Significance of Omnidirectional Fisheye Cameras for Feature-based Visual SLAM

by

Raphael Chang

S.B., Massachusetts Institute of Technology (2019)

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2020

Author .....
Department of Electrical Engineering and Computer Science
December 6, 2019

Certified by .....
Nicholas Roy
Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

Significance of Omnidirectional Fisheye Cameras for Feature-based Visual SLAM

by

Raphael Chang

Submitted to the Department of Electrical Engineering and Computer Science
on December 6, 2019, in partial fulfillment of the
requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Robust GPS-denied navigation of mobile robots is becoming increasingly important as robots become more ubiquitous. Cameras are powerful sensors for this application due to their low cost and high information density. The task of using camera-based computer vision techniques for navigation is typically referred to as visual simultaneous localization and mapping (SLAM), where a robot both estimates its pose and reconstructs its environment simultaneously using only cameras.

Most existing work for visual SLAM relies on the use of the pinhole camera model, which requires that images from wider angle, more distorted lenses be rectified before they are usable. This limits the field of view well below 180 degrees. However, cameras with omnidirectional fisheye lenses can see much more of their surroundings, which suggests they may be beneficial for the visual SLAM task; this hypothesis is supported by the trend that recent commercial products that rely on robust visual navigation use fisheye cameras.

In this thesis, we explore the apparent discrepancy between the types of cameras traditionally used for navigation tasks in the research community and in industry where robustness is critical. We propose that the scarcity of work using omnidirectional cameras is due to an ill-formed belief that adapting fisheye lenses into traditional computer vision algorithms is infeasible or not worth the effort required to redesign those algorithms. To show this, a benchmarking suite for stereo visual SLAM was developed using traditional feature-based visual odometry algorithms. The building block components of visual SLAM, including feature correspondence, odometry, and reconstruction, were evaluated for both fisheye and perspective cameras. The results show that not only do omnidirectional fisheye cameras easily plug into existing algorithms with minimal modification, they also result in better performance for navigation tasks than perspective cameras with limited field of view.

Thesis Supervisor: Nicholas Roy
Title: Professor of Aeronautics and Astronautics

Acknowledgments

I would like to firstly thank my advisor, Nick Roy, for giving me the opportunity to join his group and work on exciting research, as well as supporting me with invaluable guidance throughout the challenges of this thesis. I would also like to thank the members of RRG for their insightful advice, with special gratitude to Jake Ware and John Carter for their devoted mentorship, without which this thesis would not have been possible.

I am of course most grateful to my family, especially my parents, and my friends, both at MIT and at home, for their unwavering support through thick and thin during my four years at MIT.

Contents

List of Figures

List of Tables

Chapter 1

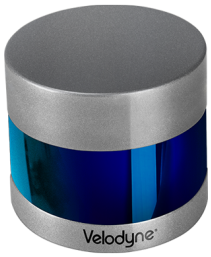
Introduction

1.1 Motivation

As robots are increasingly integrated into our daily lives, more focus has been placed on robustness in the real world. The combination of smaller, more powerful computers and sensors as well as more efficient algorithms have led to the rise of mobile robots such as autonomous cars, aerial photography drones, and search and rescue drones.

A particular type of these mobile robots known as micro-aerial vehicles (MAVs) are an active area of research due to their size and aerial mobility having a wide range of applications. The challenge of MAVs stems from their limited size and power. Because MAVs must be able to navigate in unknown environments with potentially limited communication, onboard computational resources, and perception systems, it can be challenging to develop robust autonomy solutions.

One of the key tasks for autonomous flight is estimating the vehicle's state in its surrounding 3D environment. One way to estimate the vehicle's position is with a GPS, but this requires wireless communication that is not available in many environments, most notably indoors. Other sensors must therefore be used to estimate the vehicle's state as well as map its surroundings, which GPS also cannot provide. The task of estimating the vehicle's pose (position and orientation) while simultaneously building a map of the obstacles around it and localizing itself relative to that map is known as the *simultaneous localization and mapping* (SLAM) problem. The challenge



(a) LiDAR

(b) Active RGB-D Camera

(c) Stereo Camera

(d) Monocular Camera

Figure 1-1: Different sensor modalities that can be used for SLAM

of simultaneously computing the pose of a vehicle relative to a map while building a map relative to the pose, all in real time, has led to SLAM being a very active area of research with many existing techniques.

Many different sensor modalities can be used for SLAM, each with different trade-offs. Some example sensors are shown in figure 1-1. For a power-constrained MAV, the choice of sensor modality must take into account the size, weight, and power requirement of the sensor. Sensors that perceive depth directly, such as LiDARs and active depth cameras (i.e., time-of-flight or structured light cameras), reduce algorithmic complexity because the depth or range of each pixel is generally provided by the sensor itself and does not have to be computed downstream of the sensor signal. However, LiDARs are large and power-consuming; structured light and time-of-flight cameras have limited range and perform poorly in sunlight. On the other hand, passive optical imaging cameras are able to provide high resolution data and are low power, lightweight, and inexpensive. A stereo camera pair can be used to to passively determine depths of pixels. The drawback of cameras is the increase in

(a) Perspective lens (b) Fisheye lens (c) Catadioptric camera

Figure 1-2: Various lenses of different types and FOVs

computation required on the vehicle to process the data, because using these sensors for navigation and localization requires solving the computer vision problem of associating points between images to infer depth and consequently estimate the pose of the camera, and performing these inference procedures robustly and accurately is still an open problem. In addition, inertial measurement units (IMUs), which measure linear accelerations and angular rates, may be added to provide more information to the algorithms, at the cost of additional complexity. Given these tradeoffs, this thesis focuses on the exclusive use of cameras to perform the SLAM task, a problem known as *visual SLAM*.

1.1.1 Why Omnidirectional Fisheye Cameras?

An important variable of vision systems is the camera's field of view (FOV). The FOV is the angular coverage of the scene in front of the camera and is determined by the geometry of the image sensor and lens and their relative positions to each other. Lenses of a wide range of FOVs exist, some of which are shown in figure 1-2. In addition to traditional perspective lenses that typically have smaller FOVs less than 120 degrees, cameras that have more than 180 degrees FOV are known as *omnidirectional cameras*. A common lens type used for omnidirectional cameras is the *fisheye* lens, which exists for FOVs up to 280 degrees. A well-known alternative is the catadioptric camera, which uses spherical or parabolic mirrors to cover nearly 360 degrees. While these cameras are able to provide very large FOVs, the large FOVs come at the expense of extreme *radial distortion*, an optical aberration that

Figure 1-3: Example images with radial distortion (left) and without (right)

causes physically straight lines to appear curved in the image, which increases as a function of the radial distance from the center of the image. Figure 1-3 shows an example of radial distortion. Because simple camera models like the pinhole model cannot model this distortion and rely on straight objects being straight in the image, the history of computer vision has been focused on perspective cameras that have minimal distortion. As a wide-angle image cannot be reasonably undistorted without losing a significant portion of its FOV [58, 49], thus rendering them pointless, traditional computer vision algorithms have relied on using perspective cameras without much consideration for wider FOVs. However, there are many potential benefits to using wider FOVs for navigation tasks. Because visual SLAM relies on tracking the environment around the camera, it is intuitively more desirable to have a wider FOV so that more of the environment can be seen in the instantaneous image and over longer periods of time as the camera moves. Seeing more of the environment would not only provide more spatial information to the navigation algorithms, but also help the algorithms be more tolerant to situations where portions of the scene are difficult to track or become obscured. This suggests that omnidirectional cameras are ideal for the visual SLAM task. Specifically, fisheye cameras are preferred for mobile robots due to their compactness compared to the bulkiness of catadioptric cameras.

Despite the potential benefits of omnidirectional fisheye cameras, the research surrounding visual SLAM has still been focused on narrow FOV cameras due to the reliance of the task on traditional computer vision algorithms which have been de-

Figure 1-4: Commercial products that use fisheye cameras for navigation

veloped for perspective cameras. All open-source, publicly available visual SLAM frameworks rely on rectifying images, or shifting the distorted pixels to make straight objects straight, which limits the usable FOV to well below 180 degrees [49]. Open-source libraries like OpenCV [13] also provide implementations that only support pinhole models and distortion models for rectification. In addition, there are significantly fewer datasets available that provide images from fisheye cameras.

As a counterpoint, we consider the recent trend in commercially viable mobile products that rely on visual navigation techniques like SLAM. Such products include an autonomous video drone from Skydio, AR/VR headsets from Microsoft and Oculus, and a visual odometry tracking camera from Intel, shown in figure 1-4. All of these products exclusively use fisheye cameras for navigation. As commercial products that must function in a wide variety of uncontrolled environments and conditions, robustness is of crucial importance. Therefore, their use of fisheye cameras suggests that wide-angle lenses are beneficial for visual navigation systems. This raises the question of why the academic and open-source systems have not been more active in trying to use this technology.

There are a few potential reasons for the discrepancy between the public trend and commercial development. The first is a reluctance to adapt the traditional computer vision algorithms used as building blocks for visual SLAM to support omnidirectional cameras, due to a belief that the distortion will cause the algorithms to fail (i.e., for feature correspondence algorithms), or that replacing the pinhole camera model is challenging problem (i.e., for geometric algorithms). The second reason is a lack of motivation to solve the challenges from the previous point, due to a lack of evidence

showing the benefits of using omnidirectional fisheye cameras for the SLAM task. Although the trend set by commercial products already suggests concrete reasons to use wide-angle cameras, the goal of this thesis is to explicitly show that not only do omnidirectional fisheye cameras improve the robustness of visual SLAM, traditional algorithms still work with and are easily adaptable to omnidirectional fisheye cameras. There is therefore little reason to use narrow FOV cameras for most computer vision tasks, including the task of visual navigation, unless the task explicitly requires detailed, high resolution imaging of a small portion of a scene.

1.2 Thesis Outline

The remainder of the thesis is structured as follows. Chapter 2 discusses the technical background of the foundational algorithms required to understand traditional feature-based visual odometry and SLAM. Chapter 3 discusses related work in visual navigation, for both traditional and omnidirectional cameras. The remaining chapters will detail the work done to adapt the traditional computer vision algorithms to work with omnidirectional fisheye cameras, and evaluate their performance compared to narrow FOV cameras. Chapter 4 focuses on the Lucas-Kanade feature tracking algorithm. Chapter 5 focuses on descriptor-based feature matching. Chapter 6 focuses on the rest of the visual SLAM pipeline, specifically stereo matching, odometry, and reconstruction. Finally, chapter 7 concludes the thesis.

Chapter 2

Background

This chapter begins with an introduction and overview of the individual components of the visual SLAM pipeline. Section 2.1 formulates the problem and discusses the high-level structure of the pipeline and where each of the individual components fit in to it. Section 2.2 introduces the feature correspondence problem, and is split into two subsections; 2.2.1 describes the frame-to-frame incremental feature tracking problem and the Lucas-Kanade algorithm, and 2.2.2 discusses the descriptor-based feature matching problem. Section 2.3 introduces geometric concepts that connect 2D pixel space concepts to 3D space, in particular camera models and epipolar geometry. Section 2.4 discusses the odometry problem, or frame-to-frame incremental pose estimation, and the algorithms related to it. Section 2.5 discusses the reconstruction problem, also known as mapping or depth estimation, including stereo matching, a specific case of depth estimation. Finally, section 2.6 covers global optimization techniques that attempt to solve for a globally consistent trajectory and map.

2.1 Visual Simultaneous Localization and Mapping

For a robot to navigate an environment containing obstacles, it typically needs to have both its *pose*, or position and orientation in the world, and a *map* of the environment around it. Once it has both of these, it can then plan collision-free paths to a given goal location within the environment. Because the pose needs to be known

for a map to be built around it, and the map needs to be known to localize a pose relative to the map, both quantities need to be computed simultaneously. Visual simultaneous localization and mapping (SLAM) is the task of using cameras to perceive the environment, and using the environment to localize the pose of the cameras while building a 3D map of the environment. The challenge of performing this fundamentally three-dimensional task using a sensor that only provides 2D images is the basis of the field of computer vision. As such, visual SLAM relies on algorithms taken from the broader study of computer vision to form an algorithmic foundation.

Visual SLAM techniques generally fall into two categories, *feature-based* (also referred to as indirect) or *direct* methods. Feature-based methods rely on extracting a limited number of *features*, or small patches of the image containing high contrast corners, lines, or blobs, associating them between frames, and using them as landmarks for the rest of the pipeline. On the other hand, direct methods use raw pixel values instead of features to directly solve for the relative transformation of the pose of the camera between images by framing it as a non-linear least squares optimization problem. The optimization problem solves for the optimal transform such that the difference between brightness values of pixels in the original image and the same pixels projected into the transformed frame (assuming known depth) is minimized. Feature-based methods are more popular and have a longer history, but direct methods have shown benefit from not needing the computationally expensive feature extraction step and being able to use more of the image due to not being limited to where features are extracted [29]. Despite this, feature-based methods are still more widely used (therefore more proven) because they are simpler to understand and analyze (compared to a single large optimization problem), this thesis focuses on feature-based methods for visual SLAM. However, the challenges of using fisheye cameras for both methods are similar, so we expect the results to carry over to direct methods as well.

2.1.1 Problem Formulation

To formulate the feature-based visual SLAM problem, consider a robot moving through an environment and capturing images with a camera system at discrete times k. At

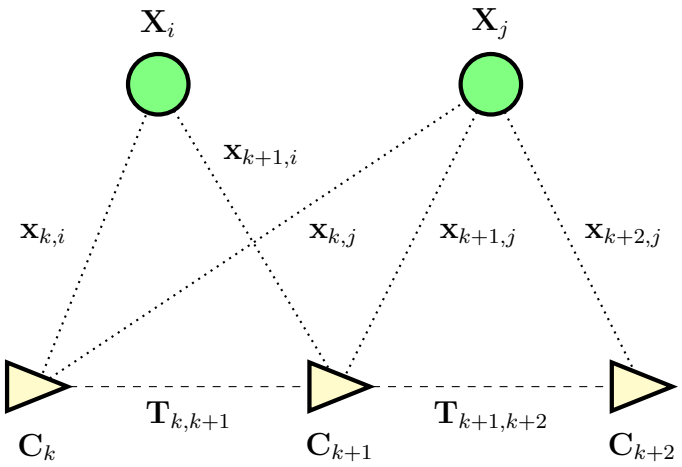


Figure 2-1: Feature-based visual SLAM problem formulation. $\mathbf{C}_k$ is the camera pose at time step k, $\mathbf{T}_{k,k+1}$ is the transformation between the poses at time k and $k+1$, $\mathbf{X}_i$ is the 3D world coordinate of landmark i, and $\mathbf{x}_{k,i}$ is the observation of landmark i by the camera at time step k, projected through a camera model as a pixel coordinate.

time k, let the set of images taken so far by the camera system be $I_{0:k} = \{I_0, ..., I_k\}$. In the case of a stereo system, each image I_n is actually a pair of images $I_{n,1}$ and $I_{n,2}$. Let the coordinate frame of the robot be the coordinate frame of the camera or one of the cameras.

The relative transformation between the poses of the camera at times k and $k-1$ can be represented as a rigid-body transformation $\mathbf{T}_{k,k-1} \in \mathbb{R}^{4\times4}$:

$$\mathbf{T}_{k,k-1} = \begin{bmatrix} \mathbf{R}_{k,k-1} & \mathbf{t}_{k,k-1} \\ 0 & 1 \end{bmatrix} \tag{2.1}$$

where $\mathbf{R}_{k,k-1} \in SO(3)$ is a rotation matrix and $\mathbf{t}_{k,k-1} \in \mathbb{R}^{3\times1}$ is a translation vector. Let the starting pose of the camera be the origin of the world coordinate frame. The camera pose in the world frame at time k, $\mathbf{C}_k$, can then be computed as follows:

$$\mathbf{C}_k = \mathbf{T}_{k,0} = \mathbf{T}_{k,k-1}\mathbf{C}_{k-1} = \prod_{n=1}^{\overset{\curvearrowleft}{k}} \mathbf{T}_{n,n-1} \tag{2.2}$$

Consider a set of m landmarks in the environment with 3D world coordinates $\mathbf{X}_{1:m} = \{\mathbf{X}_1, ..., \mathbf{X}_m\} \subset \mathbb{R}^{3\times1}$. Let $\mathbf{O}_n \subseteq \mathbf{X}_{1:m}$ be the set of landmarks observed by image I_n, projected through a camera model as pixel coordinates $\mathbf{x}_{n,\mathbf{O_n}} \subset \mathbb{R}^{2\times1}$.

The visual SLAM problem can then be stated as follows. At every time $k > 0$, given the landmarks that are observed by both I_{k-1} and I_k, which is $\mathbf{O_{k-1} \cap O_k}$, use the projected pixel coordinates $\mathbf{x}_{k-1,\mathbf{O_{k-1} \cap O_k}}$ in I_{k-1} and $\mathbf{x}_{k,\mathbf{O_{k-1} \cap O_k}}$ in I_k, as well as optionally some of world coordinates $\mathbf{O_{k-1} \cap O_k}$, to determine the transformation $\mathbf{T}_{k,k-1}$ and subsequently the pose $\mathbf{C}_k$. Given the poses, compute or update the landmark world coordinates $\mathbf{O_{k-1} \cap O_k}$. An illustration of the problem formulation is shown in figure 2-1.

2.1.2 Pipeline Overview

An overview of the feature-based visual SLAM pipeline is shown in figure 2-2. The first part in the pipeline for feature-based systems is purely in the 2D image domain. The first step is to extract features using a *feature detector* to efficiently find small image patches that contain high contrast corners, lines, or blobs, and are robust to changes in rotation, scale, and illumination. A deeper overview of feature detection methods is given in section 2.2.1.

Once features are found in one image, the landmarks that those features represent need to be found in successive images. This *feature correspondence* problem can be solved in two ways: features can either be *tracked* into successive images, or *matched* with features extracted in successive images. Feature tracking works by searching for the feature in a neighborhood around its original location, while feature matching compares features extracted independently from both images to find the most similar matches. Details about feature tracking are provided in section 2.2.2, and details about feature matching are provided in section 2.2.3.

In the case of a stereo visual SLAM pipeline, the same feature correspondence techniques can be used to correspond features between the stereo pair and compute depth for each feature, a task known as *stereo matching*. Once the correspondence is found, the known (calibrated) transformation between the stereo pair can be used to compute the depth of the features. Stereo depth estimation is a special case of the reconstruction problem, which will be discussed in section 2.5.

Once feature correspondences between successive frames are known, there are

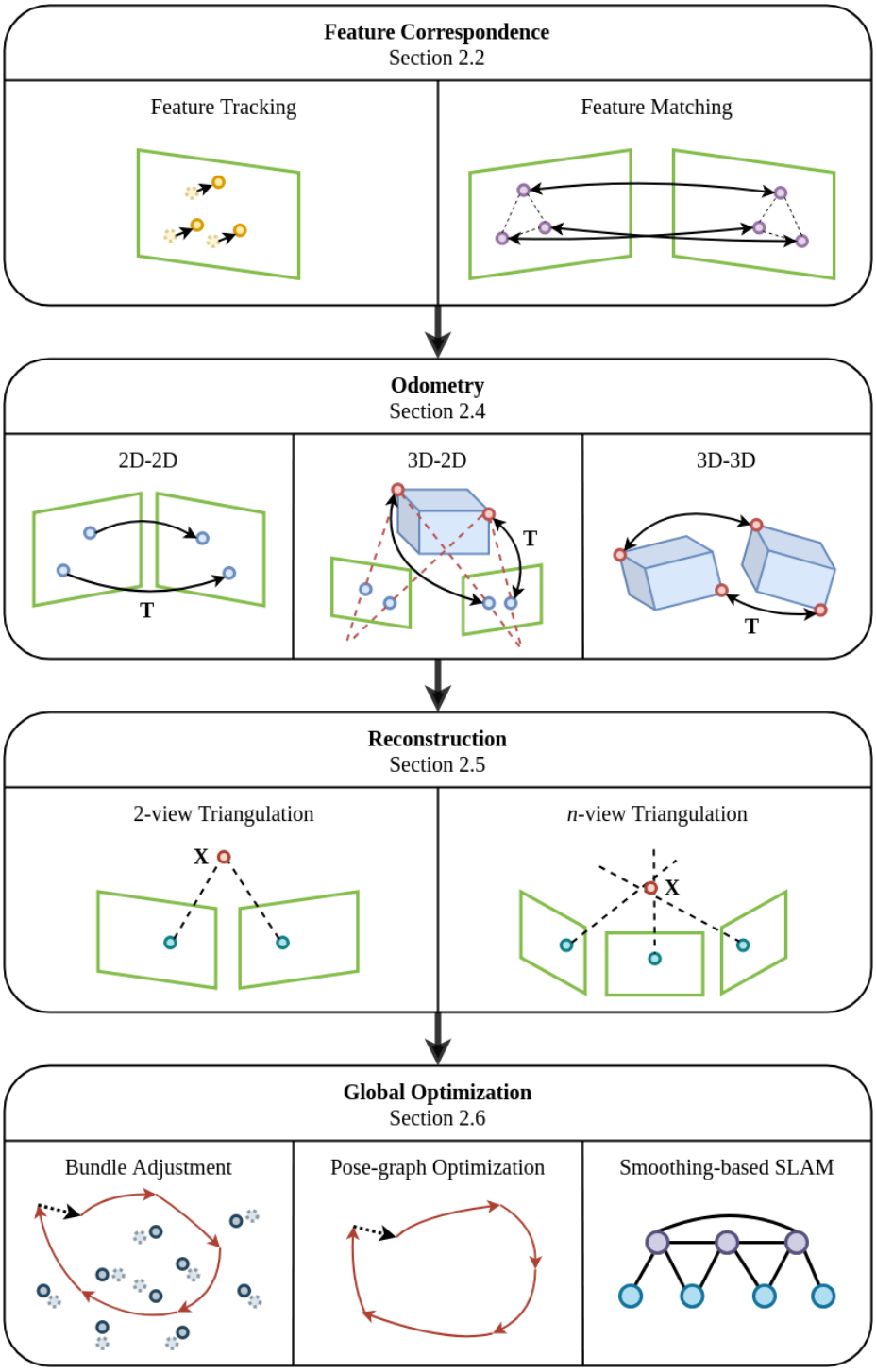


Figure 2-2: Feature-based visual SLAM pipeline diagram

three methods to obtain the relative transform between the camera poses (known as *odometry*), two of which require feature depths to be known. The first is using 2D-to-2D correspondences, which does not require depths and therefore can be used

with a monocular SLAM system (without scale). If depths are known (i.e., with a stereo pair), the information can be leveraged to compute the correct scale. The second method for pose estimation is using 3D-to-2D correspondences, where 3D coordinates of features stereo matched in one frame correspond to 2D pixel locations in the other frame. The camera geometry can then be used to find a camera pose that is consistent with the correspondences. The final pose estimation method uses 3D-to-3D correspondences, where 3D stereo matched features are used to align the camera poses such that the errors in 3D space are minimized. However, stereo depth estimation introduces large uncertainties in the depth direction, and these depth errors have detrimental effects on the alignment of 3D points [63]. Thus, only the 2D-to-2D and 3D-to-2D cases will be considered, and are discussed in detail in section 2.4.

Once the relative poses between frames are known, the 3D structure of the scene can be estimated, a task known as *reconstruction* or depth estimation. Using the known camera poses, the camera geometry can once again be used to compute the 3D coordinates of features observed across multiple frames. The resulting map can then be used in the next frame to compute odometry using 3D-to-2D correspondences, and the process repeats. Reconstruction methods are discussed in detail in section 2.5.

The pipeline introduced so far is known as the task of *visual odometry* [63], where the camera pose is estimated incrementally. Visual odometry is prone to drift because poses are only calculated based on the previous frame so errors accumulate over time, and the trajectory and map lack global consistency. An example of a global consistency problem is the problem of *loop closure*, where a robot determines that it has returned to a location previously visited. Due to drift, the current odometry estimate will not be the same as the original pose. The past trajectory, as well as the map, now have to be corrected so that the current pose matches the pose from when the location was previously visited. There are two main techniques to correct for global consistency, *bundle adjustment* and *pose-graph optimization*, which will be introduced in section 2.6.

2.2 Feature Correspondence

A fundamental computer vision problem, and the first step of any feature-based SLAM pipeline, is determining where a point in one image appears in another. Given features that are detected in one image, the feature correspondence task involves finding the precise locations of those features in another image that was taken from a different perspective. This feature correspondence is done not only for pairs of frames in a temporal sequence, but also between two images in a stereo camera. This section will first discuss the feature detection task, and then cover two methods to find correspondences, tracking and matching.

2.2.1 Feature Detection

Features can only be recognized across different images if they are distinctive and robust to rotation, scale, and lighting changes. The purpose of the feature detection task is to propose image patches that meet this criteria, such as corners, edges, and blobs. An overview of existing feature detectors will now be provided.

The earliest feature detectors were corner detectors that used image gradient-based approaches. The *Moravec corner detector* [60], developed in 1980, found corners by shifting local windows by a few pixels and finding ones that gave large changes in total intensity. In particular, it computes the sum-of-squared-differences (SSD) between the original window and shifted windows in cardinal directions (u, v), where $w(x, y)$ is a window function:

$$E(u, v) = \sum_{x,y} w(x, y)[I(x + u, y + v) - I(x, y)]^2 \tag{2.3}$$

Harris and Stephens [35] improved upon this approach in 1988 by deriving an analytical approximation for the SSD cost using image gradients. By computing the partial derivatives of the image I_x and I_y and using Taylor expansion, the following approximation was derived:

$$E(u,v) \approx \sum_{x,y} w(x,y)[I_x(x,y)u - I_y(x,y)v]^2 = \begin{bmatrix} u & v \end{bmatrix} \mathbf{A} \begin{bmatrix} u \\ v \end{bmatrix} \tag{2.4}$$

$$\mathbf{A} = \sum_{x,y} w(x,y) \begin{bmatrix} I_x(x,y)^2 & I_x(x,y)I_y(x,y) \\ I_x(x,y)I_y(x,y) & I_y(x,y)^2 \end{bmatrix} \tag{2.5}$$

The matrix $\mathbf{A}$ is the *second-moment matrix*. If the Gaussian window $w(x,y)$ contains a corner, $E(u,v)$ should have large variations over all directions of (u,v), so $\mathbf{A}$ should have two large eigenvalues. A function was then derived that used the trace and determinant of $\mathbf{A}$ to determine the magnitude of the eigenvalues rather than computing them directly, which is expensive. The resulting feature detector is known as the *Harris corner detector* and is still a widely used feature detector. Shi and Tomasi [76] later presented an improvement that calculated the minimum of the two eigenvalues directly as a corner score, known as the *Shi-Tomasi corner detector*.

More recently, for applications that require real-time feature detection, the *FAST corner detector* [69] was developed. FAST (Feature from Accelerated Segment Test) considers a set of pixels in a circle around a candidate feature point. It performs a binary test to determine if a set of N contiguous pixels in the circle are all brighter or darker than the candidate point, and the point is classified as a corner if the test passes. The feature detector used by ORB [71] is based on the FAST corner detector, and adds an orientation component. The Adaptive and Generic Accelerated Segment Test (AGAST) [57] feature detector is also inspired by FAST and adds speed and robustness improvements.

In addition to corners, blobs are also commonly used features. Methods to detect blobs in images include Difference-of-Gaussians (DoG) and Determinant-of-Hessian (DoH) approaches. The Difference-of-Gaussians algorithm involves subtraction of a Gaussian blurred version of the image with another Gaussian blurred version with a lower-variance Gaussian kernel. This is the feature detection algorithm used to detect SIFT [53] features. The Determinant-of-Hessian algorithm involves computing the determinant of the Hessian matrix, a matrix of second-order derivatives of the

image, which also results in a blob and saddle point detector. This is the approach used to detect SURF [9], KAZE [4], and AKAZE [64] features.

2.2.2 Feature Tracking

Now that features are detected in one image, the next step is to find those features in subsequent images. One way to do this is to assume that the next image in the sequence has not moved too much relative to the first one. This allows the features to be tracked into the second image, by searching around a neighborhood of the original locations of the features. The feature tracking task is also known as sparse optical flow.

The most widely used feature tracking algorithm remains the *Lucas-Kanade* [55] (LK) feature tracker. The Lucas-Kanade algorithm iteratively solves for an alignment of an image patch (template) with another image such that the intensity difference is minimized in a least-squares sense. As an optical flow algorithm, it assumes the intensity is constant between frames. The tracking problem is formulated as a non-linear optimization problem as follows, where W is the template window (image patch to be tracked) and I_{t+1} is the new image to be tracked into:

$$e = \sum_{\mathbf{x} \in W} [I_t(\mathbf{x}) - I_{t+1}(\mathbf{w}(\mathbf{x}; \mathbf{p}))]^2 \tag{2.6}$$

The function $\mathbf{w}(\mathbf{x}; \mathbf{p})$ is the tracking motion model, also called the *warping function*. It represents the space in which the template can be warped in order to align to the new image. The vector $\mathbf{p}$ is the vector of warping parameters, and is the variable that is solved for during the alignment optimization. If there are priors for where a feature could potentially be (i.e., through the use of an IMU) the warping parameters can be initialized before the optimization. Warping functions can range in complexity which directly impacts the runtime of the optimization. The simplest warping function is the translation model:

$$\mathbf{w}(\mathbf{x}; \mathbf{p}) = \mathbf{x} + \mathbf{b} \tag{2.7}$$

The translation model is fast as it only has two parameters, but can only handle pure translation with minimal rotation, scaling, and other forms of warping. To optimize for an alignment over all forms of affine warping, the affine model can be used:

$$\mathbf{w}(\mathbf{x};\mathbf{p}) = \mathbf{A}\mathbf{x} + \mathbf{b} \tag{2.8}$$

where $\mathbf{A}$ is an affine transformation matrix. However, $\mathbf{A}$ introduces four more parameters so it is computationally expensive. In practice, for real-time applications, the translation model is used as the appearance change between consecutive images is small. This requires that the template be refreshed every frame or nearly every frame so that alignment is done relative to a recent frame. As such, a feature could start to drift over many frames, a potential drawback of the tracking approach.

Although the appearance change between frames is small, is it typically not small enough for the LK algorithm described so far to converge to the correct solution, especially at higher resolutions where the change in pixel location is greater for a given motion. To solve this, a pyramidal approach [12] is used. The image is converted to a pyramidal representation where the image is recursively downsampled in half. The number of pyramids is selected based on the magnitude of expected camera motion. The LK algorithm is first performed on the highest pyramid level (lowest resolution) which gives a rough alignment but handles large offsets. The result from this is then used as the prior for the next highest pyramid level, and the alignment is repeated as such iteratively, until the lowest level which gives a full resolution alignment.

Feature selection from the detection step can also impact tracking performance. The Shi-Tomasi corner detector introduced previously has been shown to select good features for tracking, and as such is widely used in combination with the Lucas-Kanade algorithm, a combination known as the Kanade-Lucas-Tomasi (KLT) feature tracking algorithm. The KLT algorithm will be the feature tracking algorithm evaluated on fisheye cameras in this thesis.

2.2.3 Feature Matching

The other method for feature correspondence is feature matching. Contrary to tracking, which only requires detection in the first frame, feature matching requires detection of features in all frames. Features in one frame are compared to features from another, and the most similar match is used as the correspondence.

To match features robustly, features must be encoded in a way that is invariant to changes in scale, rotation, illumination, etc. *Descriptors* are vectors that are computed for each feature, and are compared to each other using vector distances. Once descriptors are computed, matches can be found in a few ways. The brute-force method involves comparing every feature in the first image for each feature in the second image, and returning the closest one in the descriptor vector space as the match. Nearest-neighbor methods improve efficiency by using data structures to quickly find approximate nearest neighbors in the descriptor vector space.

Designing descriptors that are not only robust to various changes but also compact and fast to compute has been an active computer vision problem with many existing approaches. Before descriptors, features were compared using simple SSD methods. In 1999, Lowe presented the scale-invariant feature transform (SIFT) [53] detector and descriptor. The SIFT detector not only detects features but also ensures invariance to scale and rotation by detecting at various image scales and assigning an orientation to each feature using local image gradient directions. Once these augmented feature points, known as *keypoints*, are found, a 128-dimensional descriptor is computed. A set of 16 4x4 windows around the keypoint is used and gradient magnitudes and orientations are calculated for each window and stored into an 8-bin histogram. The histogram magnitudes are then thresholded and normalized for illumination invariance, and the orientations are subtracted by the keypoint orientation for rotation invariance.

Since SIFT, similar approaches using oriented gradient histograms, like Gradient Location and Oriented Histogram (GLOH) [59], Histogram of Gradients (HoG) [18], and DAISY [81], have been introduced. The speeded up robust features (SURF) [9]

detector and descriptor, introduced in 2006, is partly inspired by SIFT and is several times faster. It uses Haar wavelets to compute keypoint orientations and descriptors. The sum of Haar wavelet responses in the horizontal and vertical directions is used to determine the orientation. To compute the descriptor, Haar wavelet responses are extracted from an oriented set of square sub-regions surrounding the keypoint, yielding a 64-dimensional descriptor. The KAZE [4] descriptor builds on SURF by using nonlinear diffusion filtering as its scale space.

The descriptors introduced so far have all used continuous real values to encode features, and therefore use Euclidean distances to evaluate descriptor similarity. More recently, descriptors have been introduced that use binary vectors, allowing for much faster computation and distance comparison, as well as more efficient memory usage. The Binary Robust Independent Elementary Features (BRIEF) [15] descriptor was introduced in 2010. To compute a BRIEF descriptor, the feature patch is first smoothed with a Gaussian. Pairs of pixels are then selected, and intensity comparisons are done on these pairs. The binary comparison results (greater or less than) are written to a binary string and returned as the descriptor vector. For the matching process, Hamming distance is used instead of Euclidean distance, allowing for much faster feature comparison.

Shortly after the introduction of BRIEF, two more noteworthy binary descriptors were introduced, Oriented FAST and Rotated BRIEF (ORB) [71] and Binary Robust Invariant Scalable Keypoints (BRISK) [45], that both add rotation and scale invariance, which BRIEF lacked, using similar approaches. ORB uses an augmented FAST detector that detects on multiple scales and computes orientation. Descriptors are then computed for the ORB keypoints by rotating the BRIEF algorithm to the keypoint orientation. On the other hand, BRISK uses the AGAST detector. For descriptors, it uses a custom deterministic sampling pattern for binary comparison, that is also pre-scaled and pre-rotated according to the keypoint scale and orientation. In addition to ORB and BRISK, several other binary descriptors have also been introduced, such as Accelerated KAZE (AKAZE) [64], a faster, binary version of KAZE that rivals ORB and BRISK, Fast Retina Keypoint (FREAK) [3], a binary

descriptor inspired by the human retina, and Learned Arrangements of Three Patch Codes (LATCH) [47], which uses comparison of patch triplets.

Compared to feature tracking, which has the advantage of being fast, feature matching has the advantage of being able to handle large motions. It is also less prone to drift because features are redetected in every frame. The drawbacks are that feature detection in every frame is computationally expensive, and the matching algorithm also has high computational complexity (quadratic time for brute force matching). As both feature tracking and matching are viable feature correspondence methods each with their own tradeoffs, this thesis will evaluate both approaches.

2.3 Camera Geometry

The feature-based algorithms discussed so far are all pixel space algorithms, as in they only require images to function, without knowledge of the camera's physical attributes. In order to use any information in the pixel space to infer information about the physical world, we must consider the relationships between pixel coordinates and physical values. This section will discuss concepts that provide these relationships.

2.3.1 Camera Models

The central concept that connects pixel space to 3D space is the *camera model*. Camera models represent the relationship between light rays originating from points in the physical world and their projected locations on the image sensor. Using a camera model, a pixel in the image can be mapped to a single ray originating from the camera origin, such that the pixel value captures the appearance of the physical object that the ray hits. Similarly, 3D points can be projected onto the image sensor plane using a camera model to determine the pixel coordinate at which a 3D point will appear. Many camera models exist to model different types of lenses. Three types of models will be discussed: the pinhole model, which is the simplest ideal camera model, the unified model, a more complex model that can represent wide-angle cameras, and the double sphere model, a modern camera model that has been

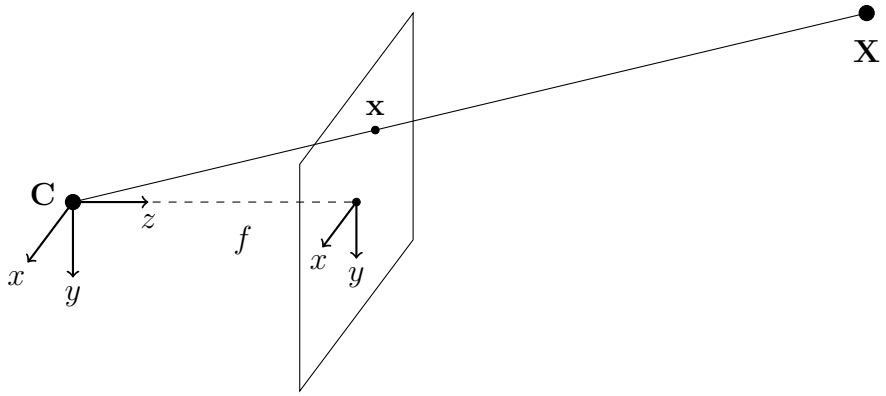


Figure 2-3: Pinhole camera model. A 3D coordinate $\mathbf{X}$ is projected onto an image plane at a distance of focal length f from the camera origin $\mathbf{C}$ as pixel coordinate $\mathbf{x}$.

shown to model wide-angle fisheye lenses well, which is used in this thesis.

Pinhole Camera Model

The traditional and simplest way to model the relationship between 3D coordinates and pixel coordinates is with the pinhole camera model. This model represents an ideal pinhole camera, where light can only enter a small hole and project onto the sensor, as shown in figure 2-3. Because of the simple geometry in this camera model due to all rays remaining straight, simple linear equations can be used to describe the pinhole camera model, derived using similar triangles. Let X, Y, and Z represent 3D coordinates of a point, and u and v represent the pixel coordinates. The parameters of the model are the focal lengths, f_x and f_y, which represent the distance of the sensor from the pinhole, and the principle points, c_x and c_y, which represent the pixel offset from the sensor center to move the pixel coordinate origin to the corner of the image. Using similar triangles as visualized in figure 2-3, the mapping from 3D coordinates to pixel coordinates (projection function) is as follows:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \propto \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.9}$$

The three-dimensional vector on the left hand side of the equation representing

a 2D pixel coordinate is in *homogeneous coordinates*, meaning that it is equal to the right-hand side up to a scalar factor. To calculate u and v, the right-hand side product is obtained, and the resulting vector is scaled so that the third coordinate is one. For this equation, this effectively means that the products of the X and Y coordinates with f_x and f_y respectively are divided by the Z coordinate, which is the same result obtained using the similar triangles derivation. The unprojection function, which maps a pixel coordinate to a 3D ray from the camera origin, can be obtained by inverting the projection matrix.

This model is widely used due to its simplicity and ability to be represented as a simple matrix. However, it fails to capture any inevitable distortion caused by real lenses that are not an infinitesimal pinhole. Typically, the larger the FOV of the lens, the stronger the distortion effect. Therefore, the pinhole model is usually accompanied by a distortion model, which allows the distortion to be rectified such that the pinhole model is satisfied. However, it can be seen that for FOVs at or beyond 180 degrees, the images cannot be represented by the pinhole model as those rays cannot be projected onto the image plane while staying straight. The images are therefore substantially cropped to an FOV less than 180 degrees by the distortion model when rectifying the image. In practice, even with distortion models, the pinhole model is suboptimal for FOVs greater than 120 degrees [49]. Therefore, to model wide-angle lenses such as fisheye, more complex camera models must be considered.

Unified Camera Model

The unified camera model [27] has been shown to model omnidirectional cameras well, in particular for catadioptric cameras for which it is typically used. In this model, a 3D point is first projected onto a unit sphere, and then projected onto the sensor with a pinhole model, as shown in figure 2-4. The use of the sphere allows FOVs beyond 180 degrees to be modeled.

The unified camera model has one additional parameter, α, where $\frac{\alpha}{1-\alpha}$ represents the offset of the unit sphere from the pinhole camera origin. The projection function is defined as follows:

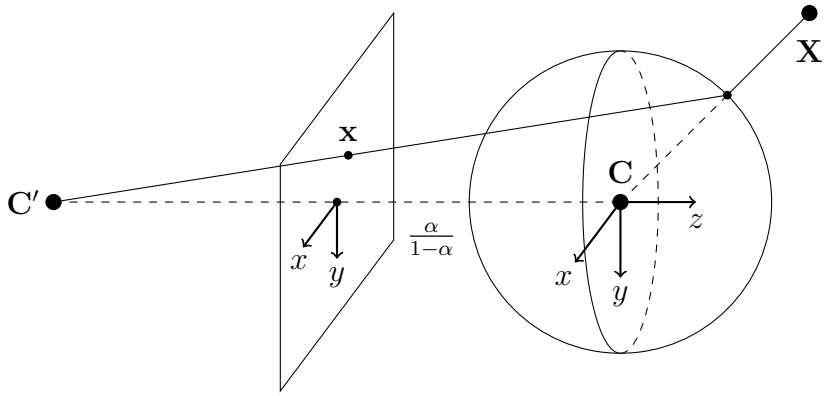


Figure 2-4: Unified camera model. The 3D coordinate $\mathbf{X}$ is projected onto a unit sphere, and then projected into a pinhole model with origin $\mathbf{C}'$ offset by $\frac{\alpha}{1-\alpha}$ as pixel coordinate $\mathbf{x}$.

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \propto \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ \alpha\sqrt{X^2+Y^2+Z^2} + (1-\alpha)Z \end{bmatrix} \tag{2.10}$$

The unprojection function is defined as follows:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \frac{\xi + \sqrt{1+(1-\xi)^2 r^2}}{1+r^2} \begin{bmatrix} \frac{1}{f_x} & 0 & -\frac{c_x}{f_x} \\ 0 & \frac{1}{f_y} & -\frac{c_y}{f_y} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ \xi \end{bmatrix} \tag{2.11}$$

$$r^2 = \left(\frac{u-c_x}{f_x}\right)^2 + \left(\frac{v-c_y}{f_y}\right)^2 \tag{2.12}$$

$$\xi = \frac{\alpha}{1-\alpha} \tag{2.13}$$

Note that for $\alpha = 0$, the model reverts back to the pinhole model. The added nonlinearity of the model introduces complications to many existing algorithms, as the projection function is no longer a linear mapping. Although the unified camera model is able to model omnidirectional cameras, it does not fit fisheye lenses perfectly and still requires an additional distortion model.

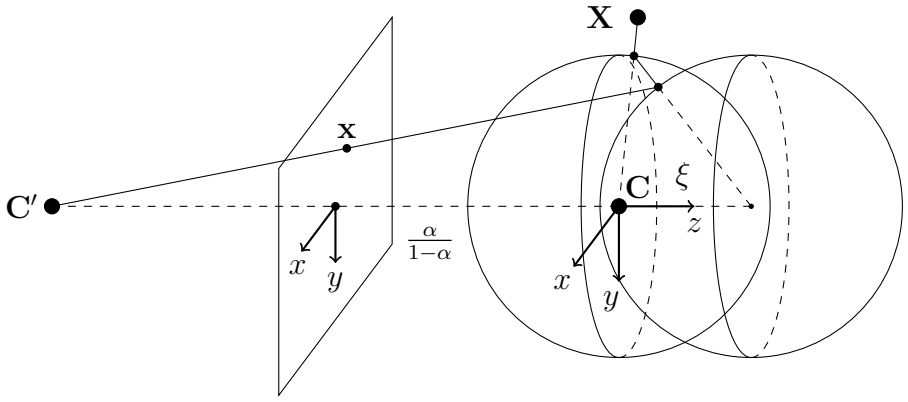


Figure 2-5: Double sphere camera model. The 3D coordinate $\mathbf{X}$ is consecutively projected onto two unit spheres offset by ξ, and then projected into a pinhole model with origin $\mathbf{C}'$ offset by $\frac{\alpha}{1-\alpha}$ from the second sphere as pixel coordinate $\mathbf{x}$.

Double Sphere Camera Model

The double sphere camera model [87] is a modern camera model that is designed to model wide-angle fisheye lenses. It builds on the unified camera model by adding a second unit sphere offset from the first sphere, as shown in figure 2-5. The 3D point is consecutively projected onto the two spheres, and then projected onto the sensor using a pinhole model offset from the second sphere.

The double sphere camera model one additional parameter from the unified camera model, ξ, which is the offset between the two spheres. The projection function is defined as follows:

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \propto \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ \alpha\sqrt{X^2 + Y^2 + (\xi d + Z)^2} + (1-\alpha)(\xi d + Z) \end{bmatrix} \tag{2.14}$$

$$d = \sqrt{X^2 + Y^2 + Z^2} \tag{2.15}$$

The unprojection function is defined as follows:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \frac{m_z\xi + \sqrt{m_z^2 + (1-\xi)^2 r^2}}{m_z^2 + r^2} \begin{bmatrix} \frac{1}{f_x} & 0 & -\frac{c_x}{f_x} \\ 0 & \frac{1}{f_y} & -\frac{c_y}{f_y} \\ 0 & 0 & m_z \end{bmatrix} \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 0 \\ \xi \end{bmatrix} \tag{2.16}$$

$$r^2 = \left(\frac{u - c_x}{f_x}\right)^2 + \left(\frac{v - c_y}{f_y}\right)^2 \tag{2.17}$$

$$m_z = \frac{1 - \alpha^2 r^2}{\alpha\sqrt{1 - (2\alpha - 1)r^2} + 1 - \alpha} \tag{2.18}$$

The double sphere model is currently the state-of-the-art for modeling fisheye lenses. Therefore, this thesis will exclusively use this model for its analysis of SLAM algorithms. Setting α and ξ to zero collapses the model into a pinhole model, allowing for easy comparison of fisheye and perspective cameras using the same model.

Radial Distortion Analysis

Now that fisheye images can be modeled accurately using the double sphere model, we can analyze the radial distortion properties of fisheye cameras more in depth. To do this, it is important to first understand and model the effect radial distortion has on images. Figure 2-6 illustrates how physically straight lines become curved as they are projected through a lens onto the image plane. The effect arises from the image magnification decreasing with distance from the optical axis, causing parts of the image at larger radial distances to be compressed inwards. Physically straight lines that are not in the center of the image are projected onto different radial distances in the image, causing parts of the line to be curved inwards towards the image center. The effect is radially symmetric because of the physical symmetry of camera lenses.

To analyze how much distortion is present in different parts of the image, we can examine the angular resolution, which is the ray angle covered by one pixel, in the radial and tangential directions as a function of distance from the center of the image. A distortion-free image region near the center would have equal radial and tangential angular resolutions, while a distorted region would have unequal resolutions, representing compression along a direction. The difference in ray angles between two

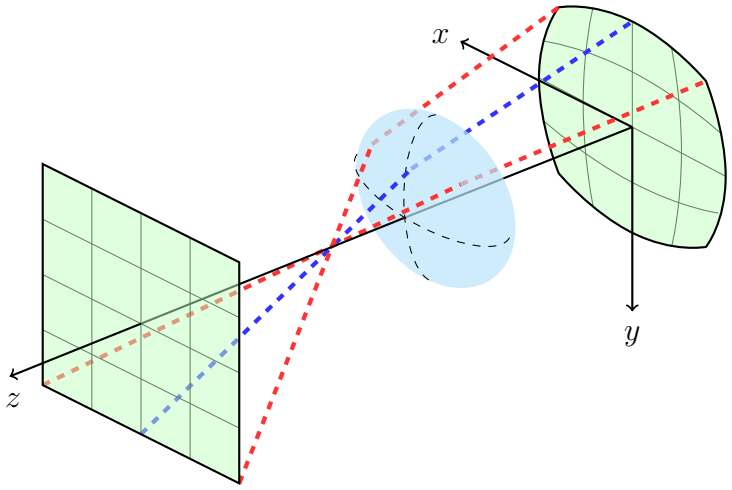


Figure 2-6: Diagram showing geometry of radial distortion. The corners of the square (red rays) are farther from the optical axis than the midpoints of the edges of the square (blue ray) so they are compressed inward by the lens, curving the projection.

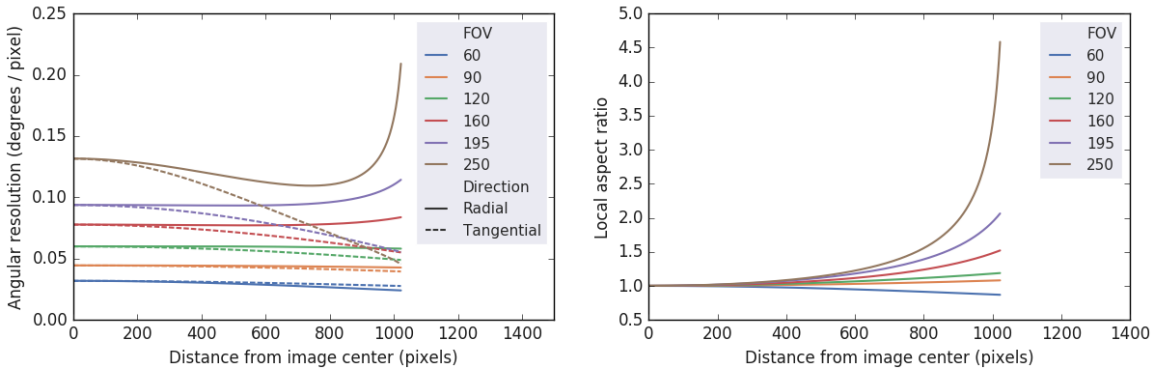


Figure 2-7: Angular resolution in radial and tangential directions (left) and local aspect ratio between the two directions (right) as a function of radial distance for cameras of various FOVs

consecutive pixels (in degrees per pixel) is used to define the angular resolution. Figure 2-7 shows a plot showing the angular resolution over distances from the image center, in both the radial and tangential directions, for cameras of various FOVs including both fisheye and perspective, modeled with the double sphere model. It also shows a plot of the aspect ratio of angular resolutions in the radial direction over the tangential direction, which corresponds to the amount of warping. It can be seen that the distortion causes the image to be compressed in the radial direction. Although the amount of warping increases as the FOV increases, the warping for wide FOVs is not substantially greater than that of pinhole models until the outer portions of the

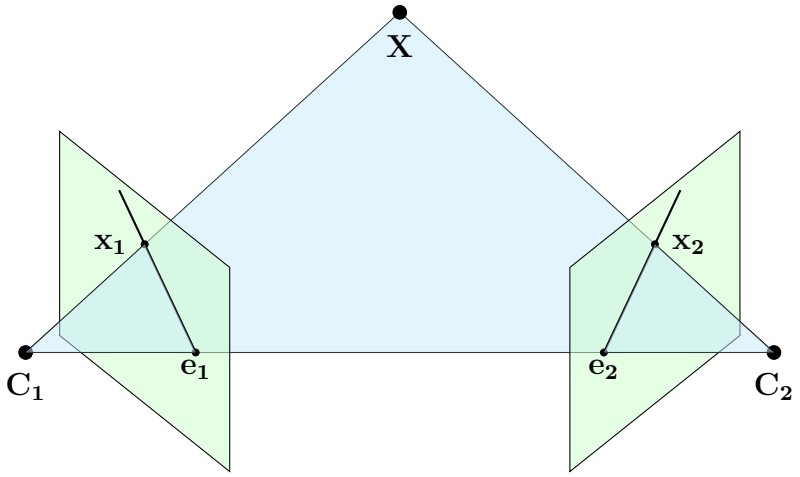


Figure 2-8: Epipolar geometry. $\mathbf{C_1}$ and $\mathbf{C_2}$ are the camera origins, $\mathbf{X}$ is a 3D point projected into both cameras as $\mathbf{x_1}$ and $\mathbf{x_2}$, and $\mathbf{e_1}$ and $\mathbf{e_2}$ are the epipoles. The lines joining $\mathbf{e_1}$ to $\mathbf{x_1}$ and $\mathbf{e_2}$ to $\mathbf{x_2}$ are the epipolar lines, and the plane formed by $\mathbf{C_1}$, $\mathbf{C_2}$, and $\mathbf{X}$ is the epipolar plane.

image. This provides the initial insight that most of the image is still very usable in terms of distortion. It is therefore potentially beneficial to not crop away the usable outer portions of the image, as is inevitably done when rectifying fisheye images to a pinhole model in order to remove radial distortion.

2.3.2 Epipolar Geometry

When two cameras view 3D points from two distinct poses, there are geometric properties that describe how the pixel coordinates in both cameras for those 3D points are related, known as *epipolar geometry*. This is useful both for stereo matching and odometry, as these geometric relations constrain where feature correspondences can be found, given a transformation between the two cameras. This information can be used to reduce the search space for feature correspondence, or perform filtering of incorrect feature correspondences.

Epipolar geometry is well-established for the pinhole camera model. Figure 2-8 depicts two pinhole cameras with known poses viewing a 3D point $\mathbf{X}$. Let $\mathbf{x_1}$ and $\mathbf{x_2}$ be the 3D coordinates of $\mathbf{X}$ projected onto the image plane of both cameras with unit focal length. Suppose $\mathbf{x_1}$ is known (i.e., an arbitrary point detected with a feature

detector), and $\mathbf{X}$ and $\mathbf{x_2}$ are unknown. We know that $\mathbf{X}$ must lie along the ray defined by the line between the camera origin and $\mathbf{x_1}$. If we sweep the potential positions of $\mathbf{X}$ along this ray and project them into the second camera, we obtain a line on the second camera's image plane, known as the *epipolar line*. The true value of $\mathbf{x_2}$ therefore lies on this line.

To express epipolar constraints mathematically, let $\mathbf{C_1}$ and $\mathbf{C_2}$ be the camera origins for camera 1 and 2 respectively. Consider two more 3D coordinates in camera 1 and 2 respectively, $\mathbf{e_1}$ and $\mathbf{e_2}$, defined by the projection of $\mathbf{C_2}$ onto the first camera plane and $\mathbf{C_1}$ onto the second camera plane, respectively. These points are effectively the intersection of the line joining the two camera origins with the image planes. It can be seen that the epipolar line for camera 2 (containing $\mathbf{x_2}$) must pass through $\mathbf{e_2}$, and vice versa.

Alternatively, we can construct a plane that is formed by $\mathbf{C_1}$, $\mathbf{C_2}$, and the ray between $\mathbf{C_1}$ and $\mathbf{x_1}$. Because $\mathbf{X}$ is on this ray, the plane is formed by $\mathbf{C_1}$, $\mathbf{C_2}$, and $\mathbf{X}$. This plane is known as the *epipolar plane*. The intersection of the epipolar plane with the second camera's image plane is the epipolar line for camera 2, and vice versa.

We can define the epipolar plane using a normal vector. To define the plane in the frame of the first camera, let $\mathbf{x_1'}$ be $\mathbf{x_1}$ expressed in the first camera's coordinate frame, in *normalized image coordinates*, obtained by scaling the unprojected ray such that $Z = 1$ so that it is effectively the projection onto an image plane at unit focal length. Note that normalizing image coordinates requires that $Z > 0$, so this applies only to pinhole models. Let $\mathbf{R}$ and $\mathbf{t}$ represent the rotation matrix and translation vector to transform a point from camera 1's frame into camera 2's frame. In the first camera's frame, $\mathbf{C_2}$ can be expressed as $\mathbf{t'} = -\mathbf{R}^T\mathbf{t}$. We can therefore express the epipolar plane normal vector $\mathbf{y_1}$ in the first camera's frame as the cross product between $\mathbf{t'}$ and $\mathbf{x_1'}$:

$$\mathbf{y_1} = [\mathbf{t'}]_\times \mathbf{x_1'} \tag{2.19}$$

where $[\mathbf{t'}]_\times$ is the matrix representation of the cross product for ease of notation. We

now want to express $\mathbf{y_1}$ in camera 2's frame so we can represent the epipolar constraint in the second camera's image. We can move the normal vector $\mathbf{y_1}$ along the plane such that it now originates from the second camera's origin, and then rotate it into the second camera's coordinate frame using $\mathbf{R}$. The epipolar plane normal vector in the second camera's coordinate frame is then defined as:

$$\mathbf{y_2} = \mathbf{R}[\mathbf{t}']_\times \mathbf{x}'_\mathbf{1} \tag{2.20}$$

The epipolar constraint dictates that any $\mathbf{x_2}$ on the second camera's image plane must lie on this epipolar plane. If we express $\mathbf{x_2}$ in camera 2's coordinate frame as $\mathbf{x}'_\mathbf{2}$, then any $\mathbf{x}'_\mathbf{2}$ and $\mathbf{y_2}$ must be perpendicular. The epipolar constraint can then be expressed as:

$$\mathbf{x}'^{T}_\mathbf{2}\mathbf{y_2} = \mathbf{x}'^{T}_\mathbf{2}\mathbf{R}[\mathbf{t}']_\times \mathbf{x}'_\mathbf{1} = \mathbf{x}'^{T}_\mathbf{2}\mathbf{E}\mathbf{x}'_\mathbf{1} = 0 \tag{2.21}$$

The matrix $\mathbf{E} = \mathbf{R}[\mathbf{t}']_\times$ is known as the *essential matrix* [36]. It is a key component of computer vision that relates points from two cameras using epipolar geometry. If the transformation between two camera frames is known, the essential matrix can be calculated, and the epipolar constraints between both images can be used. For example, if two points $\mathbf{x}'_\mathbf{1}$ and $\mathbf{x}'_\mathbf{2}$ are found by a feature correspondence algorithm, the constraint can be used to determine if the correspondence is valid. This is particularly useful for filtering stereo matches where the transformation between the stereo pair is known.

For odometry, the camera poses are not known yet, so epipolar geometry cannot be used immediately. However, just as knowing the frame transformation defines the essential matrix and sets constraints on feature correspondences, the reverse also applies, where a set of proposed feature correspondences can be used to estimate an essential matrix, and subsequently the relative rotation and translation. This both gives an odometry estimate and sets epipolar constraints for the remaining correspondences. The method for this is described in detail in the next section.

Although the above derivation is based on the pinhole model, the concept can be

generalized to any camera model. Although image planes (and therefore normalized image coordinates) can no longer be used, the points $\mathbf{x_1}$ and $\mathbf{x_2}$ do not need to represent points on the image plane; they can simply represent rays from the camera origins (which are found using the camera model) and the rest of the epipolar geometry derivation follows identically. Epipolar planes no longer project onto image planes as epipolar lines, but rather into the nonlinear camera models as epipolar curves. This is the key observation that allows fisheye camera models to be used on the remaining algorithms to be discussed.

2.4 Odometry

Now that feature correspondences between two frames are found, the camera geometry concepts can be used to determine the transformation between the two camera poses, known as the odometry task. Depending on whether a stereo camera is used or if pre-reconstructed points already exist, 2D-2D or 3D-2D approaches can be used.

2.4.1 2D-2D Correspondence

If only 2D feature coordinates are known (i.e., for a monocular system), 2D-2D correspondence methods must be used. The method involves using epipolar geometry to solve for an essential matrix from the point correspondences using a linear system of equations. The essential matrix can then be used to recover the rotation and translation up to scale.

Recall the epipolar constraint, where $\mathbf{x_1}$ and $\mathbf{x_2}$ are normalized image coordinates (assuming pinhole model) in images 1 and 2 respectively, and $\mathbf{E}$ is the essential matrix:

$$\mathbf{x_2}^T\mathbf{E}\mathbf{x_1} = 0 \tag{2.22}$$

$$\mathbf{x_1} = \begin{bmatrix} u_1 & v_1 & 1 \end{bmatrix}^T \tag{2.23}$$

$$\mathbf{x_2} = \begin{bmatrix} u_2 & v_2 & 1 \end{bmatrix}^T \tag{2.24}$$

$$\mathbf{E} = \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} \tag{2.25}$$

This constraint can be rewritten as:

$$\mathbf{e} \cdot \widetilde{\mathbf{x}} = 0 \tag{2.26}$$

$$\mathbf{e} = \begin{bmatrix} e_{11} & e_{12} & e_{13} & e_{21} & e_{22} & e_{23} & e_{31} & e_{32} & e_{33} \end{bmatrix}^T \tag{2.27}$$

$$\widetilde{\mathbf{x}} = \begin{bmatrix} u_2u_1 & u_2v_1 & u_2 & v_2u_1 & v_2v_1 & v_2 & u_1 & v_1 & 1 \end{bmatrix}^T \tag{2.28}$$

To solve for $\mathbf{e}$ uniquely, it can be seen that at least eight feature correspondences ($\widetilde{\mathbf{x}}$) are required. This is known as the *eight-point algorithm* [50]. In 2004, Nistér proposed the *five-point algorithm* [62], which is able to solve for the essential matrix using only five correspondences, by adding additional constraints based on properties of the essential matrix. This algorithm is widely used as the state-of-the-art method for estimating the essential matrix.

Because the algorithm only requires five correspondences and there is typically a much larger number of correspondences, the random sample consensus (RANSAC) algorithm [23] is applied to make the estimation robust to outliers. RANSAC involves randomly sampling a subset from a dataset, fitting a model, and counting the number of inliers that agree with the fitted model. The process is repeated iteratively and the model with the largest number of inliers is returned. In this case, five correspondences are randomly selected, the five-point algorithm is used to calculate an essential matrix, and the number of remaining correspondences that obey the epipolar constraint set by this essential matrix (i.e., $\mathbf{x_2}^T\mathbf{E}\mathbf{x_1}$ is less than a threshold) is used as the number of inliers. The essential matrix that results in the most inliers is used. In addition, if feature tracking is used as the feature correspondence method, this can also be used to detect tracking failures by removing tracks that are outliers.

To recover the rotation matrix $\mathbf{R}$ and translation vector $\mathbf{t}$ from the essential matrix, singular value decomposition is used. Given the singular value decomposition

$\mathbf{E} = \mathbf{U}\mathbf{S}\mathbf{V}^T$, the four solutions for $\mathbf{R}$ and $\mathbf{t}'$ are:

$$\mathbf{R} = \mathbf{U}(\pm\mathbf{W}^T)\mathbf{V}^T \tag{2.29}$$

$$\mathbf{t}' = \mathbf{U}(\pm\mathbf{W}^T)\mathbf{S}\mathbf{U}^T \tag{2.30}$$

$$\mathbf{W}^T = \begin{bmatrix} 0 & \pm 1 & 0 \\ \mp 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.31}$$

As there are four potential solutions, an additional *chirality* [37] check is necessary. The check involves triangulating correspondences into a 3D point (see section 2.5 for details) using the proposed camera poses, and checking if that point is visible by both cameras. For a pinhole model, the check is simply if the z coordinate in the coordinate frame of each camera is positive. For other camera models, the model itself must be used to see if the reprojection is valid. The solution with the most valid correspondences is used as the pose estimate.

The recovered pose is only correct up to scale, as there is no way to determine metric scale from 2D information only (i.e., objects can be large and far or small and close and appear the same in the image). That is, $\mathbf{R}$ will be correct, but $\mathbf{t}$ will be a unit vector in the direction of the motion. To determine the scale, information must be known about the depths of the points, which is one of the challenges of monocular SLAM systems. To perform the odometry task with the correct scale, an IMU can be added, which is outside the scope of this thesis, or a stereo camera can be used by adding second camera with a known offset.

2.4.2 3D-2D Correspondence

If 3D information is known about the points in the first frame, either through stereo depth estimation or reconstruction from previous frames, 3D-2D correspondence methods can be used to determine the odometry for the second frame using its 2D pixel coordinates. The method involves finding the best camera pose such that the error between the 3D points reprojected into the image and their corresponding pixel coor-

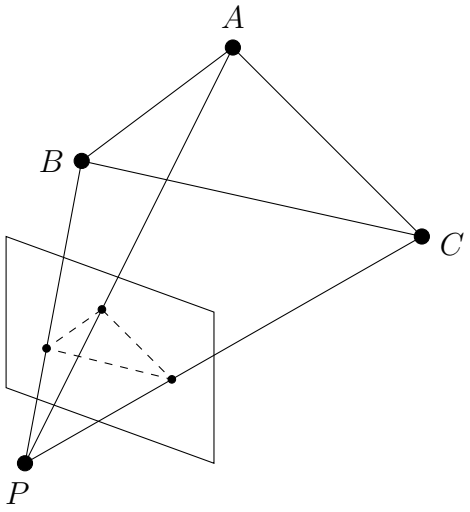


Figure 2-9: P3P Problem Formulation

dinates is minimized. This is known as the *perspective-n-point* (PnP) problem. The problem can be formulated as follows, where $\mathbf{X}_i$ is a 3D point, $\mathbf{x}_i$ is its corresponding feature pixel coordinate in the frame that observes it, $\mathbf{T}$ is the pose (transformation matrix) of the frame to be solved for, and $\mathbf{C}(\mathbf{X})$ is the camera model:

$$\mathbf{T}^* = \arg\min_{\mathbf{T}} \sum_{i}^{n} \|\mathbf{x}_i - \mathbf{C}(\mathbf{T}\mathbf{X}_i)\|^2 \tag{2.32}$$

The PnP problem is another classic computer vision problem with applications in many other fields as well. In particular, the P3P problem, the minimal form requiring three correspondences, has a long history with many approaches. The first solution was derived in 1841 by Grunert [30] for the application of photogrammetry, which formulated the solution geometrically as follows, as shown in figure 2-9. Let P be the origin of the camera, and A, B, and C be the 3D world points, which forms three triangles PAB, PBC, and PAC. Using the law of cosines:

$$|PA|^2 + |PB|^2 - 2|PA||PB|\cos\angle APB = |AB|^2 \tag{2.33}$$

$$|PB|^2 + |PC|^2 - 2|PB||PC|\cos\angle BPC = |BC|^2 \tag{2.34}$$

$$|PA|^2 + |PC|^2 - 2|PA||PC|\cos\angle APC = |AC|^2 \tag{2.35}$$

The three angles are known from the corresponding normalized image coordinates (assuming pinhole), and the system of equations can be solved. The solution leads to four feasible solutions for the camera pose P [22], so a fourth point can be used to disambiguate the solution by choosing the pose that leads to the lowest reprojection error for that point.

Several advancements have been made to the PnP problem in the last 100 years [22, 23, 28, 66, 25, 44, 40], all of which seek to improve computational speed and numerical stability. A summary was presented by Haralick [34] in 1991. In particular, in 1981, Fischler and Bolles [23] extended the P3P algorithm to an arbitrary number of correspondences by introducing the RANSAC algorithm, which has since then been the standard approach for PnP. The approach consists of running RANSAC on top of a P3P algorithm (with a fourth point to disambiguate), finding the inliers for the best solution, and finally running an optional non-linear optimization over the inliers to refine the solution. Other PnP approaches that support arbitrary values of n include Quan and Lan's method [66] and Lepetit's EPnP [44].

Up until recently, the state-of-the-art algorithm has been Gao's approach to P3P [25], who derived an analytical algebraic solution as well as a complete solution classification. It has been widely adapted due to its robustness and is currently the implementation used in OpenCV [13]. Recently, the Lambda Twist [65] P3P algorithm was introduced, which achieved state-of-the-art performance with an increase in speed and numerical stability by several orders of magnitude compared to previous methods. The Lambda Twist algorithm combined with RANSAC will be the PnP algorithm used in this thesis.

All of the existing PnP implementations take normalized image coordinates as 2D inputs, and therefore rely on the use of the pinhole model. This thesis will generalize the algorithms to support all camera models by observing that only rays are required, not image coordinates, so any unprojection function can be used.

To use the 3D-2D correspondence method, 3D points are required. They can be obtained through stereo matching or reconstruction using poses from previous frames. In the monocular case, reconstructed 3D points from previous frames are

used. For the first two frames 2D-2D correspondences are used to obtain the initial pose, then reconstruction and 3D-2D odometry are used in alternating fashion for subsequent frames. The next section will discuss algorithms used for both stereo and frame-to-frame reconstruction.

2.5 Reconstruction

The process of estimating the structure of a scene, both for building a map and for pose estimation relative to that map (i.e., PnP), as well as stereo depth estimation, is known as reconstruction or depth estimation. The task involves taking a set of 2D images that all observe a 3D scene from different known poses, and estimating the 3D structure of that scene. There are two main types of reconstruction, *sparse* and *dense* reconstruction. Sparse reconstruction involves estimating the 3D coordinates of individual sparse features independently and building a 3D point cloud of those features, and will be the type of reconstruction used in this thesis. Dense reconstruction involves computing the full surface geometry of a scene using voxels or meshes, and is outside the scope of this thesis.

To estimate the depth of a point, it must be observed by at least two frames with known poses. The 3D coordinate of the point can then be *triangulated* by unprojecting the 2D observations into rays and finding the intersection of those rays. In the case of frame-to-frame reconstruction, the past odometry estimates are used as the poses. The features are then reconstructed and used for the odometry estimate for the next frame, and so on. For stereo reconstruction, the pose transformation between the two cameras are known from calibration.

Ideally, the unprojected rays would intersect. However, because of noise, feature correspondence uncertainty, and imperfect pose estimation and camera model calibration, they rarely intersect perfectly. The simplest, most common method to obtain a close result is using the *direct linear transformation* (DLT) [1] algorithm which is solved linearly. Let $\mathbf{x} = \begin{bmatrix} u & v & 1 \end{bmatrix}^T$ be the observed image point in normalized image coordinates (assuming pinhole) on one of the images, $\mathbf{T}$ be the pose matrix of the

camera that transforms 3D points in world frame to the camera's frame, and $\mathbf{X}$ be the homogeneous 3D coordinates of the point to triangulate. Because $\mathbf{x}$ represents the point on the image plane with a focal length of 1, it is effectively a ray with a z-coordinate of 1, so the following holds, where w is an unknown scale factor and $\mathbf{t}_i^T$ is the ith row of $\mathbf{T}$:

$$\mathbf{TX} = w\mathbf{x} = w \begin{bmatrix} u & v & 1 \end{bmatrix}^T \tag{2.36}$$

$$\mathbf{t}_1^T\mathbf{X} = wu \tag{2.37}$$

$$\mathbf{t}_2^T\mathbf{X} = wv \tag{2.38}$$

$$\mathbf{t}_3^T\mathbf{X} = w \tag{2.39}$$

The following can then be derived by rearranging:

$$(u\mathbf{t}_3^T - \mathbf{t}_1^T)\mathbf{X} = 0 \tag{2.40}$$

$$(v\mathbf{t}_3^T - \mathbf{t}_2^T)\mathbf{X} = 0 \tag{2.41}$$

These linear equations can then be stacked for every image that observes $\mathbf{X}$ and written in matrix form to obtain an equation of the form $\mathbf{AX} = \mathbf{0}$, where $\mathbf{A}$ is a $2n \times 4$ matrix for n images. Because the rays will not intersect the triangulated point perfectly, the equation will not be satisfied perfectly, so a least-squares solution that minimizes $\|\mathbf{AX}\|$ can be obtained with methods like singular value decomposition.

This method is simple and fast to compute for small numbers of observing frames. In particular, for two images in the case of stereo triangulation or chirality checks for essential matrix estimation, the equation only needs to be solved for a 4×4 matrix so it is very efficient. However, for larger numbers of frames, in the case of triangulating a point from many observing frames in a sequence, the matrix can get very large. Another solution is to take the midpoint of the rays, which minimizes the distance of the point from the rays in a least-squares sense. Again, let $\mathbf{X}$ be the point to be triangulated in homogeneous coordinates, $\mathbf{T}_i$ be the pose matrix of the

ith observing camera, and $\hat{\mathbf{x}}$ be the unit ray of the normalized image coordinate $\mathbf{x}$ (or unprojected ray for non-pinhole models). The midpoint triangulation problem can then be formulated as follows:

$$\mathbf{X}^* = \arg\min_{\mathbf{X}} \sum_{i}^{n} \|\mathbf{T}_i\mathbf{X} - \hat{\mathbf{x}}\hat{\mathbf{x}}^T\mathbf{T}_i\mathbf{X}\|^2 \tag{2.42}$$

Observe that the term that is minimized is simply the distance of the triangulated point to its projection onto the observing rays, which gives the midpoint. If we now let $\mathbf{A}_i = \mathbf{T}_i - \hat{\mathbf{x}}\hat{\mathbf{x}}^T\mathbf{T}_i$ and $\mathbf{A} = \sum_i^n \mathbf{A}_i^T\mathbf{A}_i$, the problem can be rewritten as follows:

$$\mathbf{X}^* = \arg\min_{\mathbf{X}} \sum_{i}^{n} (\mathbf{A}_i\mathbf{X})^T(\mathbf{A}_i\mathbf{X}) = \arg\min_{\mathbf{X}} \mathbf{X}^T\mathbf{A}\mathbf{X} \tag{2.43}$$

The value of $\mathbf{X}^*$ can then be obtained by computing the eigenvector for the smallest eigenvalue of $\mathbf{A}$. Because $\mathbf{A}$ is now a sum of matrices, it is always a 4×4 matrix and can be solved efficiently regardless of the number of observing frames. In addition, because it only requires unit rays unprojected from the 2D observations, any camera model can be used.

In the case of frame-to-frame reconstruction, the poses need to have sufficient translation relative to the distance to the reconstructed points. That is, the observing rays need to view the points from angles that are sufficiently different, otherwise the uncertainty in depth estimation will be too large. Therefore, it is necessary to have a threshold on the angular coverage of the rays before a point can be reconstructed. This is the same problem as depth uncertainty for stereo cameras, with points at large distances compared to the stereo baseline (distance between stereo pair) having large uncertainty.

Although the methods described earlier work well, using the midpoint of the rays as the solution does not account for the distances of the cameras to the point; closer cameras have more certainty about the location of a point so should be weighted more. This relationship is captured by minimizing the reprojection error of the reconstructed point instead. However, this is a non-linear optimization problem which is not easily solved. Instead of performing this optimization at every reconstruction step, the

methods described earlier can be used as an initial estimate for the reconstruction, which can then be refined later. Minimizing the reprojection error is part of the bundle adjustment process, which is discussed in the next section.

2.6 Global Optimization

The algorithms discussed so far are core components of the visual odometry task, which is only concerned with the incremental frame-to-frame trajectory. As such, small errors in the odometry estimation can accumulate over distance and result in large drift. If the camera returns to a previously observed scene, the drift can be corrected, a task known as loop closure. To do this, there need to be mechanisms in place to fix the entire trajectory such that it is globally consistent with the map. The task of maintaining global consistency is the differentiating factor between visual odometry and SLAM [72].

2.6.1 Bundle Adjustment

One way of maintaining global consistency between the map and trajectory is to perform bundle adjustment. If a list of frame-landmark observations is maintained, obtained during both odometry and loop closure, a large optimization problem can be formulated that minimizes the total reprojection error of all the landmarks into all the frames they are observed in, by jointly optimizing over both the frame poses and the landmark coordinates. Assuming that there are n frames that observe m landmarks, let $\mathbf{x}_{ij}$ be the observed pixel coordinate of landmark j observed by frame i, $\mathbf{T}_i$ be the pose matrix of frame i, $\mathbf{X}_j$ be the 3D coordinate of the jth landmark, $\mathbf{C}(\mathbf{X})$ be the camera model, and $v(i,j)$ be the binary indicator variables that are 1 if frame i observes landmark j and 0 otherwise:

$$\underset{\mathbf{T}_i \forall i, \mathbf{X}_j \forall j}{\arg\min} \sum_{i}^{n} \sum_{j}^{m} v(i,j) \|\mathbf{C}(\mathbf{T}_i \mathbf{X}_j) - \mathbf{x}_{ij}\|^2 \tag{2.44}$$

The optimization can be implemented easily using non-linear solvers like Ceres

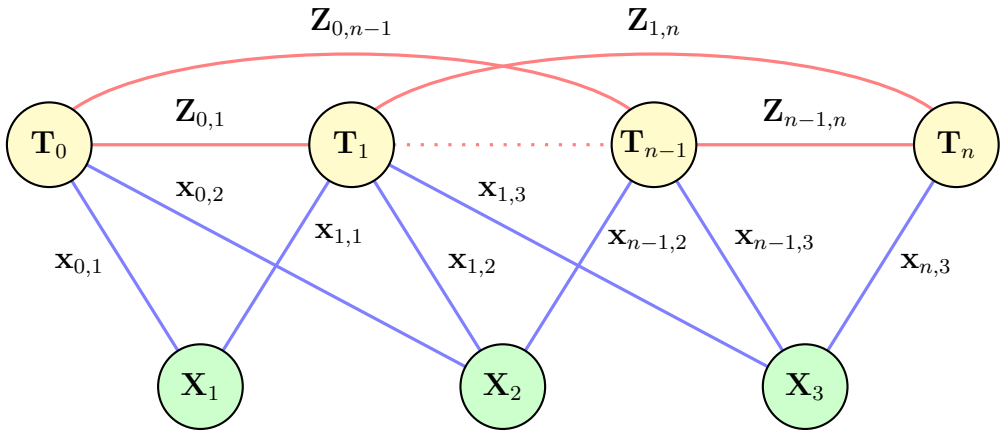


Figure 2-10: SLAM graph formulation, showing the edges used in bundle adjustment (blue) and pose-graph optimization (red). Probabilistic smoothing-based approaches represent the graph as a factor graph and use all edges. The nodes are camera poses $\mathbf{T}_i$ and landmark coordinates $\mathbf{X}_k$, and the edges are landmark observations $\mathbf{x}_{i,k}$ of landmark k by frame i, and odometry measurements $\mathbf{Z}_{i,j}$ between frames i and j.

[2]. Care must be taken to avoid adding outliers into the optimization, although loss functions can be added to mitigate the effects of outliers. The bundle adjustment problem can also be formulated as a graph, where each node is a frame pose or a landmark coordinate, and each edge between a pose and landmark is an observation and observation model (projection function in this case), as shown in figure 2-10. Note how loop closure constraints are formulated by having observation edges between later frames and earlier landmarks. The task is then to assign values to each node such that the costs of the edges (errors between observations and observation model outputs) is minimized. The graph optimization problem can be implemented using frameworks like g2o [43].

Because there are usually a large number of landmarks, the number of frame-landmark observations can be very large, making the optimization problem slow to solve. Bundle adjustment is thus usually only performed at the end of a trajectory and not useful in real-time. *Sliding-window* bundle adjustment sacrifices global consistency for real-time usability by optimizing the trajectory only over the last few frames, obtaining local consistency. Another way to obtain global trajectory-only consistency (sufficient for loop closure) is by removing the map from the optimization entirely and only optimizing over poses, known as pose-graph optimization.

2.6.2 Pose-graph Optimization

Pose-graph optimization is another graph optimization problem that optimizes a trajectory given odometry constraints between frame poses. The inter-frame pose transformations obtained from the odometry and loop closure steps are used as observations to create a *pose-graph*, where each node is again a frame pose and each edge between poses is an odometry estimate (incremental pose transformation), as shown in figure 2-10. Note that loop closure constraints are now directly represented as pose transformations as well. The observation model in this case is the relative transformation between the two poses; specifically, for an edge connecting two world-frame poses i and j with rotation matrices and translation vectors $\mathbf{R}_i$, $\mathbf{t}_i$, $\mathbf{R}_j$, and $\mathbf{t}_j$:

$$\mathbf{R}_{ij} = \mathbf{R}_j \mathbf{R}_i^T \tag{2.45}$$

$$\mathbf{t}_{ij} = \mathbf{t}_j - \mathbf{R}_{ij} \mathbf{t}_i \tag{2.46}$$

The difference between this transformation (denoted as $\mathbf{T}_{ij}$) and the observed odometry pose transformation (denoted as $\mathbf{Z}_{ij}$) is used as the error to be minimized. The error can be represented as:

$$\mathbf{e}_{ij} = \mathbf{Z}_{ji} \mathbf{T}_{ij} \tag{2.47}$$

where $\mathbf{Z}_{ji}$ is the inverse pose matrix of $\mathbf{Z}_{ij}$. Note that $\mathbf{e}_{ij}$ becomes the identity pose matrix (identity rotation matrix and zero translation vector) when the error is zero. Again, graph optimization packages can be used to solve for the global trajectory that minimizes the total error from odometry and loop closure estimates. Note that without loop closure to force correction, the solution will be the original trajectory as it will have an error of zero.

The optimization formulations so far have all been non-probabilistic, by assuming that all observations are perfect. Much better performance can be achieved by modeling probabilistic distributions of measurement uncertainties and integrating them into the graph optimizations. For example, in practice, pose-graph optimization is a

probabilistic approach, where each odometry estimate is modeled as a distribution. The optimization will then place more weight on the more certain measurements to give a maximum likelihood estimate of the global trajectory. This probabilistic approach to global optimization can be extended to optimizing over both the poses and map as well. Compared to bundle adjustment, formulating this graph problem probabilistically allows much faster, real-time performance. Although graph-based SLAM is outside of the scope of this thesis, the next section will give a brief introduction of this modern approach to SLAM for completeness.

2.6.3 Smoothing-based SLAM

If we reconsider the graphical model of SLAM in figure 2-10 probabilistically and use all edges, each observation can now be modeled as a conditional probability distribution. In particular, landmark observations by frames can be modeled with the likelihood function $P(\mathbf{x}_{ij}|\mathbf{T}_i, \mathbf{X}_j)$, and odometry estimates can be modeled with $P(\mathbf{T}_i|\mathbf{T}_{i-1}, \mathbf{Z}_{i-1,i})$. The posterior distribution of the full trajectory $\mathbf{T}$ of n frames and all m landmark coordinates $\mathbf{X}$ given landmark observations $\mathbf{x}$ and odometry $\mathbf{Z}$ can then be written as:

$$P(\mathbf{T}, \mathbf{X}|\mathbf{x}, \mathbf{Z}) \propto P(\mathbf{T}_0) \prod_{i}^{n} P(\mathbf{T}_i|\mathbf{T}_{i-1}, \mathbf{Z}_{i-1,i}) \prod_{j}^{m} P(\mathbf{x}_{ij}|\mathbf{T}_i, \mathbf{X}_j) \tag{2.48}$$

Maximizing this value then obtains the maximum likelihood trajectory and map. To solve this optimization efficiently, the graphical model can be represented as a *factor graph* [42] to model the factorization of the posterior distribution. Each observation edge becomes a factor in the factor graph which encodes the conditional distribution, and each node is a variable. The methods used to solve this factor graph formulation of SLAM efficiently are known as *smoothing*-based approaches. This is a common approach to modern SLAM and can be solved by packages like iSAM [38] and GTSAM [19].

Chapter 3

Related Work

Although not nearly as commonly used as perspective cameras, the benefits of omnidirectional cameras have motivated a number of results that use omnidirectional cameras (both fisheye and catadioptric) for navigation tasks. This section will discuss some of the related work in these areas.

Starting from the first step of the SLAM pipeline, we start with related work in the area of feature correspondence on omnidirectional images. Although several existing works claim that the KLT tracking algorithm works sufficiently well for navigation tasks without modification on omnidirectional images [77], a number of other works have presented improvements to KLT for the omnidirectional tracking task. Wang et al. [89] evaluates KLT on fisheye images for the purpose of object tracking. They claim that KLT cannot track features for any reasonable length of time because of warping, and presents improvements to compensate for short tracks for object tracking. Several works [51, 5, 67] attempt to improve on the inaccuracies introduced by the radial distortion by modifying the LK algorithm directly, by introducing additional parameters to the warping function to model the radial distortion. Lourenço et al. [51] additionally evaluates their modification, showing a decrease in pixel tracking error by half, as well as fewer tracking outliers.

In the area of descriptor-based matching, various descriptors have also been presented attempting to be invariant to the warping present in wide-angle images. SPHORB [92] and BRISKS [32] are two descriptors based on ORB and BRISK re-

spectively that operate on spherical equirectangular images, by modeling the image as a sphere and sampling in a geodesic pattern. Several works, including sSIFT and pSIFT [33], sRD-SIFT [52], Tri-SIFT [88], and OmniSIFT [6] present modifications to SIFT to improve its performance on radially distorted images. Binary descriptors based on BRIEF, such as dBRIEF, mdBRIEF [86], and TPBRIEF [93], have also been presented. In particular, dBRIEF and mdBRIEF are used by MultiCol-SLAM [84], a SLAM system designed for fisheye cameras. A number of these descriptors handle the fisheye distortion by using the camera model or distortion model to sample pixels at rectified coordinates to compute the descriptor, effectively undistorting the image. As such, they are only able to operate on cameras with less than 180 degrees of FOV. Three works [48, 91, 54] have also shown good results using the center-symmetric local binary patterns (CSLBP) operator as a descriptor to perform matching on fisheye images. Analysis papers that evaluate existing descriptors have also been presented. In particular, Benseddik et al. [10] evaluates the BRISK descriptor on catadioptric images and claims it performs well. Kropp [41] evaluates several of the descriptors based on SIFT for matching on omnidirectional images, and concludes that the performance improvements are marginal.

Continuing on to full visual odometry and SLAM, several systems have been presented that use omnidirectional cameras. Early approaches [17, 80, 73] use catadioptric or multi-camera systems to perform omnidirectional monocular visual odometry. Corke et al. [17] and Scaramuzza et al. [73] use catadioptric cameras modeled by the unified camera model. Corke et al. [17] uses optical flow for feature correspondence, while Scaramuzza et al. [73] uses SIFT features. Tardif et al. [80] uses a multi-camera system for an omnidirectional view, and also uses SIFT features due to KLT not performing adequately at the frame rate used. More recently, full SLAM and visual odometry systems have been developed that support omnidirectional fisheye cameras, using the full image directly and not cropping or rectifying. MultiCol-SLAM [84] is a feature-based multi-camera SLAM system that supports fisheye cameras, by adding support for non-pinhole camera models and using dBRIEF and mdBRIEF descriptors. OpenVSLAM [79] is another feature-based SLAM system that supports

many different camera models including fisheye, and uses ORB descriptors. Liu et al. [49] extends ORB-SLAM [61] to support omnidirectional fisheye cameras by modifying traditional algorithms like PnP to adapt the extended unified camera model [39], an extension of the unified camera model to better model fisheye lenses. The authors claim that the use of fisheye cameras improves accuracy and robustness over normal ORB-SLAM. In addition to feature-based SLAM, direct methods have also been adapted to support omnidirectional cameras. Large-scale Direct SLAM (LSD-SLAM) [21] has been extended to support using the full image from omnidirectional cameras [16] by reformulating the direct image alignment algorithm using the unified camera model. Similarly, Omnidirectional Direct Sparse Odometry (OmniDSO) [58] extends DSO [20] by also adapting the unified camera model. Both of the omnidirectional direct systems have been compared to systems that crop or rectify the images, and have shown improvements in accuracy and robustness.

To evaluate any system on omnidirectional fisheye images, it is critical to have datasets containing image sequences captured from fisheye camera systems. Because of the lack of mainstream use of fisheye cameras, standard benchmarking datasets like KITTI [26] and EuRoC [14] only have image sequences from perspective cameras, so alternative datasets containing fisheye images must be used, which are a lot less common. Published datasets containing images captured using real fisheye cameras for the purpose of navigation include the TUM omnidirectional dataset [75], the LaFiDa dataset [85], and the Oxford RobotCar dataset [56]. In particular, the TUM dataset uses a fisheye stereo pair, useful for evaluating stereo odometry systems. Synthetic datasets generated in simulation have also been published, in particular a multi-FOV dataset [94] containing images from both perspective and fisheye cameras on the same trajectory, useful for comparing performance between the two types of cameras, and another omnidirectional multi-camera dataset [90].

Finally, other analysis papers that compare omnidirectional and traditional cameras for the SLAM task have also been published, which are the most relevant related works to this thesis. Zhang et al. [94] implements a semi-direct visual odometry pipeline to compare performance between wide and narrow FOV cameras, including

catadioptric cameras. Experiments are conducted to compare performance between camera types on KLT feature tracking, pose optimization against a fixed map, and a full visual odometry pipeline. For odometry, translation and rotation errors with respect to lens FOV are shown for different navigation environments. The authors conclude that omnidirectional cameras are preferable for indoor environments, while for urban canyon environments (streets flanked by buildings), narrow FOV cameras perform better. They also state that the tradeoff comes from sacrificing angular resolution for wider angular coverage of features. Rituerto et al. [68] implements a monocular SLAM system using the Extended Kalman Filter (EKF) and SIFT feature matching to compare traditional cameras with catadioptric cameras. Because of lack of scale, only orientation errors were compared between different cameras. It was observed that the accuracy performance of omnidirectional cameras was much better, although the feature track lifetimes were not significantly different and did not have an impact on performance. The reconstructed map was also compared between cameras, and also proved to be more accurate for omnidirectional cameras. The authors thus conclude that omnidirectional cameras are better suited for the SLAM task. It is important to note that both analyses used the unified camera model. As the double sphere model is relatively new, its performance has not been analyzed in depth, which is one of the main contributions of this thesis.

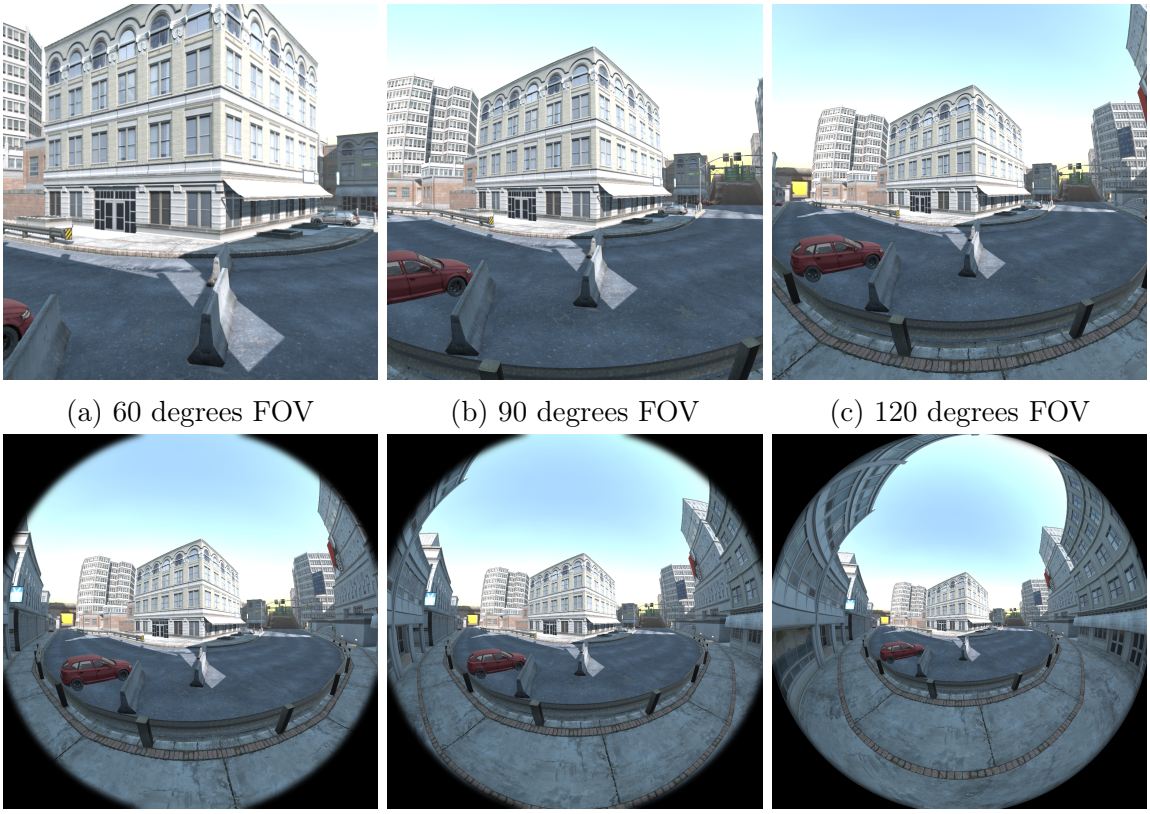
Chapter 4

Feature Tracking Evaluation

The first part of the SLAM pipeline that will be evaluated is the feature correspondence task. As the feature correspondences will be used by the rest of the SLAM pipeline, its performance directly affects the performance of the navigation task. In this section, the Kanade-Lucas-Tomasi (KLT) feature tracking algorithm will be evaluated on both perspective and omnidirectional fisheye cameras. Several performance metrics will be analyzed, including tracking accuracy and track lifetime, and compared over different fields of view and camera motion types.

4.1 Evaluation Method

In order to analyze accuracy, the ground truth locations of the original feature detections must be known. To obtain accurate ground truth, evaluation datasets are generated from a simulated urban environment in Unity. The camera is simulated using the double sphere model, using calibration parameters from real lenses. Resolution is fixed at 1024×1024 for all FOVs. Ground truth camera poses and depth maps with the same camera model are also included in the datasets. In addition to FOV, datasets are categorized by motion type, such as yawing, translation, and composite motions. Yawing and pitching motions are grouped together as the image changes the same way, just along a different axis. Translation along the optical axis is separated into forward and backwards translation, as one causes features to move



(a) 60 degrees FOV (b) 90 degrees FOV (c) 120 degrees FOV

(d) 160 degrees FOV (e) 195 degrees FOV (f) 250 degrees FOV

Figure 4-1: Images of various FOVs from the simulated dataset

outwards and vice versa, so the behavior of algorithms may be different for those two motions. Sample images from the datasets are shown in figure 4-1.

Feature detection and tracking algorithms use implementations from OpenCV [13]. The Shi-Tomasi corner detector [76] is used to detect good features for tracking. To ensure uniform feature distribution throughout the image, detection is performed independently in separate regions. Because fisheye images are circular, the regions are divided in the polar coordinate system, into five radial rings with eight angular regions each, as shown in figure 4-2. Each newly detected feature is unprojected using the camera model into a ray. The depth of the ray is found using the depth map, and the ground truth 3D coordinate of the feature is calculated using the ground truth camera pose.

Because optical flows relies on the brightness constancy assumption and is not intended to handle lighting changes, the images are first histogram equalized to com-

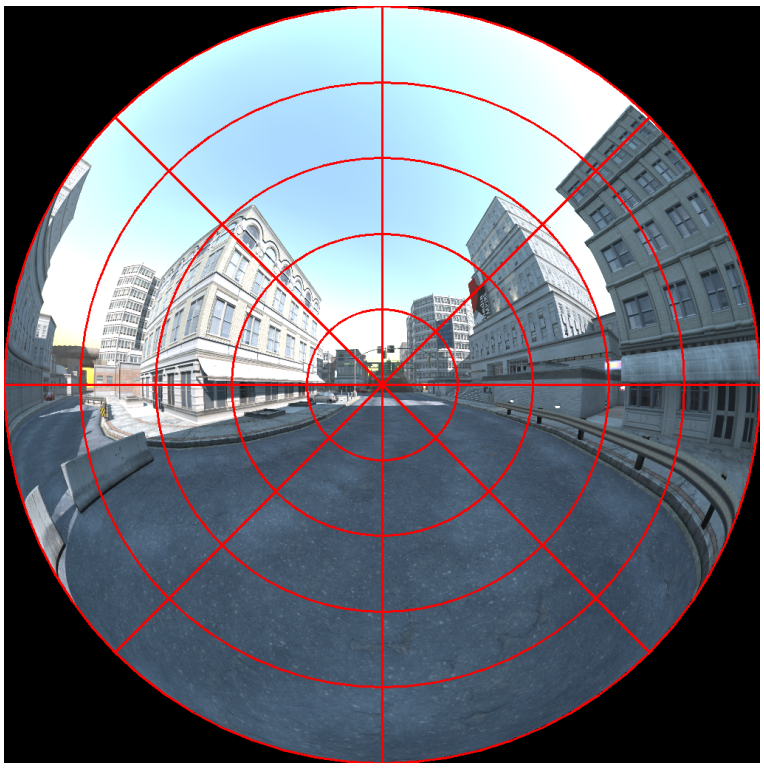


Figure 4-2: Example omnidirectional fisheye image with polar feature regions overlaid

pensate for sudden exposure changes. Tracking is then performed incrementally, with each frame tracking off of the previous frame, using an 8×8 window size and 4 pyramid levels. Failed tracks are detected in a few ways. First, the KLT tracker implementation itself returns failures for features for which it could not find optical flows. The algorithm also returns a similarity error between the original patch and the tracked patch, as the normalized L1 distance (average intensity difference over the patch size). Errors above a fixed threshold indicate that tracks may have jumped to different features so are treated as track failures. In addition, features that are tracked into regions of the image that do not have a valid camera model unprojection (i.e., outside the image circle) are also failures. Finally, the five-point RANSAC algorithm is run on all tracks to fit an essential matrix, and outliers from this process are treated as tracking failures. This is the primary, most robust method for detecting failures; more details about this process are provided in chapter 6. Failed tracks for the current frame are recorded and the tracks are aborted. For the remaining tracks, the ground truth feature locations are obtained by reprojecting the 3D coordinates

into the camera model. The ground truths are used to calculate various error metrics which are recorded for each track at each frame.

As the camera moves, tracks will inevitably fail or move outside the image. To maintain the distribution of tracks throughout the image, remaining tracks in each region are counted after each frame is tracked. Feature detection is performed in regions with track counts that fall below a threshold to create new tracks. This allows longer trajectories, where the initial features detected from the first frame eventually leave the image, to be analyzed.

4.2 Evaluation Metrics

Metrics from optical flow evaluation [8, 24, 74, 7] will be used to evaluate the feature tracking task. A simple metric is the L2 pixel coordinate error between the tracked point $\mathbf{p}$ and the ground truth $\mathbf{g}$, also known as *endpoint error*:

$$e_E = \|\mathbf{p} - \mathbf{g}\| \tag{4.1}$$

This metric has been argued to be the preferred evaluation metric for optical flow [7]. Another common metric for optical flow is the *angular error*, which gives a relative measure of error for the flow between two frames by only considering the error in the flow direction vector. Let the flow vector $\mathbf{v}$ between two pixel coordinates $(u_1, v_1)^T$ and $(u_2, v_2)^T$ be represented as $\mathbf{v} = (u_2 - u_1, v_2 - v_1, 1)^T$, where the third component is present to avoid division by zero problems for zero flows. The angular error is defined as follows, where $\hat{\mathbf{v}}_\mathbf{p}$ and $\hat{\mathbf{v}}_\mathbf{g}$ are the unit flow vectors for the tracked point and ground truth respectively:

$$e_A = \cos^{-1}(\hat{\mathbf{v}}_\mathbf{p} \cdot \hat{\mathbf{v}}_\mathbf{g}) \tag{4.2}$$

The error metrics introduced so far have been purely in pixel space, which, for pinhole models, are roughly equivalent to errors in unprojected ray angles in 3D space, due to a mostly constant angular resolution per pixel. For other camera models,

however, the angular resolution per pixel is less constant as a function of distance from the center of the image. Therefore, a given pixel space error in the outer edge of a fisheye image can have a very different error in unprojected ray angle than the same error in the center of the image. As the errors in unprojected ray angles are the driving factor in the accuracy of the downstream SLAM pipeline, this error will also be used as a metric to normalize for the effects of uneven angular resolution in pixel space. We will call this error *bearing error* and define it as follows, where $\hat{\mathbf{C}}^{-\mathbf{1}}(\mathbf{x})$ is the normalized camera model unprojection function:

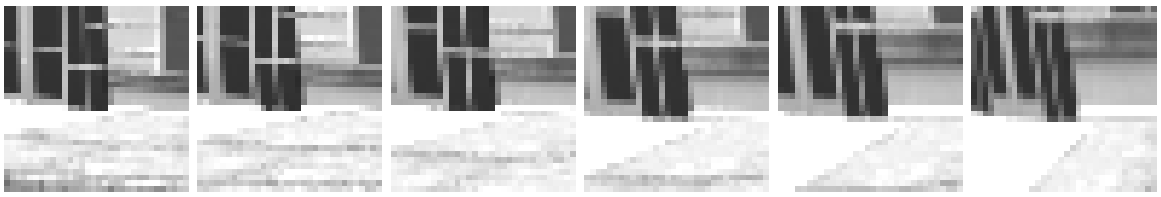
$$e_B = \cos^{-1}(\hat{\mathbf{C}}^{-\mathbf{1}}(\mathbf{p}) \cdot \hat{\mathbf{C}}^{-\mathbf{1}}(\mathbf{g})) \tag{4.3}$$

A final metric is specifically for the feature tracking task combined with essential matrix estimation to determine good tracks. The inlier ratio, which is the number of inliers from the RANSAC estimation over the total number of tracks for the frame, gives a measure of failure rate, which is useful as a higher level metric for evaluating tracking performance.

4.3 Evaluation Results

From looking at fisheye images, it is evident that the amount of warping in the image increases as a function of radial distance from the center. Because warping is the primary factor that would cause pixel space algorithms to perform differently on fisheye cameras than traditional cameras, analysis will be focused on performance as a function of radial distance, which correlates with the amount of fisheye distortion. The radial distances used are normalized with respect to the image size, with 0.5 being the horizontal or vertical distance to the edge of the image (assuming a square image).

The endpoint and angular error metrics will first be analyzed as a function of radial distance. As the errors are recorded every frame for every track, and tracks tend to accumulate error over many frames, using the absolute endpoint error for all frames will introduce a dependency on track length. To remove this dependency, we



(a) $r = 0.1$ (b) $r = 0.2$ (c) $r = 0.3$ (d) $r = 0.4$ (e) $r = 0.45$ (f) $r = 0.48$

Figure 4-3: Example feature patch at different radial distances in yaw motion for 250 degree FOV camera

instead consider the change in endpoint error between consecutive frames for features at different radial distances, which better reflects how the distortion in different radial regions affects KLT's incremental tracking accuracy, and also shows which portions of the image tend to contribute most to the track errors throughout their lifetimes. Angular error, on the other hand, is already inherently an incremental metric as it is calculated from flow vectors between consecutive frames.

Figures 4-4 and 4-5 show distributions of relative endpoint error and angular error, respectively, over different radial distance ranges for different FOV cameras and motion types. For most motion types, it can be seen that the errors are generally similar across FOVs, except for the slight increases in error for fisheye cameras at the outermost radial distances where distortion is most extreme. This can be explained by visualizing how the local distortion changes across the image. Figure 4-3 shows an example feature patch as it moves outwards in a yaw motion. The patch does not distort much locally, with only slight stretching in one direction, until the outermost edge where the change is more drastic, which explains the results. In addition to the distortion changing the appearance of features too drastically for accurate tracking, feature patches also undergo extreme stretching near the outer edges due to the difference in angular resolution for fisheye lenses. Any tracking errors will therefore be amplified as the ground truth point gets stretched away from the tracked point. A notable exception is for pure roll motion where the camera rotates about its optical axis. The features in this case stay at the same radial distances and do not move radially, so no change in distortion is experienced. Thus, the errors are mostly constant throughout the image, with drift being the only source of error. Other exceptions

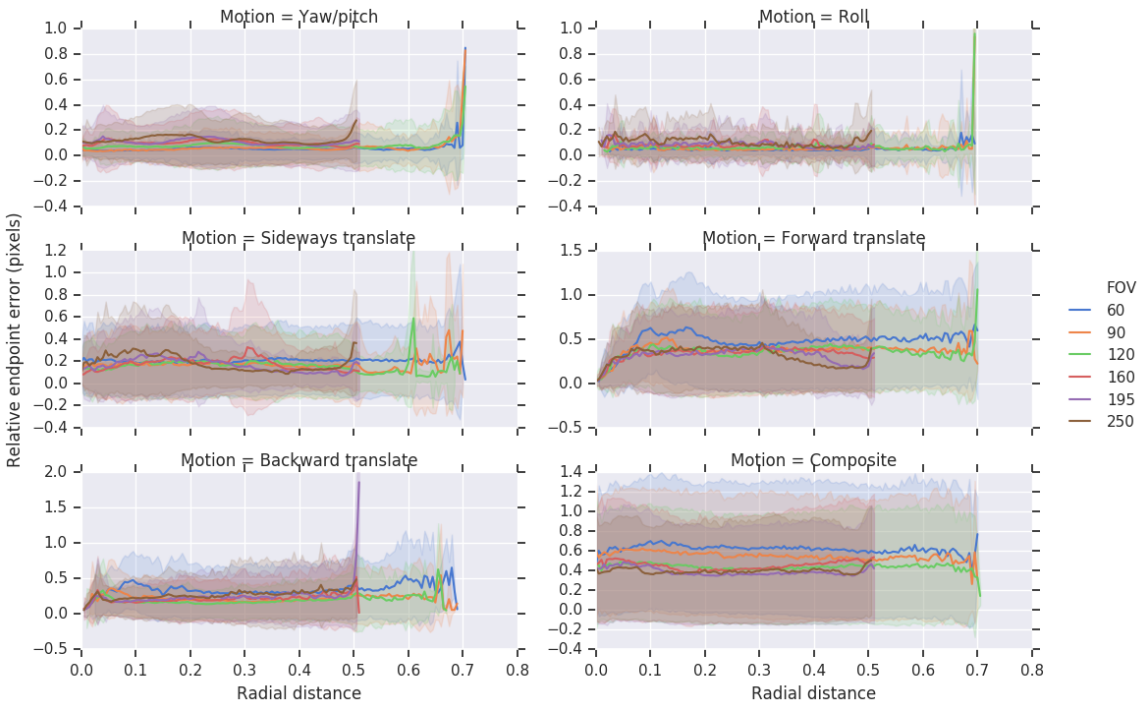


Figure 4-4: Distribution of change in endpoint error over various radial distances for different FOVs and motion types

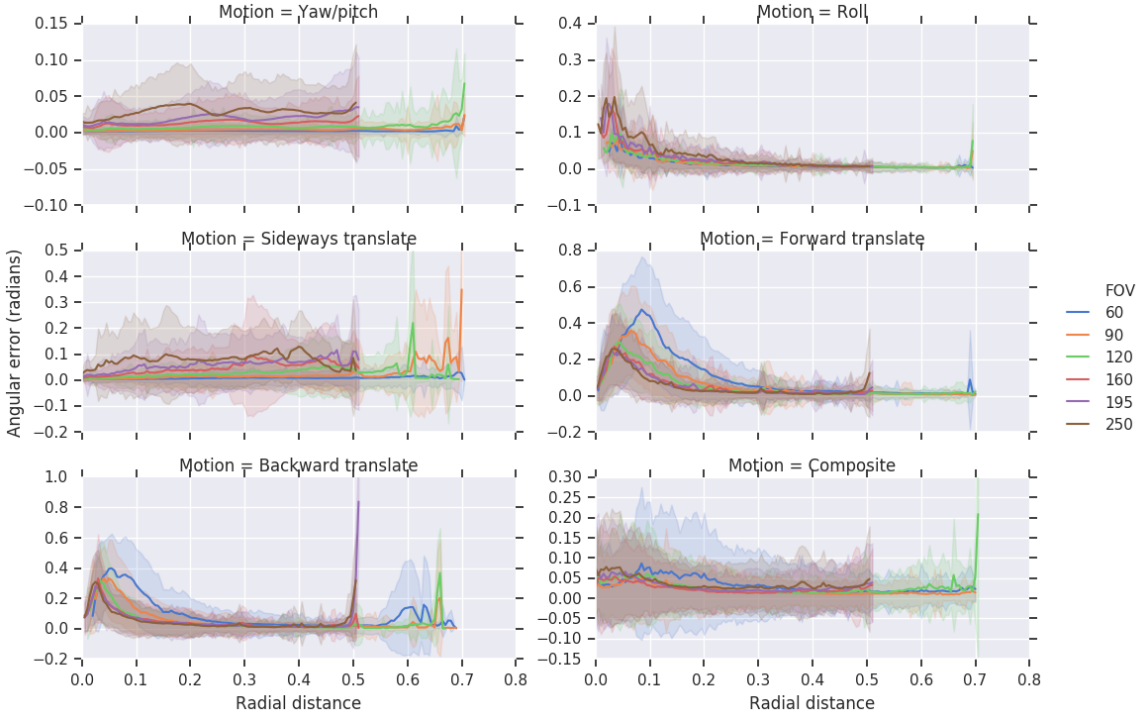


Figure 4-5: Distribution of angular error over various radial distances for different FOVs and motion types

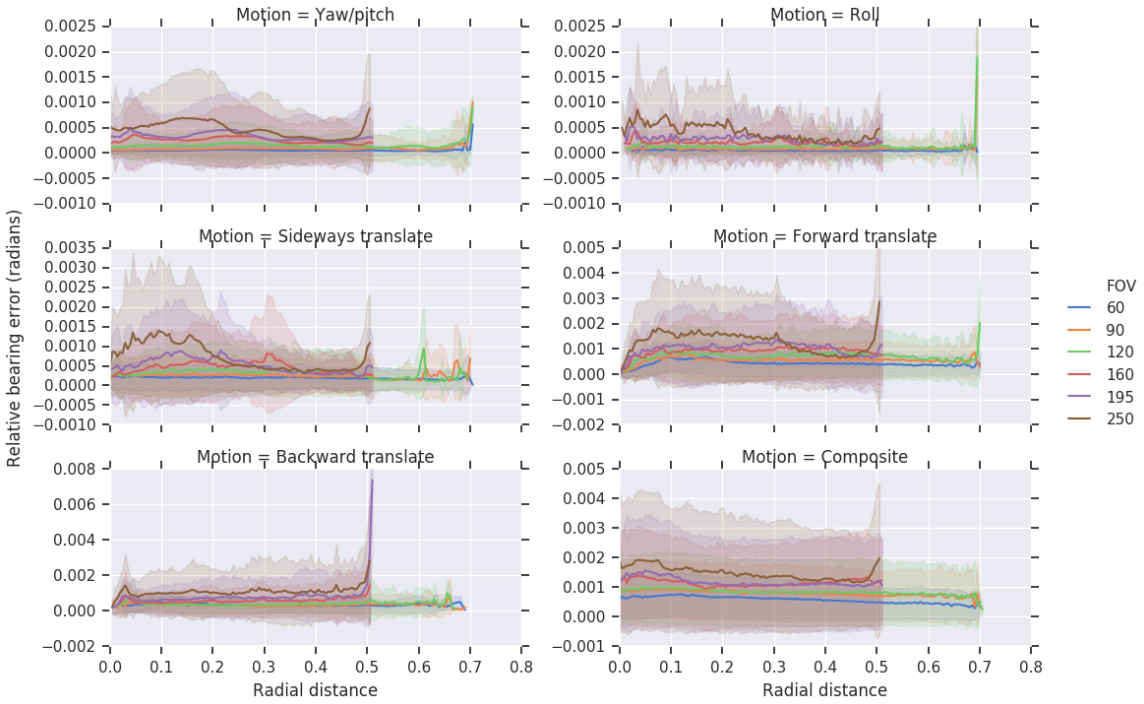


Figure 4-6: Distribution of change in bearing error over various radial distances for different FOVs and motion types

to the trends are the angular errors for the roll, forward translate, and backward translate motions, where the errors are higher near the center of the image. This is because features move very little near the center for these motions, so any tracking error will result in large discrepancies in the flow vector direction.

To normalize for the effects of error amplification due to stretching from uneven angular resolution, the bearing error will also be analyzed as a function of radial distance. Similar to endpoint error, bearing error is also an absolute error metric and tends to accumulate over a track's lifetime. Thus, only the change in bearing error between consecutive frames will be considered. Figure 4-6 shows similar plots as before, using bearing errors instead. It can be seen that the error trends are similar, with wider FOVs having increased errors near the outer edges. This shows that the change in distortion at the edges still contributes to the tracking error, rather than just the stretching amplifying the errors in pixel space. In addition, while the endpoint errors are very similar across FOVs, the bearing errors exhibit greater differences between FOVs, where larger FOVs have higher bearing errors. This is due to the

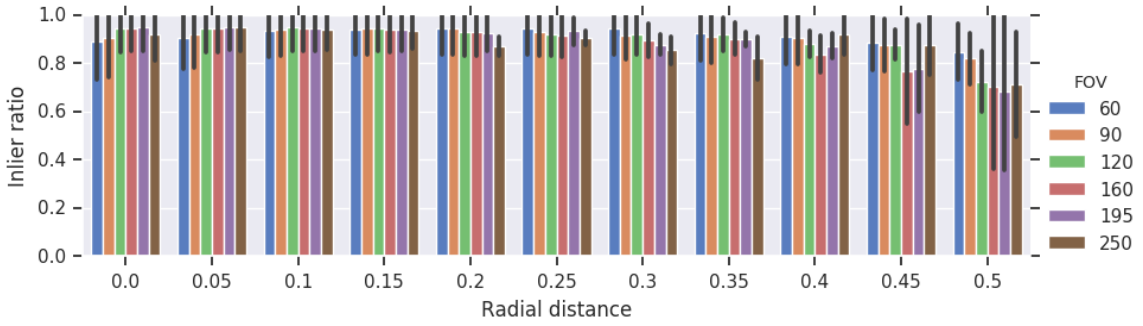


Figure 4-7: Inlier ratio over radial distance for different FOVs

lower angular resolution of higher FOV cameras, so the same errors in pixel space result in greater differences between unprojected ray angles.

In addition to analyzing which portions of the images contribute most to tracking errors, contributions of radial regions to track failures can also be analyzed. This can be done by analyzing the inlier ratio as a function of radial distance. Figure 4-7 shows average inlier ratios (with standard deviations) in different radial regions for various FOVs over all motion types. It can be seen that the distributions are very similar across FOVs, with a slightly higher outlier (tracking failure) rate in the outer portions of the image for wider FOVs.

Other useful results to consider are distributions of track lifetime until failure, as well as absolute endpoint and bearing errors over a track's lifetime in frames. This gives information about whether wider FOV cameras actually provide longer tracks that are usable. Table 4.1 shows statistics for track lifetimes, and figures 4-8 and 4-9 show error accumulation over track lifetime in frames, for various FOVs and motion types. Most motion types demonstrate longer tracks for wider FOVs, although the increase in track length tapers off. The errors increase at the same rate, but the longer lifetimes for wider FOVs cause the errors to accumulate higher. Notable exceptions are the forward and backward translation motions along the optical axis down an urban canyon, where there is no significant difference in track lifetimes. This can be explained by analyzing the depths of features returned by the feature detector. For wide-angle cameras, the image is more zoomed out, so closer features are detected as further ones are too small. Narrower FOV cameras are more zoomed in so further

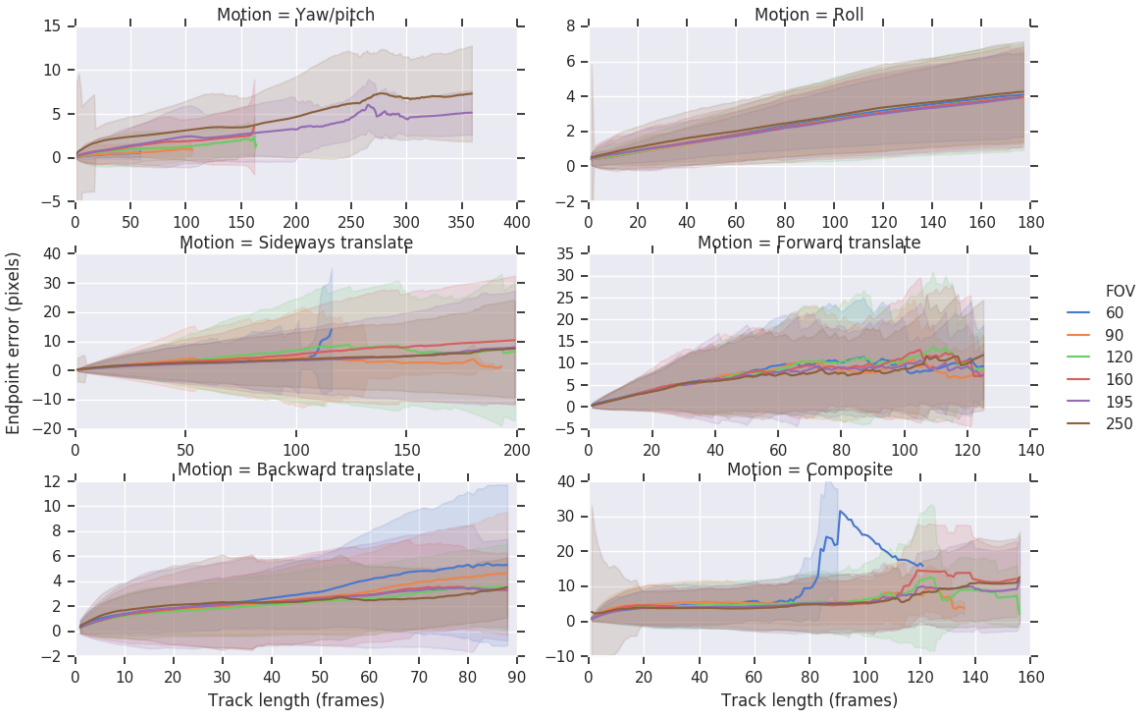


Figure 4-8: Endpoint error over track lifetime for various FOVs and motion types

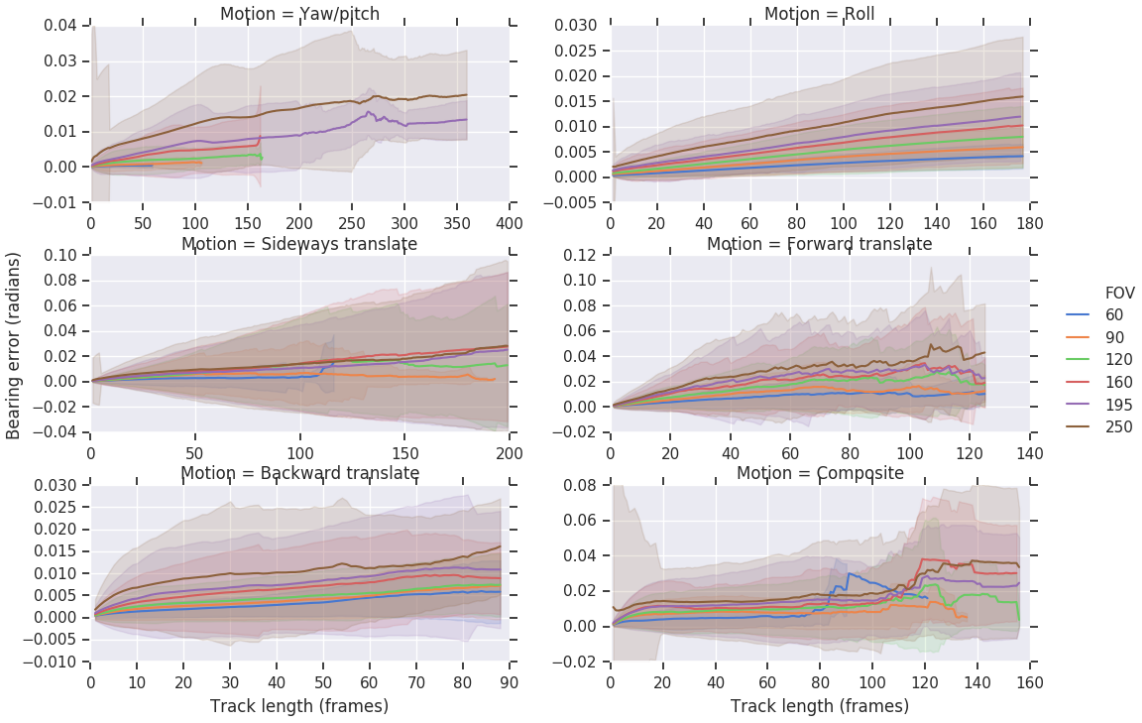


Figure 4-9: Bearing error over track lifetime for various FOVs and motion types

Motion	FOV	Mean	Median	3rd quartile	SD
Yaw/pitch	60	56.16	54.00	71.00	27.06
	90	75.67	71.00	94.00	41.03
	120	89.21	85.00	124.00	48.49
	160	114.26	125.00	157.00	58.43
	195	133.01	128.00	202.00	77.83
	250	146.99	135.00	230.00	99.47
Roll	60	86.39	44.00	177.00	74.15
	90	85.89	44.00	177.00	73.67
	120	83.11	40.00	177.00	73.33
	160	123.56	177.00	177.00	66.81
	195	117.12	176.00	176.00	67.64
	250	112.69	139.00	177.00	67.10
Sideways translate	60	52.14	47.00	73.00	35.64
	90	64.27	53.00	92.00	45.25
	120	68.75	52.00	95.00	56.28
	160	72.03	56.00	110.00	61.78
	195	85.30	67.00	152.00	72.47
	250	83.72	70.00	143.00	68.73
Forward translate	60	21.25	14.00	28.00	21.67
	90	20.70	14.00	26.00	21.05
	120	18.69	12.00	22.00	20.28
	160	17.36	12.00	21.00	18.66
	195	17.56	12.00	21.00	17.95
	250	17.64	13.00	23.00	15.33
Backward translate	60	44.06	42.00	64.00	25.86
	90	48.13	47.00	74.00	28.05
	120	46.94	46.00	73.00	28.24
	160	44.84	44.00	71.00	28.60
	195	44.21	43.00	70.00	28.84
	250	41.70	40.00	66.00	29.33
Composite	60	13.07	8.00	14.00	14.34
	90	14.16	8.00	15.00	16.89
	120	17.82	10.00	21.00	20.61
	160	15.99	10.00	18.00	18.46
	195	20.06	12.00	25.00	21.71
	250	20.28	12.00	24.00	21.60

Table 4.1: Track lifetime statistics for various FOVs and motion types

features are detected. Thus, in the urban canyon, all cameras cover a similar distance along the scene. Therefore, as the cameras move along this axis, features move outwards or inwards the same amount for all cameras, causing the track lifetimes to

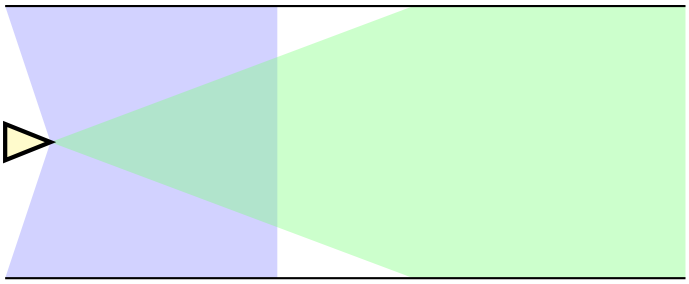


Figure 4-10: Illustration of feature detection coverage while translating along optical axis in urban canyon environment. Narrow (green) and wide (blue) cameras cover a similar depth range, so moving along this direction results in similar track lifetimes.

be similar. An illustration of this effect is shown in figure 4-10. This is an important observation that affects performance over FOVs for these types of trajectories in the downstream pipeline as well.

The results shown thus far have been based on motions at reasonable speeds for a MAV, captured at a reasonable frame rate. However, increasing the FOV of the camera may allow for more extreme speeds seen in other applications. Features move less across the image for a given motion due to the increased coverage per pixel, allowing more features to stay in the image and for the tracker to keep up. On the other hand, the features will now undergo much more change in radial distortion, which may cause tracking to fail and nullify the potential gains. To analyze this, frames are skipped to simulate different motion speeds for the yaw and pitch, sideways translate, and backward translate motions. Figure 4-11 shows the average inlier ratio (with standard deviations) for frame-to-frame tracks as a function of pose difference (i.e., degrees or meters) between frames. To ensure that the inlier tracks are still accurate, figure 4-12 shows average change in endpoint errors (with standard deviations) over all tracks for different rates. The results show that there is a clear benefit of fisheye cameras for high rotational and sideways translation rates. That is, features tend to leave the images of narrow FOV cameras before they fail from distortion in the wide FOV cameras, as reflected by the earlier drop-offs. The gains taper off, however, within the wide FOV range as the increased distortion starts to reduce the potential benefits. An exception is the backward translation motion, where higher FOVs perform worse at faster rates. This is because of the effect explained previously for translational

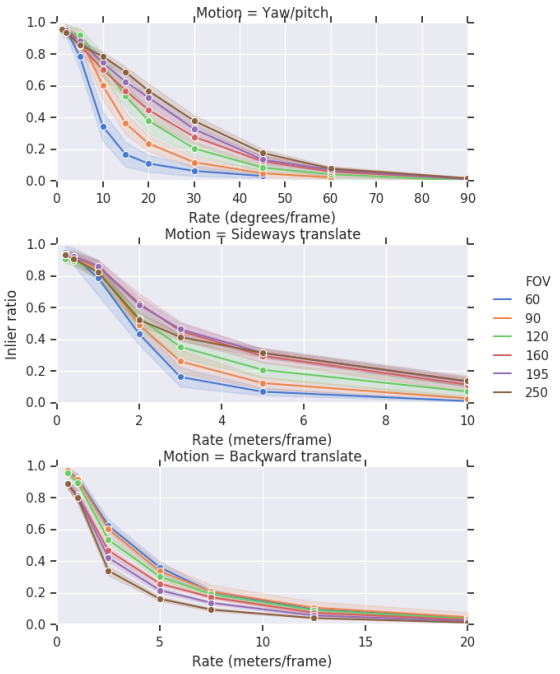


Figure 4-11: Inlier ratio over motion rates for various FOVs

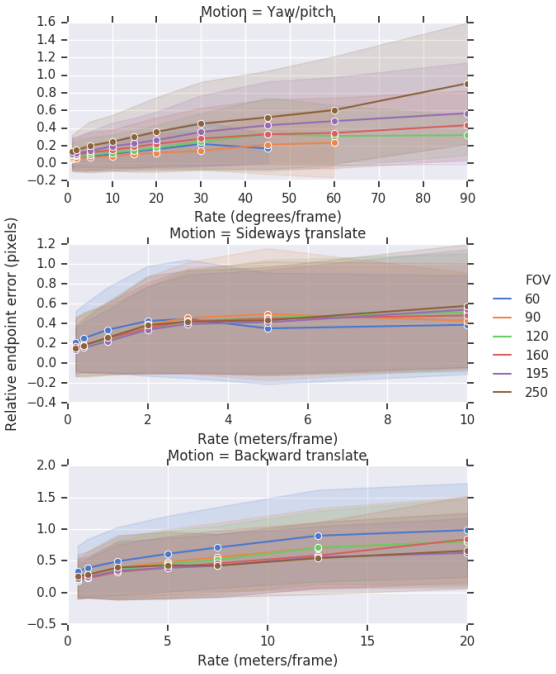


Figure 4-12: Change in endpoint error over motion rates for various FOVs

motions along the optical axis where features move the same amount in the image for all FOVs, so the only difference is in the change in distortion experienced, from which fisheye cameras suffer. Nevertheless, the error plots show that even for the most extreme and unrealistic motions, the inlier tracks of fisheye cameras are still accurate and usable despite being small in number. This allows the downstream SLAM pipeline to be robust to large motions, as will be seen in chapter 6.

From this analysis, we can conclude that tracking accuracy for fisheye cameras is similar to perspective cameras, except for slight degradation near the outer edges due to distortion. Despite this, the wider FOV brings benefits to the tracking task. With the exception of translating down an urban canyon along the optical axis, omnidirectional fisheye cameras provide longer usable tracks. Furthermore, fisheye cameras also provide clear benefits for more extreme rotational motion rates, giving accurate tracks even at unrealistic rates. Finally, the distribution of usable tracks throughout a much larger portion of the scene provides improvements to the stability of the downstream SLAM pipeline, as will be analyzed in chapter 6.

Chapter 5

Feature Matching Evaluation

In addition to feature tracking, descriptor-based feature matching is another approach to find feature correspondences. In this section, various feature descriptors will be evaluated for how invariant they are to the forms of warping introduced by fisheye cameras, and the performance of the matching task across different fields of view will be evaluated. Additionally, a novel method will first be introduced that locally rectifies keypoints to allow descriptors to become more invariant to fisheye distortion, which will also be evaluated.

5.1 Local Rectification

In addition to evaluating existing off-the-shelf descriptors, several novel descriptors [33, 52, 88, 6, 86, 93] have shown promise in handling radial distortion from wide-angle images. It is thus potentially valuable to evaluate these approaches alongside existing descriptors. Although none of the novel descriptors have implementations readily available for evaluation, we note that a number of them take similar approaches to handle distortion. Specifically, they use the calibrated camera model or distortion model to remap how the descriptor is sampled, by sampling on the undistorted pixel coordinates instead. This is effectively rectifying the entire image using the distortion model and computing descriptors on that image. Although this allows the descriptors to be unaffected by the distortion, rectification can only be performed on parts of

the image that are within 180 degrees FOV, limiting the usable image area. We therefore present a method that only locally rectifies the pixels inside a keypoint. This method can be applied to any descriptor, and will be evaluated later in this chapter to encapsulate and substitute for the evaluation of the novel descriptors.

The local rectification method assumes the use of a calibrated camera model, and remaps the pixels inside a keypoint such that they resemble being captured in the center of the image where distortion is minimal. The feature patches will therefore appear similarly (up to interpolation errors) in all regions of the image, so the descriptors that later sample these pixels will effectively become more invariant to radial distortion. The pixel coordinate map is created as follows, for every detected keypoint in the image. Let the pixel coordinate of the center of a keypoint be $\mathbf{k}$, and let $\mathbf{p} = \begin{bmatrix} c_x & c_y \end{bmatrix}^T$ be the principal point of the full image where the optical axis lies, where c_x and c_y are from the camera model. The function $\mathbf{m}(\mathbf{x}')$ which maps a locally rectified pixel coordinate $\mathbf{x}'$ inside the keypoint to a pixel coordinate in the original image is defined as follows:

$$\mathbf{m}(\mathbf{x}') = \mathbf{C}(\mathbf{R}\mathbf{C}^{-1}(\mathbf{x}' - \mathbf{k} + \mathbf{p})) \tag{5.1}$$

$$\hat{\mathbf{C}}^{-1}(\mathbf{k}) = \mathbf{R} \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{5.2}$$

where $\mathbf{C}(\mathbf{X})$ and $\mathbf{C}^{-1}(\mathbf{x})$ are the camera model projection and unprojection functions, and $\mathbf{R}$ is the rotation matrix that rotates the optical axis vector to the unprojected ray of $\mathbf{k}$, the keypoint coordinate. The map essentially calculates the unprojected rays as if the keypoint were captured in the optical center of the image, and rotates and reprojects the rays to determine which pixels in the original image to sample. A descriptor can now operate in the locally rectified space by remapping the pixels it samples for a given keypoint. That is, for each coordinate $\mathbf{x}'$ sampled by the descriptor, sample the pixel at location $\mathbf{m}(\mathbf{x}')$ in the original image instead. This has the effect of removing radial distortion from the feature sampled by the descriptor and

will potentially help descriptors become more invariant to warping, so this approach will be evaluated as part of the main feature matching evaluation.

5.2 Evaluation Method

As ground truth feature locations are needed to evaluate correctness of matches, the same datasets generated from simulated urban environments will be used for evaluating feature matching. Descriptors will be evaluated on how well they can be matched to the first frame in the sequence, over increasing magnitudes of motion. As such, only simple motion sequences (i.e., pure rotation or translation) that keep features in the first frame visible will be used. The descriptors that will be evaluated are SIFT [53], SURF [9], ORB [71], BRISK [45], AKAZE [64], KAZE [4], FREAK [3], DAISY [81], LATCH [47], and BINBOOST [82]. An attempt was also made to evaluate SPHORB [92], but it was found to be exclusively designed for use on equirectangular panoramic images and did not function on raw fisheye images. The off-the-shelf descriptors will be evaluated first, and then the local rectification method will be used on the descriptors to evaluate improvement.

Feature detection is performed on every frame, using the region-based detection method described in the previous chapter. For descriptors that are designed for a specific detector, that detector is used. Otherwise, the SIFT detector is used, except for FREAK which is used with SURF, as has been done in a previous evaluation study [11]. The threshold parameters for the detectors are tuned to return approximately equal numbers of features. Descriptors are then computed for every keypoint, using implementations from OpenCV [13]. Descriptors from the first frame are matched against by subsequent frames. Because real-time performance is not needed, the brute force matcher is used, which compares every descriptor in the first frame for every descriptor in the current frame and finds the closest match in the descriptor space. A cross check is also performed to check if the descriptor in the current frame is the closest one to the descriptor in the first frame. If the descriptor distance between the closest match is below a threshold, then the match is returned. This threshold is

adjusted dynamically to evaluate the descriptor, as will be discussed.

Ground truth feature locations are obtained the same way as before, by unprojecting the detected keypoints into 3D coordinates in world frame, and reprojecting them into the current frame. To determine if a match is correct, the size of the keypoint is used, which is a circle representing the image patch from which the descriptor is calculated. A match is correct if it overlaps sufficiently with its ground truth keypoint. To quantify overlap, the intersection-over-union (IOU) measurement is used:

$$IOU = \frac{K \cap K_g}{K \cup K_g} \tag{5.3}$$

where $K \cap K_g$ is the overlapping area between a matched keypoint with its ground truth keypoint, and $K \cup K_g$ is the combined area covered by both keypoints. In addition, the L2 pixel distance between the keypoint centers is also used to determine match correctness. If the IOU is above a threshold and pixel distance is below a threshold, the match is correct. For this evaluation, an IOU threshold of 50% and pixel distance threshold of 10 pixels is used.

Another important quantity for evaluating metrics is the number of correspondences, or the number of features from the first frame that have also been detected in the current frame. It represents the maximum possible number of correct matches. To obtain this number, all keypoints from the first frame are reprojected into the current frame. For each keypoint in the current frame, each reprojected keypoint from the first frame is checked to see if it is the same feature using the same IOU and pixel distance threshold method. Instead of simply counting the number of keypoints in the current frame that have correspondences, which can double count (i.e., two keypoints overlapping with the same keypoint), the following approach is used. For each correspondence found, the keypoints from both frames are flagged. Flagged keypoints are then counted for each frame, and the smaller number is used as the number of correspondences.

5.3 Evaluation Metrics

The feature matching task can first be framed as a binary classification problem, which allows metrics for those problems to be used. The task can be framed as follows: for every pair of keypoints between the two images, classify it as either a match or not a match, using the descriptor distance as a decision threshold. If there are n and m keypoints in the images respectively, then the total number of pairs is nm. The matching algorithm only finds one match for every keypoint, so the maximum number of positive classifications is reduced to $\min(n, m)$.

As a binary classification problem, the terms true positive, false positive, true negative, and false negative can now be defined in the context of the matching problem. A *true positive* (TP) is a pair of keypoints that represents the same feature and is determined correctly to be a match. A *false positive* (FP) is a pair that is incorrectly determined to be a match. A *true negative* (TN) is a pair of keypoints that do not represent the same feature and is rejected correctly. A *false negative* (FN) is a pair of keypoints that do represent the same feature but is incorrectly rejected. Note that both true positives and false negatives are pairs of keypoints that represent the same feature, so the total number of true positives and false negatives is the number of correspondences (number of potential correct matches) discussed earlier.

Common binary classification metrics will now be defined. The *precision* of a classifier measures how many of the positive classifications are correct, or the ability of the feature matcher to distinguish between correct and incorrect matches. It is defined as the ratio between the number of true positives and total number of true positives and false positives, which in this case is the number of correct matches over the total number of returned matches:

$$Precision = \frac{TP}{TP + FP} = \frac{\#\text{ of correct matches}}{\#\text{ of total matches}} \tag{5.4}$$

The *recall* or *true positive rate* (TPR) of a classifier measures how many of the potential positive classifications are identified, or the ability of the feature matcher to identify correct matches out of the set of feature correspondences. It is defined as the

ratio between the number of true positives and the total number of true positives and false negatives, which in this case is the number of correct matches over the number of possible correct matches (number of correspondences):

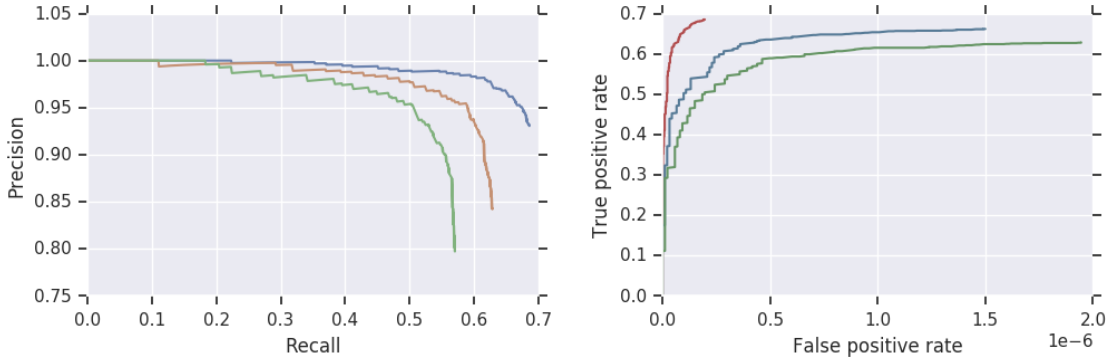
$$Recall = TPR = \frac{TP}{TP + FN} = \frac{\#\text{ of correct matches}}{\#\text{ of correspondences}} \tag{5.5}$$

The *false positive rate* (FPR) of a classifier measures how many of the negative classifications are incorrectly identified as positive, or the likelihood of the feature matcher to incorrectly return incorrect matches out of the pairs of keypoints that are not the same feature. It is defined as the ratio between the number of false positives and the total number of false positives and true negatives, which in this case is the number of incorrect matches over the number of keypoint pairs that are not the same feature:

$$FPR = \frac{FP}{FP + TN} = \frac{\#\text{ of incorrect matches}}{\#\text{ of pairs} - \#\text{ of correspondences}} \tag{5.6}$$

The value of the decision threshold, which in this case is the descriptor distance at which to reject matches, has large effects on these metrics. Raising the threshold will allow more matches to be returned, increasing the number of both true positives and false positives. This has the effect of lowering precision and raising recall and FPR. Lowering the threshold has the opposite effect. As such, when selecting a threshold, there is a tradeoff between precision and recall, and between TPR and FPR.

One way to evaluate the best possible combination of precision and recall or TPR and FPR is with precision-recall curves (for precision and recall) or receiver operating characteristics (ROC) curves (for TPR and FPR). The curves are generated by sweeping threshold values and calculating precision and recall or TPR and FPR for different thresholds, and then plotting the two values with respect to each other. Example precision-recall and ROC curves are shown in figure 5-1. Classifiers with better optimal values are reflected by a sharper curve with a "knee" closer to the top right or left corner. This metric is typically quantified by using the area under the curves as an indicator of classifier performance, called the *AUC score*.



(a) Precision-recall (b) ROC

Figure 5-1: Example precision-recall curve (left) and ROC curve (right)

When choosing between precision-recall and ROC curves, the balance of the dataset is considered. Precision-recall curves are generally better than ROC curves for imbalanced datasets, where there are significantly fewer positives than negatives or vice versa. In the feature matching problem, the number of correct matches is significantly fewer than the number of keypoint pairs, so precision-recall curves are better suited for evaluating the feature matching task.

The metrics discussed so far are designed to evaluate the discrimination ability of classifiers. Because the data in this case (descriptor distance between pairs of descriptors) is one-dimensional, the classifier is a simple threshold. For this classifier to perform meaningfully, the distribution of descriptor distances between correct matches compared to incorrect matches needs to be significantly different. Thus, in addition to evaluating the classifier, we can also evaluate the data generated by the descriptor itself, for how distinguishably it encodes correct and incorrect matches.

For a given set of matches, if we plot the descriptor distances on a line, there would ideally be two distinct clusters, one for correct matches and one for incorrect matches. One metric for evaluating separability of clusters is the *silhouette coefficient* [70], which measures how similar a data point is to its own cluster compared to other clusters. Let the correct and incorrect matches be represented by sets, and let $C(i)$ be a function that maps a match to its respective set, $\widetilde{C}(i)$ be the function that maps a match to its opposite set, and $d(i)$ be the descriptor distance of match i. The

following can then be written for each match i:

$$a(i) = \frac{1}{|C(i)| - 1} \sum_{j \in C(i), i \neq j} |d(i) - d(j)| \tag{5.7}$$

$$b(i) = \frac{1}{|\widetilde{C}(i)|} \sum_{j \in \widetilde{C}(i)} |d(i) - d(j)| \tag{5.8}$$

$$s(i) = \frac{b(i) - a(i)}{\max(a(i), b(i))} \tag{5.9}$$

where $s(i)$ is the silhouette coefficient for match i. It ranges from -1 to 1, with 1 meaning it fits well into its cluster, 0 meaning it is between two clusters, and -1 meaning it does not fit well with its cluster. If we take the average of the silhouette coefficients over all matches, a separability metric for the distribution of descriptor distances can be obtained. High values indicate good separation of clusters, and low values indicate poor separation as all matches are equally close to both clusters.

Finally, to have a large number of correct matches, there must first be a large number of correspondences, or features from the first frame that are also detected in the current frame. This *repeatability* metric evaluates the feature detectors rather than the descriptors. It is defined as the ratio between the number of correspondences for the current frame and the number of detected keypoints in the first frame that are visible in the current frame:

$$\textit{Repeatability} = \frac{\#\text{ of correspondences}}{\#\text{ of detections in base frame with valid reprojections}} \tag{5.10}$$

5.4 Evaluation Results

The existing off-the-shelf feature descriptors will be evaluated first for performance on various FOVs, and to analyze the effect of omnidirectional fisheye cameras on the feature matching task in general. Then, the proposed local rectification method will be used on select descriptors to evaluate for performance improvements.

5.4.1 Existing Methods

Similar to the previous chapter, a large portion of the analysis will be with respect to the normalized radial distance from the center of the image, as it provides the most insight into how fisheye distortion affects performance. In addition, because the motion types analyzed are pure motions with a fixed change in angle or distance per frame, we can also analyze matching performance as a function of *baseline width*, or the change in pose (rotational or translational distance) from the first frame. This is useful for tasks like loop closure where the camera is near the previously visited location, but not in the exact location or is facing a different direction, and matching must be performed to find a relative pose from the previous location. Reliable matching will allow loop closure to be performed at greater baselines from the past trajectory.

Intuitively, we are interested in the number of correct matches as a function of baseline, which gives an indication of whether feature matching algorithms are able to utilize the extra FOV provided by omnidirectional fisheye cameras in spite of the distortion. However, the number of matches depends on other factors, including the number of features detected in the first place, so a direct comparison would not be valid. To normalize for the total number of features, precision and recall values can be used, but this depends on the descriptor distance threshold chosen. Thus, precision-recall curves are used to provide insight into matching performance. Figures 5-2 and 5-3 show precision-recall curves for various descriptors and FOVs, for different motion baselines for yaw, sideways translate, and backward translate motions. Figure 5-4 summarizes these curves by showing AUC scores as a function of baseline. For yaw motions, performance of most descriptors for FOVs beyond 90 degrees drops as the baseline is increased. Performance degrades faster for higher FOVs, but matches are able to be found at greater baselines. Notably, for FOVs greater than 180 degrees, the AUC is non-zero through the entire rotation, meaning that matches can be found in all orientations. A noteworthy characteristic of these FOVs is the local peak in AUC at 180 degrees rotation baseline, due to features at the edges of the first image

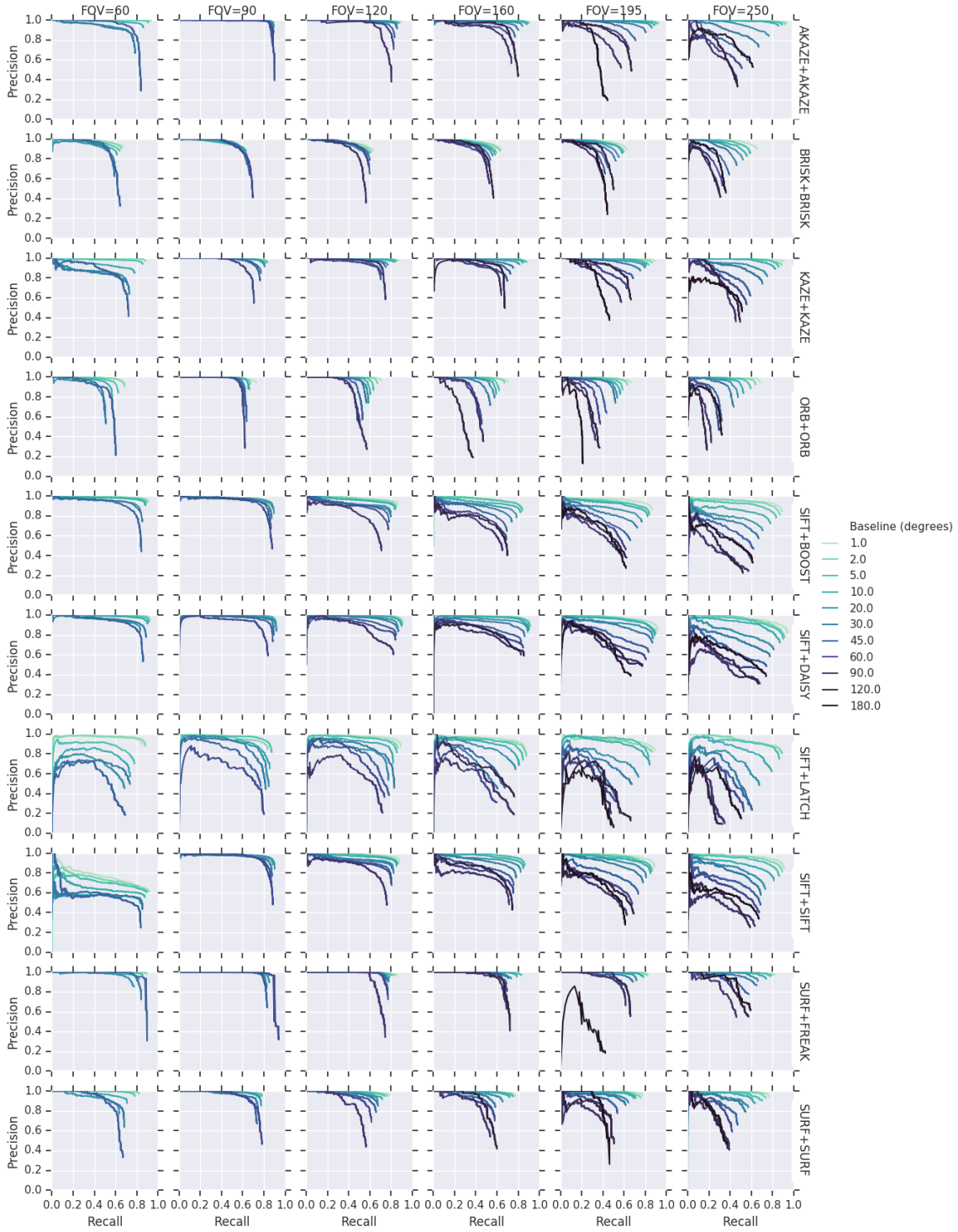


Figure 5-2: Precision-recall curves for various descriptors and FOVs, for yaw motion baselines

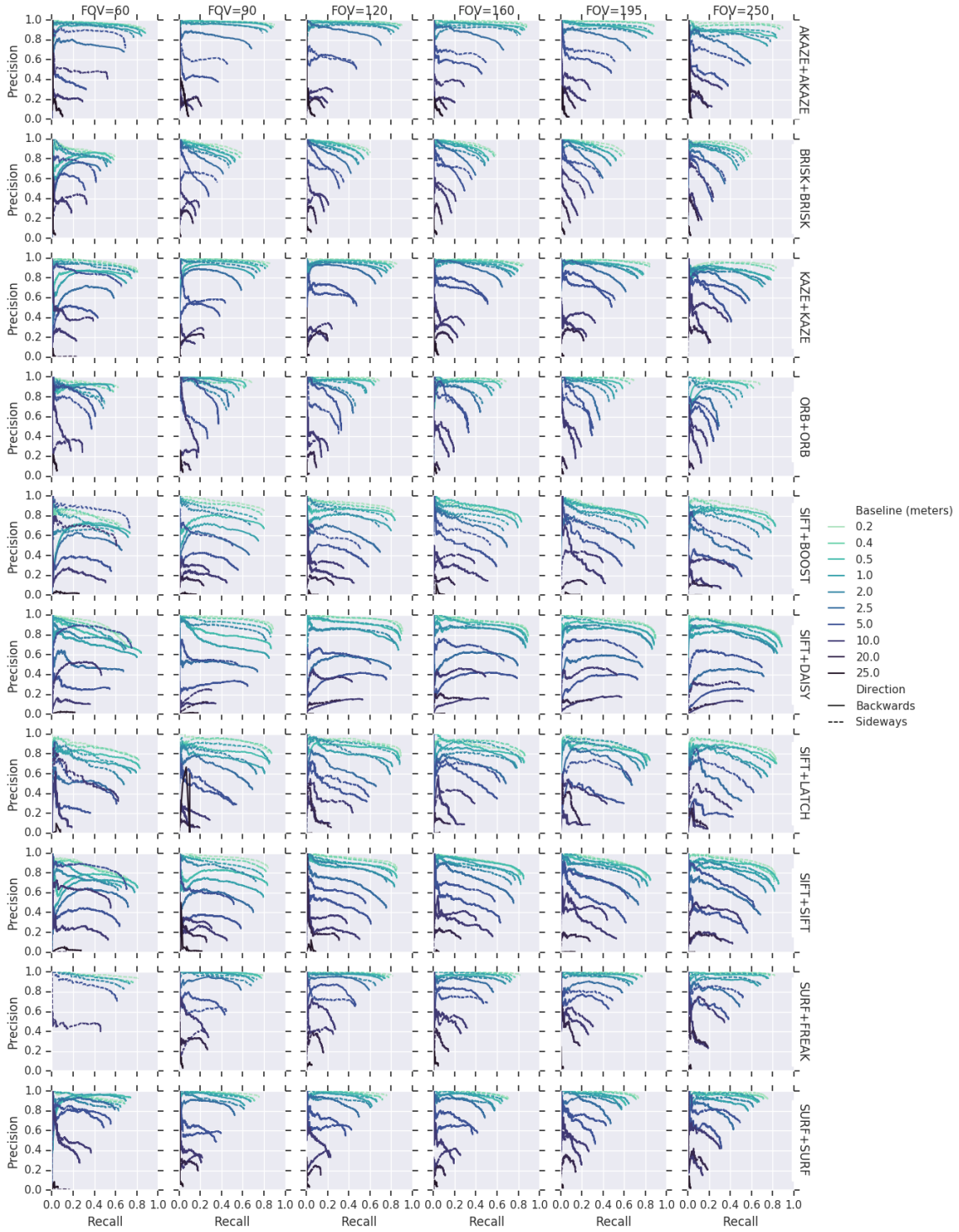
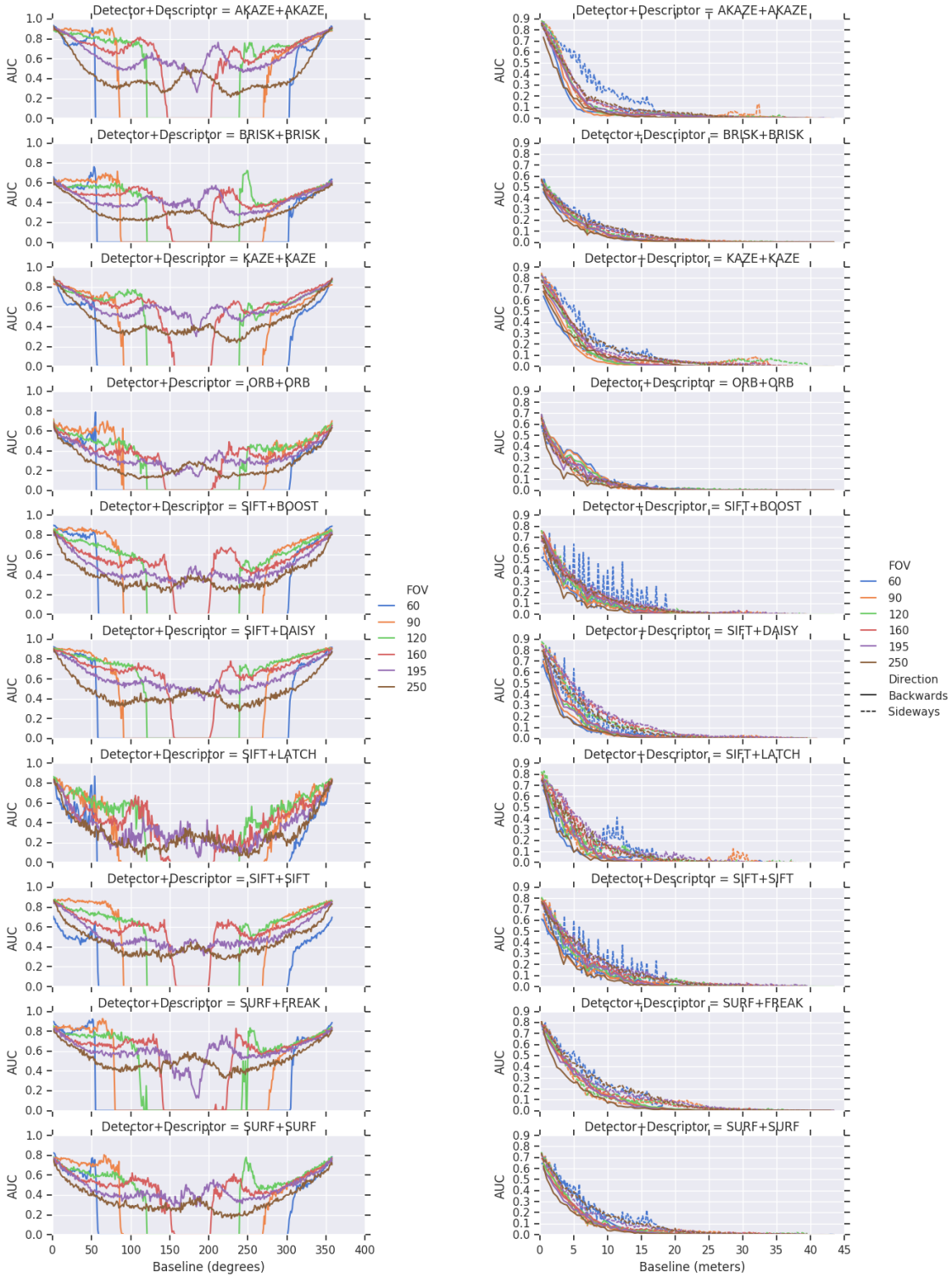


Figure 5-3: Precision-recall curves for various descriptors and FOVs, for translation motion baselines



(a) Yaw (b) Translation

Figure 5-4: AUC scores over motion baselines for various descriptors and FOVs

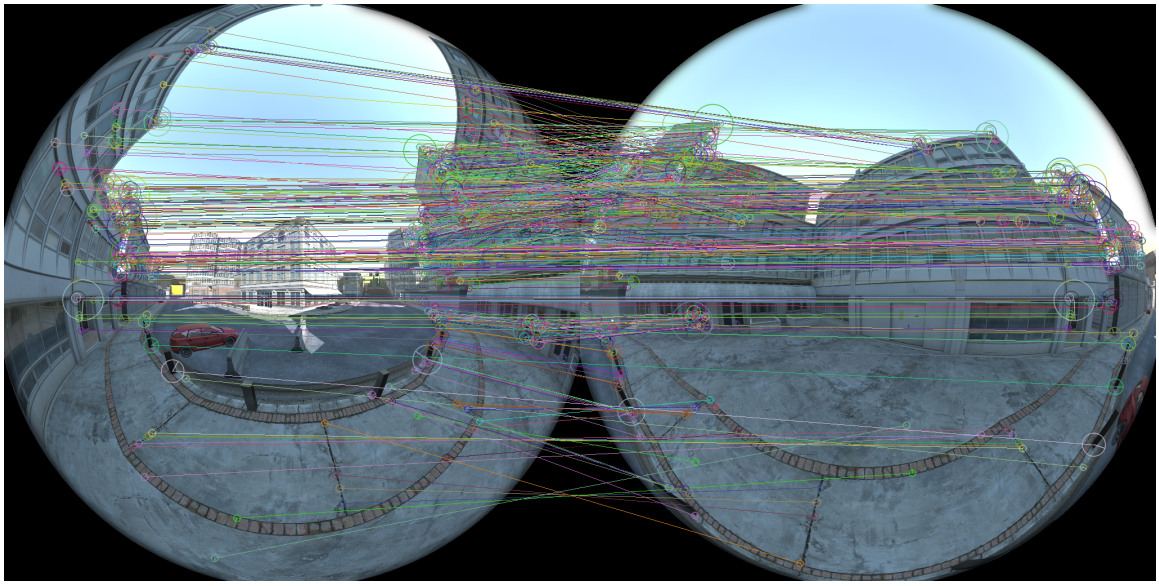


Figure 5-5: Matches between two images from a 250 degree FOV camera facing opposite directions

reappearing at the opposite edges, as shown in figure 5-5. This suggests the ability of omnidirectional fisheye cameras to localize in a scene regardless of orientation. For translational motions, all FOVs drop in performance at a similar rate, with no significant degradation with fisheye cameras. For sideways translation, smaller FOVs reach zero AUC earlier due to all features leaving the cameras, although at these points the AUC for higher FOVs is already very close to zero so the gain is not significant.

To demonstrate the ability of omnidirectional fisheye cameras to localize in all orientations, an essential matrix is estimated between the first frame and each subsequent frame in the yaw dataset by using the five-point algorithm [62] on the matches. Because the camera only undergoes pure rotation, the lack of translational scale in the essential matrix does not matter. The rotation matrix is extracted from the essential matrix and the rotational error from the ground truth pose is obtained. Figure 5-6 shows the rotational errors over rotation baseline for various descriptors and FOVs, and figure 5-7 shows the inlier ratios for the pose estimates. It is clear that omnidirectional fisheye cameras with FOVs over 180 degrees are able to achieve accurate pose estimates in all orientations, with accuracy increasing with FOV. Despite the

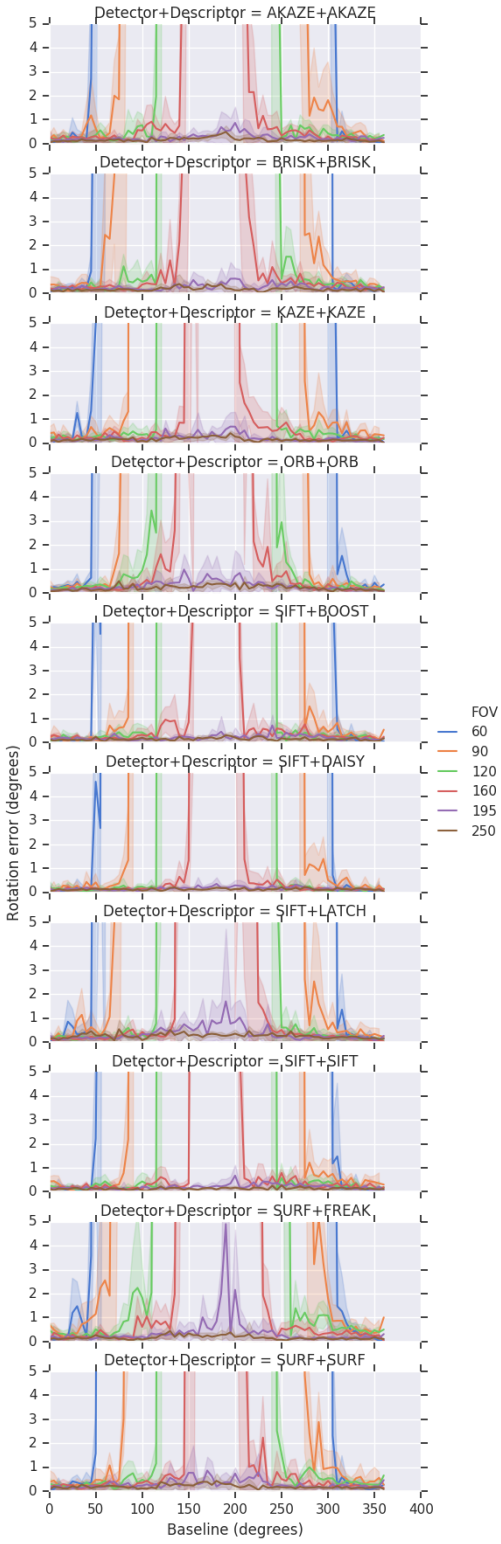


Figure 5-6: Rotation error over yaw baseline for various descriptors and FOVs

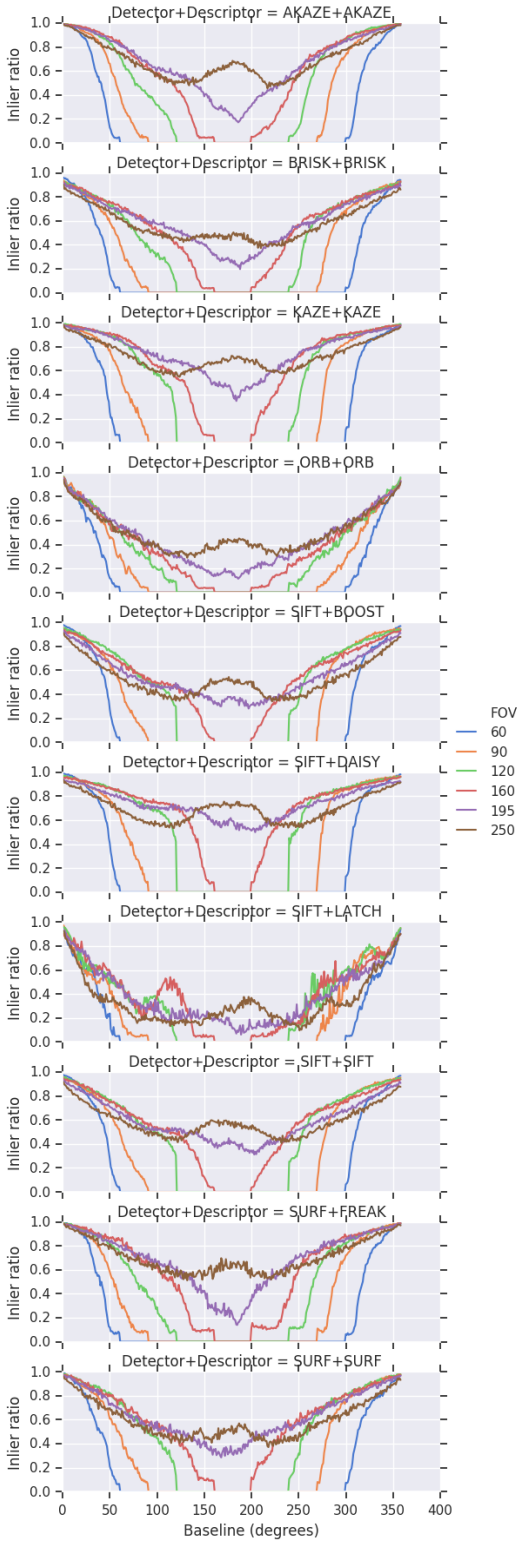


Figure 5-7: Pose estimate inlier ratio over yaw baseline for various descriptors and FOVs

decreased performance of descriptors on wide-angle cameras, they are still able to generate more inlier matches than with narrow FOV cameras, allowing for accurate pose estimation at wide rotation baselines.

Although the performance of descriptors does not detract from the benefits of omnidirectional fisheye cameras, it is still useful to understand the cause of the performance degradations for large FOVs. To do this, we can analyze the distributions of descriptor-space distances within match pairs. In particular, we are interested in whether correct matches, where both descriptors encode the same feature, have significantly closer descriptor distances than incorrect matches. This will allow correct matches to be distinguished from incorrect matches based on the descriptor distance threshold. If these distributions are analyzed as a function of change in radial distance, which represents the change in distortion of a feature, we are effectively evaluating how invariant the descriptors are to radial distortion. An ideal descriptor would encode a feature with the same descriptor vector regardless of how much change in distortion it experiences, so the descriptor distances would be low for all changes in radial distance. Figure 5-8 shows these distributions for various descriptors and FOVs, over all motions. It is clear that for fisheye cameras, the distributions overlap for large changes in radial distance, regardless of the descriptor used. This indicates that none of the descriptors handle radial distortion particularly well, and correct matches cannot be easily distinguished when a feature moves too much across the image. To summarize the overlap in distributions, figure 5-10 shows plots of silhouette coefficients over changes in radial distance for various FOVs and descriptors.

Although this analysis shows that existing descriptors do not handle fisheye distortion well, it does not mean that there are no benefits from using fisheye cameras for feature matching. The previous analysis shows performance as a function of radial distance, but a given change in radial distance in a wide-angle camera covers much more of a scene than the same change in a narrow FOV camera. Therefore, the previous analysis misrepresents how much a feature can physically move before descriptors can no longer handle the distortion. Instead of using change in radial distance, change in unprojected ray angle from the optical axis can be used. For an

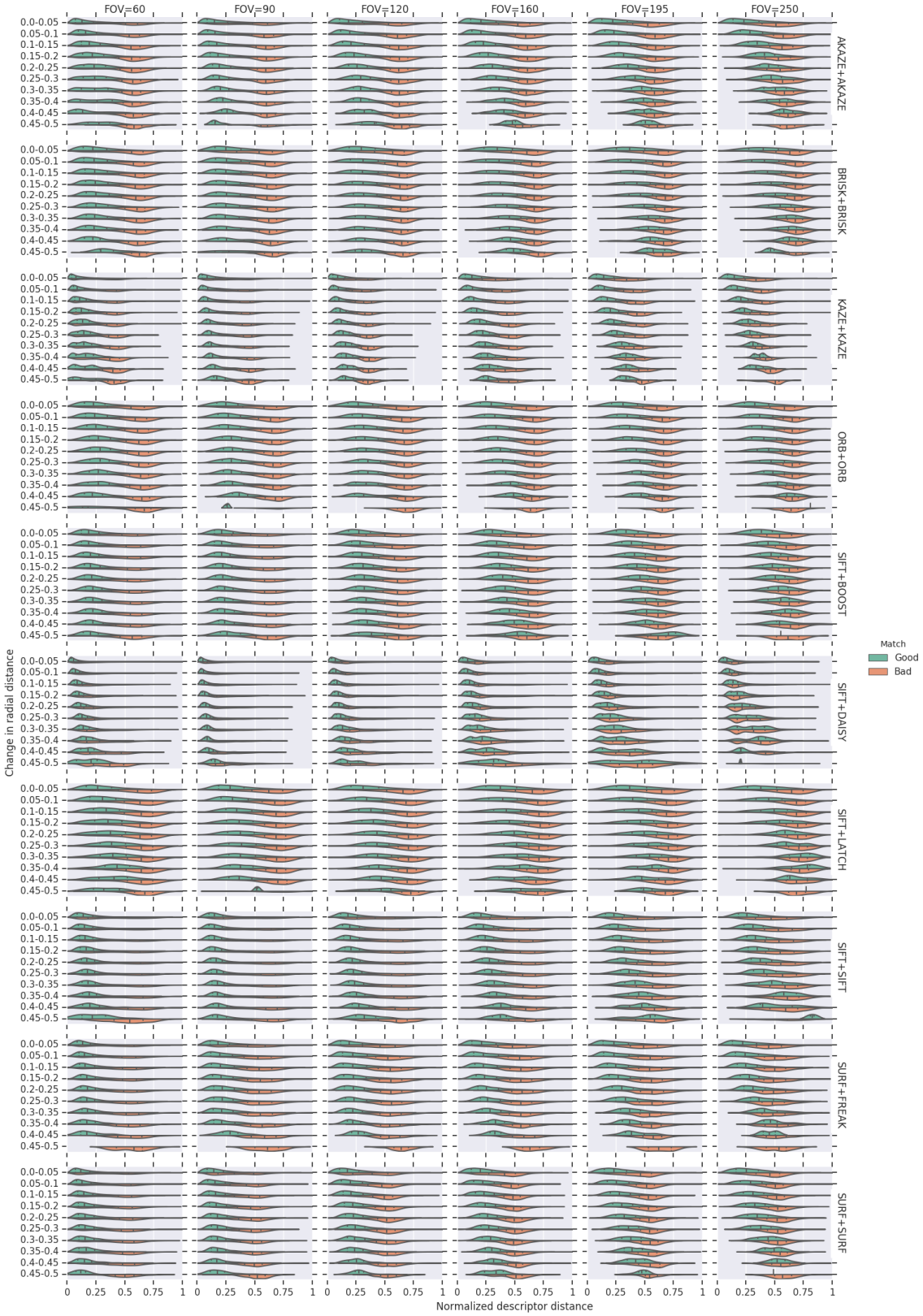


Figure 5-8: Descriptor distance distributions over changes in radial distance for various descriptors and FOVs

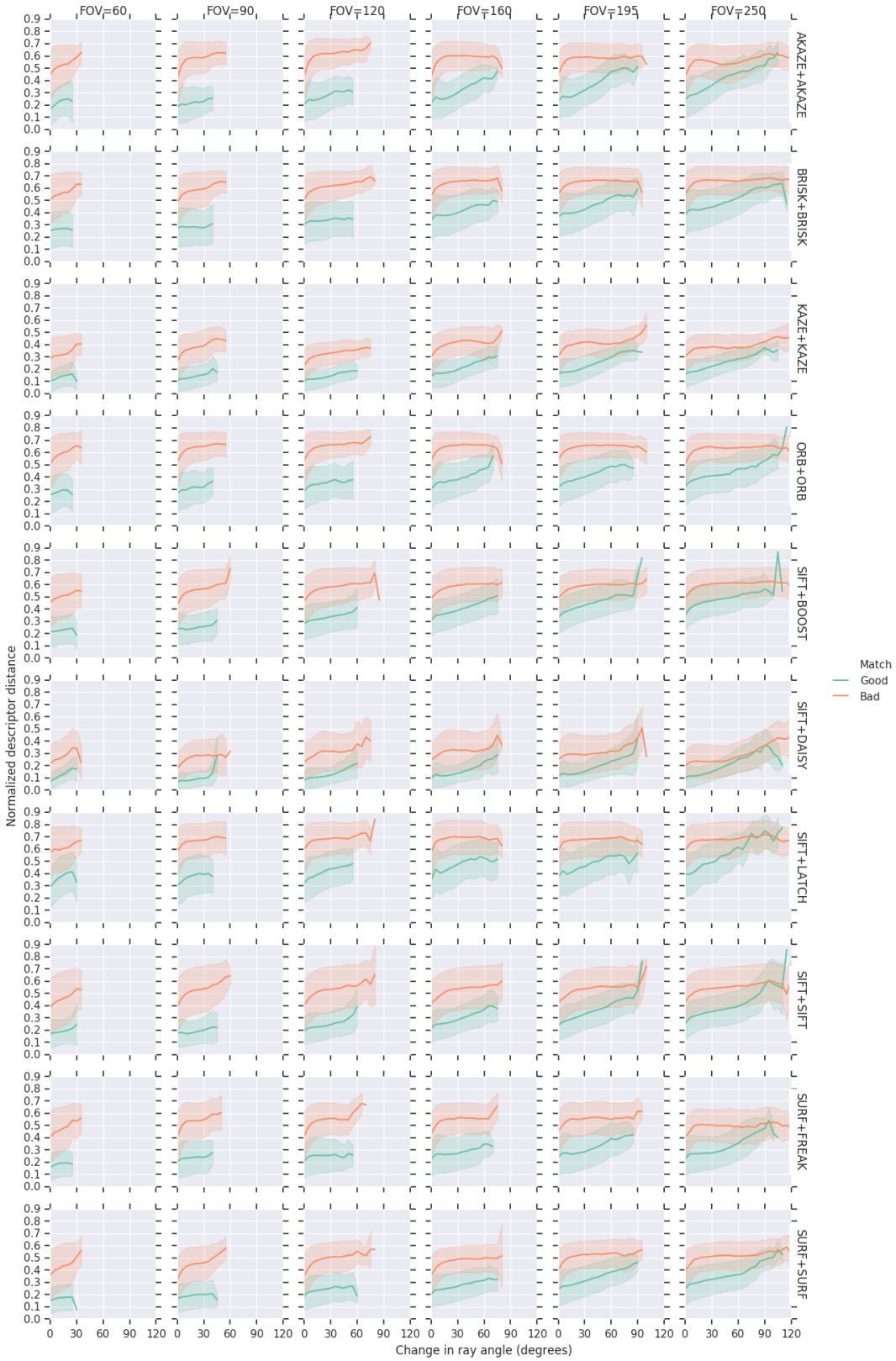


Figure 5-9: Descriptor distance distributions over changes in ray angle for various descriptors and FOVs

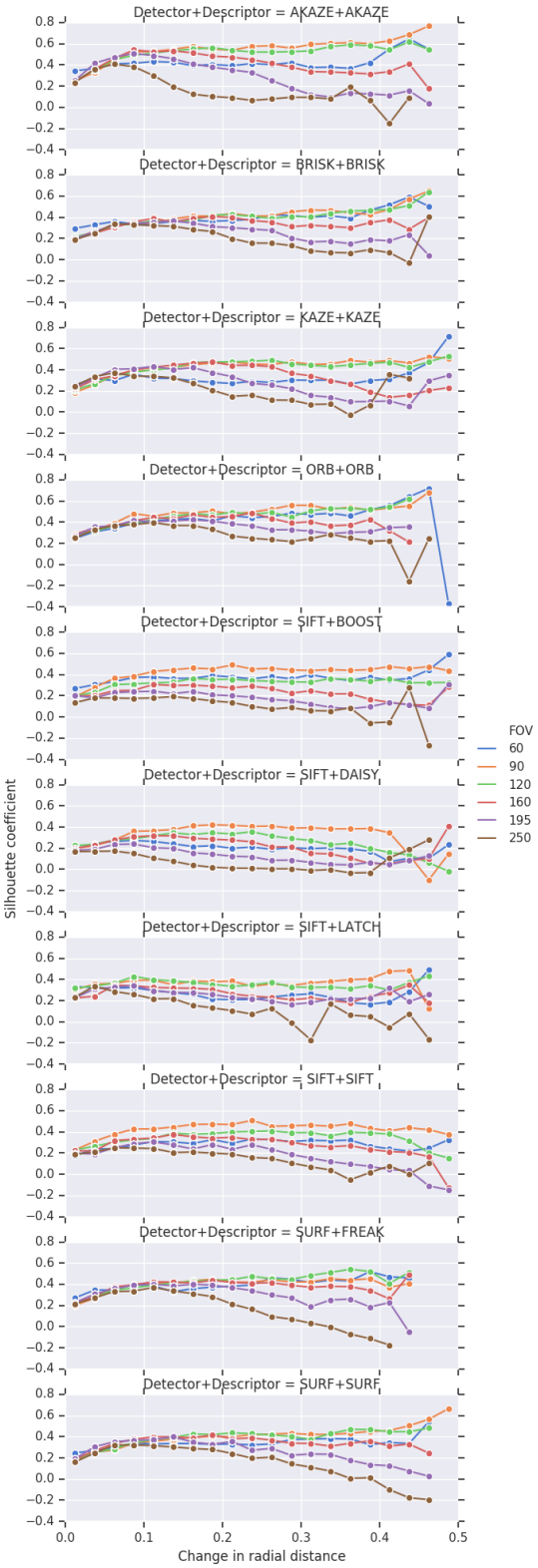


Figure 5-10: Silhouette coefficient over changes in radial distance for various descriptors and FOVs

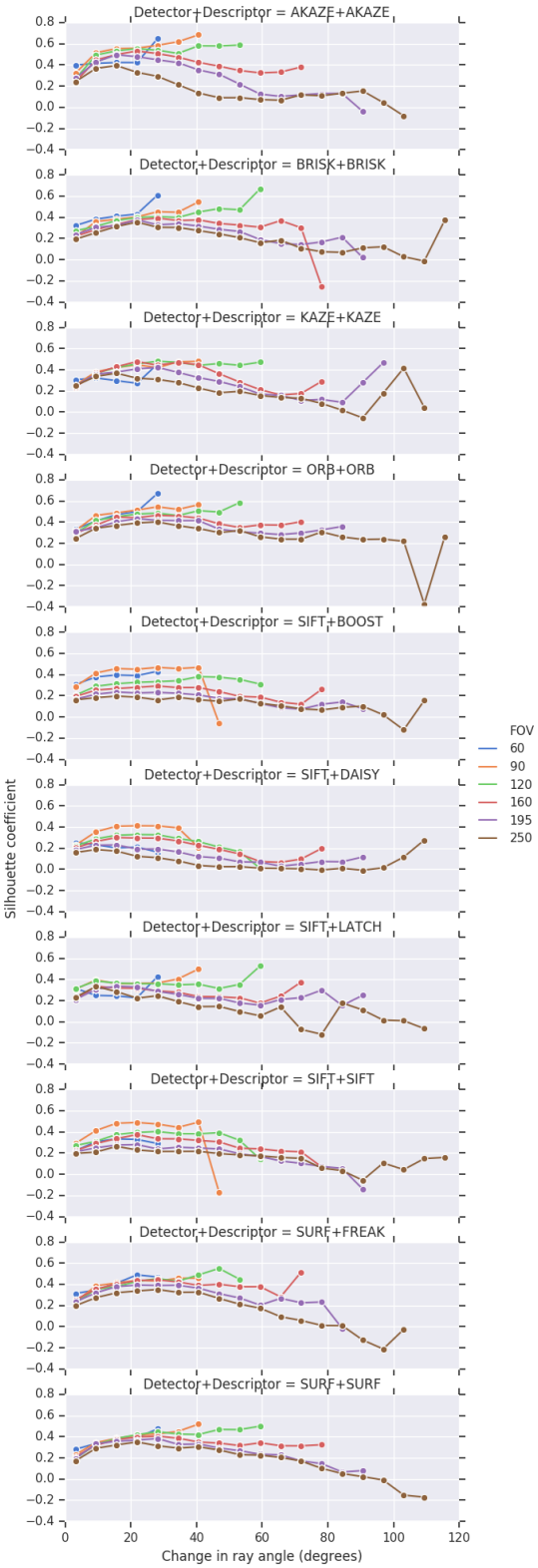


Figure 5-11: Silhouette coefficient over changes in ray angle for various descriptors and FOVs

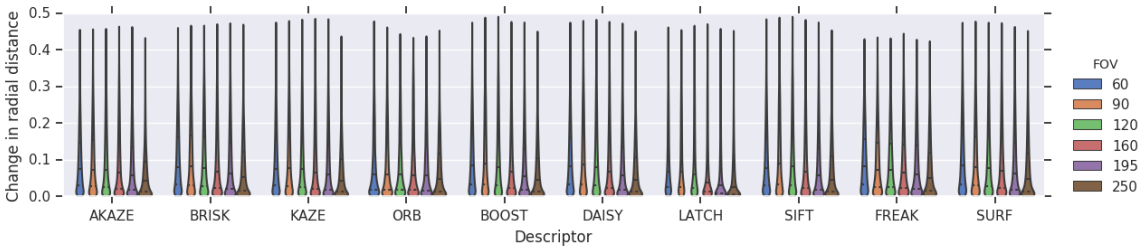


Figure 5-12: Distribution of correct matches over changes in radial distance for various descriptors and FOVs

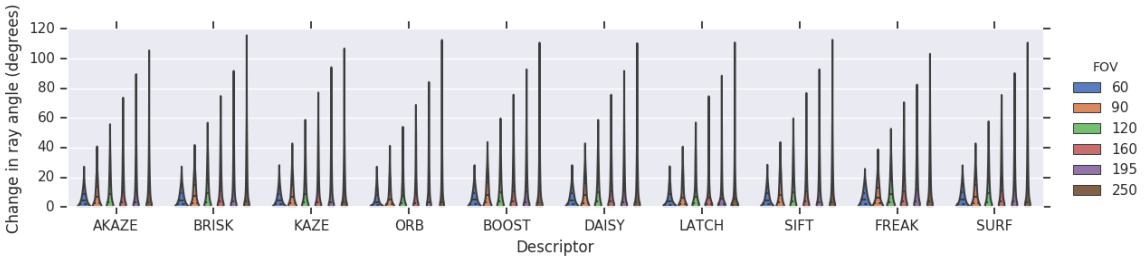


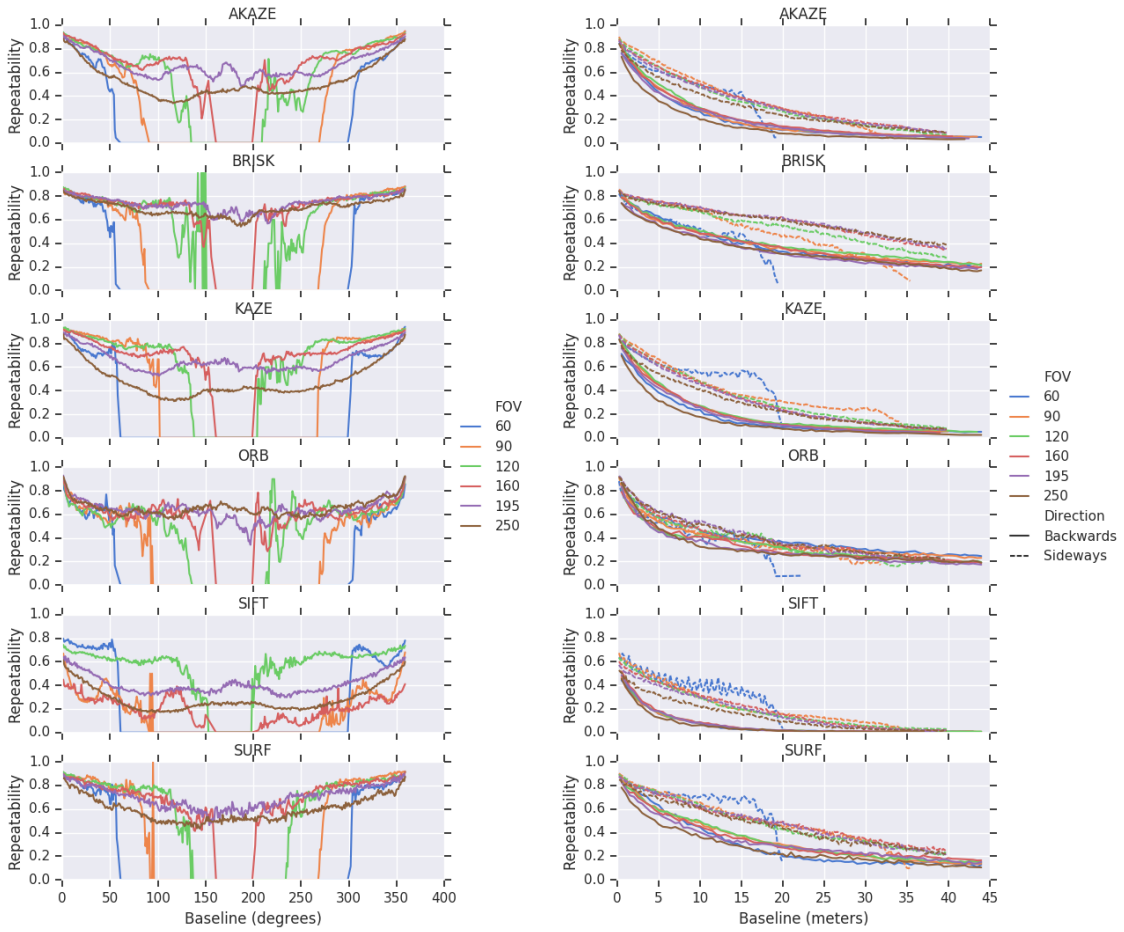
Figure 5-13: Distribution of correct matches over changes in ray angle for various descriptors and FOVs

unprojected unit ray $\hat{\mathbf{C}}^{-\mathbf{1}}(\mathbf{x})$, the angle from the optical axis is:

$$\theta = \cos^{-1}(\hat{\mathbf{C}}^{-\mathbf{1}}(\mathbf{x}) \begin{bmatrix} 0 & 0 & 1 \end{bmatrix}) \tag{5.11}$$

Figures 5-9 and 5-11 show similar plots as before but as a function of change in ray angle. Representing the data with respect to how much the features physically move, rather than in the image space, brings the performance between FOVs much closer for most descriptors. For smaller changes in ray angle, the descriptor separability is similar across FOVs, with only a slight decrease for omnidirectional fisheye cameras for some descriptors because of the slight distortion still present near the center of the images. Larger FOVs are thus able to provide matches for greater feature motions despite an increase in false positives, while retaining similar performance as perspective cameras for smaller feature motions.

To summarize the effects of change in distortion on matching ability, figures 5-12 and 5-13 show distributions of correct matches (before thresholding) over changes in radial distance and ray angle respectively. As before, there are fewer correct matches



(a) Yaw

(b) Translation

Figure 5-14: Detector repeatability over motion baselines for various FOVs and detectors

at high changes in radial distance for wider FOVs, but with respect to change in ray angle, using wide-angle lenses allows for features to physically move more from the optical axis and still be matched. There is therefore still a benefit to using omnidirectional fisheye lenses for wide-baseline feature matching despite reduced descriptor performance.

Finally, the feature detectors can be evaluated for repeatability, to analyze how consistently the same features can be detected as the camera moves. Figure 5-14 shows repeatability as a function of yaw and translation motion baselines, for various detectors and FOVs. For yaw motions with omnidirectional fisheye cameras, all detectors perform well across the entirety of a full rotation. The ORB detector (based

on FAST [69]) and BRISK detector (based on AGAST [57]) perform especially well, with minimal decrease in performance from perspective cameras. For translational motions, detectors also perform similarly for all FOVs, with performance dropping off earlier when translating sideways with narrow FOV cameras.

5.4.2 Local Rectification Evaluation

The analysis so far has shown that existing feature descriptors do not perform well with fisheye distortion, which indicates that there is room for improvement in the feature matching problem on fisheye cameras. To help introduce more invariance to distortion to existing descriptors, we will apply the proposed local rectification method on select descriptors from the previous section. The only modification to the descriptors is remapping the pixel coordinates such that they sample in the locally rectified space.

It was found that only BRISK and ORB showed improvements with the modification. Other descriptors, like SIFT and SURF, did not show improvements. This is potentially due to BRISK and ORB being sample-based binary descriptors, so a simple modification like changing their sampling coordinates has the intended effect. SIFT and SURF, on the other hand, use more complex properties of the continuous image signal to compute descriptors, like gradients and Haar wavelet responses, so the interpolation effects introduced by rectification may be detrimental.

To analyze the improvements to BRISK and ORB, we will once again look at the AUC scores over rotation and translation baselines, as well as the descriptor-space distance distribution over changes in radial distance. Figure 5-15 shows AUC scores for BRISK and ORB with local rectification applied to fisheye cameras, with previous curves plotted for comparison. The improvements are clear for yaw motions, with the fisheye cameras nearing or equaling the performance of perspective cameras over the entire rotation. For BRISK in particular, the curves for the omnidirectional fisheye cameras are nearly flat, indicating its invariance to distortion. There are slight improvements to the performance on translation motions as well. Particularly for sideways translation, the performance of fisheye cameras is now better than or equal

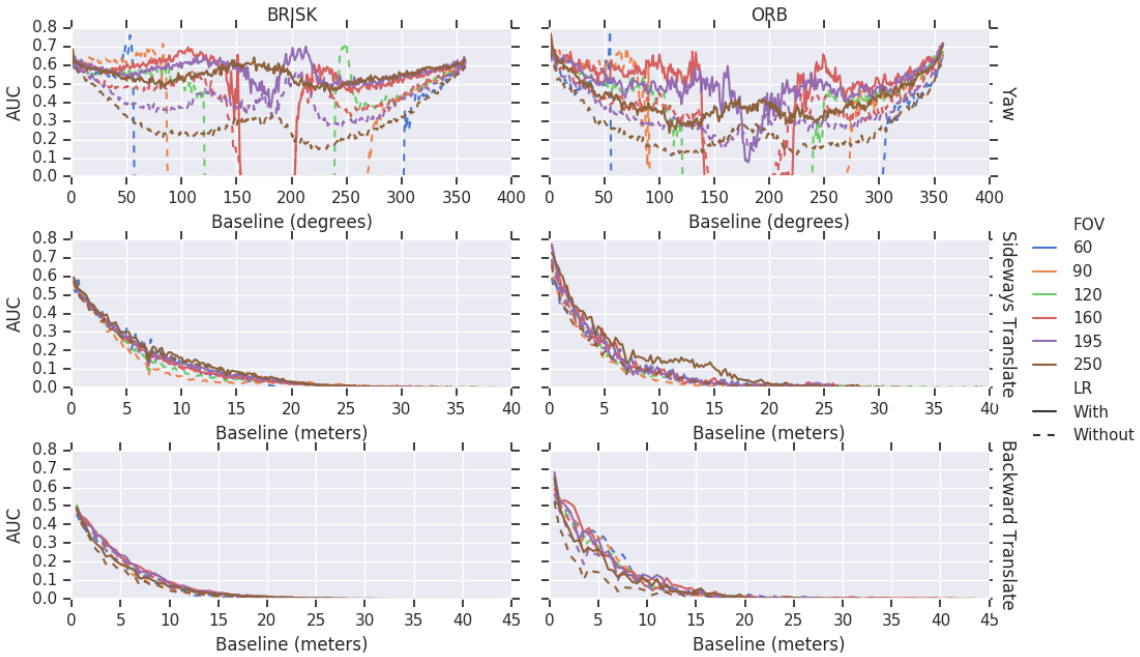


Figure 5-15: BRISK and ORB AUC scores over motion baselines for various FOVs with and without local rectification (LR)

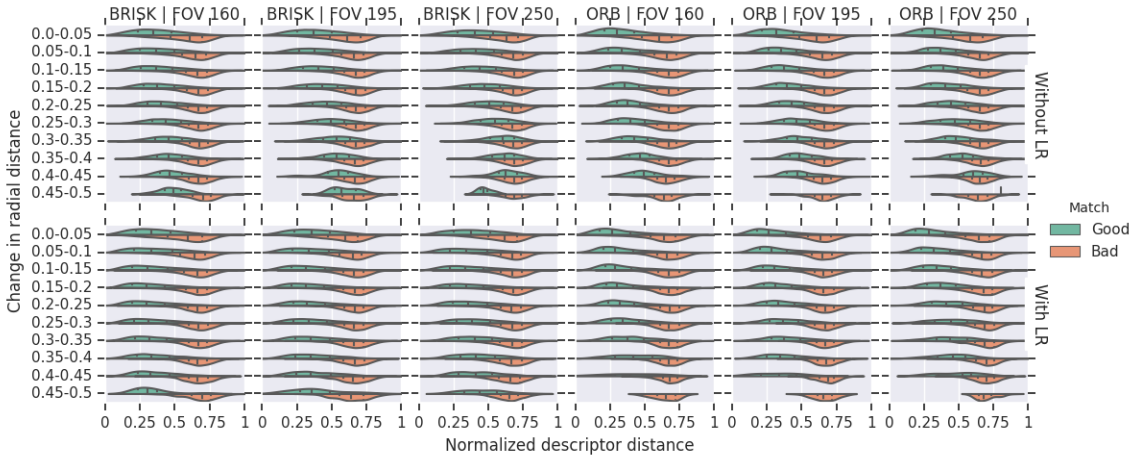


Figure 5-16: BRISK and ORB descriptor distance distributions over changes in radial distance for fisheye cameras with and without local rectification (LR)

to that of perspective cameras at all baselines. Figure 5-16 shows descriptor distance distributions over changes in radial distance with and without local rectification, and figure 5-17 shows silhouette coefficients with previous values for reference. It can be seen that both descriptors with local rectification, especially BRISK, are much more separable at all changes in radial distance, again indicating improved invariance

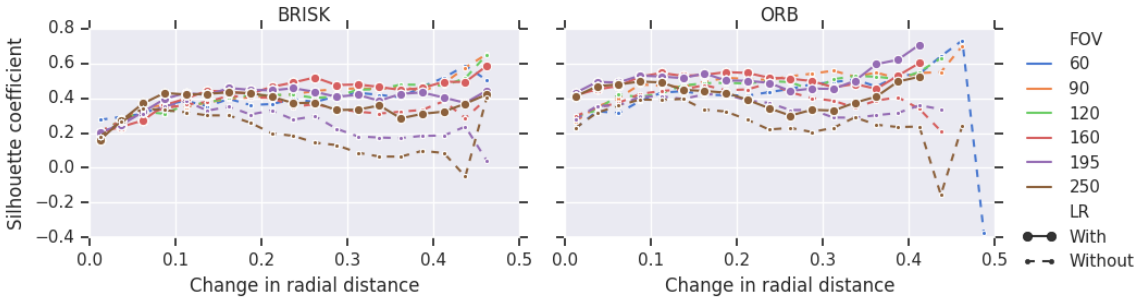


Figure 5-17: BRISK and ORB silhouette coefficients over changes in radial distance for fisheye cameras with and without local rectification (LR)

to fisheye distortion. We can thus conclude that the local rectification technique does provide significant improvements to descriptor-based feature matching on fisheye images, when using binary descriptors like BRISK and ORB.

The analysis of the feature matching task in this chapter has shown that existing feature descriptors do not handle the distortion introduced by fisheye cameras well. Matching performance is reduced more drastically for the same motion baselines, due to descriptors not being able to encode a feature similarly when it undergoes changes in distortion. To improve performance of descriptors, a novel approach was presented to locally rectify a feature patch before descriptors are computed. The local rectification method provides significant improvements to the BRISK and ORB descriptors, making them much more invariant to fisheye distortion, which suggests that novel descriptors that use a similar approach [52, 6, 86, 93] may have potential despite some of them being limited to 180 degrees FOV. Regardless of the performance of descriptors, omnidirectional fisheye cameras provide benefits for the wide-baseline matching task, by providing usable matches at greater baselines. In particular, for rotational motions, omnidirectional fisheye cameras allow for feature correspondences in all camera orientations. This is especially beneficial for the loop closure task as loops can be closed without restriction on the camera orientation. Descriptor-based matching can be used on omnidirectional fisheye cameras for the frame-to-frame feature correspondence task for odometry as well, especially with the local rectification improvement that allows fisheye cameras to be used with minimal decrease in performance.

Chapter 6

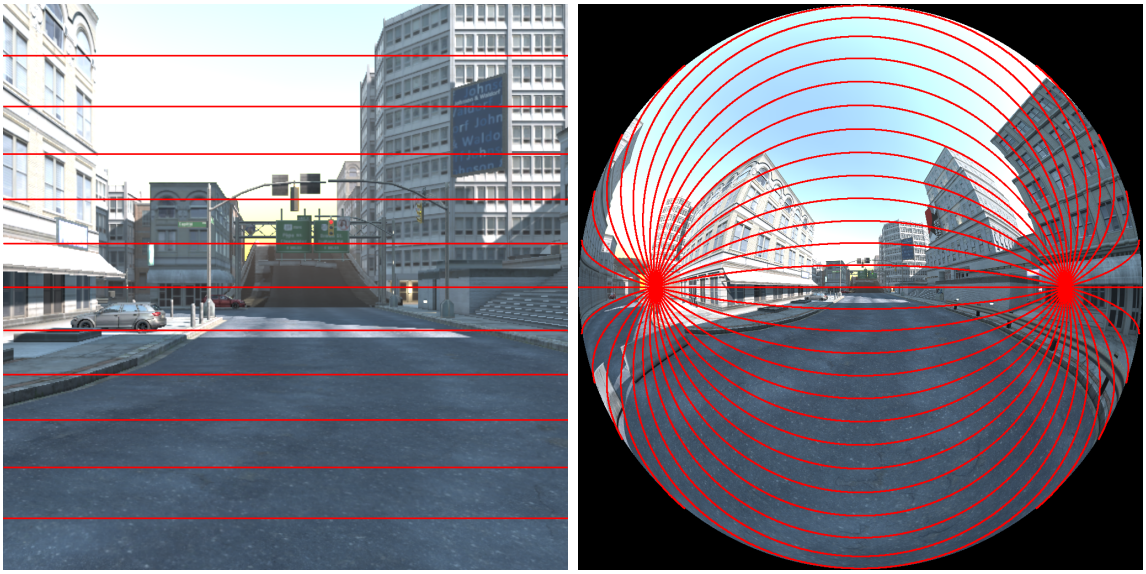
Visual SLAM Evaluation

Once feature correspondences are found, they can be used to perform the rest of the SLAM task. In this section, the odometry and reconstruction tasks will be evaluated for performance with omnidirectional fisheye cameras, and compared to performance with traditional cameras, using a custom stereo visual SLAM pipeline that supports arbitrary camera models. To allow existing algorithms to function on non-pinhole camera models, small modifications must first be made, which will also be discussed in this section. The simplicity of these modifications reinforces the point that omnidirectional fisheye cameras can be used in place of perspective cameras without significant effort.

6.1 Camera Model Adaptation

A number of the algorithms introduced in chapter 2 rely on having normalized homogeneous pixel coordinates as inputs, which we observe are simply rays with $Z = 1$. This requires assuming the rays intersect with an image plane of unit depth, meaning that all points must lie in front of the camera. For omnidirectional camera models, points that lie behind the camera cannot be converted to homogeneous coordinates. Therefore, existing pose estimation and triangulation algorithms need to be modified such that they take general rays as inputs rather than only rays with $Z = 1$.

We can first reformulate the epipolar geometry concepts to extend to arbitrary



(a) Perspective (b) Fisheye

Figure 6-1: Epipolar lines and curves for perspective and fisheye stereo cameras

camera models. Recalling the epipolar plane constraints, which originally project into pinhole models as epipolar lines, the plane can now be projected into the fisheye camera models as epipolar curves, as shown in figure 6-1. Feature correspondences between frames of known poses must now lie on these curves. However, checking a correspondence for this constraint requires parameterizing the curve in pixel space and calculating distance to the curve, which is potentially expensive. Instead, we can operate in 3D space, and check the constraints by unprojecting feature coordinates into rays. The epipolar constraint equivalent in 3D space is that the rays must lie on the epipolar plane. The dot product of each ray with a vector normal to the epipolar plane can thus be used as a distance metric. Letting $\mathbf{x_1}$ and $\mathbf{x_2}$ represent rays from a feature correspondence in two images, this effectively gives the epipolar constraint with the essential matrix $\mathbf{E}$:

$$\mathbf{x_1}^T \mathbf{E} \mathbf{x_2} = 0 \tag{6.1}$$

The value of $|\mathbf{x_1}^T \mathbf{E} \mathbf{x_2}|$ can thus be used as the error metric for checking the epipolar constraint. For calculating $\mathbf{E}$ using the eight-point or five-point algorithms, recall the reformulation of the above constraint in normalized homogeneous coordinates:

$$\mathbf{x_1} = \begin{bmatrix} u_1 & v_1 & 1 \end{bmatrix}^T \tag{6.2}$$

$$\mathbf{x_2} = \begin{bmatrix} u_2 & v_2 & 1 \end{bmatrix}^T \tag{6.3}$$

$$\mathbf{E} = \begin{bmatrix} e_{11} & e_{12} & e_{13} \\ e_{21} & e_{22} & e_{23} \\ e_{31} & e_{32} & e_{33} \end{bmatrix} \tag{6.4}$$

$$\mathbf{e} \cdot \widetilde{\mathbf{x}} = 0 \tag{6.5}$$

$$\mathbf{e} = \begin{bmatrix} e_{11} & e_{12} & e_{13} & e_{21} & e_{22} & e_{23} & e_{31} & e_{32} & e_{33} \end{bmatrix}^T \tag{6.6}$$

$$\widetilde{\mathbf{x}} = \begin{bmatrix} u_2u_1 & u_2v_1 & u_2 & v_2u_1 & v_2v_1 & v_2 & u_1 & v_1 & 1 \end{bmatrix}^T \tag{6.7}$$

If we let $\mathbf{x_1}$ and $\mathbf{x_2}$ represent general rays with coordinates $\mathbf{x_1} = \begin{bmatrix} x_1 & y_1 & z_1 \end{bmatrix}^T$ and $\mathbf{x_2} = \begin{bmatrix} x_2 & y_2 & z_2 \end{bmatrix}^T$ instead of homogeneous coordinates, $\widetilde{\mathbf{x}}$ then becomes:

$$\widetilde{\mathbf{x}} = \begin{bmatrix} x_2x_1 & x_2y_1 & x_2z_1 & y_2x_1 & y_2y_1 & y_2z_1 & z_2x_1 & z_2y_1 & z_2z_1 \end{bmatrix}^T \tag{6.8}$$

The equation $\mathbf{e} \cdot \widetilde{\mathbf{x}} = 0$ can then be written for each correspondence, and the remaining formulations of the eight-point [50] and five-point [62] algorithms follow as before.

For the PnP algorithm, the problem is already formulated to use rays as inputs. In particular, for the Lambda Twist [65] P3P implementation, although it is intended to take homogeneous coordinates as input, the algorithm immediately normalizes the homogeneous coordinate inputs into unit vectors at its first step, so any ray can be used as input.

Continuing onto triangulation algorithms, recall the derivation of the DLT algorithm for computing a 3D world point $\mathbf{X}$ from normalized homogeneous coordinates. For each of the observing cameras with pose $\mathbf{T}$ observing $\mathbf{X}$ at homogeneous coordinate $\mathbf{x}$:

$$\mathbf{TX} = w\mathbf{x} = w \begin{bmatrix} u & v & 1 \end{bmatrix}^T \tag{6.9}$$

$$\mathbf{t}_1^T\mathbf{X} = wu \tag{6.10}$$

$$\mathbf{t}_2^T\mathbf{X} = wv \tag{6.11}$$

$$\mathbf{t}_3^T\mathbf{X} = w \tag{6.12}$$

$$(u\mathbf{t}_3^T - \mathbf{t}_1^T)\mathbf{X} = 0 \tag{6.13}$$

$$(v\mathbf{t}_3^T - \mathbf{t}_2^T)\mathbf{X} = 0 \tag{6.14}$$

The last two equations are obtained by rearranging, and are the expressions minimized over all observations to find the optimal $\mathbf{X}$. If we now replace $\mathbf{x}$ with a general ray $\begin{bmatrix} x & y & z \end{bmatrix}^T$:

$$\mathbf{TX} = w\mathbf{x} = w \begin{bmatrix} x & y & z \end{bmatrix}^T \tag{6.15}$$

$$\mathbf{t}_1^T\mathbf{X} = wx \tag{6.16}$$

$$\mathbf{t}_2^T\mathbf{X} = wy \tag{6.17}$$

$$\mathbf{t}_3^T\mathbf{X} = wz \tag{6.18}$$

$$(x\mathbf{t}_3^T - z\mathbf{t}_1^T)\mathbf{X} = 0 \tag{6.19}$$

$$(y\mathbf{t}_3^T - z\mathbf{t}_2^T)\mathbf{X} = 0 \tag{6.20}$$

The last two equations can then be written in matrix form and stacked for each observation, and the triangulation can be solved as before. The other triangulation algorithm introduced, which finds midpoints of rays and is suitable for a large number of observations, already uses unit rays as inputs, so no modification is necessary.

The changes required in these algorithms, if any, are only small changes to the inputs and not major changes to the core of the algorithms themselves. As such, adapting existing SLAM pipelines to use omnidirectional fisheye cameras requires

minimal effort, and doing so provides significant benefits as will be shown in the remainder of this chapter.

6.2 Evaluation Method

To evaluate the SLAM pipeline on omnidirectional fisheye cameras, the odometry and reconstruction steps are first evaluated independently of each other. Odometry is evaluated without dependence on the map or stereo matching by using the ground truth depth maps from the simulated dataset. Reconstruction is evaluated without dependence on the pose estimate by using the ground truth poses. The full stereo SLAM pipeline is then evaluated on both simulated and real data. From the simulated dataset, only composite trajectories will be used to represent realistic situations.

For odometry, feature correspondences are extracted at every frame using KLT feature tracking. Both feature tracking and matching were found to give similar results for the downstream SLAM pipeline, so feature tracking was chosen for speed of computation. As before, when new features are detected, the ground truth depth map for the current frame and the current odometry pose estimate are used to calculate the estimated world coordinate for each feature landmark. To align trajectories, the first frame pose is initialized to the ground truth pose. Subsequent poses are then estimated using the Lambda Twist P3P [65] and RANSAC version of the PnP algorithm. Tracks are first filtered by estimating an essential matrix using the five-point RANSAC algorithm [62], and removing tracks that fail the epipolar constraint check, as described earlier. A threshold of 0.5 degrees from the epipolar plane is used. Remaining features that are tracked into the current frame are unprojected into rays and used as inputs into the PnP algorithm, as well as their 3D world coordinates calculated upon detection. RANSAC is used to iteratively select sets of four features, estimate up to four poses using the Lambda Twist P3P algorithm on three of the points, and use the fourth point to select the pose that gives the minimum reprojection error. The world coordinates of all features are then projected back into the estimated pose to calculate reprojection error, and errors that fall below

a threshold are counted as inliers at each iteration of RANSAC. The estimated pose that gives the most inliers is used as the RANSAC output. As a final refinement step, the pose is optimized over all inliers to minimize reprojection error using Ceres. After the odometry estimate is calculated, the features in each binning region are counted, as before. Because imbalanced feature distribution in the image can cause RANSAC to sample features from only one part of the image, causing instability in the pose estimate, redetection and pruning are done to ensure even distribution of features throughout the image. Feature detection is run on regions with too few features. For regions with too many features, the excessive tracks are removed in order of track lifetime, as the oldest tracks are most likely to have accumulated the most drift. The entire process described is repeated for each frame to obtain an estimated trajectory, which is then compared to the ground truth trajectory for evaluation.

For reconstruction, features are tracked and filtered as before. At each frame, each track is unprojected into a ray originating from the ground truth camera pose. For each track, all rays from its current and past frames are input into the midpoint triangulation algorithm to obtain a triangulated point for the landmark. The rays are first checked for sufficient angular coverage by computing dot products between all pairs of rays. If the minimum dot product is below a threshold, the triangulation is used as the reconstructed point; otherwise, the result is discarded because of high uncertainty. As the feature distribution is not important for this task, only redetection is performed in regions with too few remaining tracks so that a complete map can be built over long trajectories. After all frames in the trajectory are processed, bundle adjustment is performed using Ceres [2] as the midpoint triangulation algorithm does not optimize for reprojection error. As the trajectory uses the ground truth poses, only the landmark coordinates are adjusted to optimize for total reprojection error over all frames each landmark is observed in. The final map is then compared to the ground truth map for evaluation.

The full SLAM pipeline without ground truths is performed using the same approaches as the individual tasks. Instead of ground truth depth maps, a stereo pair with known baseline is used to estimate depth for each feature upon detection. The

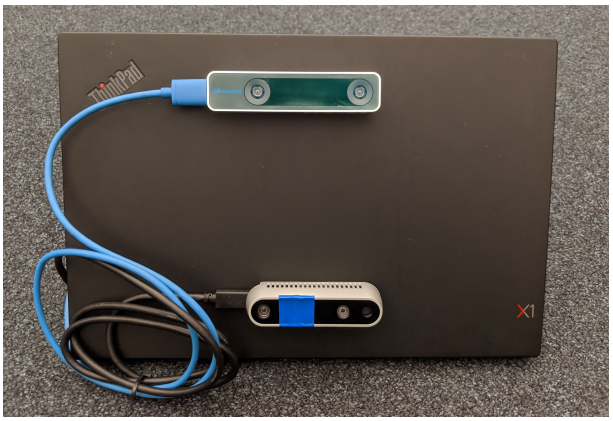


Figure 6-2: Setup used for custom RealSense dataset

Lucas-Kanade algorithm is used to match features between the cameras. An essential matrix calculated from the known stereo pair transformation is used to filter matches that fail the epipolar constraint check, using a threshold of 0.5 degrees from the epipolar plane. The DLT triangulation algorithm is then used to compute the 3D coordinates of the features in the camera frame, which are then placed into the world frame using the current estimated camera pose. For each frame, the odometry step proceeds as before to obtain a pose estimate, followed by the reconstruction step, which uses the pose estimate to triangulate remaining features that were not stereo matched to build a more complete map. At the end of the trajectory, full bundle adjustment is performed to obtain a globally consistent map and trajectory estimate.

As ground truth depth maps and trajectories are no longer required, real datasets can be used to evaluate the system. The TUM omnidirectional dataset [75], which contains images from a stereo pair with 185 degree FOV cameras and ground truth poses for evaluation, is used to evaluate the feasibility of using fisheye cameras for SLAM. Only the indoor trajectories are used as those are the only ones that contain ground truths. In addition, a calibration dataset is provided, allowing a double sphere model to be calibrated. For comparing fisheye and traditional cameras, a custom dataset was collected using an Intel RealSense T265 tracking camera, which has a stereo pair with 170 degrees FOV, and an Intel RealSense D435 stereo camera, with 80 degrees FOV, mounted together as shown in figure 6-2. The T265 also performs its own odometry which can be used as an approximate ground truth for reference.

In addition, the trajectories start and end at the same location, which allows the drift to be compared between the two cameras without requiring a ground truth trajectory. To validate the comparison in different environments, both indoor and outdoor trajectories are included.

6.3 Evaluation Metrics

To evaluate trajectories, existing metrics for comparing trajectories [78] will be used. The first metric is the *absolute pose error* (APE), which evaluates the global quality of the trajectory by comparing absolute distances between between each pose in the estimated and ground truth trajectories. Given a ground truth pose matrix $\mathbf{G}_i$ and estimated pose matrix $\mathbf{P}_i$ at time step i, the absolute pose error can be computed as:

$$\mathbf{E}_{\mathbf{a}i} = \mathbf{G}_i^{-1}\mathbf{P}_i \tag{6.21}$$

For evaluating trajectories, comparing the translational component is sufficient as rotational errors will also reflect as translational errors [78]. As such, a meaningful statistic for evaluating APE over a trajectory of n poses is defined as the root mean squared error over the translational components $\mathbf{t}_{\mathbf{E}_\mathbf{a}}$ of the absolute pose errors:

$$RMSE(\mathbf{E}_{\mathbf{a}1:n}) = \sqrt{\frac{1}{n}\sum_{i=1}^{n} \|\mathbf{t}_{\mathbf{E}_{\mathbf{a}i}}\|^2} \tag{6.22}$$

As the visual odometry task is prone to drift, it is also meaningful to evaluate incremental pose estimation errors between frames. The *relative pose error* (RPE) metric measures the local accuracy of a trajectory between two time intervals. Given ground truth pose matrices $\mathbf{G}_i$ and $\mathbf{G}_{i+1}$ and estimated pose matrices $\mathbf{P}_i$ and $\mathbf{P}_{i+1}$ at time steps i and $i+1$, the RPE is defined as:

$$\mathbf{E}_{\mathbf{r}i} = (\mathbf{G}_i^{-1}\mathbf{G}_{i+1})^{-1}(\mathbf{P}_i^{-1}\mathbf{P}_{i+1}) \tag{6.23}$$

The RPE over a trajectory can again be quantified using the root mean squared

error over the translational components $\mathbf{t}_{\mathbf{E_r}}$ of the relative pose errors:

$$RMSE(\mathbf{E}_{\mathbf{r}1:n}) = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n-1} \|\mathbf{t}_{\mathbf{E}_{\mathbf{r}_i}}\|^2} \tag{6.24}$$

For evaluating the reconstructed map, the Euclidean distance between an estimated feature world coordinate and its ground truth will be used as an accuracy metric. For stereo matching evaluation, the disparity error is typically used instead of depth error to normalize for the camera geometry and higher depth uncertainty at large depths. However, in fisheye cameras, the uneven angular resolution causes the disparity error metric to have less meaning because it maps to different depth errors in different parts of the image. As such, the *normalized depth error* will be used to evaluate stereo matching, where d_{gnd} is the ground truth depth:

$$\Delta d_{norm} = \frac{\Delta d}{d_{gnd}^2} \tag{6.25}$$

6.4 Evaluation Results

The components of the SLAM pipeline will be evaluated in order of their sequence in the pipeline, starting with stereo matching, followed by odometry and reconstruction.

6.4.1 Stereo Matching

Because stereo matching uses the Lucas-Kanade optical flow algorithm, which has already been evaluated in depth in chapter 4, it will only be evaluated briefly in the context of depth estimation in this section. This evaluation will also verify the performance of the DLT algorithm adapted to fisheye cameras. Depth errors will be evaluated over cameras with various FOVs moving through composite trajectories.

Figure 6-3 shows normalized depth errors over radial distances for various FOVs. Because most of the errors are very small but there are larger outliers, the plot uses logarithmic scale. There are slight increases in error for fisheye cameras at larger radial distances, but the magnitude of the errors is still negligible. In addition, there

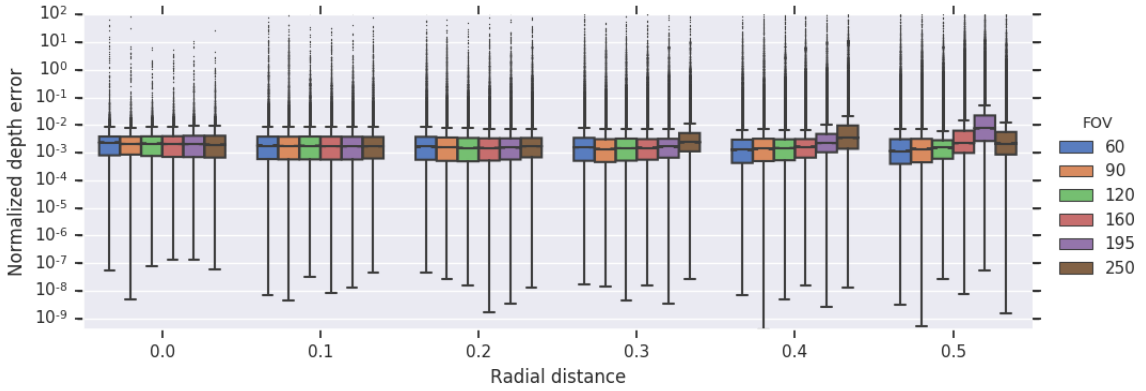


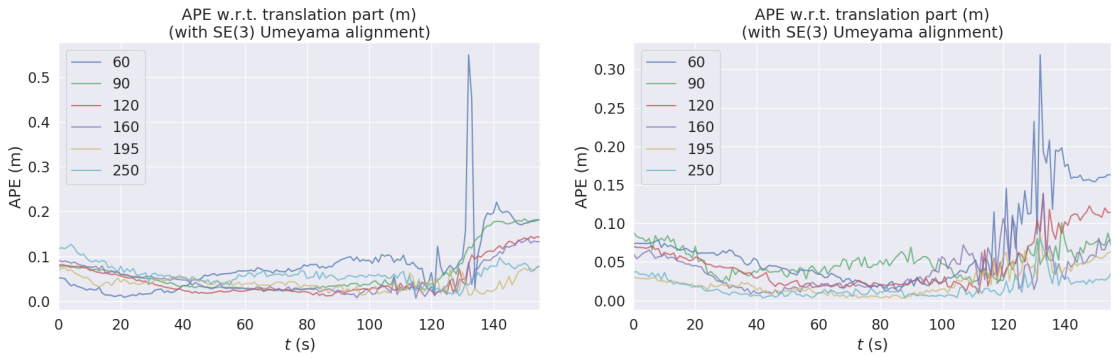
Figure 6-3: Normalized stereo depth errors for different radial distances

are more outliers at higher radial distances for all cameras. These results show that using omnidirectional fisheye cameras does not have a significant impact on stereo depth estimation performance, so it can be relied on by the rest of the SLAM pipeline.

6.4.2 Odometry

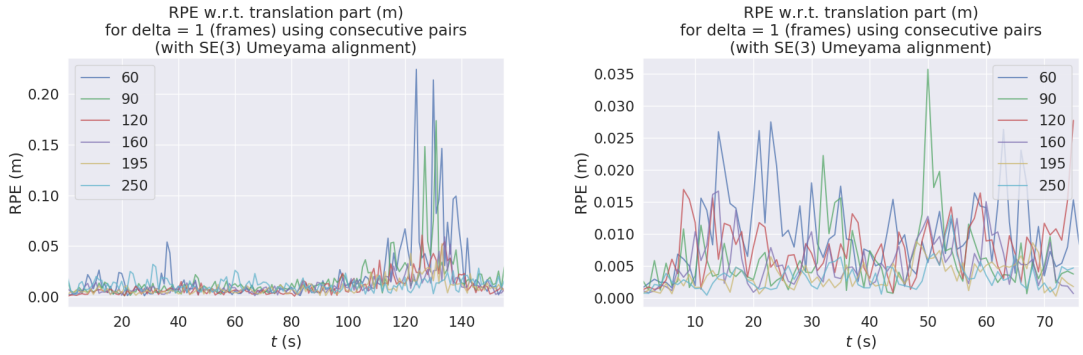
Although odometry was first evaluated using ground truth depth maps, it was found that the odometry performance from the full SLAM pipeline using stereo depth estimates was very similar due to the accuracy of stereo matching. Therefore, the odometry evaluation will directly use the results from the full pipeline.

Trajectories from the simulated urban environment will be used to compare performance across FOVs. The trajectories involve rotations throughout the motion, which causes the cameras to be pointed towards low-texture surfaces during certain periods. As expected, narrow FOV cameras, in particular less than 90 degrees, fail immediately whenever the camera faces a low-texture wall, ground, or sky, or whenever the motion is too fast, due to KLT losing all tracks and PnP not having enough correspondences to estimate a pose. It was found that the higher the motion rate for a given trajectory, the higher the required FOV to prevent odometry failures, which reflects the KLT rate study from chapter 4. This makes it clear that narrow FOV cameras cannot be used for navigation if robustness is desired. For the purpose of analysis, we will select trajectories in which all FOVs do not fail, so that the accuracy can be analyzed. Two such trajectories will be evaluated in depth: one with the cam-



(a) Trajectory 1 (urban canyon) (b) Trajectory 2 (enclosed)

Figure 6-4: APE over trajectory length for two trajectories



(a) Trajectory 1 (urban canyon) (b) Trajectory 2 (enclosed)

Figure 6-5: RPE over trajectory length for two trajectories

era moving along roads in an urban canyon, facing various directions, and one with the camera moving in a circular motion in an enclosed area surrounded by buildings.

Figures 6-4 and 6-5 show the APE and RPE respectively over the two trajectories for various FOVs, with Umeyama alignment [83] which aligns the trajectories with the ground truth to minimize error. Tables 6.1 and 6.2 show statistics for these metrics, for both translation and rotation parts. It can be seen that even when the odometry does not fail, the errors for narrow FOV cameras are higher in certain parts of the trajectory and much noisier, as reflected by the higher standard deviations and the spikes in the plots. In particular, in both trajectories, the largest sources of error for narrow FOVs are from parts of the trajectories where the camera is facing a surface and moving towards or along it, due to the lower angular coverage of features in

Trajectory	FOV	APE (translation / rotation)			
		Mean	SD	RMSE	Max
Trajectory 1 (urban canyon)	60	0.0815m	0.0697m	0.107m	0.549m
		0.298°	0.152°	0.335°	0.832°
	90	0.0625m	0.0483m	0.079m	0.184m
		0.3°	0.0748°	0.31°	0.54°
	120	0.048m	0.0363m	0.0602m	0.145m
		0.252°	0.0623°	0.26°	0.372°
	160	0.0463m	0.0327m	0.0558m	0.14m
		0.174°	0.057°	0.183°	0.261°
	195	0.0403m	0.0148m	0.043m	0.0775m
		0.155°	0.0427°	0.161°	0.286°
	250	0.0506m	0.0216m	0.0534m	0.127m
		0.236°	0.0586°	0.243°	0.346°
Trajectory 2 (enclosed)	60	0.0686m	0.0548m	0.0878m	0.319m
		0.173°	0.0633°	0.184°	0.328°
	90	0.0506m	0.0312m	0.0538m	0.101m
		0.162°	0.0568°	0.171°	0.395°
	120	0.0463m	0.0237m	0.0558m	0.138m
		0.146°	0.0436°	0.15°	0.293°
	160	0.038m	0.0169m	0.0448m	0.139m
		0.12°	0.0355°	0.128°	0.241°
	195	0.024m	0.0171m	0.0295m	0.0883m
		0.0898°	0.0288°	0.0933°	0.211°
	250	0.0167m	0.0128m	0.021m	0.08m
		0.0876°	0.0253°	0.0922°	0.152°

Table 6.1: APE statistics

narrow FOV cameras causing instability in the pose estimation algorithms. This supports the motion analysis from previous chapters, which again proves that wider FOVs are needed for robustness against various motion types.

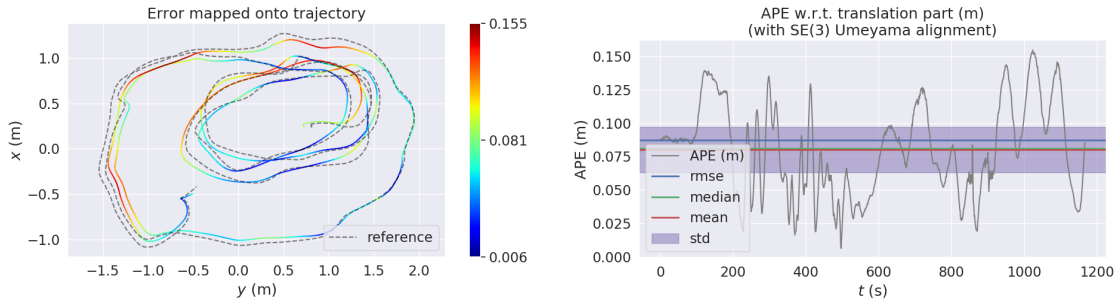
While narrow FOV cameras consistently perform worse, wider FOVs greater than 90 degrees perform much more similarly across the trajectories, especially in parts of the trajectories where the camera moves along an urban canyon. It was found that optimal performance is usually with fisheye lenses near 180 degrees FOV. Extremely large FOVs, such as the 250 degree FOV analyzed, sometimes perform slightly worse than the 160 and 195 degree FOV cameras in these situations. This is evidenced in trajectory 1 which contains motions through urban canyons, where the performance

Trajectory	FOV	RPE (translation / rotation)			
		Mean	SD	RMSE	Max
Trajectory 1 (urban canyon)	60	0.0208m	0.0325m	0.0385m	0.224m
		0.0504°	0.0431°	0.0663°	0.354°
	90	0.0163m	0.0202m	0.026m	0.174m
		0.0372°	0.0465°	0.0595°	0.214°
	120	0.0107m	0.0107m	0.015m	0.0607m
		0.0443°	0.025°	0.0508°	0.161°
	160	0.0106m	0.0084m	0.0136m	0.0591m
		0.029°	0.0398°	0.0493°	0.32°
	195	0.0092m	0.0074m	0.0119m	0.0523m
		0.0252°	0.0275°	0.0372°	0.156°
	250	0.012m	0.0061m	0.0134m	0.0321m
		0.0354°	0.0353°	0.05°	0.2°
Trajectory 2 (enclosed)	60	0.0102m	0.0062m	0.0119m	0.0275m
		0.0365°	0.0376°	0.0477°	0.23°
	90	0.0078m	0.0055m	0.0091m	0.0357m
		0.0326°	0.0255°	0.0418°	0.132°
	120	0.0061m	0.0047m	0.0083m	0.0277m
		0.0294°	0.0205°	0.0413°	0.104°
	160	0.0056m	0.0038m	0.0067m	0.0167m
		0.0224°	0.0186°	0.0291°	0.103°
	195	0.0035m	0.0021m	0.004m	0.0129m
		0.0153°	0.0119°	0.0194°	0.063°
	250	0.0033m	0.002m	0.0039m	0.0088m
		0.0144°	0.0101°	0.0176°	0.053°

Table 6.2: RPE statistics

for the 250 degree FOV is marginally worse than the other fisheye cameras. This is consistent with previously shown results [94] which concluded that FOVs beyond 210 degrees stop providing gains in performance, as well as the motion analyses from previous chapters regarding urban canyon motions. However, for indoor or enclosed environments with potential large motions, such as trajectory 2, performance still monotonically increases with FOV because of robustness from angular coverage. Regardless of the environment, fisheye cameras still provide more robustness and accuracy than perspective cameras; the only factor in which environment plays a part in is the extremity of the FOV.

To verify that the conclusions drawn from the analysis on simulated data carry



(a) Trajectory

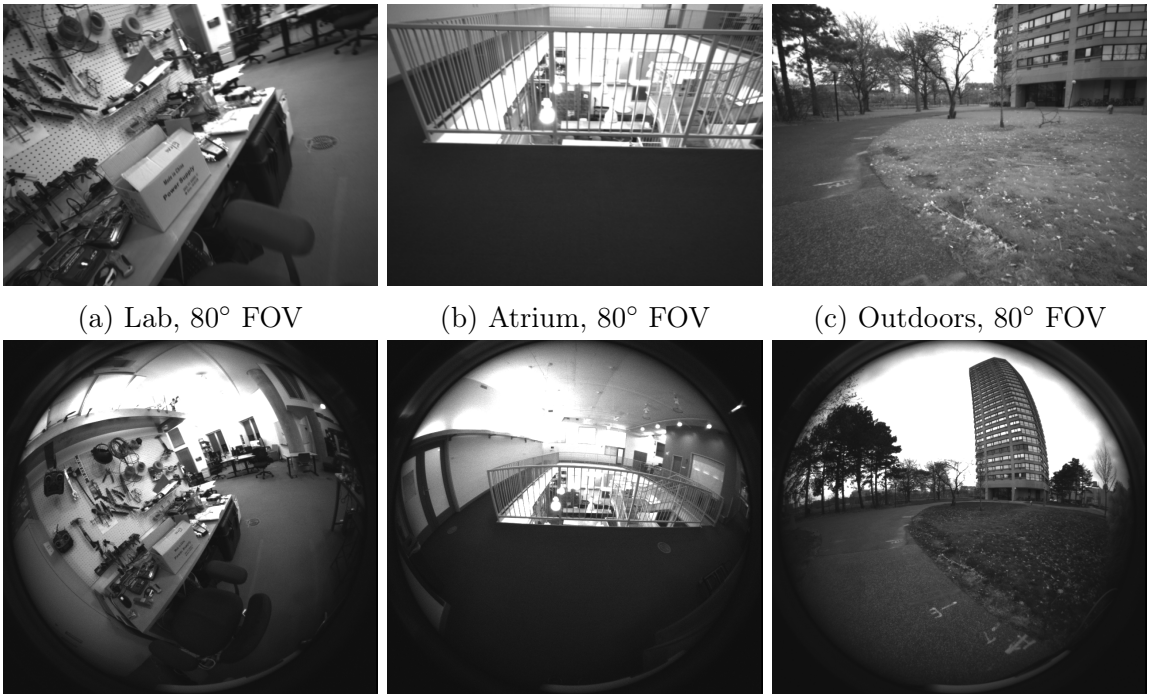
(b) Absolute pose error

Figure 6-6: TUM trajectory results

over into the real world, the stereo SLAM pipeline will first be run on a dataset from TUM [75] that contains indoor sequences from a stereo pair with 185 degrees FOV to validate the SLAM implementation. Figure 6-6 shows a trajectory from the dataset with the resulting trajectory estimate, as well as a plot of APE through the trajectory. The APE RMSE is 0.0872m. The dataset paper reports the highest performance on this trajectory as an RMSE of 0.012m by OKVIS [46]. Although this is lower than the RMSE of the custom SLAM pipeline, OKVIS is a state-of-the-art system so having an error on the same order of magnitude is sufficient to verify functionality of the custom pipeline.

To verify that the comparisons between FOVs based on simulated data carry over to the real world, the SLAM pipeline will now be evaluated on the custom dataset containing image sequences of a 170 degree FOV stereo pair and an 80 degree FOV stereo pair moving together through various trajectories, both indoors and outdoors. Sample images from the dataset are shown in figure 6-7. An approximate ground truth is also provided by the tracking module in one of the Intel RealSense cameras. The trajectories start and end at the same location so the difference between the endpoints of the estimated trajectories is used as a performance measure.

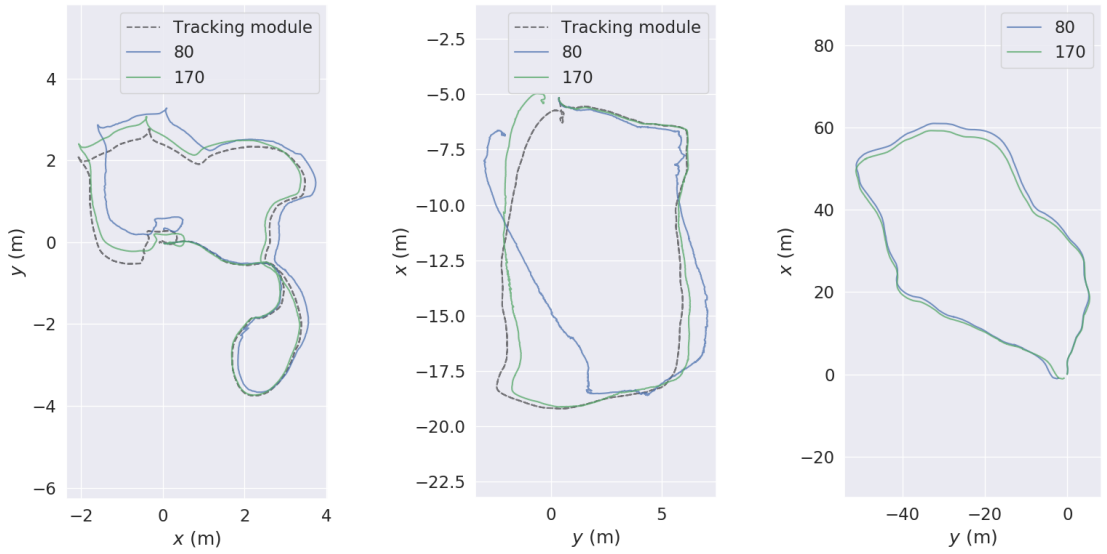
Figure 6-8 shows some of the trajectories in the dataset, with the odometry from the narrow and wide cameras as well as the approximate ground truth. It is immediately clear that the wide FOV camera performs better in all trajectories, especially in indoor environments where textureless walls are common. For quantitative compari-



(a) Lab, 80° FOV (b) Atrium, 80° FOV (c) Outdoors, 80° FOV

(d) Lab, 170° FOV (e) Atrium, 170° FOV (f) Outdoors, 170° FOV

Figure 6-7: Images from the custom RealSense dataset



(a) Lab (b) Atrium (c) Outdoors

Figure 6-8: Custom RealSense dataset trajectories

son, the dataset lacks accurate ground truth so APE and RPE will not be analyzed. Instead, table 6.3 shows the distance between start and end points of the odometry

Trajectory	FOV	Endpoint distance
Lab	80	0.4m
	170	0.117m
Atrium	80	3.841m
	170	0.755m
Outdoors	80	7.569m
	170	2.924m

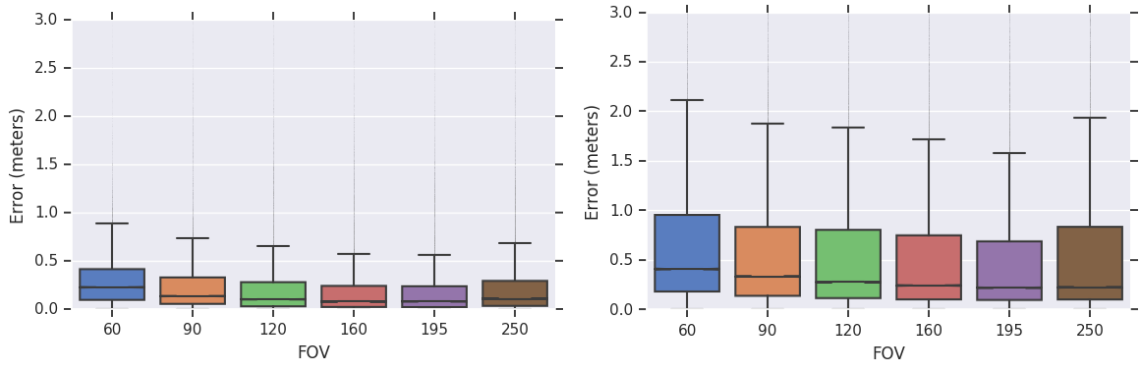
Table 6.3: Distance between trajectory endpoints for custom RealSense dataset

estimates. For all trajectories, the differences are much larger for the narrow FOV camera, indicating more drift or unstable incremental pose estimates throughout.

6.4.3 Reconstruction

The final component of the pipeline to be evaluated is the quality of the reconstructed map. The main benefit provided by omnidirectional fisheye cameras is the area coverage of the map after the camera moves along a trajectory. Narrow FOV cameras are focused on one part of a scene so the coverage of the map will naturally be limited, although the higher resolution will produce a denser map. For the navigation task, however, being able to map out a larger portion of the environment with less motion is more preferable, for potential reasons like providing motion planning with more information so it can plan further ahead or make alternative decisions.

To verify this hypothesis, as well as determine how the use of fisheye cameras affects the accuracy of the maps, the triangulation algorithms are first evaluated using ground truth poses, to isolate the analysis from the errors introduced by the odometry estimates. Then, the reconstructed maps from the full SLAM pipeline are also evaluated. The same simulated trajectories are used, with the ground truth depth maps and poses providing ground truth coordinates for each landmark. Figure 6-9 shows Euclidean distance error distributions for various FOVs over all composite trajectories, for both ground truth and estimated odometry. The errors using ground truth poses are about half the magnitude of the errors using estimated odometry, and are marginally higher for smaller FOVs. The trends are similar with the full SLAM



(a) Ground truth odometry (b) Estimated odometry

Figure 6-9: Euclidean distance error distributions of reconstructed points for various FOVs

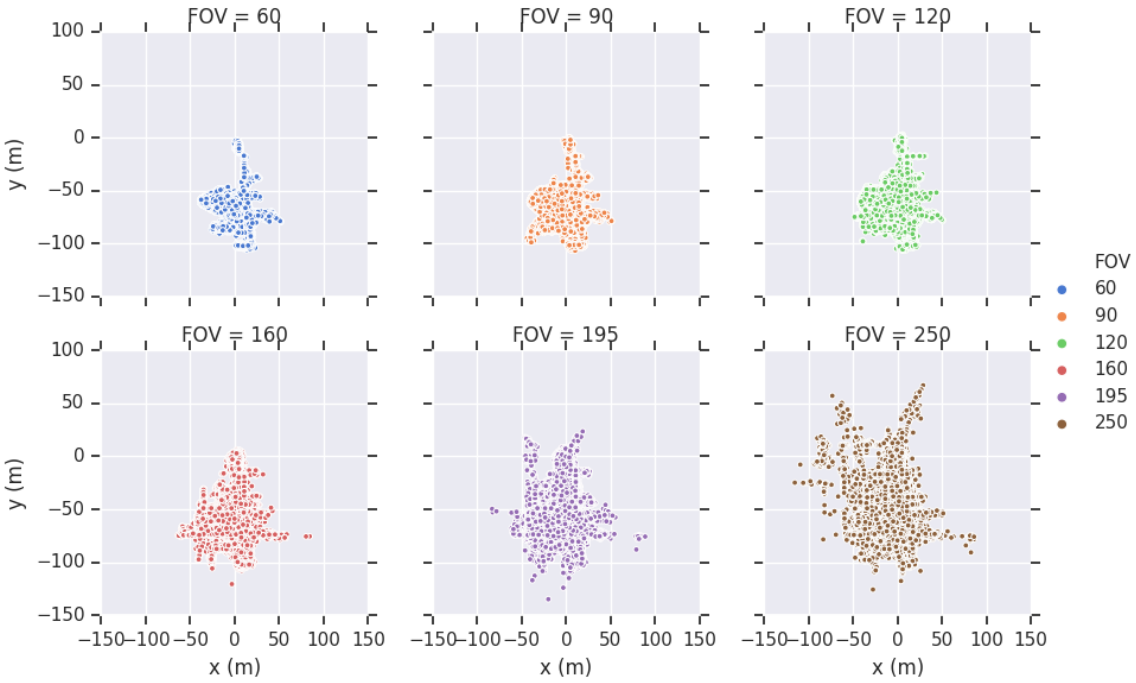


Figure 6-10: Area distributions of reconstructed maps

pipeline using estimated odometry, with fisheye cameras also performing slightly better than perspective cameras, due to the increased accuracy of odometry estimates with larger FOV cameras.

To show that omnidirectional fisheye cameras provide accurate maps with more area coverage, the distribution of coordinates of correctly mapped points (using an error threshold of 10cm) is shown in figure 6-10. The range of space covered is also

FOV	X range	Y range	Z range
60	90.17m	103.54m	50.42m
90	95.39m	108.63m	56.98m
120	99.11m	122.12m	58.47m
160	158.42m	130m	74.1m
195	174.2m	159.24m	80.38m
250	214.16m	192.9m	88.52m

Table 6.4: Range of coordinates mapped for various FOVs

quantified in table 6.4 which shows the distance between map extremes in the three dimensions. It is clear that the wider the FOV, the more area the reconstructed map covers for the same trajectory. This result therefore verifies one additional benefit of using omnidirectional fisheye cameras for navigation, which is the increased area of a scene that can be mapped with the same camera motion.

The evaluation of the custom stereo SLAM pipeline on both simulated and real datasets shows that the wide FOV from omnidirectional fisheye cameras provides benefits for odometry largely in robustness and also in accuracy, especially in indoor environments where textureless walls are common. Even when narrow FOV cameras do not fail, fisheye cameras still provide higher performance by allowing for more stable odometry and therefore smoother, more accurate estimates. This is especially true in indoor environments, where performance benefits greatly from increased angular coverage of features, and increases monotonically with FOV. This mostly carries over to outdoor scenarios as well, except for translation down the urban canyon environment where extremely large FOVs start to lose their benefits due to lower angular resolution and increased distortion. Despite this, omnidirectional fisheye cameras with slightly more than 180 degrees FOV still provide higher performance than perspective cameras in these scenarios. In addition, omnidirectional fisheye cameras provide benefits to the reconstruction task as well, being able to map a larger area with less camera motion, which is desirable for navigation tasks.

Chapter 7

Conclusions

The results presented in this work show that omnidirectional fisheye cameras provide benefits to all parts of the SLAM pipeline. Starting with the KLT feature tracking algorithm, its performance on fisheye cameras is only marginally worse due to the radial distortion. However, these drawbacks are outweighed by the benefits, which are the increased track lengths and tolerance to high motion rates or low frame rates. With the exception of moving down an urban canyon along the camera's optical axis, the length of feature tracks before they fail or leave the image monotonically increases with the FOV used. Furthermore, the increased FOV also allows more features to be tracked at higher motion rates because they are able to stay in view between frames. While these results do not apply to moving down an urban canyon, where fisheye cameras perform slightly worse with speed due to distortion, the rate at which pixels move in an image scales much more with rotational and sideways translation motions, so in practicality the gains provided by wide-angle cameras are still very relevant.

Continuing onto the descriptor-based feature matching task, the results conclude that existing feature descriptors readily available in libraries like OpenCV do not handle the extreme radial distortion of fisheye cameras well, as matching performance for all descriptors decreases as a function of how much a feature moves radially in the image. Using the proposed local rectification method largely solves this by allowing binary descriptors to sample in a rectified pattern, making them more invariant to distortion. Regardless of descriptor performance, using omnidirectional fisheye cam-

eras allows features to be matched in all camera orientations, greatly benefiting the wide-baseline matching task needed by tasks like loop closure. Analyzing matching performance as a function of how much a feature physically moves with respect to the angle from the camera's optical axis shows that omnidirectional fisheye cameras allow features to be matched over a wider range of motion baselines.

Finally, the performance of omnidirectional fisheye cameras on the odometry and reconstruction tasks was evaluated using a custom stereo visual SLAM framework. Fisheye cameras not only provide large gains in robustness, being able to handle low-texture images or high motion rates where narrow FOV cameras consistently fail, but also provide accuracy improvements due to the increased stability introduced by greater angular coverage of features. This increase in robustness and accuracy is particularly true for indoor environments, where textureless surfaces are common. Robustness and accuracy in these environments increases monotonically as a function of FOV. For outdoor environments, fisheye cameras provide the same benefits, but FOVs much larger than 180 degrees stop providing gains for translational motions down urban canyons, where the lower angular resolution and increased distortion start to dominate. Therefore, while higher FOVs are strongly preferred for indoor navigation, these results combined with the results from feature tracking show that applications exclusively in urban canyons like self-driving cars are less in critical need of omnidirectional cameras, although they will still benefit from using fisheye cameras up to around 200 degrees of FOV. In any other scenario where the camera could be rotating quickly or facing a surface, such as for MAVs, as large of an FOV as possible is preferred. Finally, the reconstruction evaluation results show that for the same amount of motion, omnidirectional fisheye cameras allow a larger area of the environment to be mapped with no loss of accuracy, only density due to angular resolution. Therefore, unless the application requires detailed mapping of a small area, omnidirectional fisheye cameras are preferred for the navigation task. All of the aforementioned benefits, combined with the ease of adaptation of existing algorithms to use fisheye cameras, show that these cameras should be preferred for general computer vision tasks that require robustness in the real world.

Bibliography

[1] Y.I Abdel-aziz and H.M. Karara. Direct linear transformation from comparator to object space coordinates in close-range photogrammetry. *Photogrammetric Engineering & Remote Sensing*, 81(1):103–107, 2015.

[2] Sameer Agarwal and Keir Mierle. Ceres solver. `http://ceres-solver.org`.

[3] Alexandre Alahi, Raphael Ortiz, and Pierre Vandergheynst. FREAK: Fast retina keypoint. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '12, pages 510–517, Washington, DC, USA, 2012. IEEE Computer Society.

[4] Pablo Fernández Alcantarilla, Adrien Bartoli, and Andrew J. Davison. KAZE features. In *Proceedings of the 12th European Conference on Computer Vision - Part VI*, ECCV '12, pages 214–227, Berlin, Heidelberg, 2012. Springer-Verlag.

[5] Yucel Altunbasak, Russell M. Mersereau, and Andrew J. Patti. A fast parametric motion estimation algorithm with illumination and lens distortion correction. *IEEE Transactions on Image Processing*, 12(4):395–408, 2003.

[6] Zafer Arican and Pascal Frossard. OmniSIFT: Scale invariant features in omnidirectional images. In *2010 IEEE International Conference on Image Processing*, pages 3505–3508, 2010.

[7] Simon Baker, Daniel Scharstein, J. P. Lewis, Stefan Roth, Michael J. Black, and Richard Szeliski. A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1):1–31, 2011.

[8] John Barron, David Fleet, and Steven Beauchemin. Performance of optical flow techniques. *International Journal of Computer Vision*, 12:43–77, 1994.

[9] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (SURF). *Comput. Vis. Image Underst.*, 110(3):346–359, 2008.

[10] H. E. Benseddik, H. Hadj-Abdelkader, B. Cherki, and O. Djekoune. Binary feature descriptor for omnidirectional images processing. In *Proceedings of the 2015 International Conference on Intelligent Information Processing, Security and Advanced Communication*, IPAC '15, pages 81:1–81:6, New York, NY, USA, 2015. ACM.

[11] Jiawang Bian, Ruihan Yang, Yun Liu, Le Zhang, Ming-Ming Cheng, Ian D. Reid, and WenHai Wu. MatchBench: An evaluation of feature matchers. *ArXiv*, abs/1808.02267, 2018.

[12] Jean-Yves Bouguet. Pyramidal implementation of the Lucas Kanade feature tracker. Technical report, Intel Corporation, Microprocessor Research Labs, 1999.

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

[14] Michael Burri, Janosch Nikolic, Pascal Gohl, Thomas Schneider, Jörn Rehder, Sammy Omari, Markus W Achtelik, and Roland Siegwart. The EuRoC micro aerial vehicle datasets. *The International Journal of Robotics Research*, 35:1157–1163, 2016.

[15] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. BRIEF: Binary robust independent elementary features. In *Proceedings of the 11th European Conference on Computer Vision - Part IV*, ECCV '10, pages 778–792, Berlin, Heidelberg, 2010. Springer-Verlag.

[16] David Caruso, Jakob Engel, and Daniel Cremers. Large-scale direct SLAM for omnidirectional cameras. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 141–148, 2015.

[17] Peter I. Corke, Dennis Strelow, and Sanjiv Singh. Omnidirectional visual odometry for a planetary rover. *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 4:4007–4012, 2004.

[18] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1 of *CVPR '05*, pages 886–893, 2005.

[19] Frank Dellaert. Factor graphs and GTSAM: A hands-on introduction. Technical Report GTRIM-CP&R-2012-002, Georgia Institute of Technology, 2012.

[20] Jakob Engel, Vladlen Koltun, and Daniel Cremers. Direct sparse odometry. *IEEE Trans. Pattern Anal. Mach. Intell.*, 40(3):611–625, 2018.

[21] Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *European Conference on Computer Vision (ECCV)*, volume 8690, pages 1–16, 2014.

[22] S. Finsterwalder and W. Scheufele. Das rückwärtseinschneiden im raum. *Sebastian Finsterwalder zum 75. Geburtstage*, pages 86–100, 1937.

[23] Martin A. Fischler and Robert C. Bolles. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Graphics and Image Processing*, 24(6):381–395, 1981.

[24] Ben Galvin, Brendan McCane, Kevin L. Novins, David Mason, and Steven Mills. Recovering motion fields: An evaluation of eight optical flow algorithms. In *Proceedings of the British Machine Vision Conference*, pages 20.1–20.10. BMVA Press, 1998.

[25] Xiao-Shan Gao, Xiao-Rong Hou, Jianliang Tang, and Hang-Fei Cheng. Complete solution classification for the perspective-three-point problem. *IEEE Trans. Pattern Anal. Mach. Intell.*, 25(8):930–943, 2003.

[26] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: the KITTI dataset. *The International Journal of Robotics Research*, 32:1231–1237, 2013.

[27] Christopher Geyer and Konstantinos Daniilidis. A unifying theory for central panoramic systems and practical applications. In *Proceedings of the 6th European Conference on Computer Vision - Part II*, ECCV '00, pages 445–461, London, UK, UK, 2000. Springer-Verlag.

[28] E.W. Grafarend, P. Lohse, and B. Schaffarin. Dreidimensionaler ruckwartsschnitt, teil I: die projektiven gleichungen. *Zeitschrift für vermessungswesen. Stuttgart: Geodätisches Institut, Universität*, pages 172–175, 1989.

[29] W. Nicholas Greene. Real-time dense simultaneous localization and mapping using monocular cameras. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, USA, 2016.

[30] J.A. Grunert. Das pothenotische problem in erweiterter gestalt nebst über seine anwendungen in der geodäsie. *Grunerts Archiv für Mathematik und Physik*, 1:238–248, 1841.

[31] Michael Grupp. evo: Python package for the evaluation of odometry and SLAM. `https://github.com/MichaelGrupp/evo`, 2017.

[32] Hao Guan and William A. P. Smith. BRISKS: Binary features for spherical images on a geodesic grid. In *2017 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '17, pages 4516–4524, 2017.

[33] Peter Hansen, Wageeh Boles, and Peter Corke. Spherical diffusion for scale-invariant keypoint detection in wide-angle images. In *2008 Digital Image Computing: Techniques and Applications*, pages 525–532, 2008.

[34] Robert M. Haralick, Chung-Nan Lee, Kars Ottenburg, and Michael Nölle. Analysis and solutions of the three point perspective pose estimation problem. *Proceedings of the 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 592–598, 1991.

[35] Chris Harris and Mike Stephens. A combined corner and edge detector. In *In Proc. of Fourth Alvey Vision Conference*, pages 147–151, 1988.

[36] Richard I. Hartley. An investigation of the essential matrix. Technical report, GE-CRD, Schenectady, NY, USA, 1993.

[37] Richard I. Hartley. Chirality. *International Journal of Computer Vision*, 26(1):41–61, 1998.

[38] Michael Kaess, Ananth Ranganathan, and Frank Dellaert. iSAM: Incremental smoothing and mapping. *IEEE Transactions on Robotics*, 24(6):1365–1378, 2008.

[39] Bogdan Khomutenko, Gaëtan Garcia, and Philippe Martinet. An enhanced unified camera model. *IEEE Robotics and Automation Letters*, 1:137–144, 2016.

[40] Laurent Kneip, Davide Scaramuzza, and Roland Siegwart. A novel parametrization of the perspective-three-point problem for a direct computation of absolute camera position and orientation. In *Proceedings of the 2011 IEEE Conference on Computer Vision and Pattern Recognition*, CVPR '11, pages 2969–2976, Washington, DC, USA, 2011. IEEE Computer Society.

[41] Timo Kropp. Matching omnidirectional images in indoor environments. Master's thesis, Technische Universität Wien, Vienna, Germany, 2013.

[42] Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. Factor graphs and the sum-product algorithm. *IEEE Transactions on Information Theory*, 47(2):498–519, 2001.

[43] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g2o: A general framework for graph optimization. In *Proceedings of the 2011 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3607–3613, 2011.

[44] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. EPnP: An accurate O(N) solution to the PnP problem. *International Journal of Computer Vision*, 81(2):155–166, 2009.

[45] Stefan Leutenegger, Margarita Chli, and Roland Y. Siegwart. BRISK: Binary robust invariant scalable keypoints. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2548–2555, Washington, DC, USA, 2011. IEEE Computer Society.

[46] Stefan Leutenegger, Simon Lynen, Michael Bosse, Roland Siegwart, and Paul Furgale. Keyframe-based visual-inertial odometry using nonlinear optimization. *The International Journal of Robotics Research*, 34(3):314–334, 2015.

[47] Gil Levi and Tal Hassner. LATCH: Learned arrangements of three patch codes. *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–9, 2015.

[48] Na Liu, Baofeng Zhang, Yingkui Jiao, and Junchao Zhu. Feature matching method for uncorrected fisheye lens image. In Liandong Yu, editor, *Seventh International Symposium on Precision Mechanical Measurements*, volume 9903, pages 295–304. International Society for Optics and Photonics, SPIE, 2016.

[49] Shuoyuan Liu, Peng Guo, Lihui Feng, and Aiying Yang. Accurate and robust monocular SLAM with omnidirectional cameras. *Sensors*, 19:4494, 2019.

[50] H. C. Longuet-Higgins. A computer algorithm for reconstructing a scene from two projections. In Martin A. Fischler and Oscar Firschein, editors, *Readings in Computer Vision: Issues, Problems, Principles, and Paradigms*, pages 61–62. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[51] Miguel Lourenço and João Pedro Barreto. Tracking feature points in uncalibrated images with radial distortion. In *Proceedings of the 12th European Conference on Computer Vision - Part IV*, ECCV '12, pages 1–14, Berlin, Heidelberg, 2012. Springer-Verlag.

[52] Miguel Lourenço, Joao P. Barreto, and Francisco Vasconcelos. sRD-SIFT: Keypoint detection and matching in images with radial distortion. *IEEE Transactions on Robotics*, 28(3):752–760, 2012.

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

[54] Huimin Lu, Hui Zhang, and Zhiqiang Zheng. A novel real-time local visual feature for omnidirectional vision based on FAST and LBP. In Javier Ruiz-del Solar, Eric Chown, and Paul G. Plöger, editors, *RoboCup 2010: Robot Soccer World Cup XIV*, pages 291–302, Berlin, Heidelberg, 2011. Springer-Verlag.

[55] Bruce D. Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence*, volume 2 of *IJCAI'81*, pages 674–679, San Francisco, CA, USA, 1981. Morgan Kaufmann Publishers Inc.

[56] Will Maddern, Geoff Pascoe, Chris Linegar, and Paul Newman. 1 year, 1000km: The Oxford RobotCar dataset. *The International Journal of Robotics Research*, 36(1):3–15, 2017.

[57] Elmar Mair, Gregory D. Hager, Darius Burschka, Michael Suppa, and Gerhard Hirzinger. Adaptive and generic corner detection based on the accelerated segment test. In *European Conference on Computer Vision (ECCV)*, ECCV '10, pages 183–196, 2010.

[58] Hidenobu Matsuki, Lukas von Stumberg, Vladyslav Usenko, Jörg Stückler, and Daniel Cremers. Omnidirectional DSO: Direct sparse odometry with fisheye cameras. *IEEE Robotics and Automation Letters*, 3:3693–3700, 2018.

[59] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(10):1615–1630, 2005.

[60] Hans Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical Report CMU-RI-TR-80-03, Carnegie Mellon University, Pittsburgh, PA, 1980.

[61] Raúl Mur-Artal, J. M. M. Montiel, and Juan D. Tardós. ORB-SLAM: A versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31:1147–1163, 2015.

[62] David Nistér. An efficient solution to the five-point relative pose problem. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(6):756–777, 2004.

[63] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, volume 1 of *CVPR '04*, pages 652–659, 2004.

[64] Adrien Bartoli Pablo F. Alcantarilla, Jesus Nuevo. Fast explicit diffusion for accelerated features in nonlinear scale spaces. In *Proceedings of the British Machine Vision Conference*, pages 13.1–13.11. BMVA Press, 2013.

[65] Mikael Persson and Klas Nordberg. Lambda Twist: An accurate fast robust perspective three point (P3P) solver. In *European Conference on Computer Vision (ECCV)*, ECCV '18, pages 318–332, 2018.

[66] Long Quan and Zhongdan Lan. Linear n-point camera pose determination. *IEEE Trans. Pattern Anal. Mach. Intell.*, 21(8):774–780, 1999.

[67] Amina Radgui, Cédric Demonceaux, El Mustapha Mouaddib, Driss Aboutajdine, and Mohammed Rziza. An adapted Lucas-Kanade's method for optical flow estimation in catadioptric images. In *The 8th Workshop on Omnidirectional Vision, Camera Networks and Non-classical Cameras (OMNIVIS)*, Marseille, France, 2008. Rahul Swaminathan and Vincenzo Caglioti and Antonis Argyros.

[68] Alejandro Rituerto, Luis Puig, and J.J. Guerrero. Comparison of omnidirectional and conventional monocular systems for visual SLAM. In *The 10th Workshop on Omnidirectional Vision, Camera Networks and Non-classical Cameras (OMNIVIS)*, 2010.

[69] Edward Rosten and Tom Drummond. Fusing points and lines for high performance tracking. In *Proceedings of the Tenth IEEE International Conference on Computer Vision*, volume 2 of *ICCV '05*, pages 1508–1515, Washington, DC, USA, 2005. IEEE Computer Society.

[70] Peter J. Rousseeuw. Silhouettes: A graphical aid to the interpretation and validation of cluster analysis. *Journal of Computational and Applied Mathematics*, 20:53–65, 1987.

[71] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *Proceedings of the 2011 International Conference on Computer Vision*, ICCV '11, pages 2564–2571, Washington, DC, USA, 2011. IEEE Computer Society.

[72] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry [tutorial]. *IEEE Robotics & Automation Magazine*, 18:80–92, 2011.

[73] Davide Scaramuzza and Roland Siegwart. Appearance-guided monocular omnidirectional visual odometry for outdoor ground vehicles. *IEEE Transactions on Robotics*, 24:1015–1026, 2008.

[74] Daniel Scharstein, Richard Szeliski, and Ramin Zabih. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. In *Proceedings of the 2011 IEEE Workshop on Stereo and Multi-Baseline Vision (SMBV)*, pages 131–140, 2001.

[75] David Schubert, Thore Goll, Nikolaus Demmel, Vladyslav Usenko, Jörg Stückler, and Daniel Cremers. The TUM VI benchmark for evaluating visual-inertial odometry. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1680–1687, 2018.

[76] Jianbo Shi and Carlo Tomasi. Good features to track. In *Proceedings of the 1994 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR '94, pages 593–600, 1994.

[77] Irem Stratmann. Omnidirectional imaging and optical flow. In *Proceedings of the 3rd Workshop on Omnidirectional Vision (OMNIVIS)*, pages 104–111, 2002.

[78] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of RGB-D SLAM systems. *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 573–580, 2012.

[79] Shinya Sumikura, Mikiya Shibuya, and Ken Sakurada. OpenVSLAM: A versatile visual SLAM framework. In *Proceedings of the 27th ACM International Conference on Multimedia*, MM '19, pages 2292–2295, New York, NY, USA, 2019. ACM.

[80] Jean-Philippe Tardif, Yanis Pavlidis, and Kostas Daniilidis. Monocular visual odometry in urban environments using an omnidirectional camera. *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2531–2538, 2008.

[81] Engin Tola, Vincent Lepetit, and Pascal Fua. DAISY: An efficient dense descriptor applied to wide-baseline stereo. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(5):815–830, 2010.

[82] Tomasz Trzcinski, Chris Christoudias, Pascal Fua, and Vincent Lepetit. Boosting binary keypoint descriptors. *Proceedings of the 2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2874–2881, 2013.

[83] Shinji Umeyama, Shinji Umeyama, and Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 13(4):376–380, 1991.

[84] Steffen Urban and Stefan Hinz. MultiCol-SLAM - a modular real-time multi-camera SLAM system. *ArXiv*, abs/1610.07336, 2016.

[85] Steffen Urban and Boris Jutzi. LaFiDa - a laserscanner multi-fisheye camera dataset. *J. Imaging*, 3:5, 2017.

[86] Steffen Urban, Martin Weinmann, and Stefan Hinz. mdBRIEF - a fast online-adaptable, distorted binary descriptor for real-time applications using calibrated wide-angle or fisheye cameras. *Comput. Vis. Image Underst.*, 162(C):71–86, 2017.

[87] Vladyslav C. Usenko, Nikolaus Demmel, and Daniel Cremers. The double sphere camera model. *CoRR*, abs/1807.08957, 2018.

[88] Ende Wang, Jinlei Jiao, Jingchao Yang, Dongyi Liang, and Jiandong Tian. Tri-SIFT: A triangulation-based detection and matching algorithm for fish-eye images. *Information*, 9:299, 2018.

[89] Wei Wang, Tim Gee, Jeff Price, and Hairong Qi. Real time multi-vehicle tracking and counting at intersections from a fisheye camera. In *2015 IEEE Winter Conference on Applications of Computer Vision*, pages 17–24, 2015.

[90] Changhee Won, Jongbin Ryu, and Jongwoo Lim. SweepNet: Wide-baseline omnidirectional depth estimation. *CoRR*, abs/1902.10904, 2019.

[91] Baofeng Zhang, Yanhui Jia, Juha Röning, and Weijia Feng. Feature matching method study for uncorrected fish-eye lens image. In Juha Röning and David Casasent, editors, *Intelligent Robots and Computer Vision XXXII: Algorithms and Techniques*, volume 9406, pages 82–90. International Society for Optics and Photonics, SPIE, 2015.

[92] Qiang Zhao, Wei Feng, Liang Wan, and Jiawan Zhang. SPHORB: A fast and robust binary feature on the sphere. *International Journal of Computer Vision*, 113:143–159, 2015.

[93] Qinyi Zhu, Zhijiang Zhang, and Dan Zeng. Robust and efficient method for matching features in omnidirectional images. *Optical Engineering*, 57(4):1–11, 2018.

[94] Zichao Zhang, H. Rebecq, C. Forster, and D. Scaramuzza. Benefit of large field-of-view cameras for visual odometry. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 801–808, 2016.