IMPROVING DENSE REAL-TIME 3D SLAM USING SPARSE GEOMETRIC CONSTRAINTS

by

John Papadakis

A thesis submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Master of Science in
Electrical Engineering

Charlotte

2018

Approved by:

Dr. Andrew Willis

Dr. James Conrad

Dr. Thomas Weldon

Dr. Yogendra Kakad

ABSTRACT

JOHN PAPADAKIS. Improving Dense Real-Time 3D SLAM using Sparse Geometric Constraints. (Under the direction of DR. ANDREW WILLIS)

This thesis explores the extension of a state of the art dense RGBD SLAM system to include detected geometries as elements in the estimated global map. The existing approach leverages an algorithm for dense visual odometry, which is analyzed in detail and reimplemented. It is demonstrated how the inclusion of these detected geometries can improve the state estimate and reduce reconstruction error. These geometric map elements contain invaluable semantic information about scene content that more dense map representations lack, and serve to improve localization, reduce dense reconstruction error, improve scene understanding.

ACKNOWLEDGEMENTS

First of all, I would like to thank my advisor Dr. Andrew Willis for his unwaivering support and guidance, and for the opportunities he has provided for me to work on truly engaging research. As his student and advisee I have undergone tremendous academic and professional development, and he has my eternal gratitude for his role as professor, advisor, and friend. I would like to thank the members of my committee Dr. James Conrad, Dr. Thomas Weldon, and Dr. Yogendra Kakad- I have benefited greatly from their instruction. I would also like to thank Jamie Gantert, Dr. Kevin Brink, and others at the Air Force Research laboratory for their support and mentorship.

TABLE OF CONTENTS

LIST OF FIGURES viii

LIST OF TABLES x

CHAPTER 1: INTRODUCTION 1

1.1. Contribution 3

1.2. Outline 3

CHAPTER 2: BACKGROUND 4

2.1. Pinhole Camera Model 4

2.2. Camera Calibration 6

2.2.1. Intrinsic Parameterization 6

2.2.2. Extrinsic Parameterization 7

2.2.3. Calibration 7

2.3. RGBD Cameras 8

2.4. Rigid Body Transformations 9

2.4.1. Rotation in 3 Dimensions 9

2.4.2. Translation in 3 Dimensions 11

2.4.3. Representing Transformations 11

2.5. Least Squares Optimization 14

2.5.1. Nonlinear Least Squares 15

2.5.2. Gauss-Newton 15

2.5.3. Levenberg-Marquardt 16

2.6. Random Sample Consensus (RANSAC) 17

2.7. Bayesian Classification 18

2.7.1. Inference 19

2.7.2. Decision 20

2.8. Least Squares Plane Fitting 21

2.8.1. Implicit Plane Fitting to Point Cloud Data 21

2.8.2. Explicit Plane Fitting to Point Cloud Data 23

CHAPTER 3: DENSE VISUAL ODOMETRY 25

3.1. Direct Visual Odometry 25

3.2. Image Alignment 26

3.3. Formalisms 26

3.4. Inverse Compositional Approach 27

3.4.1. Minimization 29

3.4.2. Algorithm Outline 30

3.5. Error Functions 32

3.6. Minimization 33

3.7. Implementation 40

CHAPTER 4: STATE OF THE ART IN DENSE SLAM 43

4.1. Simultaneous Localization and Mapping 43

4.1.1. Graph SLAM 43

4.2. Visual SLAM Overview 46

4.2.1. Pose Graph (Keyframe) SLAM 46

4.3. Dense Visual SLAM 47

4.3.1. Related Work 47

4.3.2. DVO SLAM 50

CHAPTER 5: IMPROVING DENSE SLAM USING DETECTED GEOMETRIES 55

5.1. Extending Dense SLAM Using Landmarks 56

5.1.1. Front-end Modification 56

5.1.2. Back-end Modification 57

5.2. Planar Surface Estimation 59

5.2.1. RANSAC Plane Fitting 59

5.2.2. Fast Least Squares Plane Fitting using Range Images 60

5.2.3. Plane Landmark Correspondence Implementation 63

5.3. Shape Grammars for Object Recognition 65

5.3.1. Framework 66

5.3.2. Recognition of the Colored Ball Semantic 67

5.3.3. Sphere Landmark Correspondence Implementation 72

CHAPTER 6: RESULTS 74

6.1. Dense SLAM with 3D Point Landmarks 74

6.2. Dense SLAM Using Multiple Geometries 75

6.3. Fast Planar Surface Fitting Benchmark 76

CHAPTER 7: CONCLUSION 79

REFERENCES 81

LIST OF FIGURES

FIGURE 2.1: Pinhole Camera Model 5

FIGURE 2.2: Camera Calibration Pattern 8

FIGURE 2.3: Orbbec Astra RGBD Sensor 9

FIGURE 3.1: Warp Function 27

FIGURE 4.1: DVO SLAM Overview 51

FIGURE 4.2: DVO SLAM Local Graph Structure 52

FIGURE 4.3: DVO SLAM Local-Global Merge 53

FIGURE 4.4: DVO SLAM Keyframe Radius Search 53

FIGURE 4.5: DVO SLAM Visualization 54

FIGURE 5.1: DVO SLAM Reconstruction Example 55

FIGURE 5.2: Modified DVO SLAM Local Graph Structure with Landmark Constraints 57

FIGURE 5.3: Modified DVO SLAM Global Graph with Local Merge and Landmark Correspondence 58

FIGURE 5.4: Relationship of $\tan\theta$ 61

FIGURE 5.5: Generated Chair Grammar Instances 65

FIGURE 5.6: Recognition Framework Overview 67

FIGURE 5.7: PSML Program for the Colored Ball Semantic 67

FIGURE 5.8: Rendered Colored Ball Instances 68

FIGURE 5.9: The Hue-Chroma Plane 69

FIGURE 5.10: Colored Circle Detections 71

FIGURE 6.1: Comparison of DVO SLAM and Modified Version Including Colored Sphere Landmarks 75

FIGURE 6.2: Comparison of DVO SLAM and Modified Version Including Both Colored Sphere and Infinite Plane Landmarks 77

LIST OF TABLES

TABLE 6.1: Approximate maximum spread of reconstructed landmark points. 75

TABLE 6.2: Frame-rate comparison between the proposed explicit plane fitting method and two methods available in the OpenCV library. 78

CHAPTER 1: INTRODUCTION

Endowing robotic agents with the ability to reliably navigate and understand complex environments is a crucial prerequisite for performing advanced tasks and for establishing more meaningful scene interaction. This problem is often described as the simultaneous localization and mapping problem (SLAM), whereby a robotic agent must, in real-time, construct a map of its surroundings and accurately keep track of its position and orientation within this map.

Estimation of position and orientation, or “pose”, is on its own a challenging problem. Odometry refers to the use of sensor data to estimate change in pose over time. Odometry measurements, sometimes from multiple sensors, are integrated to form an estimate of position and orientation. The accuracy of an odometry only pose estimate is clearly then limited by the accuracy of the odometry estimation. Without consideration of outside information of an agent’s pose, a pose estimate based purely on odometry will be sure to accumulate drift- and an accurate estimate of pose is essential to the success of any SLAM system.

Therefore, agents performing SLAM are equipped with sensors capable of detecting scene features. Detected scene features can then be used to build a map of the environment and act as landmarks to aid in localization. Visual SLAM refers to the class of solutions that leverage vision sensors such as conventional cameras and, more recently, systems capable of detecting 3D geometry such as stereo camera pairs, LiDAR (Light Detection and Ranging) sensors, and RGBD (color RGB + Depth) sensors. SLAM systems that leverage RGBD sensors are particularly popular as they provide both surface color and surface geometry information at every pixel location.

Modern SLAM solutions that leverage these newly available sensors, e.g., RGBD sensors, have an unprecedented amount of scene information available to them. This has spurred development and has led to improved solutions with increased camera tracking and scene reconstruction accuracy. Recent advancements in RGBD SLAM adopt pixel-based or "direct" approaches for camera tracking that improve localization results by utilizing all of the measured image data. While direct methods improve localization accuracy, the global map is often represented as a very large collection of unorganized primitives such as 3D points. Unfortunately, these solutions do not capitalize on fundamental geometric constraints typical to real-world structures such as surface continuity, surface topology (closedness), etc. Further, maps consisting solely of dense point clouds provide no structural or semantic information about the scene. Absence of this information limits the utility of the constructed map to occupancy-type queries, e.g., "is there an obstruction here?", and therefore limits the use of these maps to coarse-level obstacle avoidance and path-planning for robotic navigation, guidance, and control systems.

Utilization of higher order scene features at the surface and object level, specifically those which have been identified semantically, for SLAM is of particular interest. Surface model approximations of dense point cloud data can be used to improve map accuracy, compress the map representation, and provide semantic information for map contents. For this reason, this thesis explores the use of 3D shapes and shape primitives for SLAM, resulting in semantic geometric map contents. These types of primitives are inherently more distinct when compared to raw data, and should be able to be robustly detected. If used for mapping, these features exhibit high compression by representing large areas of point cloud data, thereby reducing storage and subsequent computational costs. More importantly, object level maps contain invaluable semantic information about scene content that more dense map representations lack.

1.1 Contribution

This thesis explores the extension of a state of the art dense RGBD SLAM implementation to include detected objects and surfaces as geometric elements in the estimated global map. These elements relay important semantic scene information otherwise unavailable from the current dense map representation (an unorganized collection of colored 3D points). In addition, these map elements act as landmarks to aid in localization, improving both camera tracking and reconstruction accuracy.

Additionally, this thesis analyses the theory and implementation of the direct visual odometry algorithm leveraged by the aforementioned dense SLAM approach. This includes the development of a simplified reference implementation more amenable to understanding and extension than currently available open source implementations.

1.2 Outline

This thesis is outlined as follows: Chaper 2 presents background information and concepts that will be applied in later chapters. Chapter 3 investigates the theory and implementation of a direct visual odometry algorithm, and provides in depth analysis and presents a reference implementation. Chapter 4 provides a background on graph based SLAM, dense SLAM, and the details the specific dense SLAM implementation to be extended. Chapter 5 details the modifications necessary to extend a state of the art dense 3D SLAM implementation to include detected geometries as landmarks. It also discusses the detection of of the specific geometries to be leveraged. Chapter 6 evaluates the efficacy of including detected geometries to improve the SLAM estimation. Chapter 7 summarizes the discussed approaches and results, and provides outlook on possible future research.

CHAPTER 2: BACKGROUND

This chapter presents background information and concepts that will be applied in later chapters. Topics include those necessary for image formation and reconstruction, rigid body motion, solving least-squares problems, Bayesian classification, and model fitting.

2.1 Pinhole Camera Model

The pinhole camera model describes mathematically a lenseless camera with a pinhole aperture and image plane as shown in figure 2.1. This model assumes that only light passing directly through the pinhole aperture intersects the image plane. This model provides a projection mapping, ρ, between 3D world points $\mathbf{p} = [\begin{array}{ccc} X & Y & Z \end{array}]^T$, and 2D points on the image plane $\mathbf{x} = [\begin{array}{cc} x & y \end{array}]^T$:

$$\rho(\mathbf{p}) \rightarrow \mathbf{x} \tag{2.1}$$

or equivalently:

$$\rho(X, Y, Z) \rightarrow (x, y) \tag{2.2}$$

The pinhole camera model defines this perspective projection in terms of a set of parameters intrinsic to the camera, including focal length f, in meters, principle point or optical center (c_x, c_y), in pixels, and pixel size (s_x, s_y), in meters/pixel. Axis skew and lens distortion can also be modeled, however, they are neglected in this case. The simplifications $f_x = \frac{f}{s_x}$ and $f_y = \frac{f}{s_y}$ are often made to express the focal length in units of pixels. The resulting 2D pixel coordinates after projection are given by:

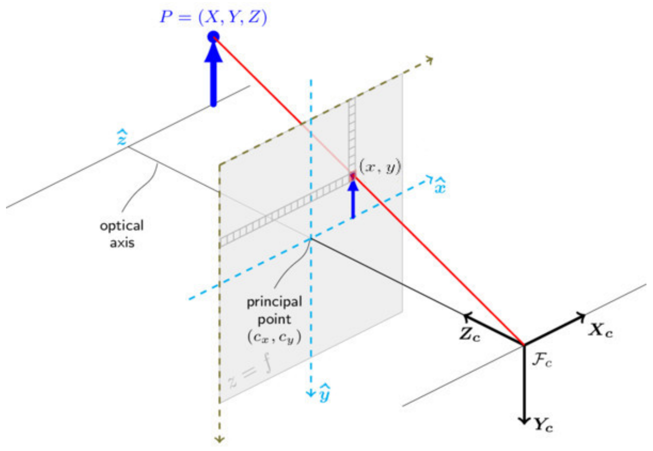


Figure 2.1: Pinhole camera model showing the relationship between 3D world points and their projection onto the image plane. Modified from OpenCV API Reference: Camera Calibration and 3D Reconstruction [1].

$$\begin{aligned} x &= f_x \frac{X}{Z} + c_x \\ y &= f_y \frac{Y}{Z} + c_y \end{aligned} \tag{2.3}$$

In addition, if the distance from the image plane (Z) is known for a given pixel value (as in a depth image), one can reconstruct the associated 3D point using the inverse mapping:

$$\rho^{-1}(x, y, Z) \rightarrow (X, Y, Z) \tag{2.4}$$

or equivalently:

$$\rho^{-1}(\mathbf{x}, Z) \rightarrow \mathbf{p} \tag{2.5}$$

such that

$$\begin{aligned} X &= \tfrac{Z}{f_x}(x - c_x) \\ Y &= \tfrac{Z}{f_y}(y - c_y) \\ Z &= Z \end{aligned} \tag{2.6}$$

These methods for image formation and reconstruction are essential to the dense visual odometry algorithm discussed in chapter 3, which is leveraged in the dense SLAM algorithm discussed in chapters 4 and 5.

2.2 Camera Calibration

Camera calibration refers to the determination of the intrinsic and extrinsic parameters of a given vision system. Each individual vision system will have a unique set of camera parameters that must be determined through calibration in order to form an accurate system model. With this model, we have for a particular system a mathematical understanding about the relationship between points in a 3D scene and their projection onto the image plane, and how image data is represented in various frames of reference.

2.2.1 Intrinsic Parameterization

A camera can be modeled by the pinhole camera model, discussed in section 2.1, which describes the relationship between points in 3D space and their corresponding 2D image pixel locations in terms of a set of parameters intrinsic to the camera. These intrinsic parameters are often expressed in matrix form as the so called K matrix:

$$K = \begin{bmatrix} \frac{f}{s_x} & 0 & c_x \\ 0 & \frac{f}{s_y} & c_y \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

Assuming the world origin coincides with the focal point of the camera, as shown in figure 2.1, the perspective projection of the 3D point (X, Y, Z) to the image point(x, y) can be expressed using equation (2.8)

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.8}$$

where $x = {}^{u}/{}_{w}$, and $y = {}^{v}/{}_{w}$.

2.2.2 Extrinsic Parameterization

Extrinsic camera parameters are a set of geometric parameters used to determine accurately the fixed transformation between the camera frame and the world frame. Unlike in section 2.2.1 the world origin is no longer located at the camera focal point, as we have allowed the camera to translate and rotate in space. These extrinsic parameters are expressed as a rotation matrix $R_{3\times3}$ and translation vector $t_{3\times1}$. The relationship between $P_{image} = [\ x \quad y \quad 1\ ]^T$ and $P_{world} = [\ X \quad Y \quad Z \quad 1\ ]^T$ in homogeneous coordinates can then be expressed as:

$$P_{image} = K \begin{bmatrix} R & | & t \end{bmatrix} P_{world} \tag{2.9}$$

or more explicitly:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.10}$$

where $x = {}^{u}/{}_{w}$, and $y = {}^{v}/{}_{w}$.

Note that if we allow the world frame to coincide with the focal point of the camera, as it did in section 2.2.1, then R is identity and $t = [\ 0 \quad 0 \quad 0\ ]^T$, and equation 2.10 simplifies to that of equation (2.8).

2.2.3 Calibration

The projection matrix $P = K[R|t]$ then has 11 free parameters which we wish to determine through calibration. The first 5 of these parameters are the intrinsic parameters f, c_x, c_y, s_x, and s_y. The remaining 6 are the translation components t_x, t_y, t_z, and the 3 angles of rotation specified in R. By establishing a set of known correspondences between world and image points these parameters can be estimated.

To aid in this endeavor, a calibration pattern, as shown in figure 2.2, is often used. The dimensions of the pattern are known accurately, and the pixel coordinates the corners of the checkerboard can be extracted from images taken of the pattern. Correspondence is computed between the physical coordinates of the pattern features and those extracted from images, resulting in a set of measurements that can be used to estimate the unknown parameters of the camera model, i.e. those found in the matrix K. [2, 3]

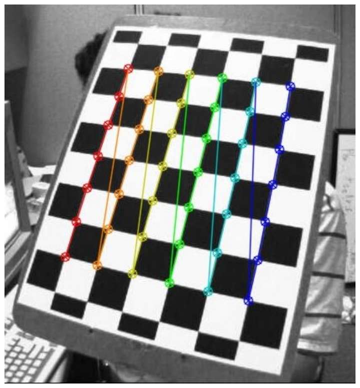


Figure 2.2: A checkerboard pattern used for camera calibration. [1]

The parameters of the camera model must be known in order to properly project 3D points onto the image plane, and reconstruct 3D points from depth values (see section 2.1).

2.3 RGBD Cameras

RGBD sensors are a popular low cost, high performance addition to robotic sensor suites. These sensors combine a traditional color camera with an infrared depth sensor to produce full HD color and range images at real-time frame rates ($\geq$ 30 fps). Unlike traditional 2D cameras, RGBD sensors also provide depth measurements that directly impart a sense of scene geometry, without the use of techniques such as stereoscopic reconstruction. Utilization of RGBD sensors for SLAM has become particularly popular as they provide both surface color and surface geometry information

Figure 2.3: The Orbbec Astra RGBD sensor. Image courtesy of roscomponents.com.

at every pixel location.

A common class of RGBD sensors is those that leverage structured light approaches for depth estimation. Structured light sensors are active sensors that project a known infrared pattern out into the scene. These sensors are also equipped with infrared cameras located at a known baseline from the projector, which detect the pattern. Using the difference between the known pattern and the detected pattern, depth is calculated using triangulation. A number of common RGBD sensors utilize structured light technology including the Microsoft Kinect, the Asus Xtion, and the Orbbec Astra. For the results generated in this thesis, the Orbbec Astra sensor, shown in figure 2.3, was used.

2.4 Rigid Body Transformations

The 3D Euclidean transformations discussed in this thesis are the proper rigid transformations of the 3D special Euclidean group, **SE**(3), that preserve distance and orientation between point pairs. This group of transformations compactly describes the motion of rigid bodies in 3D. These transformations contain both rotational and translational components, totaling of 6 degrees of freedom.

2.4.1 Rotation in 3 Dimensions

Rotational components of these transformations alter the orientation of the coordinate frame, leaving its origin unchanged. Rotations of this type belong to the rotation group **SO**(3), and can be represented using a variety of formalisms [4]. Perhaps the most common is the 3×3 orthogonal matrix, $\mathbf{R}$:

$$\mathbf{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \tag{2.11}$$

Rotations corresponding to zero rotation are given by $\mathbf{R} = I$.

Also used is the so called axis-angle vector or rotation vector, $\boldsymbol{\omega} \in \mathbb{R}^3$, whose direction defines the axis of rotation and whose magnitude corresponds to the rotation angle in radians. This is equivalently parameterized by the directional unit vector $\hat{\boldsymbol{\omega}}$, and accompanying rotation angle $\theta = \|\boldsymbol{\omega}\|$. This parameterization is therefore a minimal representation of the rotation- it has three free parameters corresponding to the three rotational degrees of freedom. The rotation matrix R corresponding to the to the rotation vector $\boldsymbol{\omega}$ is given by the Rodrigues formula [5]:

$$\mathbf{R} = I + \sin(\|\boldsymbol{\omega}\|) \left[\hat{\boldsymbol{\omega}}\right]_\times + (1 - \cos(\|\boldsymbol{\omega}\|)) \left[\hat{\boldsymbol{\omega}}\right]_\times^2 \tag{2.12}$$

or equivalently,

$$\mathbf{R} = I + \frac{\sin(\|\boldsymbol{\omega}\|)}{\|\boldsymbol{\omega}\|} \left[\boldsymbol{\omega}\right]_\times + \frac{(1 - \cos(\|\boldsymbol{\omega}\|))}{\|\boldsymbol{\omega}\|} \left[\boldsymbol{\omega}\right]_\times^2 \tag{2.13}$$

where the operator

$$\left[\mathbf{u}\right]_\times = \begin{bmatrix} 0 & -u_3 & u_2 \\ u_3 & 0 & -u_1 \\ -u_2 & u_1 & 0 \end{bmatrix} \tag{2.14}$$

gives the skew-symmetric cross product matrix for the vector $\mathbf{u}$. This formula is a closed form of the exponential map, which maps $\mathfrak{so}(3) \to \mathbf{SO}(3)$ such that,

$$\mathbf{R} = e^{[\boldsymbol{\omega}]_\times} = \sum_{k=0}^{\infty} \frac{1}{k!} [\boldsymbol{\omega}]_\times^k \tag{2.15}$$

where the operator $e^{(\cdot)}$ is the matrix exponential. This maps skew symmetric matrices corresponding to infinitesimal rotations about an axis into the space of orthogonal rotation matrices [5]. The inverse mapping termed the log map, maps $\mathbf{SO}(3) \rightarrow \mathfrak{so}(3)$ by

$$\begin{aligned} \|\boldsymbol{\omega}\| &= \arccos(\tfrac{\text{trace}(\mathbf{R})-1}{2}) \\ \hat{\boldsymbol{\omega}} &= \tfrac{1}{2\sin(\|\boldsymbol{\omega}\|)} \begin{bmatrix} r_{32}-r_{23}, & r_{13}-r_{31}, & r_{21}-r_{12} \end{bmatrix}^T \end{aligned} \tag{2.16}$$

then $\boldsymbol{\omega} = \|\boldsymbol{\omega}\| \cdot \hat{\boldsymbol{\omega}}$.

2.4.2 Translation in 3 Dimensions

The translational component of these transformations is represented as a vector $\mathbf{t} \in \mathbb{R}^3$, where

$$\mathbf{t} = \begin{bmatrix} t_X \\ t_Y \\ t_Z \end{bmatrix} \tag{2.17}$$

and $\mathbf{t}$'s components specify translation along the axes. Translations represented in this way are also minimal.

2.4.3 Representing Transformations

A proper rigid transformation is commonly represented using the homogeneous 4x4 transformation matrix, $\mathbf{T}$, which is comprised of the block matrix $\mathbf{R}$ and the translation vector $\mathbf{t}$ such that

$$\mathbf{T} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} \tag{2.18}$$

The inverse of $\mathbf{T}$ is then given by

$$\mathbf{T}^{-1} = \begin{bmatrix} \mathbf{R}^T & -\mathbf{R}^T\mathbf{t} \\ 0 & 1 \end{bmatrix} \tag{2.19}$$

Transformation matrices in this form can be easily chained via premultiplication. For example, a point undergoing the transformation $\mathbf{T}_2 \cdot \mathbf{T}_1$ would be first transformed by $\mathbf{T}_1$ followed by $\mathbf{T}_2$. A point $\mathbf{p} = [\ X \quad Y \quad Z\ ]^T$ represented using homogeneous coordinates as undergoing transformation, τ, is then given by

$$\begin{bmatrix} \mathbf{p}' \\ 1 \end{bmatrix} = \tau(\mathbf{T}, \mathbf{p}) = \mathbf{T} \cdot \begin{bmatrix} \mathbf{p} \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_X \\ r_{21} & r_{22} & r_{23} & t_Y \\ r_{31} & r_{32} & r_{33} & t_Z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{2.20}$$

where $\mathbf{p}' = [\ X' \quad Y' \quad Z'\ ]^T$ is the transformed point.

Transformations of this type can also be compactly represented using exponential coordinates [6]. The transformations of the Lie group $\mathbf{SE}(3)$ can be mapped from their corresponding parameterization in the tangent space $\mathfrak{se}(3)$ through the exponential map. These parameters are represented using a 6x1 vector

$$\xi = \begin{bmatrix} \mathbf{v} \\ \boldsymbol{\omega} \end{bmatrix} \tag{2.21}$$

where $\mathbf{v} = [\ v_1 \quad v_2 \quad v_3\ ]^T$ determines the amount of translation along the rotation axis [7], and $\boldsymbol{\omega}$ is the rotation parameterization discussed in section 2.4.1. These parameters are called the “twist” parameters, as they correspond to the 4x4 matrix known as a twist (a generator of the Euclidean group), given by

$$\hat{\xi} = \begin{bmatrix} [\boldsymbol{\omega}]_\times & \mathbf{v} \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 & v_1 \\ \omega_3 & 0 & -\omega_1 & v_2 \\ -\omega_2 & \omega_1 & 0 & v_3 \\ 0 & 0 & 0 & 0 \end{bmatrix} \tag{2.22}$$

The exponential map is then given by

$$\mathbf{T}_\xi = e^{\hat{\xi}} \tag{2.23}$$

where $e^{\hat{\xi}}$ is the matrix exponential of the twist and $\mathbf{T}_\xi$ is the corresponding homogeneous transformation matrix in $\mathbf{SE}(3)$. This matrix exponential has the solution

$$\begin{aligned} \mathbf{T}_\xi &= e^{\hat{\xi}} \\ \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ 0 & 1 \end{bmatrix} &= \begin{bmatrix} e^{[\boldsymbol{\omega}]_\times} & \mathbf{V}\mathbf{v} \\ 0 & 1 \end{bmatrix} \end{aligned} \tag{2.24}$$

where

$$\mathbf{R} = I + \frac{\sin(\|\boldsymbol{\omega}\|)}{\|\boldsymbol{\omega}\|} [\boldsymbol{\omega}]_\times + \frac{(1 - \cos(\|\boldsymbol{\omega}\|))}{\|\boldsymbol{\omega}\|} [\boldsymbol{\omega}]_\times^2 \tag{2.25}$$

$$\mathbf{V} = I + \frac{1 - \cos(\|\boldsymbol{\omega}\|)}{\|\boldsymbol{\omega}\|^2} [\boldsymbol{\omega}]_\times + \frac{(\|\boldsymbol{\omega}\| - \sin(\|\boldsymbol{\omega}\|))}{\|\boldsymbol{\omega}\|^3} [\boldsymbol{\omega}]_\times^2 \tag{2.26}$$

Note the exponential map for the rotational component, $e^{[\boldsymbol{\omega}]_\times}$, is given by the Rodrigues formula.

The inverse mapping, termed the log map, can also be computed as

$$\log(\mathbf{T}_\xi) \rightarrow \xi \tag{2.27}$$

The result for the rotational component is given in equation (2.16), reproduced here

$$\begin{aligned} \|\boldsymbol{\omega}\| &= \arccos(\tfrac{\text{trace}(\mathbf{R})-1}{2}) \\ \hat{\boldsymbol{\omega}} &= \tfrac{1}{2\sin(\|\boldsymbol{\omega}\|)} \begin{bmatrix} r_{32}-r_{23}, & r_{13}-r_{31}, & r_{21}-r_{12} \end{bmatrix}^T \end{aligned} \tag{2.28}$$

then $\boldsymbol{\omega} = \|\boldsymbol{\omega}\| \cdot \hat{\boldsymbol{\omega}}$. To compute the translational component the matrix $\mathbf{V}$ has a closed form inverse given by

$$\mathbf{V}^{-1} = I - \frac{1}{2}[\boldsymbol{\omega}]_\times + \frac{1}{\|\boldsymbol{\omega}\|^2}\left(1 - \frac{\|\boldsymbol{\omega}\|}{2}\frac{\sin(\|\boldsymbol{\omega}\|)}{1-\cos(\|\boldsymbol{\omega}\|)}\right)[\boldsymbol{\omega}]_\times^2 \tag{2.29}$$

and

$$\mathbf{v} = \mathbf{V}^{-1} \cdot \mathbf{t} \tag{2.30}$$

These representations for rigid body motion are used throughout this thesis to describe and solve for unknown camera motions and other transformations necessary for SLAM. These formulations are essential for the visual odometry algorithm discussed in chapter 3, and for graph based map-building methods discussed in chapter 4

2.5 Least Squares Optimization

Least squares problems are show up often in the areas of robotics and computer vision. In this thesis they are used to solve for unknown camera motion (chapter 3), and for graph SLAM optimization (chapters 4 and 5).

The least squares problem seeks the set of model parameters, $\theta = [\ \theta_1, \ \ \theta_2, \ \ \dots \ \ \theta_m\ ]^T$, that minimizes the difference between observations and the values predicted by the model. The observed differences between observations and predictions are termed residuals, with the vector of residuals defined as $\mathbf{r}(\theta) = [\ r_1(\theta), \ \ r_2(\theta), \ \ \dots \ \ r_n(\theta)\ ]^T$. We then define an error function or objective function,

$$f(\theta) = \frac{1}{2}\sum_{i=1}^{n} r_i(\theta)^2 = \frac{1}{2}\mathbf{r}^T\mathbf{r} \tag{2.31}$$

which consists of the sum of squared residual values. We then seek the values of the parameters that minimizes the sum of squared residual values, this is the least squares estimate for the parameters given by,

$$\theta_e = \arg\min_{\theta}(f(\theta)) \tag{2.32}$$

which can be minimized by computing the gradient of $f(\theta)$ and setting it equal to zero:

$$\frac{\partial f(\theta)}{\partial \theta} = \sum_{i=1}^{n} r_i \frac{\partial r_i}{\partial \theta} = 0 \tag{2.33}$$

2.5.1 Nonlinear Least Squares

For a nonlinear system, the derivatives of the residuals in equation (2.33) do not have a closed form solution, and we therefore move into the realm of nonlinear least squares problems.

2.5.2 Gauss-Newton

Nonlinear least squares problems can often be iteratively by first guessing an initial value for the parameters. Then, at each iteration, linearization around the current parameter estimate is performed, followed by solving the resulting linear system and updating the parameters. This process is repeated until convergence is achieved. This is known as the Gauss-Newton (GN) method.

The linearized residual about the point θ_0 is given by:

$$r_i(\theta)|_{\theta=\theta_0} = r_i(\theta_0) + J_i(\theta_0)(\theta - \theta_0) \tag{2.34}$$

where

$$J_i(\theta_0) = \frac{\partial r_i}{\partial \theta}|_{\theta=\theta_0} \tag{2.35}$$

This is performed at every iteration such that

$$r_i(\theta_{k+1})|_{\theta_k} = r_i(\theta_k) + J_i(\theta_k)\Delta\theta \tag{2.36}$$

applying this linearization to equation (2.33) yields

$$\begin{aligned} \sum_{i=1}^{n} J_i(\theta_k)^T(r_i(\theta_k) + J_i(\theta_k)\Delta\theta) = 0 \\ \sum_{i=1}^{n} J_i(\theta_k)^T J_i(\theta_k)\Delta\theta = -\sum_{i=1}^{n} J_i(\theta_k)^T r_i(\theta_k) \end{aligned} \tag{2.37}$$

and after simplification

$$\mathbf{J}(\theta_k)^T\mathbf{J}(\theta_k)\Delta\theta = -\mathbf{J}(\theta_k)^T\mathbf{r}(\theta_k) \tag{2.38}$$

where $\mathbf{J}(\theta_k)$ is the $n \times m$ Jacobian matrix with i^{th} row $J_i(\theta_k)$. The matrix is known as the Gauss-Newton approximation for the Hessian, which is the matrix of second order partial derivatives.

The parameter update is then given by

$$\Delta\theta = -\left(\mathbf{J}(\theta_k)^T\mathbf{J}(\theta_k)\right)^{-1}\mathbf{J}(\theta_k)^T\mathbf{r}(\theta_k) \tag{2.39}$$

and the parameters are iteratively refined:

$$\theta_{k+1} = \theta_k + \Delta\theta \tag{2.40}$$

2.5.3 Levenberg-Marquardt

The Levenberg-Marquardt (LM) method [8, 9], sometimes referred to as damped least-squares, is quite similar to the Gauss-Newton method. In fact, LM interpolates between the method of gradient-descent and Gauss-Newton using a varying damping parameter λ. Instead of the using the Gauss-Newton approximation to the Hessian, as in the previous section, Levenberg replaced this term is replaced by a "damped" version:

$$\mathbf{J}(\theta_k)^T\mathbf{J}(\theta_k) \rightarrow \mathbf{J}(\theta_k)^T\mathbf{J}(\theta_k) + \lambda\mathbf{I} \tag{2.41}$$

where the damping parameter is adjusted at every iteration. Large values of λ steer the minimization towards gradient-descent. Conversely, for small values ofλ the minimization is closer to GN.

Marquardt later improved convergence by scaling the identity matrix according to the diagonal terms of GN Hessian approximation as in equation (2.42).

$$\lambda\mathbf{I} \rightarrow \lambda\text{diag}\left(\mathbf{J}(\theta_k)^T\mathbf{J}(\theta_k)\right) \tag{2.42}$$

The parameter update of equation (2.39) becomes:

$$\Delta\theta = -\left(\mathbf{J}(\theta_k)^T\mathbf{J}(\theta_k) + \lambda\text{diag}\left(\mathbf{J}(\theta_k)^T\mathbf{J}(\theta_k)\right)\right)^{-1}\mathbf{J}(\theta_k)^T\mathbf{r}(\theta_k) \tag{2.43}$$

This method along with other methods for solving nonlinear least squares problems are used often in the areas of computer vision and robotics.

2.6 Random Sample Consensus (RANSAC)

Often, we wish to fit some model to a set of observations, even when the set contains outliers- observations which are distant from other observations. A common method for model fitting in the presence of outliers is the Random Sample Consensus method, or RANSAC [10]. This iterative method estimates the parameters of a model from a set of observations which are assumed to contain a subset of observations known as inliers. Inliers are observations that can be well explained by some set of model parameters.

RANSAC generally proceeds as follows:

1. A random set of samples is selected from all observations, often the minimum amount needed to fit the model in question.

2. A model is fit to the randomly selected samples using some method of parameter estimation.

3. All other samples are then checked against the model using some model specific error function and partitioned into two sets: inlers, which lie close to the model (below some problem specific threshold value), and outliers which do not.

4. The model is considered reasonable if a sufficient proportion of samples have been classified as inliers, in this case the model may be refined by reestimating it using the inlier set to see if the model fit improves.

5. The entire procedure is repeated until some number of iterations is exceeded or the model is considered to be a good fit to the data.

RANSAC is then capable of robust estimation, estimation of accurate model parameters even in the case of outliers. However, RANSAC is unbounded in terms of the number of iterations required to compute accurate model parameters. In addition, there is no guarantee that the accuracy of the currently estimated model increases with the number of iterations, only that the probability of finding an accurate model increases.

RANSAC is applied in this thesis to estimate the parameters of both planar and spherical models from detected 3D point samples. See sections 5.2.1 and 5.3.2 for more detail.

2.7 Bayesian Classification

The general goal of classification is to find the class that the observed data best fits in. Bayesian classifiers probabilistically relate input data to a finite set of class labels using conditional probability models for each class. Each of the K classes is modeled by a conditional distribution which yields the likelihood of the class, C_k, given the observed data, $\mathbf{x}$, in terms of how likely the data is to be observed, given the class is known. These probabilities are then used to make a classification decision for the observation.

2.7.1 Inference

Conditional relationships can be written using Bayes' theorem for conditional probability as equation (2.44)

$$P\left(A|B\right) = \frac{P\left(B|A\right)P\left(A\right)}{P\left(B\right)} \tag{2.44}$$

where A and B are events, with $P(A) > 0$. Here the conditional probability, $P(A|B)$, which describes the likelihood of the occurrence of A, given B has occurred, is related to the inverse conditional probability, $P(B|A)$, and the marginal probabilities: $P(A)$ and $P(B)$, which describe the likelihood of independently observing events A and B.

In terms of the class labels and the observed data, this relationship is given by equation (2.45).

$$P\left(C_k|\mathbf{x}\right) = \frac{P\left(\mathbf{x}|C_k\right)P\left(C_k\right)}{P\left(\mathbf{x}\right)} \tag{2.45}$$

The class conditional density, $P(\mathbf{x}|C_k)$, is called the likelihood term, which yields the likelihood that the data is observed given we know the observation belongs to the class C_k. This term is often found by selecting an appropriate statistical model for the class, e.g. Gaussian, and employing Maximum Likelihood Estimation (MLE) to estimate the class conditional model parameters given class data. For more complex models, e.g. mixture models, compound distributions, etc., iterative algorithms such as the Expectation Maximization (EM) algorithm (see [11], section 9.4) or RANSAC (see section 2.6) can be used to estimate these parameters. Note a likelihood density is not required to be a probability distribution, i.e. $\sum_{\mathbf{x}} P\left(\mathbf{x}|C_k\right)$ is not required to be 1.

The prior probability, $P(C_k)$, gives the probability of the class before (prior to) considering the observation. This density is used to express one's prior beliefs about the likelihood of the class before any observations are made. For example, a simple prior for the k^{th} class can be generated by taking the proportion of the training data belonging to the k^{th} class.

The evidence term, $P(\mathbf{x})$, is the same across classes and can be found strictly using terms from the numerator of Bayes' theorem as shown in equation (2.46).

$$\begin{aligned} P(\mathbf{x}) &= \sum_{k} P\left(\mathbf{x}|C_k\right) P\left(C_k\right) \\ &= P\left(\mathbf{x}|C_1\right) P\left(C_1\right) + P\left(\mathbf{x}|C_2\right) P\left(C_2\right) + ... + P\left(\mathbf{x}|C_K\right) P\left(C_K\right) \end{aligned} \tag{2.46}$$

The evidence term is often treated as a normalizer, since it normalizes the product of the likelihood and prior across all possible classes, allowing them to be expressed as a probability density.

Finally, the posterior probability, $P(C_k|\mathbf{x})$, yields the probability that the class label is C_k given the observed data. Directly estimating this distribution can be challenging, but as shown, can be described in terms of known quantities using Bayesian inference. *Discriminative* models seek to model this posterior directly, while *generative* models model the class conditional densities and class priors, and are capable of generating synthetic input data by sampling the resulting distribution.

2.7.2 Decision

Once the class posterior distributions have been estimated, we wish to estimate the the class, $\hat{C}_k$, that maximizes the posterior. This is known as the maximum a posteriori (MAP) decision rule and is shown in equation (2.47).

$$\hat{C}_k = \underset{k}{\operatorname{argmax}}\, P\left(C_k|\mathbf{x}\right) = \underset{k}{\operatorname{argmax}}\, P\left(\mathbf{x}|C_k\right) P\left(C_k\right) \tag{2.47}$$

We can ignore the evidence term since it is but a normalization constant, and replace the posterior with the product of the likelihood and the prior. Furthermore, we can maximize what is known as the log likelihood, as in equation (2.48), which can greatly simplify computation. The natural logarithm is a monotonically increasing function, and thus does not change the location of the maximum.

$$\begin{aligned}\hat{C}_k &= \underset{k}{\operatorname{argmax}}\, \ln\left(P\left(\mathbf{x}|C_k\right) P\left(C_k\right)\right) \\ &= \underset{k}{\operatorname{argmax}}\left(\ln P\left(\mathbf{x}|C_k\right) + \ln P\left(C_k\right)\right)\end{aligned} \tag{2.48}$$

In this thesis, we apply a Bayesian classifier in order to classify color image pixels as belonging to a set of color classes of interest. See section 5.3.2 for more detail.

2.8 Least Squares Plane Fitting

In this thesis, in order to fit planar models to sets of 3D points, we leverage a least squares formulation to compute the coefficients of the plane that minimizes the error functions presented.

2.8.1 Implicit Plane Fitting to Point Cloud Data

This section discusses the typical approach for 3D plane fitting which estimated the plane when expressed as an implicit polynomial. This method seeks to minimize the square of the perpendicular distance between N measured (X, Y, Z) data points and the estimated planar model, i.e.,

$$\epsilon(a,b,c,d) = \min_{a,b,c,d} \sum_{i=1}^{N} \left\|aX_i + bY_i + cZ_i + d\right\|^2 \tag{2.49}$$

We re-write this objective function as a quadratic matrix-vector product by defining the vector $\alpha = [\ a \quad b \quad c \quad d\ ]^t$ as the vector of planar coefficients and the matrix $\mathbf{M}$ as the matrix of planar monomials formed from the measured (X, Y, Z) surface data having i^{th} row $\mathbf{M}_i = [\ X_i \quad Y_i \quad Z_i \quad 1\ ]$. Using this notation, the optimization function becomes:

$$\epsilon(\alpha) = \min_{\alpha} \alpha^t \mathbf{M}^t \mathbf{M} \alpha$$

Solving for the unknown plane equation coefficients requires a constraint on the coefficient vector: $\|\alpha\|^2 = \alpha^t\alpha = 1$ to avoid the trivial solution $\alpha = \mathbf{0}$. Equation (2.50) incorporates this constraint as a Lagrange multiplier.

$$\epsilon(\alpha) = \min_{\alpha} \left(\alpha^t \mathbf{M}^t \mathbf{M} \alpha - \lambda \left(\alpha^t \mathbf{I} \alpha - 1 \right) \right) \tag{2.50}$$

Taking the derivative of the error function then provides

$$\frac{d\epsilon(\alpha)}{d\alpha} = \min_{\alpha} \left(\left(\mathbf{M}^t \mathbf{M} - \lambda \mathbf{I} \right) \alpha \right) \tag{2.51}$$

Then from [?, ?, ?] the minimizer is known to be $\widehat{\alpha}$, the eigenvector associated with the smallest eigenvalue of the matrix $\mathbf{M}^t\mathbf{M}$ (also known as the scatter matrix). In general, $\mathbf{M}^t\mathbf{M}$ is a symmetric matrix and, for the monomials $\mathbf{M}_i = [\ X_i \quad Y_i \quad Z_i \quad 1\ ]$, the elements of this matrix are

$$\mathbf{M}^t\mathbf{M} = \sum_{i=1}^{N} \begin{bmatrix} X_i^2 & X_iY_i & X_iZ_i & X_i \\ X_iY_i & Y_i^2 & Y_iZ_i & Y_i \\ X_iZ_i & Y_iZ_i & Z_i^2 & Z_i \\ X_i & Y_i & Z_i & 1 \end{bmatrix} \tag{2.52}$$

Fitting implicit planar surfaces to point cloud data has become the de-facto standard for point cloud processing algorithms and is now part of many standard point cloud and image processing libraries, e.g., OpenCV and PCL (Point Cloud Library)

[12, 13]. It's popularity is due to it's relatively low computational cost and Euclidean invariance.

2.8.2 Explicit Plane Fitting to Point Cloud Data

The explicit formulation seeks to minimize the square of the distance between the measured data points and the estimated planar model with respect to the plane at $Z = 0$ or the $XY-$plane as shown by the objective function in equation (2.53).

$$\epsilon(a, b, c) = \min_{a,b,c} \sum_{i=1}^{N} (aX_i + bY_i + c - Z_i)^2 \tag{2.53}$$

Note that, in contrast to equation (2.49), this planar model has explicit form $f(X, Y) = Z$. Minimization of this error seeks to estimate the Z-offset, c, and slope of the plane with respect to the x-axis, a, and y-axis, b.

To optimize this function, we re-write the objective function as a quadratic matrix-vector product by defining the vector $\alpha = [\ a \quad b \quad c\ ]^t$ as the vector of planar coefficients, the vector $\mathbf{b} = [\ Z_0 \quad Z_1 \quad \ldots \quad Z_N]^t$ which denotes the target depth values and the matrix $\mathbf{M}$ as the matrix of planar monomials formed from the 3D (X, Y, Z) surface data having i^{th} row $\mathbf{M}_i = [\ X_i \quad Y_i \quad 1\ ]$. Using this notation, the optimization function becomes:

$$\epsilon(\alpha) = \min_{\alpha} \left(\alpha^t \mathbf{M}^t \mathbf{M} \alpha - 2\alpha^t \mathbf{M}^t \mathbf{b} + \mathbf{b}^t \mathbf{b} \right)$$

Taking the derivative of the error function and setting it to zero provides:

$$\frac{d\epsilon(\alpha)}{d\alpha} = \mathbf{M}^t \mathbf{M} \alpha - \mathbf{M}^t \mathbf{b} = 0 \tag{2.54}$$

A solution to this system of equations is obtained via $\widehat{\alpha} = (\mathbf{M}^t\mathbf{M})^{-1} \mathbf{M}^t \mathbf{b}$. Again, $\mathbf{M}^t\mathbf{M}$ is a symmetric matrix and, for the monomials $\mathbf{M}_i = [\ X_i \quad Y_i \quad 1\ ]$, the elements of the matrix-vector product are

$$\mathbf{M}^t\mathbf{M} = \sum_{i=1}^{N} \begin{bmatrix} X_i^2 & X_iY_i & X_i \\ X_iY_i & Y_i^2 & Y_i \\ X_i & Y_i & 1 \end{bmatrix}, \mathbf{M}^t\mathbf{b} = \sum_{i=1}^{N} Z_i \begin{bmatrix} X_i \\ Y_i \\ 1 \end{bmatrix} \tag{2.55}$$

When surface measurements are normally distributed, explicit fitting methods perform similarly to implicit methods. However, there is bias associated with the explicit fitting objective function. Specifically, errors for explicit fits are are measured along the Z axis. This has the effect that, in contrast to the implicit fitting approach, the estimated coefficients are not Euclidean invariant. For this reason explicit fitting methods are less popular for point cloud data.

CHAPTER 3: DENSE VISUAL ODOMETRY

This chapter investigates the theory and implementation of a direct visual odometry algorithm leveraged by the SLAM approach discussed in later chapters. This chapter provides in depth analysis and presents a reference implementation for this odometry approach.

3.1 Direct Visual Odometry

Methods of visual odometry that utilize all incoming visual data are referred to as direct methods. This is in contrast to sparse or feature-based odometry approaches, where incoming data is processed to extract image features or key points, which are then matched between images. However, during this feature extraction process, sparse approaches leave much of the remaining image information to be discarded. The thought is by making use of all available information in the image, a more accurate estimate of camera motion can be attained.

Recently, a number of direct visual odometry methods based on the minimization of reprojection error have been shown to provide high accuracy camera tracking and the ability to run in real-time. Comport et al [14], detailed a method based on stereo images which minimizes intensity error. Steinbrücker et al. [15], minimize intensity error over RGBD image pairs and show that their method is more accurate than a state of the art ICP approach for small displacements. Tykkälä et al. [16], minimize both intensity and depth error over RGBD image pairs and show that this improves the estimation compared to minimizing intensity error alone. Kerl et al. [17, 18], also minimize both intensity and depth error for RGBD images, and improve robustness by incorporating motion priors and more sophisticated residual weighting. Kerl et al. also provide an optimized C++ implementation which they call Dense Visual Odometry (DVO). These methods are all variations and extensions of Lucas-Kanade image alignment [19], using alignment results between images for odometry.

3.2 Image Alignment

Fundamentally, the dense formulation of the motion estimation is an image alignment problem. That is, we seek the transformation that best explains differences between two images of the same scene taken at different points in time. Given two images and the camera motion between them, one can take the information (intensity, depth, or both) in one image, and through a series of projections and transformation, map these values into the viewpoint of the other image. In this case the difference between expected and observed values of these images should be zero if the motion of the camera is known exactly and sensor noise and other outside influences can be ignored. In reality this is not the case, however, the error metric described is a suitable metric for an optimization formulation that can be used to estimate this camera motion.

3.3 Formalisms

More formally, given an image with intensity and depth information (I_1, Z_1) and a second image taken at a later point in time (I_2, Z_2) we seek to estimate the set of transformation parameters ξ_e, that best aligns the images after a warp function $w(T, \mathbf{x})$ is applied. The notation here follows loosely the notation of Kerl et al. in [17]. Here, ξ_e is the 6x1 vector of twist parameters corresponding to the rigid body transformation being estimated (see section 2.4 for more detail). The warp function, $w(T, \mathbf{x})$, computes the warped pixel location in second image, $\mathbf{x}'$, of the image pixel, $\mathbf{x}$, from the first image given a transformation τ:

$$\mathbf{x}' = w(\tau_\xi, \mathbf{x}) = \rho(\tau_\xi(\rho^{-1}(\mathbf{x}, Z_1(\mathbf{x})))) \tag{3.1}$$

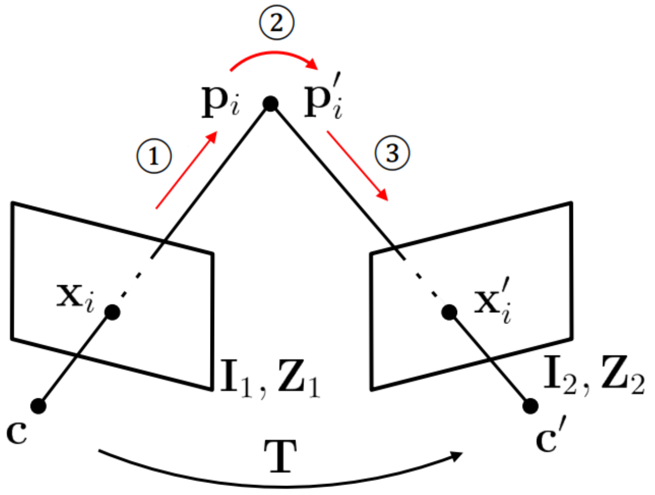


Figure 3.1: The warp function that map pixels between images through: (1) reconstruction (inverse projection with known depth), (2) transformation of the reconstructed point, and (3) projection projection of the transformed point into the other image. Figure modified from [20].

where the projection, ρ, and the transformation τ are discussed in Sections 2.1 and 2.4.3 respectively. Here, $\tau_\xi(\mathbf{p})$ is defined as

$$\tau_\xi(\mathbf{p}) = \tau(\mathbf{T}_\xi, \mathbf{p}) \tag{3.2}$$

where $\mathbf{T}_\xi$ is the 4x4 homogeneous transformation matrix corresponding to the twist parameters ξ. As shown in figure 3.1, the warp function maps pixel $\mathbf{x}$ to pixel $\mathbf{x}'$ through reconstruction of the 3D point ($\mathbf{x} \rightarrow p$), transformation ($p \rightarrow p'$), and projection ($p' \rightarrow \mathbf{x}'$).

3.4 Inverse Compositional Approach

Baker and Matthews in [21], discuss four major variations of the original Lucas-Kanade algorithm for iterative image alignment. The goal of Lucas-Kanade is to align a template image $T(\mathbf{x})$ to an input image $I(\mathbf{x})$ by least squares minimization as discussed heretofore. In terms of the intensity and depth images already defined, the template image corresponds to the second image and the input image corresponds to first image. Formally,

$$T_I(\mathbf{x}) = I_2(\mathbf{x}) \tag{3.3}$$

$$T_Z(\mathbf{x}) = Z_2(\mathbf{x}) \tag{3.4}$$

$$I_I(\mathbf{x}) = I_1(\mathbf{x}) \tag{3.5}$$

$$I_Z(\mathbf{x}) = Z_1'(\mathbf{x}) \tag{3.6}$$

They prove equivalence between all methods discussed but note that there are important considerations in regards to computational efficiency.

The approach they find to be most efficient is termed the inverse compositional approach. This is the approach most similar to the methods used by Kerl et al., and others. It is a *compositional* method as it estimates an incremental transformation that is then composed with the current estimate at every iteration, $\xi \leftarrow \xi \circ \Delta\xi$, unlike the more traditional additive approach where the increment is added to the estimate, $\xi \leftarrow \xi + \Delta\xi$. This leads to the linearizion being performed about $\Delta\xi = 0$ (the identity transformation), which simplifies the Jacobian evaluation, allowing it to be precomputed.

This approach is deemed an *inverse* approach as the roles of template image and input image in the traditional Lucas-Kanade formulation are reversed. Simply put, the estimated transformation that would move to align the template image to the input image at each iteration is computed, but instead of applying this transformation on the template image, we compute the inverse and transform the input image in its stead. This leads to a number of advantages compared to other methods. First the image gradients of the template image do not change, and therefore only need to be computed once, instead of at every iteration. This means the gradient vector contributions for each pixel can also be precomputed and therefore the Gauss-Newton approximation to the Hessian.

3.4.1 Minimization

Using this formulation, the error function can be written as

$$\begin{aligned} f(\xi) &= \frac{1}{2}\sum_{\mathbf{x}}(T(w(\tau_\xi, w(\tau_{\Delta\xi}, \mathbf{x}))) - I(\mathbf{x}))^2 \\ &= \frac{1}{2}\sum_{\mathbf{x}} r(\mathbf{x})^2 \end{aligned} \tag{3.7}$$

Note that the warp function defined in equation (3.1) is in a sense the opposite from the one used by Baker and Matthews. In [21], the warp function maps pixel values in the template image to pixel values in the input image, while in this case the warp function maps pixel values in the input image to pixel values in the template image. Note also that the error term above is nonlinear, and we therefore find ourselves in the midst of a nonlinear least squares problem. As discussed in section 2.5.1, such problems can often be solved by the method of Gauss-Newton. Taking the derivative and equating to zero yields

$$\frac{\partial f(\xi)}{\partial \xi} = \sum_{\mathbf{x}} r(\mathbf{x})\frac{\partial r(\mathbf{x})}{\partial \xi} = 0 \tag{3.8}$$

after linearization about $\Delta\xi = 0$, this becomes

$$\sum_{\mathbf{x}} \frac{\partial r_o(\mathbf{x})}{\partial \xi}(r_o(\mathbf{x}) + \frac{\partial r_o(\mathbf{x})}{\partial \xi}\Delta\xi) = 0 \tag{3.9}$$

$$\sum_{\mathbf{x}} J_{\mathbf{x}}(0)^T(r_o(\mathbf{x}) + J_{\mathbf{x}}(0)\Delta\xi)) = 0 \tag{3.10}$$

where

$$r_o(\mathbf{x}) = T(w(\tau_\xi, \mathbf{x})) - I(\mathbf{x}) \tag{3.11}$$

$$J_{\mathbf{x}}(0) = \frac{\partial r_o(\mathbf{x})}{\partial \xi} = \frac{\partial}{\partial \xi} T(w(\tau_\xi, \mathbf{x})) - 0 = \nabla T(\mathbf{x})^T \frac{\partial w}{\partial \xi}|_{\xi=0} = \nabla T(\mathbf{x})^T J_w(\mathbf{x}, 0) \quad (3.12)$$

Here $J_w(0) = \frac{\partial w}{\partial \xi}|_{\xi=0}$ is derivative of the warp function evaluated at the identity. The warp is evaluated here as the template image never moves. Then, equation (3.10) after simplification is given by

$$\sum_{\mathbf{x}} J_{\mathbf{x}}(0)^T J_{\mathbf{x}}(0) \Delta\xi = -\sum_{\mathbf{x}} J_{\mathbf{x}}(0)^T r_o(\mathbf{x}) \quad (3.13)$$

where the Gauss-Newton approximation to the Hessian is given by

$$\mathbf{H} = \sum_{\mathbf{x}} J_{\mathbf{x}}(0)^T J_{\mathbf{x}}(0) = \sum_{\mathbf{x}} \left[\nabla T(\mathbf{x})^T J_w(\mathbf{x}, 0)\right]^T \left[\nabla T(\mathbf{x})^T J_w(\mathbf{x}, 0)\right] \quad (3.14)$$

Finally, the parameter update is then given by

$$\Delta\xi = -H^{-1} \sum_{\mathbf{x}} J_{\mathbf{x}}(0)^T r_o(\mathbf{x}) \quad (3.15)$$

$$= -H^{-1} \sum_{\mathbf{x}} \left[\nabla T(\mathbf{x})^T J_w(\mathbf{x}, 0)\right]^T (T(w(\tau_\xi, \mathbf{x})) - I(\mathbf{x})) \quad (3.16)$$

Once solved for, the transformation parameters are then updated via composition, $\xi \leftarrow \xi \circ (\Delta\xi)^{-1}$.

3.4.2 Algorithm Outline

The inverse compositional approach is outlined below. This solves the nonlinear least squares problem as discussed in section 2.5.1.

- Precomputation Stage
  - Compute the image gradients of the template image, $\nabla T_I(\mathbf{x}) = \nabla I_2$ for intensity, or $\nabla T_Z(\mathbf{x}) = \nabla Z_2$ for depth

  - Evaluate the Jacobian mapping the image (x, y) gradients to the space of the parameters at the identity, $J_w(\mathbf{x})$
  - Compute the gradient vectors for each residual $J(\mathbf{x}) = \nabla T(\mathbf{x})^T J_w(\mathbf{x}, 0)$
  - Compute the Gauss-Newton Hessian approximation, $\mathbf{H} = \sum_{\mathbf{x}} J(\mathbf{x})^T J(\mathbf{x})$

- Iteration
  - Compute the residuals $r_I(\mathbf{x}) = T_I(w(\tau_\xi, \mathbf{x})) - I_I(\mathbf{x})$ for intensity, or $r_Z(\mathbf{x}) = T_Z(w(\tau_\xi, \mathbf{x})) - I'_Z(\mathbf{x})$ for depth
  - Compute the sum of the residual weighted gradient vectors, $\sum_{\mathbf{x}} J(\mathbf{x})^T r(\mathbf{x})$
  - Compute the parameter update, $\Delta\xi = -\mathbf{H}^{-1} \sum_{\mathbf{x}} J(\mathbf{x})^T r(\mathbf{x})$
  - Update the parameters through composition with inverse transformation of the update, $\xi \leftarrow \xi \circ (\Delta\xi)^{-1}$

The parameter update step is performed by computing the homogeneous transformation matrices of ξ and $\Delta\xi$ via the exponential map, and the update is applied via *premultiplication* with the current parameters as shown in equation (3.17)

$$\hat{\xi} \leftarrow \log((e^{\hat{\Delta\xi}})^{-1} \cdot e^{\hat{\xi}}) \tag{3.17}$$

where $\hat{\xi}$ is the twist matrix with corresponding parameters ξ, and $\log(\cdot)$ is the log map as discussed in section 2.4.3.

Note, in practice not all of the techniques for computational savings discussed in section 3.4 are applied since this would require interpolation over these precomputed gradient vectors. We find this interpolation leads to a degradation in the convergence and accuracy of the estimation. Therefore, computational savings that one might afford from some precomputation strategies are offset by degraded convergence characteristics.

3.5 Error Functions

A variety of error functions can now be formulated to minimize errors in intensity, depth, or both. To simplify notation, let

$$R_{I,i} = I_2(w(\tau_\xi, w(\tau_{\Delta\xi}, \mathbf{x}_i))) - I_1(\mathbf{x}_i) \tag{3.18}$$

$$R_{Z,i} = Z_2(w(\tau_\xi, w(\tau_{\Delta\xi}, \mathbf{x}_i))) - Z_1'(\mathbf{x}_i) \tag{3.19}$$

be the i^{th} residual in intensity and depth respectively, and the vectors $\mathbf{R}_I$ and $\mathbf{R}_Z$ represent the $N \times 1$ vectors of intensity and depth residuals. To minimize intensity error alone,

$$\xi_I = \arg\min_\xi \sum_{i=1}^{N} R_{I,i}^2 = \arg\min_\xi (\mathbf{R}_I^T \mathbf{R}_I) \tag{3.20}$$

similarly, to minimize depth error alone,

$$\xi_Z = \arg\min_\xi \sum_{i=1}^{N} R_{Z,i}^2 = \arg\min_\xi (\mathbf{R}_Z^T \mathbf{R}_Z) \tag{3.21}$$

The simultaneous minimization of intensity and depth errors can be accomplished in a variety of ways. The most simple would be a minimization over the weighted linear combination of the residuals such as,

$$\xi_{I,Z} = \arg\min_\xi \sum_{i=1}^{N} R_{Z,i}^2 + \lambda^2 R_{Z,i}^2 = \arg\min_\xi (\mathbf{R}_I^T \mathbf{R}_I + \lambda^2 \mathbf{R}_Z^T \mathbf{R}_Z) \tag{3.22}$$

Tykkälä et al. in [16], discuss various methods for choosing an appropriate value for the weight λ. The choice of this parameter has a direct effect on the minimization since it steers the reduction of the error towards depth or intensity depending on its value. This value must strike a balance between errors in depth, which can small or on the order of a few meters, and errors in intensity which can be on the order of

0-255 and have a less clear geometric interpretation.

Of course, other more sophisticated cost functions can be used for the minimization. For example, Kerl et al. in [17], dynamically weight residuals by a scaling matrix based on the covariance of the intensity and depth residuals. Gutiérrez-Gómez et al. in [22], parameterize the depth residuals using an inverse depth metric. These authors also weight all residuals based on how well the errors fit a given distribution, such as the Tukey, Huber, or Student's t-distribution. This is performed in an effort to remove outlier residuals and improve the robustness of the estimator. Babu et al. in [20] extend DVO by using a known noise model for depth estimates to weight depth residuals by their uncertainty, resulting in more accurate odometry.

3.6 Minimization

Suppose we wish to minimize over the linear combination of intensity and depth residuals, as in equation (3.22):

$$\xi_{I,Z} = \arg\min_{\xi} \frac{1}{2}\sum_{i=1}^{N}(R_{I,i}^2 + \lambda^2 R_{Z,i}^2) = \arg\min_{\xi}(\frac{1}{2}(\mathbf{R}_I^T\mathbf{R}_I + \lambda^2\mathbf{R}_Z^T\mathbf{R}_Z))$$

We note the error function for this problem is given the linear combination of error functions

$$f(\xi) = \frac{1}{2}\sum_{i=1}^{N} R_{I,i}^2 + \frac{1}{2}\sum_{i=1}^{N}(\lambda R_{Z,i})^2 \tag{3.23}$$

we can then linearize and solve these individually as in section 3.4.1. The solution has the form

$$\sum_{i=1}^{N}\left(J_{I,i}^T J_{I,i} + J_{Z,i}^T J_{Z,i}\right)\Delta\xi = -\sum_{i=1}^{N}\left(J_{I,i}^T r_{I,i} + \lambda J_{Z,i}^T r_{Z,i}\right) \tag{3.24}$$

where the linearized residuals in intensity and depth respectively for a given pixel, $\mathbf{x}$, are be defined as

$$r_{lin,I} = r_I + J_I \Delta\xi \tag{3.25}$$

$$r_{lin,Z} = r_Z + J_Z \Delta\xi \tag{3.26}$$

and

$$r_I = I_2(w(\tau_\xi, \mathbf{x})) - I_1(\mathbf{x}) \tag{3.27}$$

$$r_Z = Z_2(w(\tau_\xi, \mathbf{x})) - Z_1'(\mathbf{x}) \tag{3.28}$$

where $Z_1'(\mathbf{x}) = [\tau_\xi(\rho^{-1}(\mathbf{x}, Z_1(\mathbf{x})))]_Z$, and the operator $[\cdot]_Z$ extracts the third component (Z) from the point $\mathbf{p}$. The intensity error is then the difference between I_2 evaluated at the warped pixel location, and I_1 evaluated at the pixel $\mathbf{x}$. The depth error is then the difference between Z_2 evaluated at the warped pixel location, and the Z value of the point reconstructed from Z_1 at pixel location $\mathbf{x}$ after being transformed by the transformation corresponding to the values of ξ. The image, Z_1' is considered constant at each iteration and therefore does not depend on the values of the transformation.

In order to solve for the parameter update, $\Delta\xi$, as in equation (2.39), we must compute the Jacobian of the residuals, $\mathbf{J}(\theta_k)$, whose rows correspond to the gradient of the i^{th} residual with respect to the transformation parameters, ξ. That is, we must provide a mapping from the space of the residuals into the space of the parameters, ξ. However, we note that we have two types of residual, in both intensity and depth, which impose their constraints on the minimization in different ways. Thus, we can expect that the gradient vectors for these residuals will differ. The gradient vector for a given intensity residual is then

$$\begin{aligned} J_{I,i} &= \frac{\partial r_{I,i}}{\partial \xi} = \frac{\partial}{\partial \xi}(I_2(w(\tau_\xi, \mathbf{x})) - I_1(\mathbf{x})) \\ &= \frac{\partial}{\partial \xi} I_2(w(\tau_\xi, \mathbf{x})) - 0 \\ &= \left[\frac{\partial I_2}{\partial \rho}\right]_{1\times 2} \left[\frac{\partial \rho}{\partial \tau}\frac{\partial \tau}{\partial \mathbf{T}}\frac{\partial \mathbf{T}}{\partial \xi}\right]_{2\times 6} = J_{I_2} J_\rho J_\tau J_{\mathbf{T}} = J_{I_2} J_w \end{aligned} \tag{3.29}$$

and the gradient vector for a given depth residual is

$$\begin{aligned} J_{Z,i} &= \frac{\partial r_{Z,i}}{\partial \xi} = \frac{\partial}{\partial \xi}(Z_2(w(\tau_\xi, \mathbf{x})) - Z_1'(\mathbf{x})) \\ &= \frac{\partial}{\partial \xi} Z_2(w(\tau_\xi, \mathbf{x})) - 0 \\ &= \begin{bmatrix} \frac{\partial Z_2}{\partial x} & \frac{\partial Z_2}{\partial y} & \frac{\partial Z_2}{\partial Z} \end{bmatrix} \begin{bmatrix} \frac{\partial \rho}{\partial \tau}\frac{\partial \tau}{\partial \mathbf{T}}\frac{\partial \mathbf{T}}{\partial \xi} \\ \left[\frac{\partial \tau}{\partial \mathbf{T}}\frac{\partial \mathbf{T}}{\partial \xi}\right]_Z \end{bmatrix}_{3\times 6} = \begin{bmatrix} \frac{\partial Z_2}{\partial \rho} & \frac{\partial Z_2}{\partial Z} \end{bmatrix}_{1\times 3} \begin{bmatrix} J_\rho J_\tau J_{\mathbf{T}} \\ [J_\tau J_{\mathbf{T}}]_Z \end{bmatrix}_{3\times 6} \end{aligned} \tag{3.30}$$

$$= \begin{bmatrix} J_{Z_2} & \frac{\partial Z_2}{\partial Z} \end{bmatrix}_{1\times 3} \begin{bmatrix} J_w \\ [J_\tau J_{\mathbf{T}}]_Z \end{bmatrix}_{3\times 6} \tag{3.31}$$

where $[J_\tau J_T]_Z$ extracts the 3rd row of the product of the Jacobians, $J_\tau J_T$. The derivative $\frac{\partial Z_2}{\partial Z}$ is found by expressing $Z_2(\mathbf{x}) = Z_2(x, y)$ implicitly as

$$z_o - Z_2(x, y) = 0 \tag{3.32}$$

and taking the derivative with respect to Z

$$\frac{\partial}{\partial Z} z_o - \frac{\partial}{\partial Z} Z_2(x, y) = 0 - 1 \tag{3.33}$$

$$\frac{\partial}{\partial Z} Z_2(x, y) = -1 \tag{3.34}$$

Equation (3.30) can then be written as

$$J_{Z,i} = \begin{bmatrix} J_{Z_2} & -1 \end{bmatrix}_{1\times 3} \begin{bmatrix} J_w \\ [J_\tau J_{\mathbf{T}}]_Z \end{bmatrix}_{3\times 6} = J_{Z_2} J_w - [J_\tau J_{\mathbf{T}}]_Z \tag{3.35}$$

The Jacobians J_{I_2} and J_{Z_2} are of size 1×2 and are the (x, y) image gradients in the second intensity image, I_2, and the second depth image, Z_2, respectively. Formally,

$$J_{I_2} = \frac{\partial I_2}{\partial \rho}|_{\mathbf{x}=\rho(\tau(\mathbf{T}(0),\mathbf{p}_i))=\mathbf{x}_i} = \begin{bmatrix} \frac{\partial I_2}{\partial x} & \frac{\partial I_2}{\partial y} \end{bmatrix} \tag{3.36}$$

and

$$J_{Z_2} = \frac{\partial Z_2}{\partial \rho}|_{\mathbf{x}=\rho(\tau(\mathbf{T}(0),\mathbf{p}_i))=\mathbf{x}_i} = \begin{bmatrix} \frac{\partial Z_2}{\partial x} & \frac{\partial Z_2}{\partial y} \end{bmatrix} \tag{3.37}$$

The product $J_w = J_\rho J_\tau J_{\mathbf{T}}$ is the Jacobian of the warp function, which is the product of the Jacobians of the projection, ρ, the transformation τ, and the exponential map which maps the homogeneous transformation into the exponential coordinates of the parameters. This product maps values in the space of the (x, y) image gradients to the space of the parameters ξ. J_ρ can be derived by taking the projection mapping as a 2-vector and differentiating with respect to the point coordinates (X, Y, Z),

$$J_\rho = \frac{\partial \rho}{\partial \tau}|_{\mathbf{p}=\tau(\mathbf{T}(0),\mathbf{p}_i)=\mathbf{p}_i} = \nabla_{\mathbf{p}} \begin{bmatrix} f_x \frac{X}{Z} + c_x \\ f_y \frac{Y}{Z} + c_y \end{bmatrix} = \begin{bmatrix} \frac{f_x}{Z} & 0 & -f_x \frac{X}{Z^2} \\ 0 & \frac{f_y}{Z} & -f_y \frac{Y}{Z^2} \end{bmatrix} \tag{3.38}$$

as the result of transformation at the identity, $\tau(\mathbf{T}(0), \mathbf{p}_i)$, is no motion.

The product $J_\tau J_{\mathbf{T}}$ is a 3×6 matrix corresponding to the Jacobian of a point under transformation, mapped to the space of the transformation parameters, ξ, through the exponential map. J_τ can then be found by taking the gradient of the transformation mapping, $\mathbf{p}' = \tau(\mathbf{T}, \mathbf{p}) = \mathbf{R}\mathbf{p} + \mathbf{t}$, with respect to the 12 parameters of the homogeneous transformation, $[\ r_{11},\ r_{21},\ r_{31},\ r_{12},\ r_{22},\ r_{32},\ r_{13},\ r_{23},\ r_{33},\ t_X,\ t_Y,\ t_Z\ ]^T$,

$$J_\tau = \frac{\partial \tau(\mathbf{T}, \mathbf{p})}{\partial \mathbf{T}}|_{\mathbf{T}=\mathbf{T}(0),\mathbf{p}=\mathbf{p}_i} = \nabla_{\mathbf{T}}\mathbf{p}' = \nabla_{\mathbf{T}} \begin{bmatrix} r_{11}X + r_{12}Y + r_{13}Z + t_X \\ r_{21}X + r_{22}Y + r_{23}Z + t_Y \\ r_{31}X + r_{32}Y + r_{33}Z + t_Z \end{bmatrix}$$

$$= \begin{bmatrix} X & 0 & 0 & Y & 0 & 0 & Z & 0 & 0 & 1 & 0 & 0 \\ 0 & X & 0 & 0 & Y & 0 & 0 & Z & 0 & 0 & 1 & 0 \\ 0 & 0 & X & 0 & 0 & Y & 0 & 0 & Z & 0 & 0 & 1 \end{bmatrix} \tag{3.39}$$

J_T is then found by taking the gradient of the 12 transformation parameter with respect to the twist coordinates. This is performed taking the 12 stacked elements of $\mathbf{T}$ and taking the gradient with respect to the components of ξ at the identity ($\xi = 0$). This is given by

$$J_{\mathbf{T}} = \frac{\partial \mathbf{T}}{\partial \xi}|_{\xi=0} = \nabla_\xi \mathbf{T} = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \tag{3.40}$$

To compute the derivatives of the rotation matrix entries with respect to the rotation vector, Gallego and Yezzi in [5] show that they are given by the skew-symmetric generators of the rotation group. They provide a formula for these derivatives in terms of the rotation vector components. When written symbolically and evaluated at the identity rotation, indeed the generators of rotation group are observed. The transpose of these 3×3 skew-symmetric generators are found stacked in the upper 9×3 block of $J_{\mathbf{T}}$.

The product, $J_\tau J_{\mathbf{T}}$, is then given by

$$J_\tau J_{\mathbf{T}} = \begin{bmatrix} 1 & 0 & 0 & 0 & Z & -Y \\ 0 & 1 & 0 & -Z & 0 & X \\ 0 & 0 & 1 & Y & -X & 0 \end{bmatrix} \tag{3.41}$$

and the product, $J_w = J_\rho J_\tau J_{\mathbf{T}}$, is then given by

$$J_w = J_\rho J_\tau J_{\mathbf{T}} = \begin{bmatrix} f_x \frac{1}{Z} & 0 & -f_x \frac{X}{Z^2} & -f_x \frac{X \cdot Y}{Z^2} & f_x(1 + \frac{X^2}{Z^2}) & -f_x \frac{Y}{Z} \\ 0 & f_y \frac{1}{Z} & -f_y \frac{Y}{Z^2} & -f_y(1 + \frac{Y^2}{Z^2}) & f_y \frac{X \cdot Y}{Z^2} & f_y \frac{X}{Z} \end{bmatrix} \tag{3.42}$$

Finally, the gradient vectors for a given intensity and depth are respectively

$$\begin{aligned} J_{I,i} &= J_{I_2} J_w = J_{I_2} J_\rho J_\tau J_{\mathbf{T}} \\ &= \begin{bmatrix} \frac{\partial I_2}{\partial x} & \frac{\partial I_2}{\partial y} \end{bmatrix} \begin{bmatrix} f_x \frac{1}{Z} & 0 & -f_x \frac{X}{Z^2} & -f_x \frac{X \cdot Y}{Z^2} & f_x(1 + \frac{X^2}{Z^2}) & -f_x \frac{Y}{Z} \\ 0 & f_y \frac{1}{Z} & -f_y \frac{Y}{Z^2} & -f_y(1 + \frac{Y^2}{Z^2}) & f_y \frac{X \cdot Y}{Z^2} & f_y \frac{X}{Z} \end{bmatrix} \end{aligned} \tag{3.43}$$

$$J_{Z,i} = \begin{bmatrix} \frac{\partial Z_2}{\partial x} & \frac{\partial Z_2}{\partial y} & -1 \end{bmatrix} \begin{bmatrix} J_w \\ [J_\tau J_{\mathbf{T}}]_Z \end{bmatrix}_{3\times 6}$$

$$= \begin{bmatrix} \frac{\partial Z_2}{\partial x} & \frac{\partial Z_2}{\partial y} & -1 \end{bmatrix} \begin{bmatrix} f_x \frac{1}{Z} & 0 & -f_x \frac{X}{Z^2} & -f_x \frac{X \cdot Y}{Z^2} & f_x(1 + \frac{X^2}{Z^2}) & -f_x \frac{Y}{Z} \\ 0 & f_y \frac{1}{Z} & -f_y \frac{Y}{Z^2} & -f_y(1 + \frac{Y^2}{Z^2}) & f_y \frac{X \cdot Y}{Z^2} & f_y \frac{X}{Z} \\ 0 & 0 & 1 & Y & -X & 0 \end{bmatrix}$$

$$= J_{Z_2} J_w - [J_\tau J_T]_Z$$

$$= \begin{bmatrix} \frac{\partial Z_2}{\partial x} & \frac{\partial Z_2}{\partial y} \end{bmatrix} \begin{bmatrix} f_x \frac{1}{Z} & 0 & -f_x \frac{X}{Z^2} & -f_x \frac{X \cdot Y}{Z^2} & f_x(1 + \frac{X^2}{Z^2}) & -f_x \frac{Y}{Z} \\ 0 & f_y \frac{1}{Z} & -f_y \frac{Y}{Z^2} & -f_y(1 + \frac{Y^2}{Z^2}) & f_y \frac{X \cdot Y}{Z^2} & f_y \frac{X}{Z} \end{bmatrix} - \begin{bmatrix} 0 & 0 & 1 & Y & -X & 0 \end{bmatrix}$$

Note the gradient vector for a depth residual given in equation (3.30) is of similar form to that of the intensity residual, but subtracts away the 3rd row or the Z component of $J_\tau J_{\mathbf{T}}$. This characterizes a change in the transformation parameters for a motion in the Z direction. This occurs because the nature of the values in depth images, which impose this additional geometric constraint on the minimization. For example, consider moving a camera along its optical axis while taking images. The change in the values of the depth image for this motion will be quite different from the change in the values of the corresponding intensity image. If moving towards, say, a large planar surface with little texture, the change in the intensity image will be almost zero, while the values of the depth image will change according to the camera motion.

We can now solve for the Gauss-Newton parameter update

$$\sum_{i=1}^{n}(J_{I,i}^T J_{I,i} + J_{Z,i}^T J_{Z,i})\Delta\xi = -\sum_{i=1}^{n}(J_{I,i}^T r_{I,i} + \lambda J_{Z,i}^T r_{Z,i}) \tag{3.44}$$

or equivalently over the pixels, $\mathbf{x}$,

$$\sum_{\mathbf{x}}(J_I(\mathbf{x})^T J_I(\mathbf{x}) + J_Z(\mathbf{x})^T J_Z(\mathbf{x}))\Delta\xi = -\sum_{\mathbf{x}}(J_I(\mathbf{x})^T r_Z(\mathbf{x}) + \lambda J_Z(\mathbf{x})^T r_Z(\mathbf{x})) \tag{3.45}$$

If we let $\mathbf{J}$ represent the stacked matrix of intensity and depth gradients

$$\mathbf{J} = \begin{bmatrix} \mathbf{J}_I \\ \mathbf{J}_Z \end{bmatrix} \tag{3.46}$$

and $\mathbf{r}_I, \mathbf{r}_Z$ as the vectors of intensity and depth residuals, then we may write in matrix form

$$\mathbf{J}^T\mathbf{J}\Delta\xi = -\mathbf{J}^T \begin{bmatrix} \mathbf{r}_I \\ \mathbf{r}_Z \end{bmatrix} \tag{3.47}$$

and the parameter update is given by

$$\Delta\xi = -(\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T \begin{bmatrix} \mathbf{r}_I \\ \lambda\mathbf{r}_Z \end{bmatrix} \tag{3.48}$$

3.7 Implementation

A reference implementation for this odometry approach was developed which minimizes a linear combination of intensity and depth error in the spirit of the inverse compositional approach as discussed so far. The algorithm is outlined in Algorithm 3.1. This reference implementation was written in C++ and achieves a framerate of

~25 frames per second. In an effort to increase efficiency, lines 14-26 in Algorithm 3.1 are performed in parallel. We also adopt a coarse-to-fine approach for the estimation, where the motion is estimated starting with downsampled versions of the input images and the estimate is successively refined at each level, i.e. with increasing resolution. This approach not only increases the robustness of the estimation to larger camera displacements, it also aids in convergence as the initial estimate for the motion at a given resolution has already been estimated from the previous level.

Note there are several implementation differences in this version when compared to the algorithm outline described in section 3.4.1. First, note that the gradient vectors and Hessian matrix are computed at every iteration in this implementation-they are not precomputed. As previously noted, we find that precomputation of these entities later requires them to be interpolated, and error introduced at this step leads to degradation in the convergence and accuracy of the estimation. Also note that the parameter update step is performed by computing the homogeneous transformation matrices of ξ and $\Delta\xi$ via the exponential map, and the update is applied via *premultiplication* with the current parameters as shown in equation (3.49)

$$\hat{\xi} \leftarrow \log((e^{\hat{\Delta\xi}})^{-1} \cdot e^{\hat{\xi}}) \tag{3.49}$$

where $\hat{\xi}$ is the twist matrix with corresponding parameters ξ, and $\log(\cdot)$ is the log map as discussed in section 2.4.3. As written in Baker and Matthews in [21], this update could be interpreted as a postmultiplication.

Due to its accuracy and speed, this odometry approach seems to be an appropriate candidate for use as part of a SLAM system. Incorporation of this approach into a full SLAM system can provide accurate low drift camera pose tracking. This is crucial for the successful detection, tracking, and integration of objects and object subcomponents as part of a more sophisticated recognition system.

Algorithm 3.1 Odometry Estimation from RGBD Image Pair

```
Require: $(I_1, Z_1)$ and $(I_2, Z_2)$ are valid RGBD images
 1: function EstimateOdometry($I_1, Z_1, I_2, Z_2, \xi_o$)
 2:     $\nabla I_2 \leftarrow$ gradient($I_2$)                    ▷ $(x, y)$ image gradient of $\nabla I_2$
 3:     $\nabla Z_2 \leftarrow$ gradient($Z_2$)                    ▷ $(x, y)$ image gradient of $\nabla Z_2$
 4:     $C_1 \leftarrow$ reconstruct($Z_1$)                        ▷ compute pointcloud of $Z_1$
 5:     $\xi \leftarrow \xi_o$
 6:     $\Delta\xi \leftarrow \mathbf{0}$
 7:     $e \leftarrow 0$
 8:     $e_{last} \leftarrow \infty$
 9:     $k \leftarrow 0$
10:     while $e_{last} - e > \epsilon$ and $k \leq k_{max}$ do
11:         $n \leftarrow 0$
12:         $\mathbf{r} \leftarrow [\,]$                               ▷ vector of residual values
13:         $\mathbf{J} \leftarrow [\,]$                               ▷ matrix of gradient vectors
14:         for $\mathbf{p}$ in $C_1$ do
15:             $\mathbf{x} \leftarrow$ pixel($\mathbf{p}$)               ▷ get pixel corresponding to $p$
16:             $\mathbf{p}' \leftarrow$ transform($\mathbf{p}$, $\xi$)
17:             $\mathbf{p}_{im} \leftarrow$ project($\mathbf{p}'$)
18:             if $\mathbf{p}_{im}$ in image bounds then
19:                 $i \leftarrow$ interpolate($I_2$, $\mathbf{p}_{im}$)
20:                 $z \leftarrow$ interpolate($Z_2$, $\mathbf{p}_{im}$)
21:                 $\mathbf{i}_\nabla \leftarrow$ interpolate($\nabla I_2$, $\mathbf{p}_{im}$)
22:                 $\mathbf{z}_\nabla \leftarrow$ interpolate($\nabla Z_2$, $\mathbf{p}_{im}$)
23:                 $r_I \leftarrow i - I_1(\mathbf{x})$                  ▷ intensity residual
24:                 $r_Z \leftarrow z - \mathbf{p}'_z$                     ▷ depth residual
25:                 $\mathbf{J}_I \leftarrow$ intensityJacobian($\mathbf{p}$, $\mathbf{i}_\nabla$)   ▷ intensity gradient vector
26:                 $\mathbf{J}_Z \leftarrow$ depthJacobian($\mathbf{p}$, $\mathbf{z}_\nabla$)       ▷ depth gradient vector
27:                 $\mathbf{r}$.append($r_I$, $\lambda \cdot r_Z$)
28:                 $\mathbf{J}$.append($\mathbf{J}_I$, $\mathbf{J}_Z$)
29:                 $n \leftarrow n + 1$
30:             end if
31:         end for
32:         $e \leftarrow \mathbf{r}^T\mathbf{r}$
33:         if $e < e_{last}$ and $n \geq 6$ then
34:             $\xi \leftarrow \xi \circ (\Delta\xi)^{-1}$               ▷ update $\xi$ if error decreased
35:             $\Delta\xi \leftarrow (\mathbf{J}^T\mathbf{J})^{-1}\mathbf{J}^T\mathbf{r}$   ▷ solve for increment
36:         else break
37:         end if
38:         $k \leftarrow k + 1$
39:     end while
40:     return $\xi^{-1}$
41: end function
```

CHAPTER 4: STATE OF THE ART IN DENSE SLAM

This chapter provides a background on graph SLAM, dense SLAM, and details the specific dense SLAM implementation (DVO SLAM) to be extended.

4.1 Simultaneous Localization and Mapping

Simultaneous Localization and Mapping (SLAM) is a fundamental problem in mobile robotics. The SLAM problem involves the simultaneous construction of a map of the environment as well as localization within this map. An *agent*, e.g. robot, camera, etc., capable of motion generates measurements of its own movement as well as observations of map features. These measurements are used to estimate both the agent pose as well as the pose of detected map features (often called landmarks). Landmarks are used to better estimate the agent pose, which leads to more accurate landmark estimates, and so on. Over time, a successful SLAM system can construct an accurate environmental map and estimate the pose of the agent as it moves through the environment.

4.1.1 Graph SLAM

Graph SLAM approaches model the SLAM problem as a sparse graph, and seek to estimate the *full* trajectory of the agent, i.e. the sequence of poses $\mathbf{x}_{1:T}$, along with a map, $\mathbf{m}$, of the environment. This is also known as the *full* SLAM problem, which estimates all previous states of the agent trajectory. This is in contrast to filtering based approaches, which seek to solve the incremental or on-line SLAM problem, whereby previous state estimates are marginalized out.

As the agent moves through the unknown scene it may generate estimates of its movement from odometry estimates or using a motion model. It may also observations of features or landmarks in the environment. The full SLAM problem is then formulated as the estimation of the posterior probability of the trajectory, $\mathbf{x}_{1:T}$, and the environmental map, $\mathbf{m}$, given a sequence of motion estimates, $\mathbf{u}_{1:T}$, landmark measurements, $\mathbf{z}_{1:T}$, and the initial pose x_o as in equation (4.1).

$$p(\mathbf{x}_{1:T}, \mathbf{m} | \mathbf{u}_{1:T}, \mathbf{z}_{1:T}, x_o) \tag{4.1}$$

Modeling the full SLAM problem in a graphical way leads to an optimization problem over a sum of nonlinear quadratic constraints in the graph. These constraints are built up over time as the agent explores the environment in the form of both motion and measurement constraints. In this formulation, agent and landmark poses correspond to graph vertices, while the edges represent spatial constraints (transformations) between these vertices as a Gaussian distribution.

Motion estimates form a constraint directly between agent poses, while landmark measurements form constraints between agent poses through the mutual observation of a landmark. Individual measurements therefore link poses to the landmarks in the graph, and during the optimization these measurements are then mapped to constraints between poses that observed the same landmark at different points in time.

Once the graph is constructed, we seek the configuration of the graph vertices that best satisfies the set of constraints. That is, we seek a Gaussian approximation of the posterior distribution of Equation (4.1). The optimal trajectory, $\mathbf{x}^{\star}$, is estimated by minimizing over the joint log-likelihood of all constraints as described by equation (4.2). [23]

$$\mathbf{x}^\star = \operatorname*{argmin}_{\mathbf{x}} \Bigg(x_o^T \Omega_o x_o + \sum_t [x_t - g(u_t, x_{t-1})]^T R_t^{-1} [x_t - g(u_t, x_{t-1})] + \sum_t [z_t - h(x_t, m_i)]^T Q_t^{-1} [z_t - h(x_t, m_i)] \Bigg) \tag{4.2}$$

The leftmost term, $x_o^T \Omega_o x_o$, represents our prior knowledge of the initial pose. Often, x_o, is set to zero, anchoring the beginning of the trajectory to the global coordinate system origin.

The middle term describes a sum over the motion constraints, where $g(u_t, x_{t-1})$ is a function that computes the pose resulting from applying the odometry estimate obtained at time t, u_t, to the pose x_{t-1}. The residual vector, $x_t - g(u_t, x_{t-1})$, is then the difference between the expected pose, x_t, and the pose observed by applying the motion estimate. The matrix R_t^{-1} is the information matrix or inverse covariance matrix associated with the motion constraint at time t. This matrix encodes the precision or certainty of the measurement, and serves to weight the constraint appropriately in the optimization.

The rightmost term describes a sum over landmark constraints. The function $h(x_t, m_i)$ computes a global pose estimate of a given landmark, m_i, using a local measurement of the landmark and the pose of the agent, x_t, when the landmark was observed. The residual $z_t - h(x_t, m_i)$, is then the difference between the expected landmark pose in the global coordinate system, and the estimated global landmark pose resulting from the local landmark measurement. As before, the matrix Q_t^{-1} is the information matrix or inverse covariance matrix associated with the landmark constraint at time t.

4.1.1.1 Minimization

Minimization of this error function is a nonlinear least squares problem, and can certainly be solved using one of the methods described in section 2.5.1. It can also be solved efficiently by exploiting the sparse structure of the graph SLAM formulation [24], thereby allowing the use of sparse methods such as sparse Cholesky decomposition or the method of conjugate gradient. Many graph optimization libraries, such as the g^{2}o library [25], leverage these sparse methods in order to quickly optimize large graphs.

4.2 Visual SLAM Overview

Visual SLAM refers to the class of solutions that leverage vision sensors such as conventional cameras and, more recently, systems capable of detecting 3D geometry such as stereo camera pairs, LiDAR (Light Distance and Ranging) sensors, and RGBD sensors. SLAM systems that leverage RGBD sensors are particularly popular as they provide both surface color and surface geometry information at every pixel location.

4.2.1 Pose Graph (Keyframe) SLAM

Visual graph SLAM solutions typically optimize over the poses of the vision sensor using the visual data associated with each frame. Therefore, each camera pose is associated with a frame of sensor data. As typical in graph SLAM, poses correspond to graph vertices. Measurements of motion between poses are directly estimated using visual odometry. These motion constraints are represented in the graph as edges which describe the transformation and associated uncertainty as a Gaussian distribution.

While optimizing over a complete set of poses in the graph leads to a more fine-grained trajectory estimate, this also significantly increases computational cost. Often the trajectory is sparsified by selecting a subset of the camera poses over which to optimize. These poses and their associated image data are called keyframes. Many keyframe-based pose graph SLAM approaches represent the map in terms of the data associated with the keyframes and their associated optimized poses. Once the optimized trajectory has been solved for, the keyframe data is projected into the global coordinate system forming the map.

Therefore, this class of visual SLAM solutions often represent the map in terms of the available image data directly (direct methods) or in terms of extracted image features i.e. keypoints (feature-based methods). Often the map data corresponds to the data used for odometry estimation, thus methods employing direct odometry typically build dense point cloud maps, while methods employing feature-based (keypoint) odometry typically result in sparse maps of these 3D keypoints.

4.3 Dense Visual SLAM

Visual SLAM approaches whose map representations consist of collections of many unorganized primitives, such as 3D points, are referred to as *dense* approaches. Dense visual SLAM methods often employ direct methods to estimate camera motion i.e. they use color and depth/point cloud information directly, without extracting image features. This subclass of visual SLAM methods has recently grown in popularity, and have produced solutions with marked improvements in camera tracking and map reconstruction accuracy.

4.3.1 Related Work

Much recent work in real-time visual SLAM involve direct methods where input data is used directly with minimal processing and maps are dense. One of the first of such methods, KinectFusion, proposed by Newcombe et al. [26], maintains a map in the form of a dense global implict surface model using a truncated signed distance function (TSDF), and uses a point-to-plane iterative closest point (ICP) [27] minimization between incoming depth data and this TSDF surface model to perform accurate camera tracking. Like many methods that employ dense maps, this method struggles when the map becomes large due to increased memory requirements.

Kintinuous, proposed by Whelan et al. [28], improves upon KinectFusion by allowing the mapping volume of the TSDF to move along with the camera. This drastically reduces memory requirements and permits mapping of much larger environments when compared to KinectFusion. In addition, this approach includes methods for loop closure detection through bag-of-words place recognition and improved visual odometry.

DVO SLAM, proposed by Kerl et al. [17, 18], benefits from low drift odometry via minimization of both geometric and photometric reprojection error between RGBD image pairs. The odometry is made robust by the incorporation of motion priors and the weighting of errors based on how well they fit a dynamically estimated t-distribution. This reduces the influence of residuals which have large photometric or large geometric error, as they are likely to be outliers. The map in this approach is modeled by a pose graph, where every vertex is a camera pose with associated keyframe (intensity + depth), and edges represent the relative transformation between keyframe poses. New keyframes are selected based on an entropy metric, and a similar metric is used to identify loop closures. The map in this case consists of the points associated with each keyframe/pose, which are projected into the global map based on the optimized camera trajectory.

While these approaches perform well for building accurate maps in near real-time, the resulting maps lack semantic description. Maps consisting purely of low-level primitives such as points or non-parametric surfaces do little to further understanding about what types of objects have been mapped or how an agent may interact with them. In practice, many of these low-level primitives are in fact measurements of the same surfaces and are therefore redundant if the nature of the underlying surfaces is known. In such a case where a higher level representation of these primitives is known, they may be replaced with this representation, avoiding computation associated with processing many low-level primitives.

SLAM++, proposed by Salas-Moreno et al. [29], is an attempt at real-time object-level SLAM. SLAM++ uses a database of object instances (defined offline) as potential landmarks to be detected from sensor depth data. This method uses a graph based approach to the SLAM problem, where graph vertices correspond to both camera and object poses and graph edges are relative transformations between these poses. Objects are detected and localized using a GPU accelerated version of the recognition method proposed by Drost et al. [30]. This method uses a Hough Transform like voting scheme operating in a parameter space. Votes are accumulated based on the correspondence of Point-Pair Features computed from the data and from potential object models. Detected objects with pose are then placed in the map as geometric landmarks for camera tracking. Sensed point clouds are aligned to the expected views of map object models via a point-to-plane ICP algorithm [27]. Predicted views of map objects aid in the recognition task by "masking off" areas in the data that they explain, reducing the search area for new objects.

However, this approach to geometric object detection requires object surface models to be a priori known in great detail, and has no flexibility in terms of object parameters. Fixed databases such as this suffer from a lack of generality, i.e., one must observe "the chair" from the database rather than "a chair." Authors have proposed introducing constraints from more general geometric surfaces often found in the scene, such as planar surfaces.

Salas-Moreno et al. also proposed a real-time dense planar SLAM approach [31] that leverages the predominantly planar structure of indoor scenes to efficiently map large regions of the scene that satisfy planar constraints. The map in this approach contains "surfels", small entities describing locally planar regions without connectivity. Throughout the estimation procedure surfels considered planar are incrementally labeled as belonging to one of a set of detected planes. This results in a set of bounded planar regions that explain large parts of the scene. Tracking is accomplished via frame-model ICP alignment.

Yang et al. proposed a monocular SLAM algorithm that leverages geometric cues from planar regions detected by a convolutional neural network and subsequent polyline fitting to the region boundary. Yang et al. extend a popular dense monocular SLAM algorithm, LSD SLAM, proposed by Engel et al. [32, 33] by incorporating detected planar regions as landmarks and using them to improve depth estimation, leading to improvements in both camera tracking and dense reconstruction accuracy. The estimated planar regions also provide invaluable semantic information about the scene.

4.3.2 DVO SLAM

We have selected DVO SLAM, proposed by Kerl et al., as a dense visual SLAM platform for extension. This section discusses the operation of the DVO SLAM algorithm in depth, and provides specific insight for portions of the algorithm that will be modified and extended in the next chapter.

4.3.2.1 Overview

This system uses the direct odometry approach similar to the one discussed in chapter 3, which minimizes both intensity and depth reprojection error between RGBD image pairs, resulting in estimates of camera movement and associated uncertainty. The SLAM optimization is implemented using a keyframe-based pose graph framework where graph vertices are camera poses, and where a subset of these camera poses contain associated RGBD images, called keyframes.

Keyframes form the root of spatially local maps whereby the tracking front-end estimates the transformation between consecutive RGBD frames as well as to the keyframe. This imposes odometry constraints on the camera trajectory that limits drift. When odometry estimation between the current frame and the keyframe de-

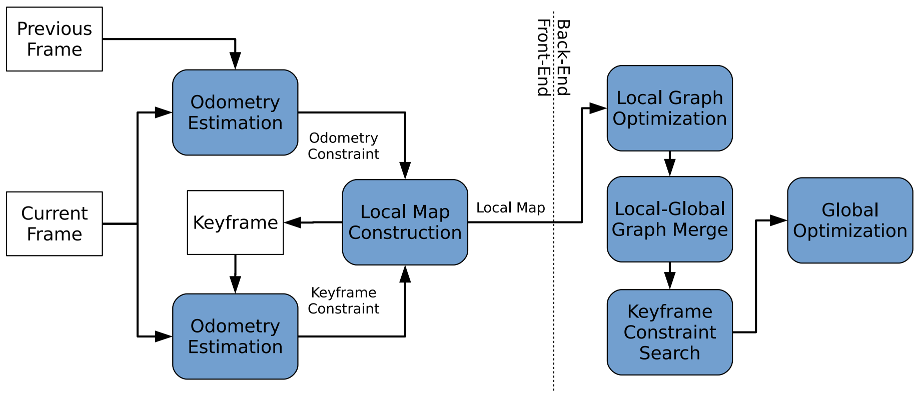


Figure 4.1: A high-level overview of the DVO SLAM algorithm showing RGBD image data in white and processes in blue. Note the separation of responsibilites between the front-end and the back-end.

grades, a new keyframe is laid off and tracking continues based off of the new keyframe. Constraints between nearby keyframes in the global graph are also considered to allow for loop closures and to improve the trajectory estimate.

The map in this approach is simply the data associated with each keyframe. When the optimal trajectory has been solved for, the data associated with each keyframe/-pose is then projected into the global coordinate system, forming a dense colored point cloud.

4.3.2.2 Front-end

The DVO SLAM front-end operates in a self-contained local mapping structure rooted at a keyframe. The local graph structure is incrementally constructed over time as the camera moves and new RGBD image frames are processed. When a new RGBD image frame is obtained, odometry with associated uncertainty is estimated between the previously processed frame as well as the keyframe, corresponding to odometry edge constraints in the graph structure as in figure 4.2. As discussed in sections 4.1.1 and 4.2, these edges encode the relative transformation as a Gaussian distribution.

When the odometry estimation between the current frame and the keyframe degrades, the current local map is packaged up and pushed to a queue to be processed

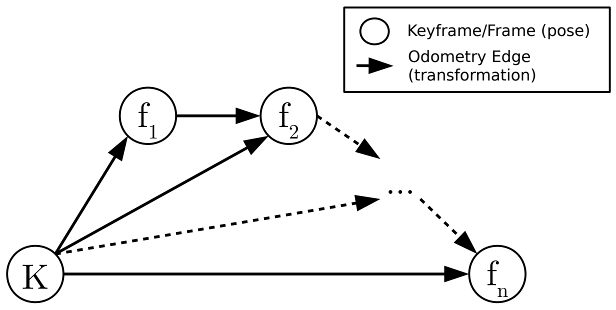


Figure 4.2: DVO SLAM local graph structure showing odometry constraints between consecutive frames/poses and the keyframe.

by the back-end. The current frame is promoted to a new keyframe which forms the root of a new local map, tracking then continues in this manner. Kerl et al. in [17] discuss various metrics used for determining when to select a new keyframe, including a maximum translational and rotational distance from the current keyframe, and a novel entropy ratio metric. The front-end does not directly interact with the global graph structure on the back-end, instead the queue of local maps, filled by the front-end, is processed by the back-end as quickly as possible.

4.3.2.3 Back-end

The DVO SLAM back-end operates in a separate thread than the front-end. The back-end manages the global graph, processes local maps from the front-end, and handles optimization over the keyframe graph. Incoming local maps are popped from the stack, and a local optimization is performed over all vertices in the local graph. The local graph is then merged into the global graph structure, where it “hooks up” to the most recent odometry vertex in the global graph as show in figure 4.3.

At this point a keyframe constraint search begins, as shown in figure 4.4. A radius search rooted at the newly inserted keyframe identifies nearby candidate keyframes which may provide additional odometric constraints. Odometry estimation is then performed between the root keyframe and the candidate keyframes, and valid constraints are added to the global graph. In this way, DVO SLAM can detect so-called loop closures, whereby the SLAM system detects that the agent has returned to a

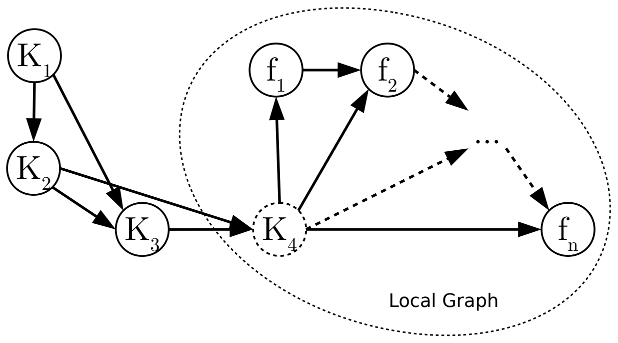


Figure 4.3: DVO SLAM graph structure showing the local graph in relation to the global keyframe graph.

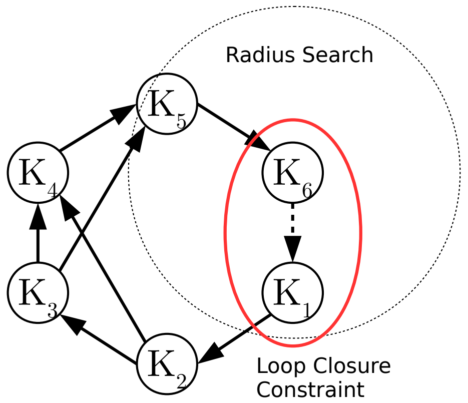


Figure 4.4: DVO SLAM identifies candidate keyframes for loop closure constraints. When a new keyframe is added to the global graph, a radius identifies existing nearby keyframes as candidates to build constraints from. Dense odometry estimation is then performed between the keyframe and candidate keyframes.

previously visited area.

The back-end then performs a global optimization over the keyframe subgraph, consisting of the keyframe vertices and their associated edges, and excluding non-keyframe vertices/edges. DVO SLAM employs Powell's dogleg method [34, 35] to solve the global optimization. After optimization, the optimized set of keyframes provides the most likely camera trajectory given all constraints. Figure 4.5 shows a visualization of the DVO SLAM algorithm.

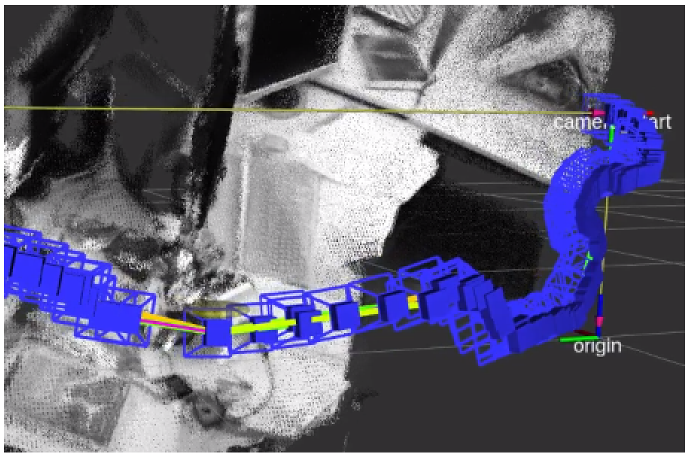


Figure 4.5: Visualization of the camera trajectory and the dense point cloud map of DVO SLAM.

CHAPTER 5: IMPROVING DENSE SLAM USING DETECTED GEOMETRIES

While dense mapping approaches perform well in terms of camera tracking and reconstruction accuracy, leveraging direct tracking approaches and incorporating sparse yet semantic geometric map elements could provide increased accuracy and superior scene understanding while reducing overall map complexity. Incorporating detected geometric primitives provides additional constraints for camera trajectory estimation, leading to increased tracking accuracy and subsequently decreased dense reconstruction error. In addition, including geometric primitives in the map provides invaluable semantic information about the scene- otherwise unavailable from a purely dense map.

This chapter begins by introducing the modifications needed to extend the DVO SLAM algorithm to incorporate landmark constraints in the optimization. It later describes the methods used to detect the aforementioned geometries, and the details relating to their inclusion in the SLAM implementation.

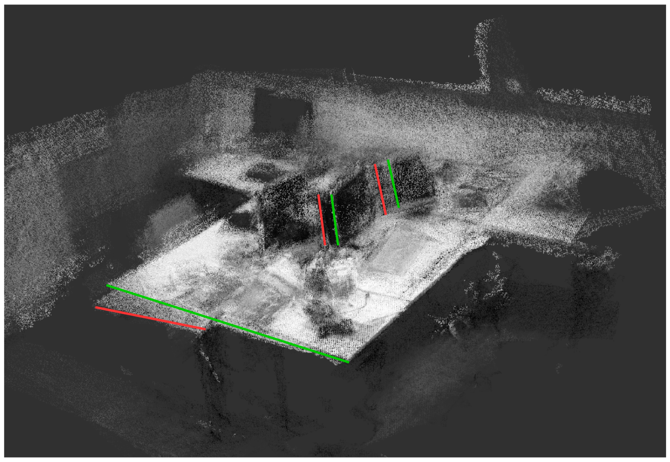


Figure 5.1: Reconstructed point cloud of the fr1/desk dataset using DVO SLAM with default settings [17, 36]. Some errors in the reconstruction are highlighted in red with the green lines showing the correctly reconstructed surface. We seek to reduce this mapping error and enhance map semantics by considering detected object poses in the SLAM optimization.

5.1 Extending Dense SLAM Using Landmarks

While the DVO SLAM algorithm leverages keyframe odometry constraints to limit drift and detect loop closures, it does not explicitly include what most robotics researchers would consider landmark constraints in the SLAM estimation. Inclusion of the pose estimates of detected objects into the optimization allows for additional constraints that serve to increase the accuracy of the trajectory estimate, increase loop closure detection range, and reduce dense reconstruction error. In addition, we benefit from the semantic information associated with known objects in the map. We extend DVO SLAM to include multiple types of geometries as landmarks in the SLAM optimization. For a more detailed description of the operation of the unmodified DVO SLAM algorithm, see section 4.3.2.

5.1.1 Front-end Modification

Additional functionality must be added to the DVO SLAM front-end in order to handle incoming landmark measurements and include these landmarks in the local map structure. In addition, correspondence must be determined between incoming landmark measurements and those already in the local map.

After odometry is estimated for each frame of incoming RGBD data, a detector for the landmark of interest operates on the frame, and extracts measurements of the landmark in the coordinate frame of the camera. It must then be determined whether or not this particular landmark has been viewed previously, a task known as correspondence estimation. If a correspondence can be established with an existing landmark in the local map, the observation edge is associated with the corresponding landmark, otherwise we consider the observation to be a measurement of a never before seen landmark.

The local graph structure will now contain a landmark vertex and an observation edge connecting this vertex and the frame/camera pose which made the observation

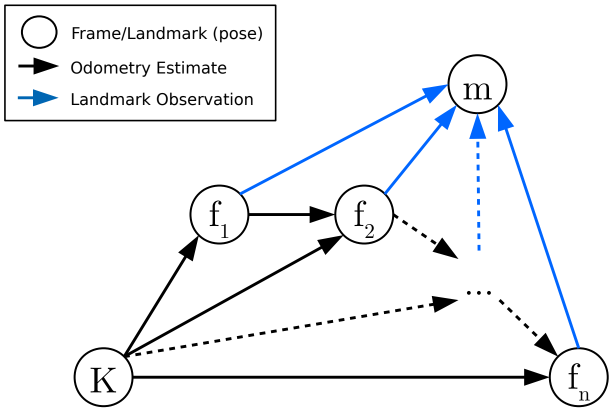


Figure 5.2: DVO SLAM local graph structure showing odometry constraints between frames/poses as well as landmark observations of a map feature labeled m.

(see figure 5.2). Landmark observations form constraints between poses that mutually observe a given landmark, which force the relative camera poses into agreement over the landmark pose. This process continues with each incoming frame, and over time the local graph is constructed consisting of both odometry and landmark constraints.

5.1.2 Back-end Modification

As discussed in section 4.3.2.3, the DVO SLAM back-end processes the queue of local maps created by the front-end. The back-end must merge the incoming local graph structure into the global graph. A few modifications must be made in order to handle the addition of landmark vertices and their associated graph edges.

The g^2o library which handles the graph structures requires that graph elements have unique integer identifiers. To this end, a manager for graph element identifiers was implemented. The user specifies specific labels for graph element types and the manager is used to generate and modify graph element identifiers which streamlines the process and allows for new graph elements to be more easily used in the algorithm.

When the back-end thread begins the process of merging a local map into the global graph structure, a correspondence check must occur to see if local measurements should be associated with an existing global landmark. If the correspondence criteria is met, then a merge procedure shown in figure 5.3 maps the landmark observations

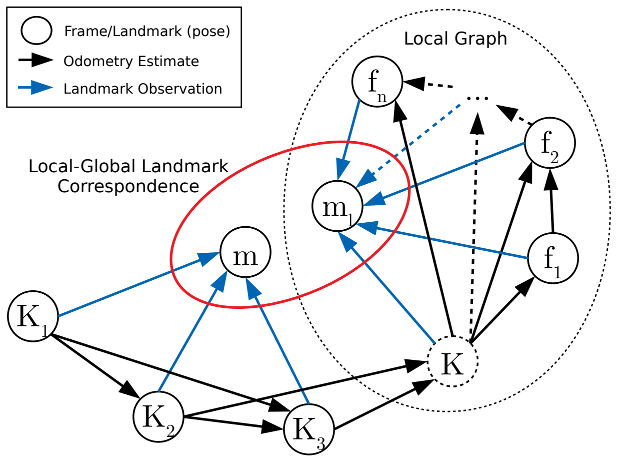


Figure 5.3: The local-to-global graph merge procedure for landmark vertices. A correspondence has been established between the local landmark vertex m_l, and the global vertex m. The merge procedure maps the observations associated with m_l onto m, and m_l is subsequently removed from the graph.

(edges) from the local landmark vertex to the corresponding global landmark vertex. This process allows for landmark based loop closures to be detected even when there are *no nearby keyframes to form odometric loop closure constraints.*

The global optimization occurs just as before, but also considers the landmark observation edges connected to keyframes. If a landmark has been observed from multiple keyframe camera poses, landmark constraints connect the poses that mutually detected the landmark. During the optimization, the landmark vertices are marginalized out, and these observation constraints are mapped to constraints between the poses that observed the landmark. These constraints adjust the camera trajectory to ensure that the camera poses that observed landmarks better agree on the landmark position.

The information matrices of the observations control the relative influence of the landmark constraints over the existing camera odometry constraints in the overall optimization. Large values for an observation's information correspond to high certainty in the measurement value, and therefore a large weight in the overall optimization. Conversely, small values for an observation's information correspond to low certainty, and the measurement will have little influence over the optimized trajectory. In our implementation, we set this information matrix to a constant value, however, ideally the value should be consistent with the certainty associated with the detection.

5.2 Planar Surface Estimation

One particular geometric constraint of interest is planar surfaces. Planar surfaces are ubiquitous in indoor scenes and therefore serve to explain large quantities of dense data. Extracted planar models can also serve as valuable geometric primitives to be used to detect more complex objects and surfaces. Individual infinite plane measurements partially constrain the relative pose estimate of the observer; all motions save for rotation and translation in the plane can be constrained by a plane measurement.

5.2.1 RANSAC Plane Fitting

The Random Sample Consensus (RANSAC) method discussed in section 2.6 can be used to fit a planar model to a set of 3D points that include outliers. In order to extract planar geometries, we apply RANSAC plane fitting to the reconstructed 3D points of incoming depth frames produced by the RGBD camera. We then leverage these planar surface measurements as landmarks to improve the performance of dense SLAM.

Large scale planar surfaces can be estimated for each frame of depth data using RANSAC. A RANSAC based planar model segmentation can be applied to the 3D point cloud associated with the depth image frame. The RANSAC plane segmentation proceeds as follows:

1. The RANSAC algorithm is used to fit a planar model to the point data
2. If the proportion of inlers to the original number of points is below a threshold value, stop
3. The inliers of the fitted plane are removed from the point data

4. While the number of points exceeds some percentage of the original: Go to step 1

Due to the high probability of randomly selecting points belonging to large scale planar surfaces (as they make up a large percentage of the total points), and the RANSAC method's inherent robustness towards outliers, this simple approach is capable of extracting large planar surfaces such as walls and floors from the scene.

5.2.2 Fast Least Squares Plane Fitting using Range Images

The least squares plane fitting formulations discussed in section 2.8 can of course be used for fitting. However, we can exploit the structure of the depth image and the known calibration parameters in order to perform the fitting task more efficiently. The following formulations are discussed in [37].

We start by investigating the reconstruction equations described in section 2.6and grouping the parameters into two sets: (1) the RGBD camera calibration parameters $\{f_x, f_y, c_x c_y, \delta_x, \delta_y\}$ and (2) the depth measurement Z. Fig. 5.4 shows the geometry of the RGBD depth measurement using these parameters. Here, we show the angle, θ_x, that denotes the angle between the optical axis and line passing through the RGBD image pixel (x, y) when viewed from the top-down. A similar angle, θ_y, is obtained using the same geometric setup from a side-view. Hence, we make the substitutions shown in equations (5.1) for the terms of the reconstruction equations resulting in the new reconstruction equations (5.2):

$$\begin{aligned} \tan\theta_x &= (x + \delta_x - c_x)/f_x \\ \tan\theta_y &= (y + \delta_y - c_y)/f_y \\ Z &= Z \end{aligned} \tag{5.1}$$

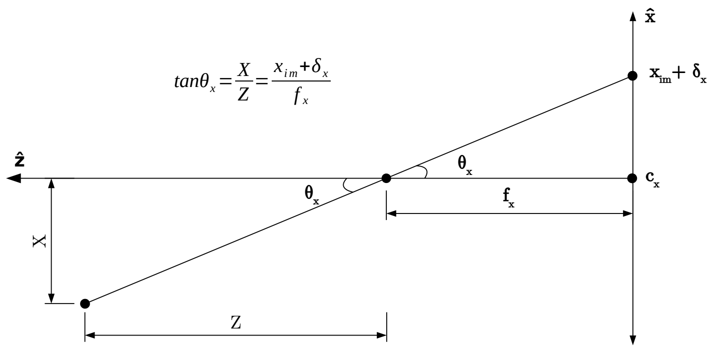


Figure 5.4: Note the relationship formed by the tangent of the angle θ_x. A similar relationship exists in the y direction.

$$\begin{aligned} X &= Z\tan\theta_x \\ Y &= Z\tan\theta_y \\ Z &= Z \end{aligned} \tag{5.2}$$

Back-substitution into the plane equation gives $aZ\tan\theta_x + bZ\tan\theta_y + cZ + d = 0$. We now multiply this equation by $1/Z$ to arrive at the equation (5.3):

$$a\tan\theta_x + b\tan\theta_y + c + \frac{d}{Z} = 0 \tag{5.3}$$

Notice the coefficients a, b, c are functions of *only the camera calibration parameters* which determine, for each image pixel, the values of $\tan\theta_x$ and $\tan\theta_y$. The coefficient d is *only* a linear function of the measured depth (inverse depth).

5.2.2.1 Implicit Plane Fitting to RGBD Range Data

For implicit plane fitting using the re-organized equation (5.3), the new monomial vector becomes $\mathbf{M}_i = [\ \tan\theta_{x_i} \quad \tan\theta_{y_i} \quad 1 \quad \frac{1}{Z_i}\ ]$ and the scatter matrix has elements:

$$\mathbf{M}^t\mathbf{M} = \sum_{i=1}^{N} \begin{bmatrix} {\tan\theta_{x_i}}^2 & & & \\ \tan\theta_{x_i}\tan\theta_{y_i} & \tan\theta_{y_i}^2 & & \\ \tan\theta_{x_i} & \tan\theta_{y_i} & 1 & \\ \frac{\tan\theta_{x_i}}{Z_i} & \frac{\tan\theta_{y_i}}{Z_i} & \frac{1}{Z_i} & \frac{1}{Z_i^2} \end{bmatrix} \tag{5.4}$$

where the symmetric elements of the upper-triangular matrix have been omitted to preserve space. It is important to note that only 4 elements of this matrix, $[\ \frac{\tan\theta_{x_i}}{Z_i}\ \ \frac{\tan\theta_{y_i}}{Z_i}\ \ \frac{1}{Z_i}\ \ \frac{1}{Z_i^2}\ ]$, depend on the measured depth data and, as such, this matrix requires less (~50% less) operations to compute. The upper left 3x3 matrix of $\mathbf{M}^t\mathbf{M}$ *does not depend upon the measured sensor data.* As such, once the calibration parameters of the camera are known, i.e., (f_x, f_y), (c_x, c_x), (δ_x, δ_y), many of the elements of $\mathbf{M}^t\mathbf{M}$ are determined and *may be pre-computed before any data is measured* with the exception of the 4 elements in the bottom row.

5.2.2.2 Explicit Plane Fitting to RGBD Range Data

For explicit plane fitting using the re-organized equation (5.3), the revised error function is shown below:

$$\epsilon(a,b,c,d) = \min_{a,b,c} \sum_{i=1}^{N} \left(a\tan\theta_{x_i} + b\tan\theta_{y_i} + c - \frac{1}{Z_1} \right)^2 \tag{5.5}$$

Let $\alpha = [\ a \quad b \quad c\ ]^t$ denote the vector of explicit plane coefficients, $\mathbf{b} = 1/[\ Z_0 \quad Z_1 \quad \ldots \quad Z_N]^t$ denote the target depth values and $\mathbf{M}$ denote the matrix of planar monomials formed from the 3D (X, Y, Z) surface data having i^{th} row $\mathbf{M}_i = [\ \tan\theta_{x_i} \quad \tan\theta_{y_i} \quad 1\ ]$.

Then equation (5.6) shows the scatter matrix, $\mathbf{M}^t\mathbf{M}$, needed to estimated the explicit plane coefficients where the symmetric elements of the upper-triangular matrix have been omitted to preserve space.

$$\mathbf{M}^t\mathbf{M} = \sum_{i=1}^{N} \begin{bmatrix} \tan{\theta_{x_i}}^2 & & \\ \tan\theta_{x_i}\tan\theta_{y_i} & \tan\theta_{y_i}^2 & \\ \tan\theta_{x_i} & \tan\theta_{y_i} & 1 \end{bmatrix} \tag{5.6}$$

It is important to note that *none* of the elements of the scatter matrix depend on the measured depth data and, as such, this matrix requires a *constant* number of operations to compute for each measured image, i.e., *it can be pre-computed given the* *camera parameters.* Hence, explicit plane fitting in using this formulation requires only computation of the vector $\mathbf{b} = 1/[\ Z_0 \quad Z_1 \quad \ldots \quad Z_N]$ for each range image and the best-fit plane is given by a single matrix multiply: $\widehat{\alpha} = \left(\mathbf{M}^t\mathbf{M}\right)^{-1}\mathbf{M}^t\mathbf{b}$. Where the value of $\mathbf{M}^t\mathbf{b}$ is given below:

$$\mathbf{M}^t\mathbf{b} = \sum_{i=1}^{N} \frac{1}{Z_i} \begin{bmatrix} \tan\theta_{x_i} \\ \tan\theta_{y_i} \\ 1 \end{bmatrix}$$

5.2.2.3 Computational Savings via Integral Images

Integral images, first introduced in [38] and popularized in the vision community by the work [39], computes the cumulative distribution of values within an image. This technique is often applied for the purpose of efficiently computing sums over arbitrary rectangular regions of an image. This is often the case for normal estimation in range images. Here, integral images are computed for the (X, Y, Z) values or perhaps for all values of the scatter matrix of equation (2.52). Computation of an integral image for a scalar value, e.g., intensity, requires 4 operations per pixel, i.e., for an image of N pixels it has computational cost of $t = 4N$ operations. More generally, computation of an integral image for an arbitrary function of the image pixel data $f(I(x, y))$ having computational cost C has computational cost to $t = N(C + 4)$.

5.2.2.4 Application

These fitting formulations allow fast least squares surface fitting to rectangular regions of depth images. We can exploit these techniques to quickly detect planar surfaces and use them as landmarks to improve dense SLAM.

5.2.3 Plane Landmark Correspondence Implementation

To include planes in the SLAM implementation correspondence must be able to be established between incoming plane measurements and previously identified plane landmarks. Two metrics were considered in order to evaluate the closeness of two infinite planes. The first computes the closest 3D point to the origin that lies on the plane. This point is unique for a plane up to the sign of the normal. In our implementation, all planar surface fits result in constrained normals which face the image plane. In this case the parameterization is unique. Assuming the plane equation is in Hessian normal form with normal $\hat{n}$ and origin distance d, this point can be computed using equation (5.7).

$$p_{plane} = -d \cdot \hat{n} \tag{5.7}$$

For two planes the Euclidean distance between these computed points is taken as the plane distance metric. In this case we apply directly the radius search technique used for 3D point landmarks in order identify correspondences between planes.

The second metric we call a plane "dissimilarity" metric between two planes α_i and α_j. It involves the plane normals, $\hat{n}_i$ and $\hat{n}_j$, the signed origin distance values, d_1 and d_2, and a weighting parameter, λ, and is given in equation (5.8).

$$\epsilon(\alpha_i, \alpha_j) = 1 - (\hat{n}_i \cdot \hat{n}_j) + \lambda \left| d_i - d_j \right| \tag{5.8}$$

This metric allows for more control because of the weighting parameter λ which weights the relative influence of the error in the normals and the error in the d values. However, it is not immediately clear what value λ should take on, and if this value should change depending on the magnitude of the d values.

At a global graph level during a local map merge, we determine plane correspondence using the same distance metric and radius search procedure used for local correspondence to search for existing nearby global plane landmarks to which the local plane may correspond.

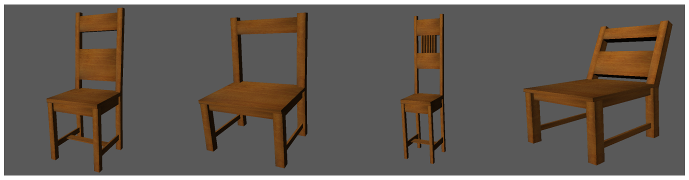


Figure 5.5: Instances of a chair generated from a chair shape grammar. All are valid instances of the grammar with different object parameters.

5.3 Shape Grammars for Object Recognition

Shape grammars seek to embed an object semantic as a model consisting of a specific collection of primitive shapes having a highly constrained relative geometry. An object's grammar is specified using a shape program- written using a custom shape programming language referred to as PSML (Procedural Shape Modeling Language). The goal of PSML is to provide a formal language where humans can concisely specify a 3D object shape semantic. This formalization encodes the object semantic unambiguously as a transformation from natural language to 3D shape. For example, a PSML shape program for the *chair* semantic represents instances of *chair* as a arrangement of fundamental chair components, e.g., *seat, backrest, legs,* each of which is constrained in their absolute and relative sizes, numbers, and relative positions (see figure 5.5). By varying the parameters of the shape, a so called shape space is formed consisting of possible shape instances that satisfy the grammar.

This section demonstrates a prototypical system that solves the inverse problem, i.e., transforming 3D shapes into natural language. This is accomplished by associating sensor measurement patterns to shape primitives and specific arrangements of shape primitives to PSML programs from which the shape arrangements could originate. Vocabularies can be constructed as collections of PSML shape programs that can have hierarchical relationships, e.g., chair $\rightarrow$legs, seat, backrest.

While existing approaches using grammatical/procedural models and L-systems are a source of inspiration [40, 41, 42], the proposed approach represents a significant departure from these models. The approach promises to facilitate creation of detection systems that extract generic hierarchies of semantic structures from 3D data. Prior work has focused on highly-specific problems including the spatial organization of generic block-like objects [43, 44], building facades [45, 46, 47], architectural drawings [48, 49], and buildings [50, 51, 52, 53] from imagery. We seek to leverage this recognition framework to recognize object geometries for the purpose of including them as landmarks in the SLAM estimation process.

5.3.1 Framework

Construction of an inverse system (transforming 3D shapes into natural language) requires PSML shape programs and specification of the shape sensors. Using PSML and sensor models, instances of measured objects can be virtually simulated to create ground truth measurement data for different object instances. This data can be used as input to supervised learning systems to generate pixel-level, part-level, and object-level classifiers to detect the whole shape in terms of its pixels and sub-components. For this work, we use an RGBD camera to sense spherical "ball" objects in conventional color camera images and depth images. The overall problem specification and estimation framework is outlined in the following five steps:

1. Write a PSML grammar to describe the object(s) to be recognized,
2. Specify the sensing modalities that will measure the object(s),
3. Construct a recognizer for each sensing modality,
4. Apply the designed recognizer to detect and describe instances of the modeled objects within the sensor data,
5. Use recognition results to solve application domain problems.

figure 5.6 describes an overview of this framework and its application towards the detection of colored spherical objects for a SLAM front-end. This thesis implements this framework using a PSML program to describe a "ball" semantic (see figure 5.7) as

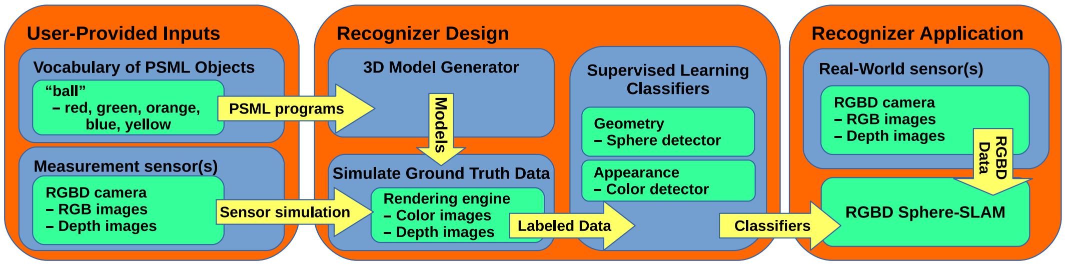


Figure 5.6: Overview of the framework used to semantically model and detect object instances. Detected object instances can be leveraged for a variety of purposes including SLAM.

```
package primitives;

public class Sphere extends ShapeGrammar {
    public Sphere() {}
    public Sphere(final Shape myShape) {}
    public Shape main(final Shape myShape) {
        rules {
            Axiom::I("sphere", new double[]{0.5}){genSphere};
            genSphere::appearance("diffuse",  new double[]{0.692, 0.269, 0.272})
                    appearance("shininess", 0)
                    appearance("specular",  new double[]{0.15, 0.15, 0.15})
                    appearance("transparency", 0.0) {j3d.terminal};
        }
        return myShape;
    }

    public static void main(String[] args) {
        Sphere s = new Sphere(); // invoke the Java constructor
        Shape sv = s.main(new Shape("root")); // invoke the Java -> PSML initial symbol
        s.setShape(sv); // set the "Sphere" object shape to that returned by the shape grammar
        s.showShapes("sphere"); // visualize the constructed object
    }
}
```

Figure 5.7: A PSML shape program describing the geometry and appearance of a colored ball instance.

a sphere geometry having five different colors that will be sensed with an RGBD sensor. Color and depth recognition models are applied to detect and classify instances of the "ball" semantic. Detection results are used as 3D point object landmarks in a dense SLAM system as geometric constraints to improve map accuracy.

5.3.2 Recognition of the Colored Ball Semantic

PSML shape programs allow users to formally define a relationship between semantic words and the 3D objects that qualify as valid instances for these words. In this thesis, we use a trivial PSML program to describe a "ball" semantic and use it to generate 3D models for colored spherical shapes we expect to observe in the scene. The PSML "ball" semantic models real-world colored 3D spherical markers of inter-

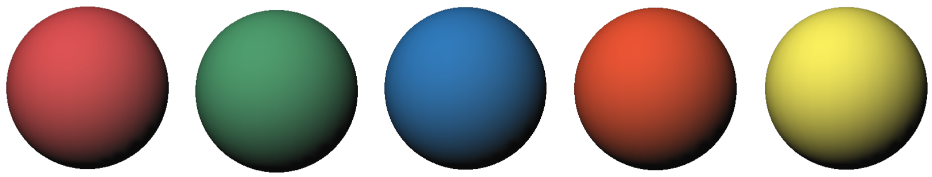


Figure 5.8: 3D “ball” instances generated from the sphere shape program of figure 5.7.

est, which will be used as landmarks in a 3D RGBD-SLAM algorithm. Specifically, we seek to recognize foam balls having a radius of approximately 3 cm, a reflectance well approximated by a diffuse or “Lambertian” surface and five distinct appearances referred to by their apparent colors: {“red”, “green”, “blue”, “orange”, “yellow”}. figure 5.7 shows the PSML “ball” program used to model the geometry and appearance of the target foam balls as instances of colored spheres. There are both geometric and appearance properties for the “ball” program. The geometric properties are the sphere position and radius. The appearance properties are the sphere reflectance properties and the five distinct surface colors.

Current software supports sensing via conventional color/grayscale camera imagery and geometric imagery, e.g., LIDAR, stereo 3D reconstruction and RGBD sensing. Here, we can create instances of each 3D object and simulate sensed RGB and geometric measurements by changing the virtual lighting conditions and the relative pose of generated instances of the virtual PSML objects and the sensor. This data then acts as ground truth labeled data that can be fed into supervised machine learning systems or generative models for the geometric shape and appearance. In this thesis we choose to implement a generative recognition approach for the PSML spheres and divide the recognition problem into two parts: (1) a maximum likelihood classification model to capture sphere appearance and (2) a 3D RANSAC spherical surface model fit to capture sphere geometry.

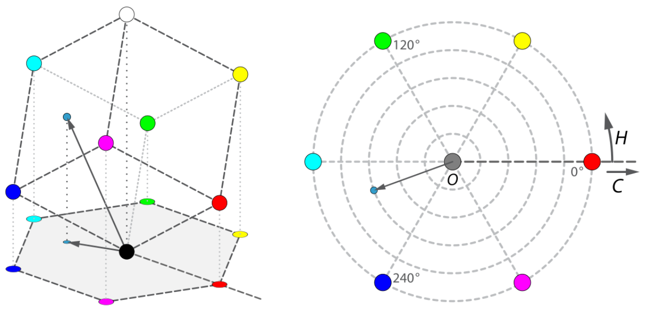


Figure 5.9: The hue-chroma plane formed by the projection of the RGB space onto the intensity axis. This polar space is used for pixel color classification. Images courtesy J. Rus, Creative Commons, Licensed under CC BY-SA 3.0, Modified.

5.3.2.1 Appearance-Based Detection

For this prototypical object, appearance models describe a set spherical 3D objects having diffuse surface albedos and distinct color appearances. To detect such objects efficiently in RGBD data, we first begin with detection of the distinctive color appearances of the object classes. Each pixel in the color image is classified as belonging to one of five color classes, or a background class.

This is accomplished quickly by parameterizing each color value as a 3-vector in the RGB colorspace, and projecting it onto the hue-chroma plane; the plane perpendicular to the grayscale or neutral axis. This gives rise to a 2D polar space as shown in figure 5.9, where angle represents the hue of the color, and radius the chroma of the color. The chroma is a measure of how colorful a color is relative to the brightness of a similarly illuminated white [54]. Here, colors whose chroma are below a threshold value are not considered further, and classified as background as all of our color classes have large chroma values.

Otherwise, a Bayesian classifier (see section 2.7 for detail) can be formulated to classify pixel values that exceed the minimum chroma threshold. Let θ denote the angle of the measured RGB pixel around the intensity axis, i.e., the RGB vector $[1, 1, 1]^t$. For each color class, we generate instances of the object illuminated from various sources and compute the value of θ for each object pixel within each image. This generates a collection of simulated color samples for each class. Let $\Theta_c = \{\theta_1, \theta_2, \ldots, \theta_N\}$ denote the set of measurements for the c^{th} class and θ_c the mean of this set. We take the scaled cosine of the angle between the candidate color vector and a given color class vector as a measure of the likelihood that the candidate color is observed given a color class. This results in a raised cosine distribution for each color class with each class having unique mean and equal variance. Let θ_q be the angle of the query color, q, in the hue-chroma plane. The likelihood that the measurement q, is observed given the color class c, is given by equation (5.9) for $-\pi \leq \theta_c - \theta_q \leq \pi$.

$$p(q|C = c) = \frac{1}{2\pi}(\cos(\theta_c - \theta_q) + 1) \tag{5.9}$$

Assuming all color classes are equally likely to be observed implies a uniform prior for each color class. The class label with maximum likelihood in this case is given by equation (5.10).

$$c^* = \operatorname*{argmax}_{c \in C} p(q|c) \tag{5.10}$$

Pixels belonging to each color class are then analyzed for connectivity using connected component labeling [55]. The bounding boxes of extracted regions as well as region area statistics are used to quickly determine whether the region resembles the circular appearance one would expect to observe if indeed a projection of a spherical object of interest was captured in the color image. Due to the circular nature of the spherical object's manifestation in the color image, we need only consider regions whose bounding boxes are sufficiently square and could be well inscribed by a circularly shaped region of pixels. The ratio of the bounding box's shortest side to the box's longest side provides a measure of the bounding box squareness and is used to refine candidate regions.

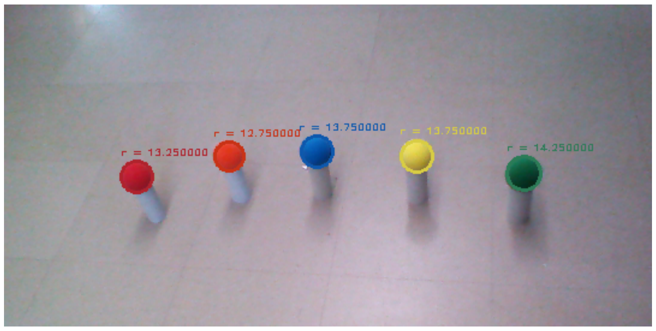


Figure 5.10: Output from the object appearance detector showing colored circle detections in the RGB image.

A candidate circle is then fit to the nearly square bounding box. The centroid position is simply taken as the midpoint of the bounding box, and the radius is computed by taking half of the value of the average side length. Two area ratios are then computed to ensure that the arrangement of the region pixels is well modeled by the candidate circle. We compute the ratio of the number of pixels that lie within the candidate circle to the expected area of the candidate circle. This ensures region pixels adequately fill the candidate circle. We also compute the ratio of the number of pixels that lie within the candidate circle to the total region area. This ensures that a large enough percentage of total region pixels lie within the candidate circle. At this point the candidate circle is considered a valid circle detection within the color image.

5.3.2.2 Geometry-Based Detection

The set of pixel measurements that generate valid circle detections within the color image provide corresponding samples in the depth image from which we can reconstruct 3D points. Using the reconstruction equations described in equation (2.6), measured 3D (X, Y, Z) positions of sensed surfaces can be directly computed from the intrinsic parameters of the camera and the measured depth image values.

These 3D surface locations should be explained well by a spherical model if indeed they are samples from a spherical surface. We therefore fit a spherical model to patches of 3D points presumed to be spherical in nature, and evaluate the model fit. If the data is well explained by the model instance, the estimated spherical model parameters should be reasonably accurate.

As discussed in section 2.6, a common method for model fitting in the presence of outliers is the Random Sample Consensus method, or RANSAC. This iterative method estimates the parameters of a mathematical model from a set of observations which are assumed to contain a large subset of observations known as inliers. We fit a spherical model to the 3D points using RANSAC to estimate the unknown sphere parameters: the center and radius. We use a simple ratio of inliers to the total number of observations to determine model confidence. Spherical models that exceed a threshold value are considered valid spherical detections and the estimated model parameters provide the sphere center and radius.

5.3.3 Sphere Landmark Correspondence Implementation

Measurements of sphere landmarks are taken as the center position of the detected sphere for the SLAM implementation, i.e. the sphere radius and color are discarded. To determine correspondence at the local map level, a 3D radius search similar to the one used for keyframes, is used to locate nearby candidate correspondences between the incoming point measurement all 3D points in the local map. The incoming point is transformed such that it exists in the coordinate frame of the local map and the radius search begins. The closest candidate point within the radius threshold is taken as the corresponding landmark if such a point exists. Otherwise we the incoming point measurement results in the inclusion of a new 3D point vertex in the local map.

At a global graph level during a local map merge, landmark correspondence is determined using the same radius search procedure used in the local map. Rooted at the incoming point estimate, we search for existing nearby global point estimates to which the incoming point may correspond. If the correspondence criteria is met, i.e.

there exists a global point estimate within the search radius, then the merge procedure associates observations of the local landmark with the corresponding global landmark.

CHAPTER 6: RESULTS

In this chapter we compare the performance of the dense SLAM algorithm and the modified dense SLAM algorithm that includes geometric landmark constraints.

6.1 Dense SLAM with 3D Point Landmarks

A simple scene including various color classes of spherical markers was used to evaluate the result of the inclusion of landmark constraints into the DVO SLAM algorithm. The spherical landmarks themselves were modeled using PSML which applies the Phong shading model [56] to rendered object instances. To provide appearance information for each color class, spheres were modeled using a diffuse reflectance value taken as the mean color value of the sphere as it appeared in a color image under fluorescent lighting as a starting point. The rendered model was then compared visually to the actual sphere marker, and both the diffuse and specular reflectance values adjusted until the rendered image agreed with the physical sphere appearance. The detected center coordinates of the colored spherical objects are included in the SLAM algorithm as 3D point landmark constraints.

An image sequence consisting of 500 RGBD images taken over ~17 seconds was produced as the camera was moved counterclockwise about the marker arrangement. This image sequence was then processed by the DVO SLAM algorithm with default settings and our own modified version that incorporated 3D point landmark constraints. As shown in Figure 6.1, the inclusion of additional landmark constraints improves mapping accuracy. In the vicinity of the spherical landmarks, mapping error is reduced significantly. For each spherical landmark present in the reconstructed point clouds, the maximum spread of the associated points was measured and compared

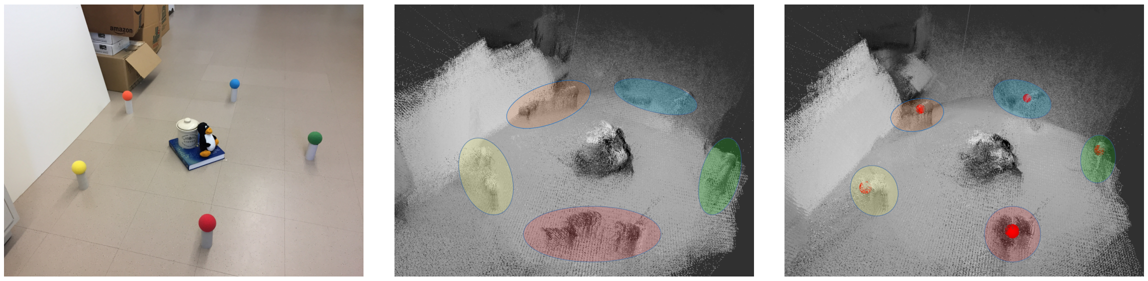


Figure 6.1: A color image of the scene (a), resulting point cloud using DVO SLAM without landmark constraints (b), and resulting point cloud with landmark constraints included (c). The reconstructed scene is viewed from the same camera viewpoint in both images. The red spheres in (c) are the optimized estimates of the detected spherical landmarks. Note the increased mapping accuracy especially in the colored regions that surround the spherical landmarks when landmark constraints are added to the graph optimization.

between SLAM implementations. Values in Table 6.1 demonstrate significant reduction in reconstruction error in regions surrounding the spherical landmarks. Other large scale planar surfaces in the scene such as the white poster board and cardboard box also see increased mapping accuracy with the inclusion of the spherical landmark constraints.

Table 6.1: Approximate maximum spread of reconstructed landmark points.

Landmark Color	DVO SLAM	DVO SLAM with Landmark Constraints	Point Spread Reduction	Connectivity (Keyframe Edges)
Red	0.352 m	0.159 m	54.8%	198
Green	0.154 m	0.079 m	48.7%	5
Blue	0.400 m	0.243 m	39.3%	122
Orange	0.351 m	0.183 m	47.8%	30
Yellow	0.370 m	0.152 m	58.9%	58

6.2 Dense SLAM Using Multiple Geometries

Another simple scene including spherical markers and large scale planar surfaces was used to evaluate the result of the addition of both point and plane landmark constraints into the DVO SLAM algorithm. Large scale planar surfaces were estimated for each frame of depth data using the approach discussed in section 5.2.1. This appraoch applies a RANSAC based planar model segmentation to the 3D point cloud associated with the depth image frame. This simple approach is capable of quickly extracting large planar surfaces such as walls and floors from the scene, which are leveraged as plane measurements for SLAM. The colored sphere detector described in section 5.3.2 was also used to provide 3D point measurements corresponding to the center positions of detected colored balls.

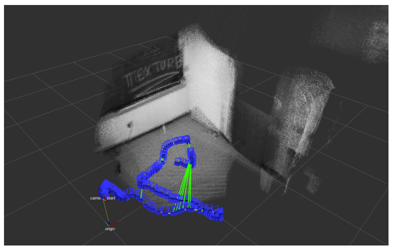
The scene features a corner of the UNCC Visionlab and includes colored sphere markers, a door, and a chalkboard with added texture. Figure 6.2 shows a few views of the reconstructed point cloud as well as the camera trajectory. The estimated poses of the object landmarks seem to correspond well with the dense reconstruction of the lab room corner, demonstrating that the overall SLAM estimation seems to be working as designed with multiple geometric landmarks.

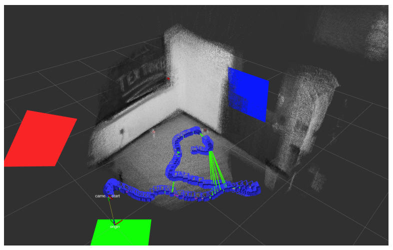
In addition, the accuracy of the dense reconstruction appears to have improved with the inclusion of these landmarks. Notice the reduction in reconstruction error of the wall near the door in subfigures (e) and (f). The chalkboard text reconstruction also seems to be improved, along with the points associated with the spherical markers. Subfigures (g) and (h) demonstrate a significant change in the overall geometric error of the room corner when the landmark constants are included in the optimization. While still not perfect, the planarity of the walls seems to have improved significantly due to the planar surface constraints.

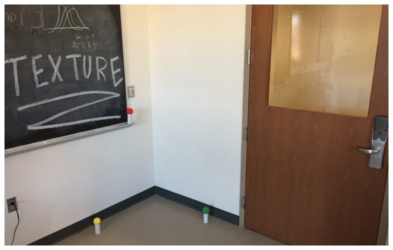
6.3 Fast Planar Surface Fitting Benchmark

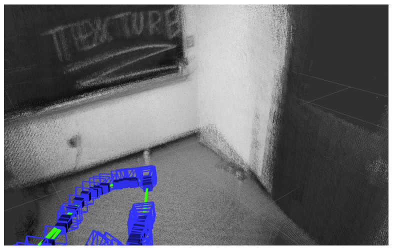
The explicit least squares planar surface fitting technique described in section 5.2.2 was implemented in C++ and compared against two plane fitting techniques available in the popular OpenCV library. The implementation was compared against the OpenCV implementations of the Fast Approximate Least Squares (FALS) and Spher-

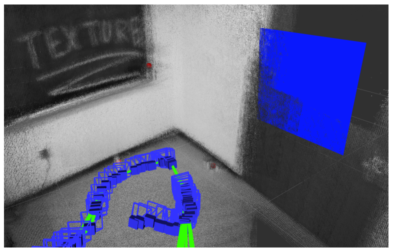
(a)

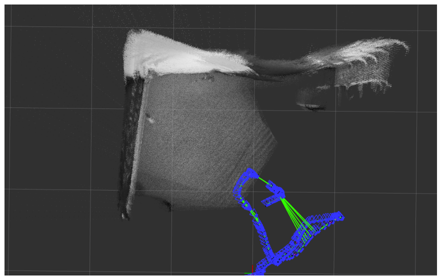

(b)


(c)


(d)


(e)


(f)


(g)

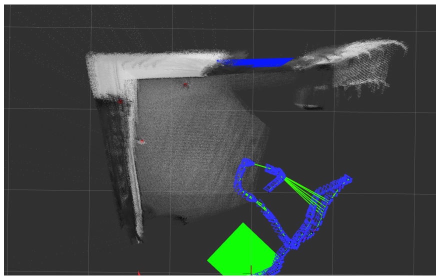

(h)

Figure 6.2: Demonstration of DVO SLAM with incorporated 3D point and planar surface constraints. Figures (a) and (d) show color images of the scene, figures (b), (e), and (g) show the reconstruction using unmodified DVO SLAM. Figures (c), (f), and (h) show the reconstruction with landmark constraints included. The square markers show sections of the estimated infinite plane landmarks while the red colored sphere markers show the estimated positions of detected colored balls. Notice the reduction in reconstruction error when landmark constraints are included in the optimization.

Table 6.2: Frame-rate comparison between the proposed explicit plane fitting method and two methods available in the OpenCV library.

Method	Frame-rate (fps)
FALS	7
SRI	1.3
Proposed	20

ical Range Image (SRI) methods described in [57]. The benchmark was performed on using an Intel i5-6200U @ 2.3 GHz , on 640x480 depth images provided by the Orbbec Astra RGBD sensor. As shown in table 6.2, the proposed fitting approach results in a significant frame-rate improvement compared to these methods.

CHAPTER 7: CONCLUSION

This thesis has explored the extension of a state of the art dense RGBD SLAM implementation to include detected objects and surfaces as geometric elements in the estimated global map. The efficacy of employing these elements as landmarks to aid in localization and mapping has been demonstrated, showing improvements in dense reconstruction accuracy. Inclusion of these elements provides landmarks from which localization may be improved, and provides opportunity for landmark-based loop closures. Also, these shape and surface elements relay important semantic scene information otherwise unavailable from the current dense map representation (an unorganized collection of colored 3D points).

Additionally, this thesis has investigated the theory and implementation of the direct visual odometry algorithm leveraged by the dense SLAM approach. This includes the development of a simplified reference implementation more amenable to understanding and extension than currently available open source implementations. Finally, this thesis presents an efficient method for least squares planar surface fitting for depth data. Benchmark results show this method is faster than two methods available in the OpenCV library, and should prove useful for fast planar surface estimation.

Future research will explore application of the SLAM platform and geometric map elements to extract and track geometric primitives. These primitives can then be used to infer the structure of more complex objects and geometries. Specifically, the estimated infinite plane surfaces currently used can be bounded, which should provide additional constraints for localization and a more precise representation of local planar scene structure. Additional geometries such as edges and corners can be included in the as map elements, which along with estimated bounded planar regions can used to aid in the identification of higher-order shapes.

REFERENCES

[1] Itseez, *The OpenCV Reference Manual*, 3.1.0 ed.

[2] R. Tsai, “A versatile camera calibration technique for high-accuracy 3d machine vision metrology using off-the-shelf tv cameras and lenses,” *IEEE Journal on Robotics and Automation*, vol. 3, pp. 323–344, August 1987.

[3] Z. Zhang, “A flexible new technique for camera calibration,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 22, pp. 1330–1334, Nov 2000.

[4] J. Diebel, “Representing attitude: Euler angles, unit quaternions, and rotation vectors,” 2006.

[5] G. Gallego and A. Yezzi, “A compact formula for the derivative of a 3-d rotation in exponential coordinates,” *Journal of Mathematical Imaging and Vision*, vol. 51, pp. 378–384, Mar 2015.

[6] R. M. Murray, S. S. Sastry, and L. Zexiang, *A Mathematical Introduction to Robotic Manipulation.* Boca Raton, FL, USA: CRC Press, Inc., 1st ed., 1994.

[7] C. Bregler and J. Malik, “Tracking people with twists and exponential maps,” in *Proceedings. 1998 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No.98CB36231)*, pp. 8–15, Jun 1998.

[8] K. Levenberg, “A method for the solution of certain non-linear problems in least squares,” *Quarterly of Applied Math*, vol. 2, pp. 164–168, 1944.

[9] D. Marquardt, “An algorithm for least-squares estimation of nonlinear parameters,” *Journal of the Society for Industrial and Applied Mathematics*, vol. 11, 1963.

[10] M. A. Fischler and R. C. Bolles, “Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography,” *Commun. ACM*, vol. 24, pp. 381–395, June 1981.

[11] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics).* Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[12] G. Bradski, “The opencv library,” *Dr. Dobb’s Journal of Software Tools*, 2000.

[13] R. B. Rusu and S. Cousins, “3D is here: Point Cloud Library (PCL),” in *IEEE International Conference on Robotics and Automation (ICRA)*, (Shanghai, China), May 9-13 2011.

[14] A. Comport, E. Malis, and P. Rives, “Real-time quadrifocal visual odometry,” *Int. J. Rob. Res.*, vol. 29, pp. 245–266, Feb. 2010.

[15] F. Steinbrücker, J. Sturm, and D. Cremers, "Real-time visual odometry from dense rgb-d images," in *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pp. 719–722, Nov 2011.

[16] T. Tykkälä, C. Audras, and A. I. Comport, "Direct iterative closest point for real-time visual odometry," in *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, pp. 2050–2056, Nov 2011.

[17] C. Kerl, J. Sturm, and D. Cremers, "Dense visual slam for rgb-d cameras," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 2100–2106, Nov 2013.

[18] C. Kerl, J. Sturm, and D. Cremers, "Robust odometry estimation for rgb-d cameras," in *2013 IEEE International Conference on Robotics and Automation*, pp. 3748–3754, May 2013.

[19] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*, IJCAI'81, (San Francisco, CA, USA), pp. 674–679, Morgan Kaufmann Publishers Inc., 1981.

[20] B. W. Babu, S. Kim, Z. Yan, and L. Ren, "σ-dvo: Sensor noise model meets dense visual odometry," in *2016 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 18–26, Sept 2016.

[21] S. Baker and I. Matthews, "Lucas-kanade 20 years on: A unifying framework," *Int. J. Comput. Vision*, vol. 56, pp. 221–255, Feb. 2004.

[22] D. Gutiérrez-Gómez, W. Mayol-Cuevas, and J. J. Guerrero, "Inverse depth for accurate photometric and geometric error minimisation in rgb-d dense visual odometry," in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 83–89, May 2015.

[23] S. Thrun, W. Burgard, and D. Fox, *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.

[24] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based slam," *IEEE Intelligent Transportation Systems Magazine*, vol. 2, pp. 31–43, winter 2010.

[25] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "G2o: A general framework for graph optimization," in *2011 IEEE International Conference on Robotics and Automation*, pp. 3607–3613, May 2011.

[26] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon, "Kinectfusion: Real-time dense surface mapping and tracking," in *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pp. 127–136, Oct 2011.

[27] S. Rusinkiewicz and M. Levoy, “Efficient variants of the icp algorithm,” in *Proceedings Third International Conference on 3-D Digital Imaging and Modeling*, pp. 145–152, 2001.

[28] T. Whelan, M. Kaess, M. Fallon, H. Johannsson, J. Leonard, and J. McDonald, “Kintinuous: Spatially extended kinectfusion,” in *RSS Workshop on RGB-D: Advanced Reasoning with Depth Cameras*, (Sydney, Australia), Jul 2012.

[29] R. F. Salas-Moreno, R. A. Newcombe, H. Strasdat, P. H. J. Kelly, and A. J. Davison, “Slam++: Simultaneous localisation and mapping at the level of objects,” in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1352–1359, June 2013.

[30] B. Drost, M. Ulrich, N. Navab, and S. Ilic, “Model globally, match locally: Efficient and robust 3d object recognition,” in *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pp. 998–1005, June 2010.

[31] R. F. Salas-Moreno, B. Glocken, P. H. J. Kelly, and A. J. Davison, “Dense planar slam,” in *2014 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pp. 157–164, Sept 2014.

[32] J. Engel, T. Schöps, and D. Cremers, “Lsd-slam: Large-scale direct monocular slam,” in *Computer Vision – ECCV 2014* (D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds.), (Cham), pp. 834–849, Springer International Publishing, 2014.

[33] J. Engel, J. Stückler, and D. Cremers, “Large-scale direct slam with stereo cameras,” in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 1935–1942, Sept 2015.

[34] M. J. D. Powell, “An efficient method for finding the minimum of a function of several variables without calculating derivatives,” *The Computer Journal*, vol. 7, no. 2, p. 155, 1964.

[35] M. L. A. Lourakis and A. A. Argyros, “Is levenberg-marquardt the most efficient optimization algorithm for implementing bundle adjustment?,” in *Tenth IEEE International Conference on Computer Vision (ICCV’05) Volume 1*, vol. 2, pp. 1526–1531 Vol. 2, Oct 2005.

[36] 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*, pp. 573–580, Oct 2012.

[37] J. Papadakis and A. R. Willis, “Real-time surface fitting to rgbd sensor data,” in *SoutheastCon 2017*, pp. 1–7, March 2017.

[38] J. P. Lewis, “Fast template matching,” in *Vision interface*, vol. 95, pp. 15–19, 1995.

[39] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, vol. 1, pp. I–511–I–518 vol.1, 2001.

[40] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky, "Instant architecture," *ACM Transactions on Graphics*, vol. 22, no. 3, pp. 669–677, 2003.

[41] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool, "Procedural modeling of buildings," *ACM Transactions on Graphics*, vol. 25, pp. 614–623, July 2006.

[42] K. Dylla, P. Müller, A. Ulmer, S. Haegler, and B. Fischer, "Rome reborn 2.0: A framework for virtual city reconstruction using procedural modeling techniques," in *Proceedings of Computer Applications and Quantitative Methods in Archaeology*, 2009.

[43] A. Gupta, A. A. Efros, and M. Hebert, "Blocks world revisited: Image understanding using qualitative geometry and mechanics," in *European Conference on Computer Vision(ECCV)*, 2010.

[44] H. Kim and A. Hilton, "Block world reconstruction from spherical stereo image pairs," *Computer Vision and Image Understanding*, vol. 139, pp. 104–121, Oct. 2015.

[45] O. Teboul, L. Simon, P. Koutsourakis, and N. Paragios, "Segmentation of building facades using procedural shape prior," in *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–8, 2010.

[46] P. Muller, G. Zeng, P. Wonka, and L. V. Gool, "Image-based procedural modeling of facades," *ACM Transactions on Graphics*, vol. 26, no. 3, pp. 1–9, 2007.

[47] M. Mathias, A. Martinovic, J. Weissenberg, and L. V. Gool, "Procedural 3d building reconstruction using shape grammars and detectors," in *3DIMPVT 2011*, May 2011.

[48] P. Dosch, K. Tombre, C. Ah-Soon, and G. Masini, "A complete system for the analysis of architectural drawings," *International Journal on Document Analysis and Recognition*, vol. 3, pp. 102–116, 2000.

[49] F. Bao, D.-M. Yan, N. J. Mitra, and P. Wonka, "Generating and exploring good building layouts," *ACM Trans. Graph.*, vol. 32, pp. 122:1–122:10, July 2013.

[50] X. Yin, P. Wonka, and A. Razdan, "Generating 3d building models from architectural drawings: A survey," *Computer Graphics and Applications*, vol. 29, no. 1, pp. 20–30, 2009.

[51] M. Mathias, A. Martinovic, J. Weissenberg, and L. V. Gool, "Procedural 3D building reconstruction using shape grammars and detectors," in *3D Imaging, Modeling, Processing, Visualization and Transmission (3DIMPVT), 2011 International Conference on*, (Washington, DC), pp. 304–311, IEEE Computer Society, 2011.

[52] J. Weissenberg, *Inverse Procedural Modelling and Applications.* PhD thesis, ETH Zurich, 2014.

[53] B. Hohmann, U. Krispel, S. Havemann, and D. Fellner, "Cityfit: High-quality urban reconstructions by fitting shape grammers to images and derived textured point clouds," in *ISPRS International Workshop*, pp. 1–8, 2009.

[54] M. D. Fairchild, *Color Appearance Models.* Wiley, 3rd ed., 2013.

[55] J. Hopcroft and R. Tarjan, "Algorithm 447: Efficient algorithms for graph manipulation," *Commun. ACM*, vol. 16, pp. 372–378, June 1973.

[56] B. T. Phong, "Illumination for computer generated pictures," *Commun. ACM*, vol. 18, pp. 311–317, June 1975.

[57] H. Badino, D. Huber, Y. Park, and T. Kanade, "Fast and accurate computation of surface normals from range images," in *2011 IEEE International Conference on Robotics and Automation*, pp. 3084–3091, May 2011.