Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

Visual Odometry Using a Focal-plane Sensor-processor

Author:
Riku Murai

Supervisors:
Prof. Paul Kelly
Dr. Sajad Saeedi

Second Marker:
Prof. Andrew Davison

June 17, 2019

Abstract

With faster processing units and more sophisticated algorithms, the field of computer vision has progressed significantly over the past few years. One of its applications is to provide spatial awareness for the machines, such that they can sense, navigate, and interact with our world. For such machines, ability to react to abrupt changes in the environment is vital. Focal-plane Sensor-processor (FPSP) is a new type of imager, which allows parallel computation to occur on the chip itself. The analog nature of the architecture allows low energy consumption while promising a high frame-rate.

Our work presents, to the best of our knowledge, the first successful 6 Degrees of Freedom visual odometry pipeline which uses the data from the FPSP device. It allows a machine to be aware of its position, while it freely moves around in a three-dimesional space. We propose improvements to the existing feature detector for the device and introduce a feature tracker with robustness against noise. Combining the two algorithms, we complete our pipeline with a non-linear pose estimator. Through the exploitation of parallelism and partitioning of the tasks, we achieve latency of less than 5ms per pose estimate and under 15mm in the absolute trajectory pose error.

Acknowledgements

I would like to thank the following people.

- Prof. Paul Kelly for giving me the opportunity to explore this topic, and his continuous support throughout the project.
- Dr. Sajad Saeedi for his expertise in the field of computer vision, and for his endless encouragements.
- Dimos Tzoumanikas for helping me setup the Vicon motion capture system.
- My friends for making my last four years of study an enjoyable experience.
- And finally, my parents and my sister for their support throughout my study.

Without their support, this project would not have been possible.

Contents

1 Introduction **5**
1.1 Contributions . . . 6
1.2 Report Structure . . . 6

2 Background **7**
2.1 Focal-plane Sensor-processor . . . 7
2.1.1 Scamp5 Device . . . 7
2.1.2 Micro-controller . . . 7
2.1.3 Programming Model . . . 8
2.1.4 Performance . . . 8
2.1.5 Error Model . . . 9
2.1.6 Development Environment . . . 9
2.2 Feature Detection and Tracking . . . 10
2.2.1 Properties of a Good Feature . . . 10
2.2.2 FAST Corner Detection . . . 10
2.2.3 Feature Descriptors . . . 11
2.2.4 Feature Matching . . . 12
2.2.5 Optical Flow Estimation . . . 12
2.3 Monocular Visual Odometry . . . 12
2.3.1 Notations . . . 12
2.3.2 Methodologies . . . 12
2.3.3 Camera Model . . . 13
2.3.4 Pose Representation . . . 14
2.3.5 2D-2D Pose Estimation . . . 15
2.3.6 3D-2D Methods . . . 17
2.3.7 Robust Methods . . . 18
2.4 Related Works . . . 19
2.4.1 Visual Odometry/SLAM on a Frame-based Camera . . . 20
2.4.2 Visual Odometry on an Event-based Camera . . . 20
2.4.3 Visual Odometry on an FPSP Device . . . 21
2.5 Summary . . . 23

3 Feature Detection and Tracking **24**
3.1 Communication with the Scamp5d Device . . . 24
3.1.1 Motivation . . . 24
3.1.2 Settings . . . 25
3.1.3 Communication Methods . . . 25
3.1.4 Limitations and Optimisation of the Communication . . . 25
3.1.5 Improvements to the Communication API . . . 27
3.1.6 Summary of the Communication with the Scamp5d Device . . . 28

3.2 Improved Feature Detection on an FPSP Device . . . 28
3.2.1 Motivation . . . 28
3.2.2 Prior Works . . . 28
3.2.3 Comparison Function . . . 29
3.2.4 Binary Counter . . . 30
3.2.5 Edge Detection . . . 30
3.2.6 Two Ring Detector . . . 31
3.2.7 Non-maximal Suppression . . . 31
3.2.8 Summary of the Improved Feature Detection on an FPSP Device . . . 32
3.3 Feature Tracking . . . 33
3.3.1 Motivation . . . 34
3.3.2 Noise . . . 34
3.3.3 Problem Formulation . . . 34
3.3.4 Particle Filter . . . 35
3.3.5 Implementation Details . . . 36
3.3.6 Summary of the Feature Tracking . . . 37
3.4 Summary . . . 37

4 Visual Odometry **39**
4.1 Initial Attempts . . . 39
4.1.1 Motivation . . . 39
4.1.2 Pose Representations . . . 39
4.1.3 Visualisation . . . 40
4.1.4 Implementation Details . . . 40
4.1.5 Summary of the Initial Attempts . . . 41
4.2 Visual Odometry Pipeline . . . 41
4.2.1 Motivation . . . 41
4.2.2 Initialisation . . . 41
4.2.3 Pose Estimation . . . 42
4.2.4 Map-point Insertion . . . 42
4.2.5 Summary of the Visual Odometry Pipeline . . . 42
4.3 Low Latency Visual Odometry . . . 43
4.3.1 Motivation . . . 43
4.3.2 Preparations . . . 44
4.3.3 Challenges . . . 45
4.3.4 Implementation Details . . . 46
4.3.5 Optimisations . . . 47
4.3.6 Summary of the Low Latency Visual Odometry . . . 48
4.4 Summary . . . 49

5 Evaluation **50**
5.1 Feature Extraction on an FPSP Device . . . 50
5.1.1 Methodologies . . . 50
5.1.2 Observations . . . 51
5.2 Tracking Features Extracted by an FPSP Device . . . 52
5.2.1 Methodologies . . . 53
5.2.2 Observations . . . 54
5.3 Low Latency Visual Odometry Using an FPSP Device . . . 59
5.3.1 Methodologies . . . 60
5.3.2 Observations . . . 62
5.4 Limitations and Further Evaluations . . . 66
5.5 Summary . . . 67

6 Conclusion and Future Works **70**
6.1 Contributions 70
6.1.1 Feature Detection and Tracking 70
6.1.2 Visual Odometry Using an FPSP Device 70
6.2 Challenges and Lessons Learnt 71
6.3 Discussions 71
6.4 Future Works 72
6.4.1 Visual Odometry 72
6.4.2 Development Environment and Testing Framework 73

Appendix A Test Data **80**

Appendix B Feasibility of the Future Works **82**

Chapter 1

Introduction

Spatial awareness is an ability which is required by many technologies, from virtual reality to autonomous robots. The knowledge is often obtainable from sources such as satellites or hardware reference markers. However, in many situations, such information is inaccessible. Visual odometry is a computer vision algorithm which recovers camera pose estimations from a sequence of images. No prior information about the environment or the motion is required, which makes the system usable in a wide variety of situations [41].

Low latency in visual odometry is critical for some of its applications. For the machines which operates around our surrounding, it is critical that they can respond to the abrupt changes in the environment.

Using a typical camera, achieving low latency in computer vision is challenging. The frame-rate of the camera is often limited, which sets a hard boundary for the latency. While use of a high-speed camera would lower this boundary, it now has a large amount of pixel data that must be transferred and processed. The data processing creates significant overhead for the processor, consuming both computational resource and energy. While hardware accelerators such as Graphics Processing Units (GPUs) can speed up the processing, they are often not available on a small robot.

Focal-plane Sensor-processor (FPSP) is a next-generation camera technology which allows pixels of the vision chip to perform computations in parallel, in a Single Instruction, Multiple Data (SIMD) manner. Processing on the pixels allows the data transferred from the vision chip to be minimal, reducing the energy and time required to digitalise the analog values. Reducing the analog to digital conversions allows the FPSP device to be energy efficient while promising a high frame-rate.

This dissertation aims to achieve a 6 Degrees of Freedom (DoF) visual odometry pipeline, utilising the FPSP device to reduce the latency.

Use of the FPSP device to perform visual odometry has been explored prior [8, 18]. However, the methods only achieved 4DoF. Furthermore, the implementation of new algorithms are necessary due to the limited instruction sets and storage space. Unfortunately, as the algorithms widely differ from a typical visual odometry algorithm, extending these to support more degrees of freedom is challenging.

Instead, we propose to break down the visual odometry pipeline and place the tasks accordingly across the different devices. Such separation allows us to borrow ideas from computer vision algorithms which were developed for a typical camera.

1.1 Contributions

The primary objective of our project was to achieve 6DoF tracking using an FPSP device. In summary, the contributions of this project are as follows

- An improved feature detection algorithm, which is specialised for the FPSP device. Through ensembling of multiple feature extraction methods, our feature detector is capable of extracting up to 500 corner features at over 600 Frames Per Second (FPS).
- A robust feature tracking algorithm, which is capable of tracking noisy features produced by the FPSP device. The feature tracker can operate at over 600FPS. Furthermore, it does not assume the motion of features, thus, the FPSP device is free to move around in space while tracking the features. Moreover, the tracker does not limit the use to a static environment. To the best of our knowledge, this is the first successful implementation of feature tracker for an FPSP device.
- A low latency monocular 6DoF visual odometry pipeline using an FPSP device, which to the best of our knowledge has never been attempted before. The pipeline utilises our feature detection and feature tracking algorithm, producing up to 250 pose estimations per second, with a small error of less than 15mm in the absolute trajectory pose error.
- Evaluations of the accuracy and performance of our implementations. We assess all of our implementations, both qualitatively and quantitatively. Comparison against some of the classical approaches such as the Kande-Lucas-Tomasi (KLT) feature tracker is performed to measure the significance of operating computer vision algorithms at a high frame-rate.
- A discussion of the limitations of our approaches, and on the further works.

1.2 Report Structure

The rest of the report can be summarised as follows

- Chapter 2, Background contains the necessary contents required to understand the rest of the dissertation, introducing one type of FPSP device, Scamp5d, and a brief overview of feature extraction, feature tracking, and monocular visual odometry.
- Chapter 3, Feature Detection and Tracking first details the methods to communicate between the Scamp5d and a standard computer. Improvements made to the FAST corner detection on an FPSP device is then introduced. Finally, a discussion about our feature tracker and its implementation detail is made.
- Chapter 4, Visual Odometry incrementally builds a visual odometry pipeline, and discuss the necessary implementation details required for our low latency 6DoF visual odometry pipeline.
- Chapter 5, Evaluation assesses the quality of the feature detector, feature tracker and visual odometry pipeline, and critically review the performance.
- Chapter 6, Conclusion summarises the works and proposes future directions.

Chapter 2

Background

In this chapter, we introduce the next-generation camera technology, FPSP device, and fundamental concepts of monocular visual odometry. In the later sections, we survey the different approaches to monocular visual odometry, and what the different types of camera technologies have to offer.

2.1 Focal-plane Sensor-processor

A Focal-plane Sensor-processor (FPSP) is a general purpose vision device, where sensor and processor arrays are embedded together on the same silicon [59]. Compared to conventional systems, where the camera transfers data to a separate processing unit, FPSP device allows processing of pixels to occur on the chip itself. Computation can produce a sparse representation of the data, reducing the data which is transferred.

2.1.1 Scamp5 Device

One example of an FPSP device is the Scamp5, developed by the University of Manchester [13]. Scamp5 is a general purpose vision device, capable of executing a variety of different algorithms such as feature detection [14] and convolutional neural networks(CNNs) [18, 57]. To aid the development of CNNs on the device, development of automatic kernel generation for the Scamp5 architecture [19] has been explored as well. In this dissertation, the latest iteration of the Scamp5, Scamp5d is used.

The chip (Figure 2.1) captures a grayscale image of 256×256 pixels, and each pixel sensor cell contains a processing element (PE).

Each PE (Figure 2.2) contains 13 general purpose binary registers (R0-R12), 6 general purpose analog registers (A-F), a FLAG register, and a NEWS register. In contrast to conventional digital processors, operations can be performed on analog registers. This allows the chip to bypass the Analog to Digital Conversion (ADC), keeping the PEs compact and further reducing power consumption.

2.1.2 Micro-controller

A Scamp5d device contains the vision chip, which is integrated with a micro-controller. The micro-controller (MCU) used is an NXP LPC4357, integrating two ARM Cortex cores, M4

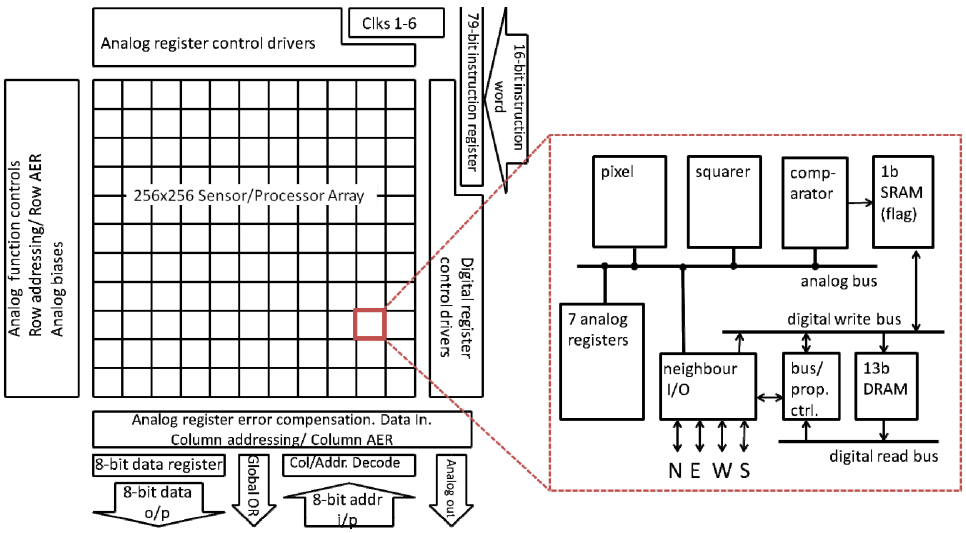


Figure 2.1: The architecture of Scamp5 [13]. The chip contains 256×256 processing elements which each corresponds to a pixel each.

and M0 [15]. The M0 core dispatches the instructions for the vision chip and is responsible for performing other vision chip control tasks. The M4 core is used to perform IO services and has the capability to run user programs. GPIO pins and communication ports such as SPI, UART, I2C are exposed to allow sensors such as inertial measurement unit (IMU) to be attached.

2.1.3 Programming Model

The analog architecture is abstracted away, allowing people who are unfamiliar with analog architecture to code as if they are working on a digital architecture [12]. However, they must be aware of the errors which are present due to the analog nature of the system.

The chip is programmed in a single instruction, multiple data (SIMD). The same instruction is executed across all the PEs, which each applies the operation on their local registers. One can freely use the 13 binary and 6 analog registers. The special registers are the FLAG register, which can be used to mask analog register operations to provide data-dependent branching across the PEs [15] and the NEWS register, which is a special register connected to the 4 closest neighbours (North, East, West, South). The data stored on the NEWS register is accessible from any of the neighbouring PEs.

The data can be transferred from the micro-controller to a host device through read-out operations. The read-out operations for the analog registers goes through the ADC, while the digital registers can be directly read [1]. The infrastructure of the Scamp5 device allows further hardware acceleration beyond parallel processing. For example, a summation across user-defined patterns on analog registers is efficiently implemented on the hardware. Moreover, the time required to a read-out of events ('1's on the digital register) is proportional to the number of events rather than the scanning area.

2.1.4 Performance

The chip is capable of executing instructions at 10MHz and achieves peak computational performance of 655GOPS (grayscale arithmetic operations) and energy efficiency of 1.9pJ/OP [13]. Furthermore, hardware acceleration such as flooding through asynchronous propagation network

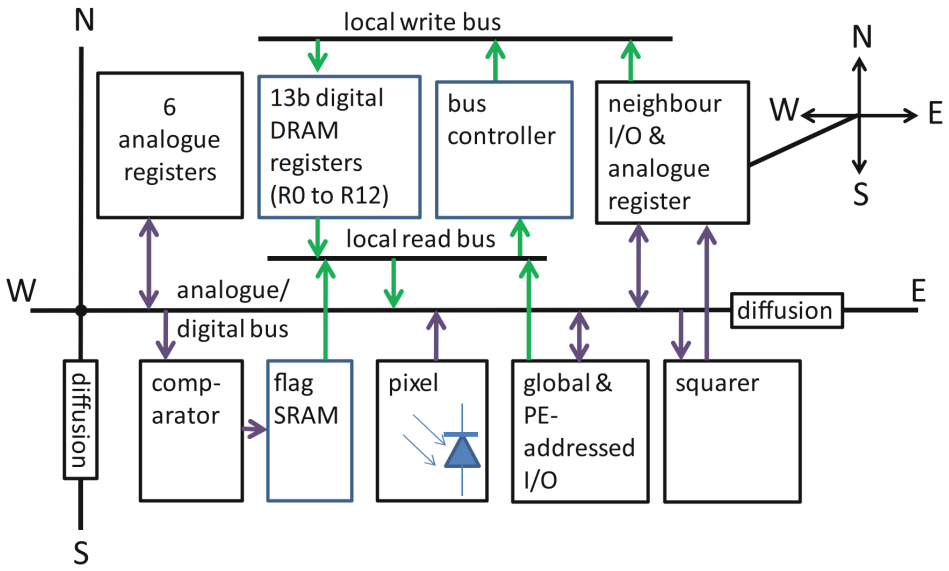


Figure 2.2: The architecture of the processing element on the chip [12]. Each PE contains mixture of analog and digital registers together with an analog photo-diode which measures the intensity.

allows 62× speedup compared to the synchronous version. The chip is capable of tracking an object at 100,000FPS [11], transferring only the coordinate of the object to the micro-controller. This allows the instructions to be executed at 10MHz while only consuming 1.23W.

2.1.5 Error Model

The error [13] for copying the content of register $B_{i,j}$ to $A_{i,j}$ can be modelled as (2.1)

$$A_{i,j} \leftarrow B_{i,j} + k_1 B_{i,j} + k_2 + \varepsilon_{i,j}(t) + \delta_{i,j} \tag{2.1}$$

The fixed error k_2 is unimportant as a simple constant error correction can remove it. The signal dependent error k_1 is non-correctable. At a 10MHz clock, considering a nominal register range of 0 to 100, k_1 is 0.07. $\varepsilon_{i,j}(t)$ indicates the random error associated with a register transfer, for example a thermal noise. Root Mean Square Error (RMSE) across the array is 0.09. $\delta_{i,j}$ is the error due to fixed pattern noise, a constant error particular to a PE location and registers being copied.

The clock speed of the chip can be increased from 10MHz to 16MHz. However, this would increase the noise present on the chip. The values stored in analog registers decays with time. While it is sufficient for short inter-frame temporary storage, to keep the values across multiple frames, one must write the data to the digital registers.

2.1.6 Development Environment

The vision system is programmed in GNU C/C++ using Scamp5d development framework [15]. The code for the micro-controller and the PEs are written together, but the instructions for the PEs are written in a special scope, created by `scamp5_kernel_begin()` and `scamp5_kernel_end()`. Scamp5 kernels get compiled into Scamp5 instructions during the run-time upon the first encounter.

The code which runs on the M0 core can be cross-compiled for the simulator client. This enables the code which runs on the hardware to run on the simulator as is. The simulator emulates the vision chip hardware and provides easy debugging through displaying state of all the registers and enabling step by step execution using a debugger.

The Scamp5 device can operate autonomously. However, a connection to a host computer can be made through a USB 2.0 port. Host application provides GUI for the chip, allowing rendering of the register states and provides an interface to interact with user-defined values such as threshold during run-time. The driver library to interface with the micro-controller is provided, allowing the development of a custom host application for the Scamp5 system.

2.2 Feature Detection and Tracking

Feature detection and matching is an essential aspect of computer vision applications. The correspondences created between the features across multiple images allows, for example, aligning, such that they can be stitched into a single picture, or a 3D model can be created using triangulation [51].

2.2.1 Properties of a Good Feature

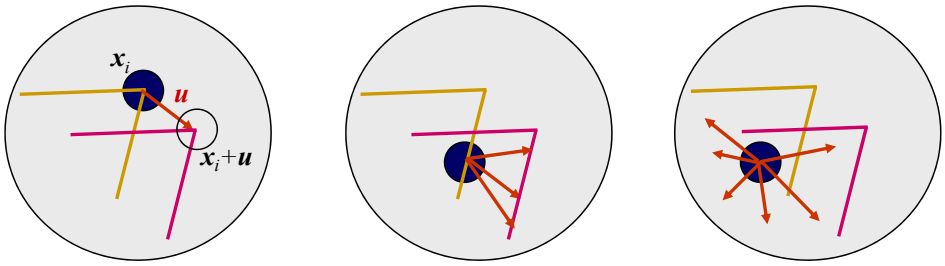


Figure 2.3: Aperture problem for different types of image patches. (a) Corner like feature, (b) Edge feature, (c) Texture-less region. The two images I_0 (yellow) and I_1 (red) are overlaid. The red arrow indicates the possible displacement of the feature between frames. [51].

One property which is required by for a good feature is that it should be possible to reliably find its correspondences in other images. The correspondence is often established using a local patch around the feature and as shown in the Figure 2.3, patches with large contrast change (gradient) is easier to track. For the straight line segments, it suffers from aperture problem, where the alignment is only possible along the normal to the edge. If the patch contains no textures, the alignment process is nearly impossible [51].

Another important property of a good feature is its repeatability in extraction. If the feature is not repeatably detectable, it is difficult to track the feature across frames.

2.2.2 FAST Corner Detection

Features from Accelerated Segment Test (FAST) corner detector [45, 46] is one type of feature detector, which is computationally efficient. Corners have large gradients, which makes it an easy feature to track. Before the FAST corner detector, feature extraction from a live video stream at full frame-rate was too time consuming and left little to no time left for further processing.

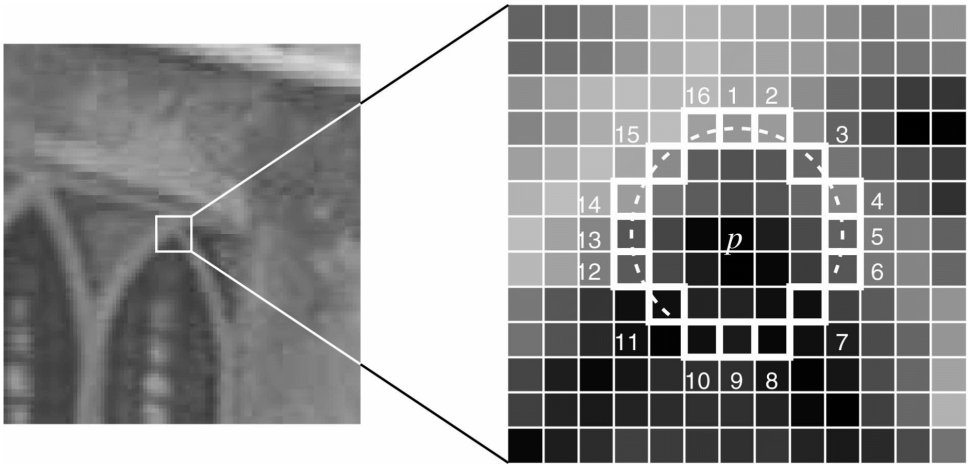


Figure 2.4: Illustration of the FAST corner detection. The highlighted squares are the pixels tested for the corner detection and the pixel p is the centre of a candidate corner. The dashed line indicates the 12 contiguous pixels which are brighter than p by more than the threshold [46].

The algorithm works as follows (Algorithm 1).

Algorithm 1 FAST corner detection

1: **procedure** FAST CORNER DETECTION
2: Select a candidate pixel p
3: Consider a ring of 16 pixels around p as shown in Figure 2.4
4: $N \leftarrow$ Length of longest contiguous pixels in the ring where intensity difference with p is all below or all above a threshold t
5: **if** $N < 12$ **then**
6: Mark p as corner

Early rejection of a candidate pixel is possible by examining 1, 9, 5 and 13th pixels. A feature can only exist if three of these test points are all above or below the intensity of p by t. The number of N contiguous pixels depends on the implementations. However, for $N < 12$ early rejection is not possible.

2.2.3 Feature Descriptors

In many of the use cases of the features, they are matched to find the correspondence. For a continuous video sequence, comparing the features patch-wise may be sufficient. However, in many cases, the patch will undergo appearance change, for example, changes in orientation, scale or illumination. These changes are more significant if one wishes to match features from non-consecutive images. The difference in the appearance makes the task of matching the features challenging.

Descriptors are often used to solve this problem, and a local patch around the feature is encoded in such a way that it is invariant to the appearance changes.

One of such feature descriptors is Scale Invariant Feature Transform (SIFT) [37]. It is invariant to image rotation and scaling, partially invariant to change in illumination and 3D camera viewpoint, making them suitable for tasks such as motion tracking.

While SIFT features are successful and they are used in many applications, it has a large computational burden which makes it impractical for real-time applications. This lead to the

research of more efficiently descriptors, in particular, binary descriptors. Oriented fast and Rotated Brief (ORB) [47] features is a common alternative to SIFT features. It is computationally efficient as feature extraction uses FAST-9 (where $N = 9$) and the feature descriptor is computed using binary comparisons.

2.2.4 Feature Matching

Given a list of features and its descriptors for two images, the simplest method to match the features is to brute-force all the possibilities. However, the complexity grows quadratically with the number of features. Thus it is impractical for most real-time applications. Instead, under an assumption that the inter-frame motion is relatively small, an indexing structure such as multi-dimensional search tree such as K-d tree can be used [51]. The indexing structures use distance as a heuristic, and as long as the assumption holds, it would not compromise accuracy.

2.2.5 Optical Flow Estimation

Under an assumption that one wishes to track features across consecutive video, it is possible to obtain a sub-pixel estimate of how features moved between frames.

One of the commonly used approaches is the Lucas and Kanade Method (LK Method) [5]. It performs gradient descent on the sum of squared difference of local patch around a feature and interpolates the relative translation of the feature [51]. Use of Shi-Tomasi features [48], together with the LK method, is often referred to as the KLT feature tracker.

2.3 Monocular Visual Odometry

The ability to accurately estimate pose is essential for various applications such as navigation and augmented reality. Multiple sensory information can be fused to solve the problem, and typically, with more information, more straightforward it is to estimate the pose. Monocular visual odometry is one of the approaches for pose estimation. It uses a single camera only, which makes the problem complex and requires high engineering effort to implement. However, it is rewarding as a single camera is compact, low energy, cheap and operates both indoors and outdoors [58].

2.3.1 Notations

The notation used to describe the algorithms are as follows. The world frame is represented with $\mathbf{w}$ and the camera frame is represented with $\mathbf{c}$. The position $\mathbf{p}$ in world frame is denoted as ${}_{\mathbf{w}}\mathbf{p}$. The rigid-body transformation $\mathbf{T_{c,w}} \in \mathbf{SE(3)}$ expresses transformation from world to camera frame. This allows a point in world frame ${}_{\mathbf{w}}\mathbf{p}$ to be mapped to camera frame by ${}_{\mathbf{c}}\mathbf{p} = \mathbf{T_{c,w}} \cdot {}_{\mathbf{w}}\mathbf{p}$

2.3.2 Methodologies

Monocular visual odometry can be categorised based on what information they use to predict the pose.

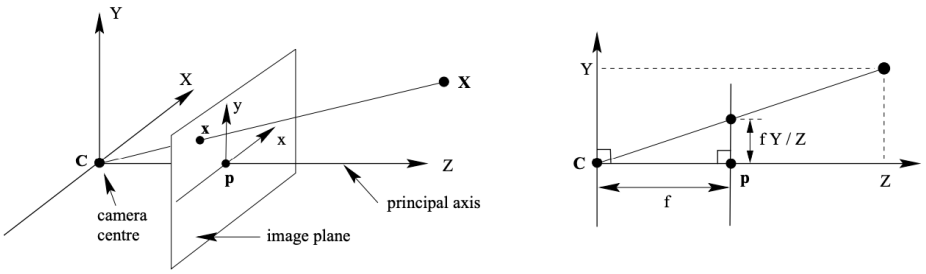


Figure 2.5: Pinhole camera geometry [27]. **C** is the camera centre and **p** is the principle point. The 3D point **X** corresponds to point **x** on the image plane.

1. **Direct Methods:** It is possible to divide this category into dense and semi-dense methods, where the dense method uses all of the available pixel information whereas semi-dense method only uses the pixel information at which the gradient of image brightness is significant [58]. Typically, they operate under the assumption that the intensity difference between two subsequent frames is small. This can be described with equation (2.2)

$$J(x, y) \approx I(x + u(x, y), y + v(x, y)) \tag{2.2}$$

   where J, I represents two consecutive images, which is indexed by a pixel coordinate (x, y). $u(x, y), v(x, y)$ denotes the displacement of the pixels between the images. Rather than considering these constraints separately, direct monocular Simultaneous Localisation And Mapping (SLAM) such as LSD-SLAM [20] estimates u and v as a rigid body transformation of the whole image and the alignment of two images are computed by minimising of photometric error through non-linear optimisation.

   In a scene with a limited amount of textures, the direct methods achieve higher accuracy and robustness when compared to a feature-based approach.

2. **Feature-based Methods:** Feature-based methods reduces the computational complexity by extracting features, rather than considering every pixel. These features are tracked across multiple frames. Thus, the ability to track the features across multiple frames is critical for feature-based methods. PTAM [32], which performs parallel camera tracking and mapping uses FAST corners [46] and local patch correlation to match them. While these methods work on a small scale, it does not support large loop closure. ORB-SLAM [39] uses ORB features [47], which allows robust matching across multiple views, and allows pose estimation to be computed through minimisation of reprojection error.

2.3.3 Camera Model

The pinhole camera is the simplest camera model but is used in many computer vision tasks. As described in [27] a point on the image **x** is computed by intersecting the image plane with line through a 3D point **X** and a camera centre **C** as shown in the Figure 2.5. The mapping between the 3D coordinate to image coordinate can be simply described as

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \mapsto \begin{pmatrix} fX/Z \\ fY/Z \end{pmatrix} \tag{2.3}$$

The centre of the projection is the camera centre, and the line from it perpendicular to the image plane is called principle axis, which intersects at the principle point. Equation (2.3) assumes

that the origin of the coordinate of the image plane is the principle point. In general, this may not be true and the offsets can be expressed as

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} \mapsto \begin{pmatrix} fX/Z + p_x \\ fY/Z + p_y \end{pmatrix} \tag{2.4}$$

Or be expressed in homogeneous coordinates as

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \mapsto \begin{pmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \end{pmatrix} = \mathbf{K} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \tag{2.5}$$

$$\mathbf{K} = \begin{bmatrix} f & & p_x & 0 \\ & f & p_y & 0 \\ & & 1 & 0 \end{bmatrix} \tag{2.6}$$

$\mathbf{K}$ is called the camera calibration matrix or intrinsic parameters. Equation (2.5) can be expressed concisely as

$$\mathbf{x} = \mathbf{K}\,[\mathbf{I} \mid 0]\,{}_{\mathbf{c}}\mathbf{X} \tag{2.7}$$

The point ${}_{\mathbf{c}}\mathbf{X}$ is expressed in the coordinate system where the camera is located at the origin of the Euclidean coordinate system with principle axis pointing down the z-axis. This is called the camera coordinate frame.

A point in space, in general, is expressed in the different coordinate frame, known as world coordinate frame and can be related to the camera coordinate frame through rotations and translations. Let $\tilde{\mathbf{X}}$ and ${}_{\mathbf{c}}\tilde{\mathbf{X}}$ be an in-homogeneous vector representing the same point in the world coordinate and camera coordinate frame respectively. These points can be related by an equation

$${}_{\mathbf{c}}\tilde{\mathbf{X}} = \mathbf{R}(\tilde{\mathbf{X}} - \tilde{\mathbf{C}}) \tag{2.8}$$

Where $\tilde{\mathbf{C}}$ is the coordinate of the camera centre in the world coordinate, and $\mathbf{R}$ is a 3×3 rotation matrix representing the orientation. These are the extrinsic parameters. Equation (2.8) can be rewritten in a homogeneous coordinate as

$${}_{\mathbf{c}}\mathbf{X} = \begin{bmatrix} \mathbf{R} & -\mathbf{R}\tilde{\mathbf{C}} \\ 0 & 1 \end{bmatrix} \mathbf{X} \tag{2.9}$$

This allows definition of a 3×4 projection matrix forms the mapping of a point in world coordinate to a point in plane coordinate is defined as

$$\mathbf{P} = \mathbf{K}\,[\mathbf{R} \mid \mathbf{t}] \tag{2.10}$$

Where $\mathbf{t} = -\mathbf{R}\tilde{\mathbf{C}}$

2.3.4 Pose Representation

A pose of a camera in 3D space can be expressed using rigid body motion in Special Euclidean Group $\mathbf{SE}(\mathbf{3})$, containing a 3D rotation $\mathbf{R}$ and a 3D translation $\mathbf{t}$. For the translation, we can represent them as a 3×1 vector. However, for the rotation, it is more complicated. Rotations in the 3D space are represented in Special Orthogonal Group $\mathbf{SO}(\mathbf{3})$ and have many possible parameterisations. One parameterisation is to use a 3×3 orthonormal rotation matrix. However, it excessively uses 9 parameters to represent 3 rotations. Furthermore, an orthogonality constraint must be enforced. Instead, compact parameterisation of a 3D rotation is often used [51].

- **Euler Angles:** A 3D rotation can be expressed using three rotations around the cardinal axes. Although it uses only 3 parameters, it is generally poor representation since the order in which the transforms are applied results in different orientations, and further, it suffers from Gimbal Lock, making it not possible to move smoothly in the parameter space.
- **Axis-angle:** A rotation can be represented using a rotation axis and an angle to rotate by. Angle-axis representation uses 3 parameters, hence they require no additional constraints on the parameters. However, the representation of the angle is not unique and adding multiples of 2π radian to the angle-axis representation yields the same rotation matrix.
- **Unit Quaternions:** The unit quaternion representation uses a 4-parameter vector with unit norm constraint to represent the 3D rotation. Apart from dual covering ambiguity, where $\mathbf{q} = -\mathbf{q}$, its representation of a rotation is unique. Furthermore, the representation is continuous, and there are no discontinuities in the representation.

The most commonly used parameterisation is the unit quaternions, which constitutes a unit sphere in four-dimensional space [16]. While the representation is not minimal, it is still compact. Furthermore, it has nice properties such as allowing smooth interpolation of the rotation along the arc of the 4D unit sphere.

2.3.5 2D-2D Pose Estimation

A 2D-2D method is a branch of pose estimation methods, which uses correspondences on two image planes to find the relative motion of the camera. It uses the epipolar geometry to find the relationship between the correspondences and decomposes the found relationship into relative motion estimate.

Fundamental Matrix

Given two perspective views, a 3D point $\mathbf{X}$ can be represented on the image plane as

$$\mathbf{x} = \mathbf{P}\mathbf{X}, \quad \mathbf{x}' = \mathbf{P}'\mathbf{X} \tag{2.11}$$

Where $\mathbf{P}$ is the projection matrix and $'$ indicates entities associated with the second view. Image point $\mathbf{x}$ and $\mathbf{x}'$ are corresponding point as they are projection of the same 3D point [27].

The epipolar geometry is used to describe the intrinsic projective geometry between two views. The algebraic representation of the epipolar geometry is the fundamental matrix $\mathbf{F}$ which is 3×3 matrix of rank 2. The fundamental matrix satisfies the relation.

$$\mathbf{x}'\mathbf{F}\mathbf{x} = 0 \tag{2.12}$$

The fundamental matrix relates the corresponding image points on two separate image planes, and if the camera calibration parameter is known, it is possible to recover the relative pose estimate.

Essential Matrix

A fundamental matrix is a general form of an essential matrix and describes a similar relationship. However, an essential matrix has fewer degrees of freedom as it operates on the normalised coordinates.

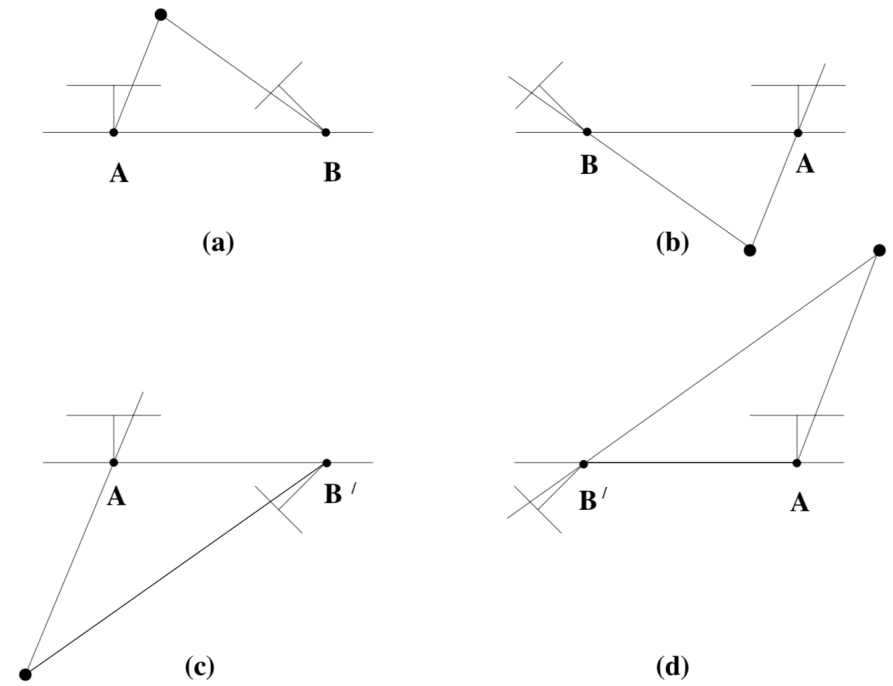


Figure 2.6: The geometric representation of possible solutions for the camera positions [27].

Let the projection matrix be decomposed as $\mathbf{P} = \mathbf{K}[\mathbf{R} \mid \mathbf{t}]$ and let $\mathbf{x} = \mathbf{PX}$ be a point in the image. If the calibration matrix $\mathbf{K}$ is knows, then the inverse can be applied to the point $\mathbf{x}$ to obtain the point $\hat{\mathbf{x}} = \mathbf{K^{-1}x} = [\mathbf{R} \mid \mathbf{t}]\mathbf{X}$. The point $\hat{\mathbf{x}}$ is the image point expressed in the normalised coordinates. The camera matrix $\mathbf{K}^{-1}\mathbf{P} = [\mathbf{R} \mid \mathbf{t}]$ is the normalised camera matrix, which is camera matrix with effect of the known calibration removed.

The essential matrix, similarly to the fundamental matrix satisfies

$$\hat{\mathbf{x}}'^T \mathbf{E} \hat{\mathbf{x}} = 0 \tag{2.13}$$

For all corresponding $\hat{\mathbf{x}}$, $\hat{\mathbf{x}}'$ pairs. Substituting for $\hat{\mathbf{x}}$, $\hat{\mathbf{x}}'$

$$\hat{\mathbf{x}}'^T \mathbf{E} \hat{\mathbf{x}} = \mathbf{x}'^T \mathbf{K}^{-T} \mathbf{E} \mathbf{K}^{-1} \mathbf{x} = \mathbf{x}'^T \mathbf{F} \mathbf{x} = 0 \tag{2.14}$$

Hence the relationship between the essential matrix and the fundamental matrix is

$$\mathbf{E} = \mathbf{K}^{-T} \mathbf{F} \mathbf{K} \tag{2.15}$$

Pose Recovery

It is possible to recover the camera's extrinsic parameters when given the essential matrix [27]. In case where fundamental matrix is computed instead, given $\mathbf{K}$, the conversion to essential matrix is possible through Equation (2.15).

Assume that the camera matrix which corresponds to the first image is $\mathbf{P} = [\mathbf{I} \mid 0]$. The camera matrix, which corresponds to the second image is $\mathbf{P}'$, and it contains the relative transformation. The essential matrix $\mathbf{E}$ can be decompose into $\mathbf{E} = \mathbf{U}\,\mathrm{diag}(1, 1, 0)\mathbf{V}^T$. There are four possible choices for the second camera matrix $\mathbf{P}'$

$$\begin{aligned} \mathbf{P}' = [\mathbf{UWV}^T \mid \mathbf{u_3}], \text{ or } \mathbf{P}' = [\mathbf{UWV}^T \mid -\mathbf{u_3}], \\ \text{or } \mathbf{P}' = [\mathbf{UW}^T\mathbf{V}^T \mid \mathbf{u_3}], \text{ or } \mathbf{P}' = [\mathbf{UW}^T\mathbf{V}^T \mid -\mathbf{u_3}] \end{aligned} \tag{2.16}$$

where

$$\mathbf{W} = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{2.17}$$

There is only one solution out of four, where the point is in front of both camera positions (Figure 2.6). Hence, using a 3D single point, it is possible to determine the correct configuration of the camera position.

8-point Algorithm

One of the methods to compute the fundamental matrix is the 8-point algorithm. It is a simple and fast algorithm, which uses 8 or more correspondences to compute the fundamental matrix. Despite its simplicity, the algorithm was often criticised for being too sensitive to noises, and there were developments of iterative approaches. However, it was shown that with a simple normalisation trick, the modifies 8-point algorithm performs as good or sometimes better than the iterative methods. The modified 8-point method remains simple and compared to the iterative approaches, the overhead of normalisation is insignificant [28].

The 8-point algorithm is an uncalibrated method since the intrinsic camera parameter is not necessary. It suffers from structure degeneracy if the viewed structure is planer [54]. In such cases, homography and fundamental matrix can be computed and be selected appropriately.

5-point Algorithm

5-point algorithm [40] is an efficient algorithm for the computation of the essential matrix. Compared to the 8-point algorithm, it is much more complex, but it has an advantage that it does not suffer from planar degeneracy.

2.3.6 3D-2D Methods

Monocular visual odometry systems suffer from scale ambiguity. For example, relative motion estimation using the Fundamental or the Essential matrix (2D-2D method) can only estimate the direction of the translation [27]. This means that between multiple successive motion estimate, there is no scale consistency. This is not desirable, as a combination of the relative motion estimates will not be consistent with the actual trajectory.

Instead, it is possible to reduce the scale ambiguity between each frames using 3D-2D methods. Given a map of 3D points and corresponding 2D points by minimising the reprojection error, it is possible to compute the pose estimate. Since the map is consistent across multiple pose estimation, the scaling between these estimates is consistent.

Bootstrapping

A key assumption made in 3D-2D methods is the availability of the 3D map. However, in reality, such prior information is often not available. One solution to the problem, which is often referred to as bootstrapping, is to use the 2D-2D methods for initialisation [32, 39] and create the 3D map through triangulation.

Perspective-n-Points

The aim of Perspective-n-Points problem (PnP) is to determine the position and the orientation of a camera given its intrinsic parameters and n correspondences between the world and image coordinates. One of the methods EPnP [35] is applicable to both planar and non-planar configurations given $n \geq 4$. It is a non-iterative solution to the PnP problem and has a complexity of $O(n)$. Compared to the other iterative methods, it is much faster, more stable, and is only slightly less accurate. While the algorithm supports $n \geq 4$ RANSAC is used on a small subset of correspondences to filter out the outliers in the data-set.

Motion Only Bundle Adjustment

Given a 3D map and its correspondences on a image plane, poses can be estimated by minimising the reprojection error, which can be formulated as Equation (2.18) [50].

$$\mathbf{T}_{\mathbf{c},\mathbf{w}} = \arg\min_{\mathbf{T}_{\mathbf{c},\mathbf{w}}} \frac{1}{2} \sum_i \|\mathbf{u}_i - \pi(\mathbf{T}_{\mathbf{c},\mathbf{w}}\, {}_{\mathbf{w}}\mathbf{p}_i)\|^2 \tag{2.18}$$

Where the error between the projected 3D point $\pi(\mathbf{T}_{\mathbf{c},\mathbf{w}}\, {}_{\mathbf{w}}\mathbf{p}_i)$ and corresponding observed pixel point $\mathbf{u}_i$ is minimised. Notice that minimisation only occurs for $\mathbf{T}_{\mathbf{c},\mathbf{w}}$ hence, no refinement to the 3D map is made.

Equation (2.18) is not a linear problem and requires iterative gradient-based methods such as Gauss-Newton method and the Levenberg-Marquardt method. Usually, the iterative method requires manual derivation of the Jacobian. However, a library such as Ceres Solver [4] performs automatic differentiation to obtain the derivatives.

2.3.7 Robust Methods

In many computer vision algorithms, the input data is noisy and can cause the algorithms to produce an incorrect hypothesis. Thus, it is important to reduce the effects of outlying data. Robust methods is a way which allows outlying data to have less or no influence on the hypothesis, allowing viable solutions to be computed from noisy data.

Random Sample Consensus

For algorithms which operate on a subset of the input data (e.g. 8-point algorithm), an approach for removing outliers is RANdom SAmple Consensus (RANSAC) [21]. It is capable of interpreting data which contains errors, generating the most sensible output by fitting multiple hypothesises and choosing one which explains the data best. Outline of the algorithm is summarised in Algorithm 2.

Robust Optimisation

Non-linear optimisation methods such as motion-only bundle adjustment use all of the available input data for the optimisation process. Unlike RANSAC, where a subset of the data is found to generate the most sound hypothesis, it is possible to modify the formulation such that the optimisation process is robust to outlying data while using all the data.

Algorithm 2 RANSAC Algorithm

```
procedure RANSAC
    best_error ← inf
    best_hypothesis ← ∅
    while i < number of iterations do
        sample_set ← k randomly sampled samples from the dataset
        Compute hypothesis H from sample_set
        for sample not in sample_set do
            if sample fits the hypothesis H then
                Insert sample to sample_set
        Compute better hypothesis H′ from updated sample_set
        error ← Mean error of sample_set with respect to hypothesis H′
        if error < best_error then
            best_error ← error
            best_hypothesis ← H′
```

Let the process of minimisation be formulated as Equation (2.19)

$$\min \sum_i r_i^2 \tag{2.19}$$

Where r_i is the residual. In an ideal world with the absence of noisy data, the process of minimising the residuals gives a reasonable estimate of the optimised parameter. However, in cases where there exists some outlying data, which is very common in computer vision tasks, outlying data influences the minimisation. Thus, the estimate is distorted [60]. Robust optimisation is a means to avoid such influences, in particular M-estimators replaces squared residuals r_i^2 with an other function ρ hence formulation is replace with Equation (2.20)

$$\min \sum_i \rho(r_i) \tag{2.20}$$

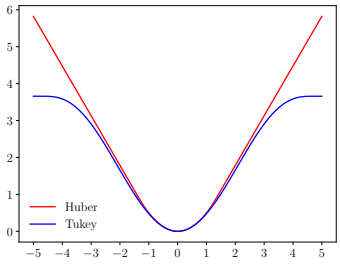


Figure 2.7: Huber and Tukey ρ. $k = 1.345$ and $c = 4.685$

M-estimators which are commonly used in visual odometry is Huber M-estimator and Tukey M-estimator.

$$\text{Huber}(x) = \begin{cases} \frac{x^2}{2} & \text{if } |x| \leq k \\ k(|x| - \frac{k}{2}) & \text{otherwise} \end{cases} \tag{2.21}$$

$$\text{Tukey}(x) = \begin{cases} \frac{c^2}{6}\left(1 - \left[1 - (x/c)^2\right]^3\right) & \text{if} |x| \leq c \\ \frac{c^2}{6} & \text{otherwise} \end{cases} \tag{2.22}$$

As shown in Figure 2.7 Tukey M-estimator more aggressively ignores the influence of outliers, by giving equal weights after a threshold. Similarly, Huber M-estimator does not assign quadratically increasing weights to outliers, making them both less susceptible to the outliers.

2.4 Related Works

The following section provides an overview of different monocular visual odometry and visual SLAM systems which operates on standard frame-based cameras, event cameras, and FPSP devices.

2.4.1 Visual Odometry/SLAM on a Frame-based Camera

Parallel Tracking and Mapping (PTAM) [32] is a frame-based SLAM system which operates effectively on a small scale. It requires no prior information about the surrounding and user interaction is only required for initialisation. Being a sparse system, they extract FAST corners and uses local patch around the features for matching. Since the FAST corners are scale variant, these are extracted at different image pyramids. Two threads are utilised to track and map simultaneously, which allows adjustment to occur over multiple keyframes. It operates in real-time for small scenes. However, as the system explores and visits more scenes, the map maintained becomes larger. Increase in the map size prevents global bundle adjustment from keeping up with the exploration, and the camera must remain stationary or return to a well-mapped area to proceed.

ORB-SLAM [39] is another frame-based SLAM-system, which extends on the ideas from PTAM. The operation of the system is not limited to a small scale, allowing real-time operation in large environments. Use of the ORB features [47] for tracking, mapping, re-localisation and loop-closing, makes the system efficient, reliable and straightforward. Unlike the initialisation process of PTAM, ORM-SLAM requires no user input, as it uses automatic and robust initialisation procedure based on model selection. The computational overhead which PTAM suffered from is reduced by the use of a co-visibility graph. Co-visibility graph creates an edge between the keyframes when the keyframes observe similar features. This allows the bundle adjustment to occur on a local scale, reducing its complexity. Furthermore, if the system notices that it is visiting the same scene, it can perform loop closure. Using essential graph, which is a spanning tree of the co-visibility graph, bundle adjustment is used to optimise the poses. Experiments show that pose graph optimisation over the essential graph is so effective such that the use of full bundle adjustment barely improves the performance.

Both PTAM and ORB-SLAM are feature-based methods since they extract features and use them for tracking. *Semi-direct Visual Odometry* (SVO) [22] is a semi-direct approach, were both image intensity and features are used for pose estimation. SVO extract features only for the keyframes and performs rough motion estimate using minimisation of photometric error of hundreds of small patches. Using the motion estimate, a correspondence between the 3D map and the current frame is established, which allows the point feature-based motion estimation to further refine the results. This method allows sub-pixel feature correspondence, and reduce the computational overhead by extracting features only on the keyframes. Furthermore, SVO uses Bayesian filter which explicitly models outlier measurements to estimate the depth of a feature, rather than using triangulation like in PTAM and ORM-SLAM. While the Bayesian filter is more computationally expensive, it contributes towards reducing the number of outliers in the 3D map.

2.4.2 Visual Odometry on an Event-based Camera

Most computer vision algorithms operate on frames of the video, which is effective as sensors can capture a scene at high resolution since the pixels of the imager is simple. However, the frame-based architecture transfers redundant information, as pixels are sampled repetitively even if their values are unchanged. The event-driven dynamic vision sensors (DVS) tackles this problem by reducing the data transfer by asynchronously reporting the log intensity changes. The pixels do not report values if there are no significant intensity changes, allowing sub-millisecond latency, which makes the sensor appealing for short latency vision problems [36].

While DVSs is an interesting approach for low latency vision sensor, it disregards absolute intensity information, which is used in tasks such as object recognition. Dynamic and Active

pixel vision sensor (DAVIS) approaches this problem by allowing the simultaneous output of asynchronous events and synchronous frames [10].

Use of the DVS in visual odometry is an active field of research. Kim et al. demonstrated that through the use of multiple probabilistic filters, not only 6DoF tracking but a real-time 3D reconstruction of the scene is possible [31]. This approach, however, requires hardware accelerator such as GPU to operate in real-time. *EVO* creates semi-dense 3D map and operates on standard CPU in real-time [44]. The method works accurately under very challenging scenarios such as aggressive motion and abrupt changes of illumination.

Combination of the DVS and other sensory information has been explored. *Ultimate SLAM* fuses the data from DVS with data from a frame-based camera and inertial sensor, providing pose estimation which is robust to fast motion and difficult lighting situations [56]. It is computationally efficient and was integrated onto a computationally-constraint quadrotor.

Kueng et al. presents the first work on event-based visual odometry with the DAVIS [33]. It extracts features from the synchronous frame, and perform feature tracking using the asynchronous events, using the Iterative Closest Point (ICP) algorithm. The tracked features are using for visual odometry using a modified pipeline of SVO.

The DVS/DAVIS and FPSP devices concepts are similar, in a sense that they both perform data reduction on the camera itself to reduce the bandwidth. However, the DVS/DAVIS often produces more redundant information, as it lacks the capability to perform computation on the vision chip itself. For example, FPSP device can output the coordinates of FAST-corner [14] directly, whereas, for DVS, a computation to reduce the event stream to corner event stream must be implemented [38]. However, in the case where there is no motion in the scene, DVS/DAVIS would produce no events, whereas an FPSP device would continuously report data.

2.4.3 Visual Odometry on an FPSP Device

Visual odometry on the FPSP devices is at its early stage of research, and there are many open problems to be solved. The current state of art visual odometry on the FPSP devices achieves 4DoF, making them suitable for agents which are mechanically restricted to move into one direction.

Debrunner et al. uses direct methods to estimate the position of the camera [18], achieving a system which is capable of operating at 400-500Hz.

The 2DoF motion estimate is computed using a gradient-based algorithm, where the previous frame and current frame is shifted in x, y direction until the sum of absolute difference is minimised. This corresponds to the translation in x, y axes, assuming that the motion is constrained in these two axes. Alternatively, it is possible to compute the yaw and pitch of the camera instead.

This idea is extended to 4DoF, namely yaw, pitch, roll, and translation in the z-axis. The camera image is partitioned into 16 titles, where on each of the tile, the relative motion vector is computed. These are combined to form

$$\mathbf{m} = \begin{bmatrix} u_1 & v_1 & u_2 & v_2 & \cdots & u_{16} & v_{16} \end{bmatrix}^T$$

The vector $\mathbf{m}$ can be re-written as the linear combinations of normalised base vector fields. Let $\{b_{yaw}, b_{pitch}, b_{roll}, b_z\}$ (Figure 2.8) be the vector of same form as $\mathbf{m}$. Then the model can be

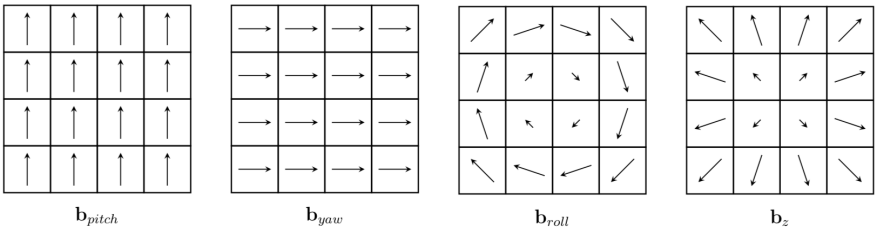


Figure 2.8: The illustration of the base vector fields [18]. The inter-frame motion can be represented with linear combinations of these vectors.

.

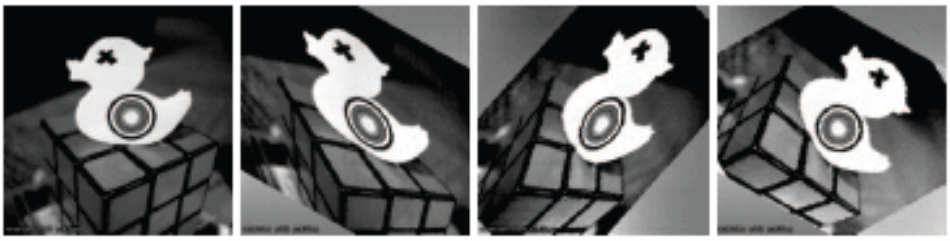


Figure 2.9: Illustration of the rotation through consecutive shearing [8].

written as

$$\mathbf{m} - (\alpha \cdot \mathbf{b}_{yaw} + \beta \cdot \mathbf{b}_{pitch} + \gamma \cdot \mathbf{b}_{roll} + \delta \cdot \mathbf{b}_z) = \epsilon$$

Where ϵ is the error in the estimate. The parameters can be recovered using Ordinary Least Squares (OLS) under the assumption that the error follows Gaussian distribution.

Let $\mathbf{B} = \begin{bmatrix} \mathbf{b}_{yaw} & \mathbf{b}_{pitch} & \mathbf{b}_{roll} & \mathbf{b}_z \end{bmatrix}$. Then using OLS and that $\mathbf{B}$ is orthonormal,

$$\begin{bmatrix} \alpha & \beta & \gamma & \delta \end{bmatrix}^{\mathrm{T}} = \left(B^{\mathrm{T}}B\right)^{-1} B^{\top}\mathbf{m} = B^{\top}\mathbf{m}$$

Furthermore, RANSAC is used to eliminate the outliers which is present in the data, making the system more robust.

Bose et al. proposed a feature-based approach, where camera ego-motion in 4DoF is estimated by conducting image alignment between two consecutive frames using image transformations. From the transformations applied to the images, it is possible to compute the camera orientation and the transformation [8]. Similar to the above method, all of the computation is performed on the Scamp5 device. The images are converted to edges through binary edge detection. The binary representation is suited for the Scamp5 architecture since it only occupies a single digital register on each of the PEs.

Due to the nature of the Scamp5 architecture, the image transformations are not simple compared to methodologies used commonly. The image scaling is performed by duplicating or deleting rows and columns, depending on whether the operation is up-scaling or down-scaling. Rotation is represented by a sequence of three shear transformations [8, 17] (Figure 2.9). The image alignment is performed iteratively, where at each step, a number of potential transformation are applied and tested, evaluating whether it improves the alignment.

Since all of the operations occur on the Scamp5, this method is capable of operating at frame-rates above 1000Hz (ranging between 1000 to 1500Hz), although sufficient lighting is required.

2.5 Summary

In this chapter, we have covered the overview of the FPSP device and monocular visual odometry and surveyed the existing approaches. We notice that frame-based camera and even DVS/DAVIS transfers redundant information. Furthermore, the field of visual odometry using an FPSP device is not fully explored, and to best of our knowledge, no 6DoF tracking has been implemented. To work towards the 6DoF tracking, we first visit feature detection and tracking, which is an essential part of feature-based visual odometry.

Chapter 3

Feature Detection and Tracking

In this chapter, we present improvements to the existing feature detection algorithm for an FPSP device, and our feature tracker, which is capable of tracking the detected features. We will first discuss the limitations in the communications between an FPSP device and a host device, and how we can reduce their impact. In the following sections, we will propose the different methods to improve the feature extraction on the FPSP device, and finally, discuss the implementation details of our feature tracker.

3.1 Communication with the Scamp5d Device

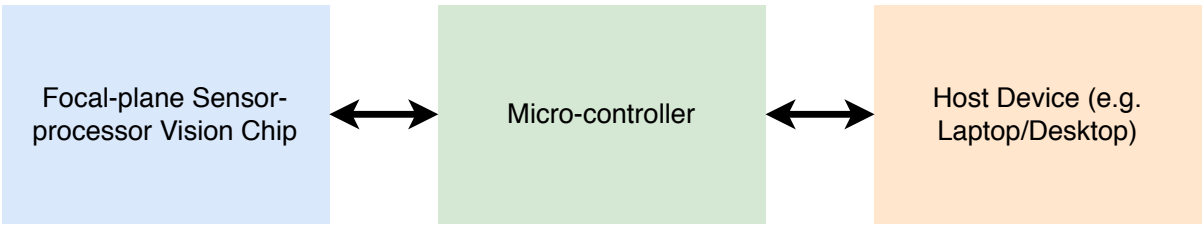


Figure 3.1: Illustration of the communication channels between the devices.

This section introduces the development environment available for communications between an FPSP device and a host device, exploring different limitations which are present in the system. The communication occurs between an FPSP vision chip and a micro-controller, a micro-controller and a host device as illustrated in Figure 3.1. Understanding of the limitations allows informed decisions to be made when offloading the computationally heavy processing from an FPSP device to a host device. Furthermore, we will discuss the implementation of the wrapper class for the API provided, which enables us to register stateful functions.

3.1.1 Motivation

The majority of the previous works [8, 18] on using the FPSP device was implemented solely on the device itself. This removes the necessity to transfer the data to the host device, however, as the computational power on an FPSP device is limited, executing computationally expensive algorithms was not possible.

Instead, an alternative approach is to transfer the data to a more capable host device. Not only the computational benefits, but this allows many libraries to be used with ease as one does not

need to cross-compile for the architecture of the micro-controller.

By evaluating the limitation of the communications between an FPSP device and a host device, we can uncover the additional constraints which we should be aware of.

3.1.2 Settings

Implementation of the Application Programming Interface (API) for the communications between an FPSP device and a host device (`scamp5d_interface`) was originally developed by Manchester University [1]. As the API was designed for Linux systems only, we have applied some modification such that it works for both Linux system and OS-X. Furthermore, we have changed the build system to use `CMake` instead of `Code::Block` build system for ease of use.

For all the experiments, measurements are taken on MacBook Pro (Retina, 13-inch, Early 2015) 3.1 GHz Intel Core i7 16GB RAM, unless otherwise stated.

The frame-rate of the FPSP device is controlled through `vs_wait_frame_trigger()`. We set the mode of the frame trigger to be `VS_FRAME_TRIGGER_PERIODIC` using `vs_frame_trigger_set()`.

3.1.3 Communication Methods

Scamp5d Vision System [1] provides multiple means of communications between an FPSP and a host device. It is capable of transferring different data-types such as analog or digital register values. For example, `scamp5_output_events()` on an FPSP device would scan a digital register and send over coordinates of '1's. On the host device, one can register a function to respond to such data, which is executed upon arrival of the data. The data can be parsed and be used for further processing. For each of the different data-types, user can register an arbitrary function.

One important factor to be considered is that `scamp5d_interface` provides `on_receive_packet` and `on_free_packet` and latter option transfers the ownership of the data to the user-defined function, thus the memory allocated for the packet must be freed implicitly.

From a host device, it is also possible to transfer data to an FPSP device. This is particularly useful for tasks such as parameter tuning since reprogramming of the FPSP device can be avoided. One method of doing this is to set the slider values of the FPSP device using `post_message(VS_MSG_ROUTE_M0, VS_MSG_GUI_UPDATE, VS_GUI_USER_ITEM_<i>, <value>)`. Since the FPSP device continuously executes the main loop whilst the power is supplied, the values of the sliders must be read inside the loop.

3.1.4 Limitations and Optimisation of the Communication

The frame-rate of an FPSP device is limited by multiple factors. The main factor is the amount of data which is transferred from an FPSP device to the micro-controller and vice-versa. This is due to the increased amount of data which goes through ADC. Another factor is the number of instructions per frame. However, this is less significant compared to increasing the amount of data transfer.

In a case where one is developing on an FPSP device without any need for a host device, the amount of data transfer and the number of instructions per frame are the two main factors to

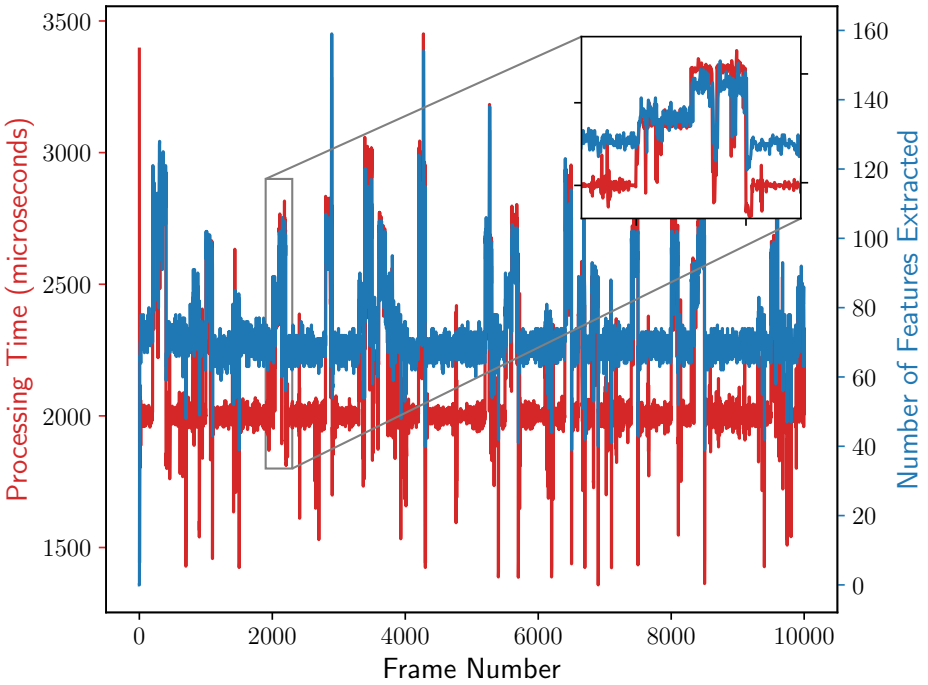


Figure 3.2: Number of features captured at 500FPS with added random delay in the host applications registered function.

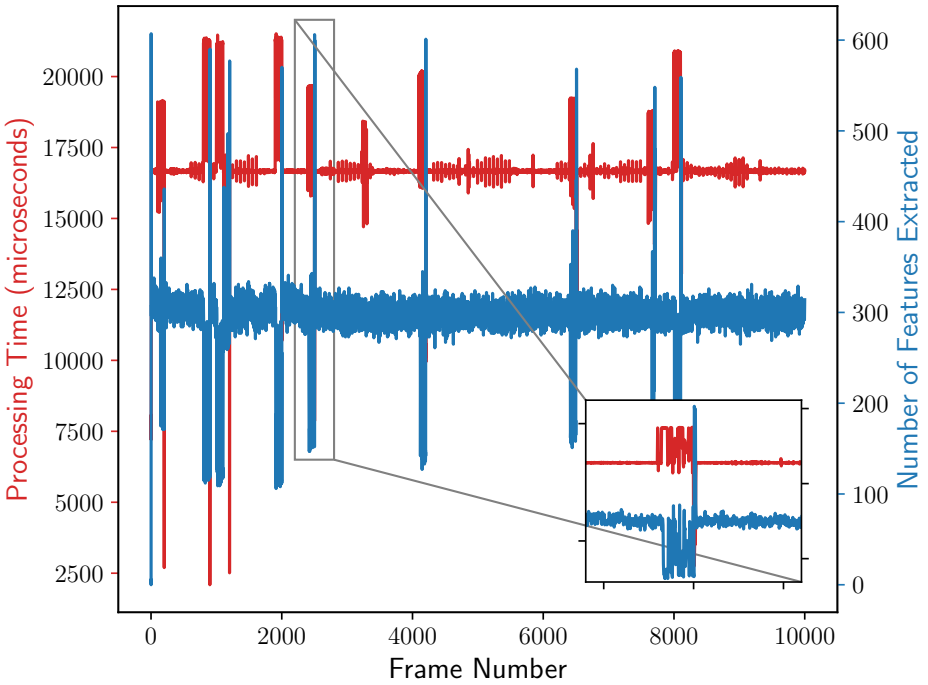


Figure 3.3: Number of features captured at 60FPS with added random delay in the host applications registered function.

be considered in terms of performance. However, when the development requires a host device, one must pay attention to the additional overheads.

When an FPSP device transfer data to a host device, a registered function is invoked on the host, and the FPSP device waits for the invoked function to complete. This means that if the function takes longer to complete than the frame-rate set on the FPSP device, the device busy-waits for the completion. The additional delay results in the FPSP device experiencing different exposure times for each frame. To summarise, the FPSP device would experience different exposure if the following conditions are met.

1. Time for completion of a registered function on the host device exceeds the frame-rate set on the FPSP device.
2. Time for completion of a registered function varies across iterations.

To evaluate the effect of different exposure time for the feature detection, we have run a feature extraction algorithm on an FPSP device and sent the data over to a host device. On the host device, we artificially added some random delays to our registered function, such that the FPSP device experiences different exposure times.

In a case where the FPSP device is operating at a high frame-rate, as shown in Figure 3.2 addition of the delay increases the number of features detected. This is expected as at 500FPS, there is insufficient exposure, and additional exposure illuminates the image better, making the features more distinct.

On the other hand, when the FPSP device is operating at a low frame-rate, as shown in Figure 3.3, the number of features detected decreases with the delay. This is caused by over-exposure, where the image is too bright, and the textures are washed away.

For our purpose, having differing exposure time effects the repeatability of feature detection. Unreliable feature detection makes the implementation of the feature tracking more challenging, hence it is in our favour to reduce this effect. To reduce the noise, we dedicate a thread for receiving data from the FPSP device, where minimal processing is performed to capture the incoming data. Minimising the time spent in the registered function reduces the variance in the time it takes to complete, allowing our feature extraction to be more repeatable, thus, more reliable.

3.1.5 Improvements to the Communication API

The API provided allows the user to register functions for different data-types. This establishes the communication methods between the Scamp5d device and the host device.

In many cases, different applications require a different set of functions to be registered for communication. For example, visual odometry and calibration would handle the incoming data differently. Due to the implementation of the API, to achieve registrations of a different set of functions, the code must be duplicated. Thus, we implemented a wrapper class for the API, which allows the user to register different functions by registering lambda functions. The wrapper class allows the setup of the communication to be encapsulated in the wrapper class, reducing duplication.

One problem with the approach is that the lambda functions must not contain references, as it would result in dangling references. This forces the lambda functions to be stateless, which is not always desirable. To counter this limitation, we allow the wrapper class to be initialised with a state object, which is passed down to the registered lambda functions. This avoids the dangling reference problem while allowing the registered function to have states. We have effectively used

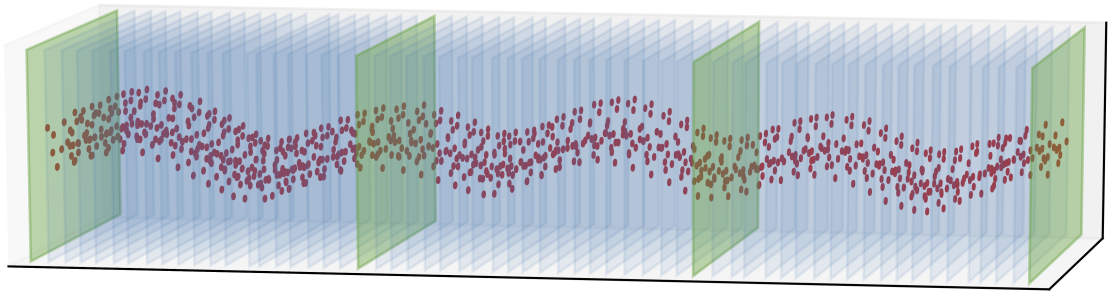


Figure 3.4: Visualisation of features extracted using a frame-based camera (green) and an FPSP (blue)

the templating feature of C++ such that the type of the state object is not restricted. This allows one to simply create an arbitrary object and use them freely in the registered functions.

3.1.6 Summary of the Communication with the Scamp5d Device

We have summarised the characteristic of the device, mainly the communications which occur between the FPSP device and the host device. We have identified a potential limitation in the communication API. The more in-depth understanding of the device has allowed us to make a decision such as dedicating a thread for communication purpose only, which aids the algorithms which operate on the FPSP device to be more stable. Furthermore, we have implemented a wrapper class for the communication API, which simplifies the interface with the Scamp5d device.

3.2 Improved Feature Detection on an FPSP Device

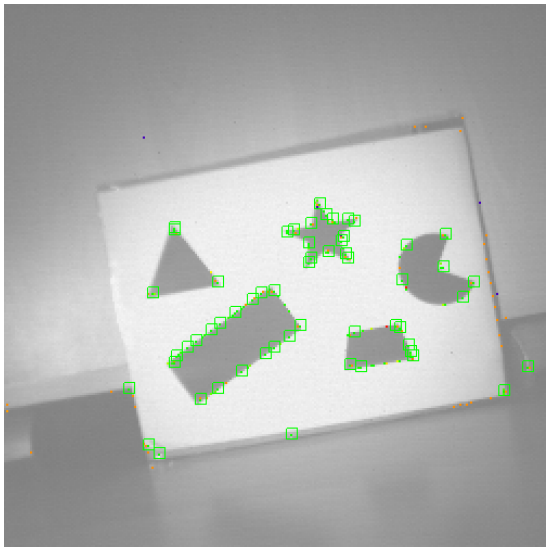
This section details the refinements applied to the existing FAST corner detection implemented for the FPSP device [14]. We examine the current implementation and compare step by step how our proposed solution improves the quality of the features extracted.

3.2.1 Motivation

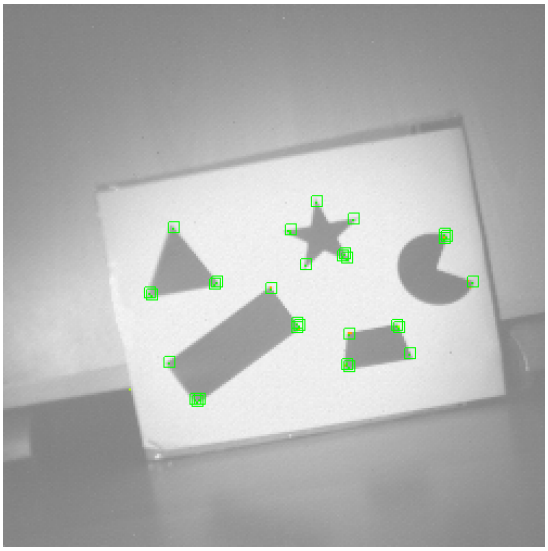
As visualised in Figure 3.4 an FPSP device can extract features at a much higher frequency when compared to a standard frame-based camera. The fast feature extraction allows different computer vision algorithms to take advantages of the small inter-frame motions. While there is an implementation of the FAST corner detection on the FPSP device, we have noticed that it is too sensitive to the features which are hard to track. For accurate feature tracking, suppression of poor quality features is necessary.

3.2.2 Prior Works

One of the challenges in implementing the FAST corner detection on the FPSP device is maintaining the count. Ability to count is required, such that the number of contiguous pixels which has passed the intensity comparison check can be tracked. This is made difficult by the limitations of the FPSPs instruction sets and the analog register, which is too noisy for the purpose. As a solution, Chen et al. has proposed the use of a ring counter. While it is simple



(a) Implementation by Chen et al.. There are many edges as classified wrongly as a corner, which makes the task of tracking more difficult.



(b) Our proposed method misclassifies almost no edges.

Figure 3.5: Comparison of FAST-feature extraction on FPSP device before and after the proposed refinements. The two methods are tested with the same configuration parameters (200 FPS with 55 as threshold).

and effective, it uses up 8 out of 13 available digital registers just for the counters, which leaves very few, if any digital registers for long term storage.

Furthermore, unlike the standard implementation of the FAST corner detection [46], the comparison function is slightly modified.

$$S_{p\to x} = \begin{cases} \textit{darker}, & I_{p\to x} \leq I_p - t \\ \textit{similar}, & I_p - t < I_{p\to x} < I_p + t \\ \textit{brighter}, & I_p + t \leq I_{p\to x} \end{cases} \tag{3.1}$$

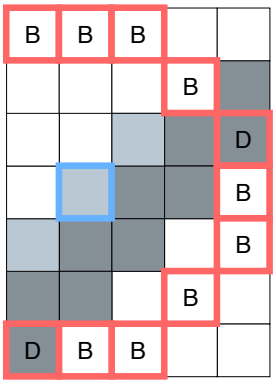
$$S_{p\to x} = \begin{cases} \textit{similar}, & |I_{p\to x} - I_p| \leq t \\ \textit{different}, & |I_{p\to x} - I_p| < t \end{cases} \tag{3.2}$$

Equation (3.1) shows how the states are assigned to the pixels in a ring in the standard implementation of FAST corner detection. Equation (3.2) is the formulation implemented for the FPSP device. While the modification speeds up the computation, it makes the detector highly sensitive to edges, especially when the edges are at an angle. The comparison function causes the increase in the sensitivity, as it does not distinguish between the *darker* and *brighter* states. The interleaving of the states allows the features to be detected along edges, as visualised in Figure 3.5.

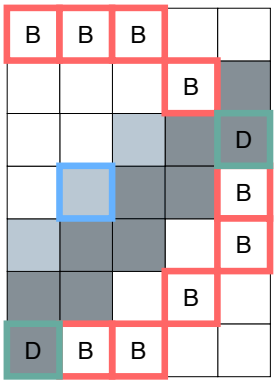
An erosion process, which reduces the number of features detected around a single visual corner, is implemented. However, the process is based on binary comparisons of the features extracted, and hence, the selection process of the features is arbitrary, as when the device is slightly moved, it will select a different feature out of the blob of features.

3.2.3 Comparison Function

As shown in Figure 3.5a, Equation (3.2) suffers from being overly sensitive to the edges. Instead, we follow the Equation (3.1) by performing two-pass feature extraction. The first pass checks



(a) Comparison function Equation (3.2). As the comparison does not distinguish brighter and darker state, the point tested will be wrongly marked as a corner



(b) Comparison function Equation (3.1). Since there is no 9 continuous block of same state in the ring, the point tested will correctly not be classified as a corner.

Figure 3.6: Demonstration of increase in sensitivity of feature detection around an edge. The pixel tested is marked with a blue border and brighter/darker state of the ring is indicated as B/D respectively. The border colour of the pixels in the ring illustrates the continuity of the ring.

for the contiguous *darker* states, and the second pass check for the contiguous *brighter* states. This reduces the sensitivity against the edges as illustrated in Figure 3.6, as it does not allow interleaved brighter/darker states to count towards the contiguous pixels.

3.2.4 Binary Counter

While using a ring counter is simple yet effective, with limited resources available on the FPSP device, it is important to be conservative with the number of registers being used. The availability of the extra registers allows the implementation of the two-pass comparison function, as additional registers are used to store the result from the preceding pass.

In the FAST-corner algorithm, a pixel is marked as a feature if N-contiguous pixels have the same state. Typically N is set to 9 or 12. This motivates us to use only 4 digital registers since a 4-bit counter can count up to 15. Using the logical operations available on the FPSP device, it is possible to create a general purpose counter, as shown in Algorithm 3. Extending the number of bits for the counter is trivial, and it could be useful if an application requires long term storage of a large value.

Note as documented in [1], *NIMP(A, B)* is $\neg(B \implies A)$ not $\neg(A \implies B)$.

3.2.5 Edge Detection

The corners extracted using FAST corner detector is further refined through use of edge detector [6]. Adding a feature detector can be computationally expensive for CPU based implementation. However, for the FPSP devices, by the nature of SIMD-based systems, it only adds a little overhead.

The intuition behind combining the edge detection is that, often, the corner points are generated when two or more edges intersect. Thus, by combining this information with the FAST corner detector, it is possible to reduce the number of false positives, where a noisy pixel is classified as

Algorithm 3 4-bit binary counter. Can increment or reset the counter.

```
1:  procedure BINARY COUNTER
2:      if RESET then
3:          FLAG ← 0
4:      else
5:          FLAG ← 1
6:      if FLAG = 0 then
7:          R1, R2, R3, R4 ← 0
8:      R7 ← FLAG
9:      R6 ← XOR(R4, R7)
10:     R4 ← R6
11:     R6 ← NIMP(R4, R7)
12:     R7 ← R6
13:     Similarly update R3, R2, R1 by repeating line 9-12
```

a corner. This can be seen in Figure 3.8. We have used an existing implementation of the edge detector, which is available for Scamp5d devices [1].

3.2.6 Two Ring Detector

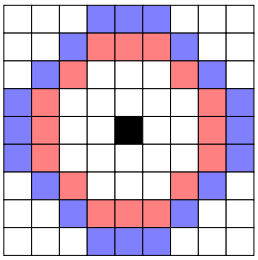


Figure 3.7: Two ring of radius 3, 4 is used to test whether pixel selected is a feature.

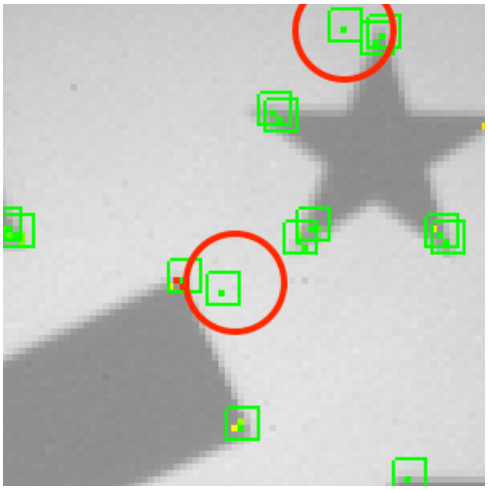
We use two-ring FAST corner detector to filter the features, as illustrated in Figure 3.7. The outer-ring, while it does not provide a good localisation, it is reliable as it is less sensitive to the sensor noise. On the other hand, the inner ring, while it is sensitive to the sensor noise, it provides a good localisation [38]. Through the combination of the two feature extractors, we can find features which are both well localised and is reliable.
This process reduces the features extracted per visual corner, as shown in Figure 3.9. We have used $N = 9$ for the inner ring and $N = 12$ for the outer ring, both of which is countable on our 4-bit counter.

3.2.7 Non-maximal Suppression

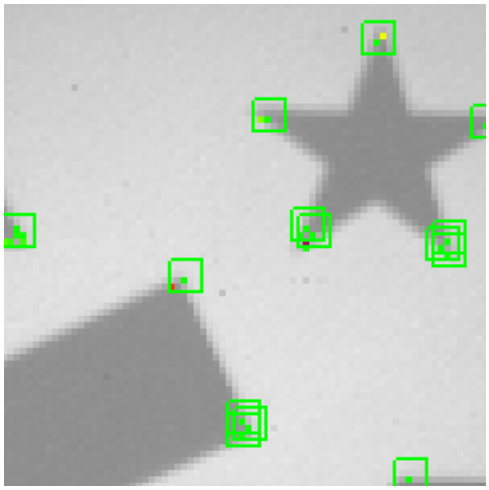
Whilst applying the above refinements improves the quality of the extracted features, multiple features are often extracted per visual corners. The multiple features extracted for a visual corner gives very little if any information and makes task such as tracking more difficult, as they all contain similar local information. Thus, they are often reduced into one feature using different strategies.

To improve the repeatability of the feature extraction, the choice of strategy is essential. Similar to the FAST corner detector [46], we have implemented a non-maximal suppression with 3×3 mask. Compared to the erosion process of the prior work, non-maximal suppression uses the analog information, which allows an informed decision to be made about the selection of features as shown in Figure 3.10

For the non-maximal suppression, the score for each feature must be defined. We define the

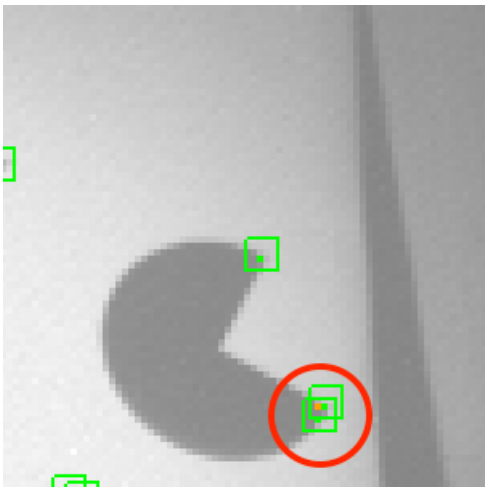


(a) Without edge filter. The noise in the image is classified as feature.

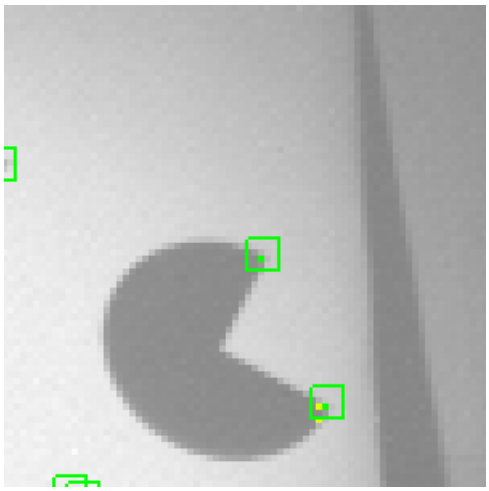


(b) With edge filter. The noisy features are removed.

Figure 3.8: Comparison of with and without edge filter.



(a) Without two ring test. Multiple features are found near the sharp corner of circular sector.



(b) With two ring test. There is reduced amount features per visual corner.

Figure 3.9: Comparison of with and without two ring method.

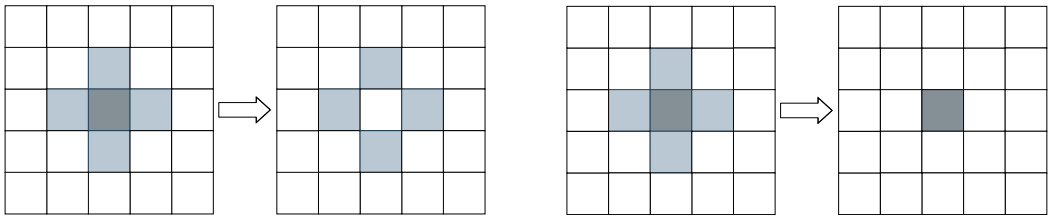
score function to be

$$V(p) = \sum_{x \in X_p} |I_x - I_p| \tag{3.3}$$

Where p is the pixel which the score is computed for and X_p is the pixels which belongs to the inner ring. We chose to use the inner ring, as the value range on the analog register is limited. This score function allows the computation to happen along-side inner-ring check.

3.2.8 Summary of the Improved Feature Detection on an FPSP Device

We have improved upon the existing feature detection algorithm implemented for an FPSP device. Due to the addition of additional instructions, the frame-rate of feature extraction has reduced from 2300FPS to 600FPS, however, for most practical applications 600FPS is sufficiently fast. The improvements which we have made allow the features which are located close to a



(a) Erosion process implemented by binary comparison only. Colour of the pixel represents score value and darker pixels have higher score. Binary comparison based erosion ignores this information and may remove feature with highest score.

(b) Erosion using non-maximal suppression. The score function is used to erode away detected features with low score.

Figure 3.10: Comparison of binary-based and analog-based erosion methods

Algorithm 4 Improved feature detector for an FPSP device.

procedure FEATURE DETECTOR
  For a pixel p, compute *Score* using Equation (3.3)
  InnerRingBrighter ← Has 9 contiguous Brighter state in the inner ring
  InnerRingDarker ← Has 9 contiguous Darker state in the inner ring
  InnerRing ← *InnerRingBrighter* ∨ *InnerRingDarker*
  OuterRingBrighter ← Has 12 contiguous Brighter state in the outer ring
  OuterRingDarker ← Has 12 contiguous Darker state in the outer ring
  OuterRing ← OuterRingBrighter ∨ *OuterRingDarker*
  IsEdge ← p lies on an edge detect by binary edge filter
  MaybeFeature ← *InnerRing* ∧ *OuterRing* ∧ *IsEdge*
  for *NeighbourScore* ∈ *NeighbouringFeaturesScore* **do**
    if *Score* < *NeighbourScore* **then**
      MaybeFeature ← *False*
  IsFeature ← *MaybeFeature*

visual corner to be extracted, which makes the later algorithms, which uses the feature extracted to operate more reliably. We summarise the outline of the algorithm in Algorithm 4.

One of the limitations of our feature detector is the illumination in the scene. With a high frame-rate, the scene must be well-lit such that the image contains distinct textures. To reduce the influence of the illumination, several methods such as increasing the frame-gain or extracting features from a gradient image was attempted. However, these methods amplified noise, reducing the reliability of our feature extractor. This limitation, unfortunately, limits the usability of our feature extractor. For high frame-rate feature extraction (e.g. 600FPS), a sunlit environment or indoor environment with lighting equipment is required. However, for lower frame-rates, such as 150-200FPS, given the lights in the room is turned on, it provides sufficient illumination for the feature extraction.

3.3 Feature Tracking

This section presents a method for tracking features detected on the FPSP device. First, we evaluate the noise present in our feature extraction method and introduce a probabilistic filter to reduce the influence of noise.

3.3.1 Motivation

Feature tracking associates correspondence to the extracted features, which allows many computer vision algorithms to combine information from multiple images. For instance, one can use such information to perform tasks such as image stitching [52]. While the field of feature tracking is well studied, most of the algorithms are designed for the frame-based camera. Such algorithms inherently suffer motion blur caused by the rapid motion.

To reduce the effect of motion blurs, the use of event camera has been explored. However, the methods suffer from severe limitations, for example, lack of ability to track features when there is motion along the optical axis [23], or real-time computation is not possible [61].

Compared to the event camera, the FPSP device transfers significantly less information, as it allows the computation to occur on the focal-plane. This reduces the computational burden for the host device, making the FPSP device promising solution for high frame-rate feature tracking, which does not impose restrictions on the motion and achieves real-time performance.

3.3.2 Noise

Due to the nature of FPSP device, there is a substantial amount of noise present in the system. This affects our feature detector negatively, as it results in noisy readings.

To understand how much our feature detector is affected by the presence of noise, we have carried out the following experiment. For each of the different parameters tested, we have executed our feature detector over 10,000 frames, in a well-lit environment with highly textured objects. The FPSP device was mounted onto a tripod, such that it is securely positioned. For each of the frames, we compared the features which were extracted in the previous frame to the current frame and computed the percentage of the features missing using

$$1 - \frac{|F_t \cap F_{t-1}|}{|F_t|} \tag{3.4}$$

Where F_t is set of features extracted at frame t. To compute the intersection of the features, we create a correspondence if the Manhattan distance between the two features is smaller than 2.

As shown in Figure 3.11, we observe that under different setting, the average probability of features missing does not vary largely. The average probability of features missing across all the run is 0.0483, which is large as after 100 frames, the chance of a feature being observed for all of the frames is below 0.01. This motivates us to develop a model which is strong against features not being observed between consecutive frames.

3.3.3 Problem Formulation

Unlike in a standard frame-based camera where pixel intensity for each pixel is transferred, our system receives only the position of features. Hence, methods such as Lucas-Kanade are not applicable here. In an ideal scenario with no noise, as proposed by Chen et al. it is possible to perform closest neighbour search for feature matching. However, in a realistic situation with noise, features can be missing in consecutive frames leading to wrong correspondences.

Instead, we view the problem of feature matching from a probabilistic angle. For simplicity, we will model the problem as tracking of one feature x given observation of multiple features. However, the reasoning is easy to extend to the tracking of multiple features.

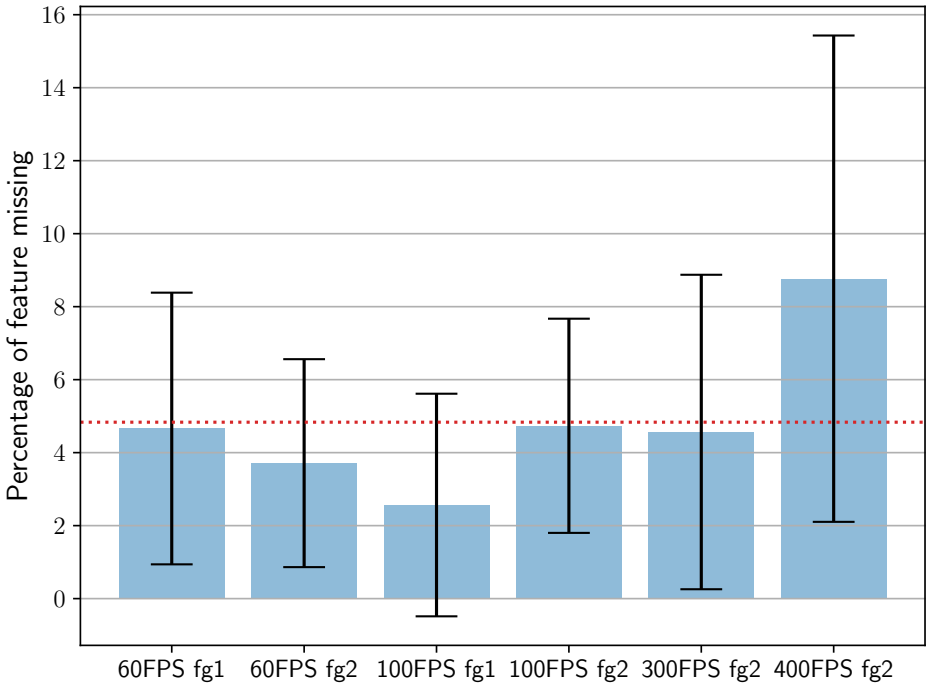


Figure 3.11: The average percentages of features missing between consecutive frames. The error bar represents standard deviation. The mean percentage across all the runs were 4.83%, as shown with a red dotted line. The only configurable parameters adjusted between the runs were frame-rate (FPS) and frame-gain (fg).

Let set of observations be $D_k = \{y_1, \ldots, y_k\}$, where y_k represents observed features at time k. Given such information, most plausible estimate of location of x_k is obtained through solving Equation (3.5).

$$x_k = \arg\max p(x_k \mid x_{k-1}, D_k) \tag{3.5}$$

Using a model for state transition $\tilde{x}_k = f(x_{k-1})$, initial prediction of x_k can be obtained from the previous state x_{k-1}. Using this as the initial guess, prediction of x_k is refined by solving Equation (3.5), which now can be formulated as Equation (3.7) using Bayes' Rule.

$$p(x_k \mid D_k) \propto p(y_k \mid \tilde{x}_k) p(\tilde{x}_k \mid D_{k-1}) \tag{3.6}$$

$$x_k = \arg\max p(y_k \mid \tilde{x}_k) p(\tilde{x}_k \mid D_{k-1}) \tag{3.7}$$

3.3.4 Particle Filter

A particle filter is a simple method for nonlinear state estimation [24]. The distribution of the state is modelled using particles, and increasing the number of particles improves the estimation at the cost of computational time.

Use of a particle filter allows estimation of a solution to the Equation (3.5). For each of the features, there are N random samples $\{x_k(1), \ldots, x_k(N)\}$ drawn from a probability density function $p(x_{k-1} \mid D_{k-1})$. Each of these samples are weighted equally with $w_i = 1/N$.

A particle filter performs state estimation using two steps, Prediction and Update. In the Prediction step, state transition model is applied to each of the particles. $\tilde{x}_k(i) = f(x_{k-1}(i))$. The state transition model may incorporate if any, known information such as system inputs to model motions of the particles.

In the Update step, measurements y_k are used to compute $p(y_k \mid x_k)$. Using the Equation (3.7) posterior for each of the particles can be computed as $w_i = w_i \cdot p(y_k \mid x_k(i))$. Resampling is performed N times, where each particle i have $w_i / \sum_{j=1}^{N} w_j$ chance of being picked. This allows the distribution of the particles to approximately follow $p(x_k \mid D_k)$.

The estimated state can be obtained either by averaging of the multiple estimates or by picking a mode of the particles.

A particle filter is effective for our application since the high-frequency observation of the features means that the displacement of each feature is small between the consecutive frames. This means that the space of probability density function $p(x_k \mid D_k)$ modelled is small and with little number particles, we can sufficiently express them

3.3.5 Implementation Details

Our feature tracker is based on the ideas from the particle filter, with modifications to handle the corner cases more effectively.

A zero-mean Gaussian is assumed for the state transition model of the features. From the high frame-rate nature of the FPSP devices, it is a valid assumption that the inter-frame motion of the observed features are small and is approximately zero.

In the update phases where the computation of $p(y_k \mid x_k)$ occurs, a local space around each of the particles is searched to find the shortest distance d from a particle to an observation. We model the distribution of $p(y_k \mid x_k)$ with Equation (3.8), which is a zero-mean Gaussian with added K to prevent good particles from all dying when there is bad observation [53].

$$p(y_k \mid x_k(i)) = \exp\left(\frac{-d^2}{2\sigma^2}\right) + K \tag{3.8}$$

Where $\sigma^2 = 1.0$ and $K = 0.01$ in our implementation.

Use of a particle filter based approach for feature tracking has downsides too. For example, if a wrong correspondence is made, the particles would favour them during the resampling stage. Once a wrong correspondence is established, it is difficult to remove them. Furthermore, when a feature goes out of the visible range of the device, the feature should no longer be tracked. The features could be missing for multiple reasons. (a) The movements of the device could have resulted in the feature leaving the field of view, (b) Other objects could occlude the feature.

(a) is simple to address, as one can stop tracking features which are close to the edges of the image plane. However, in a case where the feature is close to an edge but never leaves the image plane, it is undesirable that this feature is no longer tracked. (b) is harder to handle, as it could occur anywhere on the image plane.

To handle these problems, we use simple heuristics and measure the variance of the particles. We call this step *pruning phase*, where a feature is marked as lost if its particles variance exceeds a certain threshold.

The likelihood of wrong correspondence being established increases when a feature is not observed for multiple successive frames. This is due to the wider spread of the particles, which increases the chance of the particles reaching a wrong feature. Thus, removing the feature if its particles variance exceeds a threshold reduces the number of wrong correspondences. Furthermore, when a feature is no longer observable, whether it is caused by the feature leaving the field of view of the camera, or other objects occlude it, the features particles variance will increase. Hence, the

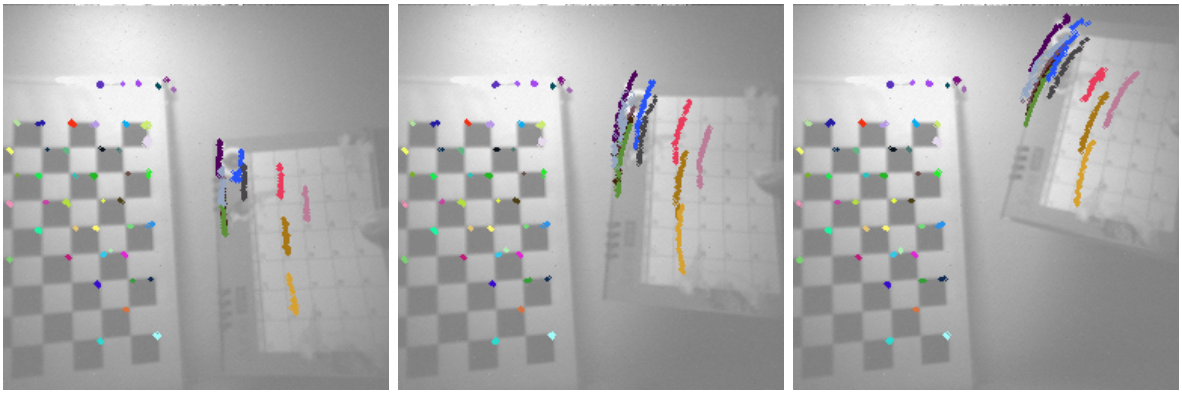


Figure 3.12: Snapshots of feature tracking in a dynamic scene. The consistent colours of the features indicate successful tracking. The images are transferred from the FPSP device for visualisation purpose only.

pruning phase prevents the feature tracker from attempting to track features which are out of view.

Consideration of other alternative approaches, such as to count the number of consecutive times that a feature is not observed was considered. However, the definition of observation of a feature is ambiguous, especially when the feature trackers non-maximal suppression occurs at a small scale. Under an assumption that the features are distributed uniformly, when a feature makes a wrong correspondence, it would likely make correspondences with multiple features. The approach of counting the number of consecutive times that a feature is not observed would fail to detect this wrong correspondence. On the contrary, the variance increases in a multi-modal distribution, allowing the pruning phase to identify the wrong correspondence.

3.3.6 Summary of the Feature Tracking

We have implemented a probabilistic feature tracker, which operates on noisy data produced by the FPSP device. Efforts were made to reduce the numbers of wrong correspondences, however, if one is made, our feature tracker struggles to identify them. This is due to the lack of information which we obtain from the FPSP device. Given more information, we might be able to detect such errors. For example, by computing the sparse optical flow or the base vector fields [18], and assuming that the nearby features move in a similar motion, we can filter out the wrong correspondences which do not coincide with the flow.

Through the use of a particle filter, we have not imposed any restrictions on the motion of the features. This allows features to be tracked in a dynamic scene, as demonstrated in Figure 3.12, where the calendar and the checkerboard undergoes a different motion. Support for dynamic scenes opens up a wide range of possibilities for the feature-tracker since a typical scene in our daily life contains many dynamic motions.

3.4 Summary

Overall, we have analysed the communication between the FPSP device and the host device and implemented a feature detector and tracker for the device. The analysis of the communication under-covered some of its limitations and the need for a dedicated thread for communication purpose. Our feature detector extends on the existing implementation of FAST corner detector

through the ensembling of techniques and heuristics from many works of literature, showing significant improvements in the result. Finally, for the feature tracker which operates on noisy features from the FPSP device, we are not aware of any other successful attempt. We have utilised the high frame-rate nature of the FPSP device, which allows feature tracking to occur at 600FPS in real-time. Furthermore, the underlying particle filter does not restrict the motion of the scene, hence tracking of the non-static scene is also possible. We hope that our work opens up new possibilities for FPSP devices and allow the development of many complex computer vision algorithms which utilises the device.

In the following chapter, we will use our implementation of the feature extractor and tracker to demonstrate its capability and perform 6DoF visual odometry which has never to the best of our knowledge been implemented for the device before.

Chapter 4

Visual Odometry

In this chapter, we will discuss the methods used to implement a low latency 6DoF visual odometry using data from an FPSP device. We will perform comparisons of the different pose estimation approaches, and discuss why we decided on our final pipeline, together with the challenges which we faced.

Low latency visual odometry is an active field of research [8, 18, 33] and has many practical applications such as Virtual Reality. Use of FPSP devices in visual odometry is promising as it can process the pixel information on the chip itself. By reducing the amount of data to be transferred, the device is energy efficient while promising high frame-rate.

4.1 Initial Attempts

In this section, we will cover the initial attempts which were made to develop the visual odometry pipeline. We used C++ for all of our implementations as the Scamp5 interface library only exposed C++ API.

4.1.1 Motivation

Before implementing a visual odometry pipeline for an FPSP device, we have developed a pipeline for a standard frame-based camera. This approach is beneficial as it removes uncertainties in case of errors. If the algorithm operates with the frame-based dataset, the cause of the error is likely to lie in the algorithm for the FPSP device and vice-versa.

In this section, we will discuss some of the early attempts made, and how we decided on our final visual odometry pipeline.

4.1.2 Pose Representations

In our implementation, the 6DoF pose is expressed using a 3D translation and a unit quaternion (7 parameters). While it is not a minimal representation compared to a 3D translation plus a yaw-pitch-roll, or angle-axis (6 parameters), it does not suffer from degenerate cases caused by Gimbal Lock [7] or non-unique representation [51]. For convenience, we use Sophus [1] to abstract away the mathematical manipulations.

[1] Sophus: `https://github.com/strasdat/Sophus`

4.1.3 Visualisation

Visualisation is important for visual odometry as it provides a good indication of how well the system is performing. We have used Pangolin [2] to implement our visualiser. It provides a simple interface to manage the OpenGL display and interactions. Initially, we used the OpenCV Viz module. However, it required OpenCV to be built from a source, which limited the portability of the application.

4.1.4 Implementation Details

Our first implementation of the visual odometry pipeline was based on the 2D-2D correspondences. This approach avoids the use of a 3D map simplifying the problem. Initially, we performed pose-estimation using the 5-point algorithm [40]. However, this did not work well. The issue arises from the small base-line distance between the frames. Skipping several frames did help, though the number of frames to skip was arbitrary, which made the system unstable.

To overcome this limitation, ideas from ORB-SLAM [39] was used. Similar to their initialisation algorithm, homography and fundamental matrix were computed, and one was chosen based on how well it explained the data.

The 8-point algorithm computes the fundamental matrix and is accurate if the scene is non-planar, and there is sufficient parallax. On the other hand, homography explains the motion when there are little parallax or the scene is planar. A choice between the two models is made based on the robust heuristic as suggested by ORM-SLAM.

$$R_H = \frac{S_H}{S_H + S_F} \tag{4.1}$$

Score takes into account of both forward (M) and backward transformation (M^{-1}) through the use of symmetric transfer error [27].

$$d^2_{transfer}(x, x', M) = d(x, M^{-1}x')^2 + d(x', Mx)^2 \tag{4.2}$$

Where x, x' is the corresponding points and d is a function to compute Euclidean distance between the two given points.

The score function S_M for each models M (H for the homography, F for the fundamental matrix) is defined as

$$S_M = \sum_i \left(\rho_M \left(d^2_{transfer} \left(\mathbf{x}_i, \mathbf{x}'_i, M \right) \right) + \rho_M \left(d^2_{transfer} \left(\mathbf{x}'_i, \mathbf{x}_i, M \right) \right) \right) \tag{4.3}$$

Where outlier rejection function ρ_M is defined to be

$$\rho_M \left(d^2 \right) = \begin{cases} T_H - d^2 & \text{if} \quad d^2 < T_M \\ 0 & \text{if} \quad d^2 \geq T_M \end{cases} \tag{4.4}$$

The values for T_M is set to $T_H = 5.99$ and $T_F = 3.84$ based to χ^2 test at 95% assuming standard deviation of 1 pixel. If $R_H > 0.45$ the homography is chosen. Otherwise, the fundamental matrix is selected.

Use of hypothesis selection did remove the need for skipping of the frames, however, as the approach is a 2D-2D method, we can only recover the direction of the translation. Recovery of only the direction of the translation means that the scale of the translation is unknown. Thus, between the frames, the scaling is inconsistent. Inconsistency in the scale causes ambiguity in the scaling of the trajectory at a frame level, which causes a significant error to build up. The error distorts our trajectory, causing its shape not to match the actual trajectory.

[2] Pangolin: https://github.com/stevenlovegrove/Pangolin

4.1.5 Summary of the Initial Attempts

We have familiarised with some of the visual odometry methodologies and explored a few of the different 2D-2D methods. We have set up the basic visualisation and established the pose representation which we are going to use in the development of our visual odometry pipeline. Several 2D-2D methods were explored, and we have understood their limitations. Taking these into consideration, we will create our full visual odometry pipeline which does not suffer from inter-frame scale ambiguity.

4.2 Visual Odometry Pipeline

In this section, we finalise the visual odometry pipeline, which we have designed for the frame-based camera. Similar to many of the visual SLAM pipeline [32, 39], we decided to use 3D-2D methods for the pose estimation. This section covers the different key components implemented to create the pipeline.

4.2.1 Motivation

Finalising our visual odometry pipeline allows us to build a solid foundation, where we can build the visual odometry using an FPSP device on top. We chose the 3D-2D method as it is resilient against the inter-frame scale ambiguity. This allows our trajectory to be representative of the real trajectory up to an unknown scale.

4.2.2 Initialisation

In monocular visual odometry, the initialisation of a 3D map is an interesting chicken and egg problem. The 3D-2D methods require the 3D map to estimate the poses, and to create the map, at least 2 positions must be known in advance. An arbitrary choice about the origin of the world-frame can be made, which allows the initial camera position to be defined at the origin. This simplifies the problem as now, only the relative motion estimate is required.

To solve the initialisation problem, we follow a similar approach to ORB-SLAM and perform bootstrapping using a 2D-2D method. As discussed in Section 4.1.4, ORB-SLAM computes both the fundamental matrix and the homography and handles planar and non-planar cases separately. Instead, we use 5-point algorithm [40] to compute the essential matrix, as the essential matrix is not degenerate against planar-cases.

One problem with using the 5-point algorithm for bootstrapping is that if there is a small base-line distance between the frames, the pose estimation produces the wrong hypothesis. However, through observations, we noticed that when the pose estimation is poor, the triangulation of the features is poor too. This allows us to handle the small base-line distance through rejection of the initialisation when the quality of the triangulation is poor.

We assess the quality of the triangulation based on the following.

1. Only the points which are triangulated in front of the cameras of two poses are considered inliers.
2. The average angle between the two cameras and triangulated points is greater than 10 degrees.
3. The average reprojection error is below 3 pixels for both cameras.

4. At least 50 inlier points are triangulated.

Once all the conditions are met, the map is initialised, and the rest of the pipeline starts. While the initialisation process is fully automatic, we must have a transitional motion to perform reliable triangulation.

Our 3D map does not store information such as local patch around a feature or descriptors. In PTAM and ORB-SLAM, this information is stored in the 3D map to associate a feature detected with a 3D point, However, as our feature detector for the FPSP device does not transfer such information, we avoid the use of the information in our implementation. This helps us use the pipeline together with the FPSP device without significant modifications.

4.2.3 Pose Estimation

Given a 3D map and its corresponding points on a 2D image plane, solving PnP problem allows the recovery of the position and orientation of the camera in 3D space. OpenCV [9] provides function `solvePnPRansac()` which solves PnP problem using RANSAC to filter out the outlying data. The filtered out data can be used to clean up the 3D map, by removing the mapped point from the set if it is part of the outliers. To solve the PnP problem, it is essential that a good initial guess is given. Thus, we use the previous pose estimate as an initial guess.

4.2.4 Map-point Insertion

As the system explores the new regions in the space and detects new features, insertion of the new features into the 3D map is necessary. However, the triangulation process requires sufficient base-line distance, and when there is insufficient distance, poor quality triangulation may damage the 3D map. While one can wait for a couple of frames until the triangulation, inserting the points as fast as possible is essential as tracking features over a long period is difficult.

We tackle this problem using a simple method [43]. All of the features which are tracked but not triangulated are stored as candidate points in a list. Insertion to the list occurs when a feature which is not triangulated is detected. The image plane coordinate of the feature and the pose estimate of the current frame is inserted as a new candidate point. For every pose update, the list is traversed to check if any candidate point has small enough reprojection error and large enough parallax. The similar criteria as the map initialisation are used to verify if the triangulation is successful. The candidate points are removed from the list upon successful triangulation, or if the feature is no longer tracked.

4.2.5 Summary of the Visual Odometry Pipeline

We have established a simple pipeline which is required for full visual odometry on frame-based camera. Our method uses a 3D-2D method to reduce the effect of inter-frame scale ambiguity. While features such as recovering from error are not implemented, we have a sufficiently good pipeline which we can extend on. We can qualitatively see the effectiveness of our pipeline in Figure 4.1. The camera moves in a straight horizontal motion in the dataset [2]. While the motion is simple, the example is challenging as features go out of view as the camera moves, meaning insertion of new map point must be successful for the tracking to work. Our pipeline, although there is a slight curve, successfully track its pose while inserting newly observed features. Furthermore, our pipeline can comfortably operate at 30FPS. We will discuss the limitations of our pipeline when used in conjunction with the data from an FPSP device in the next section.

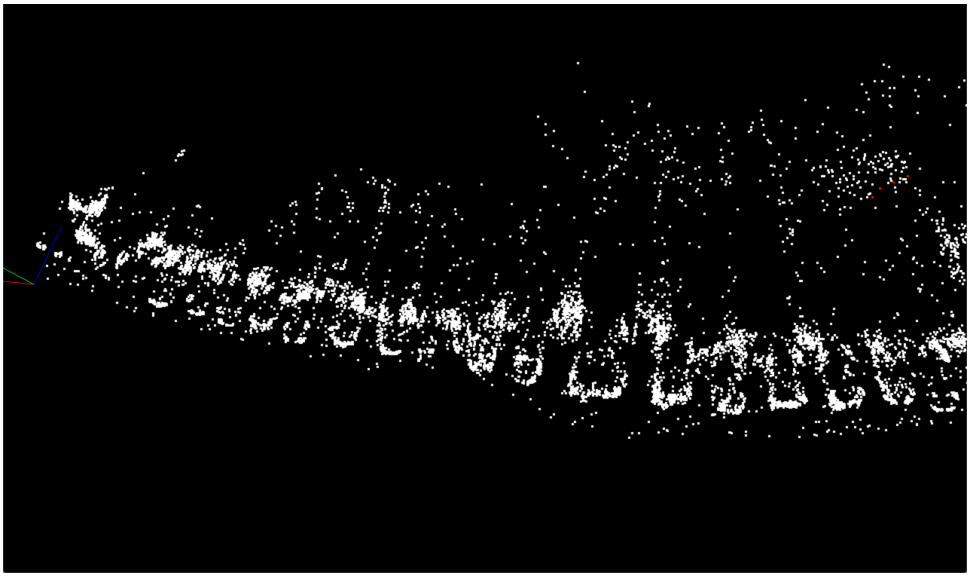


Figure 4.1: Visualisation of the 3D map created by the frame-camera pipeline.

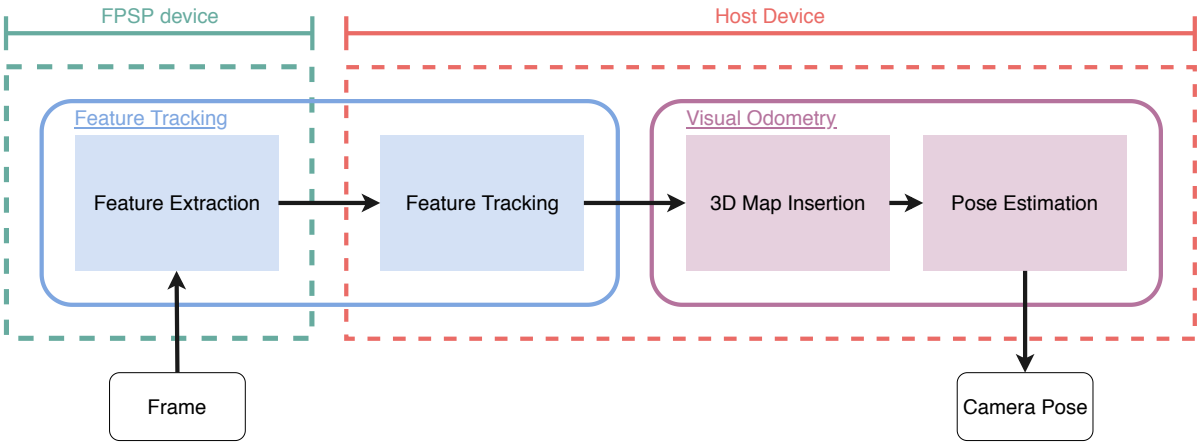


Figure 4.2: Illustration of how the visual odometry pipeline is separated.

.

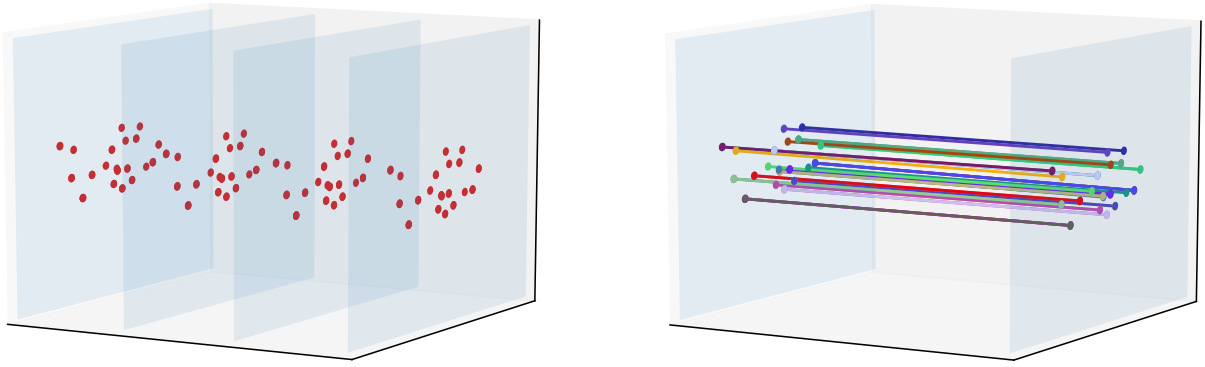
4.3 Low Latency Visual Odometry

In this section, we will discuss the implementations required to connect our visual odometry pipeline with the data from an FPSP device. Through the use of parallel processing, both on the FPSP and the host device, we present a low latency visual odometry pipeline which can operate at 250FPS.

4.3.1 Motivation

Previous to our work, there have been methods proposed to perform 4DoF tracking on an FPSP device [8, 18]. However, these methods required the development of new algorithms which are specific to the FPSP device, as the resources and the instruction sets available are limited. This disallows the use of the well-established works from the visual odometry on a frame-based camera, slowing down the development process.

Instead, we propose a method which uses a sparse data from the FPSP device and executes a



(a) Buffering of the feature data from an FPSP device

(b) Correspondences established using our feature tracker.

Figure 4.3: Visualisation of the buffering of the incoming features.

well known visual odometry algorithm which was initially designed for a frame-based camera. Demonstration of the ability to combine classical methods with the FPSP device opens up many new possibilities, unlocking the full advantage of what the FPSP device has to offer while working on top of many of the existing researches.

4.3.2 Preparations

To use the data from an FPSP device, setting up the communication between devices is necessary. Recall in Section 3.1, where we have discussed the limitations of the communication between an FPSP and a host device. We must dedicate a thread to communicate with the FPSP device such that our features are reliably extracted. The rest of the pipeline communicates with this thread through a locked object.

Our visual odometry operates under an assumption that the camera is calibrated. Hence the intrinsic parameters of the FPSP device must be obtained. Unlike DVS, as the FPSP device is programmable, it is possible to define an identity function and let the device simply acts like a grayscale frame-based camera. This allows the use of standard calibration processes, in particular, we obtain the parameters using OpenCV's `calibrateCamera()` with a checkerboard. Note that since it is possible to send information from a host device to an FPSP device, a simple branching in the code of the FPSP device allows us to switch the function it is executing from feature extraction to an identity function, removing the necessity to reprogram the device every time we wish to recalibrate.

In our previous implementations of visual odometry pipeline, we have established the correspondence of the features across frames using descriptors. We replaced this with our particle filter based feature tracker, which uses the features extracted from the FPSP device. As illustrated in Figure 4.2, we perform the feature extraction on the FPSP device and the rest of the operations on the host device.

An iteration of our pose estimation pipeline takes longer when compared to the time required to extract features on an FPSP device. Let the time it takes for an iteration of our pipeline be t_{vo}, and the time it takes for extraction of features from a single frame be t_f. Let $t_{vo} = n \cdot t_f$ where $\lfloor n \rfloor$ is the number of frames the FPSP device has extracted features from in the duration of one iteration of the pose estimation. If we simply processed a frame at a time, and $n > 1$, the unprocessed frames would accumulate, and the pipeline will not able to process the newest frames. This leads to high latency in our system. A simple solution is to limit the frequency

of feature extraction such that it coincides with the visual odometry pipeline. However, this dilutes the advantages of the FPSP device.

Instead, we use buffering as visualised in Figure 4.3. Multiple frames of features are stored in the buffer and are reduced into a set of a correspondence using our feature tracker. We will refer to the buffer of frames of features as *feature buffer* henceforth. Since the feature tracking occurs on features extracted at high frame-rate, our feature tracker is available to exploit the small inter-frame motion. The rest of the pipeline only uses these correspondences, allowing the visual odometry pipeline and the feature detection to operate at a different frequency. This, however, does mean that the overall latency of the pose-estimation is constrained by the visual odometry pipeline, which in our case is currently at 30FPS. In the later sections, we will propose methods to lower the latency.

4.3.3 Challenges

Using the data from the FPSP device is not trivial, especially for complex tasks such as visual odometry. We will list the main challenges we have faced.

1. Constant frequency of the feature extraction. As the features are extracted at a constant frequency regardless of the processing performed by rest of the pipeline, if the pipeline takes longer to remove the data from the feature buffer, the buffer is going to grow larger. When the feature buffer becomes large, the processes, in particular, the feature tracker is going to take longer to execute. This causes the feature buffer to grow even larger, putting even more stress on the system.
2. Accuracy of the feature tracker. Unfortunately, our feature tracker is not as accurate and contains noises when compared with descriptor matching methods. Furthermore, the errors in the feature tracker increase over time, and causes bad hypothesises to be generated for the pose estimation.

We immediately experienced the first challenge when the initialisation process did not terminate. The culprit behind the non-terminating process was the 5-point algorithm. The execution time of the algorithm was too long, which caused the feature buffer to build up. To avoid the buffer from building up, we throttle the execution of the 5-point algorithm. We have experimentally found that executing the 5-point algorithm once every 20 iterations reduced the stress on our system, and allows the pipeline to flow. For every iteration, feature tracker is executed to reduce the feature buffer to correspondences. However, since the feature tracker is computationally faster when compared to the 5-point algorithm, the size of the feature buffer decreases within the 20 iterations. Hence, the throttling of the 5-point algorithm acts like smoothing out of the spike in the buffer size. Furthermore, since we do not drop any frames, our feature tracker takes advantage of the high frame-rate nature of the FPSP device.

The second challenge imposed challenges to our pose estimation. Solving the PnP problem with RANSAC struggled to compute a satisfactory pose estimate. We have attempted multiple parameters and different algorithms available to solve the PnP problem, however, none of the approaches produced adequate pose estimation.

We believe that the error arises from the type of error which is present in our feature tracker. In a standard frame-based camera, performing a descriptor matching produces multiple correspondences which are very accurate, with few outliers. Applying RANSAC on such data allows the algorithm to find a subset of data which contains highly accurate correspondences. On the other hand, in our feature tracker, the correspondences suffer from slight error in tracking. The error is present due to the limited data which an FPSP device can transfer, and the accumulation over successive frames. Thus, the majority of the feature correspondences in our feature tracker

Methods	Rotation Error (Radian)			Translation Error			Time (μs)
	μ_c	$\tilde{c}$	σ_c	μ_r	$\tilde{r}$	σ_r	
Motion Only BA	0.000605	0.000569	0.000284	0.0777	0.0745	0.0364	5075
PnP Iterative	0.00216	0.00201	0.00114	0.322	0.304	0.146	66565
PnP P3P	0.00304	0.00287	0.00154	0.453	0.447	0.198	16138
PnP EPNP	0.0022	0.00204	0.00115	0.327	0.447	0.15	66052

Table 4.1: Evaluation of different methods with 100% noisy dataset. The table contains average, median, standard deviation of the rotation and translation errors and the average execution time.

Methods	Rotation Error (Radian)			Translation Error			Time (μs)
	μ_c	$\tilde{c}$	σ_c	μ_r	$\tilde{r}$	σ_r	
Motion Only BA	0.000138	0.000129	6.63e-05	0.0177	0.0167	0.00852	4058
PnP Iterative	7.34e-05	6.86e-05	4.07e-05	0.00856	0.00752	0.00489	1770
PnP P3P	8.01e-05	7.37e-05	4.32e-05	0.0104	0.00988	0.00511	534
PnP EPNP	8.58e-05	7.95e-05	4.79e-05	0.0113	0.00988	0.00597	749

Table 4.2: Evaluation of different methods with 10% noisy dataset. The table contains average, median, standard deviation of the rotation and translation errors and the average execution time.

contains some random noise. Applying RANSAC on such data does not yield a viable hypothesis as there may not be any subset which contains highly accurate correspondences.

To address the limitation, we have used motion only bundle adjustment [50]. The motion only bundle adjustment minimises the reprojection error of all of the features. To add robustness, we use Tukey M-estimator [60] as the loss function. Unlike the RANSAC approach, all of the features are used to generate the hypothesis, allowing the errors present in the feature tracking to average out. Furthermore, in noisy data, computation of motion only bundle adjustment is much faster when compared to the RANSAC approaches.

We have performed a comparison of the different methods with simulated data, which is 1000 randomly generated point cloud and its projection to a fixed pose close to the origin. All of the pose estimations use the origin as its initial guess. We artificially added random noise $z \sim \mathcal{N}(0, 1)$ to all the projected points. This simulates the errors which are present in our feature tracker. Table 4.1 summarises the result of the experiment. The motion only bundle adjustment outperforms the other methods in all rotation, translation errors and the execution time. This coincides with our hypothesis that the PnP with RANSAC only performs well when there is a subset of data which contains little error. To further verify our hypothesis, we conducted an experiment where only 10% of the data contains the artificial noise. As shown in Table 4.2, the PnP methods obtain higher accuracy and faster execution time compared to motion only bundle adjustment. However, in our scenario, we are likely to encounter noisy correspondences, thus, we opted to use the motion only bundle adjustment methods.

4.3.4 Implementation Details

Hand-tailored implementation of a non-linear optimisation method such as the Levenberg-Marquardt or Gauss-Newton method can be difficult and be time-consuming. Furthermore, manual derivation of the Jacobians is necessary, which is prone to error if one is not careful. We used Ceres Solver [4] to implement the motion only bundle adjustment. We chose Ceres Solver over other libraries such as g^{2}o [34] as we had previous experiences.

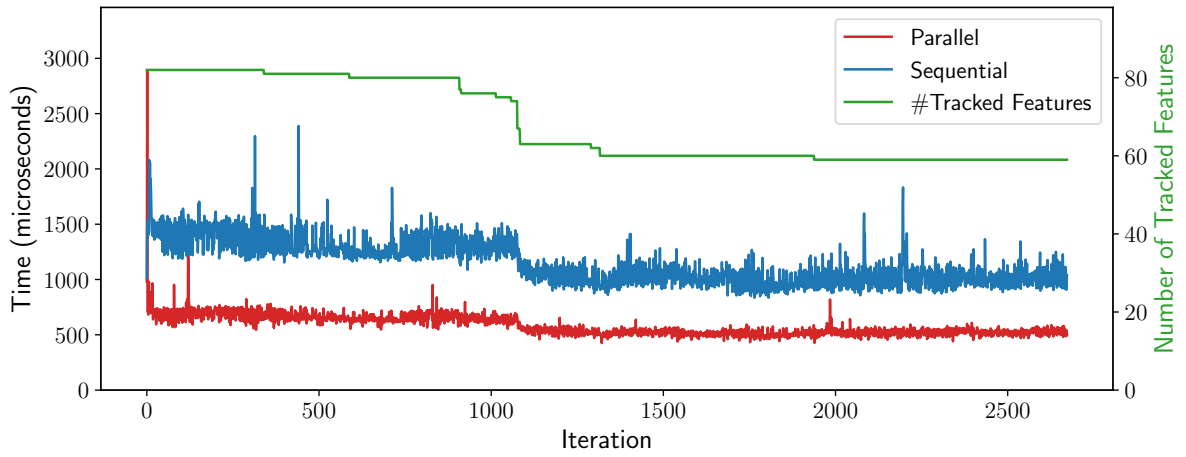


Figure 4.4: Comparison of execution time between the parallelised feature tracker and the sequential feature tracker.

For our pose representation, the rotation is expressed using a quaternion. While optimising over all 4 parameters works, however, a unit-length constraint of the quaternion must be enforced. This is wasteful as we are optimising over regions which violate the unit-length constraint. An alternative approach is to perform local parameterisation of the tangent plane to the unit sphere [27]. The local parameterisation reduces the dimension of the optimisation process, making the process more efficient as search is only performed in the feasible space. The local parameterisation of the quaternion is possible in Ceres Solver by using `QuaternionParameterization()`.

As motion only bundle adjustment is a non-linear optimisation process, the initial guess is important. Similar to PTAM, we use decaying velocity model, where the velocity reduces to zero if there are little motions.

$$\mathbf{v} = d \cdot (\alpha \mathbf{v} + (1 - \alpha)\mathbf{v}_{\mathbf{new}}) \tag{4.5}$$

d is the decay factor and α controls how much the new reading influences the velocity model. This helps the optimisation process to start with a better estimate, which allows for faster convergence. In our implementation, we have selected $\alpha = 0.5$ and $d = 0.9$.

Computation of the pose-estimate allows the reprojection error of currently tracked features to be computed. While our motion only bundle adjustment provides robustness against outlying data, build up of such data can cause a wrong hypothesis to be generated. Thus, reprojection error of all the tracked features are computed, and if the error is greater than 3 pixels, it is classified as an outlier and is removed. This reduces the amount of noise in the system and also lessens the processing required by the feature tracker.

4.3.5 Optimisations

Our implementation uses the data from the FPSP device, which extracts features at 250FPS. Even with all the modifications which we have introduced, the latency of the pipeline is around 60FPS. Although we use all of the data from the device by buffering, for latency critical application such as VR and autonomous vehicles, lower latency is necessary.

One of the most significant bottlenecks in our pipeline is the map insertion of the candidate points. For each iteration, we checked for all of the candidate points whether there is enough parallax to triangulate the points reliably, hence, the computational cost was linear to the

number of candidate points. Using ideas from PTAM, we use keyframes to insert the newly observed features. Only a subset of the frames is marked as keyframes, and the key-frames contains the features which are observed. If the current and previous keyframes contain the same features, and this feature has no corresponding 3D point, it is triangulated into the 3D map. This reduces the complexity as we are only comparing the current keyframe against the last keyframe. Selection of the keyframe is important as we want to triangulate the features as fast as we can while making sure that there is sufficient base-line translation such that quality of triangulation is good. The following criteria are used for keyframe selection.

1. More than 20 frames have been processed since the last keyframe insertion.
2. The pose distance between the last keyframe to the current frame is more than 10% of the median depth of the visible map.
3. The quality of the pose-estimation is good.

The pose distance is compared using the depth of the scene as if we are close to the object, a small translation would result in features moving out of the view. Similarly, if the depth of the scene is large, a small translation would not result in features going out of the view. The depth median is used instead of the mean to provide robustness against noisy 3D points, and we assess the quality of the pose estimation by comparing the reprojection errors, before and after the motion only bundle adjustment.

Another optimisation which we carried out was parallelisation of the feature tracker. Our particle filter based feature tracker does not share any state between the features tracked. Hence they can independently be tracked in a multi-threaded manner. We have used Threading Building Blocks (TBB) [3] to achieve the parallelism. This aids the throughput of the feature tracker and reduces the buffering of the extracted features. We have performed a comparison of the execution time between the parallel and sequential version of the feature tracker. As shown in Figure 4.4, parallelisation of the feature tracker has improved the computation time. We are not certain about the causes of the spikes in the measurements, however, the overall performance benefit is evident. On average the parallel and sequential feature tracker took 581.5μs and 1149.9μs respectively, resulting in 97.7% improvement in computational speed.

To further aid the feature tracker, we reduce the number of features which are tracked using grid/binning strategy used by [33]. When our pipeline inserts new features to be tracked, the image plane is divided into small 8×8 squares. For each of the square, we randomly sample a single feature and add them to the feature tracker.

These optimisations reduce the latency in the pipeline and allow the optimisation of the pose estimate to converge faster as the relative motion between the estimates is smaller. While the grid/binning strategy may cause a good feature to be removed, the optimisations above allow us to obtain low latency visual odometry, which can operate in 200-250FPS range.

4.3.6 Summary of the Low Latency Visual Odometry

We have discussed the challenges which we faced and the implementation details of our low latency visual odometry pipeline. We summarised the flow of our visual odometry pipeline in Figure 4.5. Operation at 200-250FPS is highly beneficial for applications such as VR and autonomous cars. Unfortunately, our implementation struggles to operate in large scenes as our feature trackers are not perfect and cause the 3D map to be slightly deformed. However, since our feature tracker can track features across multiple keyframes, use of multiple viewpoints is a promising direction to improve the accuracy of the triangulation.

[3] TBB: `https://github.com/intel/tbb`

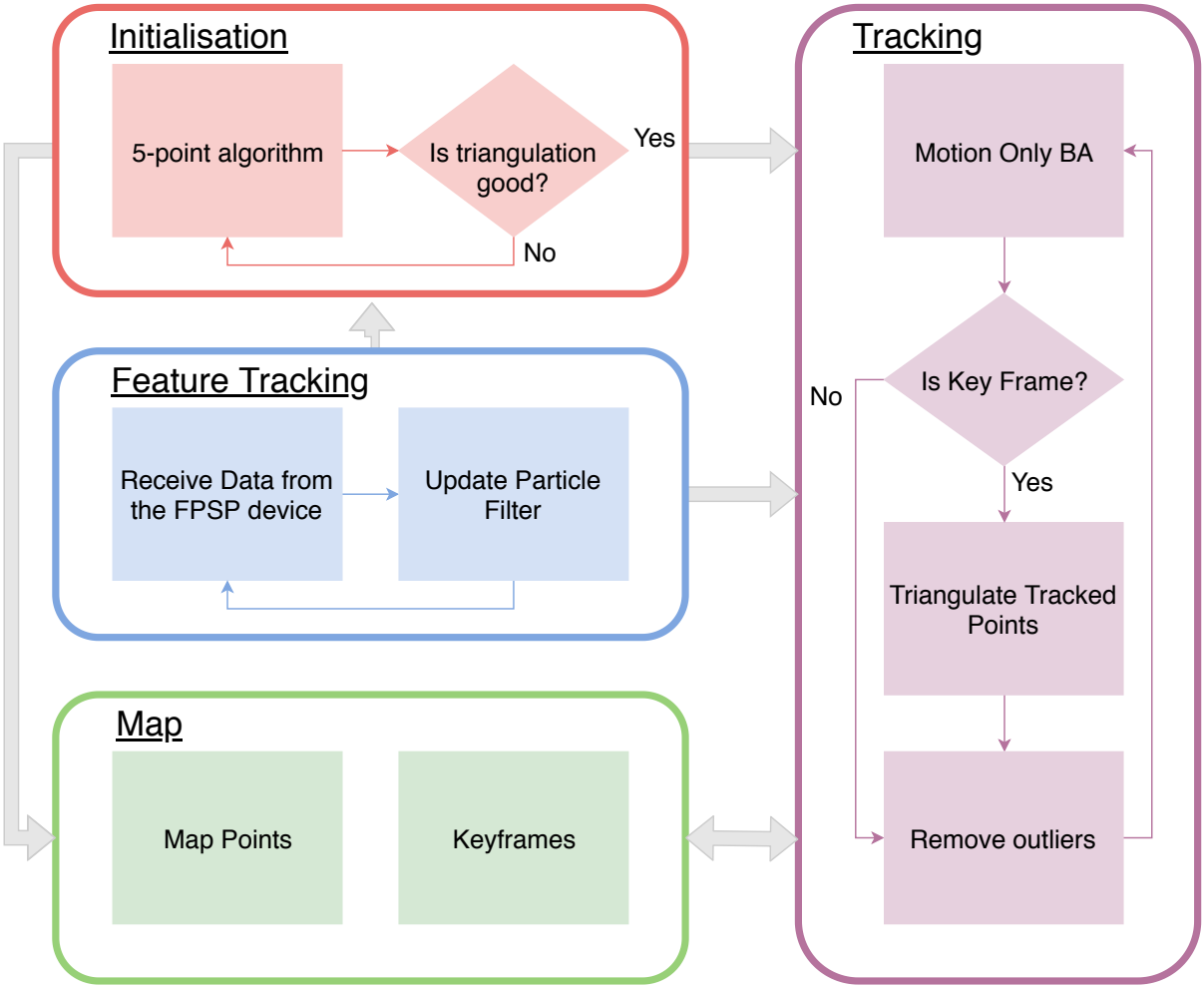


Figure 4.5: Illustration of our visual odometry pipeline.

.

4.4 Summary

From a very basic 2D-2D method visual odometry pipeline we have iteratively built a full visual odometry pipeline which is capable of operating in 200-250FPS range. To best of our knowledge, this is the first ever 6DoF visual odometry for an FPSP device, and we not only we have implemented a method, but we have also successfully utilised the low-latency of the FPSP device in our pipeline. With the successful implementation of a visual odometry pipeline, in the next chapter, we will assess the pipelines performance, together with the feature detector and feature tracker, which we have previously implemented.

Chapter 5

Evaluation

In this chapter, we critically evaluate our implementations and summarise the results. We can divide our works into three related sections:

1. **Feature Extraction:** This section aims to extract good features to track using an FPSP device.
2. **Feature Tracking:** Using the extracted features, we aim to track the features over a long period, and even under some violent motions.
3. **6DoF Pose Estimation:** The correspondence obtained from the feature tracker is used to estimate the 6DoF pose estimate in the 3D space. The 6DoF pose estimation aims to provide accurate pose estimation while maintaining low latency.

5.1 Feature Extraction on an FPSP Device

We have proposed a new method to extract feature on the FPSP device and have shown qualitatively, the improvements compared to the prior works. However, these comparisons were made on a static scene, which is a poor representation of the situations in which we expect the algorithm to be used. On an FPSP device, it is difficult to achieve fair quantitative evaluations as factors such as the motion of the device are hard to control between the run.

The assessment of the performance of our feature extraction should consider the following points.

- Out of the features detected, how many of them can be classified to be good features?
- Of the good features detected, how close are they to the reference data.

The term good feature is vague, and hence in the next section, we will discuss the methods which we used to evaluate the quality of our feature detector.

5.1.1 Methodologies

A quantitative evaluation of the feature extraction is difficult on the physical hardware, thus, we have performed the evaluation using the simulation environment [15]. Use of the simulation environment allows multiple implementations to be tested using the same datasets (Appendix A), which allows a fair comparison. Furthermore, it allows us to generate the reference data from the same video.

Definition of good feature is somewhat arbitrary. However, for our case, a feature is good if it is easy to track. Similar to the aperture problem, easy features to track is the corner features. Although our feature tracker does not use any intensity information, this still holds as the corners are more spatially distributed when compared to edge features, making them easier to match. This motivates us to create the reference data from a strong set of corner features. To produce the reference data, we have executed the OpenCVs [9] `goodFeaturesToTrack()` on the same test-data. It internally uses Shi-Tomasi method [48] and selects the strong corners.

One of the limitations of the simulation environment is that it is not possible to automatically restart the video when our feature extractors start. Also, the feature extractor cannot begin until the simulation environments have started. Lack of ability to automatically restart the video results in the logging of the features to start on an arbitrary frame, making the comparisons of the results difficult. As a solution, we have added 30 blank frames at the end of the video and used them as a synchronisation marker. This works well for the feature detection algorithms as it extracts no feature from the blank frames.

With the simulation environment and the reference data, it is possible to define the comparison methods. To evaluate how often the detector detects a good feature, we define a metric *hit-ratio* as Equation (5.1)

$$\textit{hit-ratio} = \frac{\textit{Number of good features detected}}{\textit{Total number of features detected}} \tag{5.1}$$

A feature is marked as a good feature, if it is within 3 pixel radius of any reference features.

To measure how close the data is to the reference data, we compute for all the good features the Euclidean distance to its nearest reference feature. We will refer to this as the average pixel error.

5.1.2 Observations

We have executed both the previous implementation [14] and our implementation using the same configuration parameters. In Figure 5.3a, we observe that our approach outperform the previous implementation by roughly 80% for the hit-ratio on the shapes, keyboard dataset and 18% on the bicycle dataset. On the shapes dataset, our implementation achieves almost 100% hit ratio, which suggests that all of the features extracted is a good feature to track.

The increase in the hit-ratio for the bicycle dataset is significantly smaller when compared against the other dataset. Compared to the other dataset, the lighting of the bicycle dataset is less controlled. Thus, the task of feature detection is made more challenging. For example, this can be observed in Figure 5.2 where our feature detector only finds features in the brightly lit area of the image. While the threshold can be adjusted to make sure that the features are detected in the shadowed regions, it makes the detector overly sensitive in the bright area, which produces many redundant features.

Figure 5.3b compares the average pixel error of the features. For all of the dataset, we obtain less error compared to the previous implementation. Note, that pixel errors only take into account of the features which are classified to be hit. Hence poor hit ratio does not influence this measurement. The average error of less than 1.2 pixels demonstrates the high accuracy of our feature detector.

The high hit-ratio and low pixel error can be visually compared in Figure 5.1, where our implementation only identifies visually sharp corners.

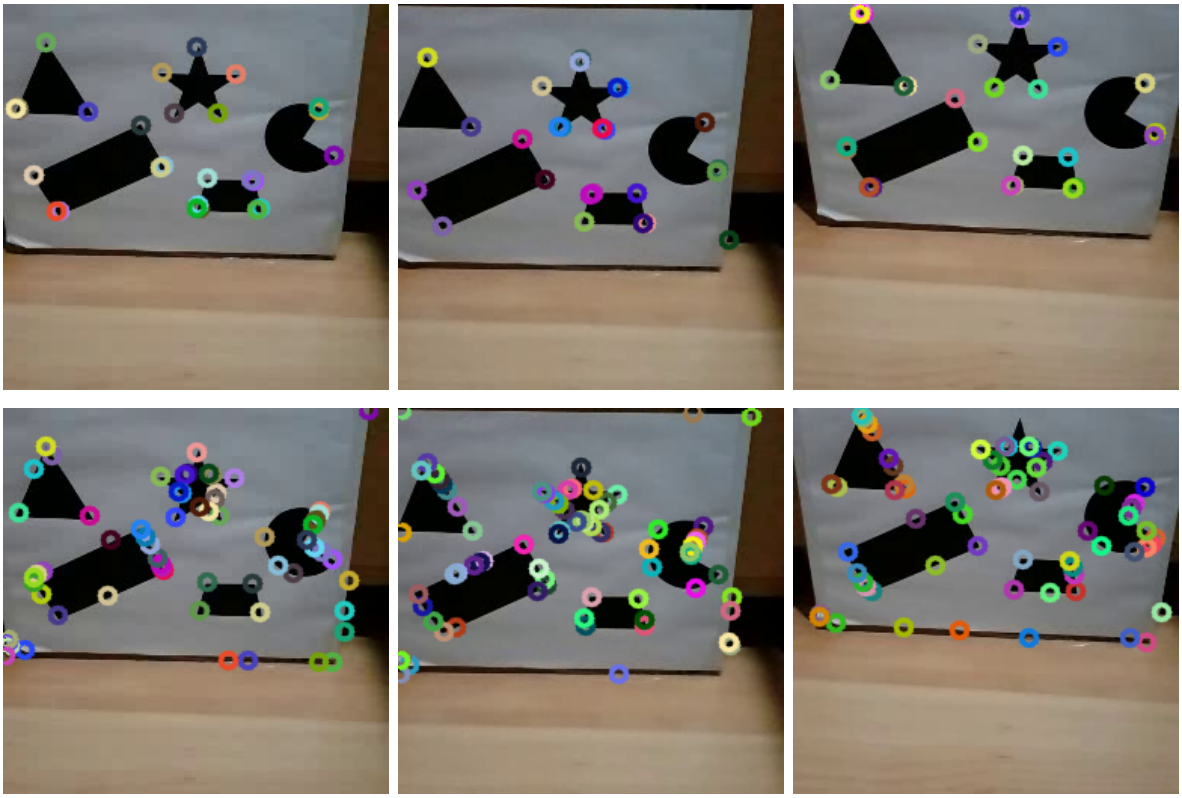


Figure 5.1: Visualisation of the features extracted on the shapes dataset. The top row is our implementation and the bottom row is implementation by Chen et al.

5.2 Tracking Features Extracted by an FPSP Device

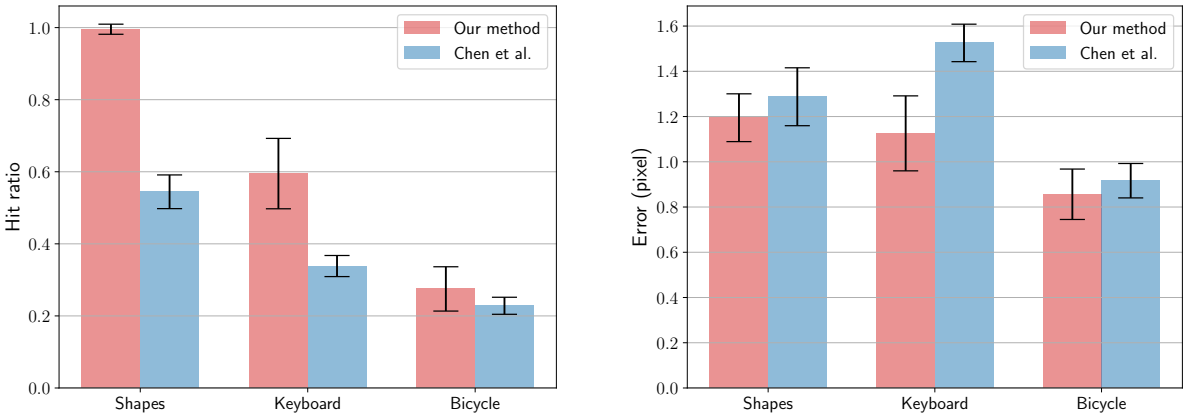
We have presented a feature tracking algorithm, which is designed to track features detected using an FPSP device. While we are not aware of any implementation of the feature tracker which uses the data from an FPSP device, Chen et al. suggests that a simple closest neighbour matching is sufficient for the task. We will evaluate our method against the proposal by comparing how accurately the features are tracked. Furthermore, we will compare our approach against the frame-based algorithm, which allows us to contrast their performance under some challenging condition.

To assess the performance of our feature tracker, we consider the following points.

- How accurately can the features be tracked?
- How long can the features be tracked for?
- How does the number of particles affect the feature tracker?
- How fast is the feature tracker?

For similar reasons as feature detection, quantitative evaluation on the hardware is difficult. Thus the simulated environment is used to measure the quality of the feature trackers. However, we will perform some qualitative evaluations on the physical hardware to demonstrate the advantages of high frame-rate, which the FPSP device has to offer.

Figure 5.2: Visualisation of the features extracted on the bicycle dataset. The top row is our implementation and bottom row is results from Shi-Tomasi Corner Detector. Our detector fails to detect features in the dark shadowed region of the image.



(a) Hit ratio

(b) Average pixel error.

Figure 5.3: Comparison of the hit ratio and the average pixel error between our method and implementation by Chen et al. The error bar represents standard deviation.

5.2.1 Methodologies

We will conduct the evaluation of our feature tracker in four stages. Firstly, we assess the quality of our feature tracker. We define the quality of feature tracker by measuring the average pixel errors and the lifetime of the tracked features. To compute the average pixel errors, the trajectory of the tracked features is compared against a reference trajectory, which is created using the KLT algorithm. We perform this assessment using the simulated data, which allows

Figure 5.4: A Scamp5d device with a mobile phone attached.

the reference trajectory to be generated using the same dataset.

To perform comparisons with the method proposal by Chen et al., we have implemented a feature tracking algorithm which uses the k-dimensional tree (kD tree) to perform the closest neighbour search. Note, we limit the search radius to 3 pixels, such that the errors do not accumulate when the features are lost.

Secondly, we assess how the number of particles affects the quality of tracking. We vary the number of particles used by our feature tracker and measure the average pixel errors against the reference trajectory.

Thirdly, we compare the execution time of our feature tracker and how it varies with the number of features tracked. Since the controlling number of features is difficult, we have created a simple simulated data which contains a fixed number of features.

Finally, we perform a qualitative evaluation of the robustness of our method against a violent motion. To test our method under a rapid motion, using the test-data (Appendix A) is unfair as the frame-rate is limited to 240FPS. Instead, we attach a standard camera onto the FPSP device (Figure 5.4) and moving them together in the same motion. The frames of the camera are recorded at 30FPS while the FPSP device operates at 600FPS. We synchronise the timing of the recording on the camera and the FPSP device by starting the program once the space key is hit and ending it similarly. Since the camera records with sound, we can use it to synchronise the start and the end time. This method is not very precise. However, it is sufficient for a qualitative comparison of the results.

5.2.2 Observations

We summarise the evaluation of the feature tracking in four stages. In the first stage, we measure the accuracy of our system and evaluate it against another method. In the second stage, the discussion of the impact of changing the numbers of the particle is made. In the third stage, we measure the computational time and how the number of features affects them. Finally, in the fourth stage, we compare our method against a method on a frame-based camera under some rapid motions.

The Quality of the Tracking

To assess the quality of the tracking, we have executed our particle filter based feature tracker and the closest neighbour feature tracker on the same test data. The test data is created by executing the feature extractor algorithms on the test data (Appendix A).

Figure 5.5a shows that the particle filter method outperforms the closest neighbour method, both in the number of features tracked and average pixel errors. We notice that the closest neighbour method rapidly loses the number of features being tracked after 3 seconds, while the particle filter based approach manages to retain the features.

Figure 5.5b shows similar performance in the number of features tracked. The closest neighbour method approach outperforms the particle filter method in the average pixel error.

Figure 5.5c shows that the particle filter method has smaller average pixel error compared to the closest neighbour method. However, as shown in Table 5.1, closest neighbour method has more features tracked until the end of the video clip.

The above results suggest that the closest neighbour approach and particle filter approach has similar performance. However, there is an important consideration to be made about the noise present on the physical hardware. Recall in Section 3.3.2 we have investigated the noise in our feature tracker on the physical device. Since in the simulated environment, the feature extractors are too reliable, we randomly the remove features from the dataset, at 5% probability. We call the dataset with the added noise, *noisy dataset* henceforth. Note, to achieve fair comparisons, both the algorithms used the same noisy datasets.

Figure 5.5d shows that particle filter method outperforms the closest neighbour method by a large margin. Closest neighbour method fails to track within 1 second of execution. On the other hand, the particle filter method manages to track features for more than 8 seconds, although there is a build up of errors over time.

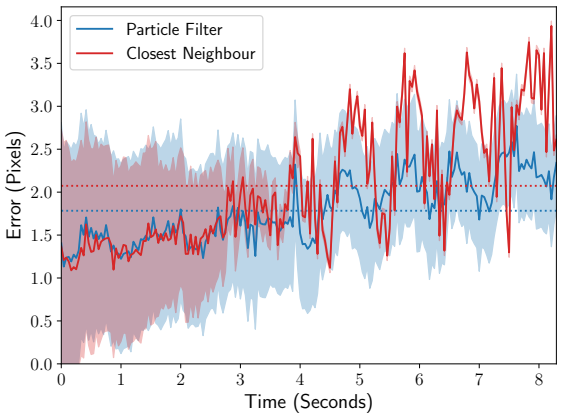
Similarly, Figure 5.5e too shows similar results, where the closest neighbour method loses all its feature within 1 second.

Figure 5.5c shows the closest neighbour approach performing almost as well as the particle filter approach, however, at around 6.5 seconds, the closest neighbour method fails to track particles. It is likely that the closest neighbour method, when compared to the other datasets, lasted longer as the number of starting particles are much larger as shown in the Table 5.1.

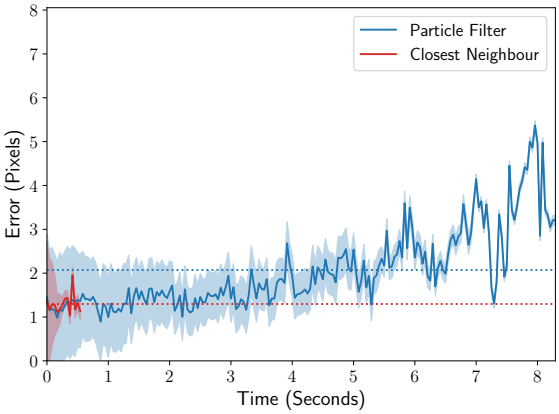
We further assess the tolerance of our approach against noise by applying a different amount of artificial noise to the dataset. As an example, a noise level of 0.1 means that a feature is removed randomly from the dataset with 10% probability. In Figure 5.6, we plot the noise level against the number of frames which contained features which are successfully tracked over the total number of frames processed. We observe that our particle filter approach outperforms the closest neighbour approach widely, demonstrating high robustness against noisy data. The green dotted line represents the expected noise present in the actual hardware. At this noise level, our particle filter based approach tracks feature in all of the frames, whereas closest neighbour approach tracks feature in around a tenth of the frames, showing that our method is 1000% more effective in tracking the feature across multiple frames.

Number of Particles Required for the Feature Tracker

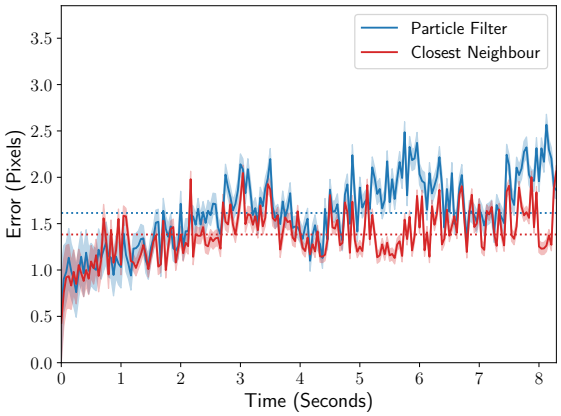
To assess the influence of using a different number of particles per feature tracked, we compare the data against the reference data, in a similar manner as the previous section.



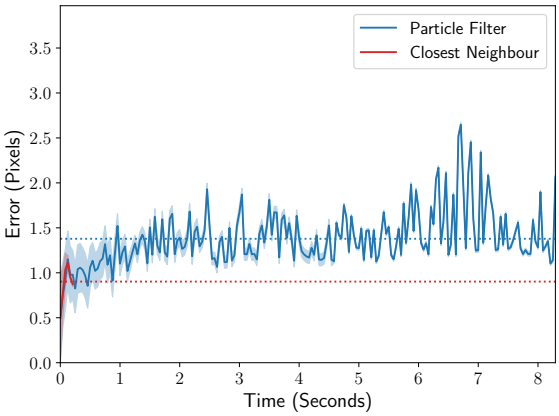
(a) Comparison on the Shapes dataset.



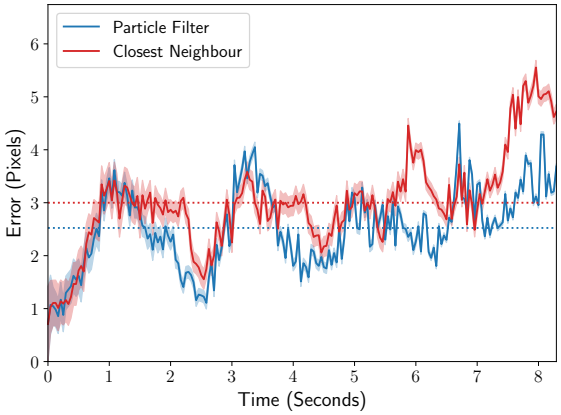
(d) Comparison on the noisy Shapes dataset.



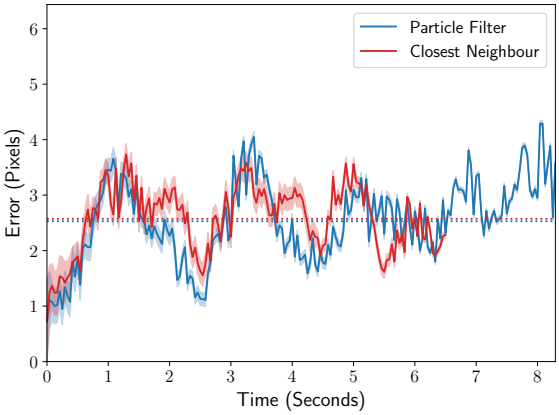
(b) Comparison on the Keyboard dataset.



(e) Comparison on the noisy Keyboard dataset.



(c) Comparison on the Bicycle dataset.



(f) Comparison on the noisy Bicycle dataset.

Figure 5.5: Average pixel error on the datasets. The centre solid line shows the average error in pixels and the bound around indicates the amount of features tracked. The dotted line shows the average of the errors.

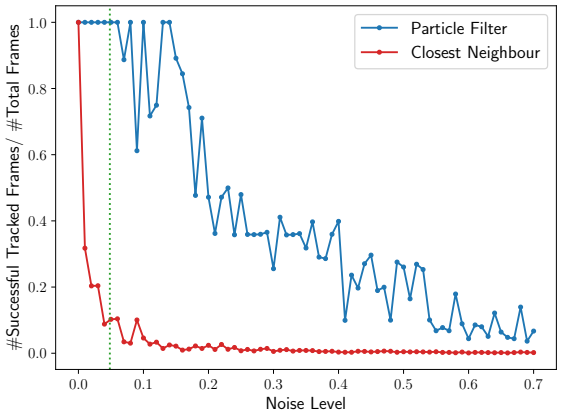


Figure 5.6: Amount of artificially added noise vs ratio of number of frames which contains successfully tracked features. Green dotted line indicates our estimated noise present in the actual hardware.

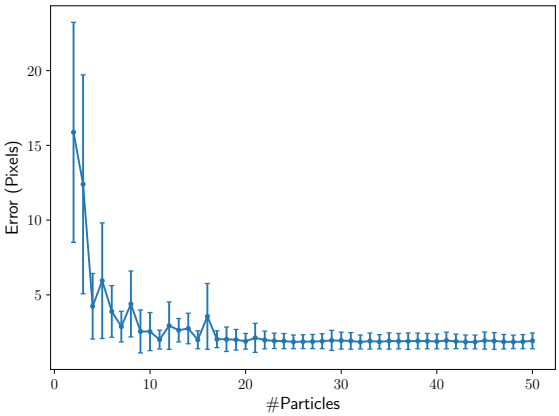
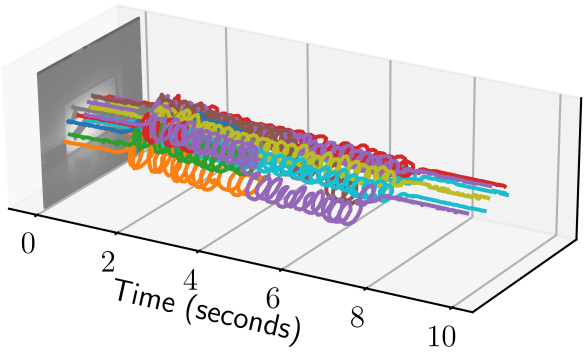
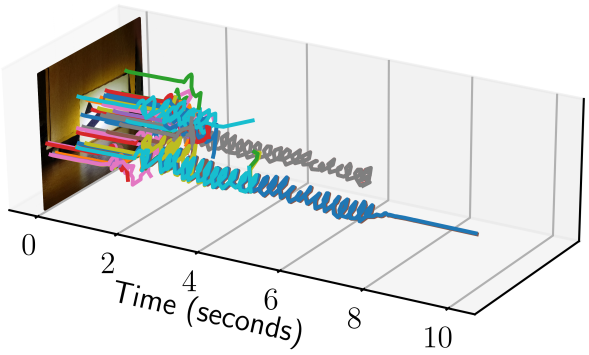


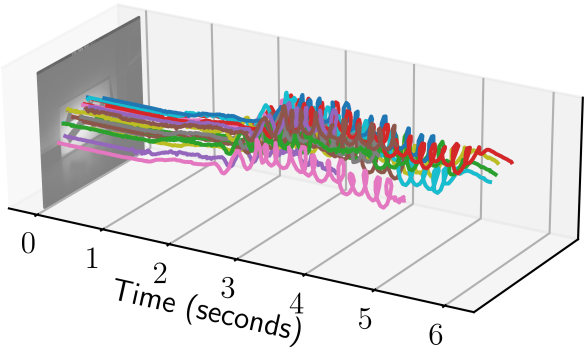
Figure 5.7: The number of particles used by the feature tracker against the tracking error. The error bar represents one standard deviation.



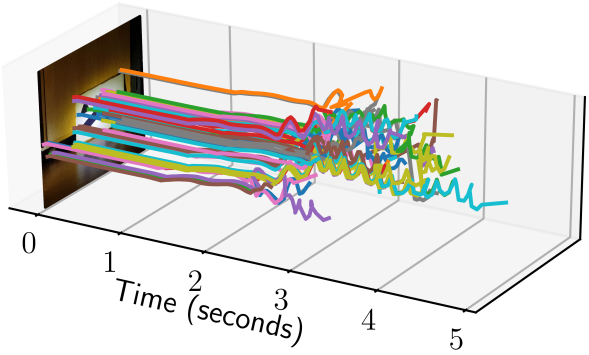
(a) Our feature tracker on circular motions.



(c) KLT algorithm on circular motions.



(b) Our feature tracker on vertical motions.



(d) KLT algorithm on vertical motions.

Figure 5.8: Visualisation of feature tracker on a rapid circular and vertical motions. The grayscale image for our feature tracker is taken for visualisation purpose only.

Dataset	Algorithm	μ_{error}	#Features Start	#Features End
Shapes	PF	1.78	29	12
	CN	2.07	29	2
Keyboard	PF	1.61	24	5
	CN	1.38	24	4
Bicycle	PF	2.52	69	6
	CN	3.00	69	11
Shapes Noisy	PF	2.07	28	3
	CN	1.30	28	0
Keyboard Noisy	PF	1.38	23	2
	CN	0.90	23	0
Bicycle Noisy	PF	2.53	68	6
	CN	2.57	68	0

Table 5.1: The summary of performance of Particle Filter (PF) and Closest Neighbour (CN) methods for feature tracking. The μ_{error} (pixels) is the mean error of the feature tracker for the specific dataset.

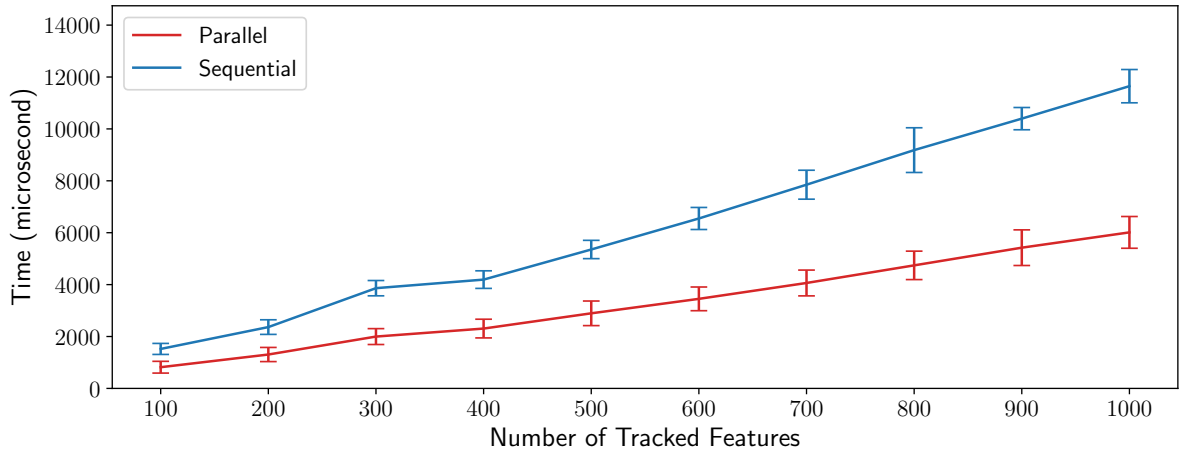


Figure 5.9: Execution time of our feature tracker against the number of tracked features. The error bar represents standard deviation.

As shown in Figure 5.7, after around 20 particles, both mean and standard deviation does not fluctuate, suggesting that the particles have successfully modelled the probability distribution. We expect that with a higher frame rate, fewer particles would be required for the feature tracking. The smaller inter-frame motion would mean that the probability distribution is easier to approximate, hence a small number of particles would be sufficient to represent. Unfortunately, our high-speed camera did not support any faster frame-rates. Thus, we could not validate this hypothesis.

Number of Features Tracked and its Computational Overheads

To assess how the number of features tracked affects the computation time, we have executed our feature tracker on simulated features for 1000 iterations. We perform the measurements on our sequential version of the feature tracker and parallelised version of the feature tracker, which we introduced in Section 4.3.5. All of the timings are measured with microsecond precision.

In Figure 5.9, we observe that for both parallel and sequential implementation, the computation

time linearly increases with the number of tracked features. Furthermore, we demonstrate that our parallelised feature tracker consistently outperforms the sequential version. The consistent performance increase is possible as our feature tracker internally uses particle filter, which is an embarrassingly parallel problem.

Our sequential feature tracker can operate at over 600FPS, given that the number of features around 100. In our experiment, we observe that we manage to track 100 features at 1.52ms, which corresponds to around 657FPS.

Our parallel feature tracker, on the other hand, can track 200 features at 1.31ms, which is around 766FPS. While tracking double the amount of features, the parallelised implementation is still faster than the sequential version. With 500 features, parallelised implementation takes on average 2.89ms to execute. This is around 346FPS, which is slower than our feature detector which operates at 600FPS to extract 500 features. However, we believe by using a machine with more computational capability (e.g. more threads), we can achieve lower execution time, allowing our feature tracker to operate at full capacity of the feature detector on the FPSP device.

Tracking Features Under Agile Motions

We compare our implementation against the KLT algorithm to assess the ability to track under rapid motions. The FPSP device is set to extract feature at 600FPS, and the frame-base camera recorded the scene at 30FPS. All of the operation for feature tracking occurs in real-time. Since the field of view and image centre of the FPSP and the frame-based camera is different, we manually checked the video recorded by the frame-based camera such that no features of interest leave the view. We configured the KLT algorithm to use four levels of the pyramid and search window to be 15×15 pixels.

Figure 5.8a and Figure 5.8c is a visualisation of the trajectory of the features under rapid circular motions. We see that our tracker tracks more features over a longer period, even under violent motions. However, we observe in Figure 5.8a that the purple feature towards us adhere to the orange trajectory. The failure to remove erroneous correspondence is one of the problems which our approach faces.

Similarly, Figure 5.8b and Figure 5.8d shows similar results where our implementation manages to track features and has consistent motions between the features, whereas, in the KLT tracker, we observe that the features motion is inconsistent each other.

5.3 Low Latency Visual Odometry Using an FPSP Device

We have demonstrated a low latency visual odometry pipeline which operates on the data from an FPSP device. As we are not aware of any visual odometry pipeline for an FPSP device which is capable of performing 6DoF tracking, we compare our trajectory against the data from the Vicon motion capture.

Evaluation of the performance of our visual odometry pipeline should consider the following points.

1. How accurately does the pipeline produce the trajectory?
2. What is the latency of the pipeline?

Figure 5.10: Scamp5d device with pearl reflective markers attached to track the position of the device using the Vicon motion capture system.

Comparison of the trajectories is made difficult by the scale ambiguity which monocular visual odometry suffers from. We will detail how we obtained our data and methods which we have used to compute the errors in the next section.

5.3.1 Methodologies

Recall that in monocular visual odometry, trajectories are only recovered up to a scale. The scale factor which maps the trajectory produced by the pipeline to the trajectory of the physical world is unknown. Furthermore, the different origins which the pipeline and the Vicon motion capture system uses must be aligned. Thus, to compare our trajectory against the trajectory from the Vicon motion capture system, the alignment process is required.

We have used evo [26] to compute the similarity transformation, which internally uses Umeyama alignment [55] to align the two trajectories. Similarity transformation is a rigid-body transformation with scaling. We align our pipelines trajectory to the trajectory produced by the Vicon motion capture system, allowing the translation component to have a unit associated with them. This allows a more natural interpretation of the results. Umeyama alignment does not align the initial orientation of the origin, hence, we offset the orientations using the first camera pose estimate. While we did not have sufficient time to implement, aligning the orientation using all of the camera pose estimates would result in better initial orientation alignments, which may further reduce the orientation error.

The Vicon motion capture system requires reflective markers to be attached to the object such that it can track the object in 3D space. As shown in Figure 5.10, we have attached 3 reflective pearl markers on the device. We log our pipelines trajectory using the TUM RGB-D dataset trajectory format [3]. We chose this format as it contains a timestamp, which allows our latency to be measured easily from the log and the rotational component is logged as a unit quaternion, which matches the pose representation of our pipeline. The timestamp is produced using the systems clock, and the epoch time is recorded with microsecond precision. The data from the Vicon motion capture system is stored as rosbag, which we parse and convert to TUM RGB-D dataset trajectory format.

When comparing the trajectory produced by the Vicon motion capture system and our visual odometry pipeline, it is crucial that the timestamp is synchronised. To achieve this, we used

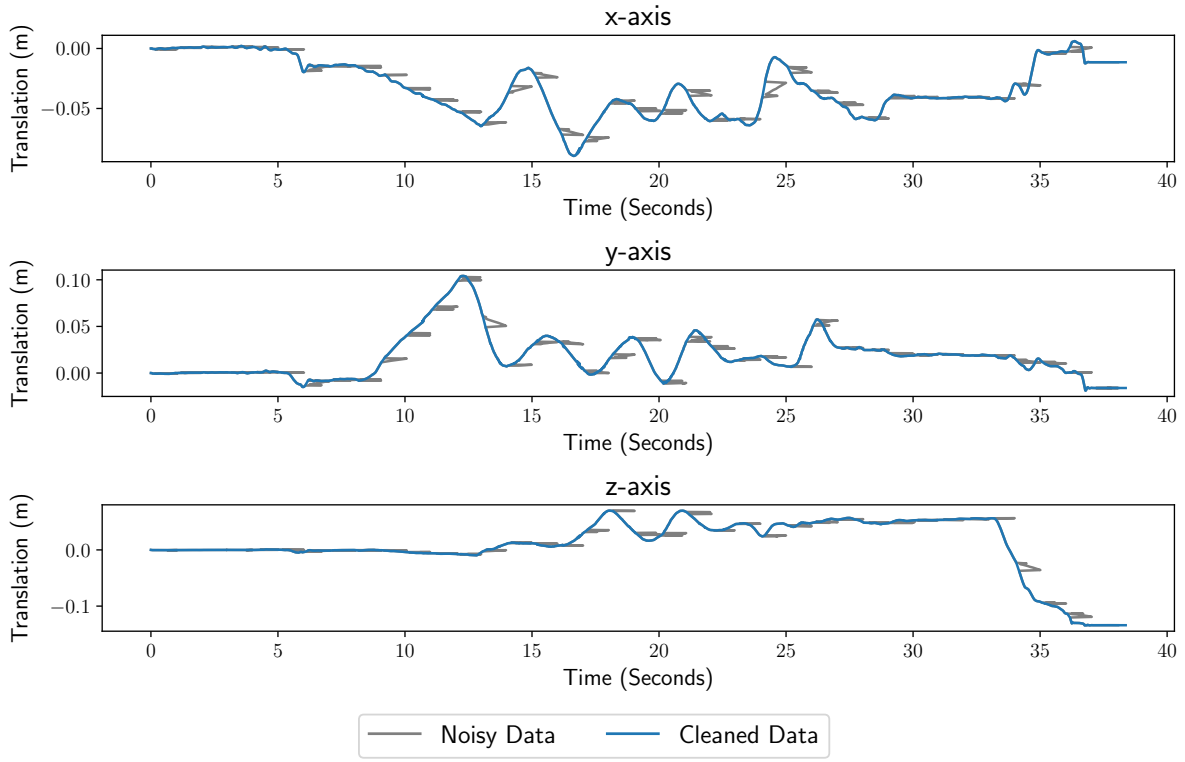


Figure 5.11: Visualisation of the noisy data from the Vicon motion capture.

vicon_bridge [1] which allows the log from Vicon motion capture system to use the system time on our machine which we are running the visual odometry pipeline on. To reduce the latency in the setup, we have connected the machine to the Vicon motion capture system using an ethernet cable, instead of communicating over Wi-Fi. To use the vicon_bridge, installation of ROS [42] was necessary. Since our development environment was OS-X, we have used Docker [2] to create an environment which is capable of executing vicon_bridge.

When plotting the data obtained from the Vicon motion capture system, we have noticed that it contained some noisy readings. The timestamp ordered the majority of the logged data, however, few noisy reading did not follow. Since the outlying data is easy to spot, removing of these data was simple. First, we compute μ_t, the average of the time intervals between the readings. For data i, let the timestamp associate to it be t_i. If $t_i - t_{i-1} > \mu_t + \epsilon$ where ϵ is a threshold, we mark the reading to be an outlier. When the reading is marked as an outlier, it is removed from the final output. However, we replace the timestamp associated with the outlier with $t_i + \mu_t$ such that the filtering can continue for the next readings. The results of the filtering are visualised in Figure 5.11. As we can observe, the general shape of the data is preserved while the obvious spikes are removed.

Given two aligned trajectories which shares the same timestamp, comparison of the trajectories is simple if the sampling rate of the two trajectories matches. However, in our case where the pose estimation updates more frequently than the sampling rate of the Vicon motion capture system, we must further perform some processing. Let a pose estimate p_i^{vo} be our pipelines pose estimate at time t_i^{vo}, which is somewhere between $[t_{j-1}^{vicon}, t_j^{vicon})$. We must find a pose estimate from the Vicon motion capture system, which we can compare our pose estimate against. The simplest method is to pair the pose estimates based on the closest time. For example, in this

[1] vicon_bridge: `https://github.com/ethz-asl/vicon_bridge`
[2] Docker: `https://www.docker.com/`

case, if $t_i^{vo} - t_{j-1}^{vicon} < t_j^{vicon} - t_i^{vo}$, we pair p_i^{vo} with p_{j-1}^{vicon} and with p_j^{vicon} otherwise. Despite its simplicity, this method is unfair, as it penalises the high-frequency visual odometry pipeline. To avoid this, we perform interpolation of the pose data. For the translation component, simple linear interpolation suffice. However, for the rotation, simple linear interpolation of either the Euler angles, the rotation matrix or the quaternion would not result in a constant angular velocity [16]. This effect is undesirable. Instead, interpolation should occur along the unit sphere of the quaternion. This method, spherical linear interpolation, often referred to as slerp provides constant angular velocity during the interpolation. This enables the rotation between the two Vicon motion capture systems reading to be smoothly interpolated. We have used SciPy [30], in particular `spatial.transform.Slerp()` to perform the slerp operation. With the smooth interpolation of both position and orientation, we can compute the errors in the trajectory.

To measure the error, we must define the metric which we measure the error with. For the position error, we use the Euclidean distance as the metric. For the orientation error, since the error of our orientations $\theta_x, \theta_y, \theta_z$ was constrained such that $\theta_x, \theta_z \in [-\pi, \pi); \theta_y \in [-\pi/2, \pi/2)$ we use the Euclidean distance of the difference in Euler angle as the metric [29]. This is a fair metric to use, under an assumption that our pipelines orientation does not differ from the ground truth too much. An alternative metric which we could have used is the geodesic distance in $\mathbf{SO(3)}$ as used by [33].

5.3.2 Observations

We have captured the motion of the device using the Vicon motion capture system, in a well-lit environment with checkerboards to provide textures for tracking. We run the feature extraction at 250FPS not at the maximum speed of 600FPS, to prevent the pipeline from being flooded by the stream of features.

We have evaluated our visual odometry pipeline under two different settings. The first settings, *slow-mode*, has higher latency. However, we expect that the tracking is accurate. The second setting, *fast-mode*, has lower latency. However, we hypothesise that the tracking is less accurate when compared to the slow-mode. The difference between the two modes is minor. The fast-mode performs grid/binning strategy, whereas the slow-mode does not. Reducing the number of features tracked reduces the computational overhead on the feature tracker and also for the pose-estimation, as fewer correspondences are established. However, with fewer features, the fast-mode is likely to be more sensitive to noisy readings.

Accuracy of the Visual Odometry Pipeline

First, we will evaluate the accuracy of our visual odometry pipeline and how the different modes affect the results. The total trajectory distance of slow-mode and fast-mode is 0.95m and 0.64m respectively In Figure 5.12, we compare the absolute position and orientation of slow-mode against the ground truth trajectory, which we have obtained from the Vicon motion capture system. For the translation, although there are some high-frequency noises, we observe that our translation estimations follow the ground truth data closely. Furthermore, on the z-axis, at around 1-4s, we notice that our translation estimation deviates from the ground truth. However, the pipeline quickly recovers, as the pose estimation is performed against the 3D map. The error would have built up if a 2D-2D method were used, as the new pose estimation is directly dependent on the previous pose.

For the rotation, we observe that the rotation along x, y-axis follows the ground truth closely. For the y-axis, between 10-18s, we demonstrate the capability of our rotation estimation by

successful tracking of a large rotation. For the z-axis, while the general shape of the curve is similar to the ground truth, the error increases over time. Figure 5.17 compares our full trajectory against the ground truth. Our trajectory closely follows the ground truth, however, it does oscillate at high frequency.

In Figure 5.13, we similarly compare our fast-mode trajectory against the ground truth. Immediately, we notice that the system contains more high-frequency noise when compared to the slow-mode. As for the translation, we notice that for all of the axis, the estimate follows the ground truth, with some added noise. Along the z-axis, we notice that there is significantly more oscillation in error. This effect is also visible in our overall trajectory as visualised in Figure 5.18.

For the rotation, we observe that while all of the trajectories follow a similar pattern to the ground truth, they are all shifted in one direction. However, unlike the slow-mode, we do not observe rotation error increasing over time.

We plot the absolute position and orientation errors of the two modes. For the slow-mode, Figure 5.14a shows the changes in the absolute position error over time. We notice that the error does not accumulate, which demonstrate the effective use of the 3D map. Figure 5.14b shows the changes in the absolute orientation error over time. There is an increasing trend in error, which we can see from Figure 5.12 is dominated by the error in the z-axis. We observe a small difference between the Root Mean Square Error (RMSE) and the mean, which suggests that there are few very large errors.

For the fast-mode, Figure 5.15a shows the changes in the absolute position over time. Similar to the slow-mode, we observe a reasonably stable error, which suggests that the error is not accumulating. Figure 5.15b shows the changes in the absolute orientation error over time. We do not observe error building up over time in the fast-mode.

The metrics taken for the two modes are summarised in the Table 5.2. As we expected, fast-mode has a higher mean, RMSE and standard deviation of error for both position and orientation when compared to slow-mode.

Latency of the Visual Odometry Pipeline

For both modes, we have logged the timestamps with microsecond precision. Figure 5.16a shows the computational time required by the slow-mode. On average, the pipeline operates at 12.0ms, which translates to around 83FPS. Figure 5.16b shows the computational time required by the fast-mode. On average, the pipeline operates at 4.007ms which translates to around 250FPS, reaching the theoretical maximum speed under our settings. We believe that with more parameter tuning, we can exceed our current latency.

For both slow-mode and fast-mode, we observe a spike in the processing time. The spike is likely to be caused by the keyframe insertion as it introduces additional computation. For the slow-mode, we notice that the frames following the spike suffers from high latency. Recall, the effect of buffering too much frame of features as discussed in Section 4.3.3. The increase in computational time on the host machine increased in the size of the buffer, creating overhead for the following iteration. We, however, notice that the overhead is resolved fast, and the computation time drops back down to the average. On the other hand, spike caused by insertion of keyframe does not affect the following frames in the fast-mode. This is because, the number of the feature being tracked is fewer, and meaning that the keyframe insertion requires less computation. Hence, the buffer does not build up in size as much, allowing the computation time of the system to drop back to the average almost immediately.

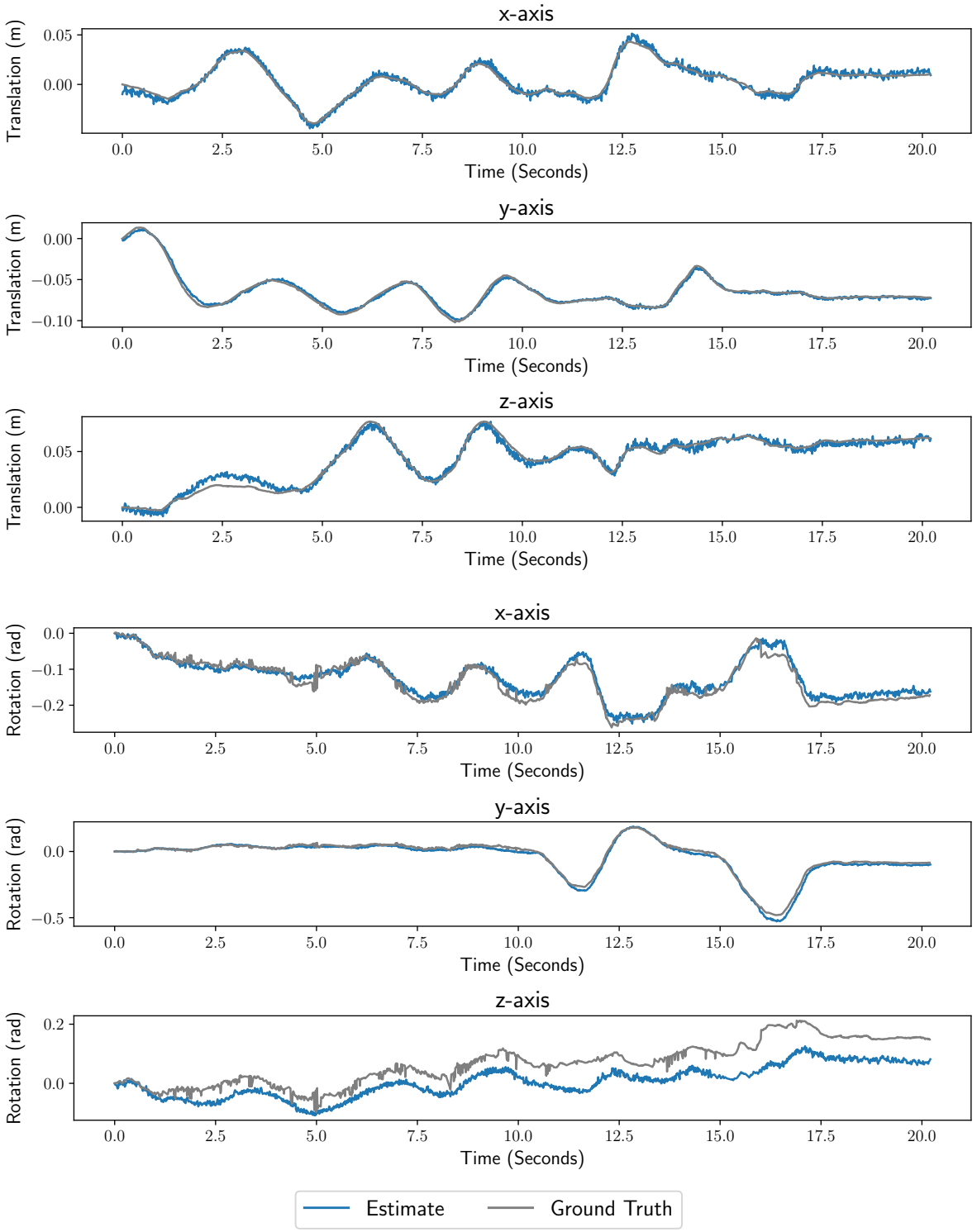


Figure 5.12: Our visual odometry pipeline (slow-mode) against the ground truth. The translations and rotations corresponds to the absolute position after alignment of the trajectories.

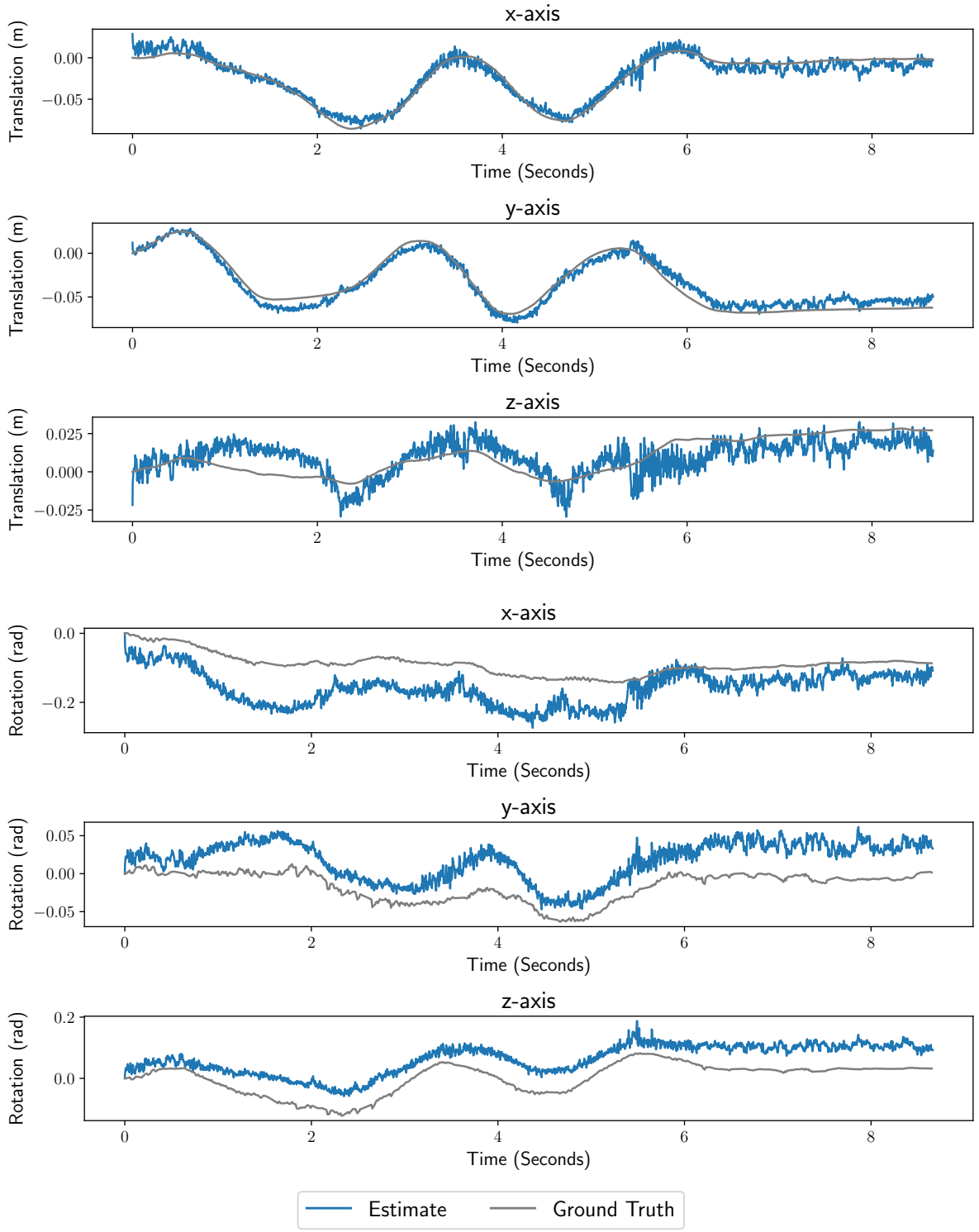
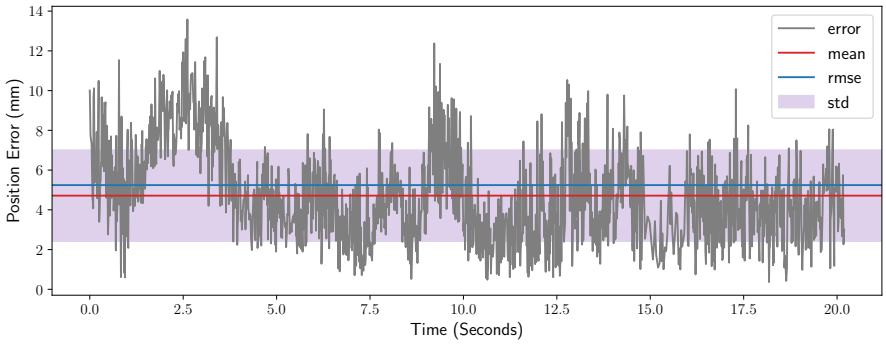


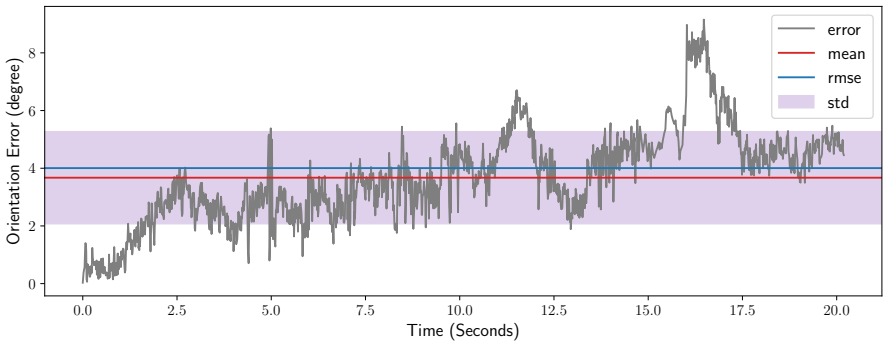
Figure 5.13: Our visual odometry pipeline (fast-mode) against the ground truth. The translations and rotations corresponds to the absolute position after alignment of the trajectories.

Mode	Position (mm)			Orientation (deg)		
	μ	RMSE	σ	μ	RMSE	σ
slow-mode	4.71	5.24	2.30	3.67	4.00	1.60
fast-mode	13.7	14.6	5.21	5.87	6.09	1.64

Table 5.2: Summary of absolute position and orientation error. The table contains average, root means square error and standard deviation of the errors.



(a) Absolute position error



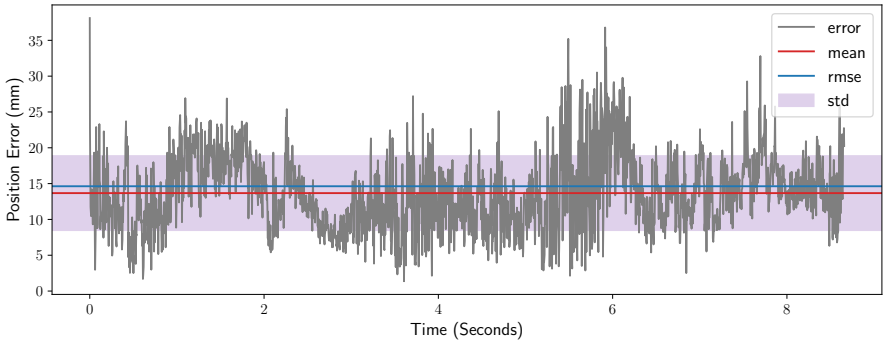
(b) Absolute orientation error

Figure 5.14: Absolute position and orientation error of slow-mode with respect to the ground truth.

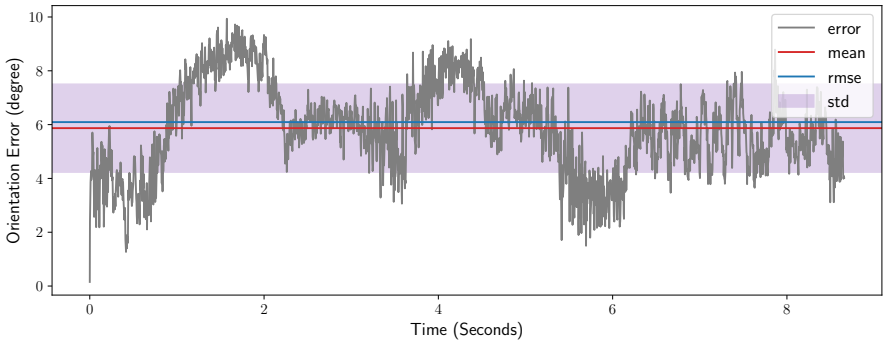
5.4 Limitations and Further Evaluations

While we have made efforts to make sure that the evaluations are carried out under fair conditions, there are some rooms for criticism of our approaches.

- Simulated environment may not be representative of the actual hardware. For instance, we artificially added noises posterior to the execution of our feature detector when evaluating the quality of the tracking. However, we are not certain whether the noise is uniformly distributed.
- Imperfect time measurements. All of the timings were measured on a laptop, which could have been executing other processes. For example, while measuring the latency of the visual odometry pipeline, docker and vicon_bridge was running to log the ground truth data. While some effort was made, such as closing applications irrelevant to the evaluation, we cannot avoid processes running in the background.



(a) Absolute Position Error.



(b) Absolute Orientation Error.

Figure 5.15: Absolute position and orientation error of fast-mode with respect to the ground truth.

- Highly controlled lighting and high contrasting texture for visual odometry evaluation. When evaluating our visual odometry pipeline, we have executed our pipeline in a highly controlled situation, which is not a very accurate representation of the environment which we expect our visual odometry pipeline to be used in.

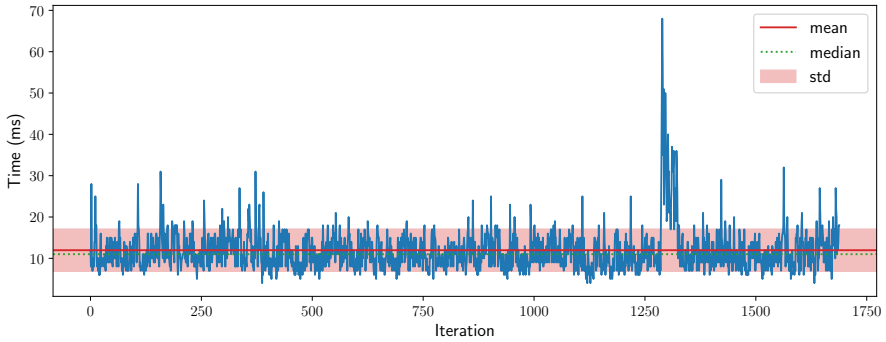
Use of the simulated environment was necessary to obtain the reference data. However, as the simulator is closed sourced, we were not able to add the error models of the Scamp5d device studied [57]. While an alternative option was to use a simulation environment which Wong has created, it would have requires large modifications to our codebase. Use of a simulator with noise carefully modelled would allow us to avoid introducing artificial noises post-execution.

Regarding the timings, most of our evaluations would only further benefit from reduced computation time. Furthermore, when we compare the time measurements of two different algorithms, the difference between them is in order of milliseconds, which suggests that the impact of the additional process executing is unlikely to negate our conclusion.

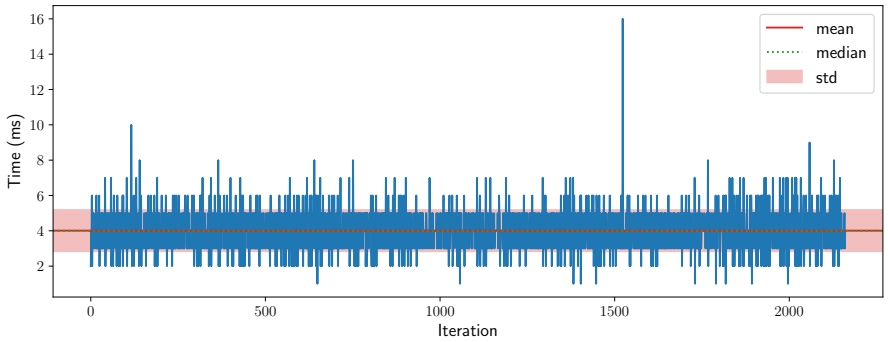
The highly controlled lighting and the high contrasting texture were necessary due to the limitations of our visual odometry pipeline. However, we are positive that improvements can be made such that we can further extend our pipeline to support a more comprehensive range of scenes.

5.5 Summary

In this chapter, we have critically evaluated our implementations, discussing the effectiveness and limitations of our approaches.



(a) slow-mode



(b) fast-mode

Figure 5.16: Visualisation of the latency of our visual odometry pipeline. For both slow and fast mode, we observe a spike in the latency, which is likely to be caused by the keyframe insertion.

We have demonstrated improvements which we have made upon the existing feature detection algorithm and contrasted our performance quantitatively. We also noticed that both our method and the existing method perform poorly in dark scenes, which is an interesting problem and may require some hardware improvements as a solution.

For the feature tracker, we have shown the effectiveness of our particle filter based approach and the necessity to implement a feature tracker which is robust against noise. Although it was a qualitative evaluation, we have demonstrated that feature tracking using an FPSP device is more reliable when compared to feature tracking using a standard frame-based camera in violently moving scene.

Finally, for the visual odometry, we have shown the high accuracy tracking which our visual odometry pipeline achieves while maintaining low latency of below 5ms per pose estimate. The success of the visual odometry pipeline, in fact, entails the success of our feature detector and feature tracker.

Discussion of the limitations of our evaluation was made, and we will propose some potential solutions to tackle the issues in the next chapter.

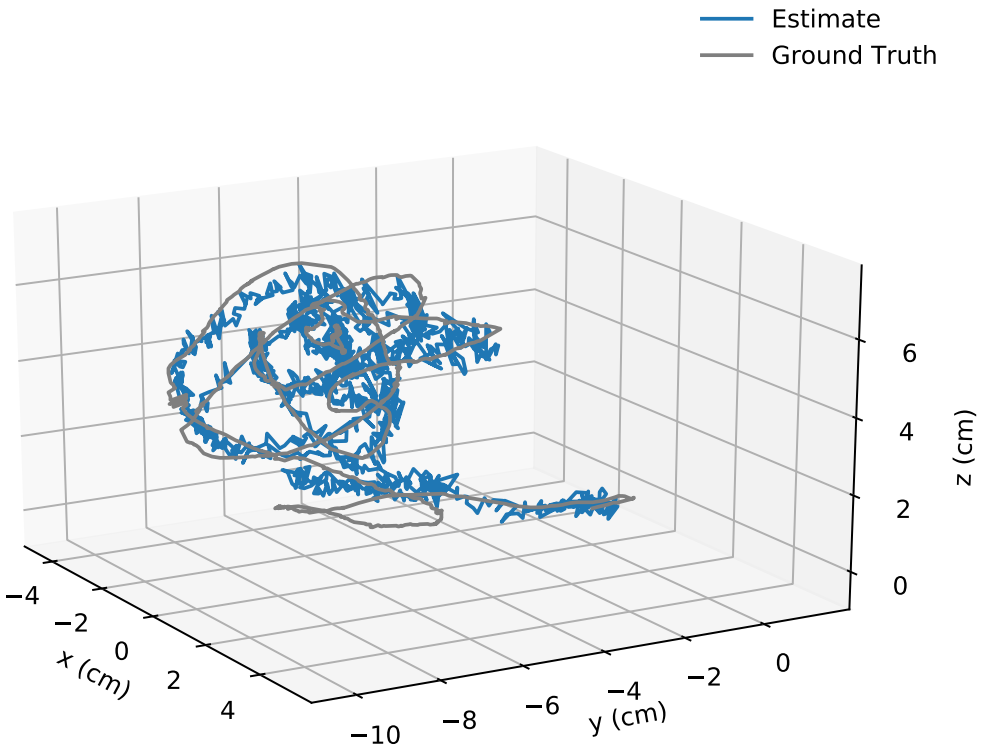


Figure 5.17: Comparison of the trajectory produced by the slow-mode against the ground truth.

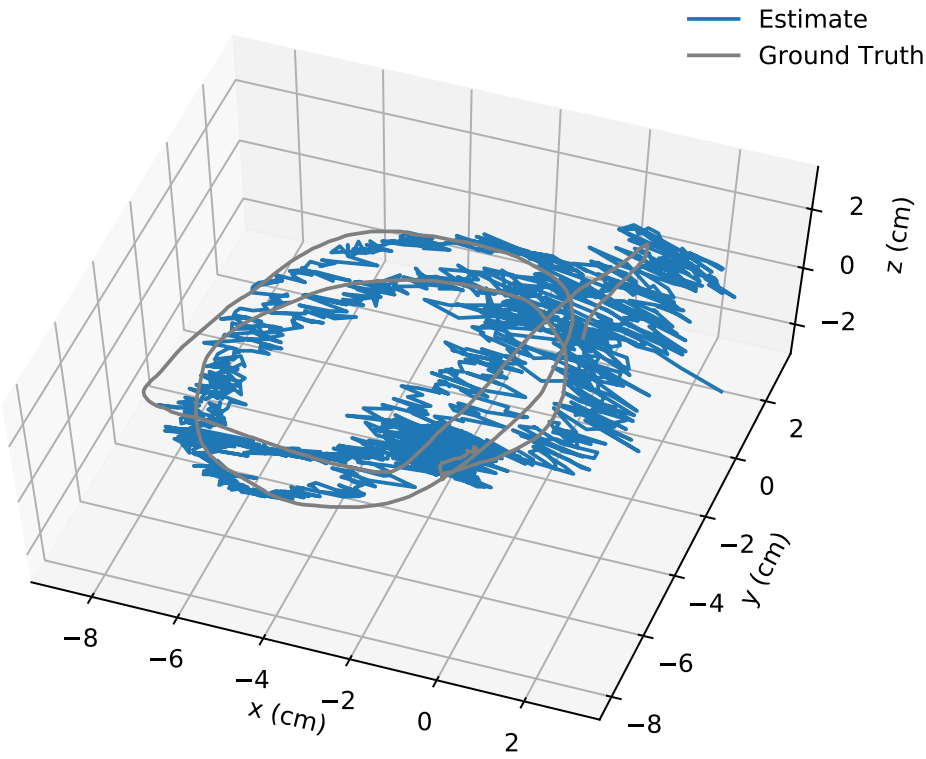


Figure 5.18: Comparison of the trajectory produced by the fast-mode against the ground truth.

Chapter 6

Conclusion and Future Works

This dissertation contains contributions to the areas of feature detection, feature tracking, and monocular visual odometry on a Focal-plane Sensor-Processor (FPSP). As of now, there are limited research [25] in further processing of the data from an FPSP device on a conventional computational unit, and our work is one of the most complicated its kind.

To best of our knowledge, this work marks the first successful 6 Degrees of Freedom (DoF) visual odometry pipeline using an FPSP device, with no prior information about the scene required. Furthermore, our work presents the first successful tracking of the features from an FPSP device. We hope that our contributions open up many future potentials for low latency computer vision.

6.1 Contributions

In this section, we summarise the contributions of our project.

6.1.1 Feature Detection and Tracking

In Chapter 3, we have presented various improvements to the existing feature detector algorithm and introduced our feature tracker, which is robust against a noisy set of data. Both of the algorithms operate at over 600FPS and has been demonstrated to operate under challenging conditions such as violent motions. Furthermore, no restriction is imposed on the motions of the features, allowing the camera and the tracked objects to move in arbitrary directions, opening up the use of our feature tracker in a wide variety of situations.

We have further shown the capability of our feature detector and feature tracker through the successful implementation of our visual odometry pipeline. This strongly entails that our feature detector and tracker applicable to many more computer vision algorithms, potentially contributing to towards reducing their latency.

6.1.2 Visual Odometry Using an FPSP Device

In Chapter 4, we have successfully demonstrated to the best of our knowledge, the first 6DoF visual odometry using an FPSP device. Not only a successful implementation, but we have also utilised the full potential of the device, and achieved a low latency of under 5ms per pose estimation. We have also shown that with a high frame-rate, little information is required to

perform the complex task of pose estimation. The majority of our pipeline operates on the coordinate information of the features only, which is a very sparse representation of the frame data. Even with such little data, our pipeline produces highly accurate trajectories.

Unlike the previous approaches to the visual odometry using an FPSP device [8, 18], our trajectory does not suffer from ambiguity in scale between each frames. Furthermore, we create a sparse representation of the scene, which is beneficial information for tasks such as loop closure.

6.2 Challenges and Lessons Learnt

We have based our work on cutting-edge camera technology, with limited documentation and examples. Not only we had to familiarise and understand the programming model for the Scamp5d, but we also worked on real hardware, hence, significant efforts were dedicated to debugging. For instance, the micro-controller on the device had a limit to the number of kernel functions which one could register. Exceeding the limit causes the program to crash silently, and moreover, we were unable to use standard tools and debuggers to identify the problem. Furthermore, the simulation environment does not model such limitations.

The work on the visual odometry has taken up a significant proportion of our time spent on the project. Lack of previous experiences in the field limited the progress at the early stages. However, as described in Chapter 4, we have incrementally built on top of our previous works, which allowed us to learn the different methodologies available in the field while guiding our way towards the goal.

One decision which was made amidst our project was to implement our visual odometry pipeline. Using an existing solution may have reduced the time which we have spent learning and implementing the different methodologies, and it could have resulted in a better system overall. However, to use the data from an FPSP device, some modification to the solution must be made. With the limited understanding of the field, effective debugging and understanding where to introduce the changes may not have been possible. These are high-risks, which we have decided not to take. However, now equipped with the knowledge, if given a chance to extend on our work, the next obvious direction is to integrate our system with the existing state-of-the-art methods.

6.3 Discussions

Although we have implemented a visual odometry pipeline for an FPSP device, we have not fully discussed its limitation and where our project positions in the widely explored field of visual odometry.

Our visual odometry pipeline is currently restricted in the area which it can explore. The limitation arises from the imperfect the feature correspondences. The small errors cause the triangulation to be deformed, and as the device moves towards the unexplored area, the number of tracked feature decreases together with the quality of the pose estimations. Hence, the next triangulation suffers from poor pose estimate, damaging the 3D map.

Furthermore, due to the high frame-rate, the bright illumination in the scene is necessary. We must use lighting equipment or have sunlight for our feature detector to operate reliably. This restriction limits the usability of our system.

Our visual odometry pipeline differs from others in many ways. When compared to the visual odometry using a frame-based camera, we achieve lower latency, as the frame-rate inherently limits the latency. The boundary can be lowered through the use of the high-speed camera, however, it consumes a considerable amount of energy, limiting its practical use. Furthermore, the processors must be able to process the large amount of pixel information transferred. While the nature of the data between a frame-based camera and an FPSP device widely differs, the majority of our pipeline share similar concepts. This is because we have carefully separated the tasks in our pipeline, in a way which allows us to borrow ideas from frame-based visual odometry.

Another camera technology which is often used to obtain low latency is the event-based camera. Use of event-based camera is actively explored, and there are many promising visual odometry/SLAM systems [31, 33, 44]. The two hardware shares similar concepts, however, there are some critical differences. First, on the event-camera, a stream of asynchronous events are transferred at sub-millisecond latency, which is not possible on our feature detector. Furthermore, when the scene is static, the device produces no output, which contributes to a reduction in the overall amount of data transferred. However, under a rapid motion, there is an increase in the number of events produced per second. This effect is undesirable as under a fast motion, we wish to perform pose estimation as quickly as we can. On the other hand, an FPSP device does not asynchronously provide output data, thus the frame rate is constant. Moreover, the total amount of data transferred is less as computation can occur on the vision chip itself. In feature-based visual odometry, features must be extracted from the given data. To extract features from the stream of events, they must either implement a sophisticated algorithm to handle the asynchronous nature of the events [38], or if Dynamic and Active pixel VIsion Sensor (DAVIS) is used to obtain a frame data [33] which they can extract the features from. These introduce additional computational overheads, and transfers large amount of redundant information. Overall, when compared to the event-based camera approaches, while some of the event-based methods have lower latency, our approach transfers less information, and our performance does not fluctuate with the different motions. Furthermore, as less information is transferred, our communication channel is less saturated, which suggests that we can transfer additional information to aid our pipeline.

6.4 Future Works

We have shown through combining an FPSP device with a standard computer, achieving a low latency visual odometry pipeline is possible. However, there are many open questions which we could not address.

6.4.1 Visual Odometry

In our visual odometry pipeline, two keyframes are used to triangulate the 3D map. However, this method is sensitive to noisy readings. As our feature tracker can track features across multiple frames, methods such as recursive Bayesian depth filter used in SVO [22], or local bundle adjustment used in ORB-SLAM [39] can be used to improve the quality of the 3D map, and hence improve the tracking.

Through a fusion of multiple sensory information, we expect our pipeline to be able to produce better trajectory estimate. For example, an inertial sensor may aid the filtering of the high-frequency noise, which is currently present in our trajectory.

We expect to see improvements in the quality of the features extracted if we perform a non-maximal suppression over a broader range. Naively increasing the patch-size would result additional computational time, thus, the use of multi-layer Convolutional Neural Networks is a viable solution. It would allow the units in the final layer to have a large receptive field while using a small kernel, allowing the network to find the best feature to extract over a large patch of pixels. Furthermore, with sufficient training data, the network should learn how to detect features from poorly illuminated scenes.

Our implementation of the feature tracker exploited the opportunity for parallelism. It has been shown [49] that the underlying particle filter could further benefit from hardware acceleration using Field-Programmable Gate Array (FPGA). Connecting an FPGA with an FPSP device has already been demonstrated in [25], making the use of FPGA for particle filter based feature tracking promising. This would reduce the computational load on the host device, reducing the latency in the pipeline even more.

Finally, in our current implementation where the feature extraction occurs at 250FPS, the FPSP device has spare computational time. One effective use of the spare time is to transfer the pixels information from the FPSP device to the host device. Unlike in DAVIS, where the frame-rate is fixed, we can transfer the pixel information at an infrequent interval, or on demand. For example, an image can be transferred upon keyframe insertion. The descriptors can be extracted from the image, allowing the 3D map for the visual odometry pipeline to be further utilsed. Addition of the descriptors may even allow implementation of low latency Simultaneous Localisation and Mapping (SLAM). To assess the feasibility of the approach, we have performed a simple evaluation. As shown in Appendix B, we execute our visual odometry pipeline while transferring pixel intensity information at 4FPS. The textures in the snapshots show high contrast, which suggests that there is sufficient intensity information from which useful information can be extracted from.

6.4.2 Development Environment and Testing Framework

We have used the simulation environment for the Scamp5d device [15]. However, the environment was somewhat limited, and as it is a closed source project, which prevented us from making modifications. There have been multiple implementations of the simulation environment for the Scamp5d device, which utilised GPU [1], or accurately models noise on the physical device [57]. These implementations, when compared to the original simulator, is limited as the code for the Scamp5d device cannot operate as is. We strongly believe that the easy to use, an efficient simulation which models the device accurately is an essential tool to increase the community for the FPSP devices. Such a tool would open up a development environment for more researchers who are interested in the device.

As of now, there are no standard test data for the FPSP device. This makes the quantitative evaluation challenging and comparing the performance of the different algorithms time-consuming. By creating a test-data, using a high-speed camera and an accurate simulator, it would allow the researchers to compare and criticise the different works easily.

Currently, the code for an FPSP device is written in a very primitive manner as the instruction sets for the limited. One difficulty of coding on an FPSP device is that some of the instructions have side-effects, which overrides the registers which were in use. Furthermore, no compiler-warnings are available to indicate the use of uninitialised registers. Since the instruction sets on an FPSP device is limited, it should be possible to implement a semantic checker, which warns the incorrect use of registers. Furthermore, using register allocation techniques such

[1] cpa-sim: https://github.com/najiji/cpa-sim

as graph-colouring, we can reduce the use of redundant registers in our code. Moreover, the register allocation can take into account the different magnitude of noise each register suffers by. A cunning register allocation scheme should be able to allocate long-lasting data into a noise resistant register.

Bibliography

[1] Scamp5d Vision System: Scamp5d Vision System Introduction. URL `https://personalpages.manchester.ac.uk/staff/jianing.chen/scamp5d_lib_doc_html/_p_a_g_e__i_n_t_r_o__d_e_v_i_c_e.html`. [Accessed: 2019-06-06].

[2] Parking dataset. `https://files.ifi.uzh.ch/rpg/teaching/2016/parking.zip`. [Accessed: 2019-06-07].

[3] Tum file formats. `https://vision.in.tum.de/data/datasets/rgbd-dataset/file_formats`. [Accessed: 2019-06-10].

[4] Sameer Agarwal, Keir Mierle, and Others. *Ceres Solver*. URL `http://ceres-solver.org`. [Accessed: 2019-06-06].

[5] Simon Baker and Iain Matthews. Lucas-Kanade 20 years on: A unifying framework. *International journal of computer vision*, 56(3):221–255, 2004.

[6] Nitin Bhatia and Megha Chhabra. Accurate corner detection methods using two step approach. *Global Journal of Computer Science and Technology*, 2011.

[7] Jose-Luis Blanco. A tutorial on SE(3) transformation parameterizations and on-manifold optimization.

[8] L. Bose, J. Chen, S. J. Carey, P. Dudek, and W. Mayol-Cuevas. Visual Odometry for Pixel Processor Arrays. In *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 4614–4622, October 2017. doi: 10.1109/ICCV.2017.493.

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

[10] Christian Brandli, Raphael Berner, Minhao Yang, Shih-Chii Liu, and Tobi Delbruck. A 240× 180 130 db 3 μs latency global shutter spatiotemporal vision sensor. *IEEE Journal of Solid-State Circuits*, 49(10):2333–2341, 2014.

[11] Stephen J Carey, David RW Barr, Bin Wang, Alexey Lopich, and Piotr Dudek. Locating high speed multiple objects using a SCAMP-5 Vision-Chip. In *Cellular Nanoscale Networks and Their Applications (CNNA), 2012 13th International Workshop on*, pages 1–2. IEEE, 2012.

[12] Stephen J Carey, David RW Barr, Bin Wang, Alexey Lopich, and Piotr Dudek. Mixed signal SIMD processor array vision chip for real-time image processing. *Analog Integrated Circuits and Signal Processing*, 77(3):385–399, 2013.

[13] Stephen J Carey, Alexey Lopich, David RW Barr, Bin Wang, and Piotr Dudek. A 100,000 fps vision sensor with embedded 535gops/W 256x256 SIMD processor array. In *VLSI Circuits (VLSIC), 2013 Symposium on*, pages C182–C183. IEEE, 2013.

[14] Jianing Chen, Stephen J Carey, and Piotr Dudek. Feature Extraction using a Portable Vision System. page 2, 2017.

[15] Jianing Chen, Stephen J Carey, and Piotr Dudek. Scamp5d Vision System and Development Framework. In *Proceedings of the 12th International Conference on Distributed Smart Cameras*, page 23. ACM, 2018.

[16] Erik B Dam, Martin Koch, and Martin Lillholm. *Quaternions, interpolation and animation*, volume 2. Citeseer, 1998.

[17] Thomas Debrunner. Automatic Code Generation and Pose Estimation on Cellular Processor Arrays (CPAs). Master's thesis, September 2017.

[18] Thomas Debrunner, Sajad Saeedi, Laurie Bose, Andrew J Davison, and Paul HJ Kelly. Camera Tracking on Focal-Plane Sensor-Processor Arrays.

[19] Thomas Debrunner, Sajad Saeedi, and Paul HJ Kelly. Auke: Automatic kernel code generation for an analogue simd focal-plane sensor-processor array. *ACM Transactions on Architecture and Code Optimization (TACO)*, 15(4):59, 2019.

[20] Jakob Engel, Thomas Schöps, and Daniel Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *European conference on computer vision*, pages 834–849. Springer, 2014.

[21] Martin A Fischler and Robert C Bolles. Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6):381–395, 1981.

[22] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. SVO: Fast semi-direct monocular visual odometry. In *2014 IEEE international conference on robotics and automation (ICRA)*, pages 15–22. IEEE, 2014.

[23] Daniel Gehrig, Henri Rebecq, Guillermo Gallego, and Davide Scaramuzza. Asynchronous, photometric feature tracking using events and frames. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 750–765, 2018.

[24] Neil J Gordon, David J Salmond, and Adrian FM Smith. Novel approach to nonlinear/non-Gaussian Bayesian state estimation. In *IEE proceedings F (radar and signal processing)*, volume 140, pages 107–113. IET, 1993.

[25] Colin Greatwood, Laurie Bose, Thomas Richardson, Walterio Mayol-Cuevas, Jianing Chen, Stephen J Carey, and Piotr Dudek. Tracking control of a uav with a parallel visual processor. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4248–4254. IEEE, 2017.

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

[27] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004.

[28] Richard I Hartley. In defence of the 8-point algorithm. In *Computer Vision, 1995. Proceedings., Fifth International Conference on*, pages 1064–1070. IEEE, 1995.

[29] Du Q Huynh. Metrics for 3d rotations: Comparison and analysis. *Journal of Mathematical Imaging and Vision*, 35(2):155–164, 2009.

[30] Eric Jones, Travis Oliphant, Pearu Peterson, et al. SciPy: Open source scientific tools for Python. `http://www.scipy.org/`, 2001. [Accessed: 2019-06-13].

[31] Hanme Kim, Stefan Leutenegger, and Andrew J Davison. Real-time 3d reconstruction and 6-DoF tracking with an event camera. In *European Conference on Computer Vision*, pages 349–364. Springer, 2016.

[32] Georg Klein and David Murray. Parallel tracking and mapping for small AR workspaces. In *Proceedings of the 2007 6th IEEE and ACM International Symposium on Mixed and Augmented Reality*, pages 1–10. IEEE Computer Society, 2007.

[33] Beat Kueng, Elias Mueggler, Guillermo Gallego, and Davide Scaramuzza. Low-latency visual odometry using event-based feature tracks. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 16–23. IEEE, 2016.

[34] Rainer Kümmerle, Giorgio Grisetti, Hauke Strasdat, Kurt Konolige, and Wolfram Burgard. g^2o: A general framework for graph optimization. In *2011 IEEE International Conference on Robotics and Automation*, pages 3607–3613. IEEE, 2011.

[35] 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, February 2009. ISSN 0920-5691, 1573-1405. doi: 10.1007/s11263-008-0152-6. URL `http://link.springer.com/10.1007/s11263-008-0152-6`.

[36] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. A 128×128 120 db 15μ s latency asynchronous temporal contrast vision sensor. *IEEE journal of solid-state circuits*, 43(2): 566–576, 2008.

[37] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[38] Elias Mueggler, Chiara Bartolozzi, and Davide Scaramuzza. Fast event-based corner detection.

[39] Raul Mur-Artal, Jose Maria Martinez Montiel, and Juan D Tardos. ORB-SLAM: a versatile and accurate monocular SLAM system. *IEEE Transactions on Robotics*, 31(5):1147–1163, 2015.

[40] David Nistér. An efficient solution to the five-point relative pose problem. *IEEE transactions on pattern analysis and machine intelligence*, 26(6):0756–777, 2004.

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

[42] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[43] Henri Rebecq and Titus Cieslewski. Mini-project: A visual odometry pipeline! `http://rpg.ifi.uzh.ch/docs/teaching/2018/vo_project_statement.pdf`. [Accessed: 2019-06-06].

[44] Henri Rebecq, Timo Horstschäfer, Guillermo Gallego, and Davide Scaramuzza. EVO: A geometric approach to event-based 6-DOF parallel tracking and mapping in real time. *IEEE Robotics and Automation Letters*, 2(2):593–600, 2016.

[45] Edward Rosten and Tom Drummond. Fusing points and lines for high performance tracking. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1508–1515. IEEE, 2005.

[46] Edward Rosten and Tom Drummond. Machine Learning for High-Speed Corner Detection. In Aleš Leonardis, Horst Bischof, and Axel Pinz, editors, *Computer Vision – ECCV 2006*, volume 3951, pages 430–443. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. ISBN 978-3-540-33832-1 978-3-540-33833-8. doi: 10.1007/11744023_34. URL `http://link.springer.com/10.1007/11744023_34`.

[47] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. ORB: An efficient alternative to SIFT or SURF. In *2011 International Conference on Computer Vision*, pages 2564–2571, Barcelona, Spain, November 2011. IEEE. ISBN 978-1-4577-1102-2 978-1-4577-1101-5 978-1-4577-1100-8. doi: 10.1109/ICCV.2011.6126544. URL `http://ieeexplore.ieee.org/document/6126544/`.

[48] Jianbo Shi and Carlo Tomasi. Good features to track. Technical report, Cornell University, 1993.

[49] BG Sileshi, Joan Oliver, and Carles Ferrer. Accelerating particle filter on fpga. In *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 591–594. IEEE, 2016.

[50] Hauke Strasdat, José MM Montiel, and Andrew J Davison. Visual SLAM: why filter? *Image and Vision Computing*, 30(2):65–77, 2012.

[51] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Science & Business Media, 2010.

[52] Rick Szeliski. Image Alignment and Stitching: A Tutorial. Technical Report MSR-TR-2004-92, October 2004. URL `https://www.microsoft.com/en-us/research/publication/image-alignment-and-stitching-a-tutorial/`.

[53] Sebastian Thrun, Dieter Fox, Wolfram Burgard, and Frank Dellaert. Robust Monte Carlo localization for mobile robots. *Artificial intelligence*, 128(1-2):99–141, 2001.

[54] Philip HS Torr, Andrew W Fitzgibbon, and Andrew Zisserman. The problem of degeneracy in structure and motion recovery from uncalibrated image sequences. *International Journal of Computer Vision*, 32(1):27–44, 1999.

[55] Shinji Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, (4):376–380, 1991.

[56] Antoni Rosinol Vidal, Henri Rebecq, Timo Horstschaefer, and Davide Scaramuzza. Ultimate SLAM? Combining Events, Images, and IMU for Robust Visual SLAM in HDR and High-Speed Scenarios. *IEEE Robotics and Automation Letters*, 3(2):994–1001, 2018.

[57] Matthew Wong. Analog Vision - Neural Network Inference Acceleration using Analog SIMD Computation in the Focal Plane. Master's thesis, September 2018.

[58] Georges Younes, Daniel Asmar, Elie Shammas, and John Zelek. Keyframe-based monocular SLAM: design, survey, and future directions. *Robotics and Autonomous Systems*, 98:67–88, 2017.

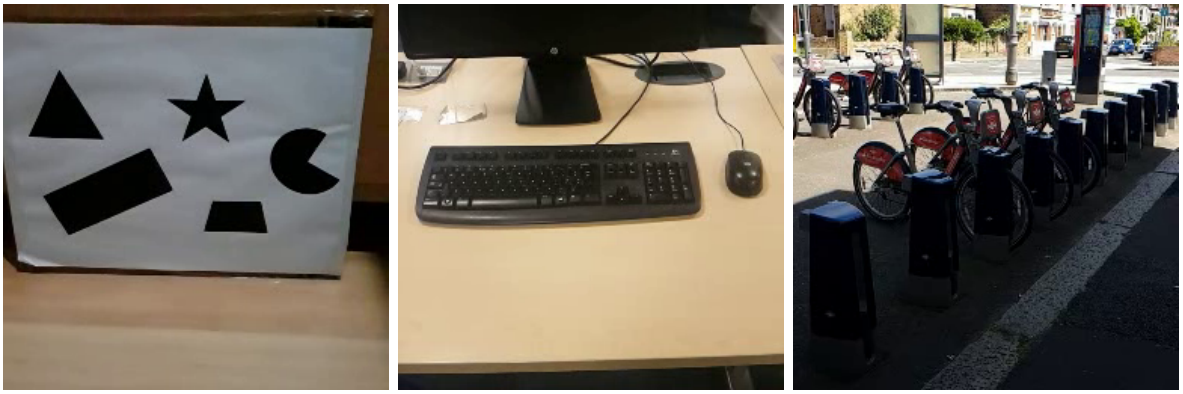
[59] A. Zarándy. *Focal-plane sensor-processor chips*. Springer New York, 2011. ISBN 978-1-4419-6474-8. doi: 10.1007/978-1-4419-6475-5.

[60] Zhengyou Zhang. Parameter estimation techniques: A tutorial with application to conic fitting. *Image and Vision Computing*, 15(1):59–76, 1997.

[61] Alex Zihao Zhu, Nikolay Atanasov, and Kostas Daniilidis. Event-based feature tracking with probabilistic data association. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4465–4470. IEEE, 2017.

Appendix A

Test Data



(a) Shapes dataset (b) Keyboard dataset (c) Bicycle dataset

Figure A.1: Snapshot of a frame from the different datasets.

To perform fair comparison of the algorithms, it is important that they are evaluated under the same configurations. The nature of the FPSP device makes this difficult, however, we present a method which allows a fair comparison between algorithms for the FPSP device, and further a fair comparison against frame-based algorithms.

The FPSP device applies to $I_t \in \mathbb{Z}^{N \times N}$ the pixel values at time t, a user defined function f on the chip, producing the data $D \in \mathbb{Z}^M$. Note that to utilise the FPSP device to its full potential, N^2 should be much larger than M. Furthermore, the functions are typically one-way, meaning that I_t is not obtainable from D. Since only D is transferred to the micro-controller, it is not possible to access I_t. For the algorithms which only depends on D, test data can be created by logging the output of the FPSP device. However, for comparison of the algorithms which operates on the device, I_t is necessary.

As I_t is not available from the device, alternative solution is to use the simulator with simulated dataset. The advantage of the simulated data is that it is possible to generate the scene at the frame-rate of the FPSP device, modelling the small inter-frame motions. However, the dataset does not contain noise, which is to be expected when capturing frames on a physical device. An alternative approach is to create test data using standard-camera. It allows test data to be created from natural scenes, which is a more accurate representation of the scene which the camera is likely to be used. However, the frame-rate is limited, thus any small abrupt motion would make the test-data unrepresentative of the scene which the FPSP device may see.

Instead, we generate our test data using high-speed camera which is capable of capturing video at

240FPS. This provides a nice middle-ground, allowing small inter-frame motions and capturing of the real-scenes. While 240FPS is insufficient for some of the applications of the FPSP devices, for our purpose, it is sufficiently fast, as long as we move the camera at a reasonable speed.

We provide 3 datasets which we use in our dissertation.

1. **shapes:** Contains black shapes on a white background, providing high-contrast and sharp edges. The scene is recorded in an indoor environment. A snapshot of the dataset is shown in Figure A.1a.
2. **keyboard:** Is a scene of a desk-top, with a keyboard and monitor in the view. There are less texture and contrast in the scene. The scene is recorded in an indoor environment. A snapshot of the dataset is shown in Figure A.1b.
3. **bicycle:** Is a scene of a bicycle docking station, with half of the image in shadow. This creates high intensity difference in different parts of the image. The scene is recorded in an outdoor environment. A snapshot of the dataset is shown in Figure A.1c.

Appendix B

Feasibility of the Future Works

To demonstrate the feasibility of transferring pixel information while performing our visual odometry pipeline, we have executed our pipeline and transferred pixel intensity information at 4FPS. Due to the limitation in the time, we could not obtain the ground truth trajectory from the Vicon motion capture system. However, the circular shape of the trajectory, as shown in Figure B.3 resembles the motion which we have performed, suggesting that the tracking was successful.

Figure B.2 shows the latency present in the system. We notice that the latency spikes up when we transfer the pixel data. However, the average latency of the system is below 5ms. Figure B.1 shows the snapshots which we have obtained during the execution of our pipeline. We notice that there is sufficient intensity information to extract useful information such as ORB descriptors.

These results suggest that transferring pixel information while performing low latency visual odometry is possible and is a promising direction for future research.

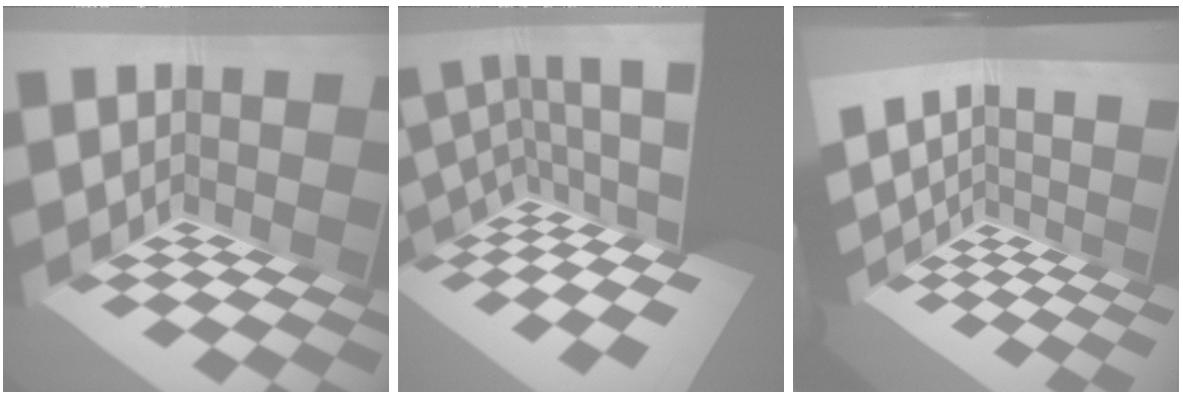


Figure B.1: Snapshots of frames obtained from the FPSP device while operating our visual odometry pipeline.

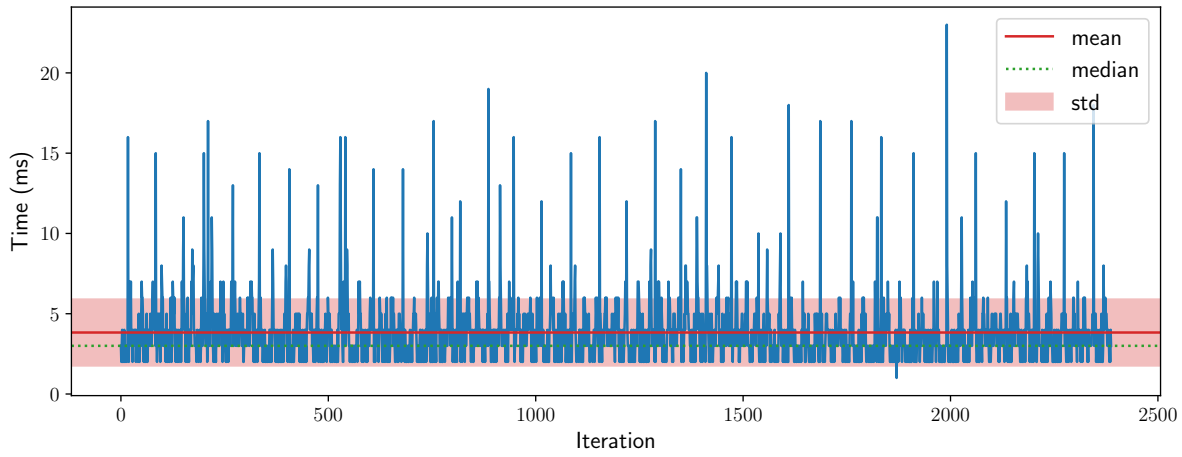


Figure B.2: Visualisation of latency of our visual odometry pipeline, while transferring pixel information .

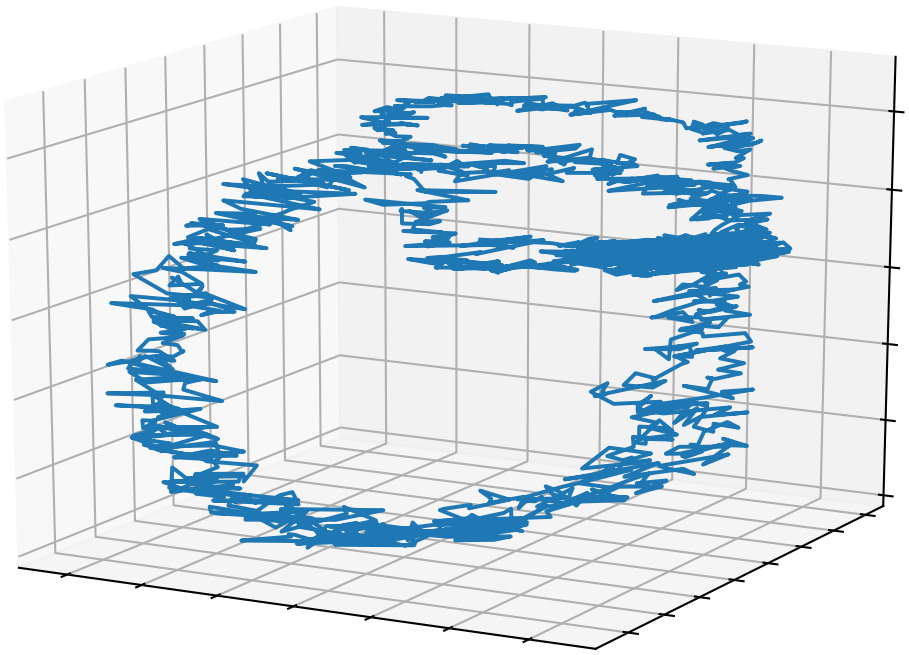


Figure B.3: Trajectory produced the visual odometry pipeline while transferring pixel information.