FROM INTEREST POINTS TO MAP TRANSFORMATION: A DISCUSSION OF RGB-D SLAM AND ITS APPLICATIONS

BY

ALEXANDER J VON ALT

THESIS

Submitted in partial fulfillment of the requirements
for the degree of Master of Science in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2013

Urbana, Illinois

Adviser:

Professor Seth Andrew Hutchinson

Abstract

This thesis examines SLAM using a single handheld projected-IR RGB-D camera. It focuses primarily on the use of different interest point detectors, their descriptor extractors, and their matchers in the estimation of pairwise alignment of frames captured using such a sensor. Such estimates give the initial pose estimate of each frame. They are encoded as edge constraints in the pose graph and integral in SLAM's recovery of the optimal trajectory of poses. BRISK and FREAK feature detectors using FLANN and brute-force matchers are analyzed. Their suitability for use in visual odometry estimation is investigated. A method of automatically tuning the BRISK-AGAST detector so as to produce a consistent number of features is developed and studied in the context of RGB-D SLAM.

An augmented reality application making use of RGB-D SLAM is also developed. This method uses RGB-D SLAM to create a map of an environment. The map is then distorted using affine transformations to change the dimensions and appearance of the environment. Views of this distorted environment are generated in real time at the camera's current pose. The effectiveness of the approach is considered and example outputs are provided.

To Grandma V: Another one for your book.

Acknowledgments

I'd like to thank my parents Chris and Mary von Alt for their unconditional support throughout the course of my education. From preschool to graduate school, they've been there for me. I'd like to thank my fellow denizens of room 1514 Beckman for their camaraderie and the many rich and interesting discussions we had. Most of all, I would like to thank my advisor Professor Seth Hutchinson for his guidance, support, and patience throughout the course of my study. Without his encouragement I probably never would have gotten into computer vision. For that and the many conversations we had shaping this project, I am eternally grateful.

Table of Contents

Chapter 1

Introduction

1.1 Background

In recent years robots have risen greatly in popularity, finding applications in a number of environments from packing pallets in factories to exploring the solar system. Two concepts fundamental to the design of robotic applications working in any environment are those of *localization* and *mapping*.

Localization is the process of determining the pose of a robot. Given a map of the robot's environment, localization algorithms discover the orientation of the robot – the direction that it is facing – and its location – the physical portion of the environment that the robot occupies.

Mapping is the process of creating a map of an environment. Mapping algorithms make use of the robot's sensor measurements and known pose to build a model of the environment describing such things as the position of landmarks, the location of objects, and the structure of the environment itself.

Simultaneous Localization and Mapping (SLAM) seeks to solve both the localization and mapping problems at once. Given a robot at an unknown pose in an unknown environment, SLAM uses the robot's sensor measurements to build a map of the environment and localize the robot in this map.

This thesis examines a popular approach to SLAM using RGB-D cameras such as the Kinect, sensors that record both color images of an environment and the depth of objects in those images. The RGB-D SLAM System [1] is utilized to investigate the use of different image features in the estimation of visual odometry during SLAM. An augmented reality application is also developed. This program modifies the structure and appearance of a map generated using the RGB-D SLAM System [1] and generates photo-realistic views of this map in real time.

1.2 Problem Description and Objectives

RGB-D SLAM is developed in [1], [2], and [3] as a GraphSLAM problem. A pose graph, with each node representing a distinct camera pose, is constructed and optimized. Edge constraints that represent the relative change in pose between nodes are found using visual odometry.

Visual odometry [4],[5],[6] between two camera poses, also known as pairwise alignment [2], is found by computing the rigid transformation between feature points found in the image at one pose and matching features in the other pose's image. Pairwise alignment is typically done using slower, more accurate feature points such as SIFT [7] in [3] and SURF [8] in [1].

The first part of this thesis examines the use of the faster feature detection systems BRISK [9] and FREAK [10] for visual odometry. A self-tuning BRISK detector termed the Adjustable BRISK detector is also developed and tested. The best parameter configuration for each of these features is determined and their relative efficacy in the pairwise alignment task is evaluated.

The second part of this thesis details the development of an augmented reality tool using the maps and poses generated by the RGB-D SLAM System. This application allows users to distort the structure of the mapped environment using affine transformations, generating consistent, photo-realistic views of the distorted environment in real time.

1.3 Organization of Thesis

The remainder of this thesis is organized as follows.

Chapter 2 will introduce and summarize fundamental concepts relating to tools and algorithms used in this thesis. It is a summary of much of the background that enabled the work done later in the thesis.

Chapter 3 introduces Simultaneous Localization and Mapping. It discusses the goals of SLAM and presents some of the core probabilistic SLAM techniques. Approaches to solving SLAM using an RGB-D camera are discussed. Lastly, the RGB-D SLAM System [1] used in much of this thesis is explained in detail.

Chapter 4 presents work done to test additional feature detectors, descrip-

tor extractors, and feature matchers in the RGB-D SLAM framework. Analysis of the tunable parameters of these tools is performed and a method of evaluating the performance of a SLAM algorithm using its discrete trajectory estimate is presented.

Chapter 5 presents the experimental results for the feature detectors presented in Chapter 4. Results are discussed and the effects of various parameters are analyzed. Detection, extraction, and matching techniques are compared and the best feature detection configuration for visual odometry is identified.

Chapter 6 presents work done involving map-based scene distortion. A method of changing the appearance of a scene in a consistent manner is developed and experimental results are presented.

1.4 Important Terms and Abbreviations

- Adjustable BRISK: A version of the BRISK-AGAST detector developed in this thesis.
- AGAST: Adaptive and Generic Corner Detection Based on the Accelerated Segment Test - A corner feature detector. [11]
- AGAST-BRISK: A version of AGAST used to detect feature points for the BRISK extractor. [9]
- BRISK: Binary Robust Invariant Scalable Keypoints - A feature detection algorithm which produces binary descriptors. [9]
- brute-force: A descriptor matching algorithm which examines all possible matches.
- brute-force-Hamming: A brute-force matching algorithm which uses the Hamming distance measure.
- brute-force-SSE3: A brute-force matching algorithm which measures Hamming distance in 1 byte increments. [10]
- FLANN: Fast Library for Approximate Nearest Neighbors - A matching algorithm for feature descriptors. [12]

- FLANN-LSH: A FLANN detector which measures distance using the Hamming distance function. [13] [12]
- FREAK: Fast Retina Keypoints - A feature detection algorithm similar to BRISK which also produces binary descriptors. [10]
- ICP: Iterative Closest Point. A point alignment algorithm. [14]
- NMS: Non-maxima Suppression - A method of selecting local maxima pixels in an image or scale-pyramid. In the context of AGAST, this method selects local maxima that are unique, ensuring that selected pixels have a greater intensity or score than all other neighboring pixels.
- RANSAC: Random Sampling and Consensus - A model fitting algorithm for data with outliers. [15]
- RGB: Red Green Blue, often referring to an image with red, green, and blue color channels.
- RGB-D: Red Green Blue Depth, often referring to a pair of RGB and depth images, where the depth image gives the depth of the corresponding RGB pixel.
- RGB-D SLAM: The RGB-D camera based SLAM algorithm developed at the University of Freiburg that is experimented with in thesis. [1]
- SIFT: Scale Invariant Feature Transform - A popular feature detection algorithm developed by Lowe. [7]
- SLAM: Simultaneous Localization and Mapping
- SURF: Speeded-Up Robust Features - A popular feature detection algorithm making use of Harr wavelets. [8]

Chapter 2

General Concepts

2.1 Chapter Summary

This chapter serves as a catch-all for the techniques and mathematical concepts that do not fall under the umbrella of SLAM (Chapter 3). Each section explains the related concept in sufficient detail to enable reader to understand its application in other parts of the thesis. Each section also serves as a primer on its subject for future reference. It explains the fundamentals of the concept and cites useful related works for a deeper discussion.

2.2 Notation

The following notations and conventions will be used in this thesis:

- Variables written in bold lower-case represent vectors.
- Variables written in bold upper-case represent matrices.
- Matrix elements will be referred to in row, column order. Pairs of trailing subscripts may be used to indicate subsets of a matrix.

  A 6×8 matrix, $\mathbf{M}$, is one with six rows and eight columns.

  $\mathbf{M}_{i,j}$ is the element of the matrix located at the ith row and jth column.

  $\mathbf{M}_{3:5,4:6}$ is the submatrix of $\mathbf{M}$ located between rows 3 and 5 and columns 4 and 6.
- Reference frame is indicated in the style of [16], using leading superscripts and subscripts.
- The coordinate frame in which a variable is expressed is indicated by the leading superscript.

- The leading subscript indicates the source coordinate frame, when transforming between two reference frames.
- Trailing superscripts indicate inverse and transpose operations.

As an example, consider a point $\mathbf{v}$ and two coordinate systems $C1$ and $C2$. ${}^{C1}\mathbf{v}$ gives the location of $\mathbf{v}$ in $C1$, and ${}^{C2}\mathbf{v}$ is the location in $C2$ coordinates. The transformation ${}^{C1}_{C2}\mathbf{T}$ relates the two coordinate systems such that:

$$ {}^{C1}\mathbf{v} \;=\; {}^{C1}_{C2}\mathbf{T}\; {}^{C2}\mathbf{v} $$

The relation between ${}^{C1}_{C2}\mathbf{T}$ and ${}^{C2}_{C1}\mathbf{T}$ can be expressed as:

$$ {}^{C1}_{C2}\mathbf{T} \;=\; {}^{C2}_{C1}\mathbf{T}^{-1} $$

The superscript T indicates the transpose of a matrix:

$$ {}^{C2}_{C1}\mathbf{T}^{\mathrm{T}} $$

2.3 ROS and OpenCV

OpenCV [17] and ROS [18] are two powerful software tools used in this thesis. The Robotic Operating System (ROS) creates a modular framework for programs to perform I/O operations and interact. Using ROS it is a simple matter to switch between real time sensor data, prerecorded data, and simulated sensor data as inputs to a program. Furthermore, ROS has modules which perform much of the low level driver work for interfacing with sensors. Using ROS, interfacing with a Kinect is as simple as booting up the related driver and "subscribing" to the "topic" Kinect images are published on.

OpenCV is a library of common computer vision algorithms created for C++ and Python. OpenCV contains classes and functions which handle much of the low level work associated with modifying and displaying images. The library contains a number of the most common algorithms for filtering images and detecting and working with interest points. There is also a wealth of algorithms performing higher level tasks such as camera calibration, object detection, and 3-D reconstruction.

2.4 Coordinate Frames, Transformations, and Pose

2.4.1 Coordinate Frames

In many applications of robotics it is useful and necessary to describe the position of objects in an environment both absolutely and relative to each other. Coordinate frames provide a useful language with which to do so. This thesis will deal primarily in a three-dimensional space of real numbers $\mathbb{R}^3$, and with 3-D coordinate frames describing it.

These coordinate frames can be considered as bases for a three-dimensional vector space $\mathbb{R}^3$ that we wish to describe. Each basis is made up of three 3×1 vectors $\langle \mathbf{b}_1, \mathbf{b}_2, \mathbf{b}_3 \rangle$ [19]. These vectors are linearly independent and span $\mathbb{R}^3$, meaning each point in $\mathbb{R}^3$ is described by a single and unique combination of the scaled vectors. Additionally, these vectors are constrained to be orthonormal, meaning they are of unit length $||\mathbf{b}_i|| = 1, i \in 1 \dots 3$ and mutually orthogonal $\mathbf{b}_i \cdot \mathbf{b}_j = 0; i \neq j; i, j \in 1 \dots 3$. This constraint simplifies the calculation of transformations relating two coordinate frames.

In accordance with ROS conventions [20], these coordinate frames are organized so that each axis, pointing in the direction of the corresponding basis vector, is ordered in a forward-left-up format, with the x axis pointing forward, the y axis pointing left, and the z axis pointing up. These coordinate systems are right-handed, meaning they satisfy the right hand rule for determining the orientation of cross products. The right hand rule states that, for vectors $\mathbf{u}$ and $\mathbf{v}$, the resulting cross product $\mathbf{u} \times \mathbf{v}$ will point in the direction of the thumb of a right hand, when that hand is aligned with $\mathbf{u}$ and curled toward $\mathbf{v}$. Right-handed coordinate systems satisfy the property $\mathbf{b}_1 \times \mathbf{b}_2 = \mathbf{b}_3$.

2.4.2 Transformations between Coordinate Frames

The relationship between two coordinate frames can be expressed in terms of the angles between their corresponding axes and the relative locations of their origins. Section 2.2 of [16] has an excellent explanation of describing one coordinate system $B = \langle \mathbf{x}_B, \mathbf{y}_B, \mathbf{z}_B \rangle$ in terms of another $A = \langle \mathbf{x}_A, \mathbf{y}_A, \mathbf{z}_A \rangle$, using the rotation matrix ${}^A_B\mathbf{R}$ and the translation term ${}^A_B\boldsymbol{\rho}$.

The basis vectors of B, $\mathbf{x}_B$, $\mathbf{y}_B$, and $\mathbf{z}_B$, can be expressed in terms of A

by projecting them onto each of A's basis vectors. Equation (2.1) illustrates this process for $\mathbf{x}_B$.

$$ {}^{A}\mathbf{x}_B = \begin{bmatrix} \mathbf{x}_B \cdot \mathbf{x}_A \\ \mathbf{x}_B \cdot \mathbf{y}_A \\ \mathbf{x}_B \cdot \mathbf{z}_A \end{bmatrix} \tag{2.1} $$

Organizing these projections columnwise yields the rotation matrix ${}^{A}_{B}\mathbf{R}$ that describes B relative to A. This rotation matrix, expanded in Equation (2.2), is known as the direction cosine matrix, as the dot product between two unit vectors is equal to the cosine of the angle between them. The rotation matrix describes the difference in orientation between the two reference frames. The translation term ${}^{A}_{B}\boldsymbol{\rho}$ simply gives the location of the origin of B in terms of A.

$$ {}^{A}_{B}\mathbf{R} = \begin{bmatrix} {}^{A}\mathbf{x}_B & {}^{A}\mathbf{y}_B & {}^{A}\mathbf{z}_B \end{bmatrix} = \begin{bmatrix} \mathbf{x}_B \cdot \mathbf{x}_A & \mathbf{y}_B \cdot \mathbf{x}_A & \mathbf{z}_B \cdot \mathbf{x}_A \\ \mathbf{x}_B \cdot \mathbf{y}_A & \mathbf{y}_B \cdot \mathbf{y}_A & \mathbf{z}_B \cdot \mathbf{y}_A \\ \mathbf{x}_B \cdot \mathbf{z}_A & \mathbf{y}_B \cdot \mathbf{z}_A & \mathbf{z}_B \cdot \mathbf{z}_A \end{bmatrix} \tag{2.2} $$

Equation (2.3) shows how the terms $\left\langle {}^{A}_{B}\mathbf{R}, {}^{A}_{B}\boldsymbol{\rho} \right\rangle$ can be used to express a point described in the B coordinate frame, ${}^{B}\mathbf{d}$, in terms of A.

$$ {}^{A}\mathbf{d} = {}^{A}_{B}\mathbf{R}\ {}^{B}\mathbf{d} + {}^{A}_{B}\boldsymbol{\rho} \tag{2.3} $$

2.4.3 Homogeneous Coordinates

Homogeneous coordinates are frequently used in the field of computer vision for tasks such as transformation and projection. Typically used with 2-D and 3-D space, homogeneous coordinates represent a point using one more dimension than that of the point. A 2-D vector $\begin{bmatrix} u & v \end{bmatrix}^T$ is represented in homogeneous coordinates by $\begin{bmatrix} u & v & 1 \end{bmatrix}^T$, and a 3-D vector $\begin{bmatrix} x & y & z \end{bmatrix}^T$ is represented by $\begin{bmatrix} x & y & z & 1 \end{bmatrix}^T$.

Homogeneous coordinates are only uniquely defined up to a scale factor.

Scalar multiples of homogeneous vectors are considered equal.

$$\begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \alpha \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} \frac{x}{w} \\ \frac{y}{w} \\ \frac{z}{w} \\ 1 \end{bmatrix}$$

Equation (2.4) demonstrates conversion to homogeneous coordinates. A vector $\mathbf{p}$ is converted into a vector in homogeneous coordinates $\mathbf{p}_h$ by appending a one to the end of the vector.

$$\mathbf{p} = \begin{bmatrix} u \\ v \end{bmatrix} \qquad \mathbf{p}_h = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \tag{2.4}$$

For a given $n \times n$ matrix $\mathbf{M}$ operating on the vector $\mathbf{b}$, an equivalent matrix $\mathbf{M}_h$ that operates on the vector $\mathbf{b}_h$ in homogeneous coordinates can be obtained by appending a row and column of zeros to the matrix and setting $\mathbf{M}_{h(n,n)}$ to 1. This process is demonstrated in Equation (2.5).

$$\begin{array}{llll} \mathbf{b} = \mathbf{M}\mathbf{a} & \mathbf{a} = \begin{bmatrix} 1 \\ 2 \end{bmatrix} & \mathbf{M} = \begin{bmatrix} 2 & 3 \\ 4 & 5 \end{bmatrix} & \mathbf{b} = \begin{bmatrix} 8 \\ 14 \end{bmatrix} \\ \mathbf{b}_h = \mathbf{M}_h\mathbf{a}_h & \mathbf{a}_h = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} & \mathbf{M}_h = \begin{bmatrix} 2 & 3 & 0 \\ 4 & 5 & 0 \\ 0 & 0 & 1 \end{bmatrix} & \mathbf{b} = \begin{bmatrix} 8 \\ 14 \\ 1 \end{bmatrix} \end{array} \tag{2.5}$$

A vector in homogeneous coordinates $\mathbf{c}_h$ can be converted to one in normal coordinates $\mathbf{c}$ by normalizing the homogeneous vector to $\bar{\mathbf{c}}_h$ and then post-multiplying it with the transform $\mathbf{S}_2$ for the 2-D case and $\mathbf{S}_3$ for the 3-D case, as seen in Equation (2.6).

$$\begin{array}{llll} & \mathbf{S}_2 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} & \mathbf{S}_3 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} & \\ \mathbf{c}_h = \begin{bmatrix} 1 \\ 2 \\ 5 \end{bmatrix} & \bar{\mathbf{c}}_h = \left(\frac{1}{\mathbf{c}_{h(3,1)}}\mathbf{c}_h\right) = \begin{bmatrix} 0.2 \\ 0.4 \\ 1 \end{bmatrix} & \mathbf{c} = \mathbf{S}_2\bar{\mathbf{c}}_h = \begin{bmatrix} 0.2 \\ 0.4 \end{bmatrix} & \end{array} \tag{2.6}$$

Note that normalizing a homogeneous vector has a meaning apart from typical normalization. Here the vector is not scaled so that the magnitude of the normalized vector is 1; rather the vector is scaled so that the last element of the vector is 1.

Homogeneous coordinates are useful for writing compact equations that combine translation and rotation operations. Equation (2.3) is expressed more succinctly in homogeneous coordinates in Equation (2.7).

$$ {}^{A}\mathbf{d}_h = {}^{A}_{B}\mathbf{T}\ {}^{B}\mathbf{d}_h \tag{2.7} $$

The components of this equation are made explicit in Equation (2.8). The 4×4 homogeneous transformation matrix is a composition of the rotation matrix and translation vector. The first 3×3 elements are the rotation matrix.

$$ {}^{A}_{B}\mathbf{T}_{(1:3,1:3)} = {}^{A}_{B}\mathbf{R} $$

The last column is the homogeneous version of the translation vector.

$$ {}^{A}_{B}\mathbf{T}_{1:4,4} = {}^{A}_{B}\boldsymbol{\rho}_h = \begin{bmatrix} {}^{A}_{B}\boldsymbol{\rho} \\ 1 \end{bmatrix} $$

The remaining elements of the transformation matrix are all zeros.

$$ {}^{A}_{B}\mathbf{T}_{4,1:3} = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix} $$

$$ \begin{bmatrix} {}^{A}\mathbf{d} \\ 1 \end{bmatrix} = \begin{bmatrix} {}^{A}_{B}\mathbf{R} & {}^{A}_{B}\boldsymbol{\rho} \\ 0 \quad 0 \quad 0 & 1 \end{bmatrix} \begin{bmatrix} {}^{B}\mathbf{d} \\ 1 \end{bmatrix} \tag{2.8} $$

2.4.4 Pose

An object's *pose* describes its position in space relative to an external reference frame. The pose can be broken down into two components – its location and its orientation [21]. An object's location describes its position relative to the origin of the reference frame. Its orientation describes the direction in which the object is pointing.

Pose representation is typically approached by rigidly fixing a reference frame to the object in question. The pose of the object relative to an external

reference frame is expressed as the transformation between the corresponding frames [16].

There are several approaches to representing pose as a transformation between the object frame C and the external frame W. In [21] pose is represented with a 6×1 vector.

$$\begin{bmatrix} tx & ty & tz & \psi & \theta & \phi \end{bmatrix}^T$$

Here x, y, and z are the location of the origin of C in W coordinates.

$$\begin{bmatrix} tx & ty & tz \end{bmatrix}^T = {}^W_C\boldsymbol{\rho}$$

The yaw, pitch, and roll angles ψ, θ, and ϕ specify rotation about the ${}^W\mathbf{z}$, ${}^W\mathbf{y}$,and ${}^W\mathbf{x}$ axes respectively in order to obtain C.

ROS represents the pose of C in W as the translation from the origin of W to the origin of C and the rotation of the axes of C in W [20]. The rotation term is given in quaternion format, which essentially specifies a vector about which to rotate and an angle to rotate, in order to obtain the desired orientation. Quaternions will be discussed in greater depth in Section 2.7.

The complete pose in ROS is represented as a 6×1 vector.

$$\begin{bmatrix} tx & ty & tz & w & x & y & z \end{bmatrix}^T$$

Here tx, ty, and tz are the translation terms and w, x, y, and z are the quaternion terms.

The book [16] and paper [1] represent pose using a direction cosine matrix and a translation term. In accordance with Equation (2.3), the pose would be given by the pair $\langle {}^W_C\mathbf{R},\ {}^W_C\boldsymbol{\rho} \rangle$.

This thesis will primarily use a quaternion based notation $\langle\ {}^W_C\mathrm{Q},\ {}^W_C\boldsymbol{\rho} \rangle$, where the quaternion ${}^W_C\mathrm{Q}$ describes the rotational component of the rigid transformation representing the pose and ${}^W_C\boldsymbol{\rho}$ gives the translational component. For more details on quaternions refer to Section 2.7.

In certain circumstances, an alternate notation $\mathbf{T}$ will be used as a pose representation for compactness and clarity. The transformation $\mathbf{T}$ is a homogeneous transformation matrix with the format described in Equation (2.8).

This transformation representation is equivalent to the quaternion based representation $\langle \mathrm{Q}, \boldsymbol{\rho} \rangle$.

$$\mathbf{T} \Leftrightarrow \langle \mathrm{Q}, \boldsymbol{\rho} \rangle$$

2.4.5 Useful References

The linear algebra course notes [19] provide an excellent and succinct explanation of vector spaces and coordinate systems. The book [21] gives a brief overview of robot pose, but limits applications to planar robots and their 2-D poses. Chapter 1 of [16] provides a fairly detailed explanation of robot pose and dynamics. Chapter 2 details coordinate frames, transformations, and their representations in excellent detail.

2.5 3-D Projection

3-D projection provides the mathematical framework with which we can model the imaging process of a camera. In the context of this thesis, 3-D projection will be considered as any method that maps a point in 3-D space onto a 2-D plane in that space. This section will present information essential to an understanding of projection and the imaging process.

First, the fundamentals of projection and imaging will be explained in the context of basic orthographic projection and the orthographic camera model. The pinhole camera model will then be introduced, and perspective projection will be explained. Transformation from 3-D world coordinates to 2-D image coordinates will also be discussed. Finally, "inverse perspective projection" will be presented, where pixels in an image with known depth are reconstructed in three-dimensional space to form a point cloud.

2.5.1 Orthographic Camera Model

The orthographic camera model is one of the most basic camera models. It maps objects in a scene directly onto the imaging plane, without changing their vertical or horizontal position relative to the plane or their size. Objects imaged using the orthographic model appear the same size, regardless of their distance from the camera. Although the orthographic model is generally

inferior to the pinhole camera model discussed later in this section, it is useful when the camera is at an approximately constant distance from all objects in the scene [22]. Examples of such cases include aerial and satellite photography. The projection method utilized by the orthographic camera model is orthographic projection.

2.5.2 Orthographic Projection

Orthographic projection is inspired by the orthographic camera model. It is a form of affine projection where distance and the ratio of distances are preserved along axes orthogonal to the projection axis. Any information encoded along the projection axis is lost in entirety. The term "orthographic projection" comes from the orthogonal relationship between the projection lines and the plane of projection.

For this explanation, orthographic projection of a 3-D point ${}^{C}\mathbf{p}$ onto a plane ${}^{C}L$ will be considered. The point ${}^{C}\mathbf{p}$ and plane ${}^{C}L$ are defined relative to a camera frame C, with axes in the direction of the basis vectors ${}^{C}\mathbf{x}$, ${}^{C}\mathbf{y}$, and ${}^{C}\mathbf{z}$. To simplify calculations, ${}^{C}L$ is defined to be a distance z_o from the origin, such that, given in point-normal form, the plane contains the point $(0, 0, z_o)$ and the camera frame basis vector ${}^{C}\mathbf{z}$ is normal to it. If this is not the case in practice, it is a simple matter to define a new reference frame $\hat{C}$ with the relationship ${}^{\hat{C}}\mathbf{p} = {}^{\hat{C}}_{C}\mathbf{D}\ {}^{C}\mathbf{p}$, such that the above property holds for $\hat{C}$. Points would then be transformed into this new frame prior to projection.

The orthographic projection ${}^{C}\mathbf{p}'$ of ${}^{C}\mathbf{p}$ onto ${}^{C}L$ is formed according to Equation (2.9). The 3-D points ${}^{C}\mathbf{p}$ and ${}^{C}\mathbf{p}'$ and the orthographic projection matrix O are defined in homogeneous coordinates.

$$ {}^{C}\mathbf{p}' = O\ {}^{C}\mathbf{p} \tag{2.9} $$

$$ {}^{C}\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \quad , \quad O = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & z_o \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad , \quad {}^{C}\mathbf{p}' = \begin{bmatrix} p'_x \\ p'_y \\ z_o \\ 1 \end{bmatrix} $$

As previously mentioned, orthographic projection and the orthographic camera model work fairly well in scenes that do not have much variety in

depth. When there are significant differences in the relative depth of objects, images generated using the orthographic model appear unrealistic. The pinhole camera model and its associated perspective projection method produce much better results in such cases.

2.5.3 The Pinhole Camera Model

The pinhole camera model is an imaging model more in line with a human being's perception of depth. Objects imaged using the pinhole camera model appear smaller as their depth in the scene increases. The pinhole camera model assumes all light used to form an image passes through a single central point – the pinhole of the camera. Light passing through the pinhole reflects off the imaging plane and an inverse image of the scene is formed. Since light in this model travels in a straight line through the pinhole, and the focal length, the shortest distance between the pinhole and imaging plane, is typically much less than the depth of objects in the scene, a projected point lying on the image plane will generally be at a much smaller vertical and horizontal distance from the pinhole than the point's real world counterpart. As an object moves closer to the image, the angle between the projecting light rays and the imaging plane decreases and projected points appear farther away from the optical center of the image. This has the net effect of making an imaged object appear larger as its scene depth decreases.

2.5.4 Perspective Projection

The pinhole camera model inspires the perspective projection technique. In perspective projection, the location of a point projected onto a plane is scaled by its depth. This explanation will detail the projection of a 3-D point ${}^{C}\mathbf{p}$ onto a plane ${}^{C}L$ in the camera frame C, forming the projected point ${}^{C}\mathbf{p'}$. The plane ${}^{C}L$ is orthogonal to the z axis and contains the point $(0, 0, z_o)$. Equations and points are given in homogeneous coordinates, using homogeneous

vectors to represent points.

$$ {}^{C}\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \quad , \quad {}^{C}\mathbf{p}' = \begin{bmatrix} p'_x \\ p'_y \\ z_o \\ 1 \end{bmatrix} $$

Perspective projection of a point ${}^{C}\mathbf{p}$ depends on the depth p_z in addition to the lateral position p_x and height p_y. The horizontal and vertical locations of ${}^{C}\mathbf{p}'$ are inversely proportional to p_z. The x and y components p_x and p_y move closer to the optical center $(0, 0, z_o)$ as the depth of the point in the scene p_z increases. This relationship is formalized in the perspective formula equation given Equation (2.10) and the homogeneous normalization in Equation (2.11).

$$ {}^{C}\mathbf{p}'_h = \mathbf{D}_p \, {}^{C}\mathbf{p} \tag{2.10} $$

$$ {}^{C}\mathbf{p}'_h = \begin{bmatrix} p'_{hx} \\ p'_{hy} \\ p'_{hz} \\ w \end{bmatrix} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ \frac{p_z}{z_o} \end{bmatrix} \quad , \quad \mathbf{D}_p = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \frac{1}{z_o} & 0 \end{bmatrix} $$

$$ {}^{C}\mathbf{p}' = \frac{1}{w} \, {}^{C}\mathbf{p}'_h = \frac{1}{w} \begin{bmatrix} p_x \\ p_y \\ p_z \\ \frac{p_z}{z_o} \end{bmatrix} = \begin{bmatrix} \frac{p_x z_o}{p_z} \\ \frac{p_y z_o}{p_z} \\ z_o \\ 1 \end{bmatrix} \tag{2.11} $$

Perspective projection is often presented in a form that encapsulates a transformation from the 3-D camera frame C to the 2-D image frame I. This allows for direct conversion from world coordinates to image coordinates in one step. In this form, ${}^{C}L$ is called the normalized image plane and is defined to be a unit distance from the origin, such that $z_o = 1$. Substituting $z_o = 1$ into Equation (2.11) yields the simplified form in Equation (2.12).

$$ {}^{C}\mathbf{p}'(z_o = 1) = \begin{bmatrix} \frac{p_x}{p_z} \\ \frac{p_y}{p_z} \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} p'_x \\ p'_y \\ 1 \\ 1 \end{bmatrix} \tag{2.12} $$

The intrinsic calibration matrix of the camera ${}^{I}_{C}\mathbf{K}$ is the transformation from points on the normalized image plane in C to the image frame I. The intrinsic calibration matrix for the Kinect's RGB camera is given in Equation (2.13). This model allows for rectangular pixels of width l_u and height l_v on the normalized image plane. A translated origin is also made possible with the u_o and v_o terms, where $\begin{bmatrix} u_o & v_o & 1 \end{bmatrix}^T$ is the location of the camera frame origin in image coordinates.

$$ {}^{I}_{C}\mathbf{K} = \begin{bmatrix} l_u & 0 & u_o \\ 0 & l_v & v_o \\ 0 & 0 & 1 \end{bmatrix} \tag{2.13} $$

The transformation from 3-D homogeneous coordinates in C to 2-D homogeneous coordinates in I is given in Equation (2.14).

$$ {}^{L}\mathbf{p}' = \begin{bmatrix} {}^{I}_{C}\mathbf{K} & 0 \end{bmatrix} {}^{C}\mathbf{p}' \tag{2.14} $$

$$ {}^{L}\mathbf{p}' = \begin{bmatrix} p'_u \\ p'_v \\ 1 \end{bmatrix} \quad , \quad {}^{C}\mathbf{p}' = \begin{bmatrix} p'_x \\ p'_y \\ 1 \\ 1 \end{bmatrix} $$

Combining the Kinect calibration matrix with perspective projection given in Equation (2.10) and the transformation in Equation (2.14) yields the perspective projection formula in its standard form in Equation (2.15). Homogeneously normalizing Equation (2.15) yields Equation (2.17).

$$ \begin{aligned} {}^{L}\mathbf{p}'_h &= \begin{bmatrix} {}^{I}_{C}K & 0 \end{bmatrix} {}^{C}\mathbf{p}'_h \\ &= \begin{bmatrix} l_u & 0 & u_o & 0 \\ 0 & l_v & v_o & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} \\ &= \begin{bmatrix} p'_{hu} \\ p'_{hv} \\ w \end{bmatrix} = \begin{bmatrix} l_u p_x + p_z u_o \\ l_v p_y + p_z v_o \\ p_z \end{bmatrix} \end{aligned} \tag{2.15} $$

$$
\begin{aligned}
{}^{L}\mathbf{p}' &= \frac{1}{w}\,{}^{L}\mathbf{p}'_h && (2.16)\\
&= \begin{bmatrix} \dfrac{l_u p_x}{p_z} + u_o \\ \dfrac{l_v p_y}{p_z} + v_o \\ 1 \end{bmatrix} && (2.17)
\end{aligned}
$$

Perspective projection is used in this thesis to generate 2-D images from the point cloud maps produced by the RGB-D SLAM System.

2.5.5 Inverse Perspective Projection

Range sensors such as the Kinect can produce range images using perspective projection. Rather than RGB color intensities, range image pixels record the depth p_z of the scene along the projection lines. It is possible to recover the original structure of the imaged scene using these range images and inverse perspective projection. Using the intrinsic camera matrix ${}^{I}_{C}\mathbf{K}$, inverse perspective projection estimates the original location in C for each point in the image frame I.

The equation governing inverse perspective projection and the transformation from I to C, Equation (2.18), can be recovered by solving Equation (2.17) for p_x and p_y. Here, a pixel ${}^{I}\mathbf{p}'$ is transformed into the camera frame C and reconstructed in 3-D space to form the point ${}^{C}\mathbf{p}$. Using inverse perspective projection on every pixel in the range image produces a 3-D point cloud.

$$
{}^{C}\mathbf{p} = \begin{bmatrix} p_x \\ p_y \\ p_z \\ 1 \end{bmatrix} = \begin{bmatrix} \dfrac{p_z}{l_u}(p_u - u_o) \\ \dfrac{p_z}{l_v}(p_v - v_o) \\ p_z \\ 1 \end{bmatrix} \qquad (2.18)
$$

In cases where both color and range images have been recorded and the transformation ${}^{I}_{D}\mathbf{D}$ from the depth image frame D to the image frame I is known, colored point clouds can be created.

First a registered range image is generated. This registered range image expresses the depth measurements in the color image's coordinate frame. Pixels in the camera image with available range data are then reconstructed in the color camera world frame using Equation (2.18). In the result of each projection $\langle {}^{C}\mathbf{p}, (r, g, b) \rangle$, ${}^{C}\mathbf{p}$ is the location of the point in C, and (r, g, b) are the red, green, and blue color intensities.

Inverse perspective projection is frequently used with the Kinect and similar sensors. The resulting point clouds play a vital role in the visual odometry measured by most SLAM systems that use an RGB-D camera.

2.5.6 Useful References

Chapters 1 and 2 of [22] provide an excellent and much more thorough explanation of orthographic and perspective projection as well as other projection methods such as weak-perspective projection and paraperspective projection. Rich camera models are also developed that model lenses and even the human eye. Perspective projection is discussed in Section 1.4 of [23]. Camera models are discussed in Section 1.5 of [23].

2.6 Interest Points

Interest points or feature points are one of the fundamental tools of computer vision. In essence, interest point detection systems are tools for reducing the dimensionality of an image and summarizing it. These systems take an image, select distinct and identifiable components, and describe those components in a manner that facilitates comparison with other images. An image containing thousands or millions of data points (pixels), each with a rather uninformative description, is reduced to tens or hundreds of interest points with much more informative descriptions. Pixels with five associated values in the form of position and color intensity, are replaced by feature descriptors consisting of hundreds of values describing local image properties.

Interest points find their way into almost all corners of computer vision. Some applications of note are 3-D modeling [24], scene description [25], object detection [26], tracking [27], and image stitching [28].

Typical use of interest points can be broken into three main stages: detection, extraction, and matching. The detection process takes an image and outputs a list of feature locations. Extraction takes those feature locations and creates a "fingerprint" of each one using local image properties. The matcher takes two or more sets of descriptors and identifies which descriptors in one set are similar to a descriptor in the other set. Each of these processes will be explained in this section and interest point retrieval will be detailed for the BRISK [9] and FREAK [10] methods.

2.6.1 Characteristics of Good Interest Point

Exactly what makes a good interest point is a difficult question to answer. What is desired in an interest point varies based on the application. Properties that are valuable in some cases might be unimportant or even detrimental in others. However, there are some properties that are useful in most cases and are certainly worth considering.

- *Rotational and translational invariance* - A feature that is detected in one image should also be detected in a rotated or translated version of that image. Interest points are often used to match different pictures taken of the same scene. Movement of the camera parallel to the projection plane should not have a significant impact on what features are detected.

- *Perspective invariance* - A similar but much more elusive goal is perspective invariance. Here, it is desired that the detected features and their descriptors should not vary with changes in perspective as the camera moves throughout the scene. An object viewed from different camera poses should produce similar interest points and descriptors.

- *Scale invariance* - The same features should be selected in a scene regardless of the scale. Objects become smaller in an image as they move farther from the camera. If an object contains features when it is close to the camera, those same features should be detected at a greater distance as well.

- *Robustness to noise* - Interest points and descriptors should be robust to noise. Introducing noise into an image should not change which

features are detected. Likewise, a feature's descriptor should not change much as noise is introduced into the image. Furthermore, introducing small amounts of noise into the descriptor should not significantly alter matching results.

- *Robustness to lighting* - The intensities of pixels in an image vary with the amount of light present in a scene. It is desirable that the same features are selected regardless of the amount of ambient light. This has applications to object matching and tracking, where detection of the same object may be desired at day or night.

- *Computation and matching time* - The amount of time allowed to compute a set of interest points varies with the application. Real time applications operating at 30 frames per second (fps) might require simpler interest points than offline problems.

- *Memory* - Although this is not a typical problem, embedded applications or large datasets may necessitate fewer features or smaller descriptors in order to satisfy physical memory constraints.

2.6.2 Detectors

An interest point detector takes an image and outputs a set of feature points $F = \{f_1, f_2, \dots\}$ found in that image. Each feature point consists of a location in the image (u, v) and, optionally, a feature scale σ. The scale is the location of the feature in a scale pyramid, a stack of versions of the image that are increasingly more blurred and downsampled.

2.6.3 Extractors

Descriptor extractors take a set of interest point locations F and construct a descriptor $\mathbf{y}_i$ for each interest point f_i. This descriptor "describes" the appearance of its feature point using local image properties such as pixel intensities or gradient magnitudes. The two main descriptor categories discussed in this thesis are binary descriptors and real valued descriptors.

A binary descriptor is formed by comparing pixel patches sampled using a predefined pattern around an interest point. Each bit in the descriptor

records the result of a patch comparison, indicating which of a pair of patches is brighter.

Real valued descriptors consist of a vector of real numbers. SIFT [7] and SURF [8], two popular interest point solutions which use real valued descriptors, build their descriptors using multiple 2-D histograms, with each histogram sampling a patch near the interest point.

Binary descriptors are typically chosen for their rapid extraction and matching speeds. Real valued descriptors usually create a more accurate description of the interest point and produce more accurate matches at the cost of additional run time.

2.6.4 Matching

When speaking of distances and interest points, there are two main notions of distance. Distance between interest points typically refers to a Euclidean distance describing the difference in location in the image between two interest points. Distance between descriptors refers to a measure of similarity between descriptors. The smaller the distance between descriptors, the more similar the surroundings of the interest points used to generate the descriptors. Matchers use the distances between descriptors to establish correspondences between their associated interest points.

An interest point matcher identifies matching interest points in two images by examining their descriptors. It examines two sets of descriptors, a query set Y that is the set being matched, and a training set T that the matches are drawn from. The k nearest descriptors in T are found for each query descriptor in Y according to the distance function $\Delta(\mathbf{y}, \mathbf{t})$. It should be noted that these matches are one-way. The fact that a descriptor $\mathbf{t}$ is the best match in T for a query descriptor $\mathbf{y}_1$ does not mean there cannot be an even closer query descriptor $\mathbf{y}_2$ with is a better match for $\mathbf{t}$.

The matching process where the set of k matches $\hat{T}$ is found for a query descriptor $\mathbf{y}$ is expressed more formally in Equation (2.19), where $\Delta(\mathbf{y}, \mathbf{t})$ is the distance function used to measure distance between two descriptors $\mathbf{y}$ and $\mathbf{t}$.

$$Y = \{\mathbf{y}_1, \mathbf{y}_2, \ldots \mathbf{y}_m\}$$

For each $\mathbf{y}_\ell \in Y$ we have:

$$\begin{aligned}
\hat{T}_\ell &= \{\hat{\mathbf{t}}_1, \hat{\mathbf{t}}_2, \ldots, \hat{\mathbf{t}}_k\} \\
\hat{\mathbf{t}}_1 &= \underset{\mathbf{t}\in T}{\operatorname{argmin}}\ \Delta(\mathbf{y}_\ell, \mathbf{t}) \\
\hat{\mathbf{t}}_2 &= \underset{\mathbf{t}\in T\setminus\{\hat{\mathbf{t}}_1\}}{\operatorname{argmin}}\ \Delta(\mathbf{y}_\ell, \mathbf{t}) \\
&\vdots \\
\hat{\mathbf{t}}_k &= \underset{\mathbf{t}\in T\setminus\{\hat{\mathbf{t}}_1\cdots\cup\hat{\mathbf{t}}_{k-1}\}}{\operatorname{argmin}}\ \Delta(\mathbf{y}_\ell, \mathbf{t})
\end{aligned} \tag{2.19}$$

2.6.4.1 Distance Function

The distance function $\Delta(\mathbf{y}, \mathbf{t})$ used during the matching process depends on the type of descriptor supplied. The two most popular distance measures are Hamming distance for use with binary descriptors and squared Euclidean distance for use with real valued descriptors. They are detailed here for descriptors of size n.

- *Binary Descriptors* - Distance between binary descriptors such as those in BRISK and FREAK is measured using Hamming distance. The Hamming distance is a measure of dissimilarity between two bit strings. It is found by performing an element by element comparison of the two bit strings and counting the number of instances where the strings do not match. This is equivalent to XORing the two descriptors and then counting the number of ones. The Hamming distance equation is given in Equation (2.20).

$$\Delta_{hamming}(\mathbf{y}, \mathbf{t}) = \sum_{i=1}^{n} y_i \oplus t_i = \sum_{i=1}^{n} b(y_i, t_i) \tag{2.20}$$

  Here, $b(x, y)$ indicates bit inequality, and y_i and t_i are the i-th bits in the descriptors $\mathbf{y}$ and $\mathbf{t}$, respectively.

$$b(x, y) = \begin{cases} 1 & x \neq y \\ 0 & x = y \end{cases}$$

- *Real Valued Descriptors* - Distance between SIFT or SURF descriptors is found by computing the squared euclidean distance. The distance equation for these descriptors is given in Equation (2.21). A squared distance metric is preferred. For purposes of comparison, typical Euclidean distance measures provide no addition information of use while requiring a time-consuming square root operation in each measurement.

$$\Delta_{Euclidean}(\mathbf{y}, \mathbf{t}) = ||\mathbf{y} - \mathbf{t}||^2 = \sum_{i=1}^{n} (y_i - t_i)^2 \tag{2.21}$$

2.6.4.2 Popular Matchers

Finding the candidate matches is a challenge in itself. Two popular search methods are brute-force and FLANN. Brute-force is an exhaustive search where the distance between every query descriptor and every training descriptor is considered. This approach is suitable when there is a small number of features or the distance metric is fast to compute. As the number of features increases, the number of comparisons grows with the square of the number of features and brute-force is no longer useful.

FLANN (Fast Library for Approximate Nearest Neighbor) [12] is a tree based method that is valuable when the search space is large. It organizes the training descriptors into a k-d tree, so that large groups of descriptors can be eliminated with one comparison [12]. FLANN does not guarantee to find the absolute nearest neighbor, but will produce a good candidate at a much more rapid speed for large datasets. For values of k larger than one, $\hat{T}$ is populated by examining leaves neighboring the approximate nearest neighbor.

The FLANN method produces only approximate nearest neighbors. This will result in a larger number of mismatches than the brute-force method. Whether this is an acceptable trade-off for the run time gains is dependent upon the application, however there are methods of mitigating the additional risk. All matches should be filtered to remove poor matches. Using a stricter filtering criterion, it is possible to reduce the chance of accepting an incorrect match at the cost of an increased rejection rate for correct matches.

2.6.4.3 Match Filtering

The descriptor matcher algorithms do not allow for the possibility of no match being found. As long as there are at least k descriptors in the training set, matches will be returned. This approach does not reflect the reality of images, where it is very possible for an object and the features representing it to move out of frame. To allow for this possibility, matching descriptors are filtered using their measured distances.

Two popular filtering approaches will be discussed here. An approach that filters using only the distance between the query descriptor and its nearest match is referred to as the 1-nn method. The second approach, the 2-nn method, examines the ratio of distances between the nearest and second nearest training descriptor matches for each query descriptor. The nearest neighbor or 1-nn method is a simple approach useful for first pass work with a new descriptor, while the 2-nn method requires a more complex search, but is better at rejecting bad matches and is more frequently used in practice.

For the purposes of these explanations, filtering of a single match with query descriptor $\mathbf{y}$ and a set of training descriptor matches $\hat{T}$, as defined in Equation (2.19), will be considered. The distance between the i-th training descriptor match $\hat{\mathbf{t}}_i$ and the query descriptor $\mathbf{y}$ will be given as $\Delta(\mathbf{y}, \hat{\mathbf{t}}_i)$. Recall that $\hat{\mathbf{t}}_1$ is the best match for $\mathbf{y}$, $\hat{\mathbf{t}}_2$ is the second best, and so on. The final match result will be given by $\bar{\mathbf{t}}$ where a null value indicates that no suitable match was found.

The 1-nn method examines the nearest descriptor $\hat{\mathbf{t}}_1$ with distance $\Delta(\mathbf{y}, \hat{\mathbf{t}}_1)$ and performs a thresholding operation. If $\Delta(\mathbf{y}, \hat{\mathbf{t}}_1)$ is less than a threshold τ_1, then the match is considered good and $\bar{\mathbf{t}}$ is set to $\hat{\mathbf{t}}_1$. If it is not within the threshold or no nearest neighbor is found at all, then $\mathbf{y}$ is considered unmatched. Equation (2.22) expresses this formally.

$$\bar{\mathbf{t}} = \begin{cases} \hat{\mathbf{t}}_1 & \Delta(\mathbf{y}, \hat{\mathbf{t}}_1) < \tau_1 \\ \emptyset & otherwise \end{cases} \tag{2.22}$$

The 2-nn method requires additional search, but reduces the likelihood of an erroneous match. It examines the two nearest descriptors $\hat{\mathbf{t}}_1$ and $\hat{\mathbf{t}}_2$ with distances $\Delta(\mathbf{y}, \hat{\mathbf{t}}_1)$ and $\Delta(\mathbf{y}, \hat{\mathbf{t}}_2)$ satisfying $\Delta(\mathbf{y}, \hat{\mathbf{t}}_1) \leq \Delta(\mathbf{y}, \hat{\mathbf{t}}_2)$. The method

examines the ratio of distances r between these two matches.

$$r = \frac{\Delta(\mathbf{y}, \hat{\mathbf{t}}_1)}{\Delta(\mathbf{y}, \hat{\mathbf{t}}_2)} \in (0, 1]$$

The value of r provides information indicative of the quality of the match candidate $\hat{\mathbf{t}}_1$. If r is small, then $\hat{\mathbf{t}}_1$ is a much better match than $\hat{\mathbf{t}}_2$ and a clear favorite, whereas as r approaches one, both candidates are of roughly equal quality. This indicates that either the feature in question is not visually distinct in the training set T, or it is not represented in T at all. In either case, a small amount of noise in the descriptor could change the matching result, so the quality of the match is suspect and the feature is left unmatched.

In practice, this filtering is performed by comparison with a user defined threshold τ_2. If r is less than τ_2 then the match is good. The 2-nn filter operation is given in Equation (2.23).

$$\bar{\mathbf{t}} = \begin{cases} \hat{\mathbf{t}}_1 & \Delta(\mathbf{y}, \hat{\mathbf{t}}_1) < \tau_2 \Delta(\mathbf{y}, \hat{\mathbf{t}}_2) \\ \emptyset & otherwise \end{cases} \tag{2.23}$$

The threshold τ_1 is not considered in the 2-nn method. This is done to give robustness to the matcher. It is argued that as long as r is small, then that match is unique and a suitable choice. For a more complete discussion of this 2-nn method refer to the SIFT paper [7] that describes this filtering technique.

2.6.5 AGAST, BRISK, and FREAK

Both BRISK and FREAK make use of a modified version of the AGAST detector that searches space and scale for "corner-like" features and supplies their locations (u, v, σ) to sub-pixel precision. The interest points are described with binary descriptors created by comparing the intensity of image patches surrounding the feature. The patch sampling pattern, descriptor order, and matching scheme differ between BRISK and FREAK, although both use a Hamming distance metric. Both methods claim excellent rotational and scale invariance.

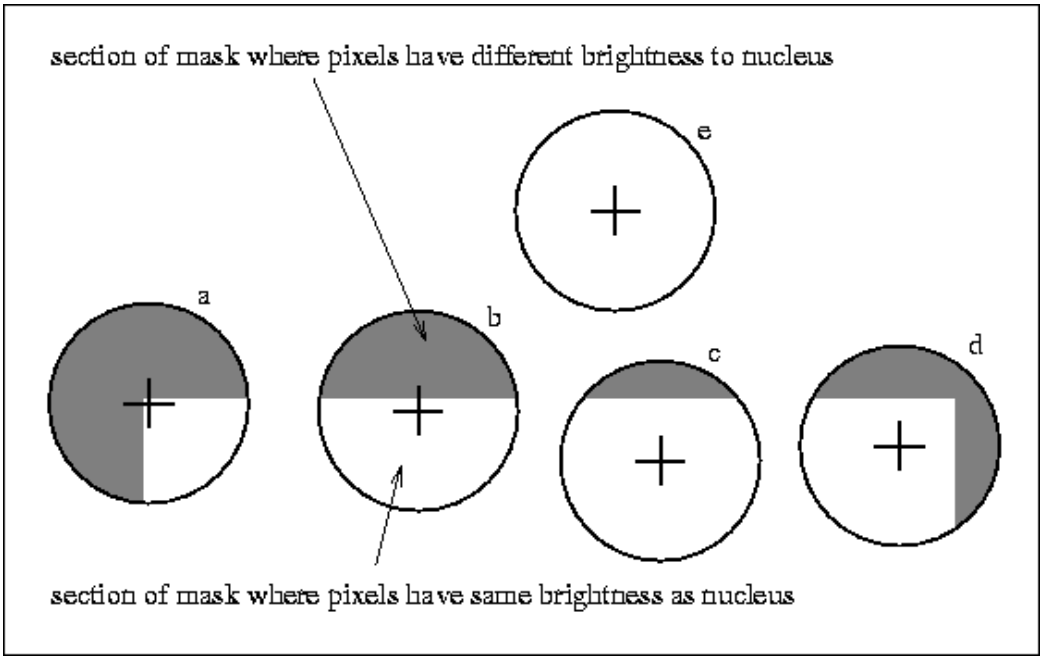


Figure 2.1: USAN kernel applied to various points. Figure from [31].

2.6.5.1 BRISK-AGAST Detector

The BRISK and FREAK systems make use of the BRISK-AGAST feature detector, a custom version of the Adaptive and Generalized Accelerated Segment Test (AGAST) specialized for use with the BRISK descriptor. AGAST [11] and its predecessor FAST [29] [30] were inspired by SUSAN [31] and its Univalue Segment Assimilating Nucleus (USAN) type edge detector. In essence, the SUSAN detector identifies points which are dissimilar from their surrounding. The intensity of a candidate interest point is compared to the intensities of pixels located inside of a circular kernel about that point. If a sufficient number of pixels are all much darker or lighter than the point under consideration, then that point is chosen as a SUSAN interest point.

Figure 2.1 is illustrative of the principle. Points like e and d, where the majority of pixels in the kernel were of similar intensity as the center point, would not be taken as interest points, whereas points like a and b would.

AGAST and FAST operate on the Accelerate Segment Test (AST) principle, an approach similar in concept to USAN. In the AST test only the pixels lying on the edge of the kernel are considered. If there is a sufficient number of contiguous pixels which are all darker or all brighter than the center point by at least a threshold t, then that center point is taken as an interest point.

Considered in this manner, points a and d in Figure 2.1 would be corner features, whereas points such as b and c would not. By examining certain pixels on the edge of the circle first, candidate feature points can be rapidly disqualified with a small number of comparisons.

The BRISK-AGAST implementation searches across space and scale for maxmimal AST features according to their FAST score s, given in Equation (2.24). Here, $L(\mathbf{x})$ denotes the intensity of the pixel $\mathbf{x}$. The sets S_{bright} and S_{dark} are the sets of pixels taken from the Bresenham circle B about $\mathbf{c}$ which are all brighter or darker than $\mathbf{c}$ by at least t. The Bresenham circle B is a discrete approximation of a circle at a fixed radius from $\mathbf{c}$. The pixels $\mathbf{b} \in B$ lie along the circumference of this circle. The set S_{unkown} is the set of Bresenham circle pixels that are not found in S_{bright} or S_{dark}. All three sets are detailed in Equation (2.25).

$$s(\mathbf{c}) = \max \left(\sum_{\mathbf{b} \in S_{bright}} |L(\mathbf{b}) - L(\mathbf{c})| - t, \sum_{b \in S_{dark}} |L(\mathbf{c}) - L(\mathbf{b})| - t \right) \quad (2.24)$$

$$\begin{aligned} S_{bright} &= \{\mathbf{b} \,|\mathbf{b} \in B, L(\mathbf{b}) - L(\mathbf{c}) > t\} \\ S_{dark} &= \{\mathbf{b} \,|\mathbf{b} \in B, \mathbf{b} - \mathbf{c} < -t\} \\ S_{unkown} &= B - (S_{bright} + S_{dark}) \end{aligned} \quad (2.25)$$

A non-maxima suppression (NMS) algorithm is used to find these points. The scores are computed using the FAST 9-16 metric, with minimum segment length of nine and a Bresenham circle of radius three, giving a circumference of 16 pixels. The scale space is composed of o octave levels c_i and intra-octave levels d_i $i = 0 \ldots o-1$. An octave or intra-octave level is a version of the original image scaled and downsampled according to the octave's scale. The first octave level c_0 is the original image and subsequent octave levels are found by half sampling the preceding octave level. The first intra-octave level d_0 is found by downsampling c_0 by 1.5 and subsequent intra-octave levels are half samples of the previous intra-octave level.

The location of the feature point (x, y, σ) is refined to sub-pixel coordinates by finding the refinement terms Δx, Δy and $\Delta\sigma$, so that the final location of the interest point is given by $(\bar{x}, \bar{y}, \bar{\sigma}) = (x + \Delta x, y + \Delta y, \sigma\Delta\sigma)$. The

sub-pixel coordinates are found by estimating the location of the maximum FAST-9-16 score near the interest point using its neighboring 26 pixels.

This is accomplished in three steps. First, a maximum FAST-9-16 score and its location are estimated at each of the three octave levels surrounding σ. A 2-D quadratic is fit to the nine nearest pixels on each of these octave levels and its critical point, the maximum local FAST score at that octave level, is found. The locations of the maximum scores are given by $\Delta\mathbf{x}'_{-1}$, $\Delta\mathbf{x}'_0$, and $\Delta\mathbf{x}'_1$, where the subscript indicates the position of the octave level relative to σ. The FAST score maxima are given by s'_{-1}, s'_0, and s'_1.

Next, the maximum score across all 27 pixels is found by fitting a 1-D parabola to the three maxima and their locations found in the first step. The peak of this parabola is the maximum score and the location of the peak is the scale refinement term $\Delta\sigma$.

Last, the sub-pixel refinements Δx and Δy are found by estimating the location in space of the maximum score found in the second step. This is accomplished using 2-D interpolation to interpolate between the location of the two maxima found in step 1 corresponding to the two octave levels surrounding $\Delta\sigma$. The interpolation weight is found using the scale refinement term $\Delta\sigma$.

2.6.5.2 BRISK Descriptor

The BRISK descriptor [9] is a binary descriptor constructed by comparing image intensities sampled at locations surrounding the interest point. The descriptor is both rotation-normalized and scale-normalized, meaning that changing the orientation and size of an image should not change the descriptors extracted from it. The implementation of the BRISK descriptor used in OpenCV [17] is built using a 60 point sampling pattern shown in Figure 2.2.

The first step in the descriptor construction process is computing an orientation α for the interest point. This orientation describes the direction that the interest point is pointing relative to the image's $\mathbf{u}$ axis. Orientation plays a vital role in providing rotational invariance to the interest point. By constructing the descriptor relative to the orientation, similar descriptors can be formed for the same interest point found in multiple images, even if there is a large amount of rotation about the imaging axis between these images.

The orientation of the interest point is found using the mean gradient

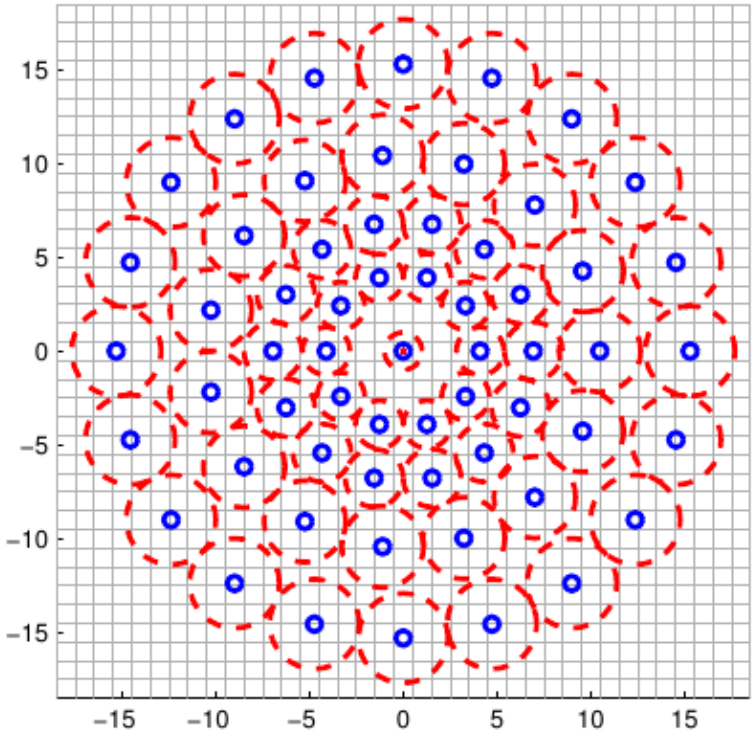


Figure 2.2: The BRISK sampling pattern from [9]. The blue circle denotes the location of a sampling point and the red dashed circle indicates the size of the Gaussian kernel used to measure the sample's intensity.

$\bar{\mathbf{g}}$, which will be explained in a following paragraph. The sampling pattern shown in Figure (2.2) is scaled by the interest point's scale $\bar{\sigma}$ and the gradient for each long point pair $(\mathbf{p}_i, \mathbf{p}_j)$ is found. These pairs are points in the sampling pattern with a distance greater than δ_{min}. The set of all such points are collectively referred to as the long point pair set $\mathcal{L}$. The values δ_{min} and δ_{max} can be specified by the user, but are typically left at the default of 5.85σ and 8.2σ, respectively.

$$\mathcal{L} = \{(\mathbf{p}_i, \mathbf{p}_j) \mid ||\mathbf{p}_j - \mathbf{p}_i|| > \delta_{min} = 5.85\bar{\sigma}\}$$

The gradient for each of these long point pairs is computed using Equation (2.26). In this equation, the term $\Upsilon(\mathbf{p}, \beta)$ is the smoothed intensity obtained by applying a Gaussian kernel of size 2β to octave level c_0 at sampling point $\mathbf{p} = \begin{bmatrix} p_x & p_y \end{bmatrix}^T$, given in Equation (2.27).

$$\mathbf{g}(\mathbf{p}_i, \mathbf{p}_j) = \big(\Upsilon(\mathbf{p}_j, \beta_j) - \Upsilon(\mathbf{p}_i, \beta_i)\big) \frac{(\mathbf{p}_i - \mathbf{p}_j)}{||\mathbf{p}_j - \mathbf{p}_i||^2} \tag{2.26}$$

$$\Upsilon(\mathbf{p}, \beta) = \sum_{u=p_x-\beta}^{p_x+\beta} \sum_{v=p_y-\beta}^{p_y+\beta} \frac{1}{2\pi\beta^2} e^{-\frac{(u-p_x)^2+(v-p_y)^2}{2\beta^2}} c_o(u, v) \tag{2.27}$$

The mean gradient $\bar{\mathbf{g}}$, the mean of these long point-pair gradients, is then found according to Equation (2.28), where $|\mathcal{L}|$ is the number of long point pairs in $\mathcal{L}$.

$$\bar{\mathbf{g}} = \begin{pmatrix} \bar{g}_x \\ \bar{g}_y \end{pmatrix} = \frac{1}{|\mathcal{L}|} \sum_{(\mathbf{p}_i, \mathbf{p}_j) \in \mathcal{L}} \mathbf{g}(\mathbf{p}_i, \mathbf{p}_j) \tag{2.28}$$

The orientation α of the interest point is found in Equation (2.29) by computing atan2 of the x and y mean gradients $\bar{g}_x$ and $\bar{g}_y$.

$$\alpha = \text{atan2}(\bar{g}_y, \bar{g}_x) \tag{2.29}$$

$$\text{atan2}(x,y) = \begin{cases} \arctan\left(\frac{y}{x}\right) & x > 0 \\ \arctan\left(\frac{y}{x}\right) + \pi & y \geq 0, x < 0 \\ \arctan\left(\frac{y}{x}\right) - \pi & y < 0, x < 0 \\ +\frac{\pi}{2} & y > 0, x = 0 \\ -\frac{\pi}{2} & y < 0, x = 0 \\ undefined & y = 0, x = 0 \end{cases}$$

Once the orientation is found, the orientation α and scale $\bar{\sigma}$ are used to rotate and scale the sampling pattern. The new sampling points in this pattern such as $\mathbf{p}^\alpha$ are indicated with a superscript α. The new Gaussian kernel sizes of the form $\beta^{\bar{\sigma}}$ are indicated by the superscript $\bar{\sigma}$.

A 512 bit descriptor is constructed by comparing all the short-distance pairings $(\mathbf{p}_i^\alpha, \mathbf{p}_j^\alpha)$, where $||\mathbf{p}_j^\alpha - \mathbf{p}_i^\alpha|| < \delta_{max} = 8.2\bar{\sigma}$. An example mapping of a bit b_ℓ in the descriptor is given in Equation (2.30). The term $\mathbf{p}^\alpha$ is given in relation to $\mathbf{p} = \begin{bmatrix} p_x & p_y \end{bmatrix}^T$ in Equation (2.31).

$$b_\ell = \begin{cases} 1, & \Upsilon(\mathbf{p}_j^\alpha, \beta_j^{\bar{\sigma}}) > \Upsilon(\mathbf{p}_i^\alpha, \beta_i^{\bar{\sigma}}) \\ 0, & \text{otherwise} \end{cases} \tag{2.30}$$

$$\mathbf{p}^\alpha = \begin{pmatrix} \sigma \cos(\alpha) p_x \\ \sigma \sin(\alpha) p_y \end{pmatrix} \tag{2.31}$$

The dissimilarity between two BRISK descriptors is measured using the Hamming distance, the number of bits where the two descriptors do not agree.

2.6.5.3 FREAK Descriptor

FREAK descriptor construction is similar in approach to BRISK. A normalized sampling pattern is used to perform Gaussian kernel intensity comparisons and construct a binary descriptor. FREAK differentiates itself in the format of its sampling pattern and the selection of point pairs for orientation computation and descriptor construction. The FREAK sampling pattern, shown in Figure 2.3, differs in format from BRISK. FREAK contains fewer sampling points, but has overlapping kernels and a much wider range of kernel sizes.

FREAK perform rotation and scale normalization in a similar manner to

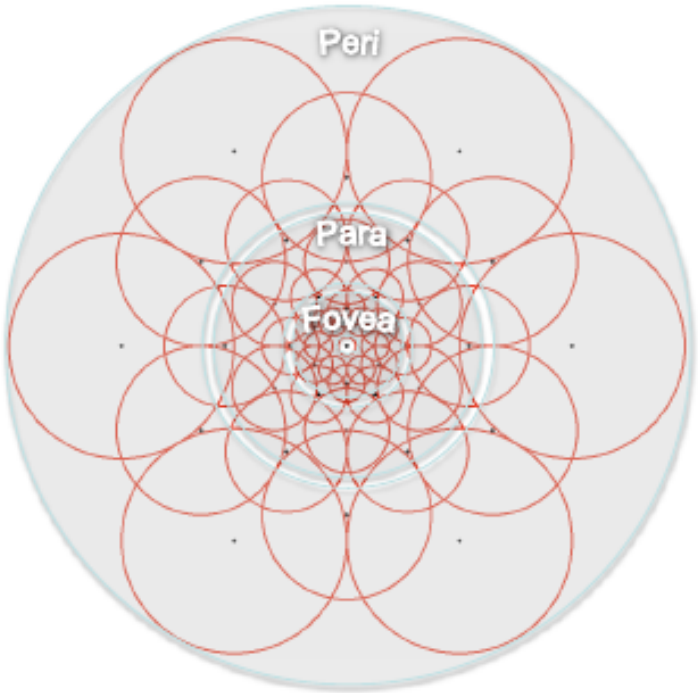


Figure 2.3: The FREAK sampling pattern from [10]. A dot denotes the location of a sampling point and a circle indicates the radius of the Gaussian kernel used to measure the sample's intensity. FREAK contains fewer sampling points than BRISK, but has overlapping fields and a much wider range of field sizes.

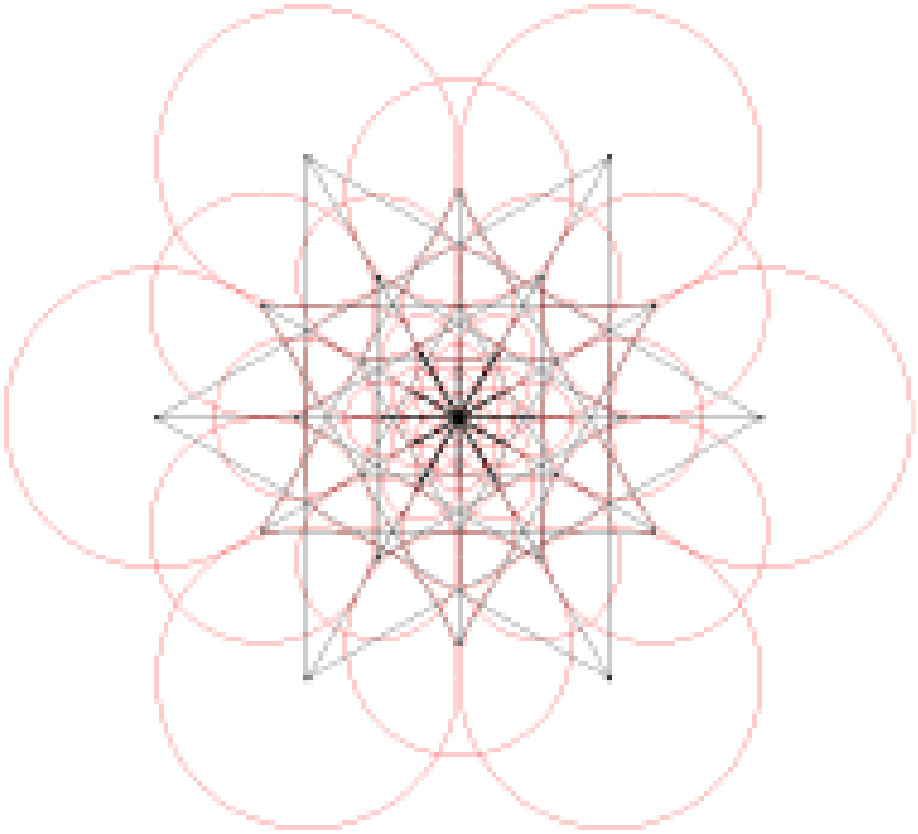


Figure 2.4: The FREAK point pairs used to compute the global orientation of the feature point. Figure from [10].

BRISK. Orientation is still found using the average gradient between point pairs, as in Equations (2.28) and (2.29), but FREAK uses fewer point pairs and selects those point pairs based on kernel size, rather than distance. The point pairs used are shown in Figure 2.4. Each circle in this figure indicates a Gaussian kernel used for comparison. At the center of each circle is its sampling point. Lines drawn between these sampling points indicate a point pairing. Notice that pairings are only made between points whose kernels are the same size.

The point pairs used for FREAK descriptor construction are indicated by the lines in Figure 2.4. The descriptor is organized in a coarse to fine manner where comparisons between points with the largest kernels are located near the front of the descriptor and the smallest kernel comparisons are located

near the end.

2.6.5.4 FREAK Descriptor Comparison

FREAK descriptor comparison is done as a chain of four comparisons. The descriptors are compared 128 bits at a time, proceeding to the next 128 only if the Hamming distance between the current 128 bit blocks is sufficiently small. This process continues until a match is ruled out or all 512 bits have been examined. In experiments, it was found that over 90% of candidates could be discarded with the first 128 bit comparison [10].

2.7 Quaternions

Quaternions are a complex number system first proposed by Hamilton [32]. A quaternion consists of four terms – a real component and three imaginary components $\boldsymbol{i}$, $\boldsymbol{j}$, and $\boldsymbol{k}$. The real component is sometimes called the scalar component and the imaginary components are collectively referred to as the vector component. Quaternions are well known for their value in working with 3-D rotations. They are frequently used to represent the orientation of an object, express rotations within a reference frame, and describe the transformation between two frames.

It is a well known fact that any change in orientation in a 3-D coordinate system can be represented as a single rotation of less than 180 degrees about some vector. Quaternions can be used to represent this vector, angle pair. The quaternion version Q of a rotation of γ about a unit vector $\begin{bmatrix} x_v & y_v & z_v \end{bmatrix}^T$ is given in Equation (2.32).

$$\mathrm{Q} = \begin{bmatrix} \cos(\frac{\gamma}{2}) \\ x_v \sin(\frac{\gamma}{2}) \\ y_v \sin(\frac{\gamma}{2}) \\ z_v \sin(\frac{\gamma}{2}) \end{bmatrix} \tag{2.32}$$

2.7.1 Quaternion Math and Properties

2.7.1.1 Notation

The vector form of a quaternion will be used by default here. A quaternion will be indicated in normal uppercase.

$$\mathrm{Q} = \begin{bmatrix} w \\ \mathbf{x} \end{bmatrix} = \begin{bmatrix} w \\ x \\ y \\ z \end{bmatrix}$$

$$\mathbf{x} \in \mathbb{R}^3; w, x, y, z \in \mathbb{R}$$

A quaternion may also be represented using the complex form:

$$\mathrm{Q} = w + \boldsymbol{i}x + \boldsymbol{j}y + \boldsymbol{k}z$$

With multiplicative identities:

$$\begin{aligned} \boldsymbol{i}\,\boldsymbol{j} &= \boldsymbol{k} \qquad & \boldsymbol{i}^{\,2} = \boldsymbol{j}^{\,2} = \boldsymbol{k}^{\,2} = -1 \\ \boldsymbol{j}\,\boldsymbol{k} &= \boldsymbol{i} \\ \boldsymbol{k}\,\boldsymbol{i} &= \boldsymbol{j} \end{aligned}$$

2.7.1.2 Scaling

The product of a quaternion Q and a scalar α is another quaternion.

$$\alpha\mathrm{Q} = \mathrm{Q}\alpha = \begin{bmatrix} \alpha w \\ \alpha\mathbf{x} \end{bmatrix} = \begin{bmatrix} \alpha w \\ \alpha x \\ \alpha y \\ \alpha z \end{bmatrix}$$

2.7.1.3 Addition

Two quaternions Q_1 and Q_2 can be added.

$$\mathrm{Q}_1 + \mathrm{Q}_2 = \begin{bmatrix} w_1 \\ \mathbf{x}_1 \end{bmatrix} + \begin{bmatrix} w_2 \\ \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} w_1 + w_2 \\ \mathbf{x}_1 + \mathbf{x}_2 \end{bmatrix}$$

2.7.1.4 Multiplication

The product of two quaternions Q_1, Q_2 is given by:

$$Q_1 Q_2 = \begin{bmatrix} w_1 \\ \mathbf{x}_1 \end{bmatrix} \begin{bmatrix} w_2 \\ \mathbf{x}_2 \end{bmatrix} = \begin{bmatrix} w_1 w_2 - \mathbf{x}_1 \cdot \mathbf{x}_2 \\ w_1 \mathbf{x}_2 + w_2 \mathbf{x}_1 + \mathbf{x}_1 \times \mathbf{x}_2 \end{bmatrix}$$

In this section, the values of quaternion will frequently be made explicit using a vector style notation. The product of quaternions in this format is indicated in the style $\begin{bmatrix} w_1 \\ \mathbf{x}_1 \end{bmatrix} \begin{bmatrix} w_2 \\ \mathbf{x}_2 \end{bmatrix}$. This product is not a matrix product. It is possible to discern between quaternion and matrix products by examining the number of elements in the brackets. Quaternions are always of size 4×1. Multiplying two 4×1 vectors is an illegal operation, so the intended quaternion multiplication should be apparent.

2.7.1.5 Identity

The additive identity for quaternions $0 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ satisfies the property:

$$Q + 0 = Q$$

The multiplicative identity for quaternions $1 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}$ satisfies the property:

$$1Q = Q1 = Q$$

2.7.1.6 Associativity

Both addition and multiplication operations are associative for quaternions.

$$(Q_1 + Q_2) + Q_3 = Q_1 + (Q_2 + Q_3)$$

$$(Q_1 Q_2) Q_3 = Q_1 (Q_2 Q_3)$$

2.7.1.7 Commutativity

Addition of quaternions is commutative.

$$\mathrm{Q}_1 + \mathrm{Q}_2 = \mathrm{Q}_2 + \mathrm{Q}_1$$

Multiplication of quaternions is not commutative in most cases, generally, $\mathrm{Q}_2\mathrm{Q}_1 \neq \mathrm{Q}_1\mathrm{Q}_2$.

2.7.1.8 Distributivity

Multiplication of quaternions is distributive.

$$\mathrm{Q}_1(\mathrm{Q}_2 + \mathrm{Q}_3) = \mathrm{Q}_1\mathrm{Q}_2 + \mathrm{Q}_1\mathrm{Q}_3$$

$$(\mathrm{Q}_1 + \mathrm{Q}_2)\mathrm{Q}_3 = \mathrm{Q}_1\mathrm{Q}_3 + \mathrm{Q}_2\mathrm{Q}_3$$

2.7.1.9 Conjugation

The conjugate Q^* of a quaternion is obtained by scaling its imaginary components by -1.

$$\mathrm{Q}^* = \begin{bmatrix} w \\ -\mathbf{x} \end{bmatrix} = \begin{bmatrix} w \\ -x \\ -y \\ -z \end{bmatrix}$$

The conjugate of a product of quaternions is the product of the conjugate of each quaternion in reverse order.

$$(\mathrm{Q}_1\mathrm{Q}_2\mathrm{Q}_3)^* = \mathrm{Q}_3^*\mathrm{Q}_2^*\mathrm{Q}_1^*$$

Conjugates can be used to extract the real and imaginary components of a quaternion.

$$w = \frac{1}{2}(\mathrm{Q} + \mathrm{Q}^*)$$

$$\mathbf{x} = \frac{1}{2}(\mathrm{Q} - \mathrm{Q}^*)$$

2.7.1.10 Magnitude and Norm

The magnitude or norm of a quaternion Q is $||\mathrm{Q}||$. It can be found by taking the square root of the product of quaternion with its conjugate. The resulting quaternion has only one non-zero term, the real component, which gives the magnitude of Q.

$$\mathrm{QQ}^* = \mathrm{Q}^*\mathrm{Q} = \begin{bmatrix} w^2 + ||\mathbf{x}||^2 \\ 0 \end{bmatrix}$$

$$||\mathrm{Q}||^2 = ||\mathrm{QQ}^*|| = w^2 + x^2 + y^2 + z^2$$

2.7.1.11 Normalization

A quaternion Q with magnitude $||\mathrm{Q}|| = 1$ is a unit quaternion. A quaternion Q_1 can be normalized to produce a unit quaternion Q by dividing by its magnitude.

$$\mathrm{Q} = \frac{\mathrm{Q}_1}{||\mathrm{Q}_1||}$$

2.7.1.12 Inversion

The inverse of a quaternion is its conjugate divided by its squared norm.

$$\mathrm{Q}^{-1} = \frac{\mathrm{Q}^*}{||\mathrm{Q}||^2}$$

2.7.1.13 Division

The quotient of two quaternions is the product of the dividend and the inverse of the divisor.

$$\mathrm{Q}_1/\mathrm{Q}_2 = \mathrm{Q}_1\mathrm{Q}_2^{-1}$$

2.7.1.14 Exponential

The exponential of a quaternion $\exp(\mathrm{Q})$ is:

$$\exp(\mathrm{Q}) = \exp(w) \begin{bmatrix} \cos(||\mathbf{x}||) \\ \frac{\mathbf{x}}{||\mathbf{x}||} \sin(||\mathbf{x}||) \end{bmatrix}$$

2.7.1.15 Logarithm

The logarithm of a quaternion $\ln(\mathrm{Q})$ is:

$$\ln(\mathrm{Q}) = \begin{bmatrix} \ln(||\mathrm{Q}||) \\ \frac{\mathbf{x}}{||\mathbf{x}||} \arccos(\frac{w}{||\mathrm{Q}||}) \end{bmatrix}$$

2.7.1.16 Power

A quaternion power Q^u is found using exponential and logarithm operations.

$$\mathrm{Q}^u = \exp(\ln(\mathrm{Q})u)$$

The variable u can be a quaternion itself or a scalar.

2.7.2 Rotation with Quaternions

2.7.2.1 Basics

Unit quaternions can be used to represent a rotation of γ about a vector $\mathbf{v}$. The quaternion version Q of such a rotation is:

$$\mathrm{Q} = \begin{bmatrix} \cos\left(\frac{\gamma}{2}\right) \\ \frac{\mathbf{v}}{||\mathbf{v}||} \sin\left(\frac{\gamma}{2}\right) \end{bmatrix}$$

A point (p_x, p_y, p_z) can be represented in quaternion form Q_p as a vector with no rotation. Quaternions representing points do not necessarily have

unit norms.

$$\mathrm{Q}_p = \begin{bmatrix} 0 \\ x_p \\ y_p \\ z_p \end{bmatrix}$$

Such a point Q_p can be rotated by a unit quaternion Q to Q'_p by pre-multiplying the point by Q and post-multiplying it by Q^{-1}.

$$\mathrm{Q}'_p = \mathrm{Q}\mathrm{Q}_p\mathrm{Q}^{-1}$$

Sequential rotations by multiple unit quaternions can be combined into a single rotation by a single quaternion.

$$\begin{aligned} \mathrm{Q}'_p &= \mathrm{Q}_1(\mathrm{Q}_2(\mathrm{Q}_3\ \mathrm{Q}_p\ \mathrm{Q}_3^{-1})\mathrm{Q}_2^{-1})\mathrm{Q}_1^{-1} \\ &= (\mathrm{Q}_1\mathrm{Q}_2\mathrm{Q}_3)\mathrm{Q}_p(\mathrm{Q}_3^{-1}\mathrm{Q}_2^{-1}\mathrm{Q}_1^{-1}) \\ &= (\mathrm{Q}_1\mathrm{Q}_2\mathrm{Q}_3)\mathrm{Q}_p(\mathrm{Q}_1\mathrm{Q}_2\mathrm{Q}_3)^{-1} \\ &= \mathrm{Q}\mathrm{Q}_p\mathrm{Q}^{-1} \end{aligned} \tag{2.33}$$

Where Q is equal to $\mathrm{Q}_1\mathrm{Q}_2\mathrm{Q}_3$.

2.7.2.2 Example

Consider a point $p = (1, 1, 0)$ with quaternion representation Q_p shown as a red x in Figure 2.5.

$$\mathrm{Q}_p = \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}$$

A rotation of $\pi/2$ radians about the vector $\begin{bmatrix}0 & 0 & 1\end{bmatrix}^T$ is given in quaternion form as Q_1.

$$\mathrm{Q}_1 = \begin{bmatrix} \cos(\frac{\gamma}{2}) \\ \begin{bmatrix}0\\0\\1\end{bmatrix} \sin(\frac{\gamma}{2}) \end{bmatrix} = \begin{bmatrix}\sqrt{2}/2\\0\\0\\\sqrt{2}/2\end{bmatrix}$$

Rotating p by Q_1 gives the point $q = (-1, 1, 0)$ with quaternion Q_q.

$$\mathrm{Q}_q = \mathrm{Q}_1\mathrm{Q}_p\mathrm{Q}_1^{-1} = \begin{bmatrix}\sqrt{2}/2\\0\\0\\\sqrt{2}/2\end{bmatrix} \begin{bmatrix}0\\1\\1\\0\end{bmatrix} \begin{bmatrix}\sqrt{2}/2\\0\\0\\-\sqrt{2}/2\end{bmatrix} = \begin{bmatrix}0\\-1\\1\\0\end{bmatrix}$$

Applying a second quaternion rotation Q_2 of $\pi/2$ radians about the vector $\begin{bmatrix}0 & 1 & 0\end{bmatrix}^T$ to q yields point $r = (0, 1, 1)$ with quaternion Q_r.

$$\mathrm{Q}_r = \mathrm{Q}_2\mathrm{Q}_q\mathrm{Q}_2^{-1} = \begin{bmatrix}\sqrt{2}/2\\0\\\sqrt{2}/2\\0\end{bmatrix} \begin{bmatrix}0\\-1\\1\\0\end{bmatrix} \begin{bmatrix}\sqrt{2}/2\\0\\-\sqrt{2}/2\\0\end{bmatrix} = \begin{bmatrix}0\\0\\1\\1\end{bmatrix}$$

An equivalent rotation from p to r is given by the quaternion $\mathrm{Q}_3 = \mathrm{Q}_2\mathrm{Q}_1$. The axis angle rotation described by this quaternion is indicated by the angle γ_3 and axis $\mathbf{x}_3$.

$$\mathrm{Q}_3 = \begin{bmatrix}\sqrt{2}/2\\0\\\sqrt{2}/2\\0\end{bmatrix} \begin{bmatrix}\sqrt{2}/2\\0\\0\\\sqrt{2}/2\end{bmatrix} = \begin{bmatrix}0.5\\0.5\\0.5\\0.5\end{bmatrix}$$

$$\gamma_3 = 2\arccos(0.5) = \tfrac{2}{3}\pi \quad , \quad \mathbf{x}_3 = \tfrac{1}{\sin(\pi/3)} \begin{bmatrix}0.5\\0.5\\0.5\end{bmatrix} = \begin{bmatrix}0.5774\\0.5774\\0.5774\end{bmatrix}$$

$$\mathrm{Q}_r = \mathrm{Q}_3\mathrm{Q}_p\mathrm{Q}_3^{-1} = \begin{bmatrix}0.5\\0.5\\0.5\\0.5\end{bmatrix} \begin{bmatrix}0\\1\\1\\0\end{bmatrix} \begin{bmatrix}0.5\\-0.5\\-0.5\\-0.5\end{bmatrix} = \begin{bmatrix}0\\0\\1\\1\end{bmatrix}$$

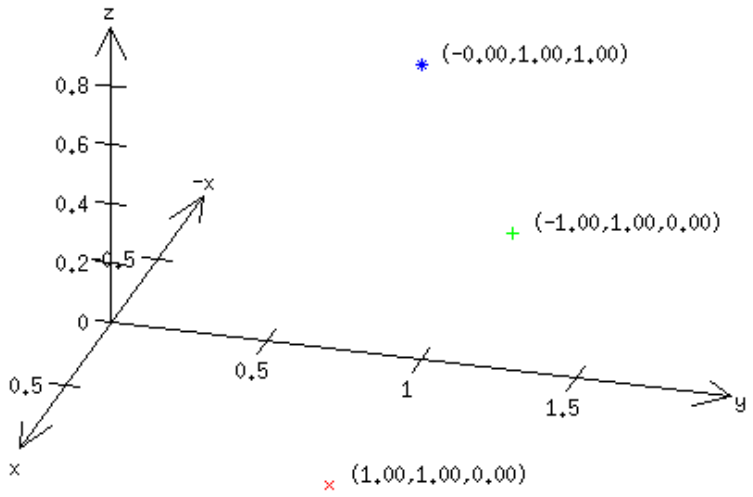


Figure 2.5: Rotation of points using quaternions. p is the red x, q the green +, and r is the blue *.

2.7.2.3 Multiple Representations

Each transformation or rotation can be represented by two different quaternions Q and Q′.

$$\mathrm{Q}' = -\mathrm{Q} = \begin{bmatrix} -w \\ -x \\ -y \\ -z \end{bmatrix}$$

$$(\mathrm{Q}')^{-1} = -\mathrm{Q}^{-1} = \begin{bmatrix} -w \\ x \\ y \\ z \end{bmatrix}$$

$$(\mathrm{Q}')\mathrm{Q}_p(\mathrm{Q}')^{-1} = (-\mathrm{Q})\mathrm{Q}_p(-\mathrm{Q}^{-1}) = \mathrm{Q}\mathrm{Q}_p\mathrm{Q}^{-1}$$

This is supported by the observation that a clockwise rotation about a vector is equivalent to a counter-clockwise rotation about a vector pointing in the opposite direction.

Borrowing from the earlier example, observe that applying the operation $(\mathrm{Q}_1')\mathrm{Q}_p(\mathrm{Q}_1')^{-1}$ still produces $q = (-1, 1, 0)$.

$$\mathrm{Q}_q = (\mathrm{Q}_1')\mathrm{Q}_p(\mathrm{Q}_1')^{-1} = \begin{bmatrix} -\sqrt{2}/2 \\ 0 \\ 0 \\ -\sqrt{2}/2 \end{bmatrix} \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix} \begin{bmatrix} -\sqrt{2}/2 \\ 0 \\ 0 \\ \sqrt{2}/2 \end{bmatrix} = \begin{bmatrix} 0 \\ -1 \\ 1 \\ 0 \end{bmatrix}$$

2.7.2.4 Measuring the Difference between Two Quaternions with a Distance Metric

For two unit quaternions Q_1 and Q_2, the unit quaternion Q_d that transforms Q_2 into Q_1 such that $\mathrm{Q}_1 = \mathrm{Q}_d\mathrm{Q}_2$ can be obtained by the equation $\mathrm{Q}_d = \mathrm{Q}_1\mathrm{Q}_2^{-1}$.

This Q_d with scalar component w_d can be used to obtain an angular distance metric γ between the two quaternions. The distance metric follows from the quaternion definition in terms of angle axis notation presented earlier in this section.

$$w_d = \cos\left(\frac{\gamma}{2}\right)$$

$$\gamma = 2\arccos(w_d)$$

Since Q_d is a unit quaternion, the domain of w_d is $[-1, 1]$, yielding a range for γ of $[0, 2\pi]$, assuming an arccos function with a range of $[0, \pi]$. However, Euler's rotation theorem states that the magnitude of the largest rotation should be π radians. A closer examination reveals that negative values of w_d produce angles larger than the predicted π radians. Recalling that $\mathrm{Q}' = -\mathrm{Q}$ produces a rotation equivalent to Q, the range of γ can be limited to $[0, \pi]$ by altering the equation for γ to:

$$\gamma = 2\arccos(max(w_d, w_d')) \quad \gamma \in [0, \pi]$$

or equivalently:

$$\gamma = 2\arccos(|w_d|) \quad \gamma \in [0, \pi]$$

2.7.2.5 Transformation

Rigid transformations between any two right-handed orthogonal coordinate systems A and B that share an origin can be accomplished using quaternions.

The rotational transformation from B to A, ${}^{A}_{B}\mathrm{Q}$ can be described as the clockwise rotation of angle γ about the unit vector $\mathbf{v}$ that would align the basis vectors of A with those of B.

$$ {}^{A}_{B}\mathrm{Q} = \begin{bmatrix} \cos(\frac{\gamma}{2}) \\ \mathbf{v}\sin(\frac{\gamma}{2}) \end{bmatrix} $$

A quaternion representation of a point expressed in the B coordinate frame ${}^{B}\mathrm{Q}_{q}$ can be expressed in terms of A by applying this transformation, detailed in Equation (2.34).

$$ {}^{A}\mathrm{Q}_{q} = {}^{A}_{B}\mathrm{Q}\ {}^{B}\mathrm{Q}_{q}\ {}^{A}_{B}\mathrm{Q}^{-1} \tag{2.34} $$

Equation (2.34) can be extended to express any rigid transformation between A and B by incorporating a translation quaternion term ${}^{A}_{B}\mathrm{Q}_{\rho}$. Equation (2.35) expresses such a transformation from ${}^{B}\mathrm{Q}_{q}$ to ${}^{A}\mathrm{Q}_{q}$.

$$ {}^{A}\mathrm{Q}_{q} = {}^{A}_{B}\mathrm{Q}\ {}^{B}\mathrm{Q}_{q}\ {}^{A}_{B}\mathrm{Q}^{-1} + {}^{A}_{B}\mathrm{Q}_{\rho} \tag{2.35} $$

This equation can be further refined to allow scaling by α, yielding the Vector Quaternion Scaling (VQS) transformation [33] in Equation (2.36).

$$ {}^{A}\mathrm{Q}_{q} = \alpha\left({}^{A}_{B}\mathrm{Q}\ {}^{B}\mathrm{Q}_{q}\ {}^{A}_{B}\mathrm{Q}^{-1}\right) + {}^{A}_{B}\mathrm{Q}_{\rho} \tag{2.36} $$

2.7.3 Conversion to Other Representations

It is often necessary to convert between quaternions and other more frequently used rotational representations such as Euler angles or rotation matrices.

2.7.3.1 Axis Angle

The most straightforward conversion of a quaternion Q is to the axis-angle format, where the 3-D rotation is given in the form of a vector $\mathbf{v}$ and an

angle γ.

The conversion from a unit vector and angle to quaternion was given at the beginning of the section in Equation (2.32). The more general conversion for a vector of any magnitude is given in Equation (2.37).

$$\mathrm{Q} = \begin{bmatrix} w \\ \mathbf{x} \end{bmatrix} = \begin{bmatrix} \cos\left(\frac{\gamma}{2}\right) \\ \frac{\mathbf{v}}{||\mathbf{v}||}\sin\left(\frac{\gamma}{2}\right) \end{bmatrix} \tag{2.37}$$

Axis angle format is recovered in two parts. First, the angle γ is found by doubling the arc-cosine of the real component of the normalized quaternion. If the angle is not zero, the vector is found by dividing the first three elements of the normalized quaternion by the sine of half the angle.

$$\begin{aligned} \gamma &= 2\arccos\left(\frac{w}{||\mathrm{Q}||}\right) \\ \mathbf{v} &= \begin{cases} \dfrac{\mathbf{x}}{\sin\left(\gamma/2\right)||\mathrm{Q}||} & \gamma \neq 0 \\ 0 & \gamma = 0 \end{cases} \end{aligned}$$

2.7.3.2 Rotation Matrix

When converting from a rotation matrix $\mathbf{R} \in \mathrm{SO}(3)$ to a quaternion Q, one of four formulas must be used, depending on the values of $\mathbf{R}$. These formulas are given in [34] and restated in Equation (2.38).

$$\mathrm{Q} = \begin{cases} \mathrm{Q}^0(\mathbf{R}) & if \quad \mathbf{R}_{(2,2)} > -\mathbf{R}_{(3,3)}, & \mathbf{R}_{(1,1)} > -\mathbf{R}_{(2,2)}, & \mathbf{R}_{(1,1)} > -\mathbf{R}_{(3,3)} \\ \mathrm{Q}^1(\mathbf{R}) & if \quad \mathbf{R}_{(2,2)} < -\mathbf{R}_{(3,3)}, & \mathbf{R}_{(1,1)} > \mathbf{R}_{(2,2)}, & \mathbf{R}_{(1,1)} > \mathbf{R}_{(3,3)} \\ \mathrm{Q}^2(\mathbf{R}) & if \quad \mathbf{R}_{(2,2)} > \mathbf{R}_{(3,3)}, & \mathbf{R}_{(1,1)} < \mathbf{R}_{(2,2)}, & \mathbf{R}_{(1,1)} < -\mathbf{R}_{(3,3)} \\ \mathrm{Q}^3(\mathbf{R}) & if \quad \mathbf{R}_{(2,2)} < \mathbf{R}_{(3,3)}, & \mathbf{R}_{(1,1)} < -\mathbf{R}_{(2,2)}, & \mathbf{R}_{(1,1)} < \mathbf{R}_{(3,3)} \end{cases} \tag{2.38}$$

$$\mathrm{Q}^0(\mathbf{R}) = \frac{1}{2}\begin{bmatrix} \sqrt{1 + \mathbf{R}_{(1,1)} + \mathbf{R}_{(2,2)} + \mathbf{R}_{(3,3)}} \\ (\mathbf{R}_{(2,3)} - \mathbf{R}_{(3,2)})/\sqrt{1 + \mathbf{R}_{(1,1)} + \mathbf{R}_{(2,2)} + \mathbf{R}_{(3,3)}} \\ (\mathbf{R}_{(3,1)} - \mathbf{R}_{(1,3)})/\sqrt{1 + \mathbf{R}_{(1,1)} + \mathbf{R}_{(2,2)} + \mathbf{R}_{(3,3)}} \\ (\mathbf{R}_{(1,2)} - \mathbf{R}_{(2,1)})/\sqrt{1 + \mathbf{R}_{(1,1)} + \mathbf{R}_{(2,2)} + \mathbf{R}_{(3,3)}} \end{bmatrix}$$

$$\mathrm{Q}^1(\mathbf{R}) = \frac{1}{2}\begin{bmatrix} (\mathbf{R}_{(2,3)} - \mathbf{R}_{(3,2)})/\sqrt{1 + \mathbf{R}_{(1,1)} - \mathbf{R}_{(2,2)} - \mathbf{R}_{(3,3)}} \\ \sqrt{1 + \mathbf{R}_{(1,1)} - \mathbf{R}_{(2,2)} - \mathbf{R}_{(3,3)}} \\ (\mathbf{R}_{(1,2)} + \mathbf{R}_{(2,1)})/\sqrt{1 + \mathbf{R}_{(1,1)} - \mathbf{R}_{(2,2)} - \mathbf{R}_{(3,3)}} \\ (\mathbf{R}_{(3,1)} + \mathbf{R}_{(1,3)})/\sqrt{1 + \mathbf{R}_{(1,1)} - \mathbf{R}_{(2,2)} - \mathbf{R}_{(3,3)}} \end{bmatrix}$$

$$\mathrm{Q}^2(\mathbf{R}) = \frac{1}{2}\begin{bmatrix} (\mathbf{R}_{(3,1)} - \mathbf{R}_{(1,3)})/\sqrt{1 - \mathbf{R}_{(1,1)} + \mathbf{R}_{(2,2)} - \mathbf{R}_{(3,3)}} \\ (\mathbf{R}_{(1,2)} + \mathbf{R}_{(2,1)})/\sqrt{1 - \mathbf{R}_{(1,1)} + \mathbf{R}_{(2,2)} - \mathbf{R}_{(3,3)}} \\ \sqrt{1 - \mathbf{R}_{(1,1)} + \mathbf{R}_{(2,2)} - \mathbf{R}_{(3,3)}} \\ (\mathbf{R}_{(2,3)} + \mathbf{R}_{(3,2)})/\sqrt{1 - \mathbf{R}_{(1,1)} + \mathbf{R}_{(2,2)} - \mathbf{R}_{(3,3)}} \end{bmatrix}$$

$$\mathrm{Q}^3(\mathbf{R}) = \frac{1}{2}\begin{bmatrix} (\mathbf{R}_{(1,2)} - \mathbf{R}_{(2,1)})/\sqrt{1 - \mathbf{R}_{(1,1)} - \mathbf{R}_{(2,2)} + \mathbf{R}_{(3,3)}} \\ (\mathbf{R}_{(3,1)} + \mathbf{R}_{(1,3)})/\sqrt{1 - \mathbf{R}_{(1,1)} - \mathbf{R}_{(2,2)} + \mathbf{R}_{(3,3)}} \\ (\mathbf{R}_{(2,3)} + \mathbf{R}_{(3,2)})/\sqrt{1 - \mathbf{R}_{(1,1)} - \mathbf{R}_{(2,2)} + \mathbf{R}_{(3,3)}} \\ \sqrt{1 - \mathbf{R}_{(1,1)} - \mathbf{R}_{(2,2)} + \mathbf{R}_{(3,3)}} \end{bmatrix}$$

To convert from a quaternion to a rotation matrix, Equation (2.39) is used. This equation is presented in various forms in the literature of quaternions. These other forms can be obtained using the property of unit quaternions that $w^2 + x^2 + y^2 + z^2 = 1$.

$$\mathbf{R} = \begin{bmatrix} w^2 + x^2 - y^2 - z^2 & 2xy + 2wz & 2xz - 2wy \\ 2xy - 2wz & w^2 - x^2 + y^2 - z^2 & 2yz + 2wx \\ 2xz + 2wy & 2yz - 2wx & w^2 - x^2 - y^2 + z^2 \end{bmatrix} \tag{2.39}$$

2.7.3.3 Euler Angles

This section assumes a body fixed 3-2-1 application of Euler angles, where 3-2-1 indicates the order of the specified rotations. In body fixed 3-2-1, a rotation of ψ about the body z-axis is applied, followed by a rotation of θ about the body y-axis, and finally a rotation of ϕ about the body x-axis.

To obtain the quaternion Q from the Euler angles roll ϕ, pitch θ, and yaw ψ, Equation (2.40) is used.

$$\mathrm{Q} = \begin{bmatrix} \cos(\frac{\phi}{2})\cos(\frac{\theta}{2})\cos(\frac{\psi}{2}) + \sin(\frac{\phi}{2})\sin(\frac{\theta}{2})\sin(\frac{\psi}{2}) \\ \sin(\frac{\phi}{2})\cos(\frac{\theta}{2})\cos(\frac{\psi}{2}) - \cos(\frac{\phi}{2})\sin(\frac{\theta}{2})\sin(\frac{\psi}{2}) \\ \cos(\frac{\phi}{2})\sin(\frac{\theta}{2})\cos(\frac{\psi}{2}) + \sin(\frac{\phi}{2})\cos(\frac{\theta}{2})\sin(\frac{\psi}{2}) \\ \cos(\frac{\phi}{2})\cos(\frac{\theta}{2})\sin(\frac{\psi}{2}) - \sin(\frac{\phi}{2})\sin(\frac{\theta}{2})\cos(\frac{\psi}{2}) \end{bmatrix} \tag{2.40}$$

To convert from a quaternion to Euler angles roll ϕ, pitch θ, and yaw ψ, Equation (2.41) is used.

$$\begin{aligned} \phi &= \text{atan2}(2wx + 2yz \ , \ 1 - 2(x^2 + y^2)) \\ \theta &= \arcsin(2(wy - xz)) \\ \psi &= \text{atan2}(2wz + 2xy \ , \ 1 - 2(y^2 + z^2)) \end{aligned} \tag{2.41}$$

2.7.4 Angular Interpolation with Quaternions

Spherical Linear Interpolation (Slerp) is a method of interpolating a 3-D rotation between two orientations [35]. The algorithm assumes that rotation at a uniform angular velocity about a fixed axis occurs between the two orientations. If the two orientations are represented using the quaternions Q_1 and Q_2 sampled at times t_1 and t_2 with $t_2 > t_1$, the trajectory of rotation would lie along the arc between the quaternions located on the sphere of all unit quaternions.

The interpolated orientation estimate represented by a quaternion $\hat{\mathrm{Q}}$ at time instant t for $t_1 \leq t \leq t_2$ is given by Equation (2.42).

$$\hat{\mathrm{Q}} = \mathrm{Q}_1(\mathrm{Q}_1^{-1}\mathrm{Q}_2)^u \tag{2.42}$$

$$u = \frac{t - t_1}{t_2 - t_1} \in [0, 1]$$

To avoid computation of quaternion products and powers, the solution is often implemented using an alternate formulation given in Algorithm 1. The algorithm uses standard linear interpolation at small angles. The numerical stability of the sine operation breaks down for very small angles, so incorrect values could be computed with Slerp. In the first step of the algorithm, the angle between the two quaternions is measured by taking the inverse cosine of $\mathrm{Q}_p \cdot \mathrm{Q}_q$. The operation $\mathrm{Q}_p \cdot \mathrm{Q}_q$ denotes the dot product of the two quaternions in vector form. This dot product is computed in the same manner as a vector dot product.

$$\mathrm{Q}_p \cdot \mathrm{Q}_q = w_p w_q + \mathbf{x}_p \cdot \mathbf{x}_q$$

It is possible to use Slerp to interpolate between points as well as quaternions by representing the points in quaternion format. Such an interpolation

Algorithm 1: SLERP: an algorithm for spherical linear interpolation

Input: Q_p Q_q quaternions representing 3-D orientations
Input: u the interpolation ratio
Output: $\hat{\mathrm{Q}}$ the interpolated orientation

$\gamma \leftarrow \arccos\left(\mathrm{Q}_p \cdot \mathrm{Q}_q\right)$
if $\gamma < 0.01$ **then**
    /* Solution unstable at small angles, do linear interpolation instead */
    $\hat{\mathrm{Q}} \leftarrow (1-u)\mathrm{Q}_p + u\mathrm{Q}_q$
else
    $\hat{\mathrm{Q}} \leftarrow \frac{\sin(\gamma(1-u))}{\sin(\gamma)}\mathrm{Q}_p + \frac{\sin(u\gamma)}{\sin(\gamma)}\mathrm{Q}_q$
return $\hat{\mathrm{Q}}$

between the points $p = (1, 0, 0)$ and $q = (0, 0.5, 0.5)$ is shown in Figure 2.6.

2.7.5 Useful References

Appendix E of [36] gives an overview of orientation representations. The properties of quaternions are given and conversion between quaternions and other orientation representations is detailed. The article [34] is a useful reference for conversion between various orientation representations. It also discusses the representation of coordinate frames, pose, and the transformation between coordinate frames. In [37], the properties of quaternions, their use as representations of orientation, and conversion to and from them are discussed. The use of quaternions to represent rotations is discussed in [35]. This paper also introduces the Slerp algorithm for interpolation using quaternions. The tutorial on quaternions [38] is useful in understanding their mathematical properties. Finally, for the adventurous, there is Hathaway's Primer on Quaternions published in 1896 [39]. The terminology in this book is a little outdated, but it makes for an interesting read and presents a different approach to discussing mathematics from an older era.

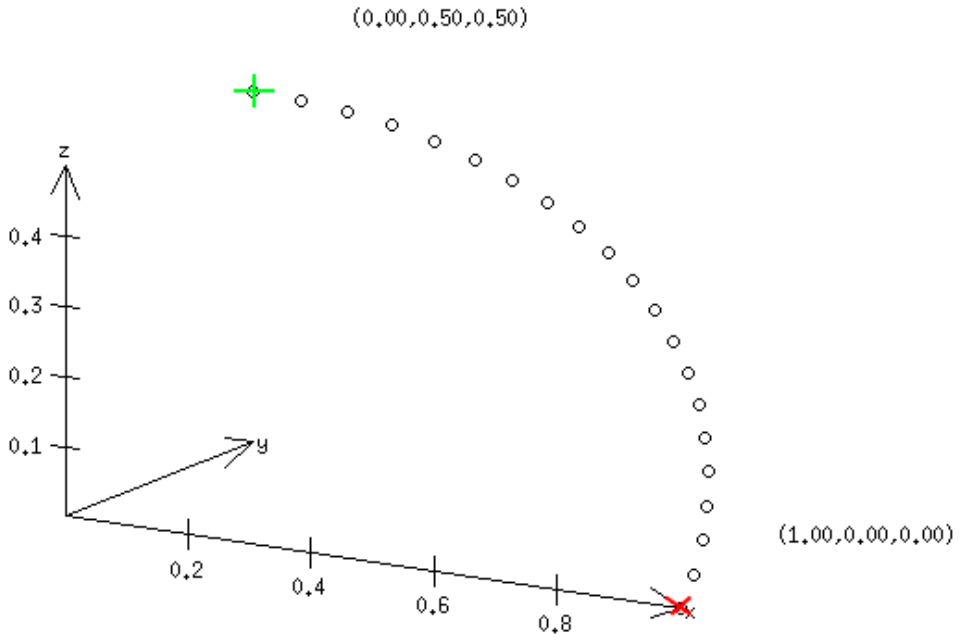


Figure 2.6: Slerp interpolation between two points. The point p is denoted by the red x and q is indicated by the green +. The black circles show sequential interpolations in step sizes of $\Delta u = 0.05$.

2.8 RANSAC

2.8.1 Motivation

Model fitting is a problem that is ubiquitous across the sciences. Least squares [40] and total least squares [41] are typical approaches that work well with data containing no or low level noise. When the experimental data contains large outliers, these approaches break down in effectiveness. Instead of attempting to identify and reject erroneous data points, the techniques incorporate the bad data, resulting in significant skewing of the model [22].

Random Sampling and Consensus (RANSAC) [15] is a model fitting approach that seeks to distinguish between the correct and erroneous components of the data and build a model only for the correct data. First proposed in [15], it is essentially a guess-and-check method where model hypotheses are generated from randomly selected subsets of the data and evaluated against the entire dataset. RANSAC is not an explicit modeling solution itself. It is a framework for model fitting where the user will specify his own modeling parameters, distance metrics, and data assumptions according to the problem at hand.

2.8.2 Basics

The RANSAC algorithm attempts to find the model M^* for a data set P which has a sufficient number of inliers while minimizing the distance between those inliers and the model. The process is described below and outlined in Algorithm 2.

RANSAC consists of a sequence of k trials. In each trial, a subset S_i of n points in the dataset P is drawn with replacement. From this subset a model hypothesis M_i is generated, typically using a least squares approach. This model is then tested against the whole dataset P, and the set of inliers S_i^* is obtained. A point $p \in P$ is considered an inlier if the distance between it and its predicted location according to M_i is less than a threshold t, according to a predefined distance metric. If more than d inliers are found, the model is considered good, and a more refined model M_i^* is generated using S_i^*. This new model is then added to a list of candidate models. The entire process is repeated until k trials have been performed. Once complete, the

Algorithm 2: RANSAC the RANSAC Algorithm

Data: P - the dataset
Input: k - the number of iterations
Input: d - the minimum number of inliers required for a model to be good
Input: n - the number of samples needed to generate a model
Input: t - the threshold specifying how close a point must be to its predicted location in order to be considered an inlier
Output: M^* - the model

```
for i ← 1 to k do
    S_i ← n uniformly drawn random samples from P
    M_i ← the model hypothesis for S_i
    S_i^* ← ∅
    forall p ∈ P do
        if p is an inlier according to M_i and t then
            add p to the set S_i^*
    if more than d inliers were found then
        M_i^* ← model hypothesis generated from all inliers
    else
        M_i^* ← ∅
M^* ← M_1^*
for i = 2 → k do
    if M_i^* exists and its model error is less than the model error of M^* then
        M^* ← M_i^*
return M^*
```

list of candidate models is evaluated against P according to the predefined distance metric, and the model with the smallest total distance between its inliers and their predicted values is returned as M^*. Selection of the number of iterations k is detailed in Section 2.8.3, selection of the number of samples n is detailed in Section 2.8.4, and selection of d the minimum inlier count is discussed in Section 2.8.6.

It should be noted that RANSAC approaches vary a little based on the reference text. The approach described above is the one outlined in [22]. The original algorithm in [15] differs in what is done once d inliers are found. Whereas [22] opts to continue searching and find the model with the smallest error satisfying the d inlier criterion, [15] takes the first model satisfying the constraint as M^* and halts the search there.

2.8.3 Number of Iterations

The number of iterations k is an important parameter that the user must determine. This parameter specifies the number of trials the algorithm must perform in order to be reasonably sure that at least one subset has been drawn which contains no outliers. Although it is possible to select k arbitrarily, most approaches calculate k using some assumptions about the data. The original work [15] proposes two approaches to selecting k. Both approaches make use of another parameter w, which is the probability that a point drawn uniformly from P is an inlier. The parameter w is supplied by the user.

1. The first approach calculates the mean and variance of g, the number of trials required to draw a subset of n samples containing no outliers, and selects a value for k that is one or two standard deviations above the expected value of g. The expected value is given by Equation (2.43). In this derivation, w^n is the probability that all n points in a trial are inliers. The probability that this event occurs for the first time in trial

j is given by $(1-w^n)^{j-1}w^n$.

$$\begin{aligned} E[g] &= P(\text{ 1 good trial in 1 trial}) + 2P(\text{1 good trial in 2 trials}) + \ldots \\ &= w^n + 2(1-w^n)w^n + \ldots j(1-w^n)^{j-1}w^n + \ldots \\ &= w^n[1 + 2(1-w^n) + \ldots j(1-w^n)^{j-1} \ldots \\ &= \frac{w^n}{1-w^n}\sum_{j=0}^{\infty} j(1-w^n)^j \\ &= \frac{w^n(1-w^n)}{(1-w^n)(1-1+w^n)^2} = w^{-n} \end{aligned} \tag{2.43}$$

The standard deviation is given by Equation (2.44) and the value of k is given by Equation (2.45).

$$\begin{aligned} &E[g^2] = \frac{2-w^n}{w^{2n}} \\ &\sigma^2 = E[g^2] - E[g]^2 = \frac{2-w^n}{w^{2n}} - \frac{1}{w^{2n}} = \frac{1-w^n}{w^{2n}} \\ &\sigma = \frac{\sqrt{1-w^n}}{w^n} \end{aligned} \tag{2.44}$$

$$k = \lceil E[g] + 2\sigma \rceil \tag{2.45}$$

Plugging Equation (2.43) and Equation (2.44) into Equation (2.45) yields a recommended value of:

$$k = \left\lceil \frac{1 + 2\sqrt{1-w^n}}{w^n} \right\rceil$$

2. An alternate approach uses z, the desired probability that at least one trial set contains no outliers. The probability z can then be applied to Equation (2.46) to determine k. The probability z is specified by the user as a bound.

$$\begin{aligned} &P(\textit{all sets contain an outlier}) = (1-z) = (1-w^n)^k \\ &\log(1-z) = \log((1-w^n)^k) = k\log(1-w^n) \\ &k = \frac{\log(1-z)}{\log(1-w^n)} \end{aligned} \tag{2.46}$$

2.8.4 Number of Samples

The parameter n is the number of data samples used in each trial. It is generally selected as the minimum number of data points required to generate a model. Applications such as 2-D line fitting use $n = 2$, 2-D circle fits use $n = 3$, and a value of $n = 7$ is used for fundamental matrix estimation [22].

2.8.5 Distance Metric

The choice of a distance and error metric varies based on the model used. In the RGB-D SLAM System, the RANSAC based pairwise alignment uses Mahalanobis distance [42] to measure the distance e between matching 3-D points which have been aligned using the modeled transformation.

The distance e is compared to a user defined threshold t to determine whether the aligned matching points form an inlier or outlier. Inliers are points where $e \leq t$, and outliers are points where $e > t$. The overall model error is found by computing the mean of e for all inliers.

2.8.6 Number of Inliers Required for a Valid Model

The parameter d specifies a lower bound on the number of points that must fall within a distance of t from the model for the model to be considered valid. This parameter is typically selected using y, the probability that a point is within t of an incorrect model generated using n points. The parameter d can then be found such that y^{d-n}, is sufficiently low.

The probability y is also unknown and must be estimated. One popular estimation technique is to assume that $y < (1 - w)$ and then choose d such that $(1 - w)^d$ is less than the desired probability. Other implementations assume that the number of outliers for the true model is less than the number for a randomly selected model, and tune d using the inlier-outlier ratio at each iteration.

2.8.7 Useful References

The original paper [15] gives a fairly detailed and complete explanation of RANSAC. Chapter 15 of [22] provides a similar description of RANSAC as

well as pseudocode for the line fitting case and several other examples of RANSAC implementations.

2.9 Rigid Transformation between Two Trajectories

2.9.1 Motivation

This section will detail an approach to solving the absolute orientation problem. The absolute orientation problem, despite its name, seeks to estimate the rigid transformation consisting of a rotation matrix $\mathbf{R}$ and a translation vector $\boldsymbol{\rho}$ relating two coordinate systems with known correspondences. The thesis [43] phrases the absolute orientation problem very succinctly:

> [The absolute orientation problem is] the determination of the translational, rotational, and uniform scalar correspondence between two different Cartesian coordinate systems given a collection of corresponding point pairs.

One such solution is a least squares approach that seeks to find the rigid transformation that minimizes the sum of squared distances between corresponding points using Singular Value Decomposition (SVD). This approach was first introduced in [44]. In notes on the matter, [45] generalizes this approach to allow for arbitrary weighting of correspondence pairs. This latter approach is detailed below.

2.9.2 Details

Consider two point sets in $\mathbb{R}^3$: $P = \{\mathbf{p}_1 \dots \mathbf{p}_n\}$ and $Z = \{\mathbf{z}_1 \dots \mathbf{z}_n\}$ with correspondences $(\mathbf{p}_i, \mathbf{z}_i)$ for $i = 1 \dots n$. Each pair of correspondences is related by the Equation (2.47), where $\mathbf{R}$ is a 3×3 rotation matrix, $\boldsymbol{\rho}$ is a 3×1 translation vector, and N_i is a 3×1 noise vector.

$$\mathbf{z}_i = \mathbf{R}\mathbf{p}_i + \boldsymbol{\rho} + N_i \tag{2.47}$$

The least squares solution is the transformation $\left\langle \hat{\mathbf{R}}, \hat{\boldsymbol{\rho}} \right\rangle$ that minimizes the weighted sum of squared distance between the transformed points in P

and their corresponding points in Z. This desired transformation is given in Equation (2.48). The weight associated with the i^{th} correspondence pair $(\mathbf{p}_i, \mathbf{z}_i)$ is $w_i \in (0, \infty)$. In this formulation, $\mathbf{R}$ is a 3×3 orthogonal matrix and is not necessarily in SO(3). In the event that it is not, additional steps will be taken to find a rotation matrix which is in SO(3).

$$\left\langle \hat{\mathbf{R}}, \hat{\boldsymbol{\rho}} \right\rangle = \underset{\mathbf{R}, \boldsymbol{\rho}}{\operatorname{argmin}} \sum_{i=1}^{n} w_i ||\mathbf{R}\mathbf{p}_i + \boldsymbol{\rho} - \mathbf{z}_i||^2 \tag{2.48}$$

The paper [45] shows that it is possible to decouple the calculation of $\hat{\mathbf{R}}$ from that of $\hat{\boldsymbol{\rho}}$ by expressing $\boldsymbol{\rho}$ in terms of the 3-D rotation matrix $\mathbf{R}$ and $\bar{\mathbf{p}}$ and $\bar{\mathbf{z}}$, the centroids of P and Z respectively. Using this substitution it is possible to first solve for $\hat{\mathbf{R}}$ and then use that result to find $\hat{\boldsymbol{\rho}}$.

$$\bar{\mathbf{p}} = \frac{\sum_{i=1}^{n} w_i \mathbf{p}_i}{\sum_{i=1}^{n} w_i} \qquad \bar{\mathbf{z}} = \frac{\sum_{i=1}^{n} w_i \mathbf{z}_i}{\sum_{i=1}^{n} w_i}$$

The substitution is found by differentiating with respect to $\boldsymbol{\rho}$ the cost function minimized in Equation (2.48). The derivative is then set to 0 and $\boldsymbol{\rho}$ is solved for using algebraic manipulations.

$$\begin{aligned}
0 &= \frac{d}{d\boldsymbol{\rho}} \left(\sum_{i=1}^{n} w_i ||\mathbf{R}\mathbf{p}_i + \boldsymbol{\rho} - \mathbf{z}_i||^2 \right) \\
0 &= 2 \sum_{i=1}^{n} w_i \left(\mathbf{R}\mathbf{p}_i + \boldsymbol{\rho} - \mathbf{z}_i \right) \\
0 &= \boldsymbol{\rho} \sum_{i=1}^{n} w_i + \sum_{i=1}^{n} w_i \left(\mathbf{R}\mathbf{p}_i - \mathbf{z}_i \right) \\
\boldsymbol{\rho} &= \frac{-\sum_{i=1}^{n} w_i \left(\mathbf{R}\mathbf{p}_i - \mathbf{z}_i \right)}{\sum_{i=1}^{n} w_i} = \bar{\mathbf{z}} - \mathbf{R}\bar{\mathbf{p}}
\end{aligned} \tag{2.49}$$

The substitution given in Equation (2.49) is then used in Equation (2.48) to express the minimization in terms of only $\mathbf{R}$, given in Equation (2.50). In this equation, the terms $\mathbf{p}'_i$ and $\mathbf{z}'_i$ are the difference between the points, $\mathbf{p}_i$

and $\mathbf{z}_i$, and their centroids, $\bar{\mathbf{p}}$ and $\bar{\mathbf{z}}$.

$$\mathbf{p}'_i = \mathbf{p}_i - \bar{\mathbf{p}} \quad , \quad \mathbf{z}'_i = \mathbf{z}_i - \bar{\mathbf{z}}$$

$$\begin{aligned}
\hat{\mathbf{R}} &= \underset{\mathbf{R}}{\operatorname{argmin}} \sum_{i=1}^{n} w_i ||(\mathbf{R}\mathbf{p}_i + \bar{\mathbf{z}} - \mathbf{R}\bar{\mathbf{p}}) - \mathbf{z}_i||^2 \\
&= \underset{\mathbf{R}}{\operatorname{argmin}} \sum_{i=1}^{n} w_i ||(\mathbf{R}(\mathbf{p}_i - \bar{\mathbf{p}}) - (\mathbf{z}_i - \bar{\mathbf{z}})||^2 \\
&= \underset{\mathbf{R}}{\operatorname{argmin}} \sum_{i=1}^{n} w_i ||(\mathbf{R}\mathbf{p}'_i - \mathbf{z}'_i||^2
\end{aligned} \tag{2.50}$$

The orthogonal matrix $\hat{\mathbf{R}}$ minimizing Equation (2.50) is then found by expressing Equation (2.50) as a maximization of a matrix trace and using properties of traces to find $\hat{\mathbf{R}}$.

Expanding the squared magnitude term in Equation (2.50) and eliminating terms that do not affect the minimization [44] gives:

$$\begin{aligned}
\hat{\mathbf{R}} &= \underset{\mathbf{R}}{\operatorname{argmin}} - 2 \sum_{i=1}^{n} w_i (\mathbf{z}'_i)^T \mathbf{R} \mathbf{p}'_i \\
&= \underset{\mathbf{R}}{\operatorname{argmax}} \sum_{i=1}^{n} w_i (\mathbf{z}'_i)^T \mathbf{R} \mathbf{p}'_i \\
&= \underset{\mathbf{R}}{\operatorname{argmax}} \operatorname{Trace} \left(\mathbf{R} \sum_{i=1}^{n} w_i (\mathbf{z}'_i)^T \mathbf{p}'_i \right) \\
&= \underset{\mathbf{R}}{\operatorname{argmax}} \operatorname{Trace} (\mathbf{R}\mathbf{H})
\end{aligned} \tag{2.51}$$

where:

$$\mathbf{H} = \sum_{i=1}^{n} w_i \mathbf{z}'_i (\mathbf{p}'_i)^T$$

The singular value decomposition (SVD) of $\mathbf{H}$ is then taken.

$$\mathbf{H} = \mathbf{U}\mathbf{\Lambda}\mathbf{V}^T$$

Substituting the SVD into the trace, it is shown in [45] that $\hat{\mathbf{R}}$ can be

expressed as in Equation (2.52).

$$\begin{aligned}\hat{\mathbf{R}} &= \underset{\mathbf{R}}{\text{argmax}}\ \text{Trace}\,(\mathbf{RH}) \\ &= \underset{\mathbf{R}}{\text{argmax}}\ \text{Trace}\left(\mathbf{RU\Lambda V}^T\right) \\ &= \underset{\mathbf{R}}{\text{argmax}}\ \text{Trace}\left(\ \mathbf{\Lambda V}^T\mathbf{RU}\right) \end{aligned} \tag{2.52}$$

Since $\mathbf{\Lambda}$ is diagonal and non-negative and the product $\mathbf{V}^T\mathbf{RU}$ is an orthogonal matrix, the Trace$\left(\ \mathbf{\Lambda V}^T\mathbf{RU}\right)$ satisfies the inequality:

$$\text{Trace}\left(\ \mathbf{\Lambda V}^T\mathbf{RU}\right) < \text{Trace}\left(\ \mathbf{\Lambda}\right)$$

Equation (2.52) is therefore maximized when $\mathbf{V}^T\mathbf{RU}$ is the identity matrix. The orthogonal matrix $\hat{\mathbf{R}}$ can then be found.

$$\begin{aligned}\mathbf{I} &= \mathbf{V}^T\hat{\mathbf{R}}\mathbf{U} \\ \hat{\mathbf{R}} &= \mathbf{VU}^T\end{aligned}$$

The orthogonal matrix $\hat{\mathbf{R}}$ found above could incorporate reflection as well as rotation. To ensure that $\hat{\mathbf{R}}$ is only a rotational matrix, the determinant of $\mathbf{VU}^T$ must be examined. If the determinant is one, $\mathbf{VU}^T$ is the optimal rotation matrix.

If the determinant is -1 and the smallest singular value is 0, then the matrix that minimizes Equation (2.48) is a reflection, and the optimal rotation matrix is given by the next best minimization $\mathbf{V}^r\mathbf{U}^T$, where $\mathbf{V}^r = \begin{bmatrix}\mathbf{v}_1 & \mathbf{v}_2 & -\mathbf{v}_3\end{bmatrix}$. If the determinant is -1 and all of the singular values are nonzero, this indicates that there is a large amount of noise in one of the point sets and the least-squares SVD approach is not suitable [44].

In [45] the first two cases are combined into a convenient general formula:

$$\hat{\mathbf{R}} = V\begin{bmatrix}1 & & & \\ & \ddots & & \\ & & 1 & \\ & & & \det(VU^T)\end{bmatrix}U^T \quad , \quad \hat{\boldsymbol{\rho}} = \bar{\mathbf{z}} - \hat{\mathbf{R}}\bar{\mathbf{p}} \tag{2.53}$$

For a more thorough discussion of derivation of this SVD based approach

and a richer discussion of the orientation rectification of reflections, the reader is encouraged to consult [44] and [45].

Applications of the least squares SVD algorithm in this thesis use a uniform unit weight $w_i = 1, \forall i$. A summarization of the approach with unit weights is given in Algorithm 3.

Algorithm 3: LEAST-SQUARES SVD: an algorithm for finding the transformation relating two corresponding point sets

Data: P, Z - corresponding point sets

Output: $\left\langle \hat{\mathbf{R}}, \hat{\boldsymbol{\rho}} \right\rangle$ - The optimal transformation

`/* Find the centroids for` P `and` Z `*/`

$\bar{\mathbf{p}} \leftarrow \frac{1}{n} \sum_{i=1}^{n} \mathbf{p}_i$

$\bar{\mathbf{z}} \leftarrow \frac{1}{n} \sum_{i=1}^{n} \mathbf{z}_i$

`/* Center each centroid at the origin */`

$P' \leftarrow \{\mathbf{p} - \bar{\mathbf{p}} \mid \mathbf{p} \in P\}$

$Z' \leftarrow \{\mathbf{z} - \bar{\mathbf{z}} \mid \mathbf{z} \in Z\}$

`/* Compute the sum of outer products` H `and its SVD */`

$H \leftarrow \sum_{i=1}^{n} \mathbf{p}' \mathbf{z}'^T$

$(U, \Lambda, V) \leftarrow \text{SVD}(H)$

`/* Compute` $\hat{\mathbf{R}}$ `*/`

$$\hat{\mathbf{R}} \leftarrow V \begin{bmatrix} 1 & & & \\ & \ddots & & \\ & & 1 & \\ & & & \det(VU^T) \end{bmatrix} U^T$$

`/* Compute` $\hat{\boldsymbol{\rho}}$ `*/`

$\hat{\boldsymbol{\rho}} \leftarrow \bar{\mathbf{z}} - \hat{\mathbf{R}}\bar{\mathbf{p}}$

return $\left\langle \hat{\mathbf{R}}, \hat{\boldsymbol{\rho}} \right\rangle$

2.9.3 Useful References

This least-squares SVD algorithm for solving the absolute orientation problem was introduced in [44]. The paper provides a thorough explanation of the algorithm as well as a detailed derivation of the optimal rotation matrix. The optimal translation derivation is not included, citing an earlier work

[46]. The algorithm proposed in [44] assigns a uniform weight to all point correspondences. The paper [45] expands the algorithm to allow for varying correspondence weights. The derivation for optimal translation and rotation is also detailed in [45].

A summary paper [47] provides a detailed description and comparison of the SVD technique and three other algorithms used to solve the absolute orientation problem. The paper explains the details of each algorithm and evaluates them in terms of robustness, accuracy, stability, and efficiency. It is a good first reference for understanding the absolute orientation problem.

Chapter 3

SLAM - Simultaneous Localization and Mapping

This chapter will introduce the topic of Simultaneous Localization and Mapping (SLAM). Section 3.1 will present the SLAM problem and discuss fundamental concepts. Section 3.2 will introduce several probabilistic SLAM techniques and discuss their salient details. Trajectory-oriented SLAM will also be discussed in this section. In Section 3.3, several SLAM approaches using an RGB-D camera will be covered. Finally, Section 3.4 will introduce and explain the SLAM algorithm used in the University of Freiburg's RGB-D SLAM System [1].

The SLAM tutorials [48],[49] are a useful introduction to simultaneous localization and mapping and its history. Many of the concepts discussed in this chapter are explained in greater detail in these tutorials.

3.1 Basics

The SLAM problem, as the name suggests, is a problem of localization and mapping. Placed in an environment with unknown structure at an unknown pose, a robot is tasked with generating a map of the environment and determining its location in this map. Given no prior knowledge of the environment, the robot uses the evolution of its sensor data over time to build a map M of the environment and estimate a history of its poses.

As with other parts of this thesis, a robot's pose $\bar{\mathbf{T}}$ is described as transformation from a coordinate frame C attached to the robot to the world frame W; the map is defined relative to W. The pose is given in quaternion format as $\langle \mathrm{Q} \ , \ \boldsymbol{\rho} \rangle$. The term Q is a quaternion describing the rotational component of the rigid transformation from C to W and $\boldsymbol{\rho}$ is the translational component of this transformation.

$$\bar{\mathbf{T}} \Leftrightarrow \langle \mathrm{Q} \ , \ \boldsymbol{\rho} \rangle$$

SLAM is performed by using the sensor data to identify landmarks in the environment and examining how the position of the robot relative to these landmarks changes as the robot moves through the environment.

SLAM can either be online or offline. Offline SLAM techniques gather all available sensor data and then use it compute the map M of the environment and estimate the trajectory $\{\bar{\mathbf{T}}_1, \bar{\mathbf{T}}_2, \dots\}$ the robot traveled.

Offline techniques typically do not run at near real time rates, processing the data at a slower rate than that at which it is supplied. Offline techniques are not causal. Each pose estimate leverages all available data, not just the data recorded prior to the robot reaching the pose in question.

Online SLAM runs in real time and estimates the current pose of the robot using only previously obtained data. The poses and map obtained using online SLAM are generally not as accurate as offline ones. Online SLAM trades accuracy for faster run times and the ability to localize the robot in real time. This thesis will focus on the online variety of SLAM.

3.2 SLAM Techniques

There are many ways to approach solving SLAM. Most popular approaches are probabilistic in nature, using Bayesian inference to estimate the pose and map. The text [21] gives a thorough explanation of several fundamental approaches to probabilistic SLAM.

Probabilistic SLAM seeks to estimate the posterior probability function given in Equation (3.1) [21]. In this function, the pose $\bar{\mathbf{T}}_t$ is the pose at the sampling instant t, $\mathbf{z}_{1:t}$ represents the sensor data recorded up to time t, and $\mathbf{u}_{1:t}$ gives the control inputs to the robot up to time t.

$$P\left(\bar{\mathbf{T}}_t, M \mid \mathbf{z}_{1:t}, \mathbf{u}_{1:t}\right) \tag{3.1}$$

This probability function is estimated and updated by sequentially applying time-update and measure-update steps. The time update step uses the motion model $P\left(\bar{\mathbf{T}}_{t+1} \mid \bar{\mathbf{T}}_t, \mathbf{u}_{1:t+1}\right)$. The motion model finds the probability distribution for a new pose $\bar{\mathbf{T}}_{t+1}$ given the current pose $\bar{\mathbf{T}}_t$ and the control

inputs $\mathbf{u}_{1:t+1}$ up to time $t+1$. This motion model is incorporated into the pose and map posterior during the time-update, giving the new pose and map posterior.

$$P\left(\bar{\mathbf{T}}_{t+1}, M \mid \mathbf{z}_{1:t}, \mathbf{u}_{1:t+1}\right)$$

The measurement update step then uses observation model

$$P\left(\mathbf{z}_{t+1} \mid \bar{\mathbf{T}}_{t+1}, M\right)$$

to incorporate a new set of measurements $\mathbf{z}_{t+1}$ into the pose and map posterior. This produces the fully updated posterior.

$$P\left(\bar{\mathbf{T}}_{t+1}, M \mid \mathbf{z}_{1:t+1}, \mathbf{u}_{1:t+1}\right)$$

Three of the most well-known classes of probabilistic SLAM are EKF-SLAM, GraphSLAM, and FastSLAM.

3.2.1 EKF-SLAM

EKF-SLAM is a widely used probabilistic SLAM approach. It uses an Extended Kalman Filter (EKF) to estimate Equation (3.1) as a Gaussian distribution. The approach assumes uncorrelated zero mean Gaussian noise terms $\mathbf{w}_t$ and $\mathbf{v}_t$ in the motion and observation models given in Equation (3.2). The functions $\mathbf{f}(\cdot)$ and $\mathbf{h}(\cdot)$ are non-linear deterministic equations modeling the motion and sensor observations of the robot.

$$\begin{aligned} \bar{\mathbf{T}}_{t+1} &= \mathbf{f}\left(\bar{\mathbf{T}}_t, \mathbf{u}_{t+1}\right) + \mathbf{w}_{t+1} \\ \mathbf{z}_{t+1} &= \mathbf{h}\left(\bar{\mathbf{T}}_{t+1}, M\right) + \mathbf{v}_{t+1} \end{aligned} \tag{3.2}$$

These equations and their Taylor series based linearizations are used to develop the Extended-Kalman filter equations used to estimate μ_t and Σ_t, the mean and covariance terms for the multivariate Gaussian approximation of Equation (3.1).

3.2.2 GraphSLAM

GraphSLAM makes use of the Extended Information Filter (EIF) [21]. Similar to EKF-SLAM, this approach approximates Equation (3.1) as a multivariate Gaussian, but does so using the canonical parameterization of the Gaussian. It finds the information matrix Ω_t and the information vector ξ_t in addition to the mean vector μ_t. A derivation of the canonical parameterization is given in Chapter 3.5 of [21]. The canonical parameterization terms are related to their standard counterparts in Equation (3.3).

$$\Omega = \Sigma^{-1} \qquad \xi = \Sigma^{-1}\mu \tag{3.3}$$

3.2.3 FastSLAM

FastSLAM [50] is a particle filter [51],[52] based SLAM approach. It estimates Equation (3.1) by factoring it into separate posteriors, with one for the robot poses and one for each landmark in the map. This factorization is made possible by the observation that the location of individual landmarks of a map are conditionally independent of each other given the poses of the robot. The re-factorization is given in Equation (3.4) [21].

$$P\left(\bar{\mathbf{T}}_t, M \mid \mathbf{z}_{1:t}, \mathbf{u}_{1:t}\right) = P\left(\bar{\mathbf{T}}_t \mid \mathbf{z}_{1:t}, \mathbf{u}_{1:t}\right) \prod_{m \in M} P\left(m \mid \mathbf{z}_{1:t}, \mathbf{u}_{1:t}\right) \tag{3.4}$$

The pose posterior is estimated using a particle filter. Each particle consists of a pose hypothesis and an individual EKF for each landmark m in the map M. This approach allows FastSLAM to maintain different data-landmark associations in separate particles, making it more robust to data association problems than EKF-SLAM or GraphSLAM. FastSLAM owes its name to the fact that it can be implemented in logarithmic time complexity, giving it computational advantages over EKF [21].

3.2.4 Trajectory-Oriented SLAM

In its standard formulation, SLAM seeks to estimate the pose of the robot and the location of landmarks in the map. An alternate approach, termed trajectory-oriented SLAM, instead seeks to estimate the robot's trajectory

in the form of a sequence of discrete poses [48].

This approach is particularly useful when working with a large amount of sensor data, such as the scans produced by a laser range finder. In these situations it is more reliable and simpler to align the scans rather than use them to identify landmarks [48].

One such variant is consistent pose estimation (CPE) [53],[54],[55],[56]. In CPE, the robot's poses are represented as the nodes of a graph. The edges in the graph give the constraints between nodes as the rigid transformations between the nodes' poses. Each edge connecting two poses $\bar{\mathbf{T}}_i$ and $\bar{\mathbf{T}}_j$ has an observation $\mathbf{T}'_{i,j}$ giving the true transformation between the poses. The transformation $\tilde{\mathbf{T}}_{i,j}$ that aligns the scans taken at those two nodes is termed the measurement. This measurement $\tilde{\mathbf{T}}_{i,j}$ is modeled as the observation $\mathbf{T}'_{i,j}$ corrupted by additive zero-mean Gaussian noise $\mathbf{v}_{i,j}$ [53].

$$\tilde{\mathbf{T}}_{i,j} = \mathbf{T}'_{i,j} + \mathbf{v}_{i,j}$$

SLAM, formulated in this manner, is a GraphSLAM variant. The optimal length n sequence of trajectories $\bar{T}$ can be found by minimizing Equation (3.5) [57]. In this equation, $\mathbf{e}(\mathbf{T}_i, \mathbf{T}_j, \tilde{\mathbf{T}}_{i,j})$ is a vector error function composed of a quaternion in vector form concatenated with a translation vector. Together, these terms measure how well the transformation between poses $\mathbf{T}_i$ and $\mathbf{T}_j$ matches the measurement $\tilde{\mathbf{T}}_{i,j}$. The set $\mathcal{C}$ is the set of all edges in the pose graph. The information matrix $\Omega_{i,j}$ is the inverse of the edge covariances.

$$\bar{T} = \left\{\bar{\mathbf{T}}_1, \ldots, \bar{\mathbf{T}}_n\right\}$$

$$\bar{T} = \underset{\{\mathbf{T}_1 \in \mathrm{SE}(3) \ldots \mathbf{T}_n \in \mathrm{SE}(3)\}}{\operatorname{argmin}} \sum_{<i,j> \in \mathcal{C}} \mathbf{e}(\mathbf{T}_i, \mathbf{T}_j, \tilde{\mathbf{T}}_{i,j})^T \; \Omega_{i,j} \; \mathbf{e}(\mathbf{T}_i, \mathbf{T}_j, \tilde{\mathbf{T}}_{i,j}) \qquad (3.5)$$

The map when using CPE is not found explicitly. It can be obtained by merging the scans taken at each pose.

3.3 RGB-D SLAM

RGB-D cameras recording color and range images have grown in popularity in recent years with the release of the low cost and widely available Kinect sensor. Most SLAM implementations using these sensors are reminiscent of a laser range finder SLAM approach, leveraging dense 3-D point clouds obtained from the range images to approximate motion between poses as a rigid transformation between corresponding point clouds. This section will discuss alternatives to the RGB-D SLAM System [1] that all make use of RGB-D cameras.

Willow Garage [2] developed a SLAM application using the Kinect. Their CPE based approach uses pairwise alignment estimates between sequential range images to build a pose graph. Pose-pose transformation constraints, point-point distance constraints, and point-point color intensity constraints are encoded into this graph. The graph is optimized using a $\mathbf{g^2o}$ non-linear graph optimizer [58] to obtain global alignment. Pairwise alignment between sequential frames is estimated using an iterative closest point (ICP) algorithm [14]. The range images are uniformly downsampled and matching points are identified. ICP is then used to recover the rigid transformation between point clouds.

The paper [59] presents a novel approach to SLAM based on image warping. Localization is done by comparing each new image to a reference image. Every time a new RGB-D image is recorded, that image is warped according to the previous pose estimate and a motion model in order to approximate the reference image. The relative motion from the last frame is then found by identifying the pose refinement that produces the warped image that is most similar to the reference image in terms of pixel intensity and depths.

The paper [3] presents a SLAM method similar in approach to [1]. SLAM is performed by using pairwise image alignments to construct a pose graph and then optimizing that pose graph to obtain global alignment. Alignment between two images is performed by finding the rigid transformation between 3-D point clouds constructed from SIFT feature points. SIFT features are extracted from both images and 3-D point clouds are constructed using inverse perspective projection and the depth of the features given by the registered range image. Matching features are identified through descriptor comparison and a RANSAC-based approach is used to estimate the trans-

formation between point clouds. This transformation is used to initialize an ICP algorithm that produces a refined transformation estimate. The camera poses at the times the images were captured are then encoded into the pose graph with the relative transformation as a constraint. The pose graph is optimized using TORO [60], a gradient decent based optimizer.

3.4 RGB-D SLAM Details

The RGB-D SLAM System [1] is an open source RGB-D SLAM implementation on the robotics platform ROS. It is a six degree of freedom SLAM technique capable of supplying real time pose estimates at a rate of 3-4 Hz and producing dense RGB point cloud based maps. At the time of this writing, the RGB-D SLAM System is one of the most prominent open source SLAM solutions for use with Kinect style RGB-D cameras.

The RGB-D SLAM System's algorithm is similar to [3]. It is a trajectory-oriented SLAM technique, meaning it seeks to estimate only the pose of the camera and not the location of landmarks in the map. As with most trajectory-oriented SLAM algorithms, the map is not explicitly estimated. It is constructed by merging 3-D point clouds constructed from the RGB and depth images taken at each pose.

Pairwise alignments between images are found and added as constraints between poses in a pose graph. A $\mathbf{g^2o}$ non-linear graph optimizer is then used to perform global alignment. In the literature, the initial pairwise alignment is referred to as the SLAM front end and the graph optimization is the back end. Figure 3.1 gives a flow chart depicting the major steps in the RGB-D SLAM System algorithm.

The pose graph plays a key role in the RGB-D SLAM Systems. Nodes of the pose graph are significant poses or "key frames" that have been selected for their distinct location in the map. Together, they make up a discrete estimate of the camera's trajectory. The pose graph is integral in localizing new frames of RGB-D data in the map and determining what to do with those frames once their relative alignment has been determined.

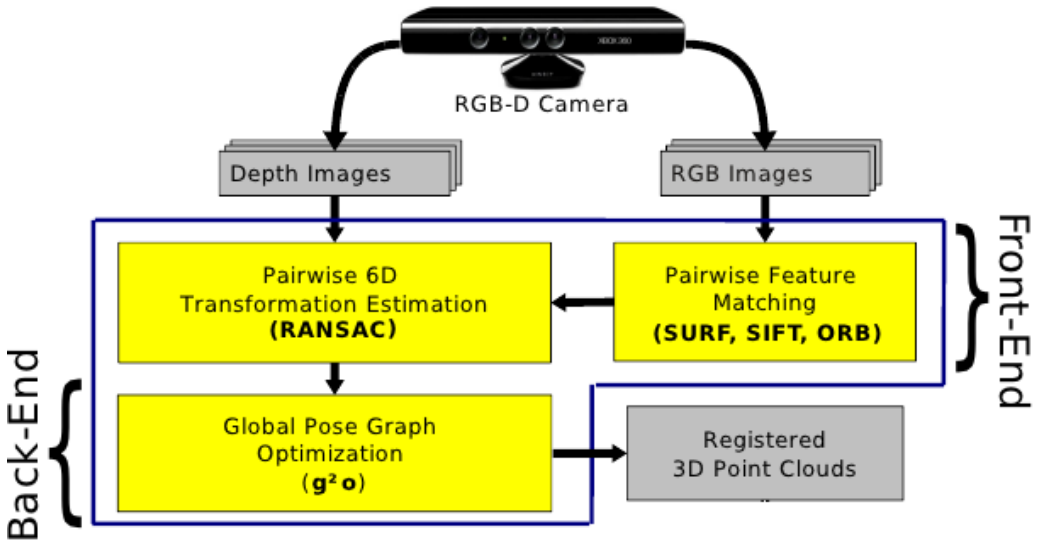


Figure 3.1: A flowchart illustrating the major steps in the RGB-D SLAM System's approach. This figure is a modified version of one found in [1].

3.4.1 RGB-D SLAM System Front End

The RGB-D SLAM System's front end takes an RGB-D image and determines the pose of the camera at the time the image was captured. This pose is expressed as $\bar{\mathbf{T}}$, the transformation from the camera frame to the world frame at that time instant.

When a new frame of data is received, features are located in the RGB image and descriptors are extracted. A 3-D point cloud is constructed, using these features and their associated depth values to perform inverse perspective projection. This point cloud, consisting of the locations of the interest points in 3-D, is termed a landmark point cloud Γ. The descriptors of these features are matched with those of the most recent key frame to identify point correspondences.

Pairwise alignment of the new frame with index i and a key frame with index j is done using a RANSAC-based transformation estimator to estimate $\tilde{\mathbf{T}}_{i,j}$ the rigid transformation that aligns Γ_i with Γ_j. If the transformation is not large in terms of translation or rotation, the pose $\bar{\mathbf{T}}_i$ of the new frame is computed by combining $\tilde{\mathbf{T}}_{i,j}$ with the key frame's pose and processing for that frame halts.

The RANSAC based transformation estimator estimates $\tilde{\mathbf{T}}_{i,j}$ by finding the transformation that minimizes the Euclidean distance between inlier matches. Matching points in the landmark point clouds are repeatedly sampled in random sets of three matches and the rigid transformations between matching points are estimated using the SVD based least squares approach detailed in Section 2.9. Inliers of the transformation which produced the largest amount of inlier matches are used to refine that transformation, yielding the final transformation between frames.

3.4.2 RGB-D SLAM System Back End

If no transformation between the most recent key frame and the new frame can be recovered, or if the transformation is large, the new frame is added as a node in the pose graph. Pairwise alignment between the new frame and each node in the pose graph is then performed. Edge weights between the new node with index i and each matching node with index j are encoded in the pose graph using an measurement given by the transformation $\tilde{\mathbf{T}}_{i,j}$ and

a matrix $\Omega_{i,j}$ which assigns weight based on the number of inliers.

Every time a new node is added to the pose graph, the pose graph is optimized using $\mathbf{g}^2\mathbf{o}$, a non-linear graph optimizer [58]. The graph optimizer finds the sequence of poses $\bar{T}$ that minimize a non-linear error function. This equation, given in the trajectory-oriented SLAM section, is restated here as Equation (3.6) [58].

$$\bar{T} = \left\{\bar{\mathbf{T}}_1, \ldots, \bar{\mathbf{T}}_n\right\}$$

$$\bar{T} = \underset{\{\mathbf{T}_1 \in \mathrm{SE}(3) \ldots \mathbf{T}_n \in \mathrm{SE}(3)\}}{\operatorname{argmin}} \sum_{<i,j> \in \mathcal{C}} \mathbf{e}(\mathbf{T}_i, \mathbf{T}_j, \tilde{\mathbf{T}}_{i,j})^T \; \Omega_{i,j} \; \mathbf{e}(\mathbf{T}_i, \mathbf{T}_j, \tilde{\mathbf{T}}_{i,j}) \qquad (3.6)$$

In Equation (3.6), the error function $\mathbf{e}(\cdot)$ measures the difference between the observed transformation $\tilde{\mathbf{T}}_{i,j}$ and the actual transformation between poses $\mathbf{T}_i$ and $\mathbf{T}_j$. This error function produces a vector of quaternion rotation and translation terms measuring how well the actual transformation matches the measured transformation. This vector is $\mathbf{0}$ when the transformations match exactly. A more complete explanation of this error function is given in [1] and [58]. The value $\mathcal{C}$ is the set of correspondences represented by connected nodes. The matrix $\Omega_{i,j}$ assigns a weight to those nodes.

3.4.3 RGB-D SLAM Map Generation

The RGB-D SLAM map is generated using keyframes and their associated point clouds. Point clouds are defined relative to the camera frame C.

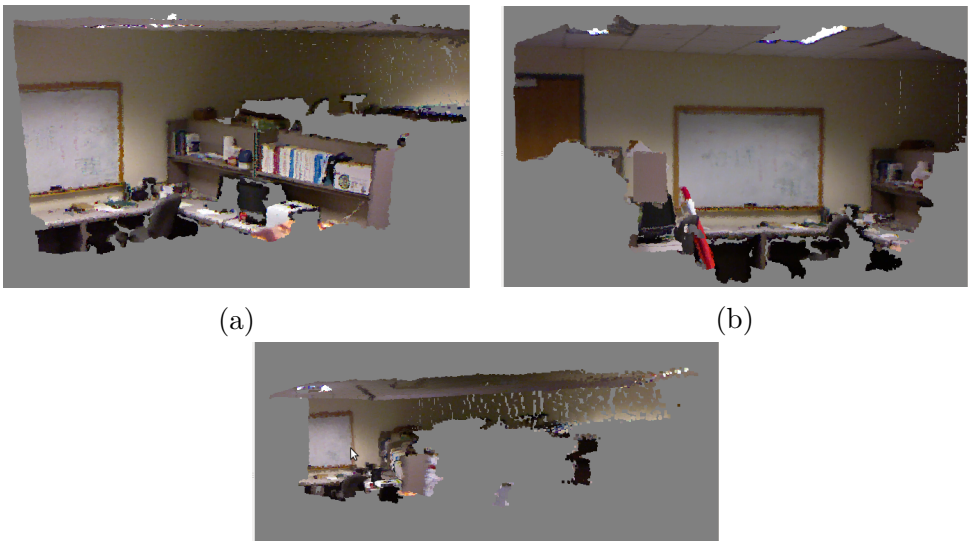
The point cloud P is formed by reconciling the 3-D locations of the pixels in the RGB frame recorded at the sampling instant using inverse perspective projection and the registered depth map, as discussed in Section 2.5. Each point in P consists of a color, given in RGB intensities, and a location, given in 3-D Euclidean coordinates.

The landmark point cloud Γ is also created using inverse perspective projection. It is found by pairing the interest point locations obtained from the RGB frame with their corresponding depths and reconciling the location of these points in 3-D space. Points in Γ consist only of a 3-D location and do not contain color intensity information.

The RGB-D SLAM System's map M is formed by transforming each keyframe point cloud P_i into the world frame W using $\mathbf{T}_i$ and then merging the point clouds for each keyframe into a single point cloud. An example map is shown in Figure 3.2 and three of the point clouds used to generate the map are shown in Figure 3.3. The map generation function is given in Equation (3.7) where n is the number of nodes in the pose graph.

$$M = \bigcup_{i=1}^{n} {}^{W}P_i \tag{3.7}$$

Figure 3.2: A map generated by the RGB-D SLAM System.



(a) (b)

(c)

Figure 3.3: Three of the keyframe point clouds used to construct the map in Figure 3.2.

A landmark map can also be formed using the keyframes. This is accom-

plished by transforming each landmark point cloud Γ_i into the W coordinate frame and merging them into a single point cloud.

Chapter 4

Interest Point Additions and Testing

This chapter will discuss work on the incorporation of an alternative feature detector, AGAST, and alternative descriptors, BRISK and FREAK, into the RGB-D SLAM System's code-base. It will explain the motivation for using more advanced binary descriptors and AST detectors, compared to the existing ORB and SURF solutions. It will also explain the tunable parameters in these descriptors and detectors, and their effects on the SLAM algorithm. Use of dynamic adaptive feature detection in OpenCV will also be discussed. This class of detector dynamically changes the parameter values of the underlying feature detector, so as to maintain a consistent number of interest points in an image sequence.

The detection, description, and extraction techniques used in obtaining Binary Robust Invariant Scalable Keypoints (BRISK) and Fast Retina Keypoints (FREAK) are developed in greater detail in Chapter 2.6. The RGB-D SLAM System that is the focus of the alterations detailed in this chapter is explained in Section 3.3.

4.1 Motivation

Interest points or features play a key role in the RGB-D SLAM System [1]. These interest points, reconstructed into 3-D point clouds using inverse perspective projection, serve as SLAM landmarks. Sets of matching interest point are used to identify the same landmarks across pairs of frames. The reconstructed 3-D locations of matching points are then used to perform pairwise alignment of the point clouds, where the rigid transformation between the camera poses used to generate the point clouds is measured. This transformation is encoded as an edge constraint in the pose graph, with the number of matching interest points assigning a weight to this edge. Interest

points are also used in the selection of key frames, where the dissimilarity between the point sets of the current frame and existing key-frames is used as a metric in determining when to add a new key-frame. The current implementation of the RGB-D SLAM System makes use of a choice of three interest point packages: SIFT [7], SURF [8], and ORB [61].

SIFT and SURF are both algorithms that produce robust and well localized features, but at a high computational cost. One study [9] using a single core of an i7 2.67 GHZ processor found average detection times of 1611 ms per frame with SIFT and 107 ms per frame with SURF for 800×640 images. Extraction and matching times were even worse with averages of 9784 ms per frame and 291.6 ms per frame for SIFT and 559.1 ms per frame and 194.6 ms per frame for SURF. While these times can be decreased through careful tuning of parameters and down-sampling of the image, it is clear that, at the moment, these feature point systems are too computationally intensive for use at high frame rates.

In pursuit of more efficient feature detection, ORB features are also included in the default installation of the RGB-D SLAM System. This FAST based detection system runs at a significantly greater speed than SURF. One paper [61] recorded detection times of 15.3 ms per 800×600 frame, compared to SURF's 217.3 ms per frame and SIFT's 5228.7 ms per frame. While ORB's speed allows for real time detection, ORB pays a price in terms of accuracy and precision. Its simpler descriptors and less robust features impair detection and matching. In testing of the ORB implementation, the RGB-D SLAM System would lose track or fail completely at times of large angular movement. SURF features, which have been found to be robust to changes in rotation and perspective [8], had no such problem.

The sluggishness of SIFT and SURF and the inaccuracy of ORB leave us wanting for a detector which is fast enough to run at near real time speeds, but robust and accurate enough to provide meaningful SLAM results. Recent developments in the field of corner detectors and binary descriptors suggest two potential candidates in the form of BRISK [9] and FREAK [10]. These two algorithms claim detection times on the order of 35-40 ms per frame with robustness and matching accuracy comparable to SURF and SIFT.

4.2 Preliminary Work

In order to begin work with BRISK and FREAK, it was necessary to first gain an understanding of their behavior. Important questions regarding their performance in the ROS environment and their response to a real time video feed from the Kinect camera needed answering. To this end, a test program was constructed.

This modular program allowed for evaluation of interest point packages without the overhead and additional complexity of a full SLAM algorithm. It facilitated the characterization of the run time performance of various combinations of interest point detectors, descriptor extractors, and feature matching algorithms. It also enabled evaluation of the quantity and quality of feature matches obtained under these configurations with varied parameter settings.

The test platform was developed using the ROS "Fuerte" robotic simulation package [18] and the OpenCV library v2.4.3 [17] on a machine running the Ubuntu 11.10 "Oneiric" Linux distribution. The visual input device was a Microsoft Kinect RGB-D camera, recording 640×480 RGB and IR-Depth images at a rate of 30 Hz. OpenNI drivers were used to interface with Kinect and the ROS openni_kinect package supplied camera video data and calibration parameters in the ROS message format.

4.2.1 Organization

In format, the test program is quite simple. The user defines the desired interest point detector, descriptor extractor, and feature matcher in an XML configuration file. Parameters for the selected interest point algorithms and additional configuration options are also specified in this file.

When the program is launched, it parses the XML file and initializes itself accordingly. A subscription service is established which listens for ROS messages containing images from the Kinect's RGB camera. Each time an image is published, a callback function is called. This callback function is responsible for preparing the image for processing, performing all feature detection, extraction, and matching operations, and displaying the results.

4.2.2 Preprocessing

First the image is prepared for feature detection. The RGB channels are converted to the single greyscale channel required by most detection algorithms. It is then downsampled to satisfy run time requirements specified by the user. Additional filtering is performed with a Gaussian filter for smoothing and with a median filter to remove salt and pepper noise.

4.2.3 Detection

The altered image is then passed to the interest point detection function. This function identifies interest points using the prescribed method and returns a length m vector of OpenCV Keypoints $\mathbf{k}$. Each keypoint $\mathbf{k}_i$ typically consists of the location (x, y) and scale σ that the feature was detected at. Depending on the detector used, it may also include an orientation θ and intensity L.

$$\mathbf{k} = \begin{bmatrix} \mathbf{k}_1 \\ \mathbf{k}_2 \\ \vdots \\ \mathbf{k}_m \end{bmatrix} = \begin{bmatrix} (x_1, y_1, \sigma_1, \{\theta_1, L_1\}) \\ (x_2, y_2, \sigma_2, \{\theta_1, L_2\}) \\ \vdots \\ (x_m, y_m, \sigma_m, \{\theta_m, L_m\}) \end{bmatrix}$$

4.2.4 Description

The altered image and vector of keypoints $\mathbf{k}$ are then passed to the descriptor extraction function. This function returns an $m \times n$ descriptor matrix where the i-th row is the length n descriptor for the interest point located at index i in the keypoint vector. The original image, keypoint vector, and descriptor matrix are then stored for future comparison.

The format of the descriptor depends on the extraction method. Binary extractors such as ORB, BRISK, and FREAK produce bit string descriptors of length $n = 512$. The more complex extractors such as those for SIFT and SURF produce descriptors composed of real numbers. These descriptors are "shorter," with a typical length of $n = 128$, and are normalized to have unit magnitude. The length 128 descriptors are actually larger than the binary descriptors since each real number is represented using 32 bits.

4.2.5 Matching

The feature sets for two frames of Kinect data are compared using the matching function. This matching function takes the two sets of descriptors, a *query* set and a *training* set, and identifies matching descriptors in the training set for each descriptor in the query set.

The matching algorithm used in this function is specified by the user as well as k, the number of matches to find in the training set for each query descriptor. For matching of SIFT or SURF descriptors, a FLANN or brute-force matcher can be used. For matching of binary descriptors, versions using Hamming distance measures are used. These are the brute-force-Hamming matcher, the brute-force-SSE3 matcher, and the FLANN-LSH matcher.

These matchers return k `DMatch`'s for each query descriptor. A `DMatch` is a datatype used to indicate correspondence between two descriptors. It consists of a pair of indices identifying the location of the descriptors in the query set and training set and a float indicating the distance between the two descriptors. This set of `DMatch`'s is then filtered in one of the manners described in Section 2.6. Matches where the distance or ratio of distances between descriptors falls below a specified threshold are kept and all others are discarded.

4.2.6 Outputs for Evaluation

The test program has a number of outputs for evaluation of interest points. Every time an image is processed, the number of feature points detected is reported, as well as the time required to detect those interest points and the time needed to compute their descriptors.

When a match between images is performed, statistics are computed for the percentage of features that have been matched and the mean distance between matching descriptors. Matching time is also reported. The matched images are displayed in a composite image showing the images side by side with detected features and correspondence lines drawn on top. An example of this output is shown in Figure 4.1.

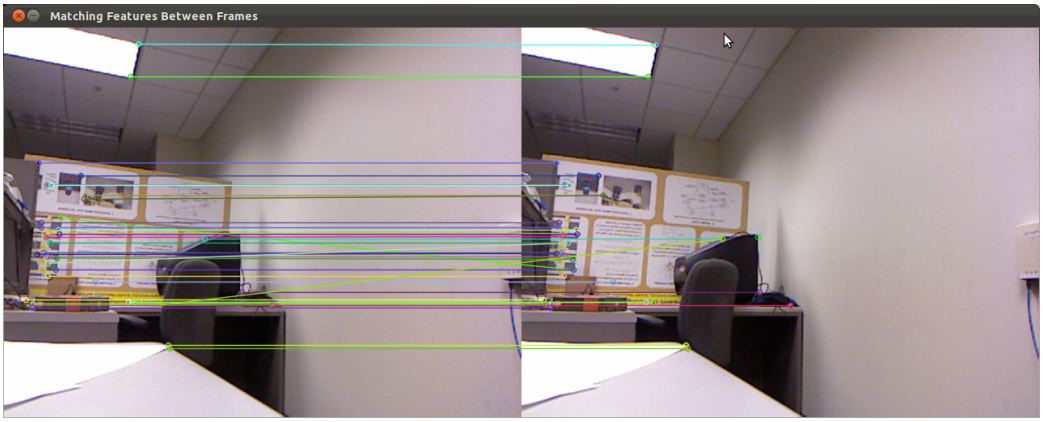


Figure 4.1: Feature matching output from the test program.

4.3 BRISK-AGAST Detection

This section details the BRISK-AGAST interest point detector implemented in OpenCV. This detector is used to identify interest points for both the BRISK and FREAK descriptors. This section will examine the adjustable parameters of the detector: the threshold value t and the number of octaves o. The threshold parameter is used in the identification of candidate interest points, and the octaves parameter is used in the creation of the scale pyramid.

The effects of these parameters on the interest point detection process will be detailed. How changes in these parameters affect the quantity and location of features will also be examined. This latter analysis is done to justify the development of a BRISK-AGAST detector called the Adjustable BRISK-AGAST detector which is able to adjust these parameters automatically in order to maintain a constant number of features per image.

4.3.1 Basics

A description of the BRISK-AGAST feature detection process is given in Section 2.6. A quick overview of the salient details will now be given.

The BRISK-AGAST detector searches for features across a scale pyramid, sometimes called a scale stack. A scale pyramid is a set of images composed of the original image as well as downsampled and smoothed versions of that image.

The octave parameter o determines the number of octave levels in the scale pyramid. Each octave level is a smoothed and downsampled version of the

level proceeding it. The scale pyramid is made up of $o + 1$ octave levels c_i and $o - 1$ intra-octave levels d_i, resulting in a total of $2o$ levels. The scale σ of each level is twice that of the previous corresponding level.

$$\sigma(c_i) = 2^i \quad \sigma(d_i) = 1.5(2^i)$$

Each level is created by down-sampling the previous corresponding level by 2, resulting in the doubling of scale. Creation of c_0 and d_0 is a special case. If o is zero, the scale pyramid is made up of a single octave level c_0 containing the original image. If o is 1, only the first intra-octave level d_0 is added. This level is obtained by downsampling c_0 by 1.5. Typically, an increase in o by one results in the addition of an octave level followed by an intra-octave level. A typical scale pyramid has between one and eight levels.

By examining each level of the pyramid, the detector can find features of different sizes using the same feature test. The BRISK detector uses the FAST-9-16 accelerated segment test (AST) as its feature test. In FAST-9-16, interest points are found by comparing the intensity of a center pixel to those on a 16 pixel circle located at a radius of 3 pixels about the center. If more than 9 connected pixels on the circle are all significantly brighter or darker than the center pixel, then the center is taken as a candidate point.

Non-maximal suppression (NMS) is used to eliminate neighboring candidates, yielding a set of finalized interest points which have the maximum FAST score of their neighbors in space and scale. The location of these points is then refined in scale and space using quadratic models and interpolation. A flowchart depicting the detection process is shown in Figure 4.2.

4.3.2 Threshold

The threshold t is used during the selection of candidate features and the generation of FAST scores for non-maximal suppression.

During candidate feature selection, the pixels on a Bresenham circle B of radius 3 about a candidate pixel $\mathbf{c}$ are examined for every pixel $\mathbf{c}$ in the scale pyramid. Each pixel $\mathbf{b}_{c,i}$ on the circle about $\mathbf{c}$ is assigned to one of three sets, S_{dark}, S_{bright}, or S_{unkown}, based on its difference in intensity with $\mathbf{c}$. If the difference in intensity is greater than t the pixel $\mathbf{b}_{c,i}$ is assigned to S_{bright}, if the difference is less than $-t$ the pixel is assigned to S_{dark}, and

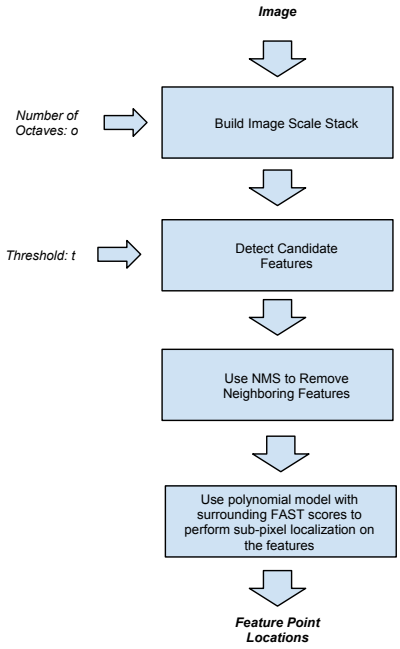


Figure 4.2: A flowchart detailing BRISK-AGAST interest point detection.

if the difference is between $-t$ and t the pixel is assigned to S_{unkown}. The center pixel $\mathbf{c}$ is labeled as a candidate interest point if there are at least 9 connected pixels in S_{bright} or at least 9 in S_{dark}.

Figure 4.3 shows an example application of the FAST-9-16 test. Here, pixels lying on the Bresenham circle B for $\mathbf{c}$ are indicated with red boxes. The dashed line indicates a connected segment of pixels which all belong to S_{bright}. Since there are nine pixels in the connected segment, $\mathbf{c}$ would be chosen as a candidate interest point.

$$B = \{\mathbf{b}_{c,1} \dots \mathbf{b}_{c,16}\}$$

Equation (2.25) describes the set assignment process for pixels lying on the Bresenham circle. In this equation, the intensity of a pixel $\mathbf{c}$ is denoted by $L(\mathbf{c})$. The threshold t is also used in calculation of the FAST score $s(\mathbf{c})$ for a candidate point $\mathbf{c}$ in Equation (2.24).

The threshold t can take integer values between 1 and 255 for an 8-bit greyscale image, but typical values range from 2 to 100, depending on the scene and desired number of features. Decreasing t results in a larger number of pixels being placed in S_{bright} and S_{dark} and a corresponding increase in the

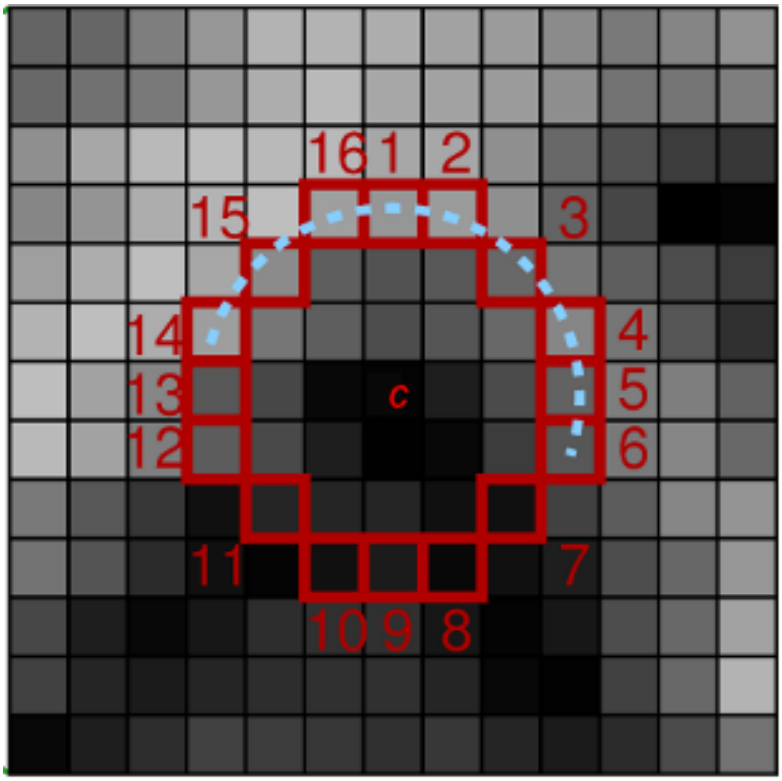


Figure 4.3: A 16 pixel Bresenham Circle used for FAST-9-16 Score Calculation. Image source: Edward Rosten.

number of candidate interest points.

Such a decrease alters all FAST scores a corresponding amount, essentially introducing an offset to all the candidate interest points' scores. The altered scores have no net effect on the final detection results. The offset introduces no new maxima, meaning the same interest points will be selected from the same set of candidates by the Non-Maxima Suppression (NMS) algorithm.

Note that it is possible for some previously detected feature points to be replaced by new neighboring candidates, if these new candidates had high FAST scores in the previous configuration, but did not have the required number of connected pixels in a single set to be taken as candidate interest points. In such cases, the decrease in t adds extra points to S_{bright} and S_{dark}, allowing the new candidates to pass the nine connected pixel test.

4.3.2.1 Change in Point Location with Respect to t

The location of previously detected interest points will not change with respect to t. This will be shown by discussing each step of the localization process for a single interest point, and showing that the relevant values remain the same. First the interest point localization process will be detailed, then it will be shown that the localization solution does not change with respect to t.

For the purposes of this explanation, consider an interest point with an initial location in the scale pyramid of (x, y, σ). Three 3×3 scores patches are given, containing the FAST scores for the interest point and the 26 pixels surrounding it in the scale pyramid.

These score patches are organized by octave level, with the first patch at the octave level below the interest point, the second patch at the octave level of the interest point, and the third patch at the octave level above the interest point. Each 3×3 patch is created from the nine pixels neighboring the interest point at that octave level. The score patch at the octave level of the interest point is created from the interest point and its eight neighbors on that octave level.

The value at each pixel in the patch is given by that pixel's FAST score. The FAST scores making up these patches are denoted $s_{(i,j)}$, where i and j indicate the position of the scored pixel relative to the interest point in space. For instance, if the interest point was located at $(2, 2, 1)$, then the

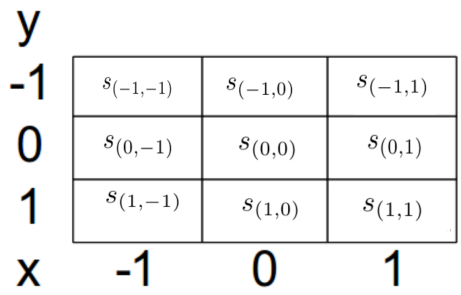


Figure 4.4: A BRISK 3×3 Score Patch.

score $s_{(-1,0)}$ at the octave level corresponding to a scale of one would be located at $(1, 2, 1)$. Figure 4.4 shows an example score patch. If this score patch were located at the interest point's octave level, $s_{(0,0)}$ would be the FAST score of the interest point.

Positional refinement of the interest point is performed using the refinements terms $(\Delta x, \Delta y, \Delta\sigma)$. The final location of the interest point is given in Equation (4.1).

$$(\bar{x}, \bar{y}, \bar{\sigma}) = (x + \Delta x, y + \Delta y, \sigma\Delta\sigma) \tag{4.1}$$

The refinement terms for the interest point are found in 3 steps.

1. 2-D quadratics are fit to the 3×3 score patches. These quadratics are used to determine the value of the maximum score in each patch and its location in sub-pixel coordinates. These three maximum scores are denoted as $s'_{(-1)}$, $s'_{(0)}$, and $s'_{(1)}$, where a subscript of -1 corresponds to the octave level below the interest point and the subscript of 1 indicates the octave level above the interest point. The locations of these three maximum scores in sub-pixel coordinates is given by $\Delta\ \mathbf{x}'_{(-1)}$, $\Delta\ \mathbf{x}'_{(0)}$, and $\Delta\ \mathbf{x}'_{(+1)}$ respectively.

$$\mathbf{x}'_{(-1)} = (\Delta x'_{(-1)}\ ,\ \Delta y'_{-1})$$

2. The maximum scores for each of the three levels are then fit to a parabola to estimate overall maximum score. The location in scale of this maximum score gives the scale refinement term $\Delta\sigma$.

3. The positional refinement terms Δx and Δy are then found by interpolating between the two maximum score locations found in Step 1 which

surround the scale refinement term found in Step 2.

In Step 1, the maximum score s' and its location $\mathbf{x}'$ are found for each of the three score patches. This is accomplished by fitting the parameters of the 2-D quadratic function given in Equation (4.2) to the score patch and finding the value and location of the quadratic's local maximum.

$$s = ai^2 + bij + cj^2 + di + ej + f \tag{4.2}$$

The parameters $\boldsymbol{\zeta}$ of the 2-D quadratic are found by performing a least squares fit. The 2-D quadratic is written in matrix form as:

$$s = \boldsymbol{\zeta}^T q(i,j)$$

where:

$$\boldsymbol{\zeta} = \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f \end{bmatrix} \qquad q(i,j) = \begin{bmatrix} i^2 \\ ij \\ j^2 \\ i \\ j \\ 1 \end{bmatrix}$$

The least squares fit can then be expressed as a minimization of the energy function $E(\boldsymbol{\zeta})$ given in Equation (4.3).

$$E(\boldsymbol{\zeta}) = \sum_{i=-1}^{1} \sum_{j=-1}^{1} (\boldsymbol{\zeta}^T q(i,j) - s_{i,j})^2 \tag{4.3}$$

Setting the derivative with respect to $\boldsymbol{\zeta}$ of the energy function equal to zero and solving for $\boldsymbol{\zeta}$ yields Equation (4.4).

$$\boldsymbol{\zeta} = \mathbf{M}^{-1} \mathcal{Q} \mathbf{s} \tag{4.4}$$

$$\mathbf{M} = \sum_{i=-1}^{1} \sum_{j=-1}^{1} q(i,j) q(i,j)^T$$

$$\mathcal{Q} = \begin{bmatrix} q(-1,-1) & q(0,-1) & \dots & q(1,1) \end{bmatrix}$$

$$\mathbf{s} = \begin{bmatrix} s_{-1,-1} \\ s_{0,-1} \\ s_{1,-1} \\ \vdots \\ s_{1,1} \end{bmatrix}$$

The local maximum for Equation (4.2) is found at the location $\Delta\mathbf{x}'$ where the partial derivatives with respect to i and j are 0, assuming the determinant of Hessian $H(s)$ passes the second derivative test stated in Equation (4.5).

$$H(s) = \begin{bmatrix} \dfrac{\partial^2 s}{\partial^2 i} & \dfrac{\partial^2 s}{\partial i \partial j} \\ \dfrac{\partial^2 s}{\partial i \partial j} & \dfrac{\partial^2 s}{\partial^2 j} \end{bmatrix} \qquad \det\left(H(s)\right) = \frac{\partial^2 s}{\partial^2 i}\frac{\partial^2 s}{\partial^2 j} - \frac{\partial^2 s}{\partial i \partial j}\frac{\partial^2 s}{\partial j \partial i} < 0$$

$$\frac{\partial s}{\partial i} = 2ai + bj + d \qquad \frac{\partial s}{\partial j} = 2cj + bi + e \tag{4.5}$$

Setting the partial derivatives to 0 and solving for i and j gives the location of the local maximum $\Delta\mathbf{x}'$, given in Equation (4.6).

$$\Delta\mathbf{x}' = \begin{bmatrix} \Delta x' \\ \Delta y' \end{bmatrix}$$

$$\Delta x' = \frac{be - 2cd}{4ac - b^2} \qquad \Delta y' = \frac{bd - 2ae}{4ac - b^2} \tag{4.6}$$

The maximum score is then given by Equation (4.7), concluding Step 1.

$$\begin{aligned} s' &= \boldsymbol{\zeta}^T q(\Delta x', \Delta y') \\ &= a(\Delta x')^2 + b(\Delta x')(\Delta y') + c(\Delta y')^2 + d(\Delta x') + e(\Delta y') + f \end{aligned} \tag{4.7}$$

In Step 2, the scale refinement term $\Delta\sigma$ is found by fitting the 1-D parabola specified in Equation (4.8) to the maximum scores $\Delta\mathbf{x}'_{-1}$, $\Delta\mathbf{x}'_0$, and $\Delta\mathbf{x}'_1$

found in Step 1 and their associated scale values. Scale values of $\hat{\sigma}_{-1} = 0.75$, $\hat{\sigma}_0 = 1$, and $\hat{\sigma}_1 = 1.5$ are used rather than the actual scale values of the score patches, giving a multiplicative refinement term $\Delta\sigma$ rather than the refined scale $\bar{\sigma}$ of the interest point itself. Posing the problem in such a manner allows for time saving pre-computations in the BRISK implementation.

$$s = \alpha\hat{\sigma}^2 + \beta\hat{\sigma} + \gamma \tag{4.8}$$

The least-squares fit of the parabola to the supplied scales and scores can be found in the same manner as the 2-D fit in Step 1. Writing Equation (4.8) in matrix form gives Equation (4.9) and its least squares solution is given in Equation (4.10).

$$s = \boldsymbol{\zeta}_{1D} \begin{bmatrix} \hat{\sigma}^2 \\ \hat{\sigma} \\ 1 \end{bmatrix} \tag{4.9}$$

$$\boldsymbol{\zeta}_{1D} = \mathbf{M}_{1D}^{-1}\mathcal{Q}_{1D}\mathbf{s}_{1D} \tag{4.10}$$

$$\mathbf{M}_{1D} = \sum_{i=-1}^{1} \begin{bmatrix} \hat{\sigma}_i^4 & \hat{\sigma}_i^3 & \hat{\sigma}_i^2 \\ \hat{\sigma}_i^3 & \hat{\sigma}_i^2 & \hat{\sigma}_i \\ \hat{\sigma}_i^2 & \hat{\sigma}_i & 1 \end{bmatrix} \qquad \mathcal{Q}_{1D} = \begin{bmatrix} \hat{\sigma}_{-1}^2 & \hat{\sigma}_0^2 & \hat{\sigma}_1^2 \\ \hat{\sigma}_{-1} & \hat{\sigma}_0 & \hat{\sigma}_1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$\mathbf{s}_{1D} = \begin{bmatrix} s_{-1} \\ s_0 \\ s_1 \end{bmatrix} \qquad \boldsymbol{\zeta}_{1D} = \begin{bmatrix} \alpha \\ \beta \\ \gamma \end{bmatrix}$$

The scale $\Delta\sigma$ of the maximum score is found by setting the derivative of Equation (4.8) equal to zero and solving for $\hat{\sigma}$. The value of $\Delta\sigma$ in terms of the parabola parameters is given in Equation (4.11), concluding Step 2.

$$\Delta\sigma = \frac{-\beta}{2\alpha} \tag{4.11}$$

In Step 3, the spatial refinement terms Δx and Δy are found using linear interpolation. The locations of the critical points for the score patches above and below $\Delta\sigma$ are used to interpolate the value of Δx and Δy at $\Delta\sigma$. This interpolation operation is detailed in Equation (4.12). In this equation $\Delta\mathbf{x}'_{(+)}$ is the location of the critical point found in Step 1 whose scale is closest to,

but greater than $\Delta\sigma$, and $\Delta\mathbf{x}'_{(-)}$ is the critical point location whose scale is closest to, but less than $\Delta\sigma$.

$$\begin{bmatrix}\Delta\mathbf{x}\\ \Delta\sigma\end{bmatrix} = \begin{bmatrix}(1-u)\mathbf{x}'_{(+)} + u\mathbf{x}'_{(-)}\\ \Delta\sigma\end{bmatrix} \tag{4.12}$$

$$u = \frac{\Delta\sigma - \hat{\sigma}_{(-)}}{\hat{\sigma}_{(+)} - \hat{\sigma}_{(-)}}$$

$$\hat{\sigma}_{(+)} > \Delta\sigma > \hat{\sigma}_{(-)}$$

This concludes the description of the interest point localization process. It will now be shown that changes in the threshold t by an amount Δt do not change the location of the interest point.

When the threshold t is changed by Δt, the FAST scores are globally offset by Δt. This fact should be intuitive given an examination of the FAST score equation shown in Equation (2.24). These new FAST scores will be denoted $\tilde{s}$. The relationship between the new and old FAST scores of the three score patches used in the interest point location refinement process is given in Equation (4.13).

$$\tilde{s}_{i,j} = s_{i,j} + \Delta t \tag{4.13}$$

This change in scores has no effect on the model fit performed in Step 1 except for the parameter f. Applying this change in scores to Equation (4.4) yields a new parameter vector $\tilde{\boldsymbol{\zeta}}$ given in Equation (4.14).

$$\begin{aligned}\tilde{\boldsymbol{\zeta}} &= \mathbf{M}^{-1}\mathcal{Q}\tilde{\mathbf{s}} = \mathbf{M}^{-1}\mathcal{Q}(\mathbf{s} + \Delta t\mathbf{1})\\ &= \mathbf{M}^{-1}\mathcal{Q}\mathbf{s} + \Delta t\mathbf{M}^{-1}\mathcal{Q}\mathbf{1}\end{aligned} \tag{4.14}$$

It can be shown through direct computation that the first eight rows of the matrix $\mathbf{M}^{-1}\mathcal{Q}$ sum to zero and the ninth row sums to one, leading to the simplification:

$$\Delta t\left(\mathbf{M}^{-1}\ \mathcal{Q}\ \mathbf{1}_{9\times 1}\right) = \begin{bmatrix}0\\ \vdots\\ 0\\ \Delta t\end{bmatrix}$$

Substituting this simplification into Equation (4.14) gives the new param-

eter matrix $\tilde{\boldsymbol{\zeta}}$, specified in Equation (4.15).

$$\tilde{\boldsymbol{\zeta}} = \mathbf{M}^{-1}\mathcal{Q}\mathbf{s} + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \Delta t \end{bmatrix} = \boldsymbol{\zeta} + \begin{bmatrix} 0 \\ \vdots \\ 0 \\ \Delta t \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \\ d \\ e \\ f + \Delta t \end{bmatrix} \tag{4.15}$$

We can conclude that the location $\Delta\mathbf{x}'$ of the 2-D quadratic's critical point does not change in the presence of a global offset, since the refinement locations terms $\Delta\tilde{x}'$ and $\Delta\tilde{y}'$ obtained by plugging $\tilde{\boldsymbol{\zeta}}$ into Equation (4.6) are equal to $\Delta x'$ and $\Delta y'$.

$$\Delta\tilde{x}' = \frac{\tilde{b}\tilde{e} - 2\tilde{c}\tilde{d}}{4\tilde{a}\tilde{c} - \tilde{b}^2} = \frac{be - 2cd}{4ac - b^2} = \Delta x'$$
$$\Delta\tilde{y}' = \frac{\tilde{b}\tilde{d} - 2\tilde{a}\tilde{e}}{4\tilde{a}\tilde{c} - \tilde{b}^2} = \frac{bd - 2ae}{4ac - b^2} = \Delta y'$$

The maximum score $\tilde{s}'$ located at $(\Delta\tilde{x}', \Delta\tilde{y}')$ is s' offset by Δt.

$$\begin{aligned} \tilde{s}' &= a(\Delta x')^2 + b(\Delta x')(\Delta y') + c(\Delta y')^2 + d(\Delta x') + e(\Delta y') + (f + \Delta t) \\ &= s' + \Delta t \end{aligned}$$

In Step 2, the increase in all of the score maxima obtained from Step 1 does not change the location of the scale refinement term $\Delta\tilde{\sigma}$ for similar reasons.

The parameter values $\tilde{\boldsymbol{\zeta}}_{1D}$ change in a similar manner to the $\tilde{\boldsymbol{\zeta}}$ terms. Substituting $\tilde{\mathbf{s}}_{1D}$ into Equation (4.10) yields Equation (4.16).

$$\begin{aligned} \tilde{\mathbf{s}}_{1D} &= \mathbf{s}_{1D} + \Delta t\ \mathbf{1}_{(3\times 1)} \\ \tilde{\boldsymbol{\zeta}}_{1D} &= \mathbf{M}_{1D}^{-1}\mathcal{Q}_{1D}\tilde{\mathbf{s}}_{1D} = \mathbf{M}_{1D}^{-1}\mathcal{Q}_{1D}\mathbf{s}_{1D} + \Delta t\ \mathbf{M}_{1D}^{-1}\ \mathcal{Q}_{1D}\ \mathbf{1}_{(3\times 1)} \\ &= \boldsymbol{\zeta}_{1D} + \begin{bmatrix} 0 \\ 0 \\ \Delta t \end{bmatrix} = \begin{bmatrix} \alpha \\ \beta \\ \gamma + \Delta t \end{bmatrix} \end{aligned} \tag{4.16}$$

Plugging $\tilde{\boldsymbol{\zeta}}_{1D}$ into Equation (4.11) shows that the spatial refinement terms

$\Delta\tilde{\sigma}$ and $\Delta\sigma$ are equivalent.

$$\Delta\tilde{\sigma} = \frac{-\tilde{\beta}}{2\tilde{\alpha}} = \frac{-\beta}{2\alpha} = \Delta\sigma$$

Since the location of the critical points $\Delta\mathbf{x}'_{(-1)}$, $\Delta\mathbf{x}'_{(0)}$, $\Delta\mathbf{x}'_{(1)}$, and $\Delta\sigma$ do not change when t is changed, the final interpolated location refinement obtained in Step 3 $(\Delta x, \Delta y, \Delta\sigma)$ remains the same.

4.3.2.2 Number of Features Detected

The number of features detected varies significantly with changes in t. Table 4.1 shows the results of setting t to 10, 15, and 20 at several different o values. As can be seen in columns 2, 3, and 4, decreasing t significantly increases the number of features detected and increases the total detection time per image. This is the result of an increase in the number of candidate features which need to be evaluated, and finalized features which must be localized. The detection time per feature remains approximately constant.

Table 4.1: BRISK-AGAST detection results with a BRISK Extractor and brute-force-Hamming matcher.

Octave	Threshold	Feature Count	Detect Time (ms)	Extract Time (ms)	Match Time (ms)	Detect Time / Feat. (ms)
0	10	1854.661	15.468	9.532	179.352	0.008
0	15	1308.360	11.112	7.032	92.484	0.008
0	20	968.099	8.549	5.296	52.340	0.009
1	10	728.261	12.657	4.198	27.272	0.017
1	15	585.079	11.212	3.444	17.767	0.019
1	20	477.365	9.154	2.577	11.768	0.019
2	10	628.538	14.315	3.679	20.431	0.023
2	15	503.082	11.904	2.977	13.188	0.024
2	20	416.191	9.984	2.315	9.186	0.024
3	10	607.029	15.269	3.530	18.972	0.025
3	15	484.509	12.669	2.871	12.386	0.026
3	20	400.750	10.829	2.365	8.513	0.027
4	10	602.046	15.462	3.489	18.600	0.026
4	15	480.162	12.675	2.759	12.052	0.026
4	20	396.635	11.038	2.346	8.345	0.028

4.3.3 Octave

The octave parameter o determines the number of octave levels in the scale pyramid used for interest point detection. There is a total of $2o$ levels in the pyramid. For a more detailed description of scale pyramid construction refer to Section 2.6.

4.3.3.1 The Number of Octaves and Feature Detection

The scale pyramid plays a role in the detection of candidate interest points, and the selection and localization of finalized interest points.

The pyramid makes up the search space for candidate interest points. Candidates are found by examining each level for points which satisfy the FAST 9-16 criterion. Candidate features are then compared to the 9×9 patch at their scale level, the level above, and the level below to determine if they satisfy the maximum criterion to become finalized interest points. The positions of these interest points are then refined using the same three levels.

Intuitively, it appears that increasing o, and thereby the search space, would increase the number of features detected, but this is not the case in practice. Some peculiarities of the NMS algorithm used to select finalized interest points result in a steady decrease in the number of detected points as the pyramid grows larger.

For a scale pyramid of l levels, the NMS algorithm behaves as expected for the first $l-1$ levels. It examines the three levels surrounding each candidate point and determines whether the candidate has a greater score than all its neighbors. However, the behavior of the NMS algorithm changes at level l. Since there are no levels above l, the NMS algorithm evaluating candidates on l will only examine levels l and $l-1$. As a result, many interest points are finalized which would not be selected if $l+1$ were available. When o is increased, regular NMS is applied to level l, and these points are eliminated. The decrease in the quantity of previously detected points coupled with the ever decreasing size of each level leads to a steady decline in the total number of features detected with each octave level added.

This behavior is exemplified in Table 4.1. Increasing the value of o from zero to one results in significant decrease in the number of features detected. With a value of $o = 0$, the scale pyramid has one level and NMS considers only the 8 surrounding scores. When o is increased to one, intra-octave levels d_0 and d_{-1} are added and considered in the NMS, resulting in 400-900 fewer interest points. Subsequent increases result in a slight decrease of 5-10 features per octave level. Each additional level eliminates some interest points that are not truly maxima and replaces them with fewer new points.

Increasing the number of octaves from $o = \frac{l}{2}$ to $o = \frac{l}{2} + 1$ has no effect on the detection or localization of interest points not detected on level l. Since

the lower levels are unchanged by the addition of new levels, points detected and localized using the lower levels will remain unchanged as well.

It is almost certain that interest points found at level l which are not eliminated by the stricter NMS check will change in location. Points located in level l of the scale pyramid will have their scale refined from $\sigma = 1.5(2^{\frac{l-1}{2}})$ to $\sigma = 1.5(2^{\frac{l-1}{2}})(\Delta\sigma)$, with $\Delta\sigma$ as defined in Equation (4.11). This in turn will result in new interpolated positional refinement terms Δx and Δy, and a completely different interest point location.

Increasing the number of octaves results in an initial decrease in detection time per image, followed by subsequent increases. The initial decrease is due to a substantial decrease in the number of features detected. Hundreds fewer detected interest points means hundreds fewer localization operations must be performed. As o continues to increase, the feature decrease slows to a few features per level, while the detection time climbs by roughly 1 ms per level as a result of the increased size of the search space. The increased space means more pixels must be examined to determine whether they are candidates and more candidates must be evaluated to determine if they are maxima. The increase in detection time is so significant that, at $o = 3$, the detector takes as much time as $o = 0$ while detecting one-third of the interest points.

The detection time per feature (Column 7 of Table 4.1) paints a clearer picture of what is happening. Each increase in o results in a corresponding increase by approximately 8 μs in detection time for each interest point found.

4.4 Remarks on BRISK-AGAST and Motivation toward the Adjustable BRISK Detector

Experiments with the Kinect and BRISK-AGAST detector revealed that the number of features varies quite substantially throughout the course of a video sequence. Figure 4.5 shows the feature count during one such sequence. Four sample frames taken from this video sequence are shown in Figure 4.6. The count oscillates between low and high values dropping as low as 200 features per frame between frames 195 and 205, and then spiking up to almost 1800 features per frame near frame 230. This type of behavior is problematic in the RGB-D SLAM System. When too many features are present, the number

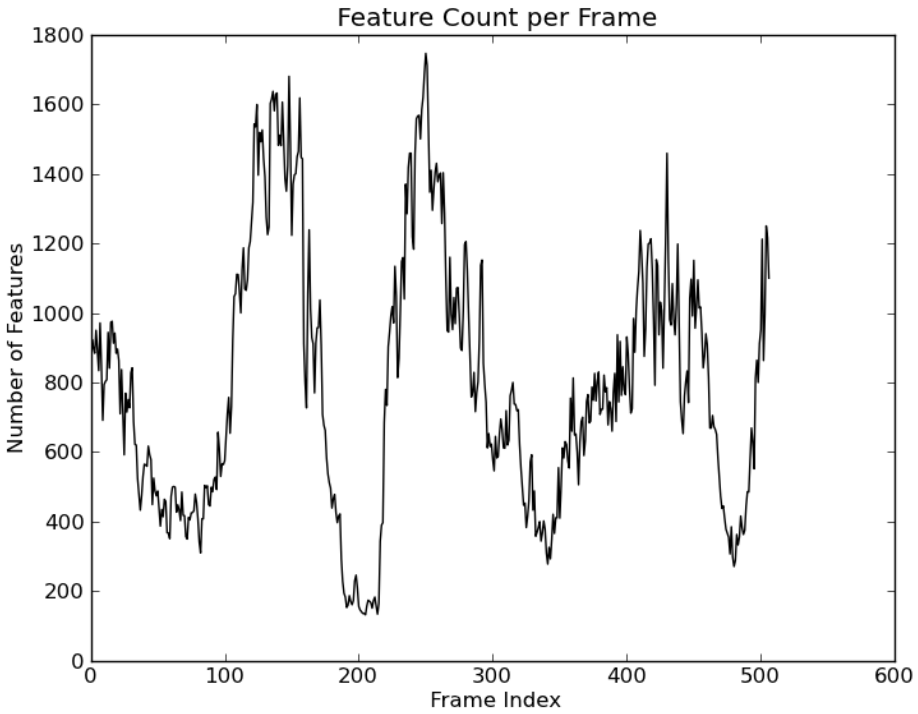
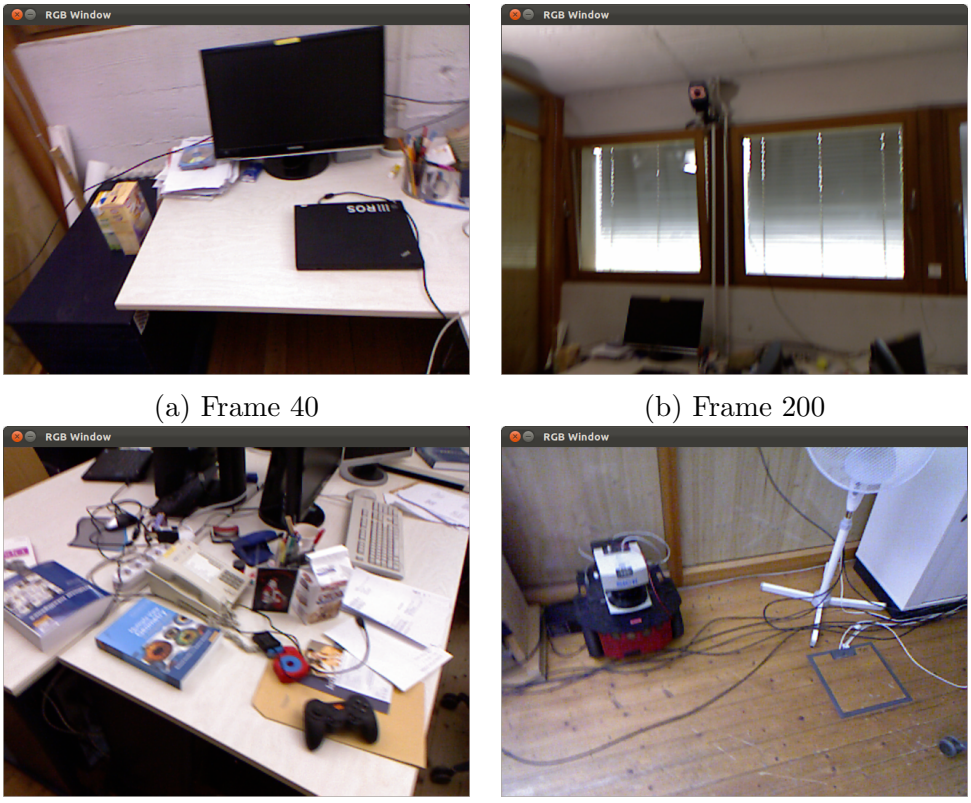


Figure 4.5: The feature count in a video sequence using a BRISK-AGAST detector with $t = 15$ and $o = 0$.

of matches used in the computation of visual odometry becomes large and the processing rate for new frames decreases. If too few are present, there may not be enough feature matches with any keyframe to recover the motion between frames at all. In the RGB-D SLAM System it is desirable to have a detector that produces a consistent number of features, no matter the current view so as to avoid such problems.

OpenCV provides a framework for such a detector in the DynamicAdaptedFeatureDetector class. This class takes a feature detector, an image, and bounds on the desired number of features, and iteratively tunes the feature detector so as to produce a feature count within the specified bounds. Unfortunately, this detector was designed for use in the processing of image sets rather than video sequences. As such it assumes no correlation between the content of sequential frames and is not designed with efficient detection times in mind.

The detector does not preserve its tuned parameter settings between images. Instead it performs an iterative refinement of these parameters for



(a) Frame 40

(b) Frame 200

(c) Frame 230

(d) Frame 500

Figure 4.6: Sample frames from the video sequence used to generate Figure 4.5.

every image. The refinement process repeatedly generates a new detector, performs feature detection on the image, and then adjusts the parameters of the next detector based on a comparison between the number of features detected and the desired bounds. This process repeats until an acceptable number of features is detected or the maximum number of iterations for the image is reached.

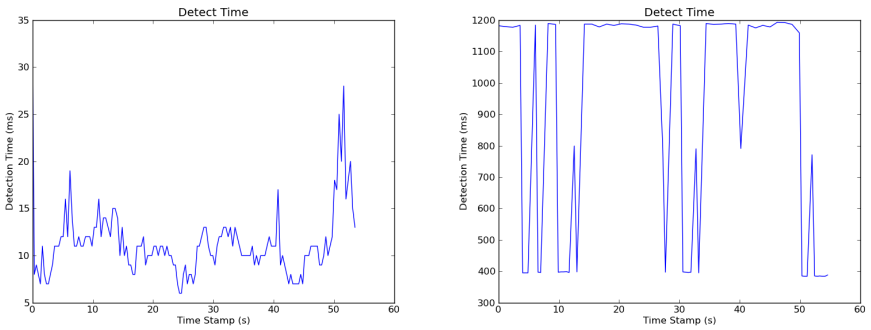
This process is very time-intensive for BRISK. A new detector must be instantiated at every iteration of the refinement– typically 3-5 times per image. With a creation time of 300 ms per BRISK detector and multiple detection passes on each image, any potential savings in matching time and improvement in accuracy due to a more consistent feature count are outweighed by the dropped frames and additional detection time.

Figure 4.7 illustrates this difference. In Figure 4.7a, the detection time per image is shown for a BRISK detector with $t = 15$, $o = 1$. A similar plot for a dynamic BRISK detector with a minimum feature count of 600, a maximum count of 800, and a maximum of 3 iterations is shown in 4.7b. The average detection time of 11 ms per frame for the BRISK detector is almost 1/80th of the 831 ms for the dynamic detector. Note the sharp jumps in detection time in 4.7b. These jumps occur when the number of iterations required to refine the parameters changes. Times near 400 ms are the result of a one iteration detection, times near 800 ms are the result of two, and those near 1200 ms are the result of three. Even at one iteration, the extra overhead involved with instantiating the dynamic detector makes detection take 40 times longer than the average BRISK detector.

A detector capable of finding a consistent number of features would be very useful in the RGB-D SLAM System, but the dynamic detectors provided by OpenCV are too slow for practical use.

4.5 Adjustable BRISK Detector

In response to experiments with the dynamic detector, I created the Adjustable BRISK detector. This interest point detector has detection times comparable to BRISK, while producing a more consistent number of features. It leverages the large amount of visual consistency between sequential frames in a video sequence to use previous detection results to inform the selection



(a) (b)

Figure 4.7: Figure 4.7a shows the detection time per frame for a BRISK detector with $t = 15$, $o = 1$. Figure 4.7b shows the detection time per frame for a dynamic BRISK detector with feature boundaries 600-800.

of future BRISK parameters.

The Adjustable BRISK detector is a modified version of the BRISK-AGAST detector. It selects interest points in the same manner, but has additional functions that allow for tuning of the threshold value to keep the number of features detected relatively constant.

Usage is similar to that of the BRISK detector. The user specifies six parameters in the initialization of the detector:

- `init_thresh` - The starting threshold value to use in the first detection.
- `nOctaves` - The number of octaves to use in the detection of feature points. This parameter is exactly the same as in the BRISK-AGAST detector.
- `min_features` - The lower-bound on the acceptable number of features.
- `max_features` - The upper-bound on the acceptable number of features.
- `min_thresh` - The lowest acceptable value that the threshold can take.
- `max_thresh` - The greatest acceptable value that the threshold can take.

The `detect` function is used to extract a list of interest points from an image. Immediately after a `detect` call, the `updateParams` function is called with the number of interest points detected, n, as a parameter. This function adjusts the detector parameters to produce more points if too few interest points were detected, and less if too many were detected. The threshold

update logic for `updateParams` is given in Equation (4.17), where t is the previous threshold and $\hat{t}$ is the new threshold value.

$$\hat{t} = \begin{cases} 0.9t & \texttt{min\_features} > n \\ 1.1t & \texttt{max\_features} < n \end{cases} \tag{4.17}$$

Although the number of octaves o and the threshold value t are both capable of changing the number of features, only t is adjusted in Equation (4.17). This is because t can take a much broader range of values, and those values give finer control of the feature count than o.

Figure 4.9 demonstrates this, showing feature detection counts for ranges of t and o on the famous "Lena" image (Figure 4.8). In Figure 4.9b, the feature count is shown for the range of all possible o values 0 - 9. As discussed in Section 4.3, increasing o from 0 to 1 results in a sharp decrease in the number of feature detected, from 1435 to 299 in this case. Subsequent increases in o result in decreases in the feature count of only 1-2 features.

Figure 4.9a shows the large set of values t can cover and the wide range of feature counts that it can produce. With a properly selected threshold value, most any range of feature counts can be achieved.

Note the detection efficiency in Figures 4.9e and 4.9f, given in μs per feature. For threshold values less than 80, an approach altering only the threshold is able to produce features more efficiently than one changing the number of octaves. It should also be mentioned that a change in the value of o used to detect features would necessitate the reinitialization of the descriptor extractor, so as to generate a new look-up table including the appropriate number of scale levels. The generation of the look-up table is a time-intensive process which requires several hundred milliseconds.

Concerns regarding the stability of the detected feature points also played a role in the choice to use only the threshold value to control the number of features. We saw in Section 4.3 that the location of an interest point is constant with respect to t. This property only extends to the lower levels of the scale pyramid for o. Interest points located at the top level will almost surely change location when o is increased, since the localization technique for those points expands to include scale refinement and inter scale level interpolation.

The change in the location of the features is an issue in the RGB-D SLAM System. When the algorithm processes a new frame, it performs visual odom-

Figure 4.8: Lena.

etry by comparing features between the new frame and existing key frames and then examining the rigid transformation between the reconciled 3-D locations of the matching pairs in the two frames.

The RGB-D SLAM System uses feature matches with previously detected frames to estimate the pose of the camera. If a change in o results in a change in the location of interest points, their descriptors might be altered enough to prevent matches from being established, resulting in a loss of tracking. Even if enough matches are found, the change in position of the interest points may alter their relative positions, resulting in an incorrect motion estimate that introduces error into the estimated trajectory.

A multiplicative threshold adjustment in Equation (4.17) is chosen over the more typical constant adjustment because of the nature of the feature count - threshold plot in Figure 4.9a. For large values of t, large changes in t are required to elicit a significant change in the feature count, while at smaller values of t very small changes in t are required to prevent gross changes in the feature count.

4.6 BRISK Descriptor

The BRISK descriptor is formed as described in Section 2.6. It is a binary descriptor formed using the sampling pattern in Figure 2.2 which has been rotated and expanded in accordance with the orientation and scale of the interest point being described.

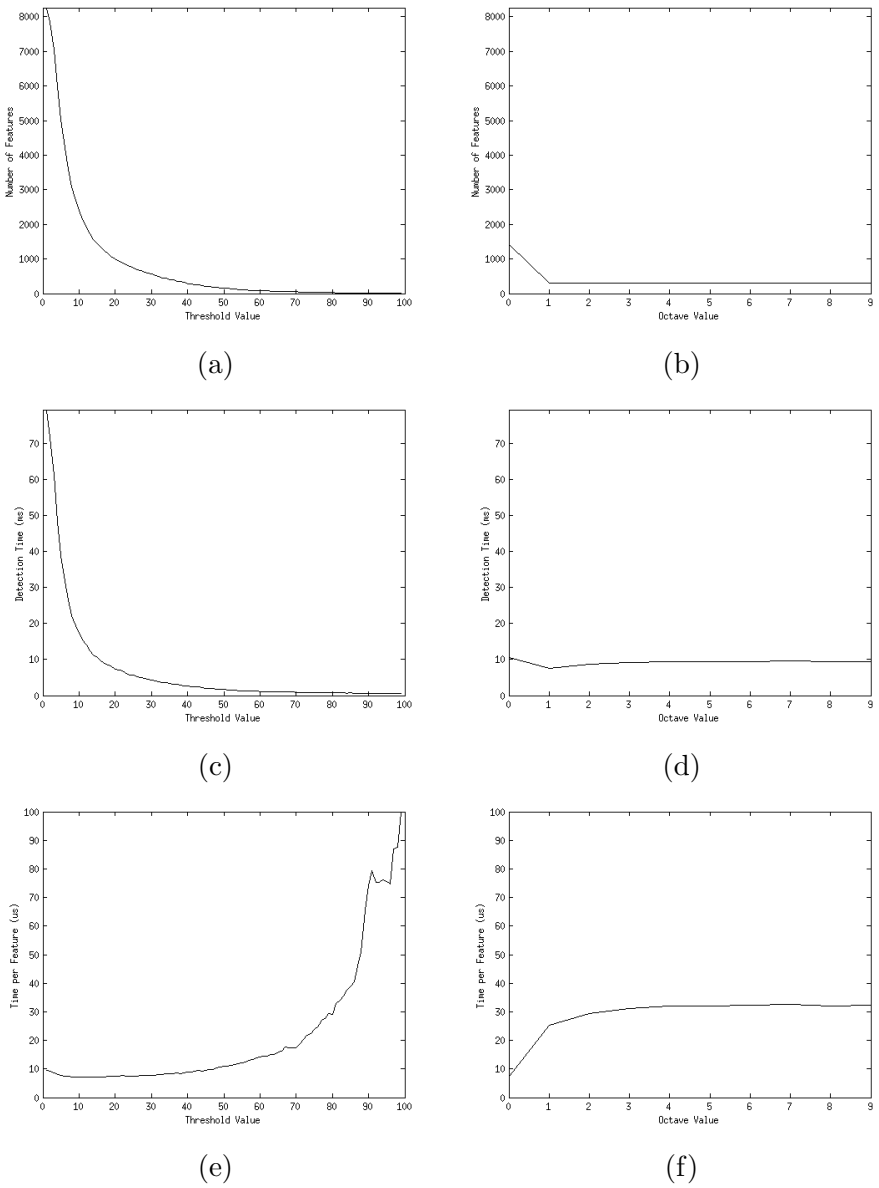


Figure 4.9: Detection results for the Lena image across a range of threshold values and octave counts. The first column shows the results for the BRISK-ÃGAST detector with $o = 0$ and $t = 1 \rightarrow 100$. The second column shows results for $t = 15$ and $o = 1 \rightarrow 9$. The first row gives the number of features detected, the second gives the total detection time at each setting, and the third gives the detection efficiency in μs per feature.

The implementation of the BRISK descriptor used in this thesis had two open parameters: `nOctaves` and `patternScale`. The parameter `nOctaves` is the number of octave levels used in detection of the interest point. This value is used in the generation of the look-up table that expedites descriptor creation. The parameter `patternScale` specifies the size of the sampling pattern used to generate the descriptor. Increasing the value of `patternScale` increases the scale of the sampling pattern, moving each sampling point farther from the interest point being described, and increasing the radius of the Gaussian kernel located at each sampling point.

In experiments with `patternScale`, it did not appear to have much of an impact on the quality of the matches, although keeping the value close to 1.0 produced the most consistent results.

4.7 FREAK Descriptor

The FREAK descriptor is described in Section 2.6. It is a binary descriptor similar in form to BRISK. Both extractors use a sampling pattern to compare image intensities and build the binary descriptor strings. Both also use the orientation and scale of the interest point to rotate and scale the sampling pattern, providing a measure of scale and rotational invariance. FREAK differs from BRISK in the pattern used to generate the descriptor and the method in which two descriptors are matched. FREAK's sampling pattern (Figure 2.3) is made up of fewer sampling points than BRISK. These sampling points have overlapping kernels with a much wider variety in kernel size, and are organized in a manner reminiscent of the human eye.

The FREAK descriptor incorporated into the RGB-D SLAM System had four open parameters: `orientationNormalized`, `scaleNormalized`, `patternScale`, and `nOctaves`.

- The parameter `orientationNormalized` is a Boolean variable which specifies whether to determine the orientation of each interest point and rotate the sampling pattern accordingly prior to description.

- The parameter `scaleNormalized` is a Boolean variable which specifies whether to expand or shrink the sampling pattern in accordance with the interest point's scale prior to description.

- The parameter `patternScale` is a positive floating point value. Exactly as in the BRISK descriptor, patternScale allows the user to expand or contract the sampling pattern for all interest points, so as to use more or less of the image in descriptor creation. The default value is 22.0, however cursory tests with the test program produced best results for the Kinect with values near 15.0.

- The parameter `nOctaves` is the number of octave levels used during the extraction of the interest point descriptor. This value is used in the generation of a look-up table that expedites the sampling and comparison step during descriptor construction.

4.8 Matchers

Brute-force matchers using Hamming and Euclidean distance measures, and FLANN matchers using Euclidean distance are included in the default installation of the RGB-D SLAM System. A FLANN matcher implementation using Locality-Sensitive Hashing (LSH) [13],[12] was added for this thesis. A FLANN-LSH matcher allows use of a Hamming distance metric in the FLANN framework, making it possible to use a FLANN matcher with binary descriptors such as BRISK and FREAK.

4.9 Incorporation into the RGB-D SLAM System

This section details changes made to the RGB-D SLAM System to facilitate testing of additional feature detectors, extractors, and matchers.

The RGB-D SLAM algorithm was modified to incorporate the BRISK-AGAST, Dynamic BRISK, and Adjustable BRISK detectors, as well as the BRISK and FREAK descriptor extractors, and the brute-force-SSE3 and FLANN-LSH matchers. Several additional changes were made to facilitate testing of these detectors, extractors, and matchers.

Cases for the three detectors were added to the `CreateDetector` function in the `misc.cpp` file. Cases for the FREAK and BRISK descriptors were also added to `CreateDescriptorExtractor` in the same file. Corresponding parameters for the initialization of the detectors and extractors were added to

the parameter server in the `parameter_server.cpp` file. An `updateParams` call was also added to the `detect` call in the `node.cpp` file, to be used only when the Adjustable BRISK detector is selected.

The brute-force-Hamming matcher in the `node.cpp` file was altered to allow use with any binary descriptor, not just ORB. Options for the brute-force SSE3 and FLANN-LSH matchers were added in the same location.

Finally, new timing code was added to monitor the performance of the new additions and a playback timer was incorporated to limit the duration of the RGB-D SLAM System's run time and provide more consistent video sequence lengths for comparison.

4.10 Evaluation of RGB-D SLAM Data

Quantitative evaluation of the feature detection, extraction, and matching functions in the RGB-D SLAM System is accomplished in this thesis through the comparison of the estimated camera trajectory to a set of ground truth poses.

4.10.1 Trajectory Evaluation

Two methods were developed for the comparison of the estimated trajectory obtained from the RGB-D SLAM System to the ground truth trajectory. The absolute trajectory error (ATE) [62] examines the difference in location and orientation between matching poses in the aligned trajectory estimate and the ground truth. The relative motion error (RME) [63] compares the relative motion between poses in the ground truth and trajectory estimate.

4.10.1.1 Goal and Notation

There are two collections of time stamped trajectories $\grave{D}$ and E. The sequence $\grave{D} = \{\grave{d}_1, \grave{d}_2, \ldots \grave{d}_i \ldots \grave{d}_n\}$ is the collection of ground-truth poses and timestamps, and $E = \{e_1, e_2, \ldots e_j \ldots e_m\}$ is the set of discrete pose estimates and their timestamps. The ground truth sequence describes the position and orientation of a camera throughout a video sequence. The estimated trajectory is the output of a SLAM algorithm which is estimating the position and

orientation of that camera throughout the same video sequence.

- The set of ground truth timestamped poses $\grave{D}$ is recorded at a rate of 100 Hz, while the set of estimated timestamped poses E is recorded asynchronously and much less frequently.
- Each ground truth timestamped pose $\grave{d}_i = (\grave{t}_{d,i}, \bar{\mathbf{J}}_i)$ consists of a timestamp $\grave{t}_{d,i}$ and a pose $\grave{\mathbf{J}}_i$, describing the rigid transformation from camera coordinate frame at time $\grave{t}_{d,i}$ to G, the ground truth's world frame.
- This transformation $\grave{\mathbf{J}}_i$ can be expressed as a translation vector $\grave{\boldsymbol{\rho}}_{d,i}$ and a quaternion $\grave{\mathrm{Q}}_{d,i}$.

$$\grave{\mathbf{J}}_i \Leftrightarrow \left\langle \grave{\mathrm{Q}}_{d,i}, \grave{\boldsymbol{\rho}}_{d,i} \right\rangle$$

- Each estimated timestamped pose $e_j = (t_{e,j}, \bar{\mathbf{T}}_j)$ gives the estimated rigid transformation $\bar{\mathbf{T}}_j$ from the camera coordinate frame at time $t_{e,j}$ to W, the estimate's world frame.
- This transformation $\bar{\mathbf{T}}_j$ can be expressed as a translation vector $\boldsymbol{\rho}_{e,j}$ and a quaternion $\mathrm{Q}_{e,j}$.

$$\bar{\mathbf{T}}_j \Leftrightarrow \left\langle \mathrm{Q}_{e,j}, \boldsymbol{\rho}_{e,j} \right\rangle$$

- The timestamps for both sets are synchronized, giving the time since Unix epoch.

4.10.1.2 Alignment in Time

Both ATE and RPE require alignment of the higher rate ground truth timestamped poses with the asynchronous trajectory estimate. This is achieved by interpolating a ground truth pose for every estimate pose, giving a set of interpolated pose and time estimates $D = \{d_1, d_2, \ldots d_i \ldots d_m\}$. The process, using linear interpolation on the translation terms and spherical linear interpolation (Slerp) [35] on the rotational terms, is as follows:
For each timestamped pose $e_j \in E$:

1. Find the two sequential timestamped poses $\grave{d}_i$ and $\grave{d}_{i+1}$ such that $\grave{t}_{d,i} < t_{e,j} < \grave{t}_{d,i+1}$.

2. Compute the interpolation weight u.

$$u = \frac{t_{e,j} - \grave{t}_{d,i}}{\grave{t}_{d,i+1} - \grave{t}_{d,i}}$$

3. Obtain $\boldsymbol{\rho}_{d,j}$ via linear interpolation.

$$\boldsymbol{\rho}_{d,j} = \grave{\boldsymbol{\rho}}_{d,i} + u(\grave{\boldsymbol{\rho}}_{d,i+1} - \grave{\boldsymbol{\rho}}_{d,i})$$

4. Obtain $\mathrm{Q}_{d,j}$ via Slerp.

$$\mathrm{Q}_{d,j} = \frac{\sin\left(\gamma(1-u)\right)}{\sin(\gamma)}\grave{\mathrm{Q}}_{d,i} + \frac{\sin\left(u\gamma\right)}{\sin(\gamma)}\grave{\mathrm{Q}}_{d,i+1}$$

$$\gamma = \arccos(\grave{\mathrm{Q}}_{d,i} \cdot \grave{\mathrm{Q}}_{d,i+1})$$

5. Construct $d_j = (t_{e,j}, \bar{\mathbf{J}}_j)$.

$$\bar{\mathbf{J}}_j \Leftrightarrow \left\langle \mathrm{Q}_{d,j}, \boldsymbol{\rho}_{d,j} \right\rangle$$

4.10.1.3 Absolute Trajectory Error

The absolute trajectory error is similar to the error metric used in [1]. The error is measured at each matching pose pair $\left({}^{G}\bar{\mathbf{T}}_i, \bar{\mathbf{J}}_i\right)$. Here, ${}^{G}\bar{\mathbf{T}}_i$ is the estimated timestamped pose at i expressed in terms of the ground truth world frame G.

In order to calculate the error, the rigid transformation $\mathbf{T}$ from W to G must be found. This transformation is given using a quaternion as $\langle \mathrm{Q}, \boldsymbol{\rho} \rangle$ and using a rotation matrix as $\langle \mathbf{R}, \boldsymbol{\rho} \rangle$.

$$\mathbf{T} \Leftrightarrow \langle \mathrm{Q}, \boldsymbol{\rho} \rangle \Leftrightarrow \langle \mathbf{R}, \boldsymbol{\rho} \rangle$$

The transformation $\mathbf{T}$ is estimated by computing the rigid transformation between the translation terms of matching poses $(\boldsymbol{\rho}_{e,j}, \boldsymbol{\rho}_{d,j})$ as in [44]. For a more complete documentation of this process refer to Section 2.9.

The approach estimates $\mathbf{T}$ as the rigid transformation $\hat{\mathbf{T}}$ that minimizes the sum of squared distance between the estimated pose translation terms transformed according to $\mathbf{T}$ and the corresponding ground truth pose trans-

lation terms.

$$\hat{\mathbf{T}} \Leftrightarrow \left\langle \hat{\mathrm{Q}}, \hat{\boldsymbol{\rho}} \right\rangle \Leftrightarrow \left\langle \hat{\mathbf{R}}, \hat{\boldsymbol{\rho}} \right\rangle$$

$$\left\langle \hat{\mathbf{R}}, \hat{\boldsymbol{\rho}} \right\rangle = \underset{\mathbf{R},\boldsymbol{\rho}}{\operatorname{argmin}} \sum_{i=1}^{m} ||\mathbf{R}\boldsymbol{\rho}_{e,i} + \boldsymbol{\rho} - \boldsymbol{\rho}_{d,i}||^2$$

The estimated poses in the ground truth world frame are given as:

$${}^{G}\bar{\mathbf{T}}_i \Leftrightarrow \left\langle \hat{\mathrm{Q}}\mathrm{Q}_{e,i} \ , \hat{\mathrm{Q}} \begin{bmatrix} 0 \\ \boldsymbol{\rho}_{e,i} \end{bmatrix} \hat{\mathrm{Q}}^{-1} + \begin{bmatrix} 0 \\ \hat{\boldsymbol{\rho}} \end{bmatrix} \right\rangle \quad , \quad i = 0 \dots m$$

ATE evaluates how closely the poses in the estimated trajectory match the ground truth poses. The error is measured using the translational error and the rotational error terms. The translational error is given by Equation (4.18). The rotational error is given by Equation (4.19).

$${}^{G}\bar{\mathbf{T}}_j \Leftrightarrow \left\langle \ {}^{G}\mathrm{Q}_{e,j}, \ {}^{G}\boldsymbol{\rho}_{e,j} \right\rangle$$

$$\epsilon_i = ||{}^{G}\boldsymbol{\rho}_{e,i} - \ \boldsymbol{\rho}_{d,i}|| \tag{4.18}$$

$$\phi_i = 2 \arccos(|Re\{ \ ({}^{G}\mathrm{Q}_{e,i})^{-1}\mathrm{Q}_{d,i}\}|) \tag{4.19}$$

Here, $Re\{\mathrm{Q}\}$ is the real, scalar component of the quaternion Q and Q^{-1} is the inverse of the quaternion Q.

4.10.1.4 Relative Motion Error

The relative motion error metric is an error measurement inspired by [63]. It evaluates how closely the motion between poses in the estimated trajectory matches that of the ground truth. The motion is recovered by finding the rigid transformation between sequential poses for both datasets. The angular difference between the two rotations gives the rotational error and the difference in translation terms gives the translational error.

The relative motion $\tilde{\mathbf{T}}_i$ between two estimated poses $\bar{\mathbf{T}}_i$ and $\bar{\mathbf{T}}_{i+1}$ is given by:

$$\tilde{\mathbf{T}}_i = \bar{\mathbf{T}}_{i+1}\bar{\mathbf{T}}_i^{-1}$$

$$\tilde{\mathbf{T}}_i \Leftrightarrow \left\langle \tilde{\mathrm{Q}}_{e,i}, \tilde{\boldsymbol{\rho}}_{e,i} \right\rangle$$

If the SLAM algorithm has estimated the poses correctly, this motion should be the same as the ground truth motion $\tilde{\mathbf{J}}_i$ between the ground truth poses $\bar{\mathbf{J}}_i$ at d_i and $\bar{\mathbf{J}}_{i+1}$ at d_{i+1}.

$$\tilde{\mathbf{J}}_i = \bar{\mathbf{J}}_{i+1}\bar{\mathbf{J}}_i^{-1}$$

$$\tilde{\mathbf{J}}_i \Leftrightarrow \left\langle \tilde{\mathrm{Q}}_{d,i}, \tilde{\boldsymbol{\rho}}_{d,i} \right\rangle$$

The difference in angle of rotation and translation between the two transformations $\tilde{\mathbf{J}}_i$ and $\tilde{\mathbf{T}}_i$ indicates how much angular and positional error was introduced by the motion from $\bar{\mathbf{T}}_i$ to $\bar{\mathbf{T}}_{i+1}$.

The angular error ϕ can be found as in Equation (4.20).

$$\phi_i = 2\arccos\left(\left|Re\left\{\tilde{\mathrm{Q}}_{e,i}\tilde{\mathrm{Q}}_{d,i}^{-1}\right\}\right|\right) \quad \text{for} \quad i = 1\ldots m \tag{4.20}$$

The translational error is found by taking the Euclidean distance between the two translation terms.

$$\epsilon_i = ||\tilde{\boldsymbol{\rho}}_{e,i} - \tilde{\boldsymbol{\rho}}_{d,i}|| \quad \text{for} \quad i = 1\ldots m$$

4.11 Chapter Conclusion

This chapter introduced several interest point detectors, descriptor extractors, and descriptor matchers, and explained their incorporation into the RGB-D SLAM System. It also introduced two methods for evaluating the pose estimates of a SLAM system against a set of ground truth poses. The next chapter will use the tools and methods developed here to discuss the results of testing the RGB-D SLAM System using these new feature detection systems.

Chapter 5

Feature Detection - Experiments

5.1 Chapter Summary

This chapter details testing of the RGB-D SLAM System using the feature detectors, descriptors, and matchers outlined in Chapter 4. Testing consisted of an evaluation of pose estimate quality obtained using the RGB-D SLAM System with BRISK-AGAST and Adjustable BRISK-AGAST detectors, BRISK and FREAK descriptor extractors, and brute-force and FLANN matchers, while processing each frame in an RGB-D video sequence. This video sequence and the associated ground truth trajectory of poses were supplied for SLAM benchmarking in [62]. The ground truth pose information was recorded using an external motion capture system at a rate of 100 Hz.

The primary video sequence used for testing was the "freiburg1_room" dataset. This video captures a scene consisting of a pair of L shaped desks forming a "T" shape with a stuffed animal sitting in an office chair at the right-most desk. The video pans across the two desks from left to right, recording the desks as well as the room they are located in. Figure 4.6 shows example frames from this video sequence. The video was selected because it provides a combination of translational and rotational movement and revisits locations in the scene. This allows for testing of the performance of pairwise alignment in the presence of changes of rotation, scale, and perspective. The ability of RGB-D SLAM to recognize loop closures is also tested by this sequence.

The feature detection systems are evaluated in terms of detection, extraction, matching, and aggregate run time as well as the Absolute Trajectory Error (ATE) and Relative Motion Error (RME) metrics for the estimated poses outlined in Section 4.10.

5.2 Chapter Organization

This chapter will first analyze the performance of each configuration and suggest suitable parameter settings for these configurations. Interest point detectors, descriptor extractors, and matchers will then be compared and an "optimal" configuration will be suggested.

Each configuration will be analyzed independently in Sections 5.3-5.11.

The major results of this chapter are found in the interest point detector comparison section (Section 5.12), the matcher comparison section (Section 5.13), and the extractor comparison section (Section 5.14). A summary of these results is given in Section 5.15.

5.3 ORB and SURF Baseline

The default RGB-D SLAM program allows for the choice from two main feature detection systems SURF and ORB. A third system, GPU-SIFT, is available only with computers containing an NVIDIA GPU. Both SURF and ORB trials were run using the 21 s video sequence in order to establish a baseline for evaluation.

The ORB feature system consisted of a FAST feature detector, an ORB descriptor extractor, and a brute-force matcher using the Hamming distance metric. The SURF feature system consisted of a SURF detector, a SURF descriptor extractor, and a FLANN matcher. Both SURF and ORB detectors were DynamicAdaptedFeatureDetector implementations similar in concept to the one discussed in Section 4.4. The dynamic implementations run up to `max_iters` passes on each image while tuning their detector's parameters, in an attempt to ensure that the number of features in each frame falls between `min_features` and `max_features`.

The default ORB system was tested using a wide range of values for `min_features`, `max_features`, and `max_iters`. A complete run processing the full 21 s video sequence was unable to be completed. The ORB system encountered significant difficulties at times of large angular motion, typically losing its localization solution when the camera was rotating about an object or panning. This is due to the nature of the ORB descriptor. Although it was designed to be robust to in-plane rotations, panning of the camera resulted

Table 5.1: SURF results.

min feat.	max feat.	max iters.	Run time (s)	RME Trans. (m)	RME Ang. (rad)	ATE Trans. (m)	ATE Ang. (rad)
300	301	1	158.227	0.019 ± 0.046	0.013 ± 0.015	0.210 ± 0.120	0.199 ± 0.067
300	301	3	174.124	0.016 ± 0.024	0.012 ± 0.010	0.119 ± 0.071	0.095 ± 0.050
300	350	2	160.671	0.015 ± 0.022	0.013 ± 0.010	0.141 ± 0.073	0.135 ± 0.054
300	700	1	156.570	0.021 ± 0.059	0.015 ± 0.021	0.274 ± 0.150	0.468 ± 0.079
300	700	2	158.354	0.015 ± 0.024	0.013 ± 0.010	0.160 ± 0.098	0.137 ± 0.060
300	700	3	172.320	0.017 ± 0.037	0.013 ± 0.015	0.143 ± 0.081	0.148 ± 0.047
300	400	1	150.693	0.018 ± 0.037	0.013 ± 0.013	0.154 ± 0.092	0.135 ± 0.060
300	400	2	160.207	0.016 ± 0.028	0.013 ± 0.012	0.140 ± 0.074	0.121 ± 0.050
300	400	3	168.596	0.021 ± 0.056	0.014 ± 0.020	0.134 ± 0.099	0.100 ± 0.061
400	500	2	165.869	0.020 ± 0.082	0.014 ± 0.029	0.177 ± 0.122	0.173 ± 0.044
400	500	3	176.769	0.016 ± 0.035	0.013 ± 0.013	0.126 ± 0.073	0.101 ± 0.050
500	600	1	154.761	0.018 ± 0.038	0.013 ± 0.014	0.156 ± 0.102	0.144 ± 0.060
500	600	2	170.537	0.017 ± 0.047	0.013 ± 0.021	0.221 ± 0.095	0.282 ± 0.065
500	600	3	187.104	0.014 ± 0.022	0.012 ± 0.011	0.111 ± 0.055	0.103 ± 0.054

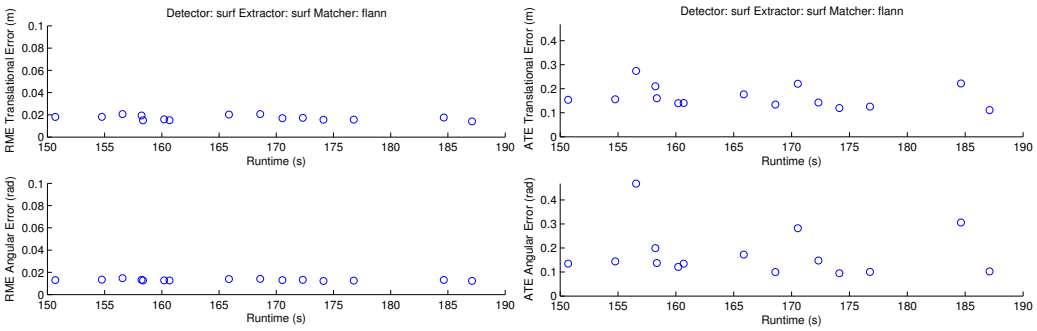
Table 5.2: Average SURF timings.

min_features	max_features	max_iters	Detect Time (s)	Extract Time (s)	Match Time (s)	Run time (s)
300	301	1	0.081	0.077	0.025	158.227
300	301	3	0.146	0.075	0.025	174.124
300	350	2	0.111	0.077	0.027	160.671
300	700	1	0.086	0.081	0.038	156.570
300	700	2	0.093	0.081	0.038	158.354
300	700	3	0.100	0.082	0.039	172.320
300	400	1	0.083	0.079	0.030	150.693
300	400	2	0.105	0.078	0.030	160.207
300	400	3	0.126	0.077	0.030	168.596
400	500	2	0.114	0.080	0.035	165.869
400	500	3	0.140	0.079	0.035	176.769
500	600	1	0.085	0.081	0.036	154.761
500	600	2	0.125	0.080	0.038	170.537
500	600	3	0.157	0.081	0.038	187.104

in a change in perspective that the system could not handle. The descriptors changed in appearance to such an extent that the number of matching features dropped to a level insufficient for pairwise alignment, resulting in a failure of the RGB-D SLAM System. The ORB configuration was able to track the camera well during periods of solely translational movement.

The DynamicAdaptedFeatureDetector SURF system was tested for target feature bounds from 300-400 features per image to 500-600 features per images and values of `max_iters` ranging from one to three. The RGB-D SLAM System was able to successfully run to completion using all of these settings. The results of these runs are summarized in Table 5.1, Table 5.2, and Figure 5.1. The scatter plots in Figure 5.1 show the mean ATE and RME values in Table 5.1 compared to the total run time required to process the video sequence.

From this data, a baseline for RGB-D SLAM was established. Processing the video sequence with SURF took between 150 s and 187 s or approximately 6.8 to 8.5 times longer than the length of the sequence. There was a mean error of 0.017 m and 0.013 rad using the RME metric and an error of 0.166 m and 0.176 rad using the ATE metric.



(a) (b)

Figure 5.1: RME (Figure 5.1a) and ATE (Figure 5.1b) for the SURF Baseline. These plots compare the total run time for each configuration in Table 5.1 to the mean estimated pose error.

Table 5.3: Trajectory errors for a BRISK-AGAST detector, a BRISK extractor, and a brute-force matcher.

Thresh	Oct	Scale	Avg. Feat.	RME Trans. (m)	RME Ang. (rad)	ATE Trans. (m)	ATE Ang. (rad)
10	0	1.500	1206.4	0.074 ± 0.144	0.039 ± 0.067	0.738 ± 0.365	0.907 ± 0.458
12	0	1.500	1006.3	0.059 ± 0.108	0.035 ± 0.052	0.365 ± 0.286	0.766 ± 0.171
6	0	2.000	1893.9	0.163 ± 0.389	0.123 ± 0.336	1.551 ± 0.483	2.515 ± 0.409
8	0	2.000	1449.2	0.110 ± 0.198	0.064 ± 0.126	0.600 ± 0.340	1.022 ± 0.308
12	0	2.000	981.2	0.071 ± 0.113	0.042 ± 0.064	0.558 ± 0.318	1.058 ± 0.301
14	0	2.000	836.1	0.055 ± 0.083	0.033 ± 0.044	0.349 ± 0.282	0.360 ± 0.208
8	0	3.000	1294.1	0.096 ± 0.152	0.057 ± 0.078	0.584 ± 0.234	1.085 ± 0.225
10	0	3.000	1056.3	0.067 ± 0.088	0.041 ± 0.052	0.441 ± 0.347	0.809 ± 0.277
12	0	3.000	887.7	0.066 ± 0.114	0.040 ± 0.064	0.554 ± 0.342	0.970 ± 0.209
14	0	3.000	760.1	0.062 ± 0.112	0.039 ± 0.075	0.445 ± 0.285	0.596 ± 0.155
16	0	3.000	659.7	0.072 ± 0.165	0.054 ± 0.204	0.446 ± 0.313	0.644 ± 0.185
6	1	1.500	661.5	0.048 ± 0.126	0.026 ± 0.044	0.210 ± 0.146	0.316 ± 0.050
6	1	2.000	624.7	0.060 ± 0.212	0.032 ± 0.077	0.661 ± 0.328	1.067 ± 0.293

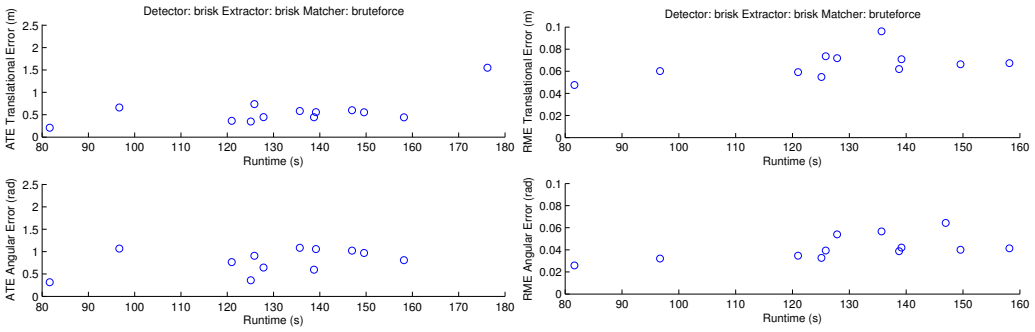
5.4 BRISK-AGAST and BRISK with Brute-Force

The BRISK-AGAST detector (Section 4.3), BRISK extractor (Section 4.6), and brute-force matcher configuration has three open parameters `thresh`, `nOctaves`, and `patternScale`. The parameters `thresh` and `nOctaves` affect the location and number of features detected in each frame, while `patternScale` affects the size of the sampling region used to construct descriptors. Cursory testing produced the best results for values of `thresh` between six and twenty, values of `nOctaves` of zero and one, and `patternScale` values between 0.5 and 3.0. A more complete set of tests was then run using `thresh` values between six and twenty in increments of two, `nOctaves` values of zero and one, and `patternScale` values of 0.8, 1.0, 1.2 , 1.5, 2.0, and 3.0. Error and timing results for trials which ran to completion are shown in Table 5.3 and Table 5.4 and plotted in Figure 5.2.

Results were only found for `patternScale` values of 1.5, 2.0, and 3.0. For

Table 5.4: BRISK-AGAST detector, BRISK descriptor, brute-force matcher timing results. Detection, extraction, and match timings are given as average per frame.

Threshold	Octaves	Scale	Detect Time (s)	Extract Time (s)	Match Time (s)	Run time (s)
10	0	1.500	0.012	0.008	0.011	125.853
8	0	2.000	0.015	0.009	0.011	146.960
6	0	2.000	0.019	0.012	0.012	176.204
12	0	2.000	0.010	0.007	0.011	139.166
14	0	2.000	0.009	0.006	0.010	125.084
12	0	1.500	0.010	0.007	0.011	120.976
16	0	3.000	0.008	0.005	0.010	127.844
14	0	3.000	0.009	0.005	0.010	138.745
12	0	3.000	0.010	0.006	0.011	149.556
10	0	3.000	0.012	0.007	0.011	158.164
8	0	3.000	0.015	0.009	0.012	135.669
6	1	1.500	0.016	0.005	0.011	81.640
6	1	2.000	0.015	0.005	0.011	96.693



(a) (b)

Figure 5.2: Error metrics for the runs using a BRISK detector BRISK extractor and brute-force matcher shown in Table 5.3. Figure 5.2a shows the ATE values and Figure 5.2b gives the RME values.

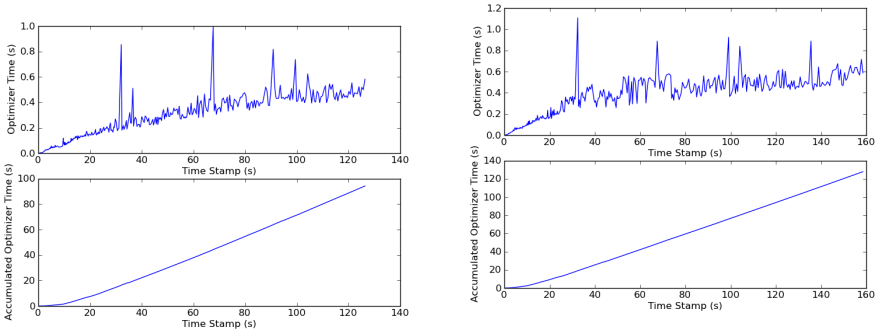
small values of `patternScale`, the sampling kernels in the sampling pattern used to generate the BRISK descriptor contain very few pixels. A shift in the location of an interest point by just one pixel or a small out-of-plane rotation of the camera can have a significant impact on the contents of these kernels. This, in turn, changes the result of the binary comparisons between these Gaussian kernels and alters the content of descriptors. With larger values of `patternScale`, small changes in the location of an interest point do not alter the BRISK descriptor as significantly, so more matches are found, and the RGB-D SLAM System is able to obtain a localization solution more frequently. However, increasing the size of the sampling kernels also makes the descriptors of nearby interest points appear more similar, leading to a rise in the incidence of false matches between interest points which may result in incorrect estimates of the transformations found by pairwise alignment.

This effect can be seen in Table 5.3. The lowest error trajectory estimates are found for a `patternScale` value of 1.5. While there are more successful completions for scale values of 2.0 and 3.0, the ATE translation and rotation errors are two to three times larger than the best trajectory estimate generated using smaller kernels.

Increasing `patternScale` also increases the time needed to process the video sequence. The increase in the number of erroneous matches introduces more edges to the pose graph and alters the weights of the already existing edges. This means that the $\mathbf{g^2o}$ pose graph optimizer must operate on a more complex graph and must run for more iterations during each optimization, while it optimizes the pose graph using incorrect constraints. Increasing the scale from 1.5 to 2.0 adds roughly 20 s to optimization time, while increasing from 2.0 to 3.0 adds another 10 s. Figure 5.3 shows this increase in the accumulated optimization time for settings of `thresh` = 10, `nOctaves` = 0, and `patternScale`'s of 1.5 and 3.0. The increase in total optimization time of 33.5 s from 93.41 s to 127.92 s almost completely explains the increase in total run time from 125.85 s to 158.16 s.

These observations suggest that it is best to select a value of `patternScale` which is just large enough to allow for reliable matching of descriptors, while not being too large to result in erroneous matches. This data suggests a value of `patternScale` between 1.5 to 2.0 would be suitable.

As discussed in Section 4.3, increasing `nOctaves` from zero to one results in a significant decrease in the number of features detected and a drop in run



(a) BRISK-AGAST detector

(b) Adjustable BRISK-AGAST detector

Figure 5.3: Comparison of optimization time and accumulated optimization time in the RGB-D SLAM System using an AGAST detector, a BRISK descriptor, and a brute-force matcher. Figure 5.3a shows optimization time for `thresh` = 10 `nOctaves` = 0 `patternScale` = 1.5 and Figure 5.3b shows optimization time for `threshold` = 10 `nOctaves` = 0 `patternScale` = 3.

times across the board. Subsequent increases in the number of octaves raise detection times and slightly reduce the number of features. Due to the added detection time and no perceivable benefit, the value of `nOctaves` was limited to zero and one. Table 5.3 shows that only two of the forty-eight trials with an `nOctaves` value of one were successful. With this setting, a comparatively small number of features were detected at each frame. Consequently, at times when the appearance of objects changes significantly such as during large angular movement, or at times when few features are detected, the number of matching features between the current frame and previous keyframes may drop to a level where visual odometry cannot be estimated and localization is lost. Only by relaxing the threshold constraint significantly can enough features be detected to allow for reliable localization.

When it is possible to use an `nOctaves` value of one and still reliably detect enough features for continuous localization, doing so is preferable, as the location of interest points detected using the scale stack is more precise and consistent than that of interest points localized using solely the original image.

The value of the threshold, `thresh`, for the AGAST-BRISK detector is discussed in detail in Section 4.3. In short, increasing `thresh` decreases the number of features detected but also decreases detection, extraction and matching times. Care has to be taken not to decrease `thresh` too low, or

each frame will be saturated with features and a large number of erroneous matches will be found. As such, the general goal of tuning `thresh` is to select a value that is just small enough to consistently provide a pose solution without increasing run time too much.

The experimental results in Tables 5.3 and 5.4 support this approach. Comparing rows of the same number of octaves and sampling pattern size shows that increasing the threshold will decrease the RME and ATE translational and rotational errors and also decrease total run time. The best result was found with a `thresh` value of six, an `nOctaves` value of one, and a `patternScale` value of 1.5. Here, a mean absolute trajectory error of 0.210 ± 0.146 m and 0.316 ± 0.050 rad was found with a run time of 81.640 s. The value of `thresh` was actually quite low with these settings, but, more importantly, the number of features was near its lowest. Comparing the average number of features to the error values shows that, in general, the errors are smallest when the feature counts are lowest.

Even this detector configuration is not ideal in terms of the number of interest points found. Figure 5.4 shows the number of features detected using the "best result" configuration. The feature count changes quite a bit depending on the content of each frame. In order to ensure the bare minimum of approximately 200 features per frame needed for consistent localization are found at all times, the value of `thresh` must be kept low, resulting in a larger number of features and erroneous matches near the peaks at frame indices 150 (22 s), 250 (37 s), and 450 (70 s).

It would be best to use a detector that produces a consistent number of features regardless of the scene make up, so as to limit the frequency of erroneous matches while maintaining consistent localization. The Adjustable BRISK-AGAST detector (Section 4.5) is such a detector and will be discussed in Sections 5.8-5.11.

5.5 BRISK-AGAST and FREAK with Brute-Force

The BRISK-AGAST detector (Section 4.3), FREAK extractor (Section 4.7), and brute-force-Hamming matcher configuration has five open parameters. The parameters `thresh` and `nOctaves` affect the location and number of features detected, `patternScale` affects the size of the sampling region used to

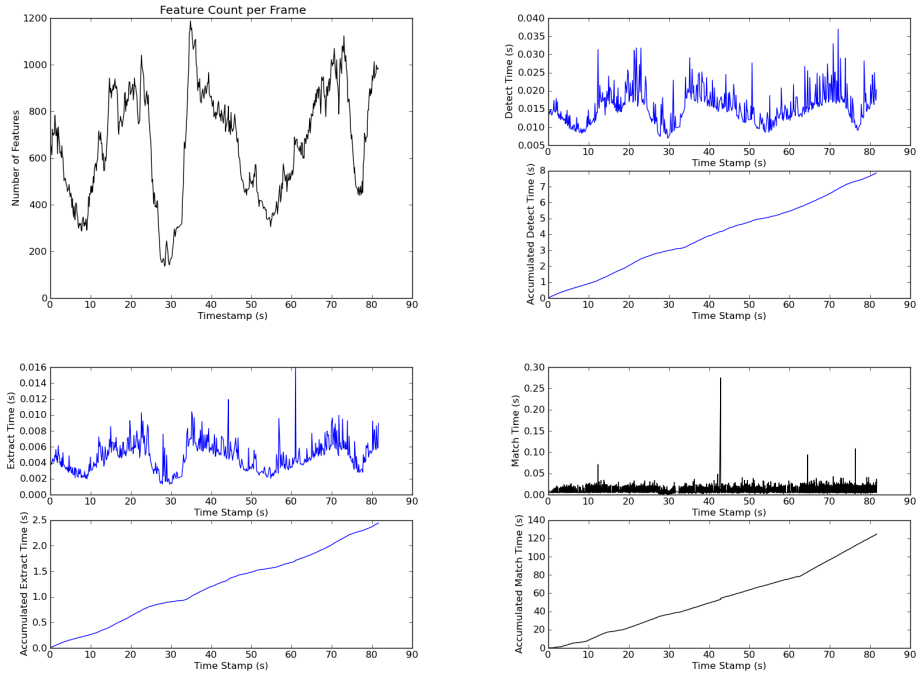


Figure 5.4: Number of features detected and detection, extraction, and matching timings for each frame for `thresh`=6 `nOctaves`=1 `patternScale`=1.5.

construct descriptors, and the Boolean variables `scaleNormalization` (SN) and `orientationNormalization` (ON) determine whether the sampling pattern is scaled and rotated prior to descriptor extraction so as provide a measure of scale and rotation invariance. Cursory testing produced the best results for values of `thresh` between eight and twenty, values of `nOctaves` of zero and one, and `patternScale` values between 8.0 and 22.0. A more complete set of tests was then run using `thresh` values between eight and twenty in increments of four, `nOctaves` values of zero and 1, and `patternScale` values of 8.0, 10.0, 12.0, 16.0 and 20.0 . Combinations of `scaleNormalization` and `orientationNormalization` with both enabled, both disabled, and only SN enabled were used. Settings with SN disabled and ON enabled were not used because consistent localization could not be obtained in that configuration. Error and timing results for trials that ran to completion are shown in Table 5.5 and Table 5.6.

Contrary to the BRISK description case, increasing `thresh` resulted in a slight increase in the ATE translation and rotation errors as the number of features decreased. This is indicative of the FREAK and brute-force com-

Table 5.5: Results using a BRISK-AGAST detector, a FREAK extractor, and a brute-force-Hamming matcher.

Thr.	Oct	Sc.	SN/ ON	Avg. Feat	RME Tr. (m)	RME Ang. (rad)	ATE Tr. (m)	ATE Ang. (rad)
8	0	8.0	FF	1540.9	0.046 ± 0.149	0.043 ± 0.172	0.219 ± 0.185	0.443 ± 0.125
8	0	10.0	FF	1540.9	0.049 ± 0.320	0.025 ± 0.113	0.181 ± 0.319	0.342 ± 0.109
8	0	12.0	FF	1540.9	0.026 ± 0.061	0.017 ± 0.024	0.194 ± 0.173	0.431 ± 0.049
8	0	14.0	FF	1540.9	0.020 ± 0.049	0.014 ± 0.017	0.132 ± 0.090	0.251 ± 0.072
8	0	16.0	FF	1540.9	0.024 ± 0.059	0.016 ± 0.020	0.156 ± 0.130	0.296 ± 0.036
8	0	18.0	FF	1540.9	0.024 ± 0.057	0.015 ± 0.019	0.147 ± 0.118	0.320 ± 0.039
10	0	12.0	FF	1242.1	0.040 ± 0.158	0.021 ± 0.050	0.325 ± 0.307	0.683 ± 0.066
10	0	14.0	FF	1242.1	0.042 ± 0.208	0.022 ± 0.075	0.245 ± 0.196	0.392 ± 0.038
10	0	18.0	FF	1242.1	0.023 ± 0.064	0.015 ± 0.024	0.156 ± 0.092	0.322 ± 0.051
12	0	16.0	FF	1033.7	0.045 ± 0.191	0.021 ± 0.054	0.231 ± 0.185	0.520 ± 0.062
12	0	18.0	FF	1033.7	0.050 ± 0.197	0.028 ± 0.095	0.187 ± 0.166	0.368 ± 0.091
8	0	10.0	TF	1540.9	0.050 ± 0.260	0.023 ± 0.081	0.191 ± 0.312	0.159 ± 0.098
8	0	12.0	TF	1540.9	0.023 ± 0.079	0.016 ± 0.026	0.125 ± 0.105	0.241 ± 0.032
8	0	14.0	TF	1540.9	0.024 ± 0.057	0.015 ± 0.016	0.146 ± 0.090	0.277 ± 0.038
8	0	16.0	TF	1540.9	0.021 ± 0.054	0.014 ± 0.018	0.112 ± 0.095	0.251 ± 0.051
8	0	18.0	TF	1540.9	0.019 ± 0.038	0.014 ± 0.014	0.124 ± 0.100	0.178 ± 0.027
10	0	12.0	TF	1242.1	0.040 ± 0.141	0.020 ± 0.045	1.025 ± 0.410	1.059 ± 0.132
10	0	14.0	TF	1242.1	0.032 ± 0.130	0.026 ± 0.130	0.701 ± 0.323	1.700 ± 0.490
10	0	16.0	TF	1242.1	0.036 ± 0.123	0.019 ± 0.039	0.139 ± 0.143	0.302 ± 0.047
10	0	18.0	TF	1242.1	0.025 ± 0.081	0.016 ± 0.029	0.175 ± 0.126	0.358 ± 0.045
12	0	14.0	TF	1033.7	0.031 ± 0.118	0.018 ± 0.038	0.212 ± 0.157	0.419 ± 0.040
12	0	16.0	TF	1033.7	0.036 ± 0.120	0.019 ± 0.036	0.224 ± 0.189	0.459 ± 0.043
12	0	18.0	TF	1033.7	0.041 ± 0.165	0.031 ± 0.114	0.173 ± 0.139	0.524 ± 0.230
8	0	16.0	TT	1540.9	0.036 ± 0.124	0.021 ± 0.049	0.181 ± 0.118	0.167 ± 0.052
8	0	18.0	TT	1540.9	0.021 ± 0.053	0.015 ± 0.021	0.174 ± 0.111	0.227 ± 0.026
10	0	18.0	TT	1242.1	0.039 ± 0.127	0.021 ± 0.045	0.350 ± 0.212	0.657 ± 0.117

Table 5.6: Timing results for RGB-D SLAM using a BRISK-AGAST detector, a FREAK descriptor extractor, and a brute-force matcher. Matching, extraction, and detection times are given in seconds per frame.

Threshold	Octaves	Scale	SN/ON	Detect Time (s)	Extract Time (s)	Match Time (s)	Run time (s)
8	0	8.0	FF	0.017	0.004	0.176	257.749
8	0	10.0	FF	0.017	0.004	0.178	256.906
8	0	12.0	FF	0.017	0.004	0.172	268.341
8	0	14.0	FF	0.016	0.004	0.173	263.032
8	0	16.0	FF	0.016	0.004	0.177	266.874
8	0	18.0	FF	0.016	0.004	0.174	275.894
10	0	12.0	FF	0.015	0.004	0.138	212.240
10	0	14.0	FF	0.014	0.004	0.139	225.409
10	0	18.0	FF	0.014	0.004	0.137	230.736
12	0	16.0	FF	0.012	0.003	0.106	184.934
12	0	18.0	FF	0.012	0.003	0.105	195.961
8	0	10.0	TF	0.017	0.004	0.179	266.824
8	0	12.0	TF	0.017	0.004	0.179	263.745
8	0	14.0	TF	0.017	0.004	0.178	273.765
8	0	16.0	TF	0.016	0.004	0.175	272.098
8	0	18.0	TF	0.016	0.004	0.178	291.475
10	0	12.0	TF	0.015	0.004	0.135	213.724
10	0	14.0	TF	0.014	0.004	0.139	219.921
10	0	16.0	TF	0.014	0.004	0.144	237.775
10	0	18.0	TF	0.014	0.004	0.141	234.662
12	0	14.0	TF	0.013	0.003	0.106	186.431
12	0	16.0	TF	0.012	0.003	0.105	190.798
12	0	18.0	TF	0.012	0.003	0.106	202.901
8	0	16.0	TT	0.017	0.006	0.178	266.104
8	0	18.0	TT	0.017	0.006	0.179	269.047
10	0	18.0	TT	0.014	0.005	0.137	219.186

bination being more discerning than the BRISK and brute-force combination, trending more toward disqualifying correct matches than allowing false matches. It therefore requires comparatively more features than the BRISK variant to estimate pairwise alignment, but it is better able to handle higher feature counts.

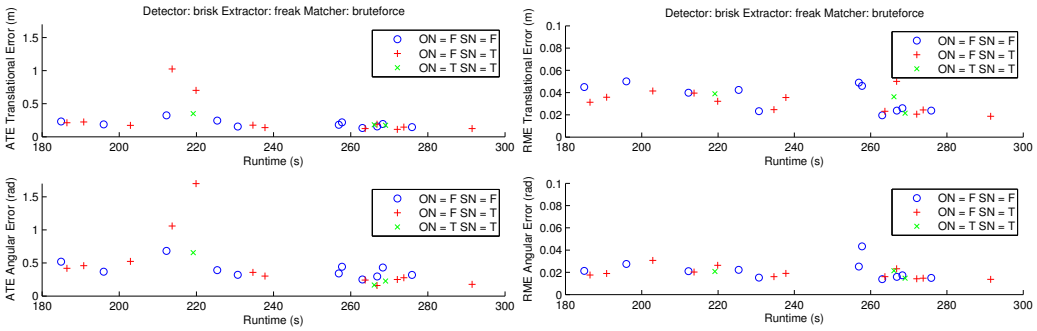
As would be expected if this were the case, the non-zero `nOctaves` values resulted in the detection of too few features to perform pairwise alignment. Since there were no successful runs with these settings, `nOctaves` should be set to zero.

Increasing `patternScale` resulted in a greater total run time on the order of a 20-40 s increase for an increase from 12.0 to 18.0. Increasing `patternScale` also affected the ATE and RME metrics, with the error values decreasing until the sampling pattern scale neared 16.0, and then increasing again as the scale neared 20.0. The best results were found near a scale of 14.0 or 18.0, depending on whether `scaleNormalization` was enabled.

Figure 5.5 shows a breakdown of the ATE and RME metrics for the three ON/SN configurations. Enabling `scaleNormalization` and leaving `orientationNormalization` off yielded a decrease in ATE translation error on the order of 0.02 to 0.04 m and an improvement in the angular error of 0.1 to 0.2 rad for low `thresh` values, but made no discernible improvement when `thresh` was around twelve. Total run time increased by 5-10 s when only SN was enabled, and fell below normal values when both SN and ON were enabled.

While enabling SN did not affect the number of successfully completed runs, enabling ON caused nine of the twelve previously successful configurations to fail. As such, it is recommended that both ON and SN be disabled as neither provides any tangible benefit.

The best results were observed for settings with ON and SN disabled, a `thresh` value of twelve, an `nOctaves` value of zero, and a `patternScale` of 18.0. This setup gave a run time of 195.96 s and a mean ATE of 0.187 m and 0.368 rad.



(a) (b)

Figure 5.5: ATE and RME error values for the values in BRISK-AGAST detector, BRISK extractor, brute-force-Hamming matcher configuration. The blue circles are runs where ON and SN are off, the red crosses are runs where just SN is enabled, and the green x's are runs where both SN and ON are enabled. Figure 5.5a gives the absolute trajectory error and Figure 5.5b gives the relative motion error.

Table 5.7: ATE and RME values for runs using a BRISK-AGAST detector, a BRISK descriptor extractor, and a FLANN matcher.

Thresh.	Oct.	Scale	Avg. Feat.	RME Trans. (m)	RME Ang. (rad)	ATE Trans. (m)	ATE Ang. (rad)
14	0	1.000	877.8	0.067 ± 0.170	0.041 ± 0.134	0.511 ± 0.284	0.805 ± 0.211
12	0	1.500	1006.3	0.054 ± 0.105	0.031 ± 0.047	0.415 ± 0.310	0.721 ± 0.204
14	0	1.500	856.0	0.056 ± 0.121	0.033 ± 0.075	0.648 ± 0.295	1.777 ± 0.710
8	0	2.000	1449.2	0.094 ± 0.148	0.058 ± 0.135	0.414 ± 0.291	0.732 ± 0.173
12	0	2.000	981.2	0.053 ± 0.075	0.033 ± 0.044	0.317 ± 0.249	0.672 ± 0.198
14	0	2.000	836.1	0.060 ± 0.103	0.036 ± 0.052	0.270 ± 0.251	0.368 ± 0.159
8	0	2.500	1382.2	0.075 ± 0.120	0.044 ± 0.065	0.501 ± 0.357	0.891 ± 0.371
12	0	2.500	941.3	0.056 ± 0.087	0.033 ± 0.044	0.399 ± 0.294	0.789 ± 0.264
14	0	2.500	804.0	0.078 ± 0.182	0.045 ± 0.110	0.576 ± 0.255	1.524 ± 0.249
8	0	3.000	1294.1	0.098 ± 0.157	0.051 ± 0.066	0.570 ± 0.347	1.155 ± 0.152
10	0	3.000	1056.3	0.065 ± 0.081	0.041 ± 0.049	0.397 ± 0.275	0.876 ± 0.210
12	0	3.000	887.7	0.066 ± 0.099	0.039 ± 0.057	0.517 ± 0.347	0.567 ± 0.251

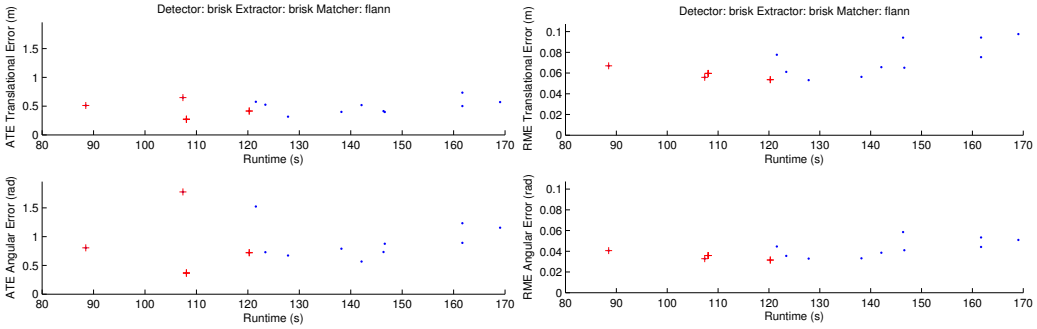
5.6 BRISK-AGAST and BRISK with FLANN

The BRISK-AGAST detector, BRISK descriptor extractor, and FLANN matcher configuration has three open parameters: `thresh`, `nOctaves`, and `patternScale`. In response to the results with the brute-force matcher, the parameter testing was centered around configurations which were known to work. Values for `thresh` between eight and fourteen in increments of two, values of `nOctaves` of zero and one, and `patternScale` values between 1.0 and 3.0 in increments of 0.5 were examined. Successfully completed runs are recorded in Table 5.7 and Table 5.8 and plotted in Figure 5.6.

All trial runs with an `nOctaves` value of one failed as well all but one run with a `patternScale` value of 1.0. This run had large ATE errors with an average translation error of 0.511 ± 0.284 m and an orientation error of

Table 5.8: BRISK-AGAST BRISK FLANN timing results. Matching, extraction, and detection results given in seconds per frame.

Threshold	Octaves	Scale	Detect Time (s)	Extract Time (s)	Match Time (s)	Run time (s)
14	0	1.000	0.009	0.006	0.012	88.496
12	0	1.500	0.011	0.007	0.010	120.269
14	0	1.500	0.009	0.006	0.010	107.381
8	0	2.000	0.015	0.010	0.010	146.389
12	0	2.000	0.011	0.007	0.011	127.816
14	0	2.000	0.009	0.006	0.010	108.068
8	0	2.500	0.015	0.009	0.010	161.709
12	0	2.500	0.010	0.007	0.012	138.190
14	0	2.500	0.009	0.006	0.010	121.562
8	0	3.000	0.015	0.009	0.010	169.040
10	0	3.000	0.012	0.007	0.010	146.625
12	0	3.000	0.010	0.006	0.010	142.105



(a) (b)

Figure 5.6: ATE and RME error values for the values in Table 5.7. Obtained using a BRISK-AGAST detector, a BRISK extractor, and a FLANN matcher. Figure 5.6a gives the absolute trajectory error compared to the total run time for each parameter setting and Figure 5.6b gives the relative motion error compared to the total run time. Points marked with red +'s are runs of note.

0.805 ± 0.211 rad. Such large errors, particularly the orientation values, are indicative of very poor performance. In general, the errors decreased as `patternScale` increased until a scale of 2.0 was reached, after which, the errors began to increase again. Performance was best when an average of 800-900 features per frame were detected, occurring for `thresh` values of twelve and fourteen.

Increasing `patternScale` increased total run time by approximately 10 s per 0.5 scale increase. Increasing `thresh` from eight to twelve decreased the average number of features detected by roughly 500 features per frame and an additional 150 features per frame drop resulted from an increase from twelve to fourteen. Total run time decreased by around 20 s in each case, with average detection and extraction times decreasing by several milliseconds per frame.

The best results were obtained with a configuration with a `thresh` value of fourteen, an `nOctaves` value of zero, and `patternScale` value of 2.0, giving a mean ATE of 0.270 ± 0.251 m and 0.368 ± 0.159 rad and a run time of 108.0768 s.

The translation and orientation errors were quite large for all runs. This was primarily due to difficulties encountered during rotations of the camera. The blurriness of the image frames during rotation and the substantial change in the appearance of the scene reduced the number of suitable matches between frames and introduced error into the pose estimate. This effect can be seen in Figure 5.7, depicting instantaneous ATE values for the best run. Note the increase in translation error at the 8 s mark, when the camera begins its horizontal rotation about a desk, and the large spike at the 10 s mark when the camera performs a vertical pan upwards from the desk and then back down. This creates a loop closure event which is not handled well, as can be seen by the constantly increasing rotational error afterwards.

5.7 BRISK-AGAST and FREAK with FLANN

The BRISK-AGAST detector, FREAK descriptor extractor, and FLANN matcher configuration has five open parameters: `thresh`, `nOctaves`, `patternScale`, `orientationNormalization`, and `scaleNormalization`. For this set of runs, `thresh` took values between six and fourteen in increments of two,

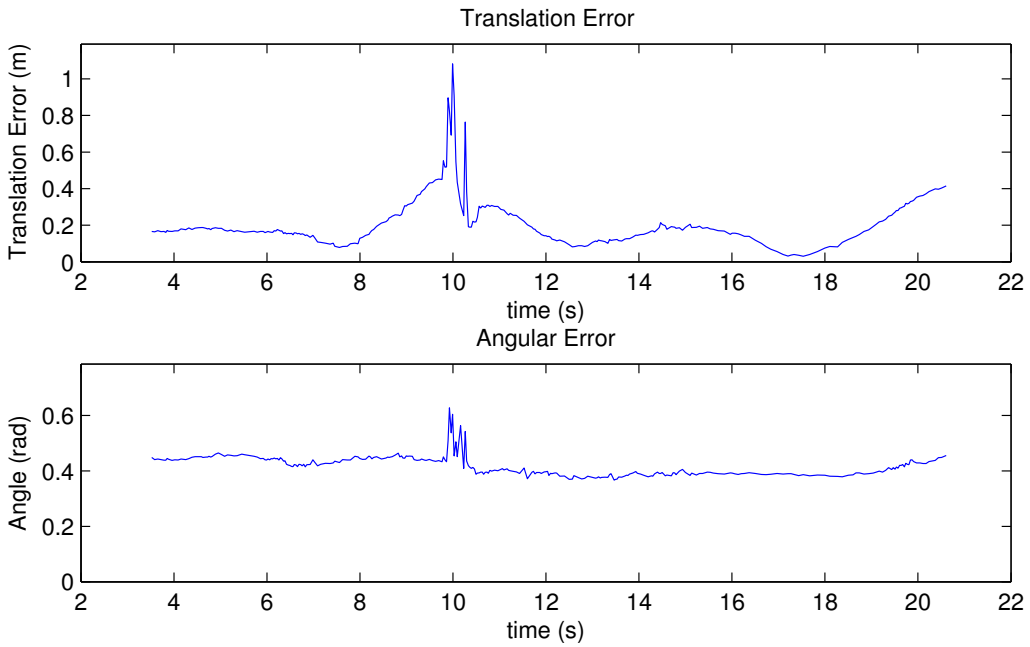


Figure 5.7: ATE values for the best BRISK-AGAST detector, BRISK extractor, and FLANN matcher run using `nOctaves` = 0, `patternScale` = 2.0 and `thresh` = 14.

`nOctaves` was set to zero and one, `patternScale` took values between 14.0 and 24.0 in increments of 2.0, and the normalization terms were configured with both ON and SN disabled, just SN enabled, and both ON and SN enabled. The successful runs are recorded in Table 5.9 and Table 5.10. Figure 5.8 plots average RME and ATE errors delimited using blue circles when no normalization was applied, using red crosses when just scale normalization was enabled, and black x's when both scale and orientation normalization were enabled.

When `nOctaves` was set to one, this configuration was unable to maintain localization throughout the entire run in any of the trials. With this `nOctaves` setting, a small number of features was detected and the system was unable to find enough matches between frames to estimate pairwise alignment. The lack of odometry information caused SLAM to fail at the start of the video in each case.

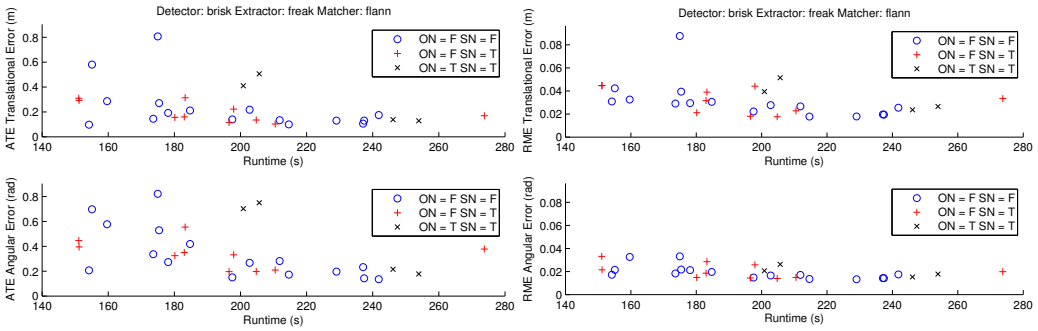
With `nOctaves` set to zero, the system was able to consistently complete runs using `thresh` values between six and twelve with average feature counts ranging from 2027 features per frame with a threshold of six to 1033 features per frame with a threshold of twelve. For a `thresh` value of fourteen, the

Table 5.9: RME and ATE values using a BRISK-AGAST detector, a FREAK descriptor extractor, and a FLANN matcher.

Thresh.	Oct.	Scale	SN/ON	Avg. Feat	RME Trans. (m)	RME Ang. (rad)	ATE Trans. (m)	ATE Ang. (rad)
6	0	12.0	FF	2027.1	0.025 ± 0.052	0.017 ± 0.022	0.175 ± 0.105	0.136 ± 0.058
8	0	12.0	FF	1540.9	0.027 ± 0.068	0.017 ± 0.026	0.133 ± 0.109	0.281 ± 0.035
10	0	12.0	FF	1242.1	0.088 ± 0.543	0.033 ± 0.145	0.807 ± 0.629	0.822 ± 0.087
6	0	14.0	FF	2027.1	0.020 ± 0.038	0.014 ± 0.014	0.105 ± 0.081	0.232 ± 0.061
8	0	14.0	FF	1540.9	0.022 ± 0.050	0.015 ± 0.016	0.140 ± 0.098	0.151 ± 0.031
10	0	14.0	FF	1242.1	0.029 ± 0.097	0.018 ± 0.036	0.145 ± 0.122	0.336 ± 0.027
6	0	16.0	FF	2027.1	0.019 ± 0.030	0.014 ± 0.012	0.130 ± 0.107	0.143 ± 0.043
8	0	16.0	FF	1540.9	0.028 ± 0.092	0.017 ± 0.030	0.218 ± 0.114	0.267 ± 0.066
10	0	16.0	FF	1242.1	0.029 ± 0.099	0.021 ± 0.071	0.192 ± 0.124	0.273 ± 0.054
12	0	16.0	FF	1033.7	0.042 ± 0.163	0.021 ± 0.053	0.580 ± 0.239	0.697 ± 0.061
6	0	18.0	FF	2027.1	0.018 ± 0.027	0.013 ± 0.011	0.130 ± 0.107	0.197 ± 0.040
8	0	18.0	FF	1540.9	0.018 ± 0.037	0.014 ± 0.014	0.100 ± 0.080	0.172 ± 0.032
10	0	18.0	FF	1242.1	0.031 ± 0.098	0.020 ± 0.045	0.211 ± 0.135	0.418 ± 0.035
12	0	18.0	FF	1033.7	0.033 ± 0.120	0.033 ± 0.161	0.286 ± 0.204	0.577 ± 0.100
8	0	10.0	TF	1540.9	0.033 ± 0.146	0.020 ± 0.054	0.170 ± 0.144	0.378 ± 0.092
8	0	12.0	TF	1540.9	0.044 ± 0.198	0.026 ± 0.087	0.222 ± 0.196	0.332 ± 0.049
8	0	14.0	TF	1540.9	0.018 ± 0.036	0.014 ± 0.016	0.116 ± 0.096	0.196 ± 0.028
10	0	14.0	TF	1242.1	0.039 ± 0.146	0.029 ± 0.126	0.314 ± 0.191	0.554 ± 0.092
8	0	16.0	TF	1540.9	0.018 ± 0.035	0.014 ± 0.016	0.135 ± 0.082	0.198 ± 0.024
10	0	16.0	TF	1242.1	0.032 ± 0.096	0.019 ± 0.033	0.160 ± 0.139	0.350 ± 0.045
12	0	16.0	TF	1033.7	0.045 ± 0.195	0.021 ± 0.055	0.293 ± 0.192	0.395 ± 0.088
8	0	18.0	TF	1540.9	0.023 ± 0.067	0.015 ± 0.023	0.103 ± 0.110	0.209 ± 0.040
10	0	18.0	TF	1242.1	0.021 ± 0.046	0.015 ± 0.018	0.156 ± 0.105	0.325 ± 0.037
12	0	18.0	TF	1033.7	0.045 ± 0.171	0.033 ± 0.159	0.310 ± 0.238	0.445 ± 0.128
8	0	16.0	TT	1540.9	0.027 ± 0.078	0.018 ± 0.037	0.129 ± 0.114	0.178 ± 0.073
10	0	16.0	TT	1242.1	0.052 ± 0.209	0.026 ± 0.070	0.506 ± 0.340	0.750 ± 0.209
8	0	18.0	TT	1540.9	0.024 ± 0.057	0.015 ± 0.019	0.138 ± 0.102	0.215 ± 0.040
10	0	18.0	TT	1242.1	0.039 ± 0.150	0.021 ± 0.047	0.410 ± 0.302	0.702 ± 0.099

Table 5.10: Timing results for a BRISK-AGAST detector, a FREAK descriptor extractor and a FLANN matcher.

Threshold	Octaves	Scale	SN/ON	Detect Time (s)	Extract Time (s)	Match Time (s)	Run time (s)
6	0	12.0	FF	0.024	0.006	0.157	241.805
8	0	12.0	FF	0.019	0.005	0.129	211.845
10	0	12.0	FF	0.016	0.004	0.096	174.986
6	0	14.0	FF	0.023	0.006	0.147	237.078
8	0	14.0	FF	0.019	0.005	0.122	197.530
10	0	14.0	FF	0.016	0.004	0.094	173.639
6	0	16.0	FF	0.023	0.006	0.139	237.392
8	0	16.0	FF	0.019	0.005	0.117	202.754
10	0	16.0	FF	0.016	0.004	0.090	178.110
12	0	16.0	FF	0.013	0.003	0.072	155.096
6	0	18.0	FF	0.023	0.006	0.133	229.029
8	0	18.0	FF	0.018	0.005	0.110	214.611
10	0	18.0	FF	0.015	0.004	0.089	184.755
12	0	18.0	FF	0.012	0.003	0.070	159.665
8	0	10.0	TF	0.017	0.004	0.187	273.743
8	0	12.0	TF	0.019	0.005	0.121	197.936
8	0	14.0	TF	0.019	0.005	0.117	196.575
10	0	14.0	TF	0.015	0.004	0.092	183.275
8	0	16.0	TF	0.018	0.005	0.113	204.757
10	0	16.0	TF	0.015	0.004	0.089	183.015
12	0	16.0	TF	0.013	0.003	0.070	151.175
8	0	18.0	TF	0.018	0.005	0.110	210.489
10	0	18.0	TF	0.014	0.004	0.086	180.158
12	0	18.0	TF	0.013	0.003	0.066	151.110
8	0	16.0	TT	0.018	0.007	0.170	253.919
10	0	16.0	TT	0.015	0.006	0.130	205.679
8	0	18.0	TT	0.018	0.007	0.161	246.134
10	0	18.0	TT	0.015	0.006	0.122	200.840



(a) (b)

Figure 5.8: ATE and RME error values for the values in Table 5.9. Obtained using a BRISK-AGAST detector, a FREAK extractor, and a FLANN matcher. Figure 5.8a gives the absolute trajectory error and Figure 5.8b gives the relative motion error.

system encountered difficulty at the peak of the large vertical rotation near the 10 s mark. At the peak of this rotation, the camera captures a drawn shade on a white wall, shown in Frame 200 of Figure 4.6b. Images at this point lack a suitable number of corners which are distinct enough to be taken as interest points using a threshold of fourteen, causing localization to fail.

The FREAK descriptor in combination with the FLANN matcher handled large numbers of features well. False matches due to the sheer number of features began to become a problem with the average of 2027 features per frame at a threshold of six, but even then the absolute trajectory errors incurred were smaller than those found using a threshold of ten or twelve and fewer features. With each scale and normalization setting, a threshold of eight, yielding an average of 1540 features per frame, produced the smallest errors.

Similar to the other BRISK-AGAST detector configurations, `thresh` was inversely proportional to detection, extraction, matching, and total run times. Each increase of `thresh` by two yielded a 4-5 ms increase in detection time per frame, a 1 ms increase in extraction time, a 30 ms increase in matching time, and a 20 s increase in total run time.

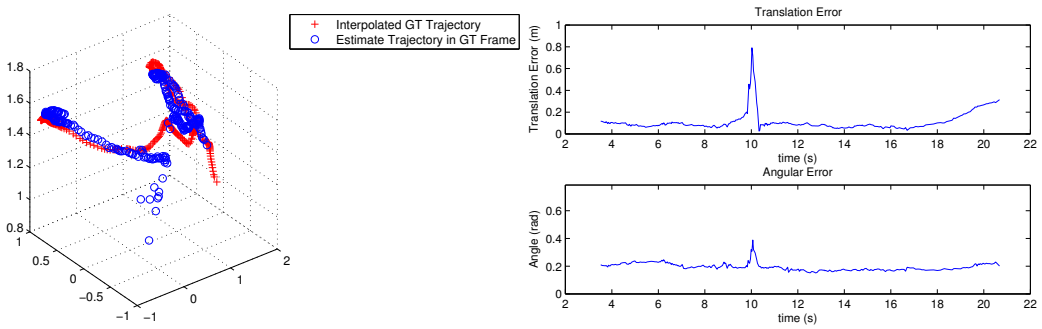
Changes to `patternScale` did not noticeably change mean ATE and RME values or run times in the successfully completed trials, but there was a slight improvement for scales of 14.0 and 18.0. No trials using sampling pattern scales larger than 18.0 were able to successfully run to completion. Trials using these larger scales failed near the 8 s mark when the camera began its

first horizontal rotation. As the camera rotated, descriptors generated using a larger sampling pattern changed more than those using a smaller pattern, resulting in a decrease in the number of matches between frames, a failure to estimate pairwise alignment, and a loss of localization.

Strange behavior occurred at low sampling pattern scales of 6.0, 8.0, and 10.0 when scale normalization and orientation normalization were both enabled. When the camera began its first horizontal rotation at the 8 s mark, the program slowed significantly, taking between 2 s and 3 s to process each frame with the RANSAC based transformation recovery algorithm taking seconds to complete where it normally took tens of milliseconds. Although the exact cause could not be confirmed, it appears that the FLANN matcher selected matching points which appear to be good matches according to the 2-nn distance ratio test described in Section 2.6.4.3, but are in fact false matches with very inconsistent transformations. This could result in the RANSAC algorithm running to its maximum number of iterations for each frame to key-frame comparison and explain the long processing time for each new frame.

Enabling scale normalization slightly improved the RME and ATE metrics but did not alter run time significantly. Enabling both orientation and scale normalization resulted in a decrease from fourteen successful runs to four successful runs, and a substantial increase in the error terms for the runs that were successful.

The best results were found using a `thresh` value of eight, an `nOctaves` value of zero, and a `patternScale` value of 14.0 with scale normalization enabled and orientation normalization disabled. This run produced one of the smallest ATE of 0.116 ± 0.096 m and 0.196 ± 0.028 rad with a run time of 196.575 s. The trajectory and ATE metrics for this run are shown in Figure 5.9. The system was able to successfully map the horizontal rotation at the 8 s mark while incurring no significant error, but encountered some difficulty with poor odometry during the vertical rotation at the 10 s mark. This can be seen by the spike in translation and angular error at 10 s and the break in the estimate trajectory. Once the vertical rotation was completed, the system was able to recover, perform loop closure, and continue with relatively little error for the rest of the run.



(a) (b)

Figure 5.9: The estimate trajectory, shown with blue circles, compared to the ground truth trajectory, shown with red crosses, is shown in Figure 5.9a. ATE metrics for the run using `thresh` = 8, `nOctaves` = 0, `patternScale` = 15, ON disabled, and SN enabled are shown in Figure 5.9b.

5.8 Adjustable BRISK-AGAST and BRISK with Brute-Force

Results using the BRISK-AGAST detector show that any benefits from using a faster AGAST based detector and binary descriptors are offset by the low threshold values required to guarantee the minimum number of features necessary for continuous localization. The Adjustable BRISK-AGAST detector seeks to remedy this problem by allowing the user to specify a desired number of features. The detector then dynamically changes the threshold value based on the number of features detected in the most recent frame so as to maintain a relatively constant feature count at the desired level. This approach takes steps to ensure that there are always enough features to perform localization, while limiting the number of excessive features to curtail false matches and reduce total run time.

The first configuration tested which utilized the Adjustable BRISK-AGAST detector was the BRISK descriptor extractor with a brute-force matcher. This setup had four open parameters, `fMin` and `fMax`, the feature bounds specifying the desired number of features, `nOctaves`, specifying the size of the detector's scale stack, and `patternScale`, specifying the size of the descriptor sampling pattern. Cursory testing showed that limiting the difference in the feature bounds to one produced the best result. More complete testing was performed using feature bounds of 200-201, 300-301, 400-401, 500-501, 600-601, and 700-701. Values of`nOctaves` of zero, one, two, and

Table 5.11: First pass ATE and RME results for SLAM using an Adjustable BRISK-AGAST detector, a BRISK descriptor extractor, and a brute-force-Hamming matcher.

Feat. Bnd.	Octaves	Scale	Avg. Feat.	RME Trans. (m)	RME Ang. (rad)	ATE Trans. (m)	ATE Ang. (rad)
600-601	0	0.750	617.4	0.020 ± 0.039	0.014 ± 0.017	0.105 ± 0.079	0.165 ± 0.048
700-701	0	0.750	720.2	0.019 ± 0.049	0.014 ± 0.017	0.086 ± 0.069	0.146 ± 0.035
800-801	0	0.750	811.3	0.020 ± 0.036	0.015 ± 0.014	0.169 ± 0.111	0.162 ± 0.063
300-301	0	1.000	317.1	0.031 ± 0.067	0.020 ± 0.029	0.141 ± 0.114	0.172 ± 0.052
400-401	0	1.000	417.3	0.021 ± 0.034	0.015 ± 0.015	0.134 ± 0.098	0.142 ± 0.045
500-501	0	1.000	515.1	0.022 ± 0.042	0.015 ± 0.017	0.174 ± 0.129	0.167 ± 0.087
600-601	0	1.000	618.1	0.017 ± 0.026	0.013 ± 0.010	0.112 ± 0.075	0.142 ± 0.042
700-701	0	1.000	721.7	0.017 ± 0.025	0.013 ± 0.012	0.103 ± 0.083	0.136 ± 0.037
800-801	0	1.000	816.3	0.015 ± 0.019	0.012 ± 0.010	0.091 ± 0.060	0.120 ± 0.036
300-301	0	1.250	317.3	0.025 ± 0.052	0.016 ± 0.018	0.134 ± 0.133	0.233 ± 0.062
400-401	0	1.250	414.0	0.023 ± 0.042	0.016 ± 0.021	0.202 ± 0.107	0.189 ± 0.099
500-501	0	1.250	517.0	0.019 ± 0.034	0.014 ± 0.014	0.181 ± 0.128	0.152 ± 0.076
600-601	0	1.250	626.5	0.019 ± 0.034	0.014 ± 0.013	0.125 ± 0.085	0.137 ± 0.041
700-701	0	1.250	720.0	0.016 ± 0.019	0.013 ± 0.011	0.130 ± 0.074	0.141 ± 0.040
800-801	0	1.250	809.7	0.016 ± 0.021	0.013 ± 0.011	0.092 ± 0.063	0.145 ± 0.032
700-701	1	0.750	692.2	0.018 ± 0.024	0.014 ± 0.011	0.090 ± 0.055	0.201 ± 0.051
800-801	1	0.750	780.0	0.021 ± 0.037	0.016 ± 0.017	0.106 ± 0.071	0.168 ± 0.030
500-501	1	1.000	504.2	0.018 ± 0.024	0.013 ± 0.010	0.093 ± 0.066	0.148 ± 0.032
600-601	1	1.000	600.0	0.016 ± 0.016	0.013 ± 0.009	0.125 ± 0.079	0.236 ± 0.072
700-701	1	1.000	690.7	0.016 ± 0.025	0.013 ± 0.011	0.104 ± 0.077	0.183 ± 0.044
800-801	1	1.000	776.6	0.016 ± 0.023	0.013 ± 0.010	0.098 ± 0.077	0.225 ± 0.049
500-501	1	1.250	500.7	0.017 ± 0.022	0.014 ± 0.010	0.109 ± 0.085	0.183 ± 0.038
600-601	1	1.250	597.3	0.017 ± 0.024	0.013 ± 0.010	0.078 ± 0.069	0.122 ± 0.041
700-701	1	1.250	689.5	0.018 ± 0.030	0.013 ± 0.010	0.132 ± 0.100	0.301 ± 0.054
800-801	1	1.250	771.3	0.017 ± 0.020	0.014 ± 0.010	0.109 ± 0.067	0.209 ± 0.039
500-501	2	1.000	500.3	0.023 ± 0.038	0.016 ± 0.014	0.131 ± 0.113	0.163 ± 0.036
600-601	2	1.000	592.4	0.021 ± 0.030	0.015 ± 0.012	0.180 ± 0.084	0.129 ± 0.049
700-701	2	1.000	681.9	0.020 ± 0.027	0.014 ± 0.010	0.164 ± 0.090	0.260 ± 0.034
800-801	2	1.000	755.7	0.023 ± 0.055	0.016 ± 0.022	0.166 ± 0.084	0.160 ± 0.058
500-501	2	1.250	496.7	0.020 ± 0.025	0.015 ± 0.011	0.139 ± 0.109	0.166 ± 0.052
600-601	2	1.250	596.2	0.020 ± 0.027	0.015 ± 0.011	0.155 ± 0.093	0.137 ± 0.069
700-701	2	1.250	677.1	0.020 ± 0.024	0.014 ± 0.010	0.146 ± 0.102	0.239 ± 0.038
800-801	2	1.250	754.5	0.020 ± 0.028	0.015 ± 0.011	0.111 ± 0.092	0.156 ± 0.036
700-701	3	1.000	680.8	0.026 ± 0.503	0.017 ± 0.018	0.123 ± 0.129	0.153 ± 0.041

three were tested as well as `patternScale` values of 0.75, 1.0, and 1.25. The results of successful trials using these values are recorded in Table 5.11 and Table 5.12. A second round of testing was performed to refine the settings, using feature bounds of 300-301 to 400-401 in increments of 10, an `nOctaves` count of zero, and `patternScale` values of 1.0, 1.1, and 1.25. The results of this second round are recorded in Table 5.13 and Table 5.14. The average ATE and RME values for all the trials are plotted in Figure 5.10a.

Testing showed that the RGB-D SLAM System was able to successfully run to completion using all the feature bounds except for the 200-201 range. At the 200-201 range, the system had difficulty finding enough inter-frame feature matches to recover odometry, typically failing at the start of the video sequence or the beginning of the first camera rotation. Increasing the feature count generally resulted in decreases in the ATE translation and orientation errors on the order of 0.01 m and 0.02 rad for each 100 feature increase. Each increase in the feature bounds also resulted in an across the board increase in detection, extraction, matching, and aggregate run times. Each 100 feature increment resulted in an approximately 1 ms growth in detection

Table 5.12: First pass timing results for SLAM using an Adjustable BRISK-AGAST feature detector, a BRISK descriptor extractor, and a brute-force-Hamming matcher. Detection, extraction, and matching times are given in seconds per frame.

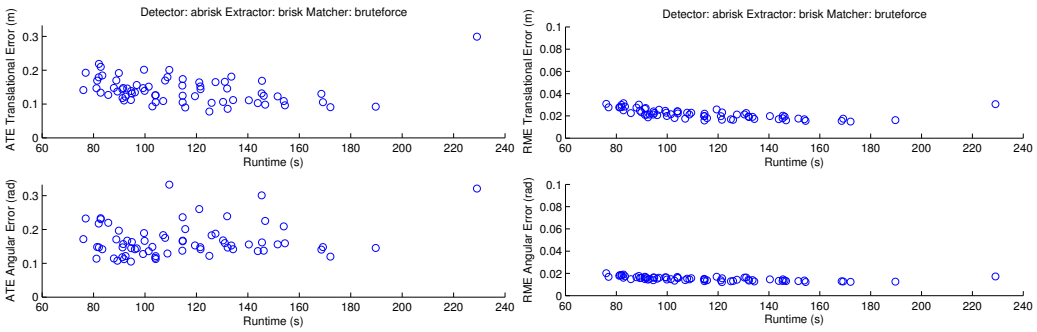
Feat. Bnd.	Octaves	Scale	Detect Time (s)	Extract Time (s)	Match Time (s)	Run time (s)
600-601	0	0.750	0.008	0.005	0.038	114.666
700-701	0	0.750	0.009	0.005	0.051	132.141
800-801	0	0.750	0.010	0.006	0.067	145.495
300-301	0	1.000	0.004	0.003	0.010	76.053
400-401	0	1.000	0.005	0.003	0.017	96.074
500-501	0	1.000	0.006	0.004	0.026	114.676
600-601	0	1.000	0.008	0.005	0.038	134.272
700-701	0	1.000	0.008	0.005	0.048	143.877
800-801	0	1.000	0.010	0.006	0.067	172.104
300-301	0	1.250	0.004	0.003	0.010	82.781
400-401	0	1.250	0.005	0.003	0.017	99.655
500-501	0	1.250	0.006	0.004	0.027	133.590
600-601	0	1.250	0.008	0.004	0.037	146.175
700-701	0	1.250	0.008	0.005	0.051	168.586
800-801	0	1.250	0.009	0.006	0.064	189.741
700-701	1	0.750	0.020	0.006	0.053	115.677
800-801	1	0.750	0.024	0.006	0.068	130.393
500-501	1	1.000	0.014	0.004	0.028	102.884
600-601	1	1.000	0.016	0.005	0.040	114.634
700-701	1	1.000	0.019	0.005	0.052	125.907
800-801	1	1.000	0.022	0.006	0.066	146.741
500-501	1	1.250	0.013	0.004	0.027	107.102
600-601	1	1.250	0.016	0.005	0.039	125.036
700-701	1	1.250	0.018	0.005	0.051	145.405
800-801	1	1.250	0.022	0.006	0.066	153.981
500-501	2	1.000	0.018	0.004	0.029	94.899
600-601	2	1.000	0.022	0.005	0.039	108.758
700-701	2	1.000	0.026	0.006	0.051	121.122
800-801	2	1.000	0.031	0.006	0.064	131.064
500-501	2	1.250	0.017	0.004	0.027	99.880
600-601	2	1.250	0.021	0.005	0.039	114.521
700-701	2	1.250	0.025	0.005	0.051	131.970
800-801	2	1.250	0.029	0.006	0.062	140.398
700-701	3	1.000	0.029	0.006	0.050	119.464

Table 5.13: ATE and RME results for the refinement pass of the Adjustable BRISK-AGAST detector, BRISK descriptor extractor, and brute-force Matcher.

Feat. Bnd.	Octaves	Scale	Avg. Feat.	RME Trans. (m)	RME Ang. (rad)	ATE Trans. (m)	ATE Ang. (rad)
300-301	0	1.000	317.1	0.028 ± 0.056	0.017 ± 0.020	0.192 ± 0.134	0.232 ± 0.073
310-311	0	1.000	328.2	0.028 ± 0.074	0.018 ± 0.028	0.147 ± 0.103	0.114 ± 0.069
320-321	0	1.000	337.4	0.030 ± 0.063	0.019 ± 0.025	0.179 ± 0.150	0.218 ± 0.070
330-331	0	1.000	346.2	0.028 ± 0.055	0.018 ± 0.028	0.169 ± 0.113	0.148 ± 0.083
340-341	0	1.000	359.8	0.023 ± 0.042	0.015 ± 0.014	0.127 ± 0.085	0.220 ± 0.051
350-351	0	1.000	371.6	0.027 ± 0.058	0.017 ± 0.020	0.148 ± 0.095	0.114 ± 0.060
360-361	0	1.000	378.4	0.026 ± 0.060	0.017 ± 0.024	0.144 ± 0.103	0.118 ± 0.055
370-371	0	1.000	390.7	0.026 ± 0.058	0.016 ± 0.021	0.156 ± 0.116	0.144 ± 0.063
380-381	0	1.000	397.7	0.021 ± 0.035	0.014 ± 0.013	0.112 ± 0.092	0.105 ± 0.038
390-391	0	1.000	406.9	0.019 ± 0.025	0.014 ± 0.012	0.122 ± 0.075	0.122 ± 0.051
310-311	0	1.100	329.7	0.028 ± 0.053	0.018 ± 0.023	0.219 ± 0.165	0.147 ± 0.086
320-321	0	1.100	341.4	0.031 ± 0.069	0.019 ± 0.026	0.210 ± 0.136	0.230 ± 0.075
330-331	0	1.100	346.3	0.028 ± 0.061	0.017 ± 0.023	0.184 ± 0.131	0.142 ± 0.059
340-341	0	1.100	361.6	0.027 ± 0.053	0.017 ± 0.020	0.147 ± 0.104	0.157 ± 0.082
350-351	0	1.100	373.5	0.025 ± 0.058	0.016 ± 0.019	0.137 ± 0.095	0.108 ± 0.049
360-361	0	1.100	387.3	0.021 ± 0.033	0.015 ± 0.012	0.118 ± 0.092	0.147 ± 0.038
370-371	0	1.100	389.3	0.021 ± 0.036	0.016 ± 0.020	0.111 ± 0.091	0.112 ± 0.045
380-381	0	1.100	401.1	0.024 ± 0.043	0.016 ± 0.019	0.139 ± 0.091	0.145 ± 0.044
390-391	0	1.100	408.2	0.025 ± 0.044	0.017 ± 0.018	0.147 ± 0.107	0.128 ± 0.055
310-311	0	1.250	331.1	0.030 ± 0.060	0.018 ± 0.022	0.170 ± 0.131	0.171 ± 0.050
330-331	0	1.250	348.3	0.023 ± 0.041	0.016 ± 0.016	0.192 ± 0.159	0.197 ± 0.071
350-351	0	1.250	372.5	0.021 ± 0.038	0.015 ± 0.015	0.146 ± 0.107	0.167 ± 0.043
360-361	0	1.250	382.9	0.024 ± 0.056	0.017 ± 0.025	0.126 ± 0.111	0.116 ± 0.056
370-371	0	1.250	385.0	0.021 ± 0.039	0.015 ± 0.016	0.151 ± 0.112	0.136 ± 0.046
380-381	0	1.250	399.7	0.024 ± 0.043	0.016 ± 0.017	0.105 ± 0.093	0.121 ± 0.046
390-391	0	1.250	408.6	0.022 ± 0.041	0.016 ± 0.019	0.126 ± 0.089	0.113 ± 0.047

Table 5.14: Timing results for the refinement pass of the configuration using an Adjustable BRISK-AGAST detector, a BRISK descriptor extractor, and a brute-force matcher. Detection, extraction, and matching times are the average time required to perform the operation on a single frame.

Feat. Bnd.	Octaves	Scale	Detect Time (s)	Extract Time (s)	Match Time (s)	Run time (s)
300-301	0	1.000	0.004	0.003	0.010	76.945
310-311	0	1.000	0.004	0.003	0.011	81.160
320-321	0	1.000	0.005	0.003	0.012	82.036
330-331	0	1.000	0.005	0.003	0.012	81.403
340-341	0	1.000	0.005	0.003	0.012	85.734
350-351	0	1.000	0.005	0.003	0.013	87.910
360-361	0	1.000	0.005	0.003	0.014	91.207
370-371	0	1.000	0.005	0.003	0.016	96.787
380-381	0	1.000	0.005	0.003	0.016	94.552
390-391	0	1.000	0.005	0.003	0.016	92.476
310-311	0	1.100	0.004	0.003	0.011	82.083
320-321	0	1.100	0.004	0.003	0.012	82.808
330-331	0	1.100	0.004	0.003	0.012	83.435
340-341	0	1.100	0.005	0.003	0.012	91.555
350-351	0	1.100	0.005	0.003	0.013	89.268
360-361	0	1.100	0.005	0.003	0.015	91.300
370-371	0	1.100	0.005	0.003	0.015	91.859
380-381	0	1.100	0.005	0.003	0.015	94.646
390-391	0	1.100	0.005	0.003	0.017	99.260
310-311	0	1.250	0.004	0.003	0.010	88.864
330-331	0	1.250	0.005	0.003	0.012	89.840
350-351	0	1.250	0.005	0.003	0.013	93.098
360-361	0	1.250	0.005	0.003	0.015	103.980
370-371	0	1.250	0.005	0.003	0.015	101.453
380-381	0	1.250	0.005	0.003	0.015	104.176
390-391	0	1.250	0.005	0.003	0.016	104.174



(a) (b)

Figure 5.10: ATE and RME error values for the trials in Table 5.11. Results obtained using an Adjustable BRISK-AGAST detector, a BRISK extractor, and a brute-force matcher. Figure 5.10a gives the mean absolute trajectory error compared to the total run time for each parameter setting and Figure 5.10b gives the mean relative motion error compared to the total run time.

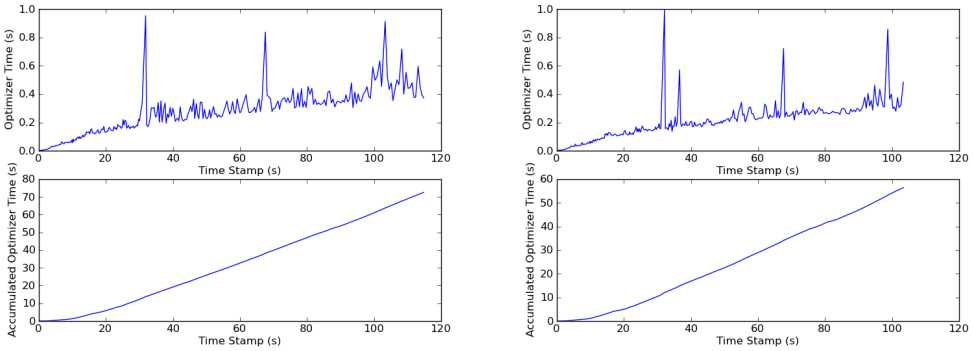
and extraction times per frame, a 10 ms addition to matching times, and a 20 s increase to total run time. Such large increases in run time for relatively small gains in trajectory accuracy make feature bounds around the 300 to 400 mark preferable.

The adjustable detector made a wider range of scale stacks viable. The RGB-D SLAM System was able to consistently run to completion utilizing `nOctaves` values of zero, one, and two, however larger feature bounds were needed for non-zero `nOctaves` values. In trials utilizing non-zero `nOctaves` values with feature bounds smaller than 500-501, the system typically failed around the time of the first horizontal rotation. With the larger scale stack, the detector was unable to react to the changing scene quickly enough in order to supply a sufficient number of features. This resulted in a decrease in the number of matches, a failure to recover the transformation between poses, and ultimately a failure of the RGB-D SLAM System program.

Increasing `nOctaves` from zero to one resulted in a 12 ms increase to average detection times and a 20 ms increase to average matching times, but a 20 s decrease to total run time. This decrease is mostly explained by an increase in the accuracy of interest point locations when `nOctaves` is increased to one. The more accurately localized features result in more consistent descriptors, better inter-frame matches and better initial pose estimates. This, in turn, means less time must be spent optimizing the pose graph resulting in a shorter total run time. Figure 5.11 illustrates this difference in optimization time for two trials using a scale of 1.0 and 500-501 feature bounds. When the value of `nOctaves` is increased from zero to one, the total pose graph optimization time drops from 72.5 s to 56.4 s.

The increase in `nOctaves` from zero to one resulted in a slight decrease in the trajectory error. Subsequent increases to `nOctaves` resulted in an increase in trajectory error. Although the slight improvement to the trajectory estimate and the relative improvement to run time using an `nOctaves` value of one is promising, the inability to use this setting with features bounds in the 300 to 400 range means that a detector operating at that range with an `nOctaves` value of zero will be more efficient.

Altering `patternScale` from 1.0 had no perceivable benefit. At a sampling pattern scale of 0.75, the RGB-D SLAM System failed for feature bounds below 600-601 and `nOctaves` values larger than one. Increasing `patternScale` to 1.25 increases total run time by an average of 11 s and made no noticeable



(a) (b)

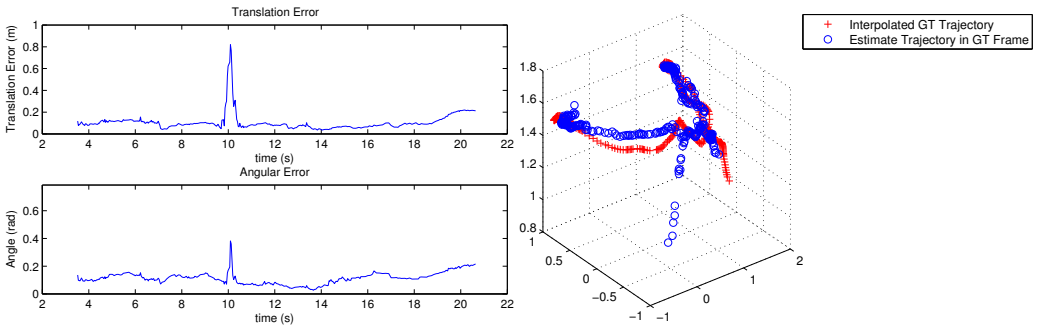
Figure 5.11: Pose graph instantaneous and accumulated optimization time for an Adjustable BRISK-AGAST detector, a BRISK extractor, and a brute-force matcher with a feature bounds of 500-501 and `patternScale` of 1.0. Figure 5.11a shows detection time for `nOctaves` = 0 and Figure 5.11b shows detection time for `nOctaves` = 1.

improvements to the trajectory estimate. As such, it is recommended that `patternScale` be left near 1.0 with this feature detection configuration.

The best results for this configuration were found using feature bounds of 370-371, an `nOctaves` value of zero, and a `patternScale` value of 1.1. With these settings, the absolute trajectory error was found to be 0.111 ± 0.091 m and 0.112 ± 0.045 rad, with a total run time of 91.86 s. Figure 5.12 shows the absolute trajectory error for this run. The system handled the horizontal rotations at the 8 s and 12 s marks well. It encountered difficulty at the 10 s mark, where it interpreted the vertical rotation of the camera as a vertical translation. This is illustrated by the difference in height of the ground truth and estimate poses near the middle of the trajectory plot (Figure 5.12b). The rest of the run was handled well, but error began to accumulate at around the 18 s when the camera rotated to the right and downward.

5.9 Adjustable BRISK-AGAST and FREAK with Brute-Force

The RGB-D SLAM System configuration using an Adjustable BRISK-AGAST detector with a FREAK descriptor extractor and a brute-force matcher had six open parameters: `fMin` and `fMax`, the feature bounds, `nOctaves`, the



(a) (b)

Figure 5.12: ATE error values and estimated trajectory for the best trial using an Adjustable BRISK-AGAST detector, a BRISK extractor, and a brute-force matcher run with `nOctaves` = 0, `patternScale` = 1.1 and a feature bound of 370-371. Figure 5.12a shows the instantaneous trajectory errors for the entire run. Figure 5.12b shows the estimated trajectory with blue circles compared to the ground truth trajectory with red crosses.

number of octaves, `patternScale`, the scale of the FREAK sampling pattern used for descriptor construction, and `orientationNormalization` and `scaleNormalization`, two Boolean variables toggling whether the sampling pattern is rotated and scaled in accordance with the feature's orientation and scale. This configuration was tested with `patternScale` values ranging from 8.0 to 24.0, feature bounds covering ranges from 300 to 700 features, `nOctaves` values of zero and one, and settings with both the orientation and scale normalization disabled, only scale normalization enabled, and both normalization terms enabled. The results of the successful runs are recorded in Table 5.15 and Table 5.16. Plots of the mean ATE and RME values for each successful run are shown in Figure 5.13.

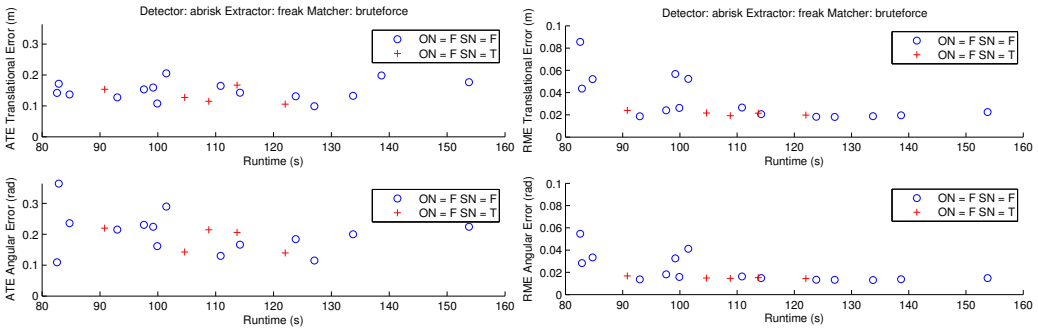
This configuration did not operate well with non-zero `nOctaves` values. Most trials using an `nOctaves` value of one failed at the first rotation due to a lack of matching features. Interestingly, trials using an `nOctaves` value of one were able to operate using a lower pattern scale than those using a value of zero. Three of the four successful runs using an `nOctaves` of one occurred with a small sampling `patternScale` of 8.0. Trials using an `nOctaves` value of zero were unable to make it past the first camera rotation at this scale. The RGB-D SLAM System using an `nOctaves` value of one operated at a limited range of feature values between 500 and 700 and had higher than average run times as a result. The error metrics for most of the trials using

Table 5.15: ATE and RME results for an Adjustable BRISK-AGAST detector with a FREAK descriptor extractor and a brute-force matcher.

Feat. Bnd.	Oct.	Scale	SN/ON	Avg. Feat.	RME Trans. (m)	RME Ang. (rad)	ATE Trans. (m)	ATE Ang. (rad)
500-600	1	8.0	FF	547.6	0.020 ± 0.025	0.014 ± 0.012	0.198 ± 0.099	0.253 ± 0.075
600-700	1	8.0	FF	644.3	0.023 ± 0.039	0.015 ± 0.015	0.176 ± 0.091	0.224 ± 0.097
600-700	0	12.0	FF	662.6	0.021 ± 0.039	0.015 ± 0.017	0.143 ± 0.090	0.166 ± 0.041
400-500	0	15.0	FF	458.1	0.019 ± 0.032	0.014 ± 0.013	0.128 ± 0.112	0.215 ± 0.047
500-501	0	15.0	FF	515.1	0.024 ± 0.055	0.018 ± 0.035	0.153 ± 0.100	0.230 ± 0.030
500-600	0	15.0	FF	558.3	0.018 ± 0.033	0.013 ± 0.014	0.131 ± 0.093	0.184 ± 0.040
600-700	0	15.0	FF	662.6	0.018 ± 0.035	0.013 ± 0.013	0.099 ± 0.083	0.115 ± 0.028
600-700	1	15.0	FF	644.1	0.019 ± 0.038	0.013 ± 0.013	0.132 ± 0.091	0.200 ± 0.064
500-501	0	16.0	FF	515.1	0.026 ± 0.065	0.016 ± 0.021	0.108 ± 0.090	0.161 ± 0.049
500-501	0	18.0	FF	515.1	0.027 ± 0.068	0.016 ± 0.024	0.165 ± 0.121	0.130 ± 0.061
400-401	0	20.0	FF	416.1	0.052 ± 0.139	0.033 ± 0.088	0.137 ± 0.122	0.236 ± 0.073
500-501	0	20.0	FF	513.5	0.057 ± 0.190	0.033 ± 0.079	0.159 ± 0.187	0.224 ± 0.068
400-401	0	22.0	FF	414.2	0.086 ± 0.302	0.055 ± 0.203	0.142 ± 0.247	0.109 ± 0.166
600-700	1	8.0	TF	642.6	0.019 ± 0.028	0.015 ± 0.011	0.115 ± 0.089	0.214 ± 0.036
600-700	0	12.0	TF	662.6	0.021 ± 0.044	0.015 ± 0.022	0.167 ± 0.115	0.206 ± 0.056
400-500	0	15.0	TF	458.1	0.024 ± 0.053	0.017 ± 0.021	0.154 ± 0.119	0.220 ± 0.031
500-600	0	15.0	TF	560.5	0.022 ± 0.051	0.015 ± 0.018	0.127 ± 0.100	0.143 ± 0.037
600-700	0	15.0	TF	662.6	0.020 ± 0.047	0.014 ± 0.019	0.105 ± 0.069	0.139 ± 0.031

Table 5.16: Timing results for RGB-D SLAM using an Adjustable BRISK-AGAST detector, a FREAK descriptor extractor, and a brute-force matcher. Detection, extraction, and matching times are given in seconds per frame.

Feat. Bnd.	Octaves	Scale	SN/ON	Detect Time (s)	Extract Time (s)	Match Time (s)	Run time (s)
500-600	1	8.0	FF	0.016	0.002	0.035	138.658
600-700	1	8.0	FF	0.018	0.002	0.048	153.775
600-700	0	12.0	FF	0.009	0.002	0.047	114.198
400-500	0	15.0	FF	0.006	0.002	0.023	92.999
500-501	0	15.0	FF	0.007	0.002	0.026	97.597
500-600	0	15.0	FF	0.007	0.002	0.034	123.837
600-700	0	15.0	FF	0.009	0.002	0.048	127.055
600-700	1	15.0	FF	0.019	0.002	0.048	133.743
500-501	0	16.0	FF	0.007	0.002	0.026	99.898
500-501	0	18.0	FF	0.006	0.002	0.026	110.865
400-401	0	20.0	FF	0.006	0.002	0.017	84.743
500-501	0	20.0	FF	0.007	0.002	0.027	99.194
400-401	0	22.0	FF	0.006	0.002	0.017	82.585
600-700	1	8.0	TF	0.019	0.002	0.048	108.815
600-700	0	12.0	TF	0.009	0.002	0.048	113.702
400-500	0	15.0	TF	0.006	0.002	0.023	90.816
500-600	0	15.0	TF	0.007	0.002	0.034	104.644
600-700	0	15.0	TF	0.008	0.002	0.047	122.012



(a) (b)

Figure 5.13: ATE and RME error values for the entries in Table 5.15. Obtained using an Adjustable BRISK-AGAST detector, a FREAK extractor, and a brute-force matcher. Figure 5.13a gives the mean absolute trajectory error compared to the total run time for each parameter setting and Figure 5.13b gives the mean relative motion error compared to the total run time.

this `nOctaves` value were worse than average. Although a trial using an `nOctaves` value of one, feature bounds of 600-700, a sampling pattern scale of 8.0, and SN enabled had better than average run time and error results, the failure of trials using similar settings with one octave, indicates that the system using a detector with an `nOctaves` value of one is inconsistent in this configuration. Therefore, it is best to use an `nOctaves` value of zero.

Successful trials occurred using feature bounds between 400 and 700. Trials with feature counts in the 300-400 range experienced difficulty in recovering the transformation between frames. Trials with counts near 300 features per frame failed completely and were unable to track any motion at all, while trials with averages near 350 features per frame had difficulty tracking the rotation of the camera and failed near the 8 s mark.

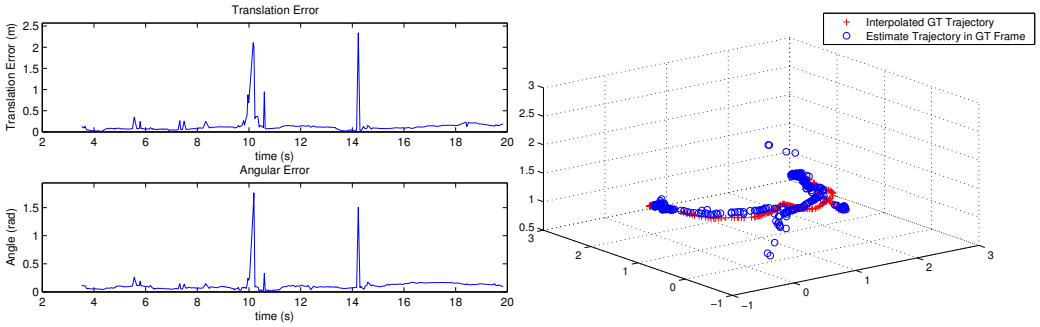
For those successful runs, the average number of features per frame did not appear to play a significant role in the quality of the trajectory estimate. Trials with averages of 410 features per frame produced errors as small as those of 660 average feature runs. As with the other Adjustable BRISK-AGAST configurations, increasing the feature bounds resulted in an increase to detection, matching, and total run times. As such, it is best to use feature bounds near the 400-500 range, thereby keeping run times low without harming the quality of the pose estimates.

This SLAM configuration worked best using `patternScale` values between

12.0 and 22.0. Trials using low values in the 8.0-12.0 range worked well with higher features counts of around 550 to 650 features per frame, while trials using `patternScale`'s between 18.0 and 22.0 worked better with lower feature counts of around 410 to 510 features per frame. A sampling pattern scale of 15.0 worked well with a wide range of feature counts from 450 to 650, but experienced difficulty tracking camera rotations with feature bounds near 410 features per frame. In general, it is best to use a scale of around 15.0 since it provides the most robust performance, but a scale in the 20.0 to 22.0 range should be used for best results if the average feature count is expected to be lower than 450.

Orientation normalization is best left disabled. All runs using orientation normalization failed completely, with trials using smaller feature bounds failing to recover any odometry at all and trials using bounds around 600-700 features per frame failing at the 8 s mark when the camera began to rotate. Enabling scale normalization resulted in a 10 s improvement to run times for most configurations and resulted in no noticeable change to the error metrics. Only runs using a `patternScale` value of 15.0 were able to consistently complete the entire test sequence with scale normalization enabled. For runs with average feature counts in the 450-650 range, scale normalization should be enabled and a `patternScale` of 15.0 should be used, but normalization should be turned off for runs with a smaller feature count.

The best results for this detector, extractor, and matcher configuration were found using feature bounds of 400-401, an `nOctaves` value of zero, a `patternScale` value of 22.0, and orientation and scale normalization disabled. With these settings, an ATE of 0.142 ± 0.247 m and 0.109 ± 0.166 rad was achieved with a run time of 82.585 s. The ATE metrics and the recovered trajectory for the entirety of the run are shown in Figure 5.14. The estimated trajectory followed the ground truth very well. The larger than typical standard deviations are due to several large outliers at the 10 s and 14 s marks. The algorithm misinterpreted the rotation of the camera as translational movement, resulting in the 2 m and 1.5 rad spikes in the error plots and the outliers in the middle of the trajectory estimate.



(a) (b)

Figure 5.14: Results for the best trial using an Adjustable BRISK-AGAST detector, a FREAK descriptor extractor, and a brute-force matcher. Settings of `nOctaves` = 0, feature bounds of 400-401, `patternScale` of 22.0 and SN and ON disabled were used. Figure 5.14a shows the ATE for the entire 21 s run. Figure 5.14b shows the estimated trajectory, indicated with blue circles, compared to the ground truth trajectory, indicated with red crosses.

5.10 Adjustable BRISK-AGAST and BRISK with FLANN

The Adjustable BRISK-AGAST detector, BRISK descriptor extractor, and FLANN matcher configuration for the RGB-D SLAM System has the same four open parameters that the corresponding brute-force configuration has: `fMin` and `fMax`, `nOctaves`, and `patternScale`. This FLANN configuration was tested using feature bounds ranging from 200-201 to 600-601 in increments of 100, `nOctaves` values of zero, one, and two, and `patternScale` values of 0.75, 1.0, 1.25, and 1.5. The results are recorded in Table 5.17 and Table 5.18. The mean ATE and RME values for these runs are plotted in Figure 5.15.

Successful runs were completed using all `nOctaves` settings. As with the brute-force tests, trials using an `nOctaves` setting of zero were able to operate using smaller feature bounds than those with `nOctaves` values of one and two. Runs where the detector used an `nOctaves` value of two only ran to completion with higher `patternScale` settings of 1.25 and 1.5, with runs using a `patternScale` of 0.75 or 1.0 failing due to an inability to find enough feature matches between frames.

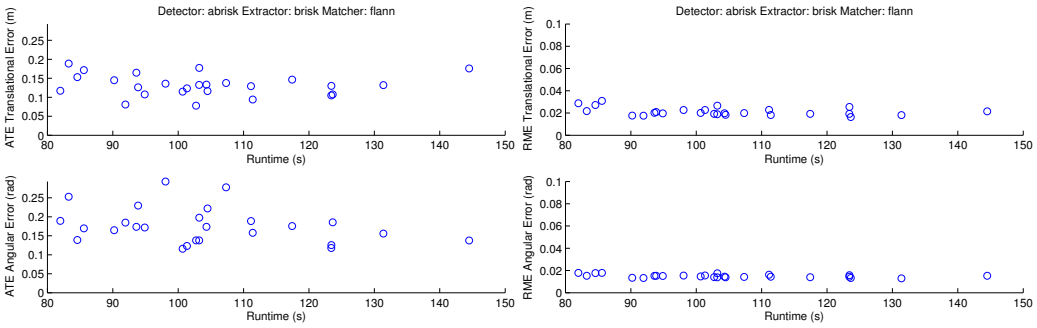
Increasing the value of `nOctaves` resulted in a slight increase in the mean

Table 5.17: AME and RME values for tests using an Adjustable BRISK-AGAST detector, a BRISK descriptor extractor, and a FLANN matcher.

Feat. Bnd.	Octaves	Scale	Avg. Feat.	RME Trans. (m)	RME Ang. (rad)	ATE Trans. (m)	ATE Ang. (rad)
500-501	0	0.750	516.9	0.018 ± 0.022	0.013 ± 0.011	0.081 ± 0.071	0.185 ± 0.025
600-601	0	0.750	617.4	0.019 ± 0.029	0.014 ± 0.013	0.078 ± 0.058	0.138 ± 0.045
600-601	1	0.750	598.9	0.020 ± 0.030	0.015 ± 0.012	0.107 ± 0.091	0.172 ± 0.035
500-501	0	1.000	515.1	0.019 ± 0.031	0.014 ± 0.012	0.133 ± 0.094	0.138 ± 0.055
600-601	0	1.000	618.1	0.019 ± 0.035	0.014 ± 0.016	0.105 ± 0.068	0.118 ± 0.038
400-401	1	1.000	403.8	0.022 ± 0.039	0.015 ± 0.015	0.189 ± 0.134	0.253 ± 0.046
500-501	1	1.000	504.2	0.023 ± 0.055	0.015 ± 0.022	0.136 ± 0.100	0.293 ± 0.088
600-601	1	1.000	600.0	0.020 ± 0.032	0.014 ± 0.014	0.138 ± 0.101	0.278 ± 0.042
300-301	0	1.250	317.3	0.029 ± 0.054	0.018 ± 0.020	0.117 ± 0.122	0.189 ± 0.054
400-401	0	1.250	414.0	0.023 ± 0.043	0.015 ± 0.016	0.123 ± 0.089	0.123 ± 0.044
500-501	0	1.250	517.0	0.019 ± 0.035	0.014 ± 0.015	0.146 ± 0.108	0.176 ± 0.050
600-601	0	1.250	626.5	0.018 ± 0.031	0.013 ± 0.011	0.132 ± 0.094	0.156 ± 0.036
400-401	1	1.250	406.1	0.018 ± 0.023	0.013 ± 0.009	0.145 ± 0.088	0.165 ± 0.051
500-501	1	1.250	500.7	0.018 ± 0.023	0.014 ± 0.010	0.116 ± 0.091	0.222 ± 0.034
600-601	1	1.250	597.3	0.018 ± 0.028	0.014 ± 0.013	0.094 ± 0.069	0.158 ± 0.038
500-501	2	1.250	496.7	0.020 ± 0.027	0.015 ± 0.011	0.165 ± 0.101	0.174 ± 0.052
600-601	2	1.250	596.2	0.020 ± 0.023	0.014 ± 0.009	0.133 ± 0.094	0.174 ± 0.043
300-301	0	1.500	315.4	0.031 ± 0.066	0.018 ± 0.022	0.172 ± 0.127	0.169 ± 0.078
300-301	0	1.500	315.4	0.027 ± 0.051	0.018 ± 0.020	0.153 ± 0.103	0.139 ± 0.056
500-501	0	1.500	516.9	0.025 ± 0.059	0.016 ± 0.021	0.130 ± 0.087	0.125 ± 0.051
600-601	0	1.500	628.6	0.021 ± 0.039	0.015 ± 0.017	0.176 ± 0.115	0.138 ± 0.094
400-401	1	1.500	403.9	0.021 ± 0.038	0.015 ± 0.014	0.126 ± 0.081	0.230 ± 0.063
500-501	1	1.500	504.6	0.020 ± 0.027	0.015 ± 0.011	0.115 ± 0.098	0.116 ± 0.043
600-601	1	1.500	598.0	0.016 ± 0.020	0.013 ± 0.009	0.107 ± 0.086	0.185 ± 0.047
500-501	2	1.500	501.6	0.027 ± 0.047	0.018 ± 0.017	0.177 ± 0.123	0.198 ± 0.035
600-601	2	1.500	589.6	0.023 ± 0.043	0.016 ± 0.017	0.129 ± 0.112	0.189 ± 0.033

Table 5.18: Timing results for an Adjustable BRISK-AGAST detector a BRISK descriptor extractor and a FLANN matcher.

Feat. Bnd.	Octaves	Scale	Detect Time (s)	Extract Time (s)	Match Time (s)	Run time (s)
500-501	0	0.750	0.007	0.004	0.022	91.930
600-601	0	0.750	0.007	0.005	0.028	102.744
600-601	1	0.750	0.016	0.005	0.033	94.892
500-501	0	1.000	0.006	0.004	0.020	103.213
600-601	0	1.000	0.007	0.004	0.025	123.413
400-401	1	1.000	0.011	0.003	0.016	83.260
500-501	1	1.000	0.013	0.004	0.023	98.066
600-601	1	1.000	0.015	0.005	0.030	107.329
300-301	0	1.250	0.004	0.003	0.010	81.985
400-401	0	1.250	0.005	0.003	0.015	101.348
500-501	0	1.250	0.006	0.004	0.019	117.426
600-601	0	1.250	0.007	0.005	0.025	131.379
400-401	1	1.250	0.011	0.003	0.016	90.212
500-501	1	1.250	0.015	0.004	0.021	104.485
600-601	1	1.250	0.015	0.005	0.029	111.403
500-501	2	1.250	0.017	0.004	0.021	93.611
600-601	2	1.250	0.020	0.005	0.028	104.319
300-301	0	1.500	0.004	0.003	0.010	85.577
300-301	0	1.500	0.004	0.003	0.010	84.584
500-501	0	1.500	0.010	0.004	0.018	123.427
600-601	0	1.500	0.007	0.004	0.026	144.497
400-401	1	1.500	0.011	0.003	0.014	93.870
500-501	1	1.500	0.013	0.004	0.020	100.680
600-601	1	1.500	0.015	0.005	0.025	123.628
500-501	2	1.500	0.017	0.004	0.020	103.220
600-601	2	1.500	0.020	0.005	0.027	111.136



(a) (b)

Figure 5.15: ATE and RME values for the entries in Table 5.17. Obtained using an Adjustable BRISK-AGAST detector, a BRISK extractor, and a FLANN matcher. Figure 5.15a gives the mean absolute trajectory error compared to the total run time for each parameter setting. Figure 5.15b gives the relative motion error compared to the total run time.

ATE on the order of 0.03 m and 0.07 rad per unit increase. Incrementing the value of `nOctaves` also resulted in an increase in detection times by around 5 ms per frame, an increase in matching times by 3 ms per frame, and a decrease in total run time by 10 to 15 s. This decrease in time is a result of a slight drop in the number of features detected at each feature bound setting, and smaller pose graph optimization times. Values for `nOctaves` of zero and one are still preferable as they can operate using a wider range of sampling pattern scales and at lower feature counts, resulting in faster overall run times.

Testing showed that at `patternScale` settings of 0.75, the RGB-D SLAM System only operated with an `nOctaves` value of zero and higher feature bounds. Increasing the `patternScale` value to 1.25 from 1.0 resulted in around an 8 s increase to run times, but decreased ATE by an average of 0.05 m and 0.09 rad. Increasing `patternScale` to 1.5 from 1.25 also increased run time by 7 s while making no noticeable improvement to the ATE translation error, but around a 0.02 rad improvement to the orientation error. A `patternScale` setting of 1.0 should be used when speed is the only concern, and a setting of 1.25 or 1.5 is preferable for general operations.

No trials succeeded using feature bounds of 200-201. There was difficulty finding enough matches at this setting resulting in a loss in odometry either at the start of the run, or at the first camera rotation. All other feature bounds resulted in successful runs, although the 300-301 range only worked

with an `nOctaves` value of zero, and an `nOctaves` value of two required feature bounds of at least 500 features per frame to run to completion.

Each 100 feature increment to the feature bounds resulted in a roughly 5 ms per frame increase to detection, extraction, and matching run times, and a 10 to 15 s increase to total run time. Such feature bounds increases resulted in slight improvements to the absolute trajectory errors until the 500-501 feature bound, and then a gradual degradation in the quality of the trajectory estimate afterwards. Feature bounds between 300 and 500 are suitable for this configuration, depending on whether accuracy or speed is preferred.

Although several trials produced comparable results, the trial that best balanced run time and accuracy was the one using feature bounds of 300-301, an `nOctaves` value of zero, and a `patternScale` value of 1.5. This implementation had absolute trajectory errors of 0.153 ± 0.103 m and 0.139 ± 0.056 rad and a total run time of 84.584 s. Figure 5.16 shows the absolute trajectory error for the entirety of this run. As with most RGB-D SLAM System runs, there was difficulty capturing the vertical rotation at the 10 s mark. There was also some low level noise in the ATE terms for the whole run, explainable by the degradation in the quality of FLANN matches compared to a brute-force matcher.

5.11 Adjustable BRISK-AGAST and FREAK with FLANN

The RGB-D SLAM System using an Adjustable BRISK-AGAST detector, a FREAK extractor and a FLANN matcher has six modifiable parameters for feature detection. In its testing, `fMin-fMax` ranges of 200-201 to 600-601 were used in increments of 100, in addition to `patternScale` values between 16.0 and 24.0 in increments of 2.0, `nOctaves` values of zero and one, and normalization settings with both scale and orientation normalization disabled, only scale normalization enabled, and both terms enabled. The results of successful runs using these settings are recorded in Table 5.19 and Table 5.20. The mean ATE and RME values for each run are plotted in Figure 5.17.

Trial runs using feature bounds of 400-401, 500-501, and 600-601 were able

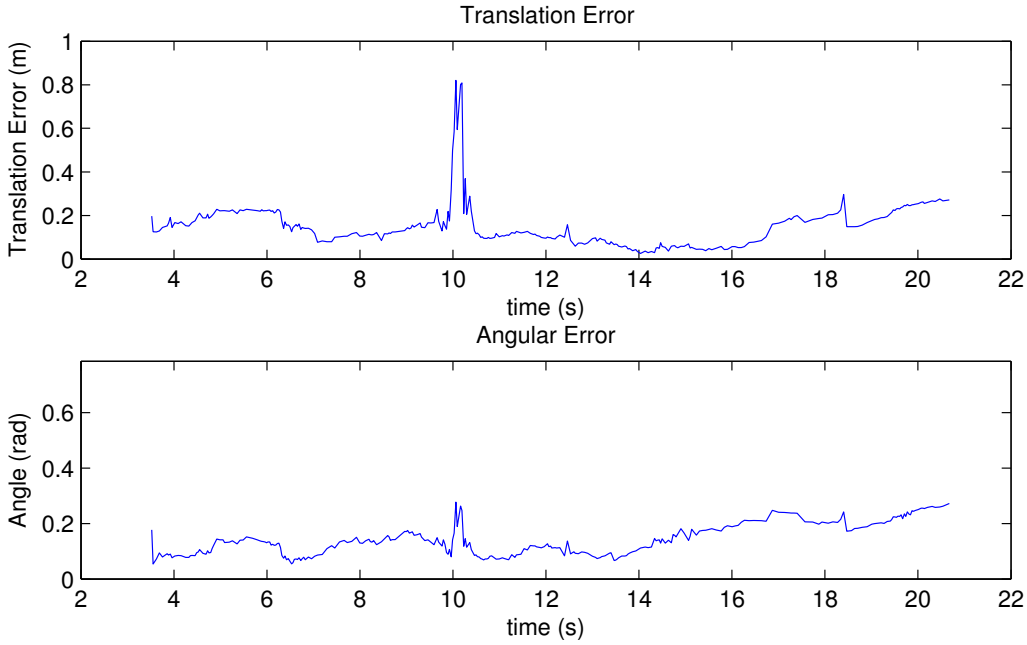


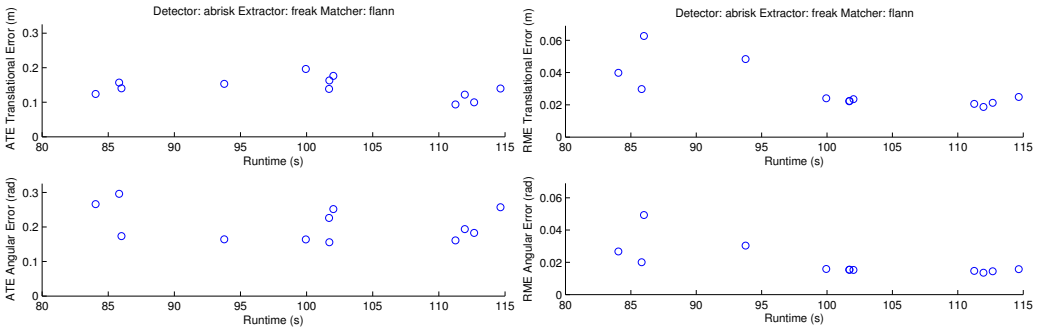
Figure 5.16: ATE translation and orientation errors for the Adjustable BRISK-AGAST, BRISK, FLANN trial run using feature bounds of 300-301, 0 `nOctaves`, and a `patternScale` of 1.5.

Table 5.19: ATE and RME values using an Adjustable BRISK-AGAST detector, a FREAK extractor, and a FLANN matcher.

Feat. Bnd.	Oct.	Scale	SN/ON	Avg. Feat.	RME Trans. (m)	RME Ang. (rad)	ATE Trans. (m)	ATE Ang. (rad)
500-501	0	16.0	FF	515.1	0.024 ± 0.048	0.016 ± 0.018	0.196 ± 0.147	0.164 ± 0.063
600-601	0	16.0	FF	618.1	0.021 ± 0.041	0.015 ± 0.016	0.094 ± 0.069	0.161 ± 0.051
400-401	0	18.0	FF	417.3	0.030 ± 0.066	0.020 ± 0.030	0.157 ± 0.121	0.296 ± 0.094
500-501	0	18.0	FF	515.1	0.023 ± 0.046	0.015 ± 0.018	0.176 ± 0.135	0.252 ± 0.047
600-601	0	18.0	FF	618.1	0.019 ± 0.033	0.014 ± 0.012	0.122 ± 0.073	0.194 ± 0.051
500-501	0	20.0	FF	513.5	0.048 ± 0.174	0.030 ± 0.100	0.153 ± 0.142	0.164 ± 0.100
400-401	0	22.0	FF	414.2	0.040 ± 0.059	0.027 ± 0.036	0.124 ± 0.100	0.266 ± 0.028
500-501	0	16.0	TF	515.1	0.022 ± 0.039	0.015 ± 0.017	0.139 ± 0.111	0.226 ± 0.037
600-601	0	16.0	TF	618.1	0.021 ± 0.043	0.014 ± 0.015	0.100 ± 0.088	0.183 ± 0.046
500-501	0	18.0	TF	515.1	0.022 ± 0.050	0.015 ± 0.018	0.163 ± 0.122	0.156 ± 0.049
600-601	0	18.0	TF	618.1	0.025 ± 0.061	0.016 ± 0.020	0.140 ± 0.110	0.257 ± 0.046
400-401	0	22.0	TF	414.2	0.063 ± 0.197	0.049 ± 0.208	0.141 ± 0.154	0.174 ± 0.150

Table 5.20: Timing results for an Adjustable BRISK-AGAST detector, a FREAK extractor, and a FLANN matcher.

Feat. Bnd.	Octaves	Scale	SN/ON	Detect Time (s)	Extract Time (s)	Match Time (s)	Run time (s)
500-501	0	16.0	FF	0.007	0.002	0.024	99.938
600-601	0	16.0	FF	0.008	0.002	0.032	111.254
400-401	0	18.0	FF	0.005	0.002	0.016	85.825
500-501	0	18.0	FF	0.006	0.002	0.023	102.019
600-601	0	18.0	FF	0.007	0.002	0.030	111.960
500-501	0	20.0	FF	0.007	0.002	0.023	93.776
400-401	0	22.0	FF	0.005	0.002	0.017	84.048
500-501	0	16.0	TF	0.007	0.002	0.024	101.698
600-601	0	16.0	TF	0.008	0.002	0.032	112.666
500-501	0	18.0	TF	0.006	0.002	0.023	101.719
600-601	0	18.0	TF	0.008	0.002	0.030	114.660
400-401	0	22.0	TF	0.006	0.002	0.017	85.999



(a) (b)

Figure 5.17: ATE and RME values for the entries in Table 5.19. Obtained using an Adjustable BRISK-AGAST detector, a FREAK extractor, and a FLANN matcher. Figure 5.17a gives the mean absolute trajectory error compared to the total run time for each parameter setting. Figure 5.17b gives the mean relative motion error compared to the total run time.

to run to completion. Runs using bounds of 200-201 and 300-301 experienced difficulty at the 8 s mark. With these lower feature bounds, enough matching features could not be found, resulting in a failure to track the camera's pose through the horizontal rotation. The 500-501 feature bound was the most consistent, able to function with almost all the `patternScale` settings. In these trials, increasing the feature bounds by increments of 100 netted an average decrease to the ATE metrics of 0.047 m and 0.006 rad, but at the cost of a 1.2 ms per frame increase to detection times, a 7.4 ms per frame increase to matching times, and a 12.6 s increase to total run time. It is best to use as small feature bounds as possible, as the improvement to pose accuracy does not warrant a greater than 10% increase to run time.

No trial runs were successful using an `nOctaves` setting of one. In every case, the detector was unable to supply enough matching features to recover the translation between poses, and tracking failed at the start of the run.

All `patternScale` values functioned adequately with the exception of 24.0. Runs using this larger value had difficulty during the first rotation, with the number of matching features dropping below acceptable levels, and localization failing. Sampling pattern scales of 16.0 and 18.0 were the most consistent, able to operate with most all feature bounds. In general, increasing the scale resulted in a small increase in ATE on the order of 0.03 m and 0.007 rad, and an increase to run time of 1.8 s; however with low feature bounds the greater `patternScale` values performed better, yielding a higher

number of correct matches, and a faster total run time due to a more rapid pose graph optimization. As such, it is best to use a scale of 16.0 with most feature bounds, but a scale of 22.0 for settings where the average number of features per frame is near 400.

Similar to the brute-force matching case, enabling orientation normalization caused all trials to fail. These runs all failed at the start of the first major turn. In each case, no matter the feature bounds, not enough matching features could be found to track the camera's pose through the horizontal rotation and localization failed. In order to achieve any results at all, orientation normalization should be disabled.

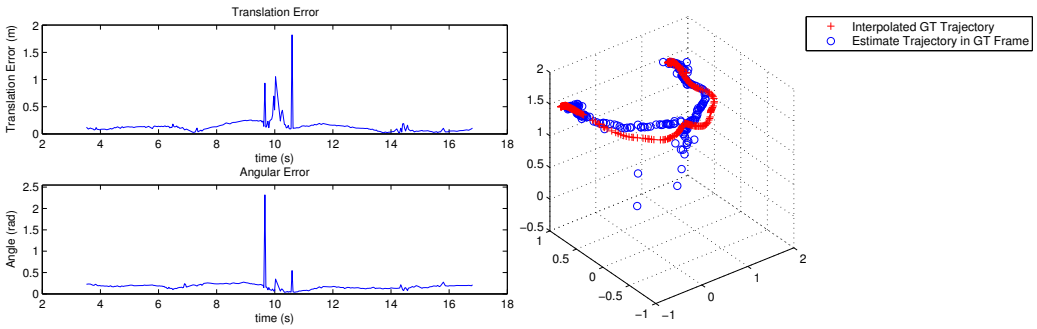
Enabling scale normalization netted a slight overall improvement in the mean ATE with a 0.013 m decrease in the translation term, but a 0.003 rad increase in the orientation term. This improvement came at the cost of an average of a 1.38 s increase to total run times.

The best results were found using feature bounds of 400-401, a `patternScale` value of 22.0, an `nOctaves` value of zero, scale normalization enabled, and orientation normalization disabled. Using these settings, mean absolute trajectory errors of 0.141 ± 0.154 m and 0.174 ± 0.150 rad were found, with a total run time of 88.99 s. Plots of the ATE terms for the entire 21 s run are shown in Figure 5.18. At the conclusion of the first horizontal rotation of the camera at the 9.5 s mark, the error spiked to 1 m and 2.5 rad when the system estimated the view of the desk was from a pose opposite of the true pose. Additional peaks in the error occurred at the 10 s and 10.75 s marks, indicating some trouble tracking the pitch of the camera.

5.12 Comparison of BRISK-AGAST and Adjustable BRISK-AGAST Detectors

The Adjustable BRISK-AGAST detector was an enormous improvement over the standard BRISK-AGAST detector. With each extractor and matcher combination, the adjustable detector reduced total run times, decreased the absolute trajectory errors, and improved the stability in pose localization.

Consider the plots in Figure 5.19. These plots show the mean ATE translation and rotation errors of representative runs using each detector and matcher combination, with BRISK runs indicated by a blue circle and Ad-



(a) (b)

Figure 5.18: Results for the best trial using an Adjustable BRISK-AGAST detector, a FREAK descriptor extractor, and a FLANN matcher. Settings of `nOctaves` = 0, feature bounds of 400-401, `patternScale` of 22.0 and SN enabled and ON disabled were used. Figure 5.18a shows the ATE for the entire 21 s run. Figure 5.18b shows the estimated trajectory, indicated with blue circles, compared to the ground truth trajectory, indicated with red crosses.

justable BRISK runs with a red triangle. These runs were the "best" in each configuration, chosen for rapid run times, small ATE and RME errors, or a combination of the two.

Configurations using a FLANN matcher showed noteworthy improvement when using the adjustable detector. Total run times were reduced by 80-140 s and there was a 0.05-0.2 m improvement to ATE translation error and a 0.2 to 0.4 rad improvement to the rotation error. Trials using the FREAK descriptor were able to reduce run times from 229.0288 s to 85.9990 s and rotation error from 0.1966 rad to 0.1737 rad with only a 0.01 m increase to translation error, from 0.1301 m to 0.1405 m.

The FREAK descriptor configurations using a brute-force matcher showed similar improvements with run time savings of 20-40 s and even better estimation results. The fastest Adjustable BRISK trial at 81.9846 s with a mean error of 0.1172 m and 0.1894 rad was faster and more accurate than the most accurate BRISK detector run at 108.0683 s, 0.2705 m, and 0.3682 rad, an improvement of over 26 s, 16 cm, and 10 °.

The BRISK descriptor and brute-force matcher combination showed the most improvement. Although there was no significant change in overall run times, the adjustable detector expanded the values of `patternScale` and `nOctaves` that RGB-D SLAM could operate at dramatically, and improved

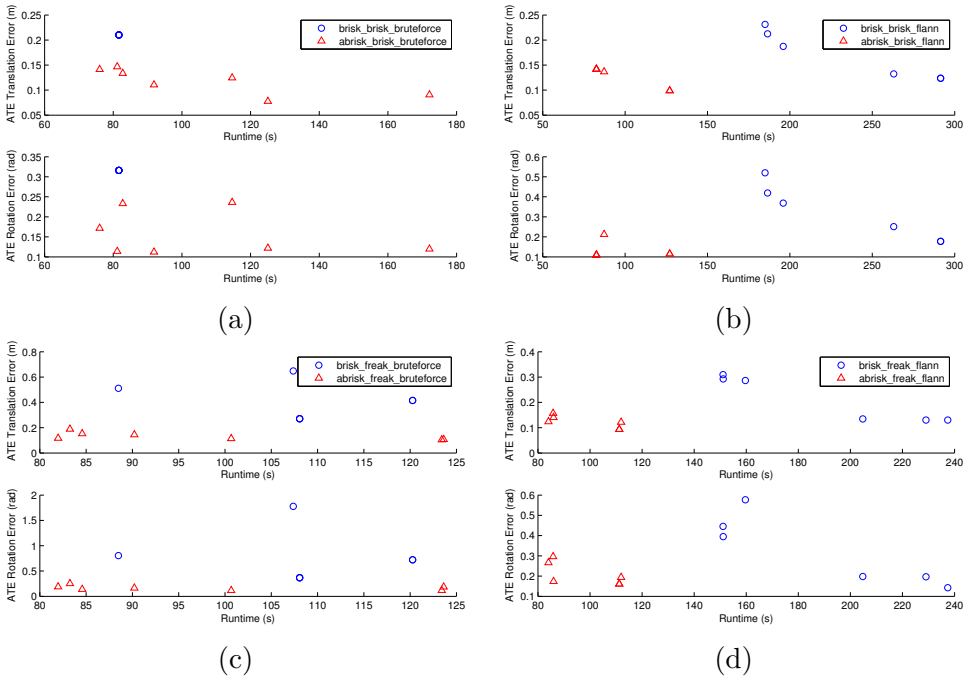


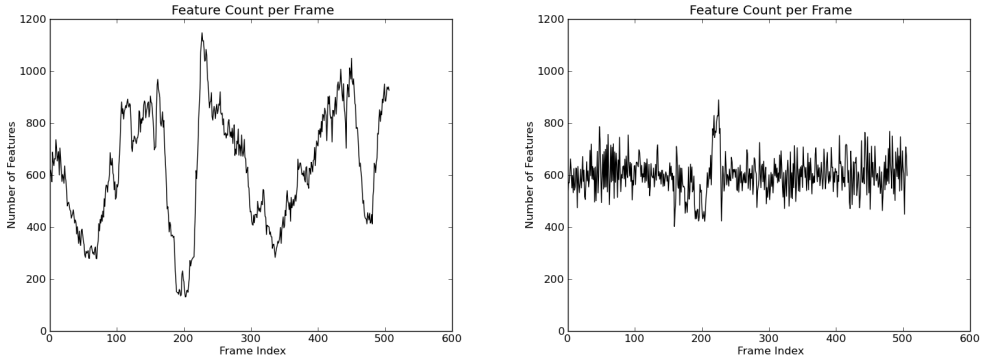
Figure 5.19: A comparison of mean ATE values and total run times for BRISK and Adjustable BRISK detectors.

ATE metrics from averages around 0.5 m and 1 rad to 0.12 m and 0.16 rad.

The more consistent number of features supplied by the adjustable detector helped improve stability, accuracy, and run time. Figure 5.20 shows the number of features detected in each frame for standard and Adjustable BRISK-AGAST detectors. Both detectors have similar average feature counts with 624.7 for the regular detector and 600.0 for the adjustable detector, but the adjustable detector is able to keep the feature count around 600, with a standard deviation of 70.6 features per frame compared to the regular detector's standard deviation of 226.1 features.

This more consistent feature count improves stability by limiting the possibility of localization failure due to not enough matches. The 200 frame index marks the peak of the vertical rotation which caused many failures during testing. At this index, the feature count for the normal detector dropped below 200 features, reaching a level where localization failure is very possible. The Adjustable BRISK detector was able to compensate for the change in image appearance at this time and reduce `thresh`, stopping the drop at 450 features, well away from the danger zone.

Trajectory estimate quality is improved by the more consistent number



(a) (b)

Figure 5.20: Feature counts per frame for a BRISK detector (Figure 5.20a) and an Adjustable BRISK detector (Figure 5.20b). Both detectors had `nOctaves` set to 1. The BRISK detector had a `thresh` value of 6 giving an average of 624.7 features per frame and the Adjustable detector had feature bounds of 600-601.

of features and matches. Preventing the feature count from dropping too low ensures that a rich set of matching features is available for the RANSAC based pairwise alignment. Limiting the total number of features from growing too large reduces the number of outliers that are introduced to RANSAC's sampling set in the form of false matches. This, in turn, limits the number of sampling iterations that RANSAC must perform and lessens the chance that RANSAC will return an incorrect odometry estimate.

The more consistent number of features actually increased total run time from 96.69 s to 114.15 s, with an increase in total matching time from 95.25 s to 321.7 s, a decrease in the total transformation recovery time from 69.11 s to 56.9 s, and a decrease in total pose graph optimization time from 65.09 s to 54.79 s. The increase in matching time is due to the fact that the matching time of a brute-force matcher, which tries all combinations between feature sets, grows quadratically with the number of features. Despite the increase in run time at similar feature averages, the Adjustable BRISK detector is still an overall improvement to run time, as smaller average feature counts can be safely used while producing trajectory estimates of comparable quality. An RGB-D SLAM System run with an average of 600 features is one of the fastest settings possible for the regular detector, and one of the slowest for the Adjustable BRISK detector.

The adjustable detector does a fairly good job at keeping the number of

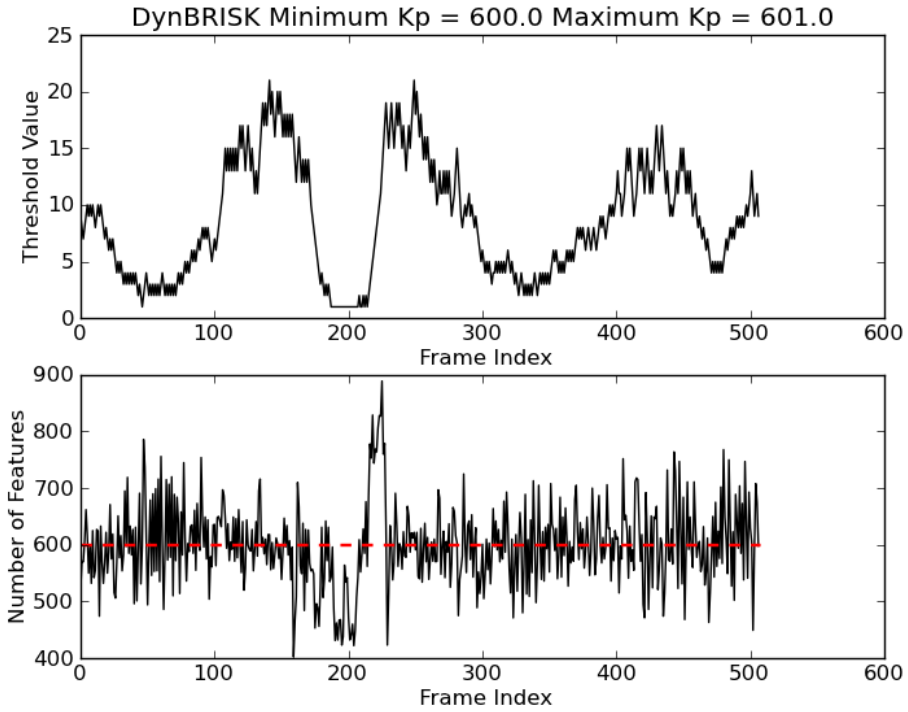


Figure 5.21: The value of `thresh` and the number of feature detected using an Adjustable BRISK-AGAST detector with feature bound of 600-601, indicated by the dashed red line.

features per frame within the desired bounds. Consider Figure 5.21, showing the numbers of features detected and the internal threshold value of a detector supplied with bounds of 600-601 features per frame. The detector was able to keep the feature count between 550 and 650 features for the majority of the run. From frame index 190-220 it experienced some difficulty. As the number of features dropped, the detector reacted by decreasing its threshold until the minimum value was reached. At this point, the detector was no longer capable of reacting, and the feature count dropped below 500. Additional difficulties were encountered when the feature count began to rise again. The detector did not increase the threshold value quickly enough to match the increasing feature count, and the count spiked to nearly 900 features, while the detector caught up.

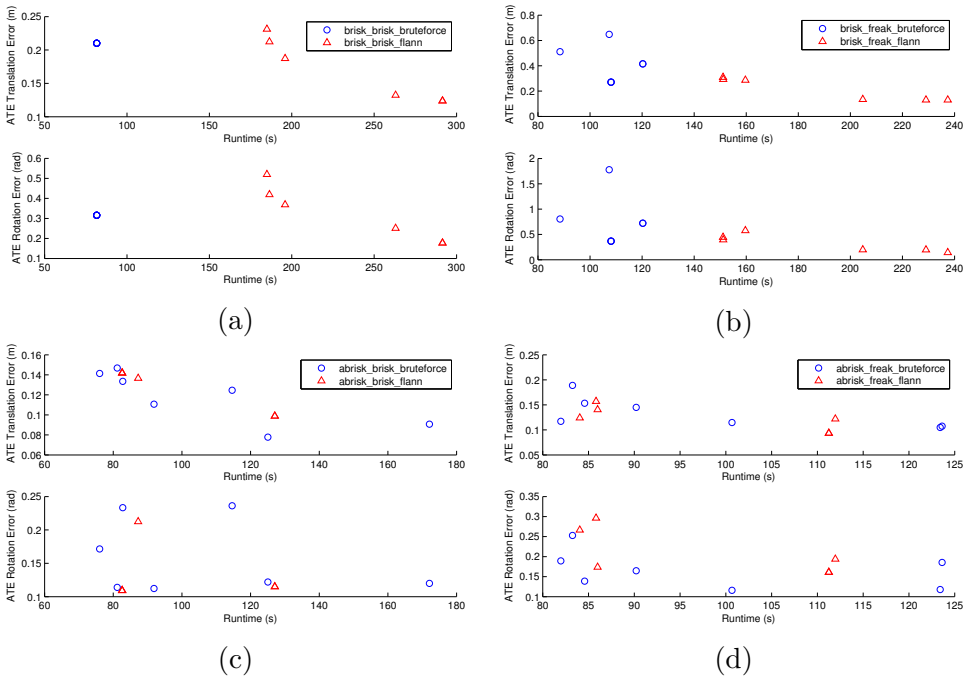


Figure 5.22: A comparison of mean ATE values and total run times for brute-force-Hamming and FLANN-LSH matchers.

5.13 Comparison of Brute-Force and FLANN Matchers

Figure 5.22 gives a breakdown of mean absolute trajectory errors for each of the four detector, extractor combinations. Brute-force-Hamming matcher results are indicated with a blue circle and FLANN-LSH results are shown with a red triangle. The brute-force matcher was a clear favorite for use with the normal BRISK-AGAST detector. Brute-force and FLANN matchers produced comparable errors, but FLANN took at least 40 s longer to process the video sequence. While the brute-force runs required 4 to 6 s to process each second of the video, the FLANN runs took between 7.5 s and 15 s.

FLANN was much more competitive with brute-force when using the Adjustable BRISK-AGAST detector. Absolute trajectory errors were comparable for the two matching methods, with average brute-force ATE values of 0.1453 ± 0.028 m and 0.2049 ± 0.0605 rad compared to FLANN's 0.1411 ± 0.0288 m and 0.02034 ± 0.0498 rad with a FREAK descriptor. Errors of 0.141 ± 0.0379 m and 0.1668 ± 0.0485 rad for brute-force and 0.1317 ± 0.0287 m and 0.1049 ± 0.0679 m and 0.1178 ± 0.0381 rad for FLANN were found with a BRISK descriptor. Average run time for FLANN matchers was 9 s

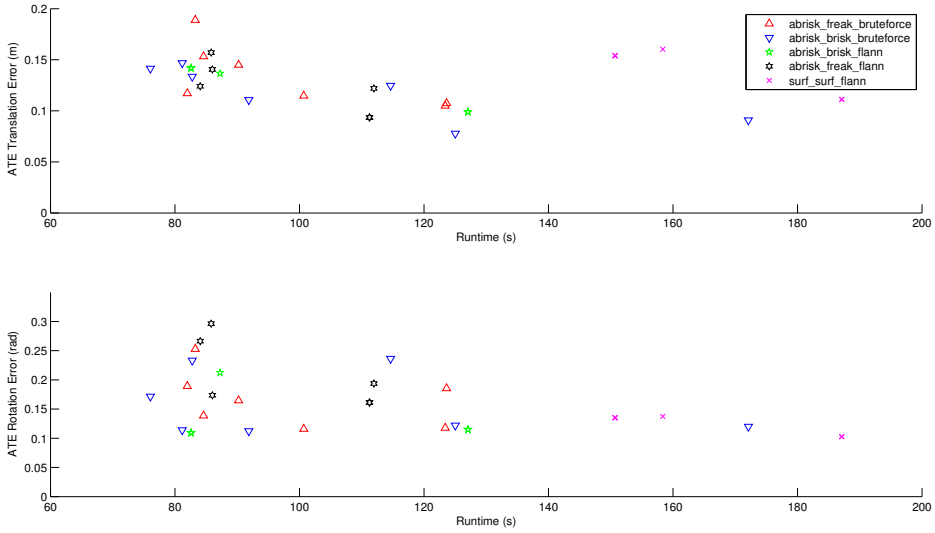
faster than brute-force for BRISK descriptor extractors and 11 s faster for FREAK extractors. Trials with feature counts in the 200-400 range ran 1 to 2 s faster with a brute-force matcher, while those with higher feature counts of 500-700 ran at least 15 s faster with a FLANN matcher.

5.14 Comparison of BRISK, FREAK, and SURF Descriptors

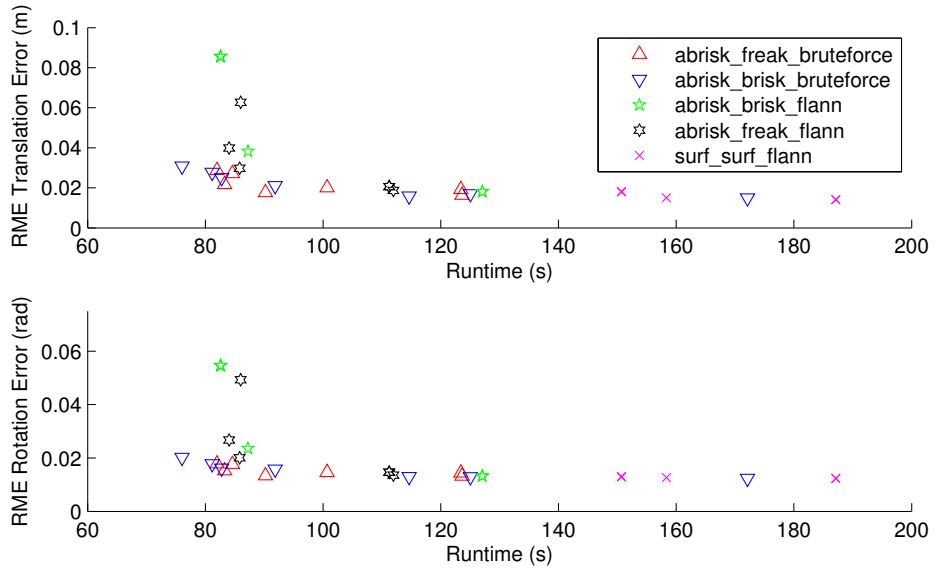
Figure 5.23 plots mean ATE and RME values for the best runs using and Adjustable BRISK detector. Several top runs using the default RGB-D SLAM System's feature detection system, SURF, are also plotted. All configurations using an Adjustable BRISK detector produced runs with errors comparable to SURF, and the BRISK extractor with brute-force matcher produced two runs that were better than SURF.

The SURF run with the lowest ATE had a run time of 187.104 s and an ATE of 0.111 ± 0.055 m and 0.103 ± 0.054 rad. The run with the overall lowest mean ATE used an Adjustable BRISK detector, a BRISK extractor, and a brute-force matcher with an `nOctaves` value of one, a `patternScale` of 1.25, and feature bounds of 600-601. These settings produced a total run time of 125.036 s an ATE of 0.078 ± 0.069 m and 0.122 ± 0.041 rad. Another setting of the same configuration produced errors equal to the best SURF run at an even faster run time of 91.86 s, with ATE means of 0.111 ± 0.091 m and 0.112 ± 0.045 rad, using an `nOctaves` value of zero, a `patternScale` setting of 1.1, and feature bounds of 370-371.

Figure 5.24 plots the absolute trajectory errors for these three runs. The SURF system is more robust to changes in perspective and was capable of tracking the camera through the vertical rotation at the 10 s mark with no notable increase in rotation error, and only a 0.2 m increase an translation error. The 125 s BRISK run also performed well during this rotation with an increase trajectory error of 0.1 rad and 0.2 m. The 91.86 s BRISK run experienced difficulties typical of the BRISK system in these circumstances with a 0.7 m increase in translation error a 0.3 rad increase in rotation error. During normal translational and small rotational movement, both BRISK settings outperformed SURF with translation and rotation errors slightly smaller than SURF's for the majority of the run.



(a)



(b)

Figure 5.23: A comparison of mean ATE and RME values and total run times for the top runs using BRISK, FREAK, and SURF descriptors. Red upward-point triangles indicate runs using FREAK descriptors and brute-force matchers, blue downward-pointing triangles are BRISK and brute-force runs, green pentagrams are BRISK and FLANN runs, black hexagrams are FREAK and FLANN runs, and the cyan x's are SURF runs.

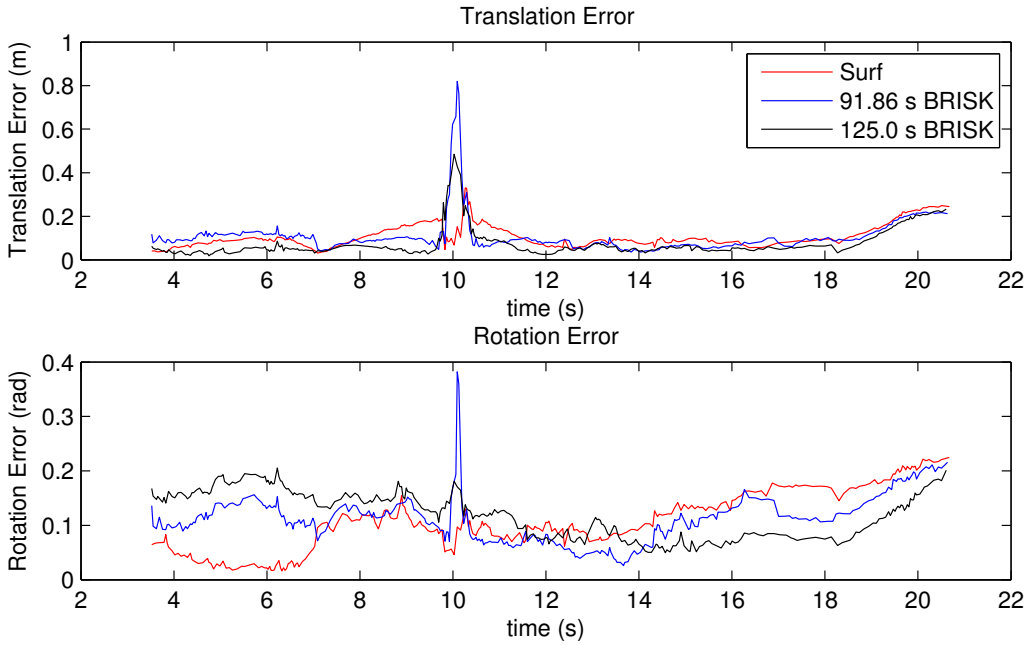


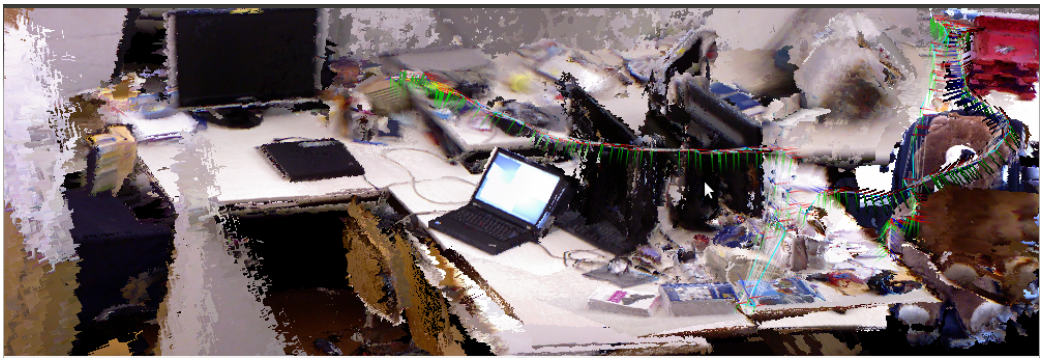
Figure 5.24: A comparison of instantaneous ATE for top SURF and BRISK runs.

Figure 5.25 shows the map generated for the 91.86 s BRISK run. Despite some errors near the middle of the run, the overall map quality was quite good. Notice the sharp vertical drop in the pose near the middle of the two desks. This corresponds to the spike in translation error at the 10 s mark. The top right portion of the map is slightly distorted. This is a result of the increasing pose and translation errors near the 18 s mark.

5.15 Summary

The Adjustable BRISK-AGAST detector is a viable alternative to SURF in the RGB-D SLAM System. It provides well localized, useful interest points in only a fraction of the detection time. FREAK descriptors and FLANN matchers are effective tools and produce useful results, but a BRISK descriptor extractor and brute-force matcher works best with lower feature bounds needed for the fastest run times. Care should be taken during the selection of parameters for the detector and extractor. Since the combination of AGAST interest points and BRISK descriptors is not as robust to changes in perspective as SURF, failure of RGB-D SLAM during large rotations is possible if

(a)



(b)

Figure 5.25: Maps generated in RGB-D SLAM. Figure 5.25a is generated using the default SURF settings. Figure 5.25b is generated using an Adjustable BRISK-AGAST detector with BRISK descriptors and a brute-force matcher. Camera pose estimates are overlaid on both images.

these values are not chosen with care.

Chapter 6

Point Cloud Transformation Work

There are many applications of SLAM systems in robotics. Most applications typically center around the localization of a robot or other objects in an environment or the description of that environment in the form of a map. The RGB-D SLAM System and similar systems produce maps with a high degree of visual fidelity that are ideal for map-based augmented reality applications. Such applications can include insertion of objects into the map, alteration of the appearance of objects in the map, and modification of the structure and appearance of the map itself.

This chapter details such an augmented reality application for an RGB-D SLAM System, where the structure of a map, represented using a 3-D point cloud of RGB pixels, is altered. This method allows the user to create distorted maps by applying transformations to the original map, and to generate photo-realistic views of these distorted maps. A method of sub-sampling the map so as to reduce projection time and to limit visual irregularities associated with the projection of occluded and partially occluded objects is also presented.

The structure of the remaining portions of the chapter is as follows. Section 6.1 will review the generation of RGB point cloud based maps using the RGB-D SLAM System. Section 6.2 will explain the technical details of the map transformation and projection application. Section 6.3 will present typical outputs of this application and discuss the method's effectiveness.

6.1 RGB-D SLAM System Map

The RGB-D SLAM map is generated using keyframes. Each keyframe $f = \langle P, \Gamma, \bar{\mathbf{T}} \rangle$ is a representative sample of the input video sequence consisting of a point cloud P, a landmark point cloud Γ, and a discrete pose $\bar{\mathbf{T}}$. Both point

clouds are defined relative to the camera frame C at the time of sampling. The point cloud P and landmark point cloud Γ are formed as described in Section 3.4.

The pose $\bar{\mathbf{T}}$ is given as the transformation from C to the world frame W. In circumstances where it is more convenient to refer to this transformation in terms of a quaternion rotation and vector translation, the notation $\langle \mathrm{Q}, \boldsymbol{\rho} \rangle$ is used.

$$\bar{\mathbf{T}} \Leftrightarrow \langle \mathrm{Q}, \boldsymbol{\rho} \rangle$$

Keyframes are chosen to represent distinct portions of the scene. They are selected by comparing the pose at the current frame to that of already existing keyframes. When there is a significant translational or rotational difference between the current pose and all existing keyframe poses, the input frame is chosen as a new keyframe.

This chapter will assume the number of keyframes is fixed at n, yielding a set of keyframes $F = \{f_1, f_2, \ldots, f_n\}$. In the RGB-D SLAM System, the number of keyframes n is not constant. As the environment is explored, more keyframes will be added and n will increase. This change in n is of no concern to the development of the transformation and projection method, as n changes at a much slower rate than the method runs.

The RGB-D SLAM System's map M that is used for visualization purposes is formed by transforming each keyframe point cloud P_i into the world frame W using $\bar{\mathbf{T}}_i$ and then combining all transformed point clouds into a single point cloud. An example map is shown in Figure 3.2 and three of the point clouds used to generate the map are shown in Figure 3.3. The map generation function is given in Equation (6.1).

$$M = \bigcup_{i=1}^{n} {}^{W}P_i \tag{6.1}$$

6.2 Method

This section will detail the generation of an $h \times d$ RGB image I_s for an RGB-D SLAM map that has been distorted using an affine transformation matrix $\mathbf{A}$. The image I_s has a height of h pixels and a width of d pixels. The image is

created by using perspective projection to project a portion of the distorted map onto the image plane located at ${}^{C}z = 1$, where C is the current camera coordinate frame. This process makes use of the set of RGB-D SLAM System keyframes F as well as the current frame $f_s = \langle P_s, \Gamma_s, \bar{\mathbf{T}}_s \rangle$.

Figure 6.1 contains a flow chart depicting this process. The point cloud P_i for each keyframe f_i is transformed according to the affine transformation matrix $\mathbf{A}$, producing a transformed point cloud $\hat{P}_i$. Since the transformation $\mathbf{A}$ will change infrequently in expected use, it is computationally efficient to transform all the keyframes once and cache the transformed keyframes for future use. The keyframes are sampled by identifying keyframes whose pose is close to the current pose, producing the keyframe set F'. This set is then sampled again by evaluating the keyframes in terms of the Hausdorff distance [64] between their landmark point clouds and Γ_s, producing the set $\bar{F}$. The transformed point clouds of up to k nearby keyframes are then used to form the sampled distorted map $\bar{M}$. The sampled distorted map $\bar{M}$ is projected and scaled according to the camera's intrinsic matrix $\mathbf{K}$. Finally, the projected cloud is translated so that its centroid falls at the center of the image $(h/2, d/2)$, and I_s is formed.

6.2.1 Point Cloud Transformations

The RGB-D SLAM System map M in Equation (6.1) is created by merging the point clouds ${}^{W}P$ for all the keyframes F. The distorted maps in this transformation and projection method are created in a similar manner. Individual distorted point clouds $\hat{P}_i$ are created for each keyframe. Distorted point clouds for select keyframes are then merged with the distorted input point cloud to form a sampled distorted map $\bar{M}$.

A keyframe point cloud P_i is transformed into a distorted point cloud $\hat{P}_i$ using the affine transformation matrix $\mathbf{A}$ supplied by the user. This transformation matrix, specified in the world frame using homogeneous coordinates, can perform operations such as rotations, translations, scales, flips, and shears to distort the point clouds. The matrix can be written as a product of three matrices $\mathbf{A}_s$, $\mathbf{A}_r$, and $\mathbf{A}_t$. In this formulation, the matrix $\mathbf{A}_s$ performs scaling, skewing, flipping, and shearing operations. The matrix $\mathbf{A}_r$ performs 3-D rotations about the origin. The matrix $\mathbf{A}_t$ performs transla-

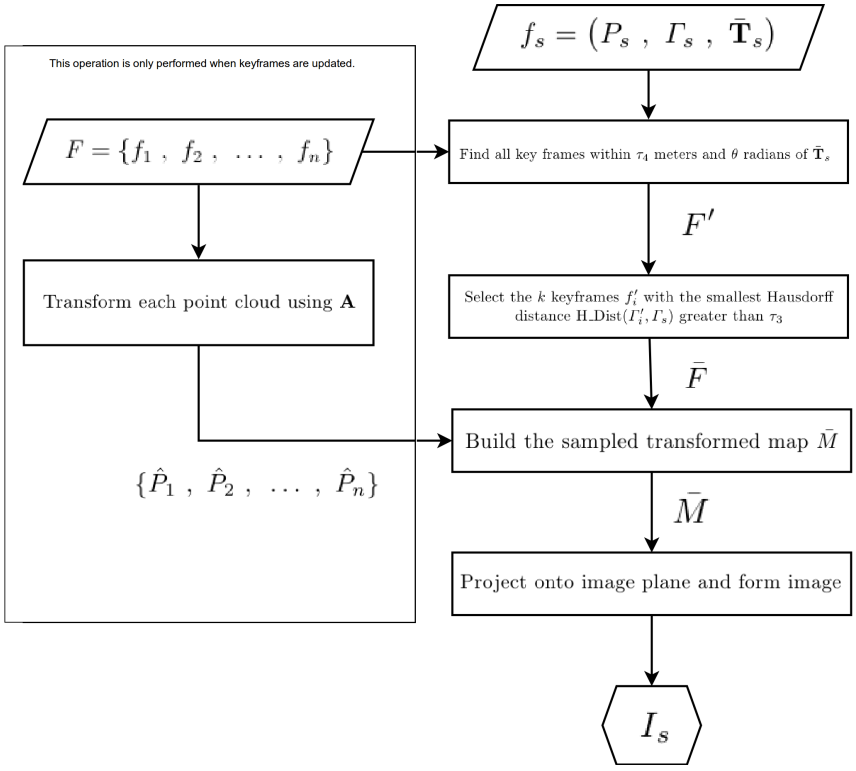


Figure 6.1: Flowchart for the point cloud transformation application of the RGB-D SLAM System.

tions.

$$\mathbf{A} = \mathbf{A}_s \mathbf{A}_r \mathbf{A}_t$$

$$\mathbf{A}_s = \begin{bmatrix} s_{xx} & s_{xy} & s_{xz} & 0 \\ s_{yx} & s_{yy} & s_{yz} & 0 \\ s_{zx} & s_{zy} & s_{zz} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The scaling matrix $\mathbf{A}_s$ has on-diagonal and off-diagonal terms. The on-diagonal terms s_{xx}, s_{yy}, and s_{zz} are responsible for scaling along the axes. The off-diagonal terms s_{xy}, s_{xz}, s_{yx}, s_{yz}, s_{zx}, and s_{zy} are responsible for skewing and shearing effects.

Before it is distorted, the point cloud P_i for each keyframe f_i is transformed into the world frame W by applying the transformation representing the pose $\bar{\mathbf{T}}_i$. This transformation, given in quaternion vector form as $\langle \mathrm{Q}_i, \boldsymbol{\rho}_i \rangle$, is applied to each point $\langle \mathbf{p}, L \rangle$ in the point cloud P_i. Each point $\langle \mathbf{p}, L \rangle$ is composed of a 3-D location $\mathbf{p}$ and RGB intensities L. The transformation is described in Equation (6.2).

$${}^{W}P_i = \left\{ \left\langle {}^{W}\mathbf{p}_1, L_1 \right\rangle, \left\langle {}^{W}\mathbf{p}_2, L_2 \right\rangle, \dots \right\}$$

$$\begin{bmatrix} 0 \\ {}^{W}\mathbf{p} \end{bmatrix} = \mathrm{Q}_i \begin{bmatrix} 0 \\ \mathbf{p} \end{bmatrix} \mathrm{Q}_i^{-1} + \begin{bmatrix} 0 \\ \boldsymbol{\rho}_i \end{bmatrix} \tag{6.2}$$

The distorted point cloud $\hat{P}_i$ is then created by applying the affine transformation matrix $\mathbf{A}$ to each point $\left\langle {}^{W}\mathbf{p}, L \right\rangle$ in ${}^{W}P_i$. This operation is described in Equation (6.3). The RGB intensities L for each point are left unaltered in both the coordinate frame transformation using $\bar{\mathbf{T}}_i$ and the affine transformation using $\mathbf{A}$.

$$\hat{P}_i = \left\{ \langle \hat{\mathbf{p}}_1, L_1 \rangle, \langle \hat{\mathbf{p}}_2, L_2 \rangle, \dots \right\}$$

$$\begin{bmatrix} \hat{\mathbf{p}} \\ 1 \end{bmatrix} = \mathbf{A} \begin{bmatrix} {}^{W}\mathbf{p} \\ 1 \end{bmatrix} \tag{6.3}$$

6.2.2 Keyframe Sampling

Keyframe sampling is used to build the distorted sampled map $\bar{M}$ from the keyframes and their distorted point clouds. By building this map using keyframes sampled at poses similar to the current pose, a realistic looking image can be constructed without the need to reason about occlusion in the distorted map. Up to k keyframes are selected using the Hausdorff distance [64] between each key frame's landmark point cloud Γ_i and the current landmark point cloud Γ_s.

The Hausdorff distance measure used to evaluate the similarity of two landmark point clouds is very computationally intensive, requiring $2ab - (a + b)$ Euclidean distance measurements to evaluate the distance between two point clouds of size a and b. With average point cloud sizes of 500 points, and on the order of 60 to 100 keyframes per map, it is important to limit the number of Hausdorff distance measurements so as to maintain reasonable run times.

For this reason, the set of keyframes F is first filtered to produce a smaller set of keyframes F'. This is accomplished by measuring the transformation between the current pose $\bar{\mathbf{T}}_s$ and the keyframe poses and selecting only those keyframes that have a translation transformation magnitude smaller than a threshold of τ_4 and a rotational transformation magnitude smaller than a threshold of θ. The τ_4 and θ constraints ensure that only keyframes with pose orientations and translations similar to that of the input frame are chosen. The translation and orientation constraints are considered separately because small changes in orientation result in much larger changes to the content of a frame's point cloud than do small changes in translation. Keyframes whose poses differ significantly from the current pose will have little to no visual overlap with the current camera image, and are therefore poor choices for use in construction of the sampled map. Such keyframes are removed, leaving the smaller set F'. Equation (6.4) details the filtering process.

$$F' = \left\{ f_i \mid f_i \in F, 2\arccos(|\Re\{\mathrm{Q}_i^{-1}\mathrm{Q}_s\}|) < \theta, ||\boldsymbol{\rho}_i - \boldsymbol{\rho}_s|| < \tau_4 \right\} \tag{6.4}$$

Hausdorff distance is then used to select k keyframes to form an even smaller set $\bar{F}$. The distance between keyframe landmark point clouds and

the current frame's landmark point cloud is considered. Similar to pairwise alignment in the RGB-D SLAM System, landmark point clouds Γ are used rather than the full point clouds P. The landmark point clouds are representative samples of the full point clouds, containing only around 500 points compared to the up to 307,200 points in a full point cloud. This leads to significant computational savings, making it possible to compute the Hausdorff distance in reasonable amounts of time.

Each landmark point cloud Γ'_j in F' is transformed to the world coordinate frame and the Hausdorff distance h_j with the current landmark point cloud Γ_s is found. The Hausdorff distance measures the largest minimal distance between points in one point cloud and the nearest points in another point cloud and vice versa. Equation (6.6) expresses this notion more formally, measuring the Hausdorff distance between two point clouds A and B. In this equation, the function $\Delta(\mathbf{x}, \mathbf{y})$ denotes a distance function between two points $\mathbf{x}$ and $\mathbf{y}$. This function can be any distance function such as Mahalanobis, Manhattan, or Euclidean distance. In the implementation developed for this thesis, squared Euclidean distance was used.

$$\begin{aligned} h &= \text{H\_Dist}(A, B) \\ &= \max\left\{\max_{a\in A}\left(\min_{b\in B}\left(\Delta(a,b)\right)\right), \max_{b\in B}\left(\min_{a\in A}\left(\Delta(b,a)\right)\right)\right\} \\ &= \max\left\{\max_{a\in A}\left(\min_{b\in B}\left(||a-b||^2\right)\right), \max_{b\in B}\left(\min_{a\in A}\left(||a-b||^2\right)\right)\right\} \end{aligned} \tag{6.5}$$

The k keyframes used to make up $\bar{M}$ are then selected by identifying the k keyframes in F' whose Hausdorff distance measure is closest to, but still larger than τ_3. A justification for this constrained minimization approach will be given in the next paragraph. Equation (6.6) details the selection of $\bar{F}$, the set of these k keyframes. The constraints $\Phi(\Gamma')$ are detailed in Equation (6.7).

$$
\begin{aligned}
\bar{F} &= \{\bar{f}_1, \bar{f}_2, \ldots, \bar{f}_k\} \\
\bar{f}_1 &= \left\{ f'_j \quad | \quad \Gamma'_j = \underset{\Gamma' s.t. \Phi_1(\Gamma')}{\operatorname{argmin}} \ \text{H\_Dist}(\Gamma', \Gamma_s) \quad , \quad f'_j \in F' \right\} \\
\bar{f}_2 &= \left\{ f'_j \quad | \quad \Gamma'_j = \underset{\Gamma' s.t. \Phi_2(\Gamma')}{\operatorname{argmin}} \ \text{H\_Dist}(\Gamma', \Gamma_s) \quad , \quad f'_j \in F' \right\} \\
&\vdots \\
\bar{f}_k &= \left\{ f'_j \quad | \quad \Gamma'_j = \underset{\Gamma' s.t. \Phi_k(\Gamma')}{\operatorname{argmin}} \ \text{H\_Dist}(\Gamma', \Gamma_s) \quad , \quad f'_j \in F' \right\}
\end{aligned}
\tag{6.6}
$$

$$
\begin{aligned}
\Phi_1(\Gamma') &= \left\{ \Gamma'_j \quad | \quad \text{H\_Dist}(\Gamma'_j, \Gamma_s) > \tau_3 \quad , \quad f'_j \in F' \right\} \\
\Phi_2(\Gamma') &= \left\{ \Gamma'_j \quad | \quad \text{H\_Dist}(\Gamma'_j, \Gamma_s) > \tau_3 \quad , \quad f'_j \in F' \backslash \{\bar{f}_1\} \right\} \\
&\vdots \\
\Phi_k(\Gamma') &= \left\{ \Gamma'_j \quad | \quad \text{H\_Dist}(\Gamma'_j, \Gamma_s) > \tau_3 \quad , \quad f'_j \in F' \backslash \{\bar{f}_1 \cdots \cup \bar{f}_{k-1}\} \right\}
\end{aligned}
\tag{6.7}
$$

The τ_3 constraint is an important step to enforce diversity in the selected keyframes. Without the term, keyframes which almost exactly resemble the current frame would be selected. There would be a very large amount of overlap between each of the keyframes' point clouds, resulting in a map $\bar{M}$ which closely resembles the input point cloud P. Enforcing a distance of at least τ_3 introduces a measure of dissimilarity between the point clouds and ensures that additional data not available at the current camera pose will be incorporated into the sampled map. This more expansive map increases the range of transformations $\mathbf{A}$ that can be applied while still presenting a realistic view. Without this dissimilarity in keyframes, operations which make the current view "zoom out," such as those increasing the apparent depth of a room, would not be practical. Increasing depth would make the displayed image shrink toward its center. With most of the sampled map already displayed, the distorted image would be a smaller version of the original image surrounded by a field of empty pixels. Expanding the map beyond the original image allows for incorporation of portions of the map not displayed in the "zoomed in" version, creating a more realistic looking image with a scaled depth.

Finally, the sampled distorted map $\bar{M}$ is constructed. The current frame's point cloud P_s is transformed into world coordinates and distorted according to Equation (6.2) and Equation (6.3), producing $\hat{P}_s$. The transformed point cloud $\hat{P}_s$ is merged with the transformed point cloud $\hat{P}$ for each keyframe in $\bar{F}$. The merged point cloud is then downsampled using a voxel grid filter [65]. A 3-D grid of voxels is fit over the point cloud and all points falling within a given voxel are replaced by a single point located at the centroid of the replaced points with RGB intensities equal to the mean intensities of the replaced points. This downsampling step reduces map redundancy introduced by overlapping component point clouds and serves as a low pass spatial filter, reducing the appearance of small alignment issues. Equation (6.8) depicts the relevant operations needed to produce the sampled transformed map $\bar{M}$.

$$\bar{M} = \text{Voxel\_Downsample}\left\{\left(\bigcup_{\bar{f}_i \in \bar{F}} \hat{P}_i\right) \cup \hat{P}_s\right\} \tag{6.8}$$

6.2.3 Projection

An image of the map $\bar{M}$ is then created at the current camera pose $\bar{\mathbf{T}}_s$. The map is transformed into the current camera frame C using the inverse of the transformation used to represent the current pose $(\bar{\mathbf{T}}_s)^{-1}$. Equation (6.9) relates the transformation of the location of a point $\langle \mathbf{p}, L \rangle \in \bar{M}$ into the camera frame using quaternion multiplication.

$$\begin{bmatrix} 0 \\ {}^{C}\mathbf{p} \end{bmatrix} = \mathrm{Q}_s^{-1} \left(\begin{bmatrix} 0 \\ \mathbf{p} \end{bmatrix} - \begin{bmatrix} 0 \\ \boldsymbol{\rho}_s \end{bmatrix} \right) \mathrm{Q}_s \tag{6.9}$$

The location of each point is then projected onto the image plane located at ${}^{C}z = 1$, using the pixel lengths p_u and p_v retrieved from the camera's intrinsic matrix $\mathbf{K}$. The translation terms c_u and c_v in $\mathbf{K}$ are not used. Since the point cloud generated from the image has been transformed, the optical center of the projected distorted map will likely no longer be at (c_u, c_v). A new optical center must be found. The terms c_u and c_v are replaced by μ_u and μ_v, the centroids of the projected point cloud. Calculation of μ_u and μ_v is detailed in Equation (6.11). In this equation, $|\bar{M}|$ is the number of points in the map.

$$\mathbf{K} = \begin{bmatrix} p_u & 0 & 0 & c_u \\ 0 & p_v & 0 & c_v \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

$$\mu_u = \frac{1}{|\bar{M}|} \sum_{m \in \bar{M}} \frac{p_u x_p}{z_p}$$

$$\mu_v = \frac{1}{|\bar{M}|} \sum_{m \in \bar{M}} \frac{p_v y_p}{z_p} \tag{6.10}$$

$$\tag{6.11}$$

The location of the projected points is given by the perspective projection equation in Equation (6.12). Here the projection of a point $\left\langle {}^{C}\mathbf{p}, L \right\rangle$ with 3-D location given by $\mathbf{p} = \begin{bmatrix} x_p & y_p & z_p \end{bmatrix}^T$ is shown. The terms h and d denote the dimensions of the $h \times d$ image I_s that is constructed. The values u_p and v_p give the location of the projected point on the imaging plane.

$$\begin{bmatrix} u_p \\ v_p \\ 1 \end{bmatrix} = \begin{bmatrix} \frac{p_u x_p}{z_p} - d/2 + \mu_u \\ \frac{p_v y_p}{z_p} - h/2 + \mu_v \\ 1 \end{bmatrix}, \quad \forall \mathbf{p} \in \bar{M} \tag{6.12}$$

Once all points contained in $\bar{M}$ have been projected onto the imaging plane, the distorted image I_s can be created. The location of a projected point is converted into pixel indices by rounding u_p and v_p to the nearest whole number. The RGB intensity of the pixel in I_s at this location is set to the intensity L of the projected point. If no point is assigned to a pixel, that pixel's intensity is estimated using a 3×3 median filter to select the median value of the nine pixel patch surrounding the unassigned pixel.

6.3 Results and Discussion

This section will present experimental results for the map transformation method detailed in Section 6.2. In general, the method worked well and was

Figure 6.2: A map generated by the RGB-D SLAM System.

able to consistently transform the map using a variety of scaling and shearing operations and supply consistent images of the distorted map at a rate of 3-4 Hz.

Care has to be taken to ensure that the map is suitable for the type of transformation desired. For instance, maps generated using mostly rotational movement of a camera fixed on a pivot will become more sparse farther away from the pivot. Objects farther way from the pivot will not be imaged in as great detail and will be represented using a comparatively sparse set of RGB points. Consequently, scaling operations which "zoom in" on these far away objects will produce sparse, poor images due to a lack of available data. In general, however, as long as the map was generated with a good variety of rotational and translational movement, the generated images are of good quality.

In order to test this distortion method, several transformation matrices were applied to the map depicted in Figure 6.2. This map was generated for the room depicted in Figure 6.3. The world frame W in this map was defined such that the z-axis points toward the left-most wall supporting the white board, the y-axis points toward the floor, and the x-axis points "into" the room, normal to the plane formed by the poster board and bookshelf. In the following figures, the world frame axes will be indicated using a set of coordinate axes labeled `/map`. In this set of axes, the red line indicates the x-axis, the green line indicates the y-axis, and the blue line indicates the z-axis.

For these experiments, $k = 3$ keyframes were chosen to form the sample map, using filtering thresholds of $\tau_4 = 2$ m and $\theta = 0.3$ rad, and a minimum Hausdorff distance of $\tau_3 = 10$ m^2.

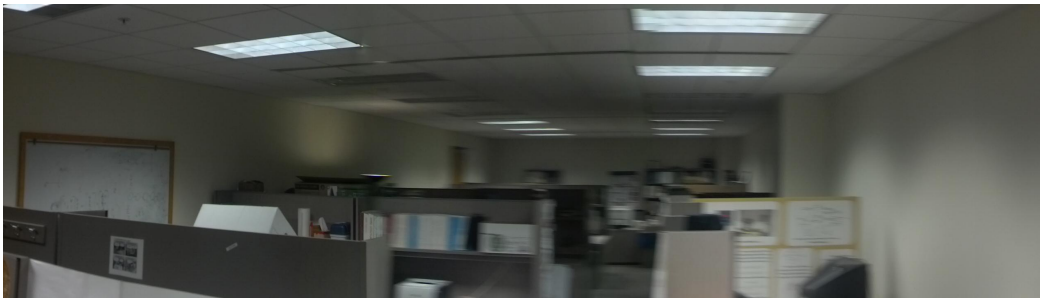
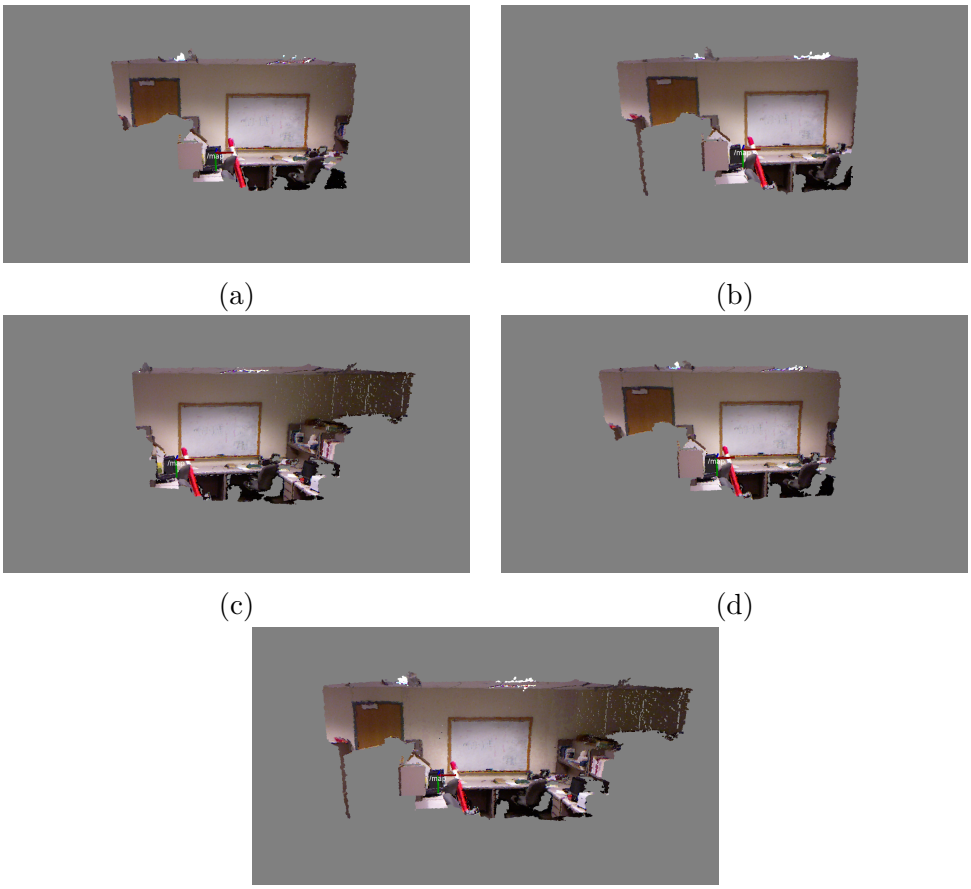


Figure 6.3: A photo of the mapped location.

Two representative poses were chosen for comparison. View 1, shown in Figure 6.4, faces the white board with its camera axes aligned with the world frame axes. View 2, shown in Figure 6.5, faces toward the poster board with ${}^{C}x$ pointing in the negative ${}^{W}z$ direction, ${}^{C}y$ pointing in the ${}^{W}y$ direction, and ${}^{C}z$ pointing in the ${}^{W}x$ direction.

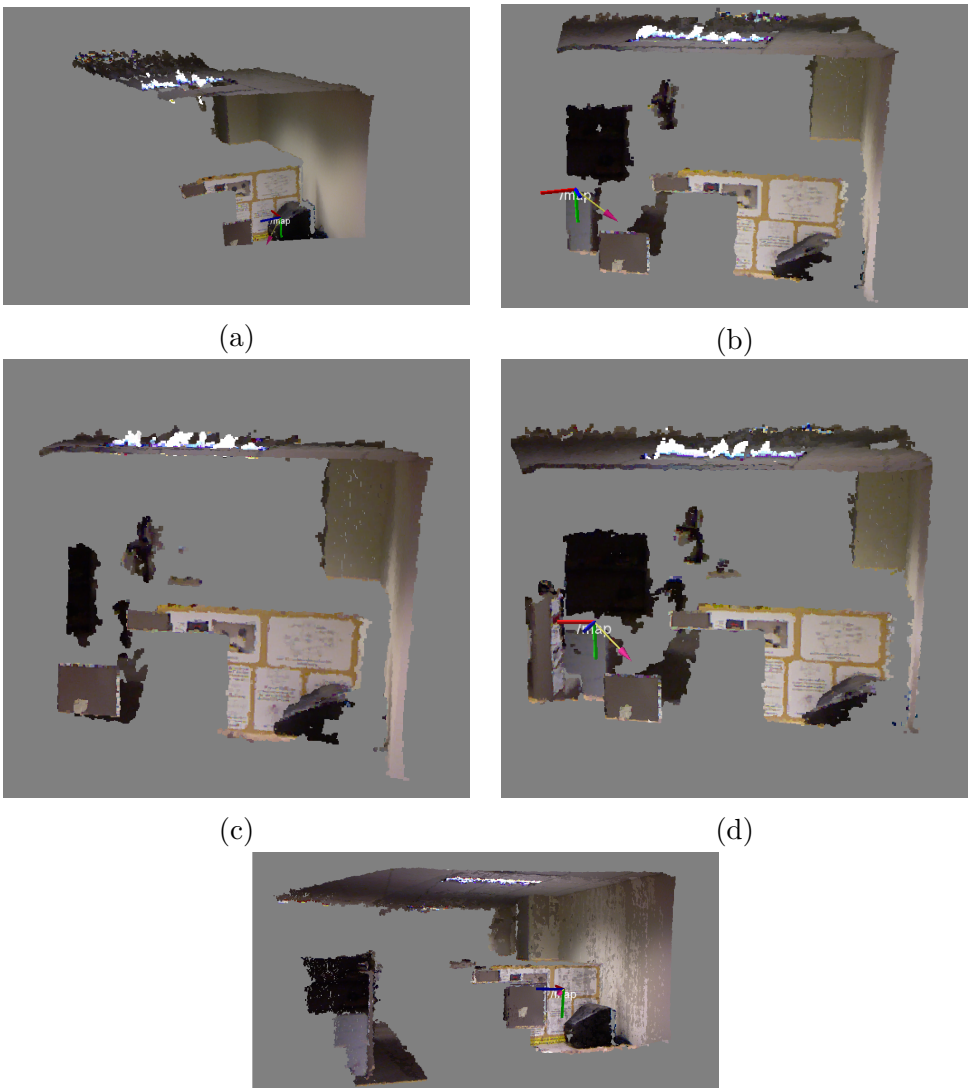
For both views, the sampling algorithm was able to select keyframes whose point clouds contain data not available to the camera at the current view, producing suitable sampled maps $\bar{M}_1$ and $\bar{M}_2$. Figure 6.6 shows the images generated for both views with no map distortions applied. Notice the black blob visible near the left portion of the View 2 image. This is the result of the finite range of the Kinect's depth sensor. While mapping, the RGB-D SLAM System never received data regarding the depth of the back wall represented by the black blob. Consequently, the wall was not incorporated into the map and cannot be displayed in the image. This same effect can occur for light fixtures that are within the maximum range of the depth sensor. These light fixtures saturate the IR camera, hindering the Kinect from detecting the IR pattern projected onto the fixtures, thus preventing recovery of a depth estimate and the incorporation of these objects into the map.

Three transformations were applied to the map and images for the two views. The first transformation was a scaling transformation. The maps' x coordinates were scaled by a factor of 0.5 and their z coordinates were scaled by 1.5. The transformation matrix $\mathbf{A}_1$ representing this operation is given in Equation (6.13). The transformation should result in an increase in apparent depth in View 1 and a contraction in the horizontal direction. Conversely, the apparent depth should be reduced in View 2 and there should be an expansion in the horizontal direction.



(a) (b) (c) (d) (e)

Figure 6.4: Components of the sampled map created for View 1. Figure 6.4a shows the input point cloud, Figures 6.4b, 6.4c, and 6.4d show the three selected keyframe point clouds, and Figure 6.4e shows the sampled map obtained by merging and downsampling these four point clouds.



(a) (b) (c) (d) (e)

Figure 6.5: Components of the sampled map created for View 2. Figure 6.5a shows the input point cloud, Figures 6.5b, 6.5c, and 6.5d show the three selected keyframe point clouds, and Figure 6.5e shows the sampled map obtained by merging and downsampling these four point clouds.

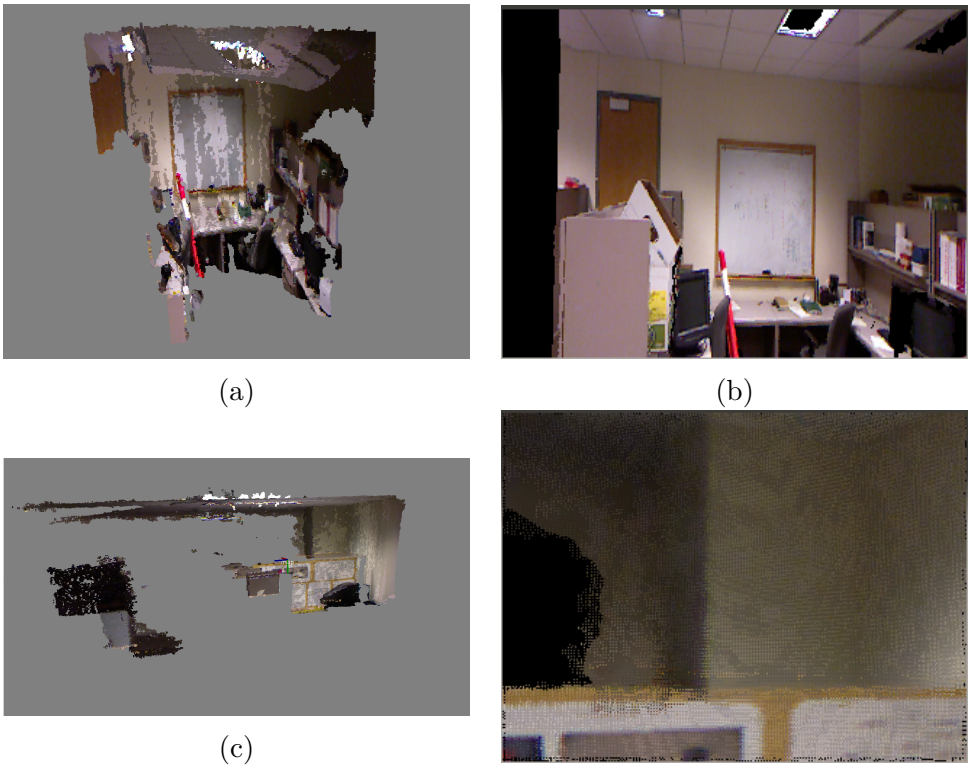
(a) (b)

Figure 6.6: Output images obtained without any map distortion. Figure 6.6a shows the output for View 1 and Figure 6.6b shows the output for View 2.

$$\begin{aligned} \mathbf{A}_1 &= \begin{bmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1.5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 0.5 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1.5 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{6.13}$$

The results of this transformation are shown in Figure 6.7. The sampled maps were distorted in the desired manner and the expected image distortions occurred. Note the decrease in quality of image 2. The "zooming in" caused by the ${}^{W}x$ scaling operation illustrates the sparseness of points representing objects farther away from the camera. Decreasing the ${}^{W}x$ scaling term any further would be inadvisable at this pose.

The second transformation was another scaling operation. In this transformation, the maps were scaled in the world x-axis by 1.5 and scaled by 0.75 in the world z-axis. This should produce the opposite effect of the first transformation. In View 1, the decrease in z will result in a decrease in apparent depth, and the increase in x will result in an expansion in the horizontal direction. In View 2, decreasing z will result in a horizontal contraction of the projected image, and the increase in x will result in an increase in appar-



(a) (b) (c) (d)

Figure 6.7: The distorted point clouds and output images for View 1 and View 2 where x was scaled by 0.5 and z was scaled by 1.5, relative to the world frame W. Figure 6.7a shows $\bar{M}$ at View 1 and Figure 6.7b shows I_s at View 1. Figure 6.7c shows $\bar{M}$ at View 2 and Figure 6.7d shows I_s at View 2.

ent depth. The transformation matrix for this transformation $\mathbf{A}_2$ is given in Equation (6.14). Figure 6.8 shows the results of this transformation.

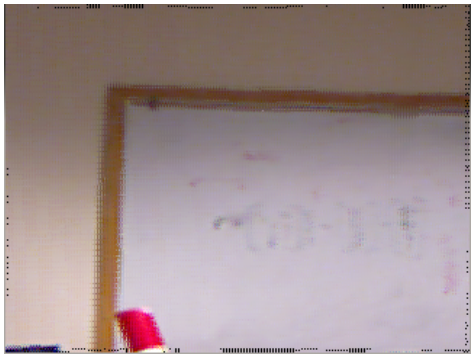
$$\begin{aligned}\mathbf{A}_2 &= \begin{bmatrix} 1.5 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.75 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1.5 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0.75 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{6.14}$$

The last transformation was a skewing operation using a factor of 0.4. The skewed transformation is given by $\mathbf{A}_3$ in Equation (6.15). This operation should make the room appear on an incline along the length of the room, with the height of the floor and ceiling increasing as x decreases. The height of the floor and ceiling should be constant with respect to z. This should result in a skewing of View 1 toward the top left corner of the image. View 2 should remain unchanged in the horizontal direction, but the height of the ceiling should decrease as apparent depth increases.

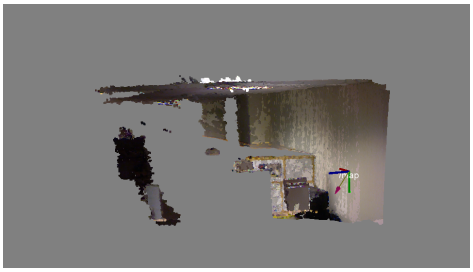
$$\begin{aligned}\mathbf{A}_3 &= \begin{bmatrix} 1 & 0.4 & 0 & 0 \\ 0.4 & 1 & 0 & 0 \\ 0 & 0 & 1.05 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0.4 & 0 & 0 \\ 0.4 & 1 & 0 & 0 \\ 0 & 0 & 1.05 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{6.15}$$

Figure 6.9 shows the results for both views using this transformation. View 1 is skewed toward the top left of the image and the depth of objects appears unchanged. In View 2, there is some skewing toward the bottom left of the image. This is due to an error in the positioning of the camera in View 2. The z-axis of the camera in View 2 is rotated slightly past the x-axis of the

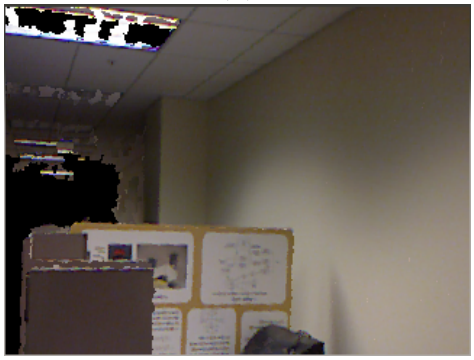
(a)



(b)

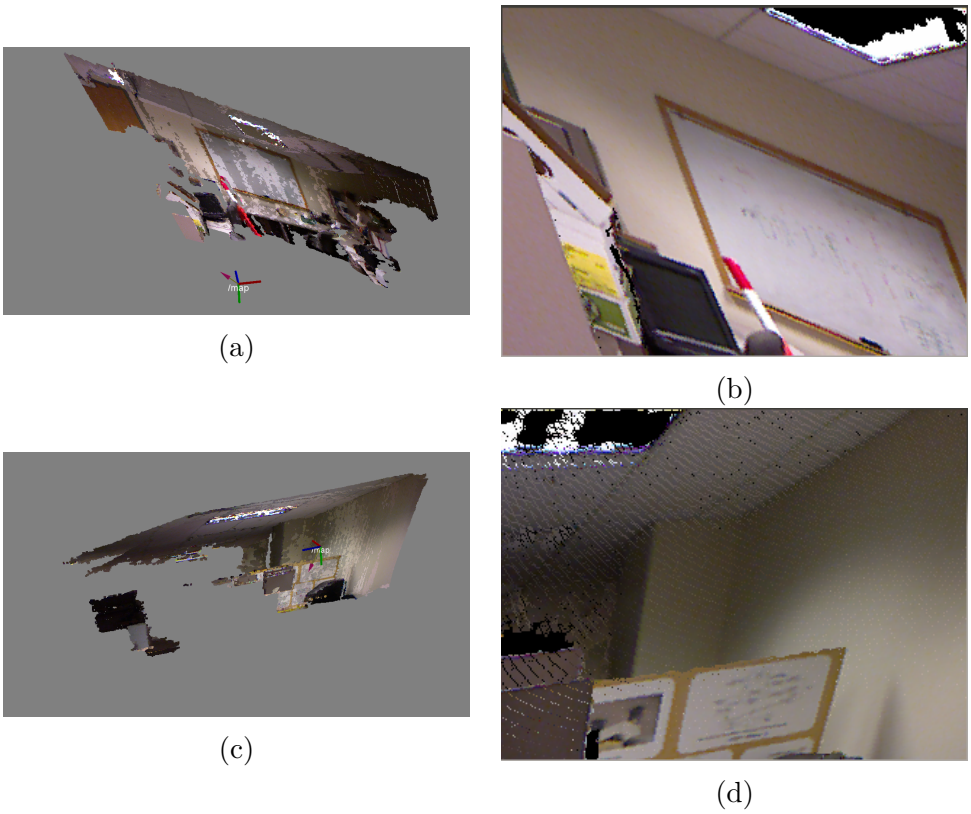


(c)



(d)

Figure 6.8: The distorted point clouds and output images for View 1 and View 2 resulting from the second transformation test. In this test, the sampled maps defined in the world frame were scaled by 1.5 in the x direction and scaled by 0.75 in the z direction. Figure 6.8a shows $\bar{M}$ at View 1 and Figure 6.8b shows I_s at View 1. Figure 6.8c shows $\bar{M}$ at View 2 and Figure 6.8d shows I_s at View 2.



(a) (b) (c) (d)

Figure 6.9: The distorted point clouds and output images for View 1 and View 2 resulting from the third transformation test. In this test, the sampled maps were skewed along the x and y dimensions. Figure 6.9a shows $\bar{M}$ at View 1 and Figure 6.9b shows I_s at View 1. Figure 6.9c shows $\bar{M}$ at View 2 and Figure 6.9 shows I_s at View 2.

world frame. This results in a slight skewing of the image, which would not occur with perfect alignment.

6.4 Concluding Remarks

Overall, the method of map transformation and projection was effective. The projected images appeared realistic and the distorted map provides consistent distortion across a variety of poses. The 3-4 Hz rate is suitable and capable of keeping up with the RGB-D SLAM System which typically runs at a rate of 3 Hz. Run time could be improved using a better optimized Hausdorff measurement algorithm. Additional run time improvements could be achieved using more stringent thresholds for τ_4 and θ so as to limit the

number of Hausdorff distances that must be computed.

Chapter 7

Conclusion

7.1 Summary

This thesis examined the use of interest point detectors, descriptor extractors and matchers to perform pairwise alignment in RGB-D SLAM. Analysis of the effects that their parameters had on feature localization and matching was performed, and the best settings for each detector, extractor, and matcher combination were determined. A self-tuning BRISK-AGAST detector was also developed, which was capable of supplying a constant number of features per image despite changes in image composition.

It was found that an Adjustable BRISK-AGAST detector in combination with a BRISK descriptor and a brute-force matcher using a Hamming distance metric produced the best results in RGB-D SLAM. A setting using a detector with feature bounds of 370-371, no scale-pyramid, and an extractor sampling pattern of scale 1.1 produced the best results, with trajectory accuracy comparable to the SURF configuration and half the run time.

An application of the RGB-D SLAM System was developed that allowed a user to change the appearance of an environment viewed through a camera in a consistent manner. A technique for applying affine transformations to the RGB-D SLAM map was formulated and a method of sampling the map using Hausdorff distance was devised. This technique allowed for efficient generation of images of the distorted map without the need to reason about occlusion. The application was able to generate photo-realistic views of the distorted scene at a rate of 3-4 Hz. Images of the map showed a consistent structure in the distorted map at a variety of poses.

7.2 Future Work

The rise in the popularity of RGB-D cameras in recent years has created a demand for good RGB-D feature points. Existing RGB feature points do work with RGB-D images, but interest points selected in RGB images tend to be near corners and edges where there is a sharp discontinuity in depth resulting in a decrease in the quality of the depth measurement. Feature detectors and descriptors that leverage the available depth data should be able to identify more distinct locations in the scene that can be found more consistently. Such detectors would greatly improve the range image sampling process prior to pairwise alignment. Descriptor extractors that consider the surface normal near an interest point and adjust the sampling pattern accordingly would produce descriptors that are much more robust to changes in perspective. Better descriptors would reduce the number of match outliers, allowing for a faster and more accurate RANSAC based visual odometry estimate.

The update rules for the Adjustable BRISK-AGAST detector could also be improved. Smarter update rules inspired by a model of the relationship between the threshold value, image characteristics, and the number of features would be more responsive and better able to regulate the feature count.

Along the same lines, a self-tuning SURF detector would be of great benefit to the RGB-D SLAM System. Such a detector would produce detection results comparable to the current implementation while halving detection times.

The map distortion application could be improved by a more informed method of sampling the map. Under certain transformations and poses, the map sampling algorithm was unable to provide enough diversity in the keyframes, resulting in grainy looking images. Incorporating the affine transformation used to distort the map into the keyframe sampling process would help identify locations where sampled map is sparse and chose keyframes accordingly, improving sample diversity and image quality.

References

[1] F. Endres, J. Hess, N. Engelhard, J. Sturm, D. Cremers, and W. Burgard, "An evaluation of the RGB-D SLAM system," in *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, 2012, pp. 1691–1696.

[2] N. Fioraio and K. Konolige, "Realtime visual and point cloud SLAM," in *Proc. of the RGB-D Workshop on Advanced Reasoning with Depth Cameras at Robotics: Science and Systems Conf.(RSS)*, vol. 27, 2011.

[3] P. Henry, M. Krainin, E. Herbst, X. Ren, and D. Fox, "RGB-D mapping: Using depth cameras for dense 3D modeling of indoor environments," in *12th International Symposium on Experimental Robotics (ISER)*, vol. 20, 2010, pp. 22–25.

[4] D. Nistér, O. Naroditsky, and J. Bergen, "Visual odometry," in *Computer Vision and Pattern Recognition, Proceedings of the 2004 IEEE Computer Society Conference on*, vol. 1. IEEE, 2004, pp. 652–659.

[5] D. Scaramuzza and F. Fraundorfer, "Visual odometry [tutorial]," *Robotics & Automation Magazine, IEEE*, vol. 18, no. 4, pp. 80–92, 2011.

[6] F. Fraundorfer and D. Scaramuzza, "Visual odometry : Part II: Matching, robustness, optimization, and applications," *Robotics Automation Magazine, IEEE*, vol. 19, no. 2, pp. 78–90, 2012.

[7] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International Journal of Computer Vision (IJCV)*, vol. 60, no. 2, pp. 91–110, 2004.

[8] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (SURF)," *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008.

[9] S. Leutenegger, M. Chli, and R. Y. Siegwart, "Brisk: Binary robust invariant scalable keypoints," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, 2011, pp. 2548–2555.

[10] A. Alahi, R. Ortiz, and P. Vandergheynst, "Freak: Fast retina keypoint," in *Computer Vision and Pattern Recognition, Proceedings of the 2012 IEEE Computer Society Conference on*, 2012, pp. 510–517.

[11] E. Mair, G. D. Hager, D. Burschka, M. Suppa, and G. Hirzinger, "Adaptive and generic corner detection based on the accelerated segment test," in *Computer Vision–ECCV 2010*. Springer, 2010, pp. 183–196.

[12] M. Muja and D. G. Lowe, "Fast approximate nearest neighbors with automatic algorithm configuration," in *International Conference on Computer Vision Theory and Applications (VISSAPP09)*, 2009, pp. 331–340.

[13] Q. Lv, W. Josephson, Z. Wang, M. Charikar, and K. Li, "Multi-probe LSH: efficient indexing for high-dimensional similarity search," in *Proceedings of the 33rd international conference on Very large data bases*. VLDB Endowment, 2007, pp. 950–961.

[14] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. 14, no. 2, pp. 239–256, 1992.

[15] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[16] J. J. Craig, *Introduction to Robotics: Mechanics and Control*, 3rd ed. Prentice Hall, 2004.

[17] G. R. Bradski and V. Pisarevsky, "Intel's computer vision library: applications in calibration, stereo segmentation, tracking, gesture, face and object recognition," in *Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on*, vol. 2. IEEE, 2000, pp. 796–797.

[18] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler, and A. Ng, "ROS: an open-source robot operating system," in *Robotics and Automation (ICRA) Workshop on Open Source Robotics, 2009 IEEE International Conference on*, Kobe, Japan, May 2009.

[19] V. Olshevsky, "Notes on applied linear algebra," 2006. [Online]. Available: http://www.math.uconn.edu/~olshevsky/classes/fall06/math227/math227.php

[20] "Coodinate frames, transforms, and tf," 2013. [Online]. Available: http://ros.org/wiki/tf/Overview/Transformations

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

[22] D. A. Forsyth and J. Ponce, *Computer Vision: A Modern Approach*, 1st ed. Prentice Hall, 2002.

[23] M. Shah, *Fundamentals of Computer Vision*. Orlando, FL: University of Central Florida, Dec. 1997.

[24] F. Rothganger, S. Lazebnik, C. Schmid, and J. Ponce, "3D object modeling and recognition using local affine-invariant image descriptors and multi-view spatial constraints," *International Journal of Computer Vision*, vol. 66, no. 3, pp. 231–259, 2006.

[25] S. Lazebnik, C. Schmid, and J. Ponce, "Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories," in *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, vol. 2. IEEE, 2006, pp. 2169–2178.

[26] S. Kim, K.-J. Yoon, and I. S. Kweon, "Object recognition using a generalized robust invariant feature and Gestalt's law of proximity and similarity," *Pattern Recognition*, vol. 41, no. 2, pp. 726–741, 2008.

[27] C. Tomasi and T. Kanade, "Shape and motion from image streams under orthography: a factorization method," *International Journal of Computer Vision*, vol. 9, no. 2, pp. 137–154, 1992.

[28] M. Brown and D. G. Lowe, "Automatic panoramic image stitching using invariant features," *International Journal of Computer Vision*, vol. 74, no. 1, pp. 59–73, 2007.

[29] E. Rosten and T. Drummond, "Fusing points and lines for high performance tracking," in *Computer Vision (ICCV2005), Tenth IEEE International Conference on*, vol. 2, 2005, pp. 1508–1515.

[30] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in *European Conference on Computer Vision*, vol. 1, May 2006, pp. 430–443.

[31] S. M. Smith and J. M. Brady, "Susan - a new approach to low level image processing," *International Journal of Computer Vision*, vol. 23, pp. 45–78, 1995.

[32] W. Hamilton and W. Hamilton, *Elements of quaternions*. Longmans, Green, & Co., 1866.

[33] W. Robinett and R. Holloway, "The visual display transformation for virtual reality," *Presence: Teleoperators and Virtual Environments*, vol. 4, no. 1, pp. 1–23, 1995.

[34] J. Diebel, "Representing attitude: Euler angles, unit quaternions, and rotation vectors," *Matrix*, 2006.

[35] K. Shoemake, "Animating rotation with quaternion curves," *ACM SIGGRAPH computer graphics*, vol. 19, no. 3, pp. 245–254, 1985.

[36] H. Choset, K. M. Lynch, S. Hutchinson, G. A. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of Robot Motion: Theory, Algorithms, and Implementations*. Cambridge, MA: MIT Press, June 2005.

[37] Y. Großekatthöfer, "Introduction into quaternions for spacecraft attitude representation," 2012. [Online]. Available: http://www.tu-berlin.de/fileadmin/fg169/miscellaneous/Quaternions.pdf

[38] K. Shoemake, "Quaternions," 2013. [Online]. Available: http://www.cs.ucr.edu/~vbz/resources/quatut.pdf

[39] A. S. Hathaway, *A Primer of Quaternions*. Macmillan and Co., 1896.

[40] E. W. Cheney and D. R. Kincaid, *Numerical mathematics and computing*. Brooks/Cole, 2012.

[41] I. Markovsky and S. Van Huffel, "Overview of total least-squares methods," *Signal processing*, vol. 87, no. 10, pp. 2283–2302, 2007.

[42] P. C. Mahalanobis, "On the generalized distance in statistics," in *Proceedings of the national institute of sciences of India*, vol. 2, no. 1. New Delhi, 1936, pp. 49–55.

[43] R. J. Michaels, "A new closed-form approach to absolute orientation," M.S. thesis, Lehigh University, Bethlehem, Pennsylvania, 1999.

[44] K. S. Arun, T. S. Huang, and S. D. Blostein, "Least-squares fitting of two 3-D point sets," *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, vol. PAMI-9, no. 5, pp. 698 –700, Sept. 1987.

[45] O. Sorkine, "Least-squares rigid motion using SVD," 2007. [Online]. Available: http://igl.ethz.ch/projects/ARAP/svd_rot.pdf

[46] T. S. Huang, S. D. Blostein, and E. A. Margerum, "Least-squares estimation of motion parameters from 3-D point correspondences," in *Computer Vision and Pattern Recognition, Proceedings of the 1986 IEEE Conference on*, 1986, pp. 24–26.

[47] D. W. Eggert, A. Lorusso, and R. B. Fisher, "Estimating 3-D rigid body transformations: a comparison of four major algorithms," *Machine Vision and Applications*, vol. 9, no. 5-6, pp. 272–290, 1997.

[48] T. Bailey and H. Durrant-Whyte, "Simultaneous localization and mapping (SLAM): Part II," *Robotics & Automation Magazine, IEEE*, vol. 13, no. 3, pp. 108–117, 2006.

[49] H. Durrant-Whyte and T. Bailey, "Simultaneous localization and mapping: Part I," *Robotics & Automation Magazine, IEEE*, vol. 13, no. 2, pp. 99–110, 2006.

[50] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit, "FastSLAM: a factored solution to the simultaneous localization and mapping problem," in *Proceedings of the National conference on Artificial Intelligence*, 2002, pp. 593–598.

[51] N. Metropolis and S. Ulam, "The Monte Carlo method," *Journal of the American Statistical Association*, vol. 44, no. 247, pp. 335–341, 1949.

[52] R. Y. Rubinstein, *Simulation and the Monte Carlo method*. Wiley-Interscience, 2009, vol. 190.

[53] F. Lu and E. Milios, "Globally consistent range scan alignment for environment mapping," *Autonomous Robots*, vol. 4, no. 4, pp. 333–349, 1997.

[54] F. Lu and E. Milios, "Robot pose estimation in unknown environments by matching 2D range scans," *Journal of Intelligent and Robotic Systems*, vol. 18, no. 3, pp. 249–275, 1997.

[55] J.-S. Gutmann and K. Konolige, "Incremental mapping of large cyclic environments," in *Computational Intelligence in Robotics and Automation, 1999. CIRA'99. Proceedings. 1999 IEEE International Symposium on*, 1999, pp. 318–325.

[56] K. Konolige, "Large-scale map-making," in *Proceedings of the 19th national conference on Artificial intelligence*, 2004, pp. 457–463.

[57] G. Grisetti, R. Kummerle, C. Stachniss, and W. Burgard, "A tutorial on graph-based SLAM," *Intelligent Transportation Systems Magazine, IEEE*, vol. 2, no. 4, pp. 31–43, 2010.

[58] R. Kummerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard, "g2o: A general framework for graph optimization," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 3607–3613.

[59] C. Audras, A. Comport, M. Meilland, and P. Rives, "Real-time dense appearance-based SLAM for RGB-D sensors," in *Australasian Conf. on Robotics and Automation*, 2011.

[60] G. Grisetti, S. Grzonka, C. Stachniss, P. Pfaff, and W. Burgard, "Efficient estimation of accurate maximum likelihood maps in 3D," in *Intelligent Robot Systems (IROS), Proceedings of the 2007 International Conference on*. IEEE, 2007, pp. 3472–3478.

[61] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "ORB: An efficient alternative to SIFT or SURF," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, 2011, pp. 2564–2571.

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

[63] R. Kümmerle, B. Steder, C. Dornhege, M. Ruhnke, G. Grisetti, C. Stachniss, and A. Kleiner, "On measuring the accuracy of SLAM algorithms," *Autonomous Robots*, vol. 27, no. 4, pp. 387–407, 2009.

[64] H. Alt and M. Godau, "Measuring the resemblance of polygonal curves," in *Proceedings of the eighth annual symposium on Computational geometry*, 1992, pp. 102–109.

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