.1031	2.29624	2.10874	2.34496	2.38949	1.95614	1.96206
8	2.52423	3.34112	2.51072	3.50523	6.46675	2.77802	2.45442
9	2.04092	2.05311	2.02229	2.77327	6.12307	1.95067	2.05096
10	2.0198	5.44524	2.05008	2.37795	1.96798	1.9625	2.0534
11	3.85712	30.6571	4.16413	3.99734	3.17363	3.08065	2.86399
12	2.33806	4.81093	2.28686	3.2123	4.6796	2.15601	2.14123
13	2.00695	2.17824	2.20605	3.21929	2.15915	3.0649	2.0055
14	5.4637	9089.85	4.84711	5.86673	5.1437	4.77534	4.22902
15	5.00933	6.18594	4.82357	6.4146	26.271	4.60187	4.32704
16	2.2422	2.57168	2.19247	2.47381	2.60619	2.23071	2.22008
17	4.69573	5.44967	4.00174	4.88796	4.90218	4.20595	3.84896
18	6.20389	9.6292	7.2672	6.02013	6.12789	5.77758	5.64114
19	4.85982	5.09415	4.84859	7.8426	5.0413	3.93801	6.27758
20	6.36364	18.559	7.50686	9.50525	9.48946	6.17812	12.3435
21	11.2961	13.3021	12.6449	20.3273	14.2754	12.5326	3181.98
22	4.12729	9.82329	3.73585	5.98633	3.51441	3.57114	3.10604
23	6.15765	29.0517	7.31286	11.2895	11.0797	6.57273	12.2758

24	4.7711	10329.3	5.02903	9.91512	6.45381	5.72149	4.91037
----	--------	---------	---------	---------	---------	---------	---------

Table 11: Raw Registration Results for the Park Plants and Tables Rotation Data Set

frame	FM-2D	FM-3D	ICP	PCA	FVR	FFVR	FVR-3D
0	2.42139	3.98966	3.00145	4.33358	2.62279	2.65714	2.67845
1	10.6682	21.9499	15.003	711.003	8.15678	333.61	10.1369
2	3.1117	9.76618	5.32062	30.7468	2.86785	4.17713	3.59876
3	18.992	26.4685	18.612	42.6827	26.3118	23.9651	21.1591
4	40.3231	18579.8	33.4187	95.087	32.0503	37.9962	23.7485
5	3.31636	7.0465	5.89047	12.6078	2.92416	3.45159	3.56535
6	11.8674	15.3232	13.978	16.1658	10.6316	12.8387	11.7163
7	34.1452	40.452	32.4676	34.7535	32.0078	70.6592	39.4567
8	3.69819	53.8474	14.44	14.385	2.84763	53.8869	5.55395
9	7.13724	78.5394	57.5641	15.4293	6.61391	645.964	12.3454
10	28.4301	2903.09	26.2899	32.857	23.8991	25.4102	27.5763
11	16.4944	19.7024	16.9908	20.4578	12.1766	14.1836	13.8317
12	37.9299	60.3178	45.6223	51.8871	27.4619	31.9236	21.9876
13	8.22564	14.5069	10.3078	10.6138	7.34445	8.03464	8.78498
14	10.1998	15.3833	8.09293	24.7569	8.03295	9.3639	13.6515
15	4.82169	9.69731	4.90147	15.4999	4.01853	7.44971	8.23432
16	13.7588	19.1244	15.0339	25.4273	13.6164	13.9563	604.836
17	8.00143	7.24363	21.9674	20.9297	6.1664	8.85773	71.8715
18	19.2616	541.736	31.7961	44.2418	17.4197	22.4332	199.204
19	8.46994	20.9764	23.5377	27.0112	7.39304	14.3357	260.01
20	10.6171	13.0617	10.0764	23.9967	11.0213	16.4687	31.5444
21	5.05361	27.8228	6.40189	13.3188	3.36639	4.02762	322.643
22	2.84116	7.86983	11.8599	10.0536	2.65861	27.5042	19.3964
23	7.49181	9.57222	100.909	14.7574	7.03093	170.28	10.2781
24	38.4181	54.546	34.2899	43.663	48.946	35.6572	56.5473

Table 12: Raw Registration Results for the Office Texture Rotation Data Set

frame	FM-2D	FM-3D	ICP	PCA	FVR	FFVR	FVR-3D
0	2.21572	4.21984	2.25092	3.1218	3.97379	2.21047	2.23502
1	2.05825	4.1862	2.04934	2.23517	2.08989	2.02422	2.07531
2	2.35681	2.73753	2.36526	2.41352	4.20494	2.40004	2.38712
3	2.128	4.80457	2.23	2.2484	2.10669	2.15907	2.12968
4	2.58431	17.0432	2.70188	2.80298	4.4093	2.65892	2.64551
5	2.20967	3.49584	2.18261	2.22853	3.74658	2.26616	2.12191
6	1.91482	5.98994	1.92767	2.80152	3.75576	2.19196	1.94071
7	1.87922	2.42052	1.95334	2.00312	1.95793	2.02037	1.96854
8	2.92633	5.06522	2.91875	3.25248	2.93665	3.19235	3.07622
9	2.25912	4.84174	2.06333	2.2941	3.77655	2.1319	2.2062
10	3.34784	6.24822	3.38867	4.89315	3.3358	3.64665	3.38224
11	3.07698	4.09064	3.01514	6.55364	4.21989	3.03761	3.10778
12	2.81336	6.67723	2.67824	3.18844	4.13691	2.79145	2.66165
13	4.36234	7.14535	4.90627	5.59819	5.29516	5.18836	4.35153
14	10.8055	38.1029	8.06682	15.5129	9.8488	9.0717	10.6538
15	6.76504	25.1977	5.80137	12.9336	5.89069	6.69544	6.76653
16	18.8346	22.8345	20.5042	30.5968	22.709	47.6484	26.4044
17	22.1577	23.9852	18.6048	26.7638	20.7796	24.5614	19.603
18	7.82136	16.5608	7.35869	11.0303	9.15703	7.04795	7.03271
19	5.63688	9.27229	5.78681	11.0463	5.90937	5.68788	5.56648
20	8.58437	7474.19	7.38113	17.2002	8.23386	8.35952	7.93491
21	4.5199	8961.38	4.76078	9.80655	5.59032	4.74313	4.59129
22	6.47399	6.86407	5.67769	12.2202	6.16859	6.24262	5.58794
23	4.86948	8507.24	4.93801	12.0852	8.93663	5.52515	5.11787
24	11.1138	19.5952	9.04695	16.3056	9.84303	10.3909	9.54403
25	33.3779	365.979	26.0885	29.635	33.284	35.4616	27.3845