FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatik

Visual Inertial Control of a Nano-Quadrotor

Oliver Montague Welton Dunkley

FAKULTÄT FÜR INFORMATIK

DER TECHNISCHEN UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatik

Visual Inertial Control of a Nano-Quadrotor

Visioinertialregelung eines Nanoquadrotors

Author:	Oliver Montague Welton Dunkley
Supervisor:	Prof. Dr. Daniel Cremers
Advisors:	Dr. Jürgen Sturm & Jakob Engel
Date:	September 15, 2014

Ich versichere, dass ich diese Masterarbeit selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel verwendet habe.
I assure the single handed composition of this master's thesis, only supported by declared resources.

München, den 15. September, 2014
Munich, September 15, 2014 Oliver Montague Welton Dunkley

Acknowledgements

I would hereby like to express my gratitude and appreciation to all the people who helped and made it possible for me to create this thesis.

Everything started after one of Dr. Jürgen Sturm's amazing lectures on Visual Navigation for Flying Robots. I met him with great enthusiasm in his office after one of his lectures together with Jakob Engel, whose thesis on camera-based quadrotor navigation I had just read, to discuss my prospective master thesis plans. Bitcraze's Crazyflie quadrotors had just started shipping and were a hot topic of discussion. I distincty remember stating to the two: 'I would like to achieve what Jakob did on that quadrotor!', referring to the autonomous control Jakob implemented for the Ardrone and pointing to a video of the Crazyflie in action. My mind was fixed and the goal was set, and I knew I had the support and mentorship required to pull it through. Looking back, I cannot believe how naïve and stubborn I must have been.

Therefore, I would first like to thank Prof. Dr. Daniel Cremers for building up and maintaining the Computer Vision Group, an amazing place with frequent discussions among intellectual, respectful, diverse and generous personalities. It was here that I met Dr. Jürgen Sturm, whose enthusiasm for applying computer vision to robotic applications was infectious and extremely inspiring. I would also like to express my huge gratitude to Jakob Engel, who mentored me throughout the entire master thesis process, put up with my stubbornness, reliably helped me and kept me in check, all while giving me the freedom I required on this endeavour.

Furthermore, I would like to thank Bitcraze AB for both their astonishingly small Crazyflie quadrotor as well as their continuous on-going support and generosity. After many valuable online discussions with the Crazyflie developers it was wonderful to finally meet Tobias Antonsson in person; despite many hurdles, I never feared failure because I felt that between my supervisors, the Crazyflie team and myself every technical challenge could be overcome.

I would also like to thank Laurent Kneip as his work on using IMU measurements in geometric vision problems greatly inspired my thesis and the e-mail exchanges we had were very enlightening.

I am extremely grateful to my friends who supported and helped me complete this thesis. Specifically I would like to thank Ross Kidson for giving me a crash course on pose graph optimisation; Thomas Rothörl for his extremely flexible and reliable support ; Karol Hausman who made the long hours we spent at the lab far more enjoyable; and Michael Shelley, with whom I spent countless hours discussing and exchanging ideas with. Without their help, I would have had to have made many compromises and the thesis writing process would have been far less enjoyable and stimulating.

I am also extremely grateful to my beloved mother, father and brother for their continuous support and emotional foundation I could always count on. They were completely selfless, gave benevolent support and were always there when I needed them and also knew exactly when I needed my own space and time.

Everyone that I did not mention explicitly should still feel deeply thanked, such as my extended family and all my lab friends at CCRL!

My last thought goes towards Nika. I would like to thank her for carrying me through the stressful times and not only providing a rock-solid emotional foundation, but also for reminding me of the joys of life, removing all the stress I brought home with positive energy and delicious food as well as for the unconditional patience and support I knew I could always count on, all while never asking for anything in return. I owe it to you that our house is still standing!

Contents

Abstract

We augment a sub-19 g nano-quadrotor with a 6 g PAL micro-camera and wireless video transmitter, with which we demonstrate autonomous hovering and waypoint flying using a monocular visual-inertial keyframe based SLAM system running on a ground-based laptop. To our knowledge this is the lightest quadrotor capable of visual-inertial navigation with off-board processing. Furthermore, we show autonomous flight with external pose estimation, using a Kinect RGB-D camera, facilitating cheap and accessible external control for small spaces. We prove that the pose estimation and control systems work by obtaining results from real life experiments which we evaluate against ground truth from a motion capture system.

The hardware is, low-cost, robust, easily accessible, can safely be used in cluttered indoor environments and has freely available detailed specifications. We release all code in the form of an open-source ROS package to stimulate and facilitate further research into using nano-quadrotors as visual-inertial based autonomous platforms.

1 Introduction

Research interest in autonomous micro-aerial vehicles (MAVs) has grown rapidly in the recent years. On the one hand, we have seen aggressive flight manoeuvres using external tracking systems [1, 2] (e.g. Vicon), which are however limited to lab environments. On the other hand, there have been significant advances regarding autonomous flight in GPS-denied environments using cameras as main sensors both with on-board [3, 4] and off-board processing [5]. Furthermore, research has also been conducted into reducing the size of MAVs, especially to tackle indoor environments [6]. Frequently, the provided solutions require either expensive, research grade hardware [4] or custom hardware [7], which is often inaccessible to the masses. Therefore, we attempt to combine the latter three goals: We wish to develop a cheap and easily accessible way to perform autonomous flight on a minuscule aerial platform.

Even though current monocular-vision controlled MAVs are already much smaller than approaches based on RGB-D cameras, most models are still too large for indoor flight in a tightly constrained environment (e.g., a small office) or too dangerous to fly close to people. Particularly the weight of the platform plays a critical role, as it directly correlates with danger the MAV poses: even with a flight weight of only 500 g, a crash could do significant damage to both the MAV itself as well as its environment or human bystanders. Furthermore, size and weight influence the transportability, the usability e.g. as a flying scout for a ground-based robot, the potential to be deployed in large swarms and – in particular if mass-produced – the per-unit cost. While most research regarding MAVs focuses on – in the context of this thesis – relatively large platforms, there are some exceptions. In [8], a visual SLAM system is used on-board a quadrotor weighing around 100 g. In [9, 6], nano-MAVs (36 g and 46 g) are stabilized using optical flow. Most notably, in [7], a 20 g flapping-wing MAV is presented using on-board stereo vision for obstacle avoidance and navigation.

In this thesis, we present a 25 g nano-quadrotor based on the open-source, open-hardware Crazyflie by Bitcraze[1] equipped with an analogue on-board camera and wireless video transmitter. Using this platform, we demonstrate autonomous flight using the on-board camera and IMU for visual-inertial pose estimation. To the best of our knowledge demonstrating the smallest and lightest quadrotor with such capability.

[1]`http://www.bitcraze.se/`

1.1 Problem Statement

1.1.1 Goals

The goal of this thesis is to develop a way to achieve autonomous flight on a nano-quadrotor MAV, where

- the solution should be very accessible, e.g. by only using mass produced parts that can be easily assembled,
- the solution should be very affordable, thereby increasing accessibility,
- the solution should only depend on only open-source/open-hardware tools, hardware and libraries,
- the hardware should be low maintenance and robust against crashes,
- the software should be reliable and stable,
- the MAV should be so small that it poses no threat and can safely be used in populated indoor areas without concern,
- the MAV should be able to perform indefinite drift-free hover, even under small disturbances, only limited by the battery, and
- ideally, the MAV should be able to accurately perform waypoint following tasks too.

To satisfy the goals listed above, we need to solve a set of sub-problems that we can categorise into the following groups:

- **Hardware**
  - we require a small and lightweight MAV
  - we might need to augment the MAV with additional sensors
- **Pose Estimation**
  - we need a way to determine the roll, pitch, and yaw of the MAV at a high frequency (attitude estimation) to keep it level
  - we require a method to obtain drift-free pose estimates, allowing the MAV to relate its motion to the environment around it
- **Control**
  - using the attitude estimate, we require a controller to keep the MAV level
  - using the pose estimates, we need a position controller to to move the MAV to desired positions and headings

Each of these categories is covered on a per chapter basis.

1.1.2 Proposed Solution

To satisfy the goals listed above, we augmented the Bitcraze Crazyflie Nano-Quadrotor, a sub-19 g nano-quadrotor, with a 6 g PAL micro-camera and wireless video transmitter. We chose this as our hardware platform as it is relatively cheap, extremely small and lightweight (and therefore safe) and fully open-source and open-hardware. Furthermore, the quadrotor performs on-board attitude estimation and control.

We designed and implemented a monocular visual-inertial keyframe based SLAM system that makes use of the added on-board camera and telemetry data from the quadrotor for realtime pose estimation. Additionally, we also developed an external pose estimation algorithm that uses a Kinect and on-board IMU to determine the quadrotor pose.

These poses are then used by multiple PID controllers to control the position of the quadrotor, thus enabling drift-free hovering capability as well as waypoint flying.

1.2 Outline

- In chapter 1 we define some *conventions*, *terminology* and cover some transformation basics.
- In chapter 3 we first look at *quadrotors* in general, then take a deeper look at the *Crazyflie* and discuss how to augment it with a *wireless camera system*.
- In chapter 4 we discuss visual pose estimation and dive into the theory of the *visual-inertial SLAM system* we propose. Specifically, we look at how IMU measurements can speed-up and robustify the pose estimation pipeline.
- In chapter 5 we take a look at basic control theory and discuss what a *PID controller* is, how one can tune the gains, and how one can use them for *attitude and position control*.
- In chapter 6 we discuss all *implementation specific details*, such as the implemented hardware drivers, dealing with analogue camera images and how all the various components are put together into one coherent system.
- In chapter 7 we first *evaluate* the pose estimation methods as well as the hover and waypoint capabilities of the implemented system.
- In chapter 8 we wrap up the thesis and discuss potential *future work*.

2 Basics and Conventions

This chapter defines some of the more common parameterisations and notations used throughout the thesis.

2.1 Conventions

A few key notations are used throughout this thesis and summarised in Table 2.1 .

Table 2.1: Notation used throughout the thesis

Notation	Example
Frames / coordinate systems are calligraphic capitals	$\mathcal{W}$
Keyframes are numbered chronologically	$\mathcal{K}_0, \ldots, \mathcal{K}_i$
Scalars are lower case italics	x
Matrices are bold capitals	$\mathbf{X}$
Vectors are bold lower case	$\vec{\mathbf{y}}$
Coordinates of a point $\mathbf{p}$ in frame $\mathcal{W}$	${}^{\mathcal{W}}\mathbf{p}$
Transformation from frame $\mathcal{B}$ to frame $\mathcal{A}$	${}^{\mathcal{A}}_{\mathcal{B}}\mathbf{T}$
Translation vector; coordinates of frame $\mathcal{A}$ in $\mathcal{W}$'s coordinates	${}^{\mathcal{W}}_{\mathcal{A}}\vec{\mathbf{t}}$
Rotation of the frame $\mathcal{A}$ in $\mathcal{W}$'s coordinate system	${}^{\mathcal{W}}_{\mathcal{A}}\mathbf{R}$
Time of *event* on time line Q	${}^{Q}t_{event}$
Identity Matrix with zeros appended to make shape $m \times n$	$\mathbf{I}_{m\times n}$
A vector of n ones $(1_1, \ldots, 1_n)$	$\mathbf{1}_n$
The i-th unit length bearing vector in the associated camera frame	$\vec{\mathbf{f}}_{\mathbf{i}}$
Unit length bearing vector expressed in frame $\mathcal{A}$	${}^{\mathcal{A}}\vec{\mathbf{f}}_{\mathbf{i}}$

2.2 Terminology

We define some frequent terminology that will be used throughout the rest of this chapter.

Landmarks Landmarks are 3D points usually expressed in the fixed world frame $\mathcal{W}$.

Camera Frame A calibrated camera with an associated pose and optionally associated IMU measurement. They observe landmarks. In this thesis, their measurements are always given in the form of unit bearing vectors pointing from the camera origin towards the landmarks.

Keyframe A keyframe is a sparse subset of camera frames

Baseline The magnitude of the translation between two frames

Map A collection of keyframes, landmarks, and their associated correspondences

Bearing Vectors A unit norm bearing 3-vector $\vec{\mathbf{f}}$ pointing away from a camera frame origin, with two degrees of freedom (azimuth and elevation) described in the camera reference frame, often refered to as 2D information.

Observation A 2D image measurement of a landmark as seen by a camera, usually represented as a bearing vector pointing towards the landmark from the camera origin.

Keypoint The two-dimensional location on the image plane, which together with a local appearance based description can be matched between images.

Correspondence A pair of observations from different camera frames (2D-2D correspondence) viewing the same landmark, or the a bearing vector and landmark it is pointing towards (2D-3D correspondence).

Absolute Pose The pose of a camera frame expressed in the world reference frame.

Relative Pose The pose of a camera frame with respect to another camera frame.

2.3 Rigid Transformations in 3D

This section discusses how to represent frames and points in three-dimensional Euclidean space $\mathbb{R}^3$. By attaching coordinate systems to objects this allows us to relate how objects are positioned and rotated relative to one another. For example, we attach an imaginary coordinate system to the center of the Crayzflie as illustrated in Figure 2.1.

2.3.1 Frames and Transformations

A frame is a coordinate system, which in this thesis will always be right handed and usually have X pointing forwards, Y left and Z up. Borrowing from aviation terminology, we call rotation around each of theses axes roll, pitch and yaw respectively. See Figure 2.1 for an illustration. Frame notation will always be capitalised calligraphic letters and we reserve $\mathcal{W}$ for the world frame, $\mathcal{I}$ for the IMU frame and $\mathcal{C}$ for the camera frame.

The relationship between two frames can be seen as a relative pose. Given two frames $\mathcal{A}$ and $\mathcal{W}$, the pose of $\mathcal{A}$ in $\mathcal{W}$ is given by the translation from $\mathcal{W}$'s origin to $\mathcal{A}$'s origin, and the rotation of $\mathcal{A}$'s coordinate axes in $\mathcal{W}$. This relative pose can also be seen as a transformation ${}^{\mathcal{W}}_{\mathcal{A}}\mathbf{T}$, mapping points from $\mathcal{A}$ to $\mathcal{W}$. See Figure 2.2 for an example.

Composition is associative but not commutative. As ${}^{\mathcal{W}}_{\mathcal{A}}\mathbf{T}$ transforms from frame $\mathcal{A}$ to frame $\mathcal{W}$, anything being multiplied from the right must be in frame $\mathcal{A}$ and anything being left multiplied must be in frame $\mathcal{W}$.

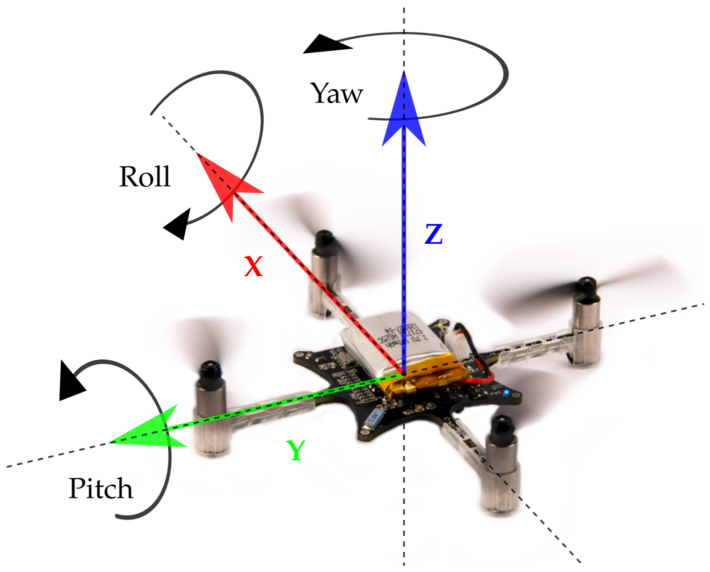


Figure 2.1: Crazyflie coordinate system convention. The forward direction is aligned with the X axis, the Z axis points up.

We describe the 3 DOF translational displacement (up/down, left/right, forward/backward) and 3 DOF rotational displacement (pitch, yaw, roll) of rigid bodies relative to a reference coordinate system with the use of rotation matrices and translation vectors respectively. For a full 6 DOF description, we use homogeneous transformation matrices, the explanation of which will conclude this section.

2.3.2 Translations

Translation is where an object is displaced along its reference frame's axis without any rotation. The translation vector ${}^{\mathcal{W}}_{\mathcal{O}}\vec{\mathbf{t}} = (t_x t_y t_z)^\intercal$ displaces an object $\mathcal{O}$ along the x, y, z axis of it's reference frame $\mathcal{W}$, i.e. it is a vector in $\mathcal{W}$'s coordinates. This could also be seen as a point ${}^{\mathcal{W}}\mathbf{p}$ in frame $\mathcal{W}$ in which case we omit the subscript as a point has no axes. Translations are concatenated with vector addition and inverted by negating the vector.

2.3.3 Rotations

A rotation matrix describes the relative orientation of an object to a reference frame. Rotation matrices are proper orthogonal matrices with the properties $|\mathbf{R}| = 1$, $\mathbf{R}^\intercal\mathbf{R} = \mathbf{R}\mathbf{R}^\intercal = \mathbf{I}$ and therefore $\mathbf{R}^\intercal = \mathbf{R}^{-1}$. They constitute the group of proper orthogonal transformations, or special orthogonal group **SO(3)**. Note that the group operator is the standard matrix product, giving the property that multiplying any two rotation matrices results in another rotation matrix.

The columns of **R** from an orthonormal vector base and represent the coordinates in the reference frame of unit vectors along the coordinate axes of the object. We can then define the three individual rotation matrices around the X,Y, and Z axes:

$$\mathbf{R}_x(\alpha) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{pmatrix} \quad \text{Roll, rotation around the X axis} \tag{2.1a}$$

$$\mathbf{R}_y(\beta) = \begin{pmatrix} \cos(\beta) & 0 & -\sin(\beta) \\ 0 & 1 & 0 \\ \sin(\beta) & 0 & \cos(\beta) \end{pmatrix} \quad \text{Pitch, rotation around the Y axis} \tag{2.1b}$$

$$\mathbf{R}_z(\gamma) = \begin{pmatrix} \cos(\gamma) & -\sin(\gamma) & 0 \\ \sin(\gamma) & \cos(\gamma) & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \text{Yaw, rotation around the Z axis} \tag{2.1c}$$

The roll, pitch and yaw rotations can then be used to place a 3D body in any orientation. A single rotation matrix can be formed by consecutively multiplying the above rotation matrices one after another to obtain:

$$\mathbf{R}(\gamma, \beta, \alpha) = \mathbf{R}_z(\gamma)\,\mathbf{R}_y(\beta)\,\mathbf{R}_x(\alpha) = \begin{pmatrix} \cos\gamma\cos\beta & \cos\gamma\sin\beta\sin\alpha - \sin\gamma\cos\alpha & \cos\gamma\sin\beta\cos\alpha + \sin\gamma\sin\alpha \\ \sin\gamma\cos\beta & \sin\gamma\sin\beta\sin\alpha + \cos\gamma\cos\alpha & \sin\gamma\sin\beta\cos\alpha - \cos\gamma\sin\alpha \\ -\sin\beta & \cos\beta\sin\alpha & \cos\beta\cos\alpha \end{pmatrix}. \tag{2.2}$$

We denote the relative orientation of an object frame $\mathcal{O}$ with respect to a reference frame $\mathcal{R}$ as ${}^{\mathcal{R}}_{\mathcal{O}}\mathbf{R}$. Note that since ${}^{\mathcal{B}}_{\mathcal{A}}\mathbf{R}$ is orthonormal,

$${}^{\mathcal{B}}_{\mathcal{A}}\mathbf{R} = {}^{\mathcal{A}}_{\mathcal{B}}\mathbf{R}^{-1} = {}^{\mathcal{A}}_{\mathcal{B}}\mathbf{R}^{\intercal} \tag{2.3}$$

holds, which is a wonderful property as it allows us to efficiently invert the rotation matrix, giving us the opposite rotation.

2.3.4 Homogeneous Transformations

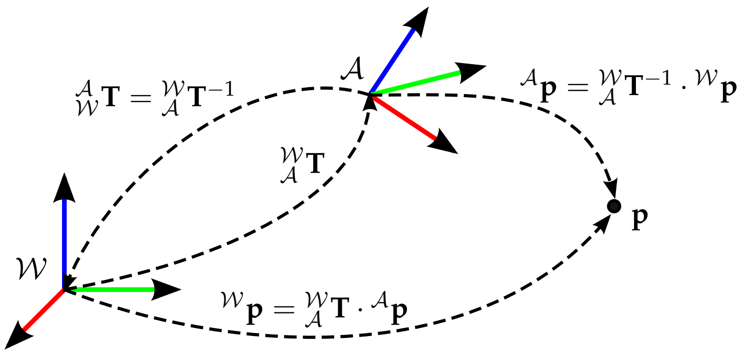
The group of rotation matrices $\mathbf{R} \in \mathbf{SO(3)}$ represent pure rotations only. In order to include translation, we first transform and then rotate. For example, given a frame $\mathcal{A}$ with respect to a frame $\mathcal{B}$ and a point ${}^{\mathcal{A}}\mathbf{p}$ with respect to $\mathcal{A}$, we can express the same point ${}^{\mathcal{W}}\mathbf{p}$ in the frame of $\mathcal{W}$ by first translating it with ${}^{\mathcal{W}}_{\mathcal{A}}\vec{\mathbf{t}}$ and then rotating it with ${}^{\mathcal{W}}_{\mathcal{A}}\mathbf{R}$ (scenario illustrated in figure 2.2 below):

$${}^{\mathcal{W}}\mathbf{p} = {}^{\mathcal{W}}_{\mathcal{A}}\mathbf{R} \cdot {}^{\mathcal{A}}\mathbf{p} + {}^{\mathcal{W}}_{\mathcal{A}}\vec{\mathbf{t}} \tag{2.4}$$

However, it is more convenient to express the full transformation as a matrix. We append a translation vector extended with a fourth *homogenous coordinate* $\vec{\mathbf{t}}$ to $\mathbf{R}$, resulting in a 4×4 matrix $\mathbf{T}$ with the following form:

$${}^{\mathcal{W}}_{\mathcal{A}}\mathbf{T} = \begin{pmatrix} {}^{\mathcal{W}}_{\mathcal{A}}\mathbf{R} & {}^{\mathcal{W}}_{\mathcal{A}}\vec{\mathbf{t}} \\ 0_{1\times 3} & 1 \end{pmatrix} \tag{2.5}$$

This set of transformation matrices $\mathbf{T}$ where $\mathbf{R} \in \mathbf{SO(3)}$ and with the matrix product as group operation form the special Euclidean group $\mathbf{SE(3)}$.



$^{\mathcal{A}}_{\mathcal{W}}\mathbf{T} = {}^{\mathcal{W}}_{\mathcal{A}}\mathbf{T}^{-1}$
$\mathcal{A}$
$^{\mathcal{A}}\mathbf{p} = {}^{\mathcal{W}}_{\mathcal{A}}\mathbf{T}^{-1} \cdot {}^{\mathcal{W}}\mathbf{p}$
$^{\mathcal{W}}_{\mathcal{A}}\mathbf{T}$
$\mathbf{p}$
$\mathcal{W}$
$^{\mathcal{W}}\mathbf{p} = {}^{\mathcal{W}}_{\mathcal{A}}\mathbf{T} \cdot {}^{\mathcal{A}}\mathbf{p}$

Figure 2.2: Duality between frames and transformations: Pose $\mathcal{A}$ in the frame $\mathcal{W}$ given by ${}^{\mathcal{W}}_{\mathcal{A}}\mathbf{T}$ can also be seen as transforming point $\mathbf{p}$ in $\mathcal{A}$'s frame (${}^{\mathcal{A}}\mathbf{p}$) to frame $\mathcal{W}$ (${}^{\mathcal{W}}\mathbf{p}$). Composing transforms and points: ${}^{\mathcal{W}}\mathbf{p}$ can be expressed via frame $\mathcal{A}$. Inverse composing of transforms and points: ${}^{\mathcal{A}}\mathbf{p}$ can be expressed via frame $\mathcal{W}$.

In order to transform points, we also append a fourth homogeneous coordinate to the point representation. Later, when we deal with perspective projections in chapter 4, we will need to the extra homogeneous coordinate to re-normalise our 3D points. The homogenous component can also be seen as a scaling factor, but for the remainder of this thesis we fix it to 1. Finally, we can simply multiply the transformation matrix with the homogeneous point to obtain the same result as in equation (2.4):

$$^{\mathcal{W}}\mathbf{p} = {}^{\mathcal{W}}_{\mathcal{A}}\mathbf{T} \cdot {}^{\mathcal{A}}\mathbf{p} \tag{2.6}$$

Looking at figure 2.2, we can easily follow the arrows to express $\mathbf{p}$ in the frame of $\mathcal{A}$ using the pose inverse composition:

$$^{\mathcal{A}}\mathbf{p} = {}^{\mathcal{A}}_{\mathcal{W}}\mathbf{T} \cdot {}^{\mathcal{W}}\mathbf{p} = {}^{\mathcal{W}}_{\mathcal{A}}\mathbf{T}^{-1} \cdot {}^{\mathcal{W}}\mathbf{p} \tag{2.7}$$

where the inverse of a transformation can efficiently be calculated thanks to equation (2.4):

$$^{\mathcal{A}}_{\mathcal{B}}\mathbf{T} = {}^{\mathcal{B}}_{\mathcal{A}}\mathbf{T}^{-1} = \begin{pmatrix} {}^{\mathcal{B}}_{\mathcal{A}}\mathbf{R}^{\intercal} & -{}^{\mathcal{B}}_{\mathcal{A}}\mathbf{R}^{\intercal} \cdot {}^{\mathcal{B}}_{\mathcal{A}}\vec{\mathbf{t}} \\ 0_{1\times 3} & 1 \end{pmatrix} \tag{2.8}$$

This will be used frequently to compute the relative position of 3D visual landmarks with respect to different camera poses in chapter 4.

Just as one can concatenate transformations and points, one can successively concatenate transforms with each other by multiplying them. See figure 2.3 for an example.

Composition is associative but not commutative. As ${}^{\mathcal{W}}_{\mathcal{A}}\mathbf{T}$ transforms from frame $\mathcal{A}$ to frame $\mathcal{W}$, anything being multiplied from the right must be in frame $\mathcal{A}$ and anything being left multiplied must be in frame $\mathcal{W}$.

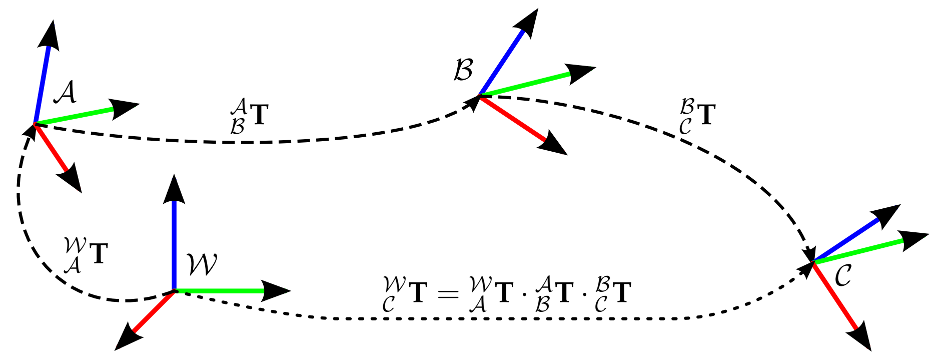

$\mathcal{A}$
$^{\mathcal{A}}_{\mathcal{B}}\mathbf{T}$
$\mathcal{B}$
$^{\mathcal{B}}_{\mathcal{C}}\mathbf{T}$
$^{\mathcal{W}}_{\mathcal{A}}\mathbf{T}$
$\mathcal{W}$
$^{\mathcal{W}}_{\mathcal{C}}\mathbf{T} = {}^{\mathcal{W}}_{\mathcal{A}}\mathbf{T} \cdot {}^{\mathcal{A}}_{\mathcal{B}}\mathbf{T} \cdot {}^{\mathcal{B}}_{\mathcal{C}}\mathbf{T}$
$\mathcal{C}$

Figure 2.3: Concatenating transformations: One can successively multiply transforms together to concatenate them. Here the pose of $\mathcal{C}$ with respect to frame $\mathcal{W}$ was computed via frames $\mathcal{A}$ and $\mathcal{B}$.

3 Hardware

In this chapter, we look at all the hardware used in this thesis. We start by exploring the history and principles of quadrotors in general before focussing on the CrazyFlie Nano Quadrotor, the vehicle of choice for this thesis. We then look at how to add a camera and image transmitter to the Crazyflie so that it can wirelessly send images to a computer for further processing.

3.1 Quadrotors

3.1.1 Background

A *rotorcraft* is a flying vehicle which generates thrust by employing rapidly spinning rotors that push air downwards. A *quadrotor* is a rotorcraft that has four such rotors, composed of two pairs of counter-rotating rotors usually equally spaced at the corners of a square frame. In addition to the four motors and associated sets of rotor blades, modern quadrotors consist of a power source, a flight control computer, various sensors and a frame that holds everything together. One often refers to hand held quadrotors as Micro Aerial Vehicles (MAV) and autonomous ones as Unmanned Aerial Vehicles (UAVs).

Flight characteristics

Quadrotors have flight behaviour similar to helicopters. They have a vertical take-off and landing (VTOL) capability, can perfectly hover and have excellent manoeuvrability and move in any direction independent of yawing. For more details on how quadrotors achieve such motion, see subsection 3.1.3. In contrast to fixed wing aircraft, this provides quadrotors with the ability to perform stationary observation tasks as well as manoeuvre slowly and precisely, allowing for motion in extremely constrained environments, such as within buildings.

Advantages

In comparison to helicopters, quadrotors use fixed pitch rotors that provide superior mechanical simplicity and require less maintenance due to the absence of complex mechanical control linkages for rotor actuation. Additionally, the use of four sets of rotor blades instead of an equivalent main rotor on a helicopter results in a smaller rotor radius. Therefore, the rotors store less kinetic energy during operation, reducing the risk of damage caused by rotor collision. These safety advantages greatly facilitate research and development, as tests can be done indoors by inexperienced pilots and with a shorter accident recover time.

Sensors

While mechanically extremely simple, a quadrotor is inherently unstable. They are almost impossible to control without the help of advanced electronic control systems that can react very quickly to counteract the tendency to tip over. Continous proprioceptive measurements (self sensing), such as those from an inertial measurement unit (IMU), are therefore required to estimate the behaviour to which a control system can react. In subsection 3.1.2 we give a brief overview of quadrotor related sensors, section 3.2.1 discusses the Crazyflie sensors suite, in section 4.3 we look at how an IMU can be used to estimate the rotation of the quadrotor, and in section 5.3 how one can use this information to stay level. Optionally, exteroceptive sensors (external sensing) can be added to move (or stay still) relative to the environment. This is one of the main motivations of this thesis: adding a camera to achieve position control. How one can use a monocular camera to estimate position is discussed in section 4.1 and how to do position control is explained in section 5.4.

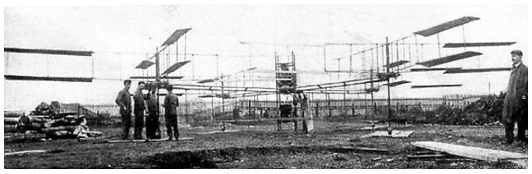
Quadrotor History

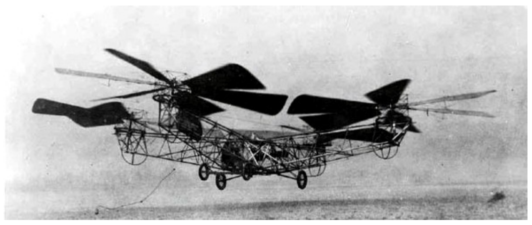
The early quadrotor designs date back to the 1920s [10], but these were unique, large machines with unstable flight characteristics and almost impossible to fly. Departing from the century-old design, modern quadrotors have evolved into small and agile vehicles that can be mass-produced [11], weigh less than 20 grams [12], fly autonomously and perform advanced aerobatic manoeuvres [1, 2].

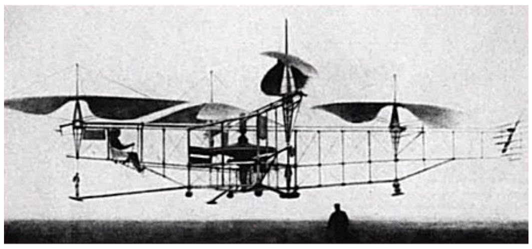
While seen as a physically mature platform today, the development of quadrotors had stalled until recently despite numerous design phases in the 1900s. The more common, traditional helicopter dominated as they were more power efficient and did not require the advanced electronic control and stabilisation systems modern quadrotors rely on.

Early Experiments, First lift off, 1907 Four years after the Wright brothers had flown the world's first controllable aircraft, the two brothers Louis and Jacques Bréguet, under the supervision of Professor Charles Richet, constructed one of the first quadrotors which they christened the *Bréguet Richet Gyroplane No. 1*. An image of it being prepared for its maiden manned flight is shown in Figure 3.1a.

In September 1907, the 46 hp, 578 kg quadrotor achieved lift off for the first time, albeit without control surfaces and only to an elevation of 60 cm [10]. It cannot however be credited as being the first rotorcraft to achieve free flight, as it could not be controlled and required to be steadied by four men surrounding it [10]. The men did not contribute to the lifting force, so it was the first rotorcraft to raise itself, with a pilot, vertically off the


(a) Bréguet and Richet, Gyroplane No. 1 (1907)


(b) George De Bothezat, Flying Octopus (1922)


(c) Étienne Oehmichen, No. 2 (1922)

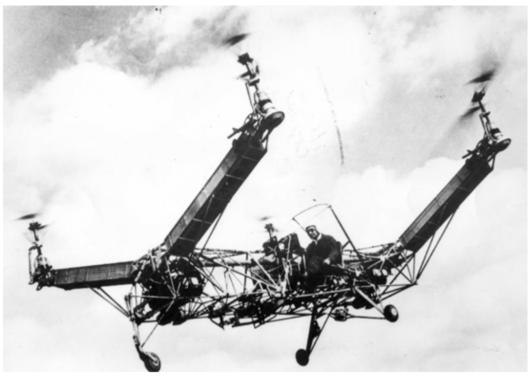

(d) Convertawings, Model A (1956)

Figure 3.1: Examples of early quadrotors of the 1900s.

ground. In subsequent experiments it achieved an altitude of just over 1.5 m and could remain airborne for a minute.

Due to its uncontrollable nature, testing was eventually discontinued in favour of building a new machine from the ground up, but the lack of contemporary engines with an adequate power/weight ratio caused Breguet to forgo rotary-winged research and development for the next 25 years.

The Flying Octopus, Progress but with complexity and Control Issues, 1922 After having written one of the first scientific papers on the aerodynamics of rotary-wing flight, the United States Army Air Service contracted George de Bothezat and his assistant Ivan Jerome to build an experimental helicopter [13]. He ended up with a 1.6 ton, 20 m $\times$20 m meter quadrotor that was officially named the de Bothezat helicopter and nicknamed the 'flying octopus', shown in Figure 3.1b.

Aside from the four 6 blade rotors, it also had two vertical 'steering air-screws' for lateral control. In 1922 it completed its first stable flight hovering around 1.8 m [14]. Once again, the quadrotor engine proved underpowered but after upgrading it a year later, over a hundred flights were conducted with a pilot and up to four passengers hanging onto the frame[14]. It set the rotorcraft records for flight time (2 minutes 45 seconds) and altitude (9.1 m)[10].

The program was cancelled in 1924, and the aircraft was scrapped due to complexity, high pilot workload, control difficulties and unreliability. However, despite the Army considering it a failure, the 'Flying Octopus' was ground breaking and the Army needed over twenty years to design a rotorcraft with better performance.

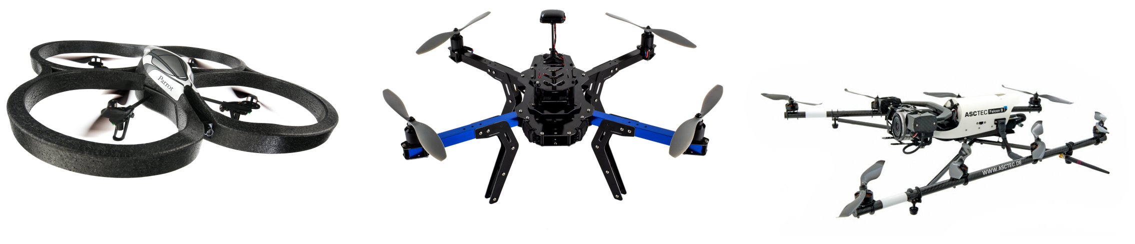
Breaking More Records, 1922 Around the same time but on the other side of the world, Etienne Oehmichen was busy building full-scale rotary-winged vehicles. Of special interest is the Oehmichen No. 2, an 800 kg quadrotor powered by a single 120 hp engine, shown in Figure 3.1c.

It featured the usual X shape with 4 dual blade rotors as known from modern quadrotors. Unlike our modern day counterparts, he could not control the individual rotor speeds, resulting in the need to add eight small propellers to control the lateral movement. He first flew it in 1922, and in 1923 broke the distance record for rotorcraft flight by flying 358 m. In May 1924 he broke his own record by flying 1 km along a triangular trajectory with a flight time of 07:40.

Despite this success, Oehmichen was dissatisfied with the Oehmichen No. 2s limited altitude and scrapped the quadrotor configuration, adopting a single main rotor design that could provide more thrust.

Further Development, 1956 Notwithstanding lots of single rotor innovation, it was not until 1956 that the next successful quadrotor was developed. Convertawing's Model 'A', the 1 ton quadrotor shown in Figure 3.1d, flew many flights and was the first quadrotor to be controlled by varying the thrust between rotors, effectively eliminating the need for additional lateral control rotors. Convertawings novel control system was a precursor to the modern control schemes used for quadrotors today.

Despite successful testing and development, the project was terminated due to a lack of interest from the commercial sectors and cutbacks in defence spending. Once again, research focused on improving the single rotor designs until the required technological advancements for small-scale quadrotors came about.



(a) Ready to Fly Toy **(b)** Hobbist Open Source Kit **(c)** Professional Grade

Figure 3.2: Examples of modern quadrotors, from off the shelf toys to professional research grade quadrotors.

Modern Quadrotors These days, quadrotors come in all forms and sizes. They are usually relatively small, unmanned aerial vehicles (UAVs), as controlling four rotors with four electric motors is very easy. Thanks to recent advances in robotics and micro-electromechanical systems, sensors have become smaller, lighter and easier to mass produce and enough processing power is available in small packages to allow for stable and efficient control at low cost. This enables UAVs to be obtainable outside research laboratories to the general public for very affordable prices, resulting in many consumer grade MAVs being sold through hobby stores and toy shops. Figure 3.2a shows an ArDrone, a mass produced quadrotor sold for €300 and Figure 3.2b shows a hobby grade drone, an example of one of the many open source drone platforms.

Quadrotors make ideal research platforms for the fields of flight control theory, navigation with obstacle avoidance, real time systems, swarm theory, robotics, and computer vision. UAVs are also being employed in various commercial and industrial applications, including the use of unmanned multicopters for crop dusting or precision farming autonomous geological remote sensing . , aerial mapping and cinematography and security-related tasks, such as remote inspections and surveillance. Figure 3.2c shows an Ascending Technologies Falcon Octocopter, a top of the line autonomous aerial camera platform.

As UAVs are ideal for reconnaissance and surveillance, the are many use cases for military and law enforcement agencies as well as search and rescue missions in urban environments.

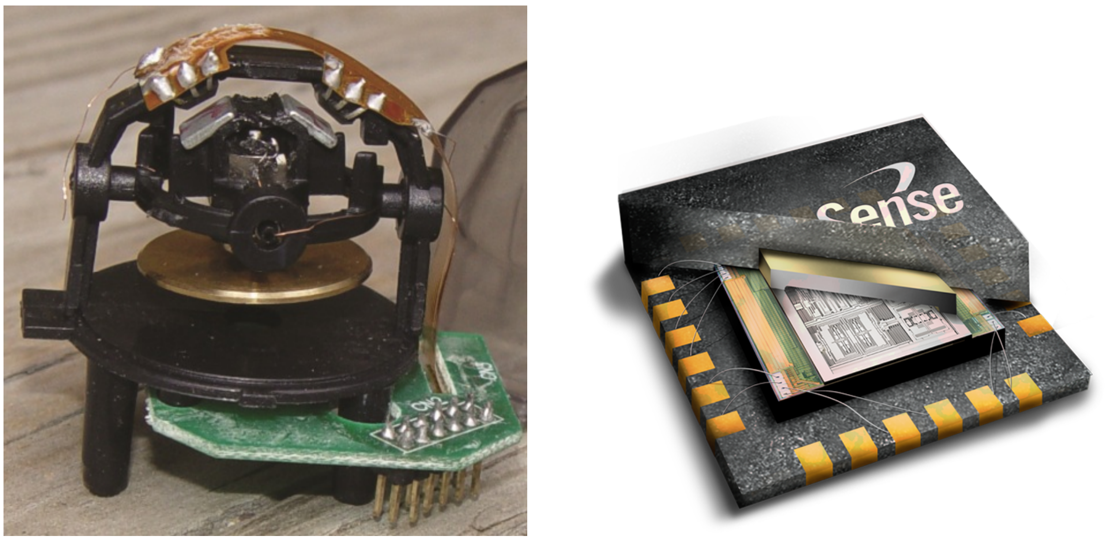
In December 2013, Deutsche Post gathered international media attention with the project *Parcelcopter*, in which the company tested the shipment of medical products by drone-delivery. Around the same time, Amazon's R&D lab shared their current project dubbed *Amazon Prime Air* with the goal of introducing a new delivery system where packages can be brought to customers in 30 minutes or less using unmanned aerial vehicles.

3.1.2 Quadrotor Sensors and Autonomy

To keep level autonomously, quadrotors require at minimum a gyroscope to stabilise their rotational velocity and an accelerometer to give the notion of which direction up is in (by measuring the direction of gravity).

Luckily, micro-electromechanical systems (MEMS) technology has come a long way recently resulting in a plethora of mass producible, very cheap and small gyroscopes and accelerometers often packaged together into a single inertial measurement unit (IMU). MEMS are one of the key enabling technologies that enabled MAVs to fly stably and shrink in size. Figure 3.3 shows modern IMU based on MEMS technology versus an older mechanical one.

However, using IMUs alone for navigation is infeasible as they typically suffer from cumulative errors. These occur as a result of integrating measurements over a time span: detected changes are only added to previously calculated estimations (also called dead-reckoning) and any incremental errors, however small, are accumulated from update to update. This leads to *drift*, an ever-increasing difference between the estimated state and the actual state.



(a) Mechanical Gyroscope **(b)** Inversense MEMS IMU

Figure 3.3: The left shows a mechanical gyroscope employing a spinning rotor and the right an MEMS IMU (accelerometer and gyroscope) using a vibrating element.

For example, consider you wish to navigate from the front door of your apartment to the fridge with your eyes closed. In your mind, you know the layout and dimensions of your home. You open the front door and take a few steps forward. At this point, you pretty much know where you are in your mental map. You walk a bit further, remembering you need to turn right soon to enter the kitchen. However, you become more insecure with every step and are uncertain when to start turning right, as you might have walked past or not yet reached the door you wish to enter. Your mental map of how you moved is based on dead-reckoning only. Now imagine you could open your eyes every 5 seconds. This would allow you to correct the uncertainty you accumulated while moving by readjusting the mental model of your path taken to adjust to your visual observations, thus absolutely positioning yourself in the mental map of your home.

For an UAV to become fully autonomous, it requires a method to determine its absolute pose, thereby alleviating the effects of drift. Only then can an MAV determine whether or not it is moving relative to its environment, enabling it to either autonomously (*a*) position hold, if the environment is not known, (*b*) move around, assuming the environment is assumed to be obstacle free (such as in the sky) or (*c*) navigate a known environment.

To do this IMUs are often coupled with a global positioning (GPS) unit to determine the quadrotor's absolute position in outdoor, uncluttered environments enabling them to autonomously follow or hover at predefined way-points. Ultrasonic range finders can be used to measure the distance to the ground, aiding in autonomous altitude control when flying at low altitudes and barometers can be used to measure air pressure to estimate altitude changes at higher altitudes. Magnetometers can be used to determine which direction north is in, removing rotational drift around the yaw axis. Laser scanners, depth/stereo cameras and even monocular cameras can also be used for pose estimation relative to the environment they observe. In this thesis, we use an IMU polled at 500 Hz to keep the quadrotor level and pose estimation from a monocular SLAM system at 25 Hz to help minimise drift.

To autonomously navigate in an unknown and/or dynamic environment, the UAV must observe its surrounds in real time to avoid collisions. Various sensors can be used for this, albeit all at a price-weight-power consumption-accuracy-precision trade-off. For example, a laser range finder provides dense and accurate distance measurements to the surrounding environment, but are power hungry, heavy and expensive; while a single monocular camera weights very little, is cheap but also has reduced accuracy, reduced frequency and is much harder to extract useful information from. This thesis does not tackle this problem and assumes the quadrotor is an obstacle free, non-hostile environment.

3.1.3 Quadrotor Flight Principles

Mechanical Simplicty vs Control Simplicity

Part of the appeal of multiple-rotor aircraft is that they can be designed without the need of complex cyclic-pitch-control systems, often allowing quadrotor propellers to be connected directly to individual motors. To achieve lateral thrust, a helicopter must rely on a mechanically complicated rotorhead that can change the pitch angle of the rotor blades cyclically by the means of a swash-plate. That is, the pitch of each rotor blade changes depending upon its position of each revolution so all blades have the same incidence at the same point in the cycle. This allows the blades to create more lift on one side of the revolution than the other, effectively tilting the helicopter in the opposite direction. A quad- (or multi-) rotor on the other hand can achieve the same effect by simply varying the thrust of each (comparatively smaller) motor individually. The complexity has shifted from a complex mechanical design (a complex rotor-head that allows cyclic and collective pitch control) to a digitally controlled control problem. Luckily, the decreasing cost and increasing computing power of microprocessors has allowed on-board flight computers to use on-board sensor data to stabilise the vehicle by changing the rotations per minute (RPM) of each of the motors commonly at 500 or 1000 times per second.

Flight Control

A quadrotor only has four motor angular velocities it can manipulate to achieve controlled flight. The motors are configured so that one pair of opposed rotors rotate clockwise and the other pair counter-clockwise. Each propeller produces both a thrust and torque around its center of rotation and a drag force opposite to the quadrotor's flight direction.

To induce yaw, the quadrotor can manipulate the net aerodynamic torque (and thus angular acceleration) by varying the ratio between the speed of the clockwise rotating motors and the counter-clockwise ones. Therefore, the yaw stabilising tail rotor of conventional helicopters is not required.

(a) HH-60H Seahawk Rotor Assembly

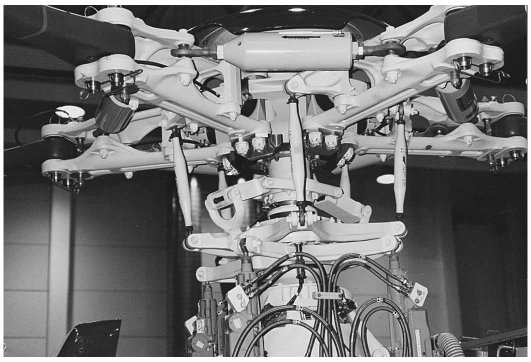

(b) EC155 Rotor head

Figure 3.4: The images above show two different rotor head assemblies. While a single rotor is more efficient, the additional complications of a swash-plate for cyclic-pitch-control might not outweigh the benefits of multiple direct-drive rotors.

Angular accelerations about the roll and pitch axes can be changed independently without affecting the yaw. The forward and rear pair of blades (rotating the same direction) pitch while the left and right pair control roll. By increasing the thrust of one rotor in a pair while decreasing thrust for the other one can induce a net torque resulting in roll or pitch motions while maintaining the torque balance needed for yaw stability. To achieve translational acceleration one simply maintains a non-zero roll or pitch angle. To increase or decrease altitude, one simply manipulates the net thrust. Thereby the quadrotor vehicle can manoeuvre in all dimensions using fixed rotor blades only. Figure 3.5 illustrates the motor velocity configurations required for different manoeuvres.

3.2 Crazyflie Nano Quadrotor

The Crazyflie is an extremely light and miniature nano quadrotor that fits in the palm of your hand. It was designed to be as simple as possible: a 4 layer printed control board (PCB) doubles as the frame, with the motors directly attached to it with the help of tiny plastic motor mounts that are simply pushed into place. This simplicity helps keep the platform robust, light-weight and easy to fix in the case of unfortunate crashes. In the remainder of this section we will briefly look at what the Crazyflie offers in terms of hardware, sensors and packaged software.

Weighing only 19 grams fully assembled and 85 mm motor to motor, it is extremely agile and safe to use even in crowded environments. Unlike most toys in this size range, it is fully programmable, has many sensors and can easily be controlled from a computer, this making it a great research development platform. The whole kit with some spare parts (as shown in Figure 3.7) can easily be acquired for roughly €180. Weights of each component are summarised in Table 3.1. Figure 3.8 highlights the tiny scale of the Crazyflie by showing its silhouettes next to a Parrot ArDrone.

The Crazyflie quadrotor project started in late 2009 as a competence development project

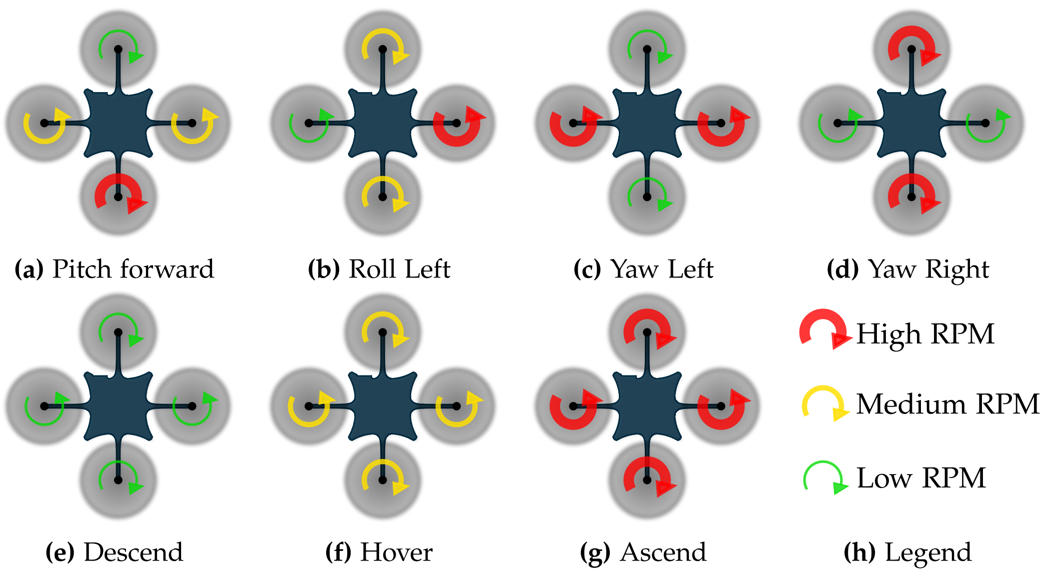


Figure 3.5: Quadrotor Mechanics: The left and right motors turn anti-clockwise, the forward and rear motors turn clockwise. By simply varying the speed of these 4 motors one can achieve positive and negative pitch, resulting in forward and backward motions; positive and negative roll, resulting in right and left translation; a yaw rotation; and altitude control.

Figure 3.6: At under 19 grams and 8.5 cm across, the Crazyflie can safely take-off from and land on the palm of your hand.

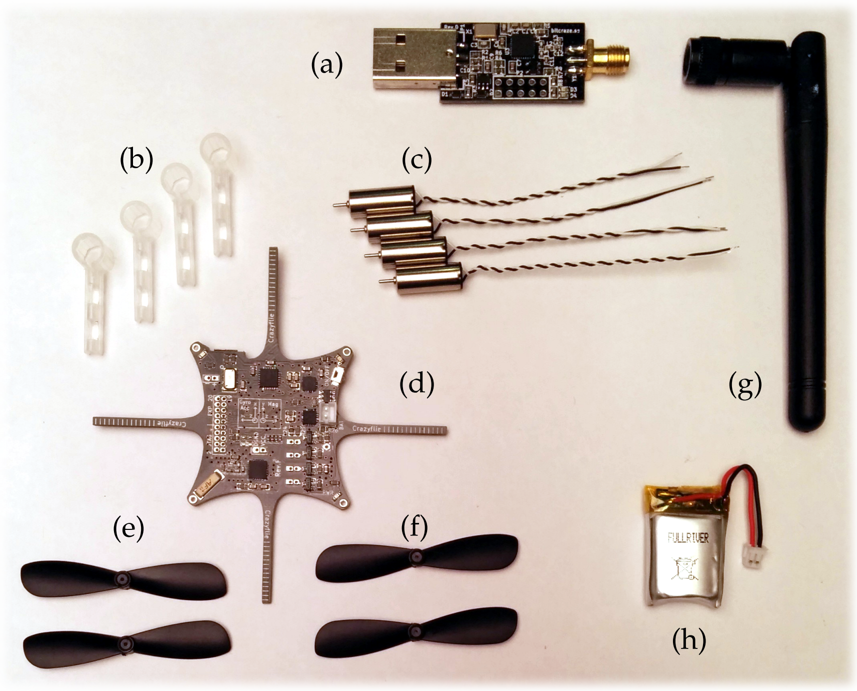


Figure 3.7: The contents of the Crazyflie kit, consiting of **(a)** 2.4 GHz USB dongle, **(b)** motor mounts, **(c)** 6 mm × 15 mm motors, **(d)** Crazyflie PCB, **(e)** clockwise rotating rotors, **(f)** anti-clockwise rotating rotors, **(g)** 2.4 GHz antenna for the dongle, **(h)** 3.7 V 170 mAh Li-Po battery. Spare parts are not shown.

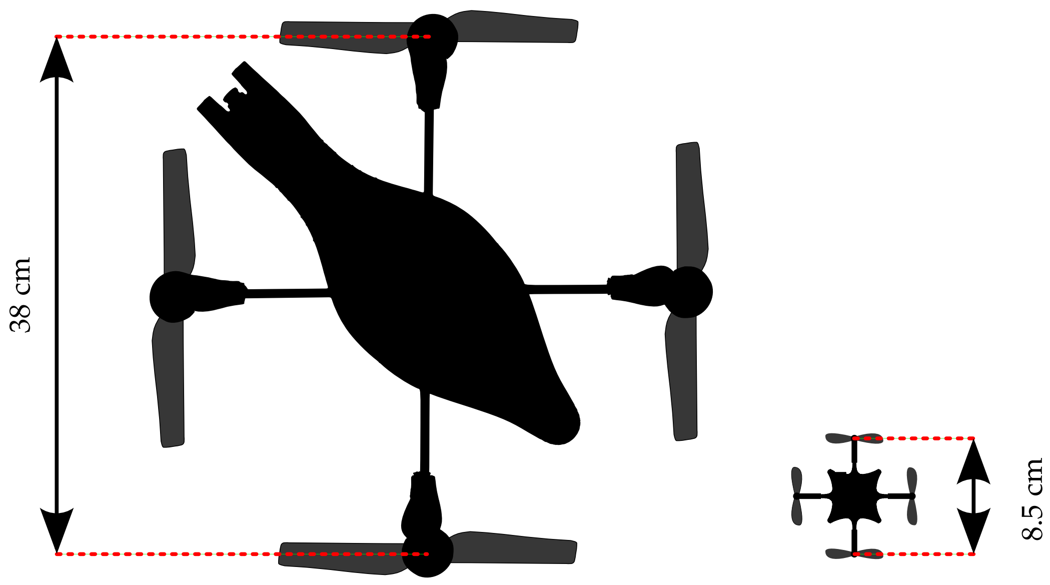


Figure 3.8: Size of the Crazyflie compared to the Parrot ArDrone. Relative scale of the silhouettes are correct.

in the Swedish consulting company Epsilon AB and resulted in the founding of Bitcraze AB in 2011. With the purpose to finance, develop and manufacture an open-source nano-quadrotor development platform, Bitcraze AB designed the Crazyflie solely using open-source tools and shipped the first units in April 2013. It is fully open-source and open-hardware.

3.2.1 Specifications

The Crazyflie comes in two versions: the slightly cheaper version omits the barometer and magnetometer sensors but is otherwise identical. For the remainder of this thesis we will assume a barometer is always present. A logical breakdown of the main hardware components is displayed in Figure 3.9.

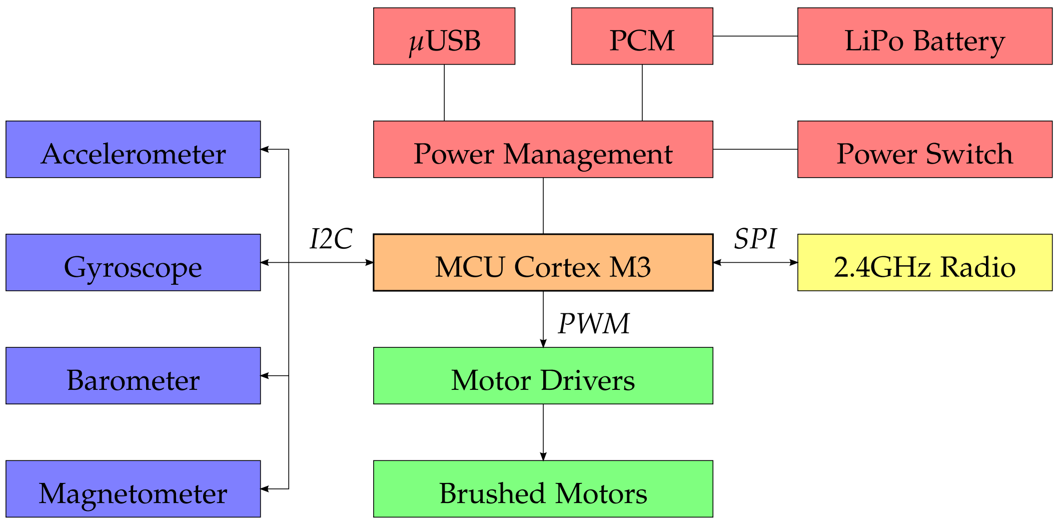


Figure 3.9: Logical break down of the main Crazyflie components. I2C is used to communicate with the on-board sensors, SPI is to communicate with the radio and PWM used to drive the brushed motors. The battery has a PCM (Protection Circuit Module) thereby significantly reducing the dangerous nature of LiPo batteries. The quadrotor handles power management and allows for charging via USB.

Sensors

The Crazyflie comes with the full sensor suite one might expect from a modern IMU. For an illustration and a photo of where the chips are located on the PCB, see Figure 3.10 and Figure 3.11 respectively.

Gyroscope and Accelerometer - MPU6050 The InvenSense MPU-6050 IMU combines a 3-axis gyroscope and a 3-axis accelerometer on the same silicon die resulting in a $4 \times 3 \times 0.9\text{mm}^3$ package. It can be polled at 1 KHz but the Crazyflie control loop only runs at 500 Hz. An accelerometer is a sensor that measures proper acceleration. Note that proper acceleration is not the same as the rate of change of velocity. For example, an accelerometer at rest on the surface of the Earth will measure an acceleration $g = 9.81m/s^2$. and measurements in free fall accelerating due to the gravity of Earth, will measure zero acceleration. This is useful, as it allows us to estimate the direction of gravity which can be used for low frequency attitude estimation.

On the other hand, a gyroscope is a device used for the measurement of angular velocity. It can only be used for high frequency attitude estimation, as one would need to integrate the angular velocity measurements to obtain a rotation estimate. Doing this is not feasible as small errors in the measurements accumulate into significant errors, causing long term drift.

Barometer - MS5611 A barometer is able to measure air pressure and temperature, from which one can determine the approximate altitude above sea level. The tiny $5 \times 3 \times 1mm^3$ sensor has a 10 cm resolution, 8.2 ms response time and can be sampled at 100 Hz.

Magnetometer- HMC5883L The Crazyflie also comes shipped with a 3-axis magnetometer, which is a type of sensor that measures the strength and direction of a local magnetic field. This would potentially allow the Crazyflie to determine its yaw relative to magnetic north. The magnetic field measured will be a combination of both the earth's magnetic field and any magnetic field created by nearby objects - resulting in the need to calibrate the compass prior to use. Two categories of distortion exist, hard iron and soft iron distortions. Hard iron distortions are created by objects that produce a magnetic field in the vicinity of the sensor, e.g. from magnets (used in motors) and power supply wires. If the piece of magnetic field is in the same reference frame as the sensor, then this type of hard iron distortion will cause a permanent bias in the sensor output. In the case of the Crazyflie, the currents generated by running the motors induces such a magnetic field which varies with the motor speed.

On the other hand, soft iron errors refer to the presence of ferromagnetic materials around the sensor that locally skew Earth's magnetic field, resulting in scaling offset errors. On the other hand, soft iron distortions are considered deflections or alterations in the existing magnetic field. These distortions will stretch or distort the magnetic field depending upon which direction the field acts relative to the sensor. This type of distortion is commonly caused by ferromagnetic materials around the sensor.

One can estimate these effects and calibrate for soft and hard iron distortions, however this is not a topic of this thesis and the magnetometer will not be used.

Motors

The Crazyflie uses four coreless, brushed, 6 mm ×15 mm 1.7 g DC motors coupled with 45 mm diameter propellers to generate enough thrust for dynamic flight and lifting small payloads. The motor driver is a simple pull down mosfet controlled by pulse width modulation (PWM). The motors rotate at roughly 21 000 RPM under load. They are probably

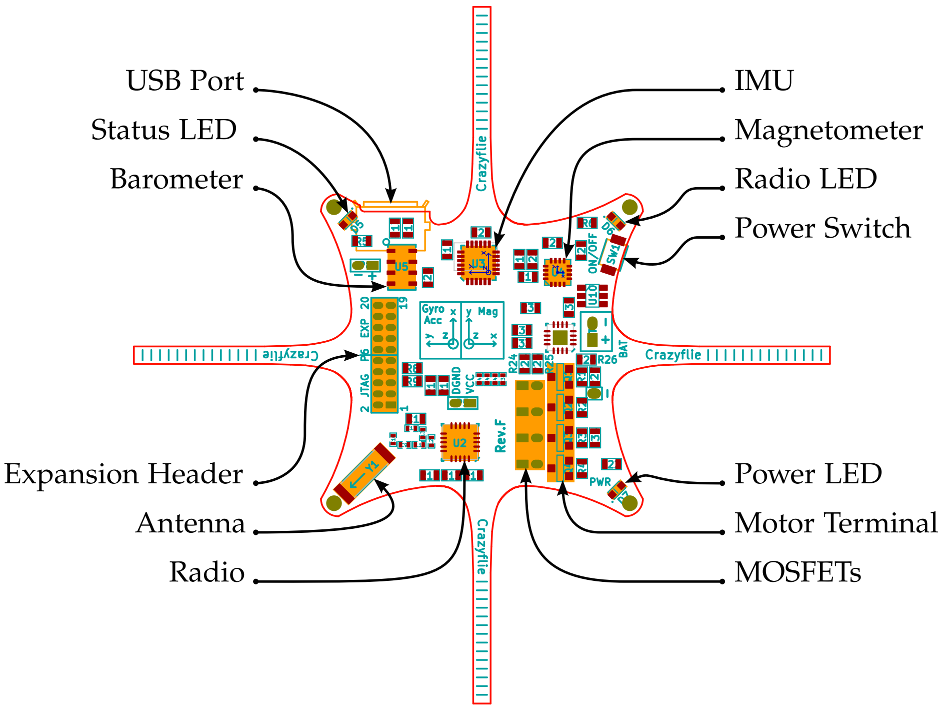


Figure 3.10: Flie PCB Components

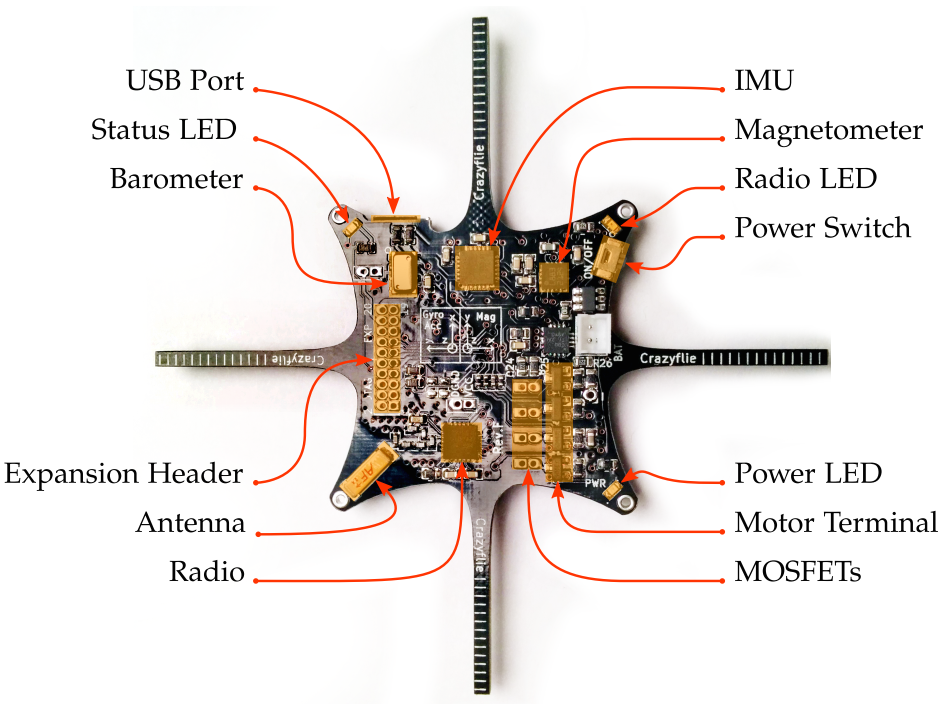


Figure 3.11: Flie PCB Components

the most fragile component of the quadrotor but can be replaced very easily and cheaply. In practice we replaced a broken motor every 8-12 severe crashes.

Communication

Communication is handled by a Nordic Semiconductor 2.4 GHz transceiver that can operate at 250 kb/s, 1 Mb/s and 2 Mb/s data-rates. It communicates with a client computer with its counterpart - a USB radio dongle called the *CrazyRadio* that comes packaged with the quadrotor. This allows for bi-directional communication with automatic address handling and packet acknowledgement. In practise, the range is enough for any line of sight in door operation but suffers from poor reception through buildings/walls. It has been tested up to 80 meters in ideal conditions. The antenna is located on the bottom right, top side, isolated part the PCB, away from any potential electrical noise (see Figure 3.11 for placement). For an overview of the communication protocol used, please see section 6.4.

On-Board Processing

The on-board MCU (Micro Controller Unit) used is the ST Microelectronics STM32F103CB. It runs at 70 Mhz, has 128 Kb flash and 20 Kb RAM. The firmware uses roughly half of the flash space and 16kb RAM. However, no optimisations have attempted to reduce the flash or RAM usage.

Power

The Crazy is powered by a Fullriver 3.7 V, 170 mAH 3.9 g Lithium Polymer battery. Flight time with different payloads is evaluated in subsection 7.1.1.

To handle some of the LiPo shortcomings, a PCM (Protection Circuit Module) is integrated into the battery that prevents the user from accidentally under/over charging or shorting the battery. The power management is handled by a Texas Instruments BQ24075 power management chip that handles the on/off logic and charging of the LiPo. The BQ24075 can limit the current to either 100 mA, 500 mA or a user selectable value, defaulting to 740 mA, this making it easier to comply to the USB standard on one hand but providing quick charge on the other. The battery is optimised for 510 mA so the quick charge is available as the users own risk. The Crazyflie PCB uses the separate power plane technique to reduce electrical noise by employing a 4 layer PCB. This is essential to avoid motor PWM noise interfering with the Radio.

3.2.2 Crazyflie Firmware

The Crazyflie runs a custom open-source firmware written in C that can be wirelessly flashed using the CrayzRadio. The firmware is based on the real time operating system

FreeRTOS which runs different tasks such as radio communication, stabilisation, power management, etc. at predefined frequencies.

Attitude Estimation

The Crazyflie performs attitude estimation, also known as *attitude and heading reference system (AHRS)*, on-board at 250 Hz. The goal is to estimate the roll, pitch and yaw (heading) as fast as possible by using gyroscope and accelerometer measurements. Please see section 4.3 for additional information.

Attitude Control

The Crazyflie does on-board attitude control using a cascaded PID controller. The inner loop runs at 500 Hz and controls angular velocity while the outer loop runs at 250 Hz and stabilises the attitude. Using the attitude estimation outlined above as input, the controller varies the power signal to the four motors keeping the Crazyflie at the desired attitude very effectively, despite the dynamic nature of such a small, lightweight platform. For more details, please see section 5.3

Changes

A few changes were made to the original firmware shipped with the Crazyflie.

CPU Utilisation FreeRTOS, the real time operating system the firmware is based on schedules *tasks* to run at specific rates. For example, the power management task might monitor battery discharge and run at 5 Hz and the attitude control tasks runs at 500 Hz. To measure the load of the Crazyflie processing ability, we measured the percentage of time it spends in idle task mode, i.e. when nothing else needs to be computed. Under normal conditions, the system load is usually around 70 Hz, depending on how much data is being sent and received over the radio.

Barometer Driver The Crazyflie can measure the air pressure and temperature using a barometer. The original drivers for the MS5611 barometric sensor were minimal and not optimised. Due to the noise of the measurements, it is important to have a high sampling rate one can then filter. Therefore, we reimplemented the driver to measure temperature at 10 Hz (the minimum amount required) and air pressure at 90 Hz - thereby reaching the maximum polling frequency the sensor can work with. For an evaluation and some example measurements, please refer to section 7.1.3.

Hover Mode using air pressure A PID controller was added to allow the Crazyflie to autonomously attempt to keep its current altitude. Once in fairly stable flight, the pilot can activate this mode and the Crazyflie will manage the net thrust itself, still allowing the pilot to manipulate target roll, pitch and yaw. Relative altitude differences are computed at 90 Hz by using air pressure measurements. However, this is only suitable for fairly short hover sessions as the air pressure generally is not reliable as shown in section 7.1.3.

Ping timing response A port was opened to handle special ping requests. When receiving a ping packet, the Crazyflie add's its current CPU time-stamp to the message and sends it back. This ping-pong with time-stamps is used in section 6.4 to help synchronise the Crazyflie clock with a computer clock to help estimate the communication latency.

3.3 Camera and Image Transmission

In this section we discuss the process of adding an on-board camera and image transmitter to the Crazyflie, allowing it to wirelessy transmit images in real-time to a ground station. This was exceptionally challenging due to the size, weight and power constraints. The recommended maximum payload is 4 g, beyond which flight performance and duration are significantly reduced. With such a small payload, we also need to directly power any additional hardware directly from the Crayzflie, as carrying an extra battery would consume the entire available payload. In subsection 3.3.1 we look at the chosen camera, in subsection 3.3.2 we discuss how to send and receive images, in subsection 3.3.3 we show how to power everything directly from the Crazyflie and in subsection 3.3.4 we demonstrate how to rigidly attach the system to the quadrotor.

3.3.1 Camera

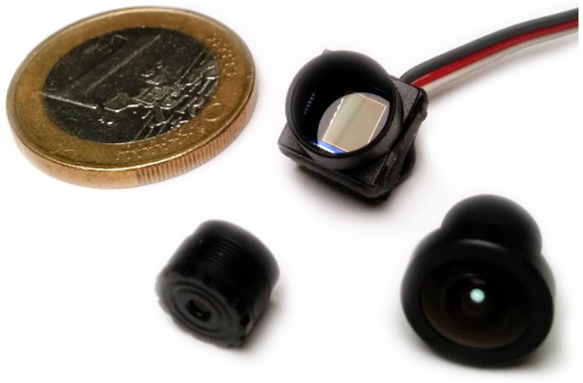


Figure 3.12: The camera and lens used relative to a €1 coin. The small lens on the lower left was shipped with the camera and not used due to its narrow 55° angle field of view. One can see the exposed imaging CMOS of the canera.

Ideally, we would have a high performance global-shutter CCD digital camera wirelessy sending images to a ground station over a digital connection. However, this is simply not feasible given the electrical power and processing power constraints. Therefore, we opted to use a conventional analogue camera.

The smallest and lightest camera we could find was the *1 Gram PAL Camera* bought online for €40.00 from `www.fpvhobby.com`, a hobby shop located in Turkey that ships world wide. The camera comes with a plastic cover that one can screw various lenses into. Two lenses come bundled with the camera, but as we wanted a wider field of view we also bought the *Nano Camera Wide Angle Lens* for €12 from the same shop. The camera and wide angle lens are are shown in Figure 3.12 and their specifications summarised in Table 3.1.

Table 3.1: Camera and Wide Angle Lens Specifications

Camera Weight	1.49g
Resolution	720×576
Sensor	1/3" CMOS
Video Standard	576i50
Current	75 mA
Voltage	3.2-5V
Camera Dimensions	$10.0 \times 10.0 \times 10.8\,\text{mm}^3$
Lens Weight	1.58g
Horizontal Field of View	110°
Diagonal Field of View	138°
Lens Diameter ×Length	11 mm ×8.7 mm

The camera is a hobby grade component at best which is reflected in the build quality. The plastic housing is attached with glue and is not very sturdy. The lens screws into the holder pretty loosly, and as the focal length is dependent on how far one screws it it, one must find a way to lock it in place. We added a thin layer of tape around the thread of the lens so that when screwing it in, it locks into place just as the the lens reaches the focal distance required.

Interlaced Video

The camera uses the 576i50 video standard, so it has vertical resolution of 576 scan lines, each 720 pixels wide. The *i* stands for interlaced, so the camera outputs 50 *fields* per second (25 even, 25 odd), giving 25 *frames* per second (FPS) where each frame contains an even and an odd field combined. See Figure 3.13 for an illustration. Note that this implies that the fields are taken in succession and not at the same time, therefore each frame has older data in the even rows which can cause visual artefacts when observing movement. As we wish to use the camera for VI-SLAM while the Crazyflie is flying, we deal with this problem in subsection 6.6.3.

Rolling Shutter

When evaluating an imaging sensor, one must distinguish between *global shutter* sensors and *rolling shutter* ones. A global shutter imager captures the entire field/frame at once while a rolling shutter sensor scans across the sensor rapidly, either vertically or horizontally (scan lines). Therefore, not all parts of the image are recorded at exactly the same

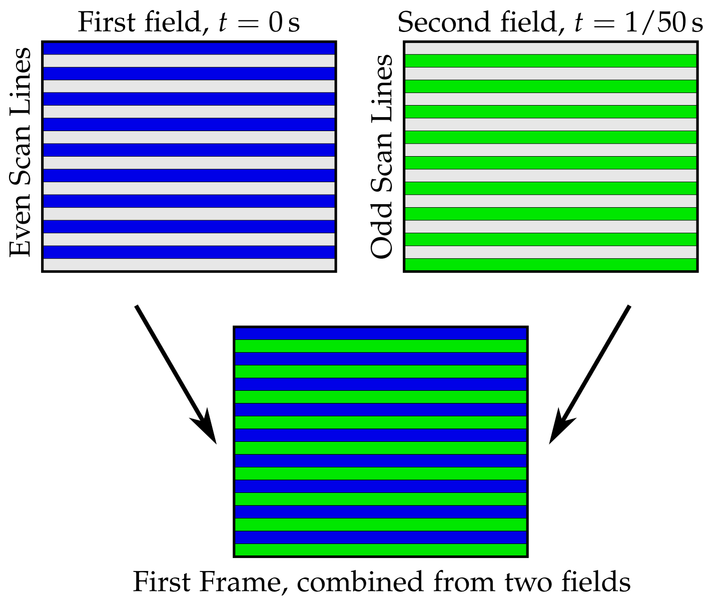

First field, $t = 0\,\text{s}$
Second field, $t = 1/50\,\text{s}$
Even Scan Lines
Odd Scan Lines
First Frame, combined from two fields

Figure 3.13: Illustration of the interpolation scheme given by the 576i50 video standard. The camera first captures a field with the even scan lines, then the odd, and finally packs them together into a frame. Each field has a 288×720 resolution resulting in an output frame with a resolution of 576×720, where the even rows were captured 1/50 s earlier than the odd rows.

instant, which can result in various distortion effects such as wobble and smear. These distortions are especially pronounced when the camera is recording a moving object or moving itself (e.g. due to vibration or when mounted onto a moving vehicle). There are ways to model and compensate for a rolling shutter. For example, [15] proposes a mathematical model of the rolling shutter to determine the relative image motion between an object and the camera. After fitting a polynomial to smooth the estimated motion, it can be used to align the scan-lines correctly. Alternatively, [16] presents a robust, real-time video stabilization and rolling shutter correction technique using commodity gyroscopes to effectively correct rolling shutter warping.

Unfortunately but not surprisingly, our chosen camera is of the rolling shutter type. However, modelling and compensating for it is not addressed further in this thesis and we assume the camera to use a global shutter.

Fish-Eye Distortion

As we use a wide angle lens with a 138° diagonal field of view, the resulting camera images are heavily distorted. For example, all the lines of the chess board in Figure 3.14 should be straight and parallel. We model and compensate for this distortion type in subsection 4.1.5

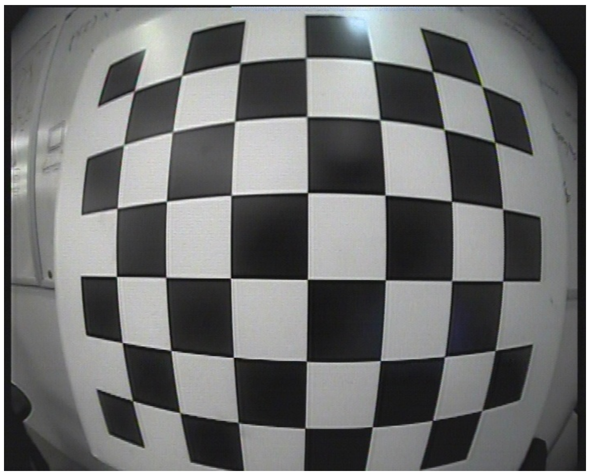
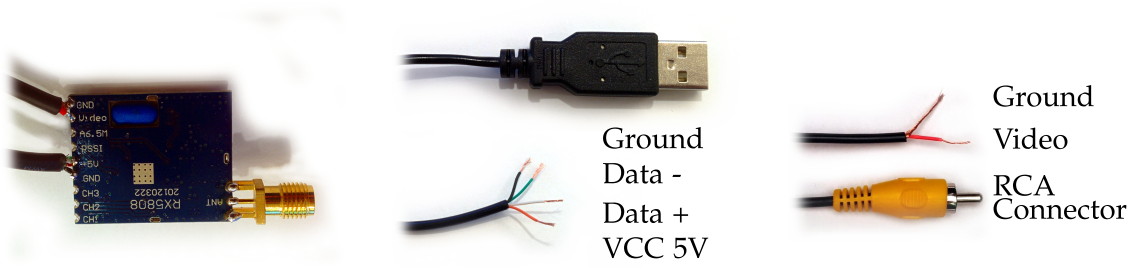


Figure 3.14: Fisheye distortion caused by the wide angle lens. Ideally, the edges of the black squares would all be straight and parallel.

3.3.2 Image Transmission and Reception

The only way to wirelessly transmit an image from the Crazyflie mounted camera to a ground station was to use an analogue transmitter (TX) and receiver (RX). The main limitations here were current draw, size and weight. Both a 2.4 GHz and 5.8 GHz TX/RX combination were tested, both of which operate at 10mW and are thereby within the German legal limits [17]. The hardware was obtained from `http://www.fpvhobby.com/` and each component TX/RX pair costs around €50. The 5.8GHz TX weighs 1.19 g and is therefore more than double the weight of the 0.57 g 2.4 GHz TX. Unfortunately, the heavier 5.8 GHz TX proved to be less prone to transmission interference and was therefore deemed the better choice, despite the larger mass and dimensions. Once again, these are hobby grade components so neither the transmitters or the receivers came with a specification sheet.



(a) Receiver **(b)** USB Cable **(c)** RCA Cable

Figure 3.15: Cables and antenna mount soldered to the 5.8GHz receiver. The RCA cable solders onto the GND and VIDEO points, and the USB cable solders onto the +5V and GND points. This allows the receiver to easily be powered by a USB port and directly interface with the USB video digitiser.

The camera outputs an analogue signal that is directly connected to the video-in through-hole of the transmitter, which is then wirelessly broadcast. See subsection 3.3.3 for details on how to connect and power the various components. The 5.8Ghz transmitter and receiver can be configured to use one of eight different channels. We found that we occasionally had to change frequencies as we seemed to be picking up interference in some environments. However, interference is not always avoidable so we developed a method to detect corrupted images in subsection 6.6.2.

The broadcast radio signal is then picked up by an antenna which is connected to a receiver. The receiver requires 5 V, so we stripped the end of a USB cable (Figure 3.15b) and soldered the ground and power cables to the corresponding points on the transmitter, allowing us to power it from a standard USB port. We also stripped an RCA cable (Figure 3.15c) and soldered the ground and video cables to the corresponding video output and ground points on the transmitter. To digitise the analogue signal video signal, we connected the soldered-on RCA cable to a USB digitiser which exposes itself as a video device at `/dev/videoX` on linux. See section 6.3 for details on how we expose this image over the ROS network and subsection 7.1.4 for a quick evaluation of the lag and range.

Note that one has to be careful when choosing a USB analogue video digitiser as not all are compatible with the linux kernel. The €35 Hauppage USB-Live2 shown in Figure 3.16 worked flawlessly out of the box.

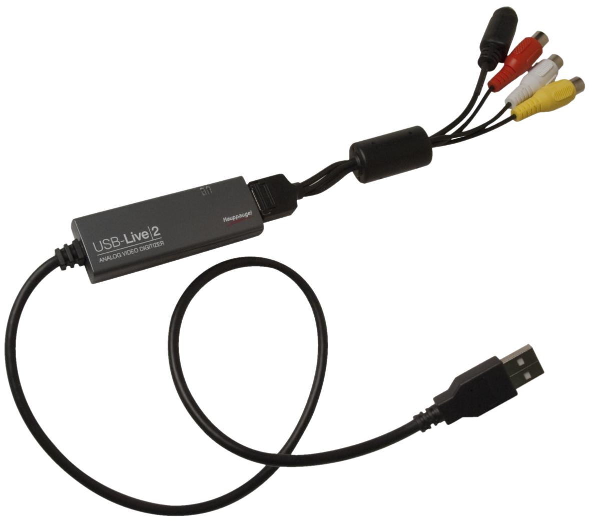


Figure 3.16: The Hauppage USB-Live2 digitises an analogue video signal (yellow RCA connector) and exposes it to the linux kernel as a video device, much like a webcam.

3.3.3 Powering from the Crazyflie

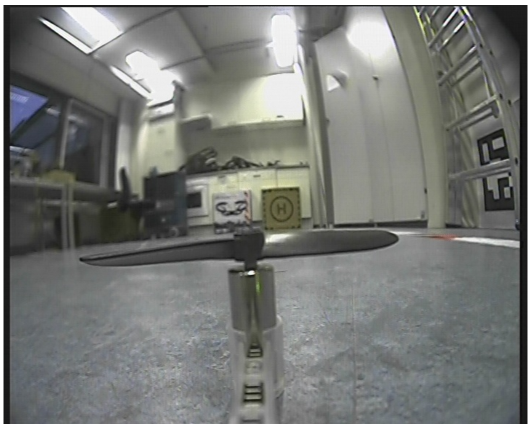
As we do not have enough payload capacity to power the camera system from a separate battery, we power it directly from the Crazyflie. This proved to be a little harder than expected as the motor-induced electrical noise was enough to render the image transmitter useless without some sort of filtering. The Crazyflie exposes `VCOM` (voltage from the battery or USB if connected, after the power management chip), `VCC` (digital power supply), `VCCA` (analoge power supply), `DGND` and `AGND` (digital and analogue ground) via a 10 $\times$2 pin 1.27 mm (0.05") pitch through-holes. `VCC` and `VCCA` bother operate at 2.8 V while `VCOM` will vary between 2.8 V-4.2 V when flying and run at 5 V when attached to USB.

Noise

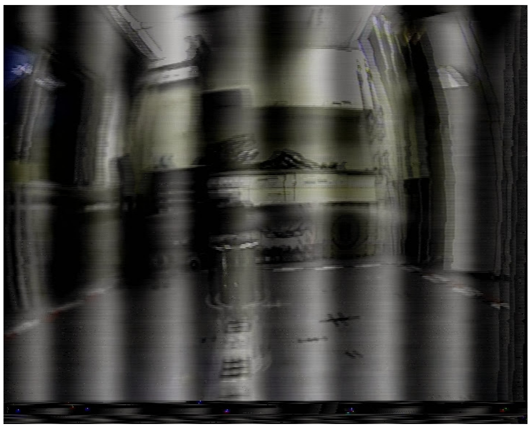
As the electrical components require 3.2 V-5 V to operate, we first attempted to directly drive them from `VCOM` and `DGND` which should provide enough voltage for most of the flight. As `VCOM` comes from the power management chip, it is only active when the Crazyflie is turned on, so one doesn't need to manually disconnect anything between flights. However, this also means that the additional current required by the camera system negatively effects charging time. Initial tests were good, with a clear image being received across rooms. However, once the motors spun up the transmission garbled and the incoming images were completely useless. We concluded the image noise was caused by electrical noise produced by the motors. When a motor runs, a commutator switches the direction of the electricity that flows in the coil windings. Occasionally sparks occur between the motor brushes and commutator at the timing of the commutation. These sparks when coupled with the inductance of the motor coils and motor leads can cause noise on the power lines. One typically solders capacitors across the motor terminals to suppress motor noise. This was not a desired solution as the capacitors would be very exposed and greatly increase the fragility of the flying platform. An additional source of noise is from the nature of using pulse width modulation (PWM) to drive the motors. We experimented with adding various ceramic inductors between `VCOM` and the camera system. While this greatly reduced the image nosie during flight, it was still dominant enough to render the received images useless for pose estimation. Figure 3.17 shows some examples of images received from the Crazyflie under different conditions. Also, occasional voltage drops caused complete frame drop-outs and after two minutes of flight the voltage decreased below the required amount.

Voltage Regulator

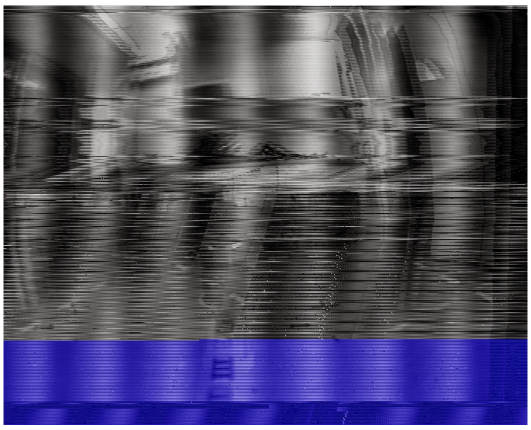
On paper, a Pololu 3.3 V Step-Up/Step-Down voltage regulator (VREG) seemed like exactly what we needed. It would take the 2.8 V-4.2 V from `VCOM` and output a stable 3.3 V the camera system required. These voltage regulators also have the added benefit of smoothing out any noise during the voltage reduction/increase. Unfortunately the 3.3 V VREG did not help the noise issues despite solving the drop-out problems. However, the Pololu S7V7F5 5.0 V VREG worked flawlessly, both in removing noise and maintaining the voltage required to drive the components, even after the inductors were removed. At only $8.2 \times 13.2\,\text{mm}^2$ and being under 0.5 g the voltage regulator was small and light enough to be added to the camera system without any negative side effects. An image of the voltage regulator is shown in Figure 3.18.



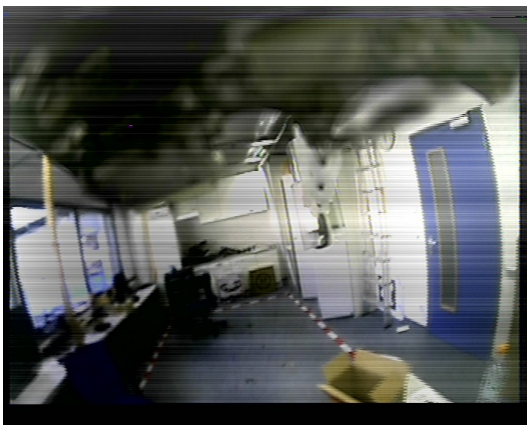
(a) No thrust



(b) 65% thrust



(c) 65% thrust after 30 seconds



(d) 65% thrust using a 22μH inductor

Figure 3.17: The four images above show the images received when the transmitter is directly connected to the Crazyflie's power. Image quality is very sensitive to the thrust, as the transmitter and camera are extremely sensitive to electrical noise created by the running motors. **(a)** Shows image transmission while the motors are not powered. It is noise free and demonstrates the absence of external interference. **(b)** Shows the noise caused by the motors. The flickering black bars change in size and position over time. Additionally, the whole image moves up and down. **(c)** Shows transmission problems caused by the voltage drop after a minute of hovering. Note the pure blue lines at the bottom: the digitiser outputs blue frames if it cannot successfully decode the image. **(d)** Shows the noise improvement over (b) by introducing an inductor between the transmitter and power source on the Crazyflie.

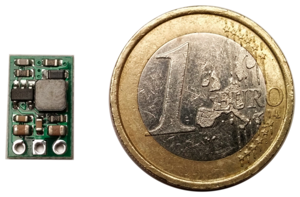


Figure 3.18: The Pololu S7V7F5 5.0 V Step-Up/Step-Down voltage regulator effectively removes all motor induced electrical noise that corrupted the camera images while providing a stable 5.0 V to the camera system.

Full Camera System

We now have all the requirements for the full camera system. We soldered a male 1.27 mm pitch JST Micro Connector to pins 16 ((`DGND`) and 18 (`VCOM`) and a compatible female connector to VIN and GND on the voltage regulator (making sure that the polarity matches up) to facilitate easy connecting and disconnecting of the system. A photo of the camera system is shown in Figure 3.19 and a wiring diagram shown in Figure 3.20.

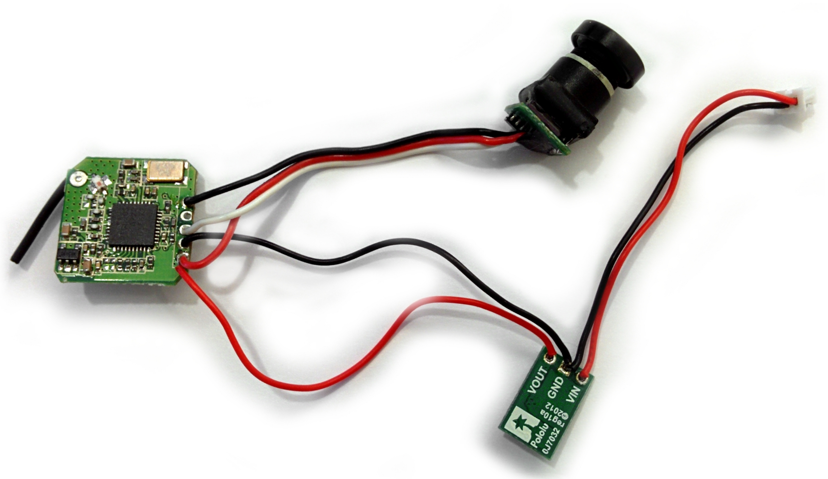


Figure 3.19: Photograph of the video system added to the Crazyflie. The 5.0 V voltage regulator on the bottom is plugged into the Crazyflie and powers the 5.8Ghz transmitter on the left and camera on the top.

3.3.4 Camera Attachment

To rigidly attach the camera to the Crazyflie, we designed a 3 dimension model in Blender [18] which we could 3D print and use to holder the camera in place.

The design needed to fulfil the following criteria:

Rigidly attach the camera without slack The holder must be designed in such a way that the camera is rigidly attached without any slack to the Crazyflie PCB as later we assume the IMU to Camera transform is constant.

Position the camera well The holder should position the camera such that (a) the camera field of view is minimally occluded by the Crazyflie itself (b) the added mass does not shift the center of mass off the Crazyflie and (c) the camera observes the environment in front of the Crazyflie. Therefore, we decided to position the camera centrally under the Crazyflie and align the optical axis with the Crazyflie X axis. The mass of the lens which is offset from the center of mass is compensated by the mass of the voltage regulator and transmitter, which are to be positioned behind the camera. Note that this position causes a small amount of self occlusion as the forward motor

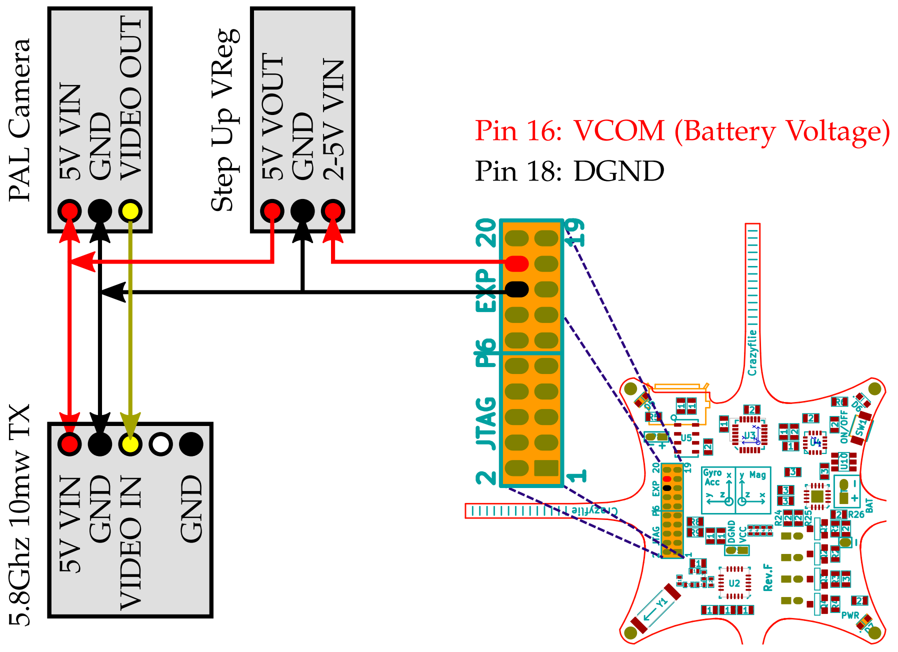
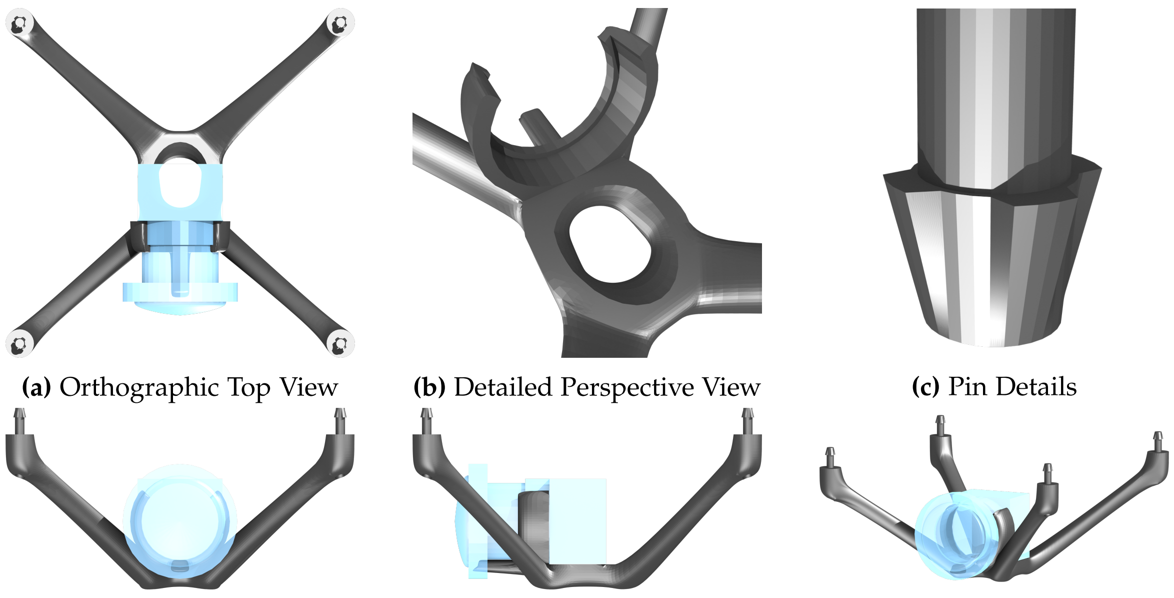


Figure 3.20: Zoom in on the Crazyflie expansion header. Identify pin 16 as DGND and and 18 as VCOM. The step up voltage regulator transforms the 2.8-5V from the Crazyflie to 5V for the camera and transmitter to use. Either GND on the transmitter can be used.



(a) Orthographic Top View **(b)** Detailed Perspective View **(c)** Pin Details

(d) Orthographic Front View **(e)** Orthographic Side View **(f)** Orthographic Top View

Figure 3.21: Renders of the designed camera holder model. The transparent blue object represents the position of where the camera will be fitted. Notice the bored hole in the center to save weight, and the arrow head like shape of the pin to allow the model to click into the Crazyflie's PCB.

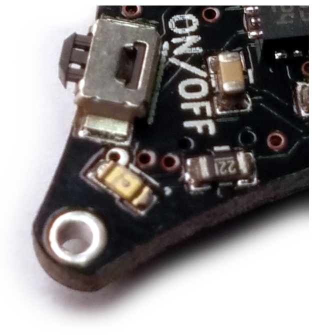
mount is within the field of view (see Figure 3.23b for an example image from the mounted camera). Alternatively, one could rotate the camera around the Crazyflie Z axis by 45° so that it looks between the motors. However, this proved to cause even worse self-occlusion as the wide angle lens captures a large portion of the PCB which is wider than and closer than the motors mounts. Additionally, one would have one pin less to suspend the holder from resulting in decreased rigidity. See Figure 3.23a for a photo of the camera mounted to the Crazyflie.

Protect the camera from crashes Ideally, the holder would protect the camera from crashes, especially as the camera is mounted under the flie. For this reason, the holder was designed to surround the camera at the cost of a small additional weight. This design also serves the purpose of a 'fixed landing gear'. As can be seen in the cross sections of Figure 3.21, the bottom part of the holder is slightly convex which defines four points of contact on the ground, allowing the Crazyflie to land and remain upright.

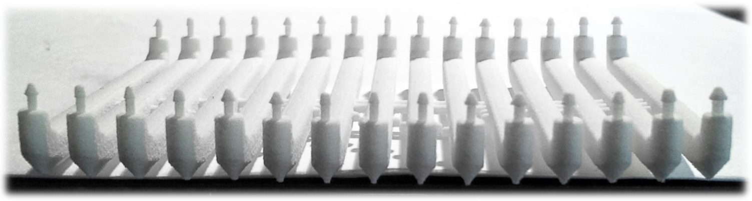
Low Weight Multiple iterations of the model with decreasing material thickness were tested until the model could not withstand a fall from two meters. An average thichkness of roughly 3 mm was determined to give the ideal strength versus weight ratio. To acheive this thickness in the base, a smooth hole was bored out of the center, as can been seen in Figure 3.21b.

Flex rather than snap A nice side effect of using a thinner model is that it flexes under stress, acting as a shock dampener in case of fast vertical descents into the floor. Three different printing technologies and materials were tested. Extruded ABS was too fragile as it would easily break apart between the printed layers. Also, the precision was not enough to make the pins work. Models printed from a stereo-lithographic printer (a laser solidifies a liquid resin from which a mode is pulled out of) had the highest resolution and stiffness. The pins even made a clicking sound as they were inserted into the Crazyflie PCB. Unfortunately, the material would snap rather than flex under high stress which was deemed too fragile for everyday use. From the tested materials, only the models printed by selective laser sintering (SLS) of a nylon based powdered (such as those printed by ShapeWays), were precise enough to function and strong and flexible enough to survive a crash.

Easily attach and detach the holder from the Crazyflie Idealy, one would be able to detach the holder from the Crazyflie without too much effort. Using super glue or other semi-permanent solutions was to be avoided. Fortunately, the Crazyflie has four 1.8 mm deep $\times$ 1 mm wide holes in the corners of the PCB. See Figure 3.22a for a close-up picture. The designed model makes use of these holes by have four pins which can slide through the holes. The pins have an arrowhead like shape which allow them to easily be inserted in one direction and then clicked into place. Finding exactly the right shape was difficult due to the small scale and printing imperfections. In order for the fins to fit through the hole but stick out enough to click into place behind the hole, a printing precision of 0.1 mm is required. We tested many different shapes and sizes (see Figure 3.22b for some prototypes) and opted for the design shown in Figure 3.21c. Inserting the pins can be a little challenging, but once inserted they are firmly in place. Alternatively, the 3d model has holes directly in line with the pin. These indicate drilling locations for a 1 mm wide drill to enable

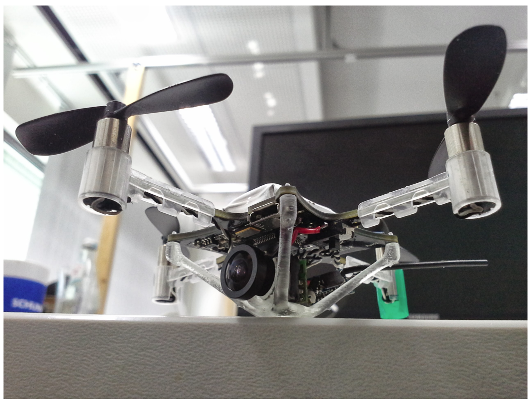


(a) PCB hole

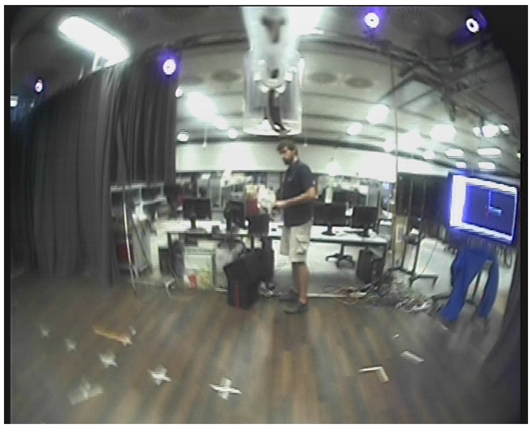


(b) Prototyping 15 different pins with varying arrowhead shapes

Figure 3.22: holepcb caption



(a) Mounted Camera System



(b) Camera's view during hover

Figure 3.23: The camera is rigidly attached to the Crazyflie. **(a)** This model was printed by a Formlabs 3d Stereolithographic printer [19]. **(b)** View from the mounted camera during a hover at 2 m altitude, note the self occlusion.

the use of tiny screws that can also fasten the holder to the PCB. In practise these were not required as the pins were more practicle and worked well enough.

Easily attach and detach the camera from the holder We also wanted a method to securely fix the camera to the holder. This is done by means of a 3/4 ring that clips around the camera housing. See Figure 3.21b for a close up picture. Note, that while this solution holds the camera in place and prevents it from rotating, it does not prevent the camera from sliding away from the ring.

Fix the focal length by not allowing the lens to rotate To lock the camera in place and prevent it from sliding out of the ring holding it in place, we added a protruding pin that pressed against the camera lens. This allows the user to shave the pin to exactly the right length, such that when the lens is tightly screwed into the camera housing it not only keeps the camera in place, but also keeps the lens from accidentally being unfocussed.

4 Pose Estimation

In this chapter we introduce the reader to the notion of pose and attitude estimation and discuss the three developed ways we use to estimate the pose of the Crazyflie in realtime.

VI-SLAM In section 3.3 we devised a way to receive images from a camera mounted to the Crazyflie. In section 4.1 we discuss how one can efficiently use these images to estimate the full 6D pose of the quadrotor by developing a monocular Visual-Inertial Simultaneous Localisation and Mapping (VI-SLAM) framework.

MoCap Pose Estimator Using a professional motion tracking system , one can accurately determine the pose of infra-red markers, which when attached to the Crazyflie allow us to track the it as outlined in subsection 4.2.1.

Kinect based Pose Estimator We developed a pose estimator using the input from a Kinect structured light depth camera. This allows us to run various control experiments at home without relying on the usage of a motion capture studio or a mounted camera. This is discussed in subsection 4.2.2 and implementation details are given in subsection 6.5.2.

In section 4.3 at the end of the chapter we briefly cover the Crazyflie's on-board attitude estimation method, which provides rotation estimates for the VI-SLAM and Kinect Tracker systems.

Pose Estimators can be categorised by how many dimensions they estimate (e.g. just a 3D translation vs. a full 6D pose) and by whether or not the estimation is being done externally or not. For example, a pose estimator using a ceiling-mounted camera could tell the robot where it is, or the robot could use an on-board camera to compute its own 3D motion (also called *egomotion* estimation). Once one has an estimated pose, one can use this to compute the error between a desired pose and the estimated pose, which one can feed into a controller that then actuates the robot so it moves towards the desired pose, thus allowing it to hover or fly via waypoints for example. Control using estimated poses as input is covered in chapter 5.

The pose estimators we implemented are summarised in Table 4.1.

Table 4.1: Pose Estimation Methods

Method	Dimensions	Rate	Type	Details
Monocular VI-SLAM	3T+3R	25Hz	Ego	section 4.1
Attitude Estimation	3R	250Hz	Ego	section 4.3
Motion Capture Studio	3T+3R	200Hz	External	subsection 4.2.1
Kinect	3T	30Hz	External	subsection 4.2.2

4.1 Visual Inertial SLAM

In this section we discuss how one can estimate the motion of an agent using images from a monocular camera attached to it.

After covering the general concept of egomotion estimation, we take a closer look at each of the required building blocks, emphasising where and how an IMU can be used to robustify and speed up the process. These building blocks include some projective geometry, our 2D measurement representation and methods to incrementally construct, update, and locate against a map. Finally we propose an entire pipeline looseley based on [20], with implementation details in section 6.6 and evaluation results in section 7.3.

4.1.1 Introduction

Structure from Motion

Structure from Motion (SfM) refers to the process of reconstructing 3D structural geometry of a previously unknown but static environment from a potentially unordered set of images, while simultaneously estimating the camera poses from which the images were taken. SfM assumes that one can identify points in at least two images that represent projections of the same point in space. This is known as the *correspondence problem* and can be solved using feature matching as described in subsection 4.1.2. Usually the final step of an SfM pipeline is to refine the entire estimated structure and all camera poses in a non-linear optimisation procedure often referred to as *Bundle Adjustment* (see subsection 4.1.11). SfM is not usually assumed to run in real-time and usually favours accuracy over raw speed.

Simultaneous Localisation and Mapping

In the field of mobile robotics, *Simultaneous Localisation and Mapping* (SLAM) is the equivalent of SfM when the measurements are sequential (i.e. temporally ordered, e.g. images from a video sequence). Usually the goal is to estimate the motion (often in real-time) of a mobile robot, while continuously mapping the unknown environment it is situated within. Essentially this is a chicken and egg problem: one requires an unbiased map to localise, but one must accurately localise to build a map. A good two-part tutorial broadly introducing SLAM methods is given in [21, 22].

Sensors Multiple sensor types or combinations can be used such as active optical sensors, tactile sensors, sonar sensors and cameras for example. *Visual SLAM* (VSLAM) is the name used when referring to SLAM systems that use one or mores cameras as their primary sensors [23]. In the remainder of this thesis, we focus on the monocular case - ego motion estimation just using a single camera. This is mainly motivated by the limited payload capability of the Crazyflie quadrotor we would like to attach the camera to (see subsection 7.1.1); the advantage of adding a second camera to the huge bandwidth of information provided by a single camera does not outweight the additional weight required.

History The desire to compute ones pose using visual information alone kicked off in the 1980ies, motivated by the requirement for planetary rovers to have the capability to determine their 6DOF pose in rough and deformable terrain where wheel odometry would fail [24, 25]. Research into monocular VSLAM has exploded since 2005 due to the increasing ubiquity of cameras in mobile devices such as mobile phones and MAVs/UAVs, [26, 27]. Furthermore, since most modern cellphones and MAVs/UAVs contain IMUs, much effort has been put into fusing inertial and visual measurements together. This especially makes sense in the monocular case as depth cannot be perceived and the measured acceleration from an accelerometer can be used to estimate or correct the scale [20, 28, 29, 30]. Unfortunately, the accelerometer we employ (section 3.2.1) is too noisy to be used to estimate scale (see section 7.1.3) reliably. However, we can use a barometer to initialise the scale instead as explained in subsection 6.6.4.

Global Consistency Like SfM, (VI-)SLAM aims to provide a globally consistent map and estimate of the robot path at the cost of complexity. This means that the entire history needs to be stored so we can recognise when we have returned to a previously visited area, often referred to as place recognition or *closing the loop*. We can then integrate the constraints derived from the detection of such a loop closures to reduce drift and improve global consistency. State of the art place recognisers include [31] and [32], both which use the bag-of-words approach to represent and compare images.

To enable the long term operation of a (VI-)SLAM system, we must restrict the size of the bundle adjustment step as the cost increases with the number of poses and landmarks. PTAM [33] for example runs full bundle adjustment in a parallel thread and is therefore limited to desktop sized spaces. Therefore, many approaches select a small subset of past frames to process, either by using sliding window [34], or using spatially distributed keyframes [33, 35] which enables long-term drift free operation. More recently [36] presented a constant-time framework with the accuracy of offline bundle adjustment in long and loopy datasets by employing a double window optimisation scheme where an inner window of point-pose constraints is supported by an outer window of pose-pose constraints. The constraints of both windows are then coupled into a single optimisation framework.

Types SLAM algorithms generally fall into one of two categories which sparsify the problem in different ways: (*a*) those based on sequential filtering techniques which marginalise out past poses and summarise the information gained over time with a probability distribution [37, 38, 39], and (*b*) those based on keyframe methods which retain the optimisation approach of global bundle adjustment [33, 36]. Work has also been done on combining both approaches [40, 36], attempting to aggravate and aleviate eachothers advantages and disadvantages.

Visual Odometry *Visual Odometry* (VO) is a particular case of VSLAM where one is mainly interested in the local consistency - only the recent history is of interest and the rest discarded. This provides performance benefits over SLAM. VO was named in [35] and chosen due to its nature being similar to that of wheel-based odometry: incrementally estimating ones pose from a previously estimated pose. Therefore, a VSLAM system without the optimisation backend can be considered a pure VO system. To help remove the accumulated drift inherent in the incremental nature of VO, one often includes an optimisation backend but employs a sliding window: only keeping and optimising the last n keyframes and their associations and discarding the rest [41]. Note that as opposed to a SLAM system, the history is eventually discarded and loop closures beyond the sliding window will not be detected. A great introduction to VO is provided in the two part tutorial [42, 43].

Direct Methods

While the remainder of this thesis only focusses on the keyframe based approach, it might be important to note a recent tend. Both keyframe based and filter based methods previously mentioned almost exclusively rely on sparse-feature extraction to associate measurements. Alternative *Dense* methods also exist such as those based on dense optical flow [44] or featureless tracking. They operate directly on pixel intensities thereby eliminating the need for expensive feature extraction & matching routines for tracking and mapping.

Recently there have been advances in the monocular case: In 2013 Engel et al presented semi-dense visual odometry algorithm which could run in real time on a single core CPU [45]. The implementation was extended in 2014 with a SLAM back-end resulting in Large Scale Dense SLAM (LSD-SLAM)[46] and even ported to a mobile phone [47] in 2015.

[8] presents a precise semi-direct monocular visual odometry algorithm that also operates directly on pixel intensities, resulting in sub-pixel precision at 55fps on an embedded computer and over 300fps on a laptop. This approach also proves to be surprisingly robust against scenes of repetitive, little, and high-frequency texture.

These algorithms are robust and fast but do require a decent quality camera, meaning that for our purposes they are not suitable due to the slow frame rate, poor quality, noisy nature and transmission losses associated with the camera we will be using (subsection 3.3.1). Therefore, for the rest of this thesis we will concentrate on sparse feature keyframe based methods.

Keyframe Based Method Concepts

For a summery of commonly used terminology regarding these concepts please see section 2.1. The keyframe based SLAM method we implement follows one of the typical paradigms commonly used and can be summarised as follows:

1. Using 2D-2D image correspondences, wait for the current frame to have enough distance from the first frame.

2. Using these correspondences, estimate the relative transformation between the images.
3. Derive local 3D structure by triangulating the correspondences - initialising a map with a point cloud of landmarks, and keyframes consisting of the first and current frames.
4. For the following incoming images, use 2D-3D image correspondences to derive incremental relative displacements with respect to the landmarks, associated via the last keyframe.
5. If required, triangulate new landmarks and add new keyframes to expand the map.
6. Maintain this map of previous landmarks and key frames we can find correspondences to and optimise over.

To execute these steps we need to implement the following methods, some of which can be aided by using IMU measurements.

Inertial Priors Measurements from the IMU can be used to help speed up and robustify many of the procedures below. Often, this includes reasoning over relative rotations priors between two camera frames with associated IMU measurements and rotating bearing vectors with these priors. How this is done is shown in subsection 4.1.7

Obtaining 2D-2D and 2D-3D correspondences We need a way to find common points between two images and between an image and landmarks. To do this, we detect features, describe their appearance and match appearances. Landmarks have associated observations and can be matched via the descriptor in the frame that saw it. Obtaining image correspondences and using IMU measurements to robustify and speed up the procedure is covered in subsection 4.1.2.

2D-2D relative pose estimation Given 2D-2D correspondences between two frames, we need to estimate the relative transformation between them. Note that we cannot estimate scale, so we have no way of knowing the magnitude of the baseline from the images alone. Relative pose estimation as well as how to incorporate IMU measurements into the process are covered in section 4.1.10. Note we can roughly estimate the baseline using the barometer as described in subsection 6.6.4.

Estimating disparity In order to triangulate landmarks from point correspondences, the baseline between the camera poses must be large enough. Therefore, the initialisation procedure must waiting until the the current and initial views exhibit enough disparity. Furthermore, the disparity can be used to trigger new keyframes. We require the IMU for this procedure. Estimating the disparity is covered in subsection 4.1.9.

Projecting 2D measurements into the world In order to triangulate points, we need to know how our 2D image measurements relate to the 3D world. We convert our 2D measurements on the imaging plane into bearing vectors. How to do this as well as how to compute errors between them is covered in subsection 4.1.6.

Triangulation Once we have the relative pose between two frames, we can triangulate a point cloud of landmarks between them. This is covered in subsection 4.1.8.

2D-3D absolute pose estimation Once we have initialised landmarks, we can determine our pose relative to them. In this way, the visual scale factor is implicitly propagated between pose estimates. This is covered in section 4.1.10. The IMU can be used to provide rotation priors which can help speed up the process.

Dealing with noisy and incorrect measurements We will have to deal with noisy measurements and outliers. This is covered in subsection 4.1.3.

Local Optimisation Everytime we triangulate new landmarks, we add them to the map. They have associated measurements from camera frames. It is not feasible to store every frame, measurement and association. Therefore we only retain a subset of the frames, called keyframes. We can then optimise the map using only the local keyframes and their observed landmarks, keeping the complexity constant. This is mentioned in subsection 4.1.11.

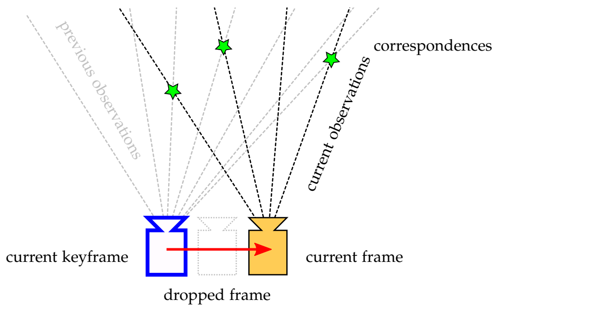
A visual summary of the steps is given in Figure 4.1 and a flow chart of the implemented solution shown in Figure 4.2.

4.1.2 2D Image Features

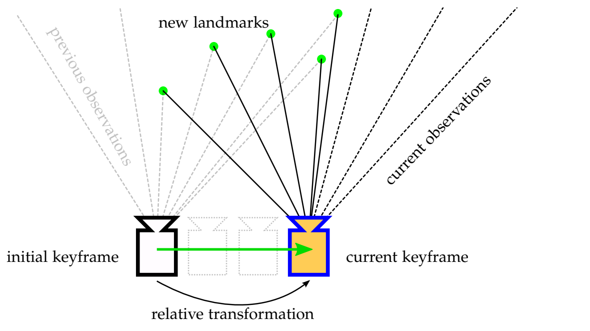
The keyframe based methods above rely on image point correspondences. That is, two projections of the same 3D point projected into two views need to be associated. Two main approaches exist: (*a*) identify points in the first image and *track* them in the following images using local search techniques, or (*b*) independently *detect* interesting points in each image and *match* these based on a similarity measure between a description of their local neighbourhoods. In this thesis we focus on the latter approach as it handles large motion between viewpoints better, a requirement due to the noise and frequent transmission loss of our quadrotor mounted wireless video system. Furthermore it allows arbitrary image pairs to be matched which simplifies the design of a recovery system in case features of the current image cannot be tracked or matched against the previous keyframe. This approach is usually a three part process summarised in Figure 4.3 and below:

Keypoint Detection As it is computationally infeasible to compare every pixel of one image with every pixel of the other, one aims to locate interesting points (often called *features*) of an image that are associated with distinctive, repeatably identifiable 3D entities. These features are subsets of the image domain, and usually in the form of points, continuous curves, lines, blobs or connected regions. In this thesis we only deal of the point type, usually referred to as *keypoints*. The process of finding such keypoints is called keypoint detection and is discussed in section 4.1.2.

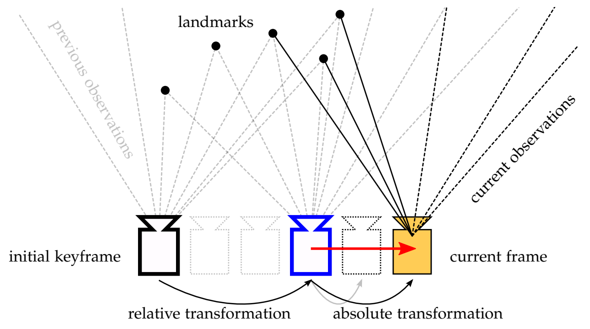
Descriptor Extraction Once one has identified a set of distinctive keypoints that can be reliably localised, one needs to compute an abstraction of the image information that describe the local appearance around the keypoint. Ideally this abstraction of the 2D appearance would be identical for all views of the 3D point, even under the presence of noise. These local neighbourhood representations are called descriptors and computing them is called descriptor extraction. They are discussed in section 4.1.2 and an IMU can be used to achieve rotation invariance.



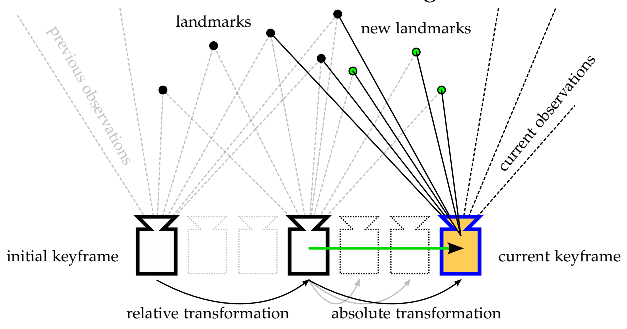
(a) Pre-Initialisation: The first frame is considered a keyframe and all successive images are matched against it. From the pairs of corresponding measurements one can estimate the angular disparity. If the disparity is too small, we drop the current frame.



(b) Post-Initialisation: The current frame displays sufficient angular disparity, so the corresponding 2D-2D bearings are used to estimate the relative transform and triangulate landmarks. The current frame is added to the map as a keyframe, which the next frames will match against.



(c) New images match against the most current keyframe, obtaining associations to landmarks via the keyframe. Using these 2D-3D correspondances an absolute pose is computed. If the angular disparity is insufficient, the frame is dropped.



(d) With sufficient angular disparity a new keyframe is added to the map as well as additionally triangulated points. Global optimisation can then be done over all correspondences, e.g. some landmarks can be seen from more than two keyframes.

Figure 4.1: Overview of a typical feature based keyframe SLAM algorithm

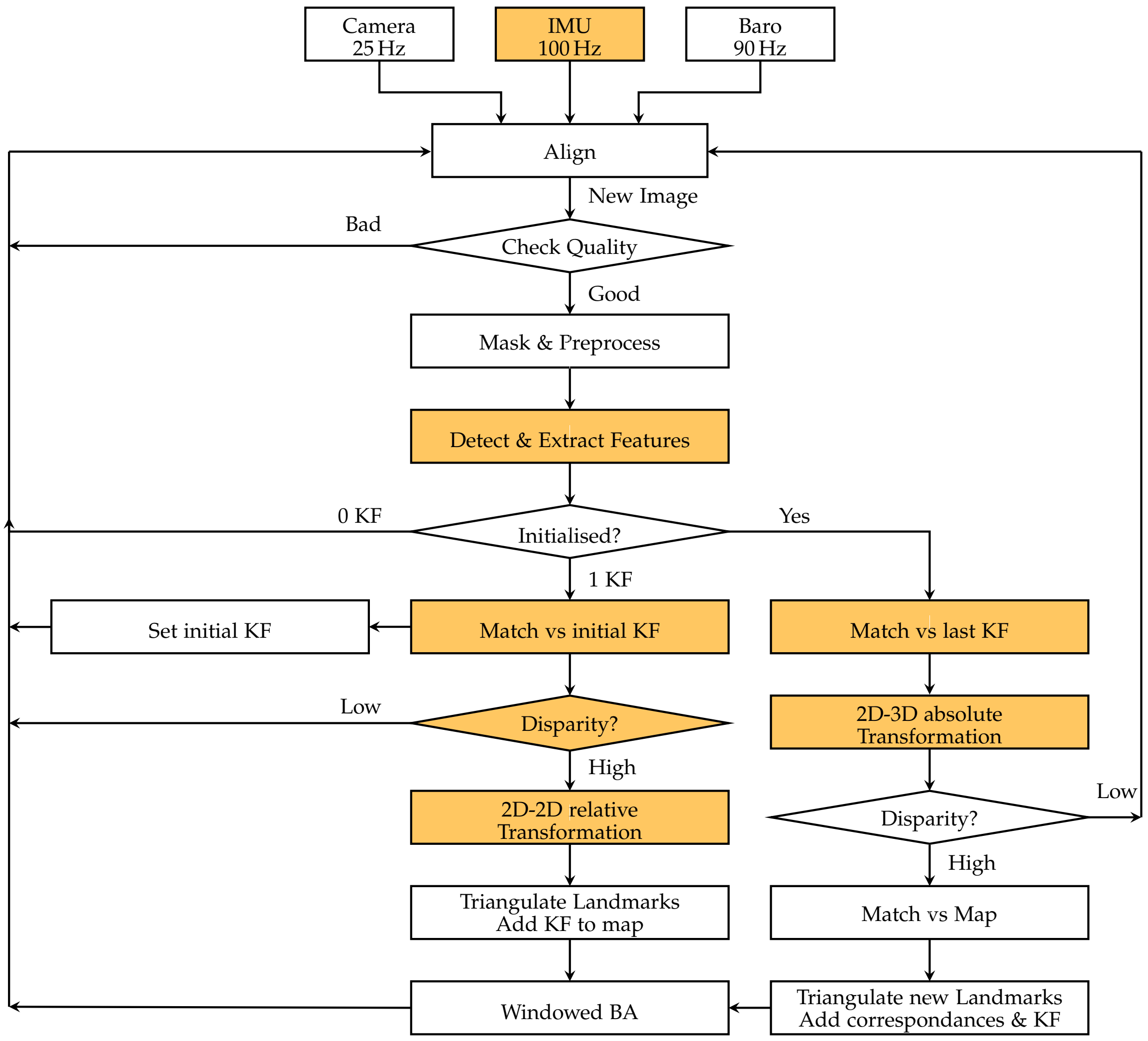


Figure 4.2: Diagram depicting the control flow of the implemented VI-SLAM algorithm. Coloured boxes indicate that an IMU is used to aid the corresponding procedure.

Descriptor Matching Once one has two sets of keypoints with associated descriptors from two different views, one can attempt to find the corresponding points between images by comparing the descriptors. Similar descriptors should yield small matching distances while incorrect descriptor matches should have large ones. How one can compare descriptor sets is discussed in section 4.1.2. Once again, an IMU can be used to speed up the process by pruning potential match combinations.

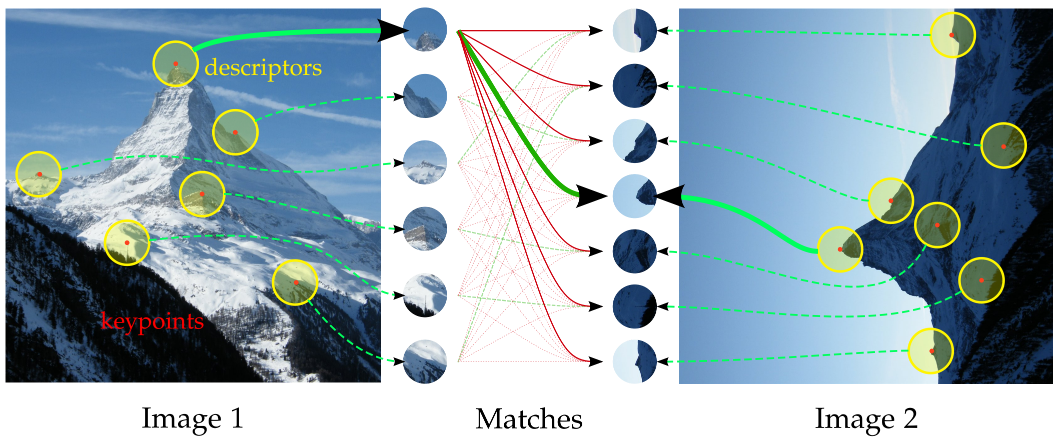


Figure 4.3: Finding corresponding points between two images. First, keyoints are detected (red); then descriptors are extracted (yellow); finally descriptors are matched, of which some yield positive matches. Ideally the descriptors encode the local information with invariance to rotation, scale, affine projection, lighting and noise.

A vast amout of research has been conducted in this field and a plethora of keypoint detection and descriptor extraction algorithms exist. It is beyond the scope of this thesis to compare, evaluate or give a full explanation of how each one works. We therefore restrict ourselves to briefly mentioning some of the more well known algorithms and only describing and justifing the algorithms we employed. We refer the interested reader to [48, 49, 50] for literature that evaluates various detection and extraction methods. Furthermore, OpenCV[51] readily offers implementations of many detectors and extractors.

Keypoint Detection

The goal of the keypoint detection step is to localise distinctive points in the image that can be repeatably detected despite varring conditions. For example, corner points (an intersection of at least two edges) or blobs (an image region differing from its immediate surroundings in colour/texture/intensity) are often sought as their position in the image can be accurately determined. Well known blob detectors include

- SURF (Speeded Up Robust Features) [52]
- SIFT (Scale Invariant Feature Transform) [53]
- AKAZE (Accelerated-KAZE). [54]

while corner detectors include

- Harris [55]
- SUSAN (Smallest univalue segment assimilating nucleus)[56]
- FAST (Features from Accelerated Segment Test) [57]
- AGAST (Adaptive and Generic Accelerated Segment Test) [58]
- Shi-Tomasi [59]
- ORB (Oriented Fast and Rotated Brief) [60]

They all have different pros and cons regarding the following characteristics:

Repeatability The same keypoints should be redetected in the previous/next image.

Distinctiveness Detected keypoints should also have a wide range of appearances so that they can be distinguished from one another allowing them to be accurately matched.

Localisation Accuracy Ideally keypoints are accurately detected with respect to scale and position. Some detectors such as SIFT[53] even achieve *sub-pixel accuracy* by fitting a 3D quadratic function to the local sample points to determine the interpolated location. Note that we require a contrast rich for this. or example, if we consider a point in a contrast uniform region, we are not able to determine its exact position, as we cannot distinguish it point from its neighbours.

Robustness Detections should be robust to various levels of noise, blur and even compression artefacts.

Computational Cost In order to be feasible for real-time operation, the detection step must be executed very quickly. Corner based approaches (FAST, Harris, etc) usually outperform blob based approaches (SURF, SIFT, etc) in terms of detection time, but often at a cost of distinctiveness.

Scale Invariance Detections should be invariant to scale. This means that two views of the same scene, where one view is zoomed in, should yield the same detections in within the overlapping observable area. This is often achieved by applying the detector at a lower-scale and upper-scale version of the input image

Illumination Invariance Detections should also not be affected by various photometric changes.

Rotational Invariance Rotation (i.e. rotating the camera round the optical axis) should not affect the detections.

Invariance to Perspective Distortion Projections of objects on to the image plane may look different between different camera poses due to perspective distortion. Ideally this would not change the detections. For simplicity, one often approximates the perspective distortion as an affine one.

Most detectors work in a two phases, with an optional third:

1. First, a detector applies a feature response function to every pixel, which yields a detection score. This function could be the Harris corner response function or the

Laplacian of Gaussian (LoG) approximation used in SURF.

2. Finally, it applies non-maximal-suppression (NMS) over the feature response. This effectively removes multiple responses that belong to the same detection, leaving only one maximal response left. For example, imagine a detection of a large blob. A group of pixels at the center of this blob will have a high blob measure, yet we are only interested in a single response to represent the detection.
3. Optionally, one can threshold the maximal responses to further eliminate current responses. This threshold parameter effectively allows us to control roughly how many responses we obtain. Furthermore, one could sort the detections by their responses and only keep the top N if fewer are required.

FAST Detector For our VI-SLAM implementation we decided to use the FAST [57] detector as it is computationally extremely inexpensive and yields many keypoints.

Remaining true to it's acronym, the FAST corner detector was designed to be as computationally efficient as possible while remaining repeatable [57, 61]. It is significantly faster than other corner detection methods, such as a $17\times$ speed up over the Harris detector and $6\times$ speed up compared to the SUSAN detector [61]. Example detections are shown in Figure 4.4.

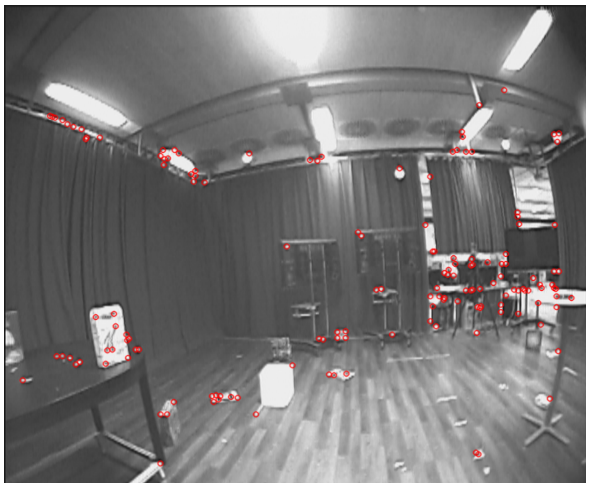


Figure 4.4: The top 150 FAST detections on an image from the quadrotor.

The FAST detector builds upon an intuitive idea: a pixel p can be considered a corner if a sufficiently large continuous arc of pixels around p are either all significantly brighter or darker than p itself. In practise, this means arranging 16 pixels on a Bresenham circle of radius 3 around p. Pixel p is deemed to be a corner if the intensities of at least 12 contiguous pixels of the 16 are all above or all below the intensity of p by some threshold, t. See Figure 4.5 for an illustration. The condition can be optimised to reject candidate pixels as early as possible. For example, by examining the top/bottom and then left/right pixels of the circle, one could potentially dismiss the pixel since a feature can only exist if three of these test points are all above or below the intensity of p by threshold t. This idea was later extended in [57] by using an offline machine learning approach to learn a decision tree that decides if each pixel in an is a corner or not, using as little comparisons as possible. The resulting decision tree can then be converted into a huge if-then-else construct and used to classify pixels as corners or not at great speed.

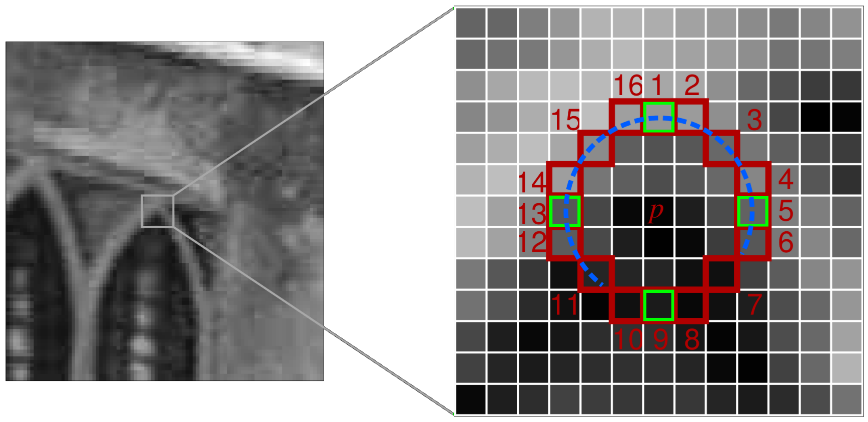


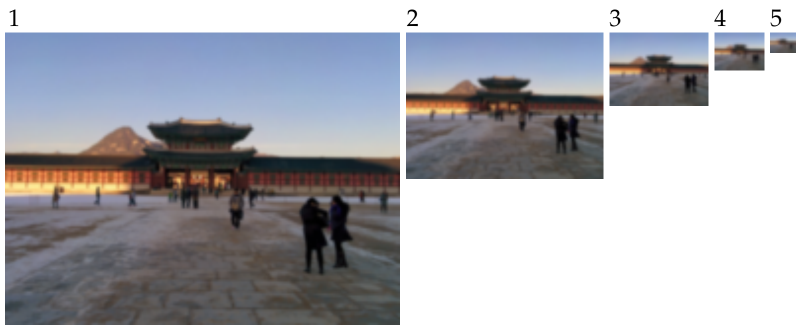
Figure 4.5: The FAST keypoint patch at pixel p: The red squares indicate the pixels used, the blue line passes through 12 contiguous pixels which are brighter than p plus a given threshold. The initial FAST detector first tested the green squares for rapid rejection. Image modified from [61]

.

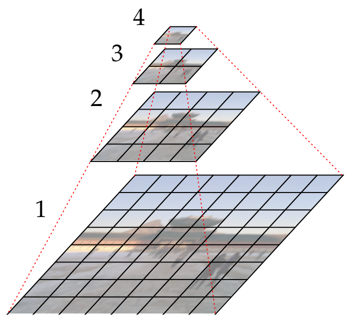
As each pixel is classified as a corner or not a corner, there is no corner response function and therefore non-maximal suppression cannot directly be applied. Therefore, [57] propose to define the corner strength to be the maximum threshold t for which a point is still detected as a corner and apply non-maximal suppression over the corner strength with a 3×3 mask. In practise this leads to total detection time in the 5 ms range over VGA image to detect 500-1000 corners.

Scale Invariant Detection Inherently, corner detectors such as FAST or Harris only detect corners of a certain size. In order to identify keypoints at different sizes, a multi-scale approach is used: the input image is blurred and sub-sampled, increasing the relative patch to image size ratio. Running the corner detector on each sub-sampled image allows 'larger' corners to be detected.

Constructing a Gaussian Pyramid is a commonly used multiscale approach which dictates how one downsizes (zooms out from) the input image. They are a collection of images that are successively down-sampled until some desired stopping criteria is reached - one often down-samples a fixed number of times. One can imagine each downsampled image as a 'layer' in a pyramid (see Figure 4.6) where the higher the layer, the smaller the image size. To produce the next layer, one blurs the current layer by convolving it with a Gaussian kernel and then sub-samples it by removing every even row and column, effectively halving the image dimensions (quartering the area). The Gaussian blurring is vital to avoid aliasing artefacts.



(a) Blurred and sub-sampled images



(b) Image Pyramid Visualisation

Figure 4.6: Multi Scale Gaussian Pyramid used for scale invariant detection. Running a corner detector on each level allows for corners of different sizes to be detected.

Grid Adapted Detection The FAST detector has its share of disadvantages. As with any corner detector, the FAST detector relies on textured surfaces or cluttered scenes. Weaker textures and blank walls or floors often have no visible features, making detection in these areas less repeatable and accurate.

The distribution of the features in the image is very important for VO/VSLAM [62]. Both the image coverage and the number of features matter. Ideally one would have many keypoints equally distributed over the image. If one has an area with strong texture and uses a corner response function to determine which keypoints to keep, one might end up only keeping the ones detected on the high texture area and dropping the rest. In order to avoid this behaviour, one can partition the image into a grid, and apply the detector to each sub-image and tune the detection thresholds independently until a minimum number of features are found in each cell [62]. An illustration showing an example is given in Figure 4.7 An implementation of this functionality is available in the OpenCV Library.

Descriptor Extraction

Once we have identified keypoints, we need a compact way to describe the region around them in a representation that allows us to efficiently reason over the similarity between them. These abstractions are called *descriptors* and comparing them to one another is called *descriptor matching*. Intuitively, detected keypoints and their descriptors provide a sparse representation of images by capturing the essence of their interesting structure. Once again, much research was conducted in this field with the goal of designing highly discriminative yet efficient descriptors. Some of the more popular approaches are listed in Table 4.2 and are available in the OpenCV Library [51].

An obvious and often effective way to construct a descriptor is simply to use its appearance directly: directly use the intensity of the pixels within a small window around the keypoint. To compare the patches, one can then simply use common error metrics such as the sum of squared differences (SSDs), the sum of absolute differences (SADs), or for a more illumination invariant approach use the normalised cross correlation (NCC). One

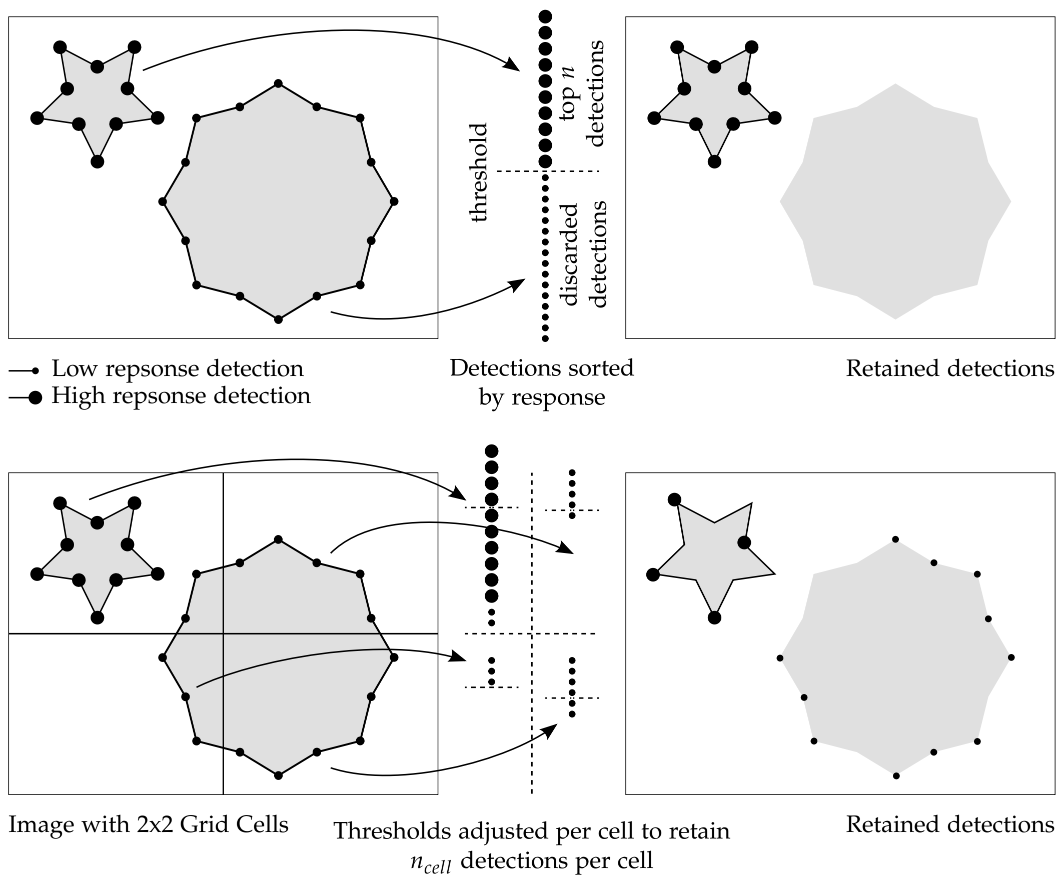

top n detections
threshold
discarded detections
—• Low repsonse detection
—● High repsonse detection
Detections sorted by response
Retained detections
Image with 2x2 Grid Cells
Thresholds adjusted per cell to retain n_{cell} detections per cell
Retained detections

Figure 4.7: Setting the response threshold to keep the top n detections can result in very uneven coverage. One way to improve this situation is by adjusting the threshold of sub-images independently, such that each sub-image has enough features.

Table 4.2: Popular Descriptor Extractors

Algorithm	Type	Rotation Inv.
SIFT (Scale Invariant Feature Transform) [53]	Float	Yes
SURF (Speeded Up Robust Features) [52]	Float	Yes
BRIEF (Binary Robust Independent Elementary Features) [63]	Binary	No
BRISK (Binary Robust Invariant Scalable Keypoints) [64]	Binary	Yes
ORB (Oriented Fast and Rotated Brief) [60]	Binary	Yes
AKAZE (Accelerated-KAZE) [54]	Binary	Yes
FREAK (Fast Retina Keypoint) [65]	Binary	Yes

can further increase the robustness of patch matching with regard to affine projection by pre-warping/rotating the patches if the camera motion has been estimated. However, local appearance based matching only works well when the images are very similar, i.e. taken from neighbouring positions. They are not invariant to scale, rotation or larger viewpoint changes.

Many more advanced techniques have since been constructed, such as the SIFT extractor which basically encodes a region into histograms of local gradient rotations. SURF uses integral images and approximations to speed up a similar process. Both SIFT [53] and SURF [52] store the descriptor as a 128 element float vector and matching scores are computed by simply taking the euclidean distance between them.

Binary descriptors represent descriptors as binary strings and have recently become popular because of the speed at which they can extract and match descriptors. For the first time, it was feasible to do real time extraction on mobile devices.

All the various descriptors are various pros and cons with regard to the same desirable properties of the keypoint detectors. It is not in the scope of this thesis to compare and contrast existing solutions, here we will soley focus on the descriptor we chose to use in our implementation: the BRIEF [63] descriptor.

BRIEF Extractor Since 2010, many binary descriptors have been proposed that attempt to reach the same levels of repeatability and accuracy as the SIFT extractor but at a fraction of the computational cost.

The BRIEF descriptor was one of the earlier ones, which was designed with computational effort and memory efficiency in mind. To shorten descriptors, approaches such as Principal Component Analysis (PCA)[66] or Linear Discriminant Embedding (LDA)[67] were proposed to apply dimensionality reduction to existing descriptors. Furthermore, [68] proposed to use hash functions to reduce SIFT descriptors to binary strings, resulting in a drastic size reduction and faster matching process. While these approaches worked and reduced the descriptor dimensionality, they did so at the cost of computational resources, as the original descriptor required additional post processing steps. As the acronym might suggest, the idea behind BRIEF was to short cut the extraction and post processing steps to a single, direct reduced binary string.

BRIEF simply uses pairwise intensity comparisons sampled within patches around keypoints, which form the bits of the descriptors. As the intensity of single pixels are used, the binary test is very noise-sensitive. Therefore, one pre-smooths the image to reduce the sensitivity and increase the stability and repeatability of the descriptors. The effect of the spatial arrangement and number of binary tests were analysed in [63], which showed that tests sampled from an isotropic Gaussian distribution proved to be most discriminative even with only 256 tests (resulting in 32 byte descriptors). Note that the testing sequence and spatial arrangement are then fixed after the first extraction to yield compatible descriptors. See Figure 4.8 for an example of a testing spatial arrangement for a 128 bit BRIEF descriptor. Furthermore, the resulting descriptors are not designed to be rotation or scale invariant, but one can achieve scale invariance by can running the extractor over patches on multiple image pyramid levels (see Figure 4.6).

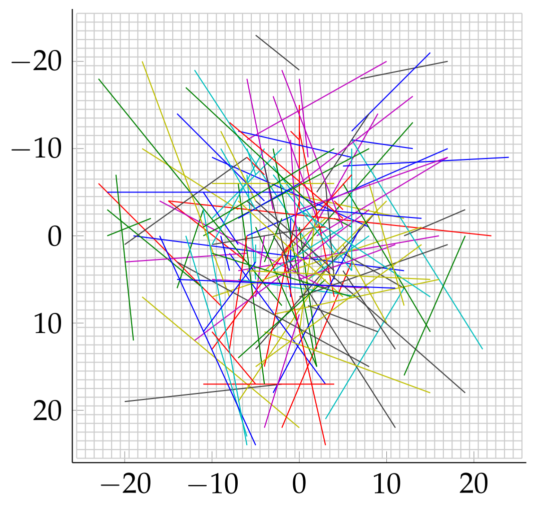


Figure 4.8: The 16 Byte BRIEF Descriptor: 128 Binary tests within a 50 × 50 patch sampled according to an isotropic Gaussian distribution. Each line represents a binary test that yields true if the pixel intensity of one predetermined end of a line is larger than the other.

Due to its simplicity and binary nature, BRIEF descriptors can be extracted with very little computational effort, and therefore work well in tandem with the large amount of keypoints the multi-scale FAST detector returns. BRIEF yeilded extraction times with a $35 - 41\times$ speed up compared to SURF without compromising recognition rates [63] where rotation invariance was not required. As we do not need rotation invariance (see the next paragraph), the BRIEF extractor became the clear candidate for our requirements.

Using IMU Rotation If one has a way to estimate the camera rotation around the principle axis, one can use it to rotate the descriptors back, thereby elimination the need for rotation invariant descriptors. As the rotation estimation steps of descriptor extractors are usually relatively expensive, one can acheive noticable speed gains.

As our quadrotor has an IMU, we can simply use the roll estimate of the quadrotor to unrotate our descriptors in the image plane. For example, one could omit the rotation estimation step of the SURF extractor and directly set the rotation using the IMU roll estimate. See Figure 4.9 for an illustration.

As we use BRIEF descriptors, this amounts to rotating the spatial configuration of the binary tests. BRIEF descriptors were shown to be rotation invarient up to ±10 degrees[63]. Therefore, we can discretise the rotations into 10 degree steps.

Descriptor Matching

Descriptor matching is the process of assigning a score to a pair of descriptors, where a high score corresponds to a high similarity between them. The goal is to determine which query descriptors from one image correspond to the same descriptors in another image.

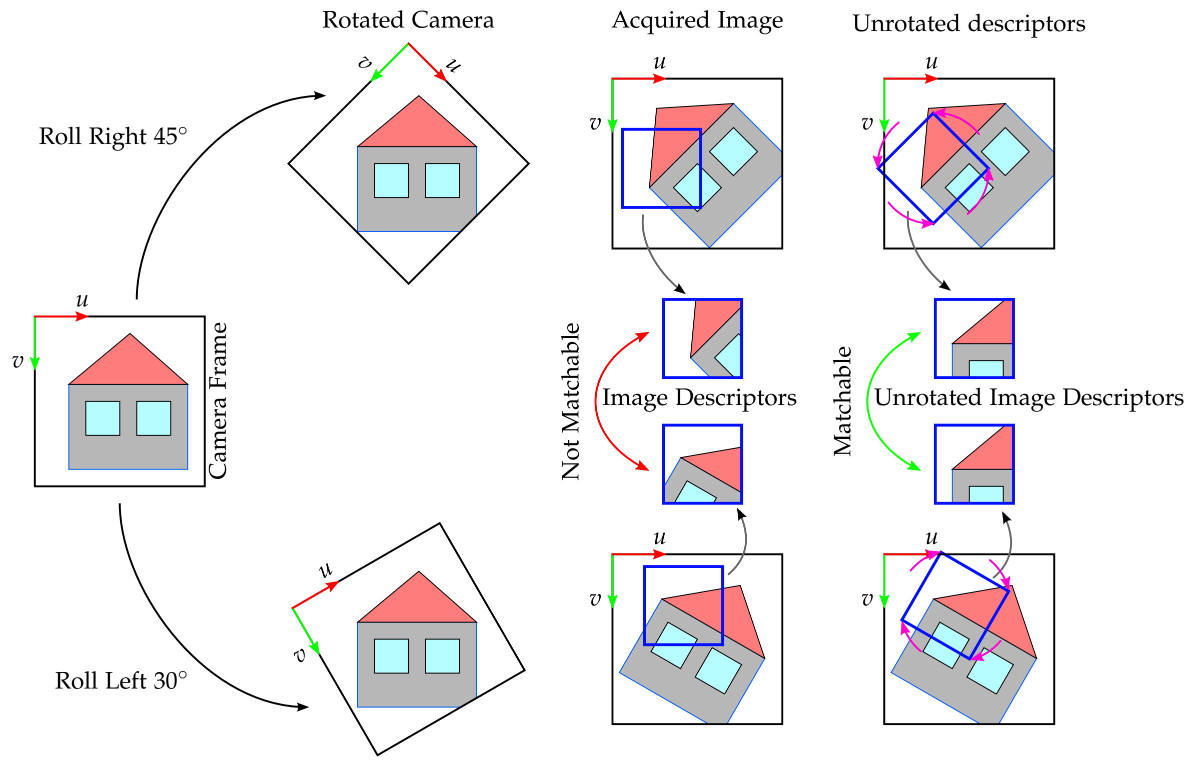


Figure 4.9: We can unrotate the BRIEF binary pattern using roll estimates from the IMU to achieve rotation invariance. Here two images of the same object but taken with different roll are shown. Unrotating the descriptors allows us to successfully match the descriptors.

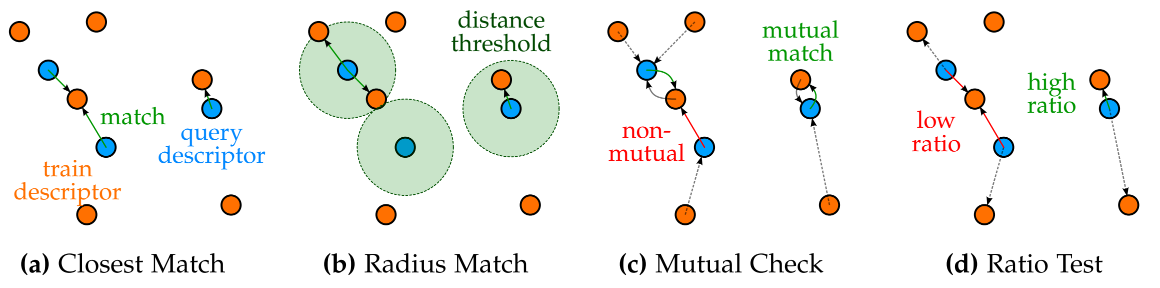


Figure 4.10: Visualisation of different matching and match filtering methods.

Comparing Descriptors For most float descriptors, one simply uses the Euclidean distance between the descriptor vectors, for appearance based descriptors SSD, SAD or NCC is used, and for binary string descriptors, the Hamming distance is used. The Hamming distance is the number of positions at which the corresponding values of two binary strings are different, which can be computed as the population count (the number of bits set to 1) of two binary strings XOR'd (exclusive or) with one another. For example, 1011 0101 XOR 1111 0001 = 0100 0100 which gives a population count of 2. Most modern CPUs that include the SSE4.2 instruction set also implement the `POPCNT`, meaning that one can compare binary descriptors extremely efficiently.

Finding Matches The simplist way to obtain matching features is the brute force strategy: compare all feature descriptors in the first image to all other descriptors of the second image. Then for each descriptor of the first image, either chose

- the N most similar descriptor(s),
- all descriptors within a specified matching distance (radius matching),
- a combination of the above,

from the second image. Note that depending on the application, a k-d tree implementing *radius search* or *k nearest neighbours search* could greatly speed up the matching lookups, where one would use Locality-sensitive hashing (LSH) for indexing. However, for our purposes building the k-d tree was computationally more expensive than brute force matching.

Filtering Matches As matching often generates many outliers, one usually performs a post processing step that aims to remove false positives. One can:

- increase the matching score threshold if more than enough matches have been found,
- perform the *mututal consistency check*,
- perform the *Lowe distance ratio test* [53],
- a combination of the above.

The mutual consistency check retains all matches that are bidirectional: only matching pairs of corresponding descriptors that mutually have each other as the closest match are kept. See Figure 4.10c for an illustration.

The Lowe ratio test requires the best two candidate matches per query descriptor based on the distance between them. By comparing the ratio between the distance of the best and second best match, we can remove ambiguous ones: If the ratio is high, we can safely accept the first match as it is unambiguously the best, and if the ratio is low, the possibility of selecting the wrong match is higher and it is better to reject both matches. See Figure 4.10d for an illustration.

Predictive Matching Using IMU Rotation Priors Exhaustive brute force matching has the disadvantage that is it quadratic in the number of descriptors. If one has additional information that could help predict where corresponding matches could be, one could avoid comparing all descriptors from one set with another, thereby reducing the number of comparisons and reducing the probability of outliers. There are two cases:

Initialisation When we first initialise our map, we have no relative transformation information between the initial keyframe and the current frame. However, we do have the relative rotation between them and can therefore rotate the bearing vectors associated to the keypoints/descriptors into the same rotational frame (covered in section 4.1.7). This allows us to efficiently compare and threshold potential matching pairs based on the efficient error calculation of the angles between. Therefore, we only match pairs where the associated bearing vectors have an angular error between a minimum and maximum threshold.

Landmark Matching In the general case we want to match the descriptors of incoming frames to the descriptors associated with landmarks. In this case we can make use of the estimated full 6DOF transform of the latest pose estimate ${}^{\mathcal{W}}_{\mathcal{K}_i}\mathbf{T} \cdot {}^{\mathcal{K}_i}_{\mathcal{C}_{t-1}}\mathbf{T}$ as well as the relative rotation ${}^{\mathcal{C}_{t-1}}_{\mathcal{C}_t}\mathbf{R}$ between the previous frame $\mathcal{C}_{t-1}$ and the current frame $\mathcal{C}_t$.

Landmark's descriptors are the descriptors from the associated keypoints/bearings of keyframes that observed it. Therefore, we express all the landmarks of the matching candidate descriptors of the closest keyframe in the frame of the previously estimated pose, and finally rotate them into the current frame using the relative rotation priors between the previous frame and the current frame. See Figure 4.11 for an illustration. As before, this allows us to remove matching candidates where the angle between the corresponding bearing vectors is beyond a certain threshold.

Adding a new Keyframe When promoting the current frame to a new keyframe we assume we know it's 6D pose. Therefore, we do the same as above but use all landmarks within the field of view and omit the additional relative rotation step.

In practise this massively reduces the number of required matches (usually by over 90% with liberal thresholds) as well as removing most outliers. The remaining outliers are removed with RANSAC, a roboust model fitting algorithm explained in the next section.

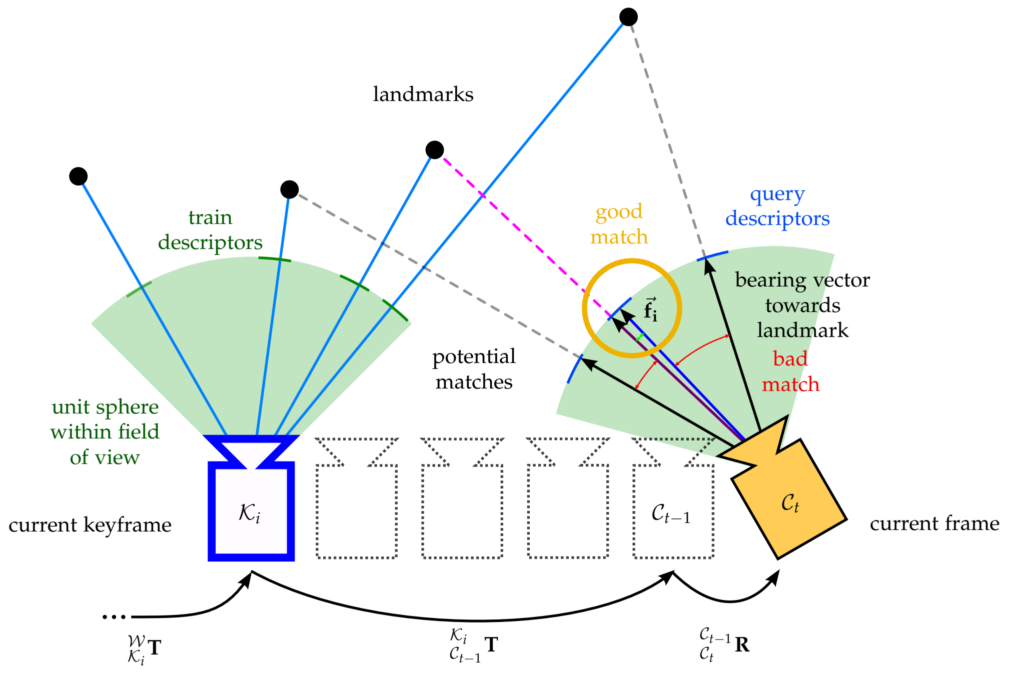

landmarks
train descriptors
unit sphere within field of view
current keyframe
$\mathcal{K}_i$
potential matches
good match
$\vec{\mathbf{f}}_\mathbf{i}$
query descriptors
bearing vector towards landmark
bad match
$\mathcal{C}_{t-1}$
$\mathcal{C}_t$
current frame
$^{\mathcal{W}}_{\mathcal{K}_i}\mathbf{T}$
$^{\mathcal{K}_i}_{\mathcal{C}_{t-1}}\mathbf{T}$
$^{\mathcal{C}_{t-1}}_{\mathcal{C}_t}\mathbf{R}$

Figure 4.11: Using the previous pose estimate and relative rotation between it and the current frame, we filter potential matches (corresponding to the descriptor of measurement $\vec{\mathbf{f}}_\mathbf{i}$) before matching, by looking at the angular error between *(a)* the bearing vectors corresponding to the descriptor and *(b)* the bearing vector formed by expressing the landmark in the rotated frame.

4.1.3 Robust Model Fitting

In real world scenarios, one must deal with data contaminated by outliers. For example, feature points may be incorrectly associated by the feature matcher. False matches could occur for a number of reasons, including (*a*) occlusions, (*b*) different but similar appearing landmarks, (*c*) blur, and (*d*) any changes in illumination or view point for which the mathematical model of the keypoints and descriptors do not correctly account for. To accurately fit a model, such as camera motion, the outliers must be removed before the data is used.

RANSAC

RANSAC (RANdom SAmple Consensus) [69] is a standard method to process such contaminated datasets by classifying data points as either inliers or outliers. It is a non-deterministic, iterative procedure that generates a hypothesis by randomly choosing a minimum number of data points required to estimate a model. The generated hypothesis is then verified on the remaining subset of the data and is deemed to be the best solution thus far it has a higher consensus than the current best estimate.

This hypothesize and test procedure allows us to identify the inlier subset, after which a least-squares result can be obtained by generating a model from or applying a non-linear optimization scheme over all inliers. The algorithm is illustrated and summarised in Figure 4.12.

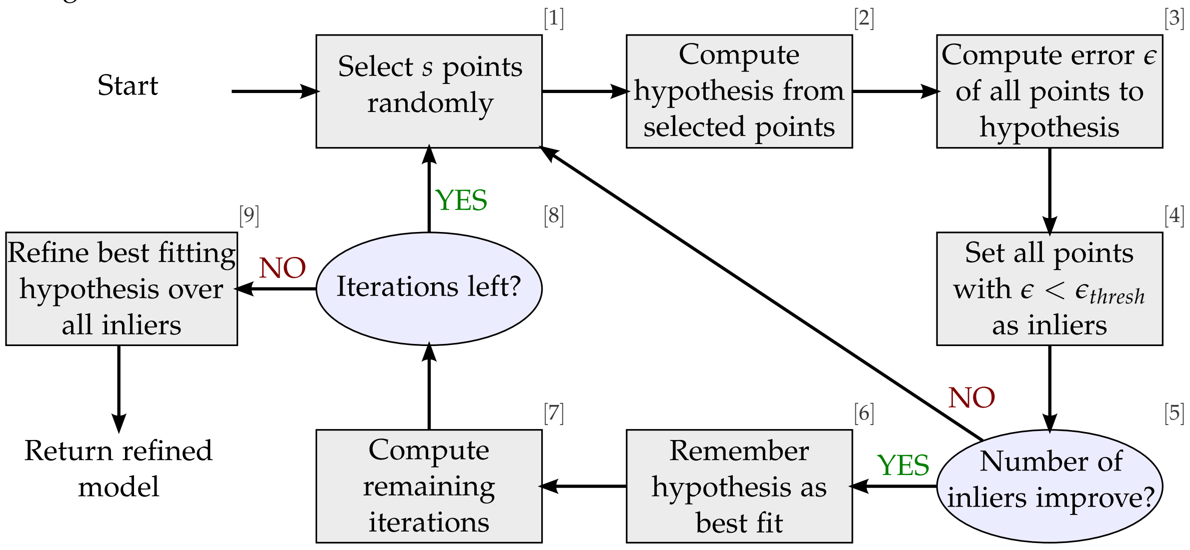

[1]
[2]
[3]
Start
Select s points randomly
Compute hypothesis from selected points
Compute error ϵ of all points to hypothesis
YES
[9]
[8]
[4]
Refine best fitting hypothesis over all inliers
NO
Iterations left?
Set all points with $\epsilon < \epsilon_{thresh}$ as inliers
NO
[7]
[6]
[5]
Return refined model
Compute remaining iterations
Remember hypothesis as best fit
YES
Number of inliers improve?

Figure 4.12: Graphical representation of of how RANSAC iteratively guesses models, evaluates them, retains the current best fitting one and ultimately terminates with a best guess.

Model Size and RANSAC Iterations

The number of remaining iterations is recomputed after every iteration using the a probabilistic heuristic. Given the following parameters

$$\begin{aligned} s &= \text{minimal size of the model} \\ \epsilon &= \text{estimated fraction of outliers} \\ N &= \text{the number of necessary iterations left to guarantee an outliers free solution} \\ p &= \text{desired probability of producing a usable result} \end{aligned}$$

we can define p as

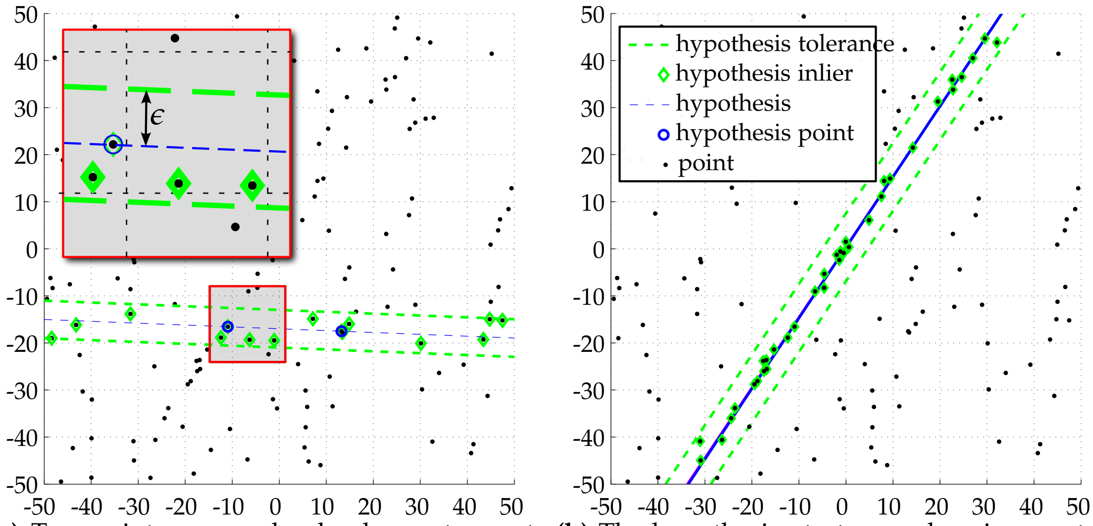
$$p = 1 - (\overbrace{1 - \underbrace{(1-\epsilon)^s}_{\text{Probability of choosing s inliers in a row}}}^{\text{Probability that N samples were contaminated}})^N \tag{4.1}$$

after which we can determine the number of required iterations by solving for N, giving

$$N = \frac{\log(1-p)}{\log(1-(1-\epsilon)^s)}. \tag{4.2}$$

Table 4.3 summarises the number of iterations N required with respect to various s and ϵ, computed using equation (4.2) with $p = 0.99$.

Viewing the table one can observe that the number of iterations N is exponential in the



(a) Two points are randomly chosen to create a model for a line. All other points within ϵ distance from the line are considered inliers.

(b) The hypothesize-test procedure is repeated N times according to Equation 4.2 resulting the in the final hypothesis with classified inliers.

Figure 4.13: A brief example of using RANSAC to estimate the best line model to the noisy data. Illustrated is a set of 250 2D points within a 100×100 grid of which 75% are outliers.

Table 4.3: Number N of RANSAC iterations given $p = 0.99$ for varying s and ϵ

Model Size (s)	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
$\epsilon = 0.2$	3	5	6	9	12	15	20	25
$\epsilon = 0.3$	4	7	11	17	25	37	54	78
$\epsilon = 0.4$	5	10	19	33	57	96	162	272
$\epsilon = 0.5$	7	16	34	71	145	292	587	1177
$\epsilon = 0.6$	9	26	70	178	447	1122	2808	7025
$\epsilon = 0.7$	13	49	168	566	1893	6315	21055	70188
$\epsilon = 0.8$	21	113	573	2876	14389	71953	359777	1798892

minimum number of points necessary to best fit the model. Therefore, it is desirable to have a minimal parametrisation of the model.

RANSAC will be used through out this thesis to robustly estimate camera motion between camera images. Introduced by [20] and highlighted in section 4.1.10 and section 4.1.10, the model estimating camera motion between two frames is reduced from size $s = 3$ to $s = 2$ by incoporating IMU measurements, thereby decreasing the number of required iterations.

4.1.4 Pinhole Camera Model

This subsection deals with relating our 2D measurements (such as detected keypoints) on the image plane to the world that has been (possibly imperfectly) projected onto the image plane.

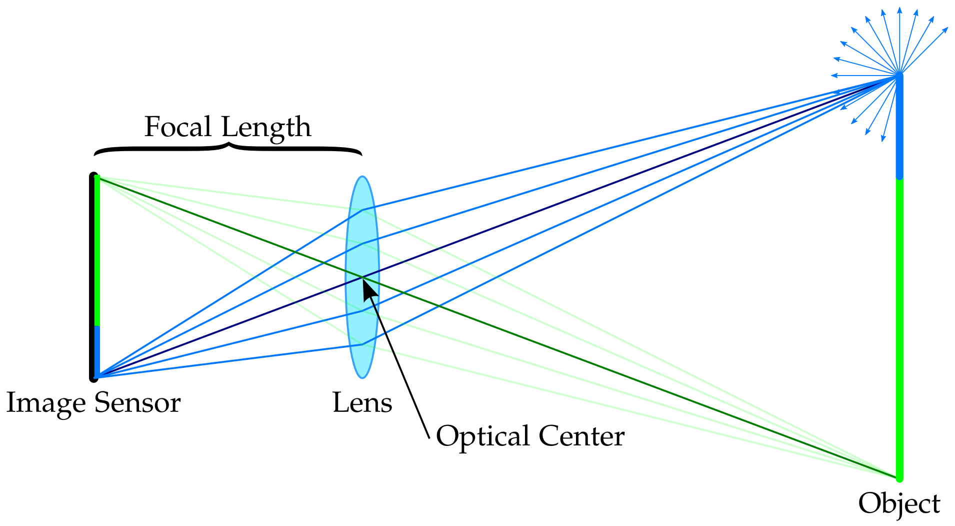


Figure 4.14: Incoming light gets reflected from a single point of an object, of which some hits the camera lens which focuses the light onto a single point on the imaging sensor.

Digital cameras capture images by measuring light projected onto an *imaging sensor*, usually a charge-coupled device (CCD) or a complementary metaloxidesemiconductor (CMOS) array. A lens ensures that all light travelling from a single point in the real world that happens to hit the lens gets focussed on to the same point on the image sensor. This only works if the points in the real world are at the correct focusing distance, if they are further or closer away, the lens cannot focus the light to a singular point but rather a disc, with a radius proportional to the focusing distance offset.

Figure 4.14 shows light rays reflecting from an object in all directions, of which a subset are captured by a lens and focussed onto the same point on the image sensor. Light incoming from different points in the world are focussed onto different points on the image sensor. In reality, cameras might use multiple lenses. Further more, no lens is perfect and minor production imperfections can cause significant distortion and/or blur.

To simplify things, we only consider rays of light that pass through the optical center.

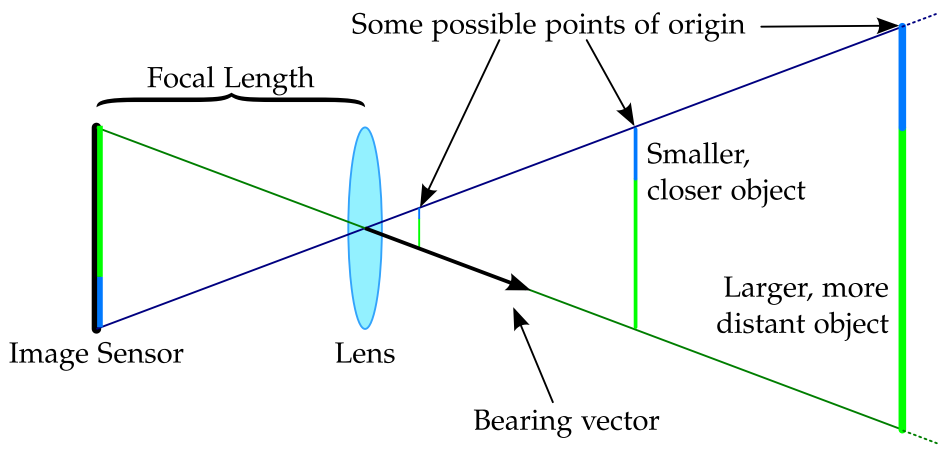


Figure 4.15: This figure shows 3 of the infinitely many size-distance combinations the green-blue object could have, without changing what the imaging sensor would register. It also shows a bearing vector point towards the bottom of the object.

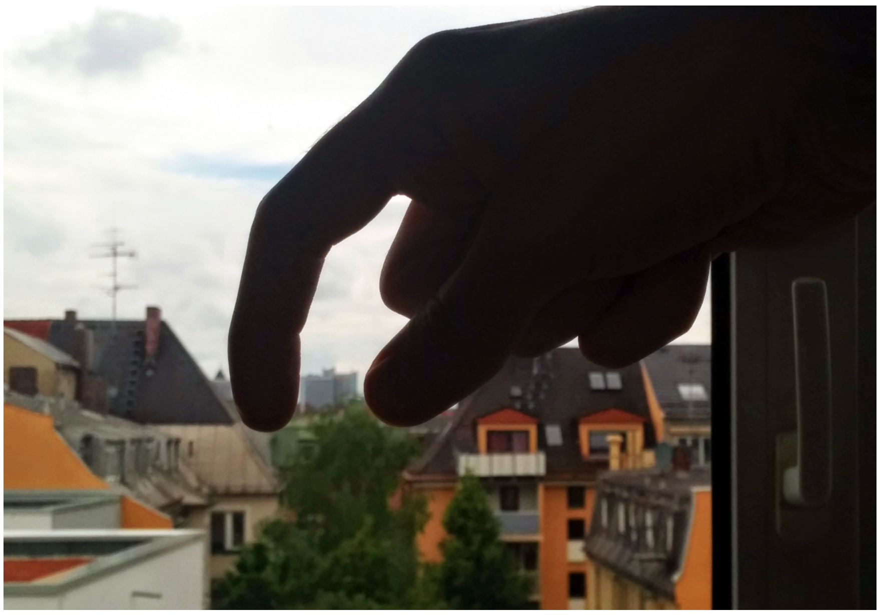


Figure 4.16: In the photo, a house is visible in the distance which is indeed larger than the gap between the thumb and index finger. Looking at this 2D image as humans we know this, because we have an intuitive understanding that buildings on the horizon are large and far away, while our own hands are at arms length and smaller. Computers do not have this ontogenic knowledge and therefore have no way of knowing the size or distance of the building.

This reduces the complexity of the geometry later as all light passing through this point travels in a straight line, giving us the direction the observed point is in from the optical center. Note that we cannot tell how far the light has travelled before it hits our imaging sensor, it could have been light-years or nano-meters away. As a side effect, we cannot tell if an object is small and close, or large and far away. This loss of depth information when projecting the 3D world onto a 2D surface is called scale ambiguity, as illustrated in Figure 4.15 and demonstrated in Figure 4.16. In the context of SfM algorithms, the ratio between the scale in the algorithm and the metric scale is call the *visual scale factor*

Later we represent the direction from the optical center, along the ray of light, to the object where the light emanated from as a vector. As we do not know the vector length, we fix the length to 1, obtaining a unit vector which we will call *bearing vector* from now on.

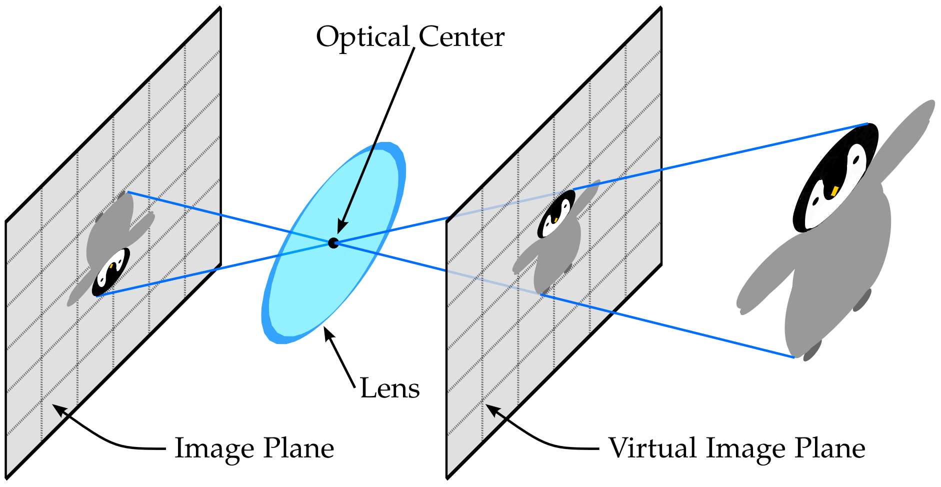


Figure 4.17: One can consider the geometrically equivalent virtual image plane which then has the uninverted image projected 'through' it.

To further simplify things, we consider the virtual image plane, which we place in front of the optical center with the same focal distance. This then has the uninverted image projected onto it and is geometrically equivalent to the real image plane as Figure 4.17 illustrates.

We essentially have described the pinhole camera model above. It describes the camera as a single point (i.e. the 'hole') in space through which light from the observed scene passes before hitting the image plane.

Camera Projection

Camera projection deals with mapping 3D points in space to and from their projection onto the 2D imaging plane. If we assume we have a perfect lens and there is no distortion the mapping function is at its simplest. Figure 4.18 defines the camera and image frames and shows the mapping of a 3D point ${}^{C}\mathbf{X} = (X, Y, Z)^T$ in the camera frame onto the image plane at point $\mathbf{x} = (u_{\mathbf{x}}, v_{\mathbf{x}})^T$. The camera z-axis is called the principal axis, and the principal point is defined to be the intersection point between the principal axis and the

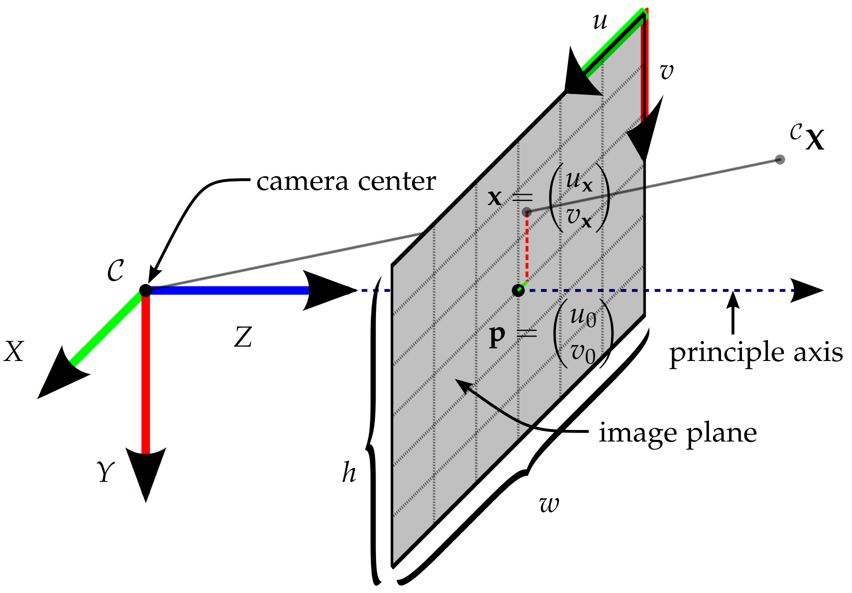

u
v
${}^{\mathcal{C}}\mathbf{X}$
camera center
$\mathbf{x} = \begin{pmatrix} u_\mathbf{x} \\ v_\mathbf{x} \end{pmatrix}$
$\mathcal{C}$
X
Z
$\mathbf{p} = \begin{pmatrix} u_0 \\ v_0 \end{pmatrix}$
principle axis
image plane
Y
h
w

Figure 4.18: Pinhole Camera Geometry

image plane. The focal length f is given by the shortest distance between the image plane and camera center as illustrated in Figure 4.19. Here one can see that one can determine $\mathbf{x}$ using geometry of similar triangles, resulting in

$$u_\mathbf{x} = f\frac{X}{Z} \qquad\qquad v_\mathbf{x} = f\frac{Y}{Z} \tag{4.3}$$

If f is given in meters we need to convert to pixel units by dividing by the pixel size d. As pixels are usually not square we distinguish between d_x and d_y. We define $f_x = f/d_x$ and $f_y = f/d_y$ giving us units in pixel distance. Furthermore, we wish to have the pixel coordinate $(0,0)^T$ be on the top left so we add the center pixel planar offset $(u_0, v_0)^T$ resulting in

$$u_\mathbf{x} = f_x\frac{X}{Z} + u_0 \qquad\qquad v_\mathbf{x} = f_y\frac{Y}{Z} + v_0 \tag{4.4}$$

Putting everything together and using homogeneous coordinates we can express the projection function as matrix operation

$$\begin{pmatrix} u_\mathbf{x} \\ v_\mathbf{x} \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & u_\mathbf{x} & 0 & 0 \\ 0 & f_z & u_\mathbf{x} & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} = \mathbf{K} \begin{pmatrix} {}^{\mathcal{C}}\mathbf{X} \\ 1 \end{pmatrix} \tag{4.5}$$

where $\mathbf{K}$ is called the matrix of intrinsic parameters.

To back-projection a point ${}^{\mathcal{W}}\mathbf{X}$ in the world frame into image space one must also consider the exterior orientation of the camera and transform the point to the camera frame $\mathcal{C}$. If

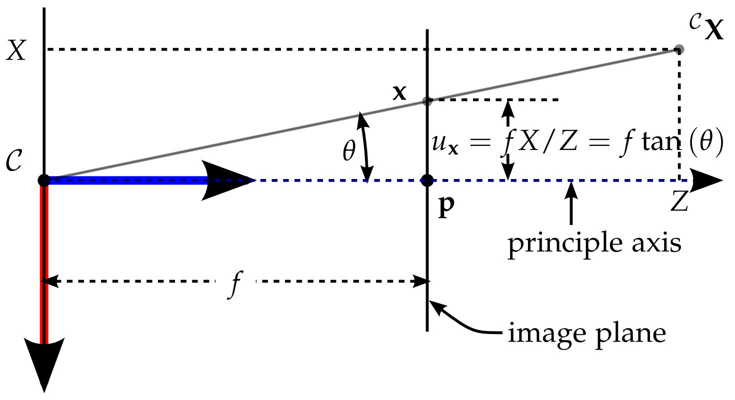


Figure 4.19: Pinhole Camera Geometry

${}^{\mathcal{W}}_{\mathcal{C}}\vec{\mathbf{t}}$ denotes the position of the camera center in a world frame and ${}^{\mathcal{W}}_{\mathcal{C}}\mathbf{R}$ the rotation, we can put everything together

$$\begin{pmatrix} u_{\mathbf{x}} \\ v_{\mathbf{x}} \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & u_{\mathbf{x}} & 0 & 0 \\ 0 & f_z & u_{\mathbf{x}} & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \left({}^{\mathcal{W}}_{\mathcal{C}}\mathbf{R} - {}^{\mathcal{W}}_{\mathcal{C}}\mathbf{R}^T \, {}^{\mathcal{W}}_{\mathcal{C}}\vec{\mathbf{t}} \right) \begin{pmatrix} {}^{\mathcal{W}}\mathbf{X} \\ 1 \end{pmatrix} = \mathbf{K} \; \mathbf{P} \begin{pmatrix} {}^{\mathcal{C}}\mathbf{X} \\ 1 \end{pmatrix} \tag{4.6}$$

where **P** is called the matrix of extrinsic parameters.

4.1.5 FOV Camera Model

The lens we employ in subsection 3.3.1 is a cheap, wide angle lens. While a wide angle lens is great as a greater portion of the scene is projected onto the image sensor, they so at the expense of introducing considerable radial distortion. In this subsection we will look at a different camera model that can handle large field of views better.

The cheap and wide angle nature of the lens results in a distorted image for which the usual projective pinhole camera model with radial and tangential factors are insufficient to accurately rectify. The best calibration achieved with the *ROS camera_calibration* package resulted in reprojection errors with over 8 pixel *RMS* (where an *RMS* < 0.5 pixels would be desired). Additionally, the calibration could not deal well with the large opening angle of the lens - the outer third of the image suffered from large incorrect distortions, rendering only the center part of the image useful.

The pinhole model is often referred to as rectilinear projection as it preserves the rectilinearity of the projected scene. This means that straight lines in the world are projected as straight lines on the image sensor. The rectilinear projection mapping function is given as

$$r_u = f \; \tan\theta \tag{4.7}$$

where r_u is the projected radial distance from the principal point on the image plane and θ is the incident angle of the projected ray to the optical axis of the camera. See Figure 4.19 (in this case $r_u = x_u$) for an illustration. However for wide FOV lenses under using the pinhole projection, the size of the projected image becomes very large, even approaching infinity when the rays are almost parallel to the image sensor at a field of view approaching 180°. Therefore, we need incorporate the wide angle nature of the used lens into the projection function.

[70] proposed a distortion model for fish-eye lenses which they call the field of view (FOV) model. It was designed from the ground up with fish-eye lenses in mind, taking their overall design into account: the angular resolution is usually roughly proportional to the image resolution along an image radius. This means the distance between an image point and the image center is roughly proportional to the angle between the bearing vector pointing to the corresponding 3D point from the optical center and the optical axis. See Figure 4.20 for an illustration.

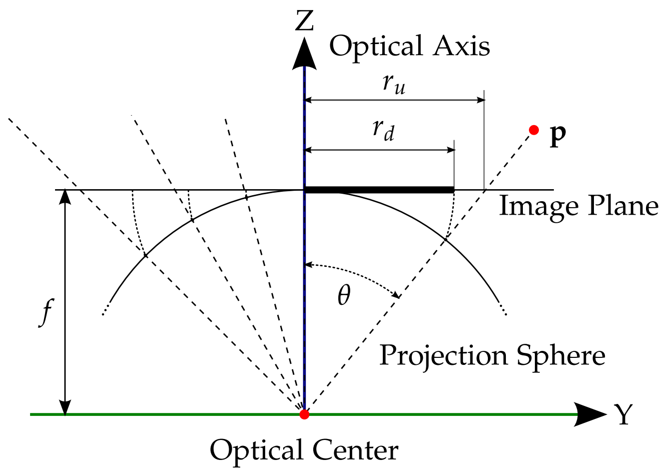


Figure 4.20: FOV distortion model projection representations, showing the projection of point **p** to the projection sphere and the following reprojection of the point on the projection sphere to the image plane. Intuitvely, the distance r_d is proportional to the angle θ.

They named this model after the only paramters it takes, the field of view (FOV) ω of the corresponding ideal fish-eye lens. This angle ω may not actually correspond to the real camera's FOV, since the fish-eye optics may not exactly follow this model. The FOV distortion function is given by

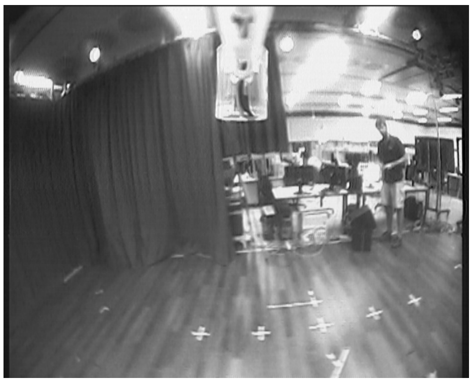
$$r_d = \frac{1}{\omega} \arctan \left(2 r_u \tan \left(\frac{\omega}{2} \right) \right) \tag{4.8}$$

and the inverse is given by

$$r_u = \frac{\tan \left(r_d \omega \right)}{2 \tan \left(\frac{\omega}{2} \right)} \tag{4.9}$$

which describe the conversion between rectilinear image space and the FOV fish-eye model image space and vice versa.

To model further imperfections in the optics, the distortion function can be concatenated with polynomial elements to account for deviations between the lens and the projection


(a) Unrectified image.

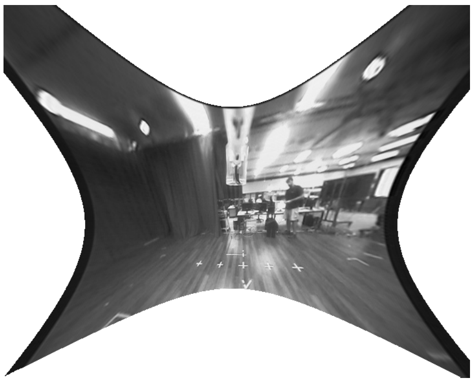

(b) Rectified image.

Figure 4.21: Comparison of the same image before and after rectification.

function. The FOV model is basically the first order parameter, so when we add the additional radial distortion coefficients A_n we obtain:

$$r_d = \frac{1}{\omega} \arctan\left(2r_u \tan\left(\frac{\omega}{2}\right)\right) + \Delta r_d \tag{4.10}$$

where

$$\Delta r_d = A_1 r_u^3 + A_1 r_u^5 + A_1 r_u^7 \tag{4.11}$$

A calibration procedure such as the one outlined in[70] and implemented in the work of [33] can be used to estimate all the parameters, given camera images viewing a rectangular pattern.

4.1.6 Bearing Vectors and Error Models

Using the notation from subsection 4.1.4, we can estimate the optimal *absolute* camera pose (extrinsic camera matrix) ${}^{\mathcal{W}}_{\mathcal{C}}\mathbf{P}_{optimal}$ from measurements ${}^{\mathcal{W}}\mathbf{x}_i = (u_\mathbf{i}, v_\mathbf{i})^T$ of observations of points ${}^{\mathcal{W}}\mathbf{X}_i$ by iteratively minimising the optimal least squares formulation

$${}^{\mathcal{W}}_{\mathcal{C}}\mathbf{P}_{optimal} = arg\ min\ E({}^{\mathcal{W}}_{\mathcal{C}}\mathbf{P}) = arg\ min\ \sum_i \epsilon_i^T \epsilon_i \tag{4.12}$$

over all n observations where

$$\epsilon_i = \begin{pmatrix} u_\mathbf{i} \\ v_\mathbf{i} \end{pmatrix} - \left[\begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \mathbf{K}\, {}^{\mathcal{W}}_{\mathcal{C}}\mathbf{P} \begin{pmatrix} {}^{\mathcal{W}}\mathbf{X}_i \\ 1 \end{pmatrix}\right]^{-1} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \mathbf{K}\, {}^{\mathcal{W}}_{\mathcal{C}}\mathbf{P} \begin{pmatrix} {}^{\mathcal{W}}\mathbf{X}_i \\ 1 \end{pmatrix} \tag{4.13}$$

is the reprojection error.

One can also optimise the optimal *relative* transformation ${}^{\mathcal{C}}_{\mathcal{C}'}\mathbf{P}$ between two camera frames $\mathcal{C}$ and $\mathcal{C}'$ by expressing all the points in the first frame $\mathcal{C}$ and minimising the joint reprojection error

$$\epsilon_i = \begin{pmatrix} \begin{pmatrix} u_\mathbf{i} \\ v_\mathbf{i} \end{pmatrix} - \left[\begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \mathbf{K}\, \mathbf{I}_{3\times 4} \begin{pmatrix} {}^{\mathcal{C}}\mathbf{X}_i \\ 1 \end{pmatrix} \right]^{-1} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \mathbf{K}\, \mathbf{I}_{3\times 4} \begin{pmatrix} {}^{\mathcal{C}}\mathbf{X}_i \\ 1 \end{pmatrix} \\ \begin{pmatrix} u'_\mathbf{i} \\ v'_\mathbf{i} \end{pmatrix} - \left[\begin{pmatrix} 0 & 0 & 1 \end{pmatrix} \mathbf{K}\, {}^{\mathcal{C}}_{\mathcal{C}'}\mathbf{P} \begin{pmatrix} {}^{\mathcal{C}}\mathbf{X}_i \\ 1 \end{pmatrix} \right]^{-1} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \end{pmatrix} \mathbf{K}\, {}^{\mathcal{C}}_{\mathcal{C}'}\mathbf{P} \begin{pmatrix} {}^{\mathcal{C}}\mathbf{X}_i \\ 1 \end{pmatrix} \end{pmatrix} \tag{4.14}$$

over all n points in both frames.

Note that the world points ${}^{\mathcal{W}}\mathbf{X}_i$ need to be re-triangulated after each optimisation step as the relative transformation is iteratively updated. This two-frame non-linear least squares optimisation essentially represents an interleaving batch optimization approach and requires the back projecting of the world points in each iteration - a potentially costly procedure depending on the complexity of the camera model. Following the framework of presented in [20], we can obtain a less computationally expensive error estimation by first transforming all 2D image measurements into unit bearing vectors and then using the angle between these bearings to define our error ϵ. A 2D image measurement $\mathbf{x_i}$ is transformed into a unit bearing vector $\vec{\mathbf{f}}_\mathbf{i} = \begin{pmatrix} f_{xi} & f_{yi} & f_{zi} \end{pmatrix}$ by

$$\vec{\mathbf{f}}_\mathbf{i} = \frac{\mathbf{K}^{-1} \begin{pmatrix} u_i & v_i & 1 \end{pmatrix}^T}{\left\| \mathbf{K}^{-1} \begin{pmatrix} u_i & v_i & 1 \end{pmatrix}^T \right\|} \tag{4.15}$$

Instead of defining our error ϵ to be the explicit angle $\alpha = atan2(\|(f_i \times f'_i)\|, f_i f'_i$ between two bearing vectors f_i and f'_i, we can use an alternative error metric. As [71] points out, the most efficient way to error representation using the angle between two bearing vectors is given by taking their dot product, which is equals to $\cos\alpha$. As this returns a value between over $[1, -1]$ and as we wish to have an error that minimises to zero, we simply subtract it from one giving

$$\epsilon = 1 - \vec{\mathbf{f}}_\mathbf{i}^{\,T} \vec{\mathbf{f}}'_\mathbf{i} = 1 - \cos\alpha \tag{4.16}$$

to express our error. Other error functions include

$$\epsilon = \left\| \vec{\mathbf{f}}_\mathbf{i} \times \vec{\mathbf{f}}'_\mathbf{i} \right\| \qquad \epsilon = \left\| \vec{\mathbf{f}}_\mathbf{i} - \vec{\mathbf{f}}'_\mathbf{i} \right\| \qquad \epsilon = (\vec{\mathbf{f}}_\mathbf{i} - \vec{\mathbf{f}}'_\mathbf{i})^T \, (\vec{\mathbf{f}}_\mathbf{i} - \vec{\mathbf{f}}'_\mathbf{i}) \tag{4.17}$$

which are all plotted in Figure 4.22.

Sometimes one needs to define if an error is within a threshold. RANSAC (see section 4.1.3) for example needs to determine if a measured bearing vector $\vec{\mathbf{f}}_{meas}$ is close enough to the actual bearing vector $\vec{\mathbf{f}}_{model}$ explained by the model. Intuitively we can adopting a threshold angle α_{thresh}, which constrains $\vec{\mathbf{f}_\mathbf{model}}$ to lie within a cone with opening angle α_{thresh} of axis $\vec{\mathbf{f}}_{meas}$ as illustrated in Figure 4.23. If one wishes to express the reprojection threshold in pixels px_{thresh}, one can approximate it with $\alpha_{thresh} = \arctan\left(px_{thresh}/f\right)$ where f is the focal length giving

$$\epsilon_{thresh} = 1 - \cos\left(\alpha_{thresh}\right) = 1 - \cos\left(\arctan\left(\frac{px_{thresh}}{f}\right)\right). \tag{4.18}$$

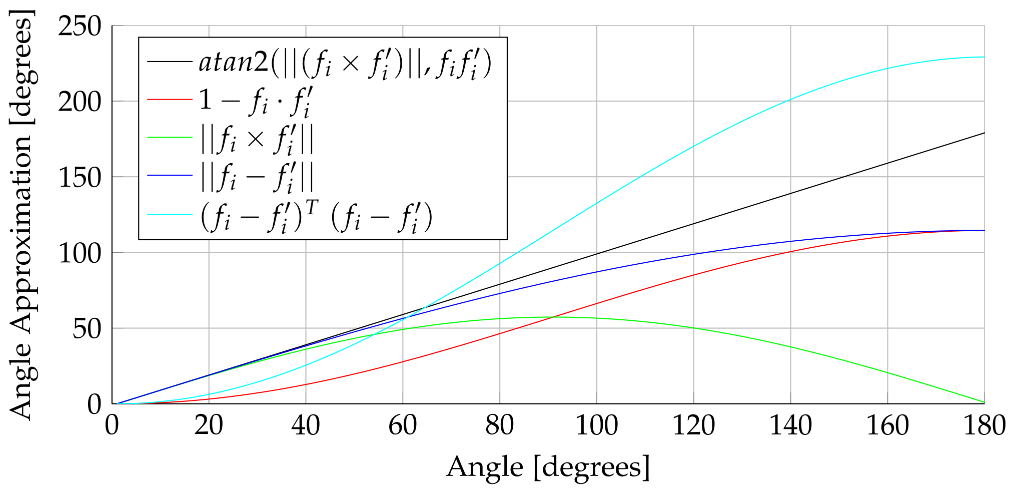


Figure 4.22: Various Approximation functions for angles between unit bearing vectors.

For the remainder of this thesis we use dot-products between the vectors on the unit

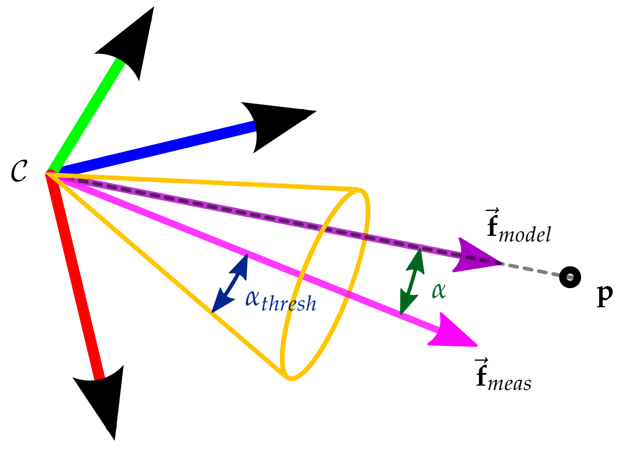


Figure 4.23: Errors between bearing vectors are given by the angle between them. The bearing vector is within the threshold if it lies within the illustrated cone.

sphere as described by equation (4.16) as our error function, resulting in the following reprojection error which can analogously be used for the relative pose case:

$$\epsilon_i = 1 - \vec{\mathbf{f}}_{\mathbf{i}}^{\,T} \cdot \left[\frac{{}^{\mathcal{W}}_{\mathcal{C}}\mathbf{P} \left({}^{\mathcal{C}}\mathbf{X}_i^T \quad 1 \right)^T}{\left\| {}^{\mathcal{W}}_{\mathcal{C}}\mathbf{P} \left({}^{\mathcal{C}}\mathbf{X}_i^T \quad 1 \right)^T \right\|} \right] \tag{4.19}$$

Any camera model can then be used as long as they support projecting image features onto the unit sphere.

4.1.7 IMU Priors

The Crazyflie does on-board attitude estimation at 250 Hz (section 4.3) so we have a readily available 3D rotation estimates at all times. This allows us to estimate relative rotations between two frames.

In [72] a similar idea was used to increase the robustness of a SLAM framework by estimating inter-frame rotation to aid tracking. There the authors estimate inter-frame rotation by using direct second-order minimisation to aling sub-sampled and blurred images instead of using IMU measurements.

When using an IMU, the yaw rotation priors drift, but if we only compare rotation estimates that are temporarily bounded (such as those temporarily aligned with successive keyframes/frames) the angular errors introduced remain small and can be easily compensated for later in nonlinear refinement steps.

In [73] the authors show how one can incorporate these short term full 3D relative rotations to support the geometric computation of the relative and absolute pose problem formulations (see section 4.1.10 and section 4.1.10 respectively), reducing the minimum number of points of both models down to two, thereby helping to minimise the required RANSAC iterations (see Table 4.3).

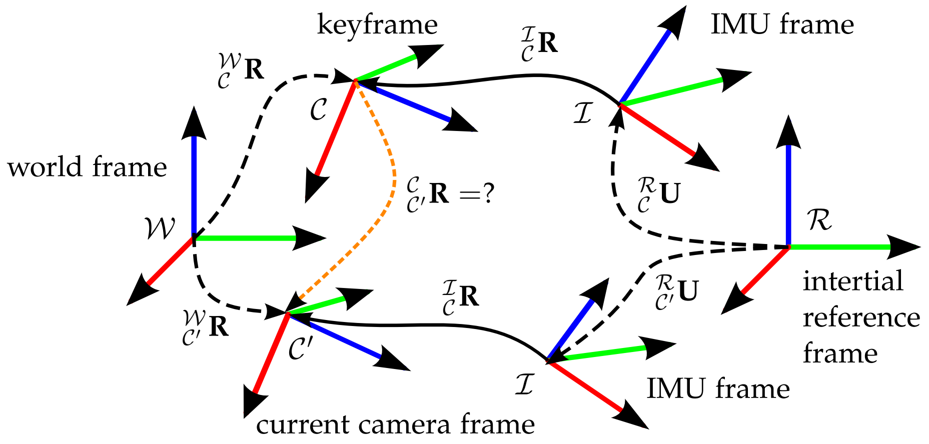

keyframe
$^{\mathcal{I}}_{\mathcal{C}}\mathbf{R}$
IMU frame
$^{\mathcal{W}}_{\mathcal{C}}\mathbf{R}$
$\mathcal{C}$
$\mathcal{I}$
world frame
$^{\mathcal{C}}_{\mathcal{C}'}\mathbf{R} = ?$
$^{\mathcal{R}}_{\mathcal{C}}\mathbf{U}$
$\mathcal{W}$
$\mathcal{R}$
$^{\mathcal{R}}_{\mathcal{C}'}\mathbf{U}$
intertial reference frame
$^{\mathcal{I}}_{\mathcal{C}}\mathbf{R}$
$^{\mathcal{W}}_{\mathcal{C}'}\mathbf{R}$
$\mathcal{C}'$
$\mathcal{I}$
IMU frame
current camera frame

Figure 4.24: The rotation between the IMU and camera $^{\mathcal{I}}_{\mathcal{C}}\mathbf{R}$ is assumed to be known and fixed. $^{\mathcal{R}}_{\mathcal{C}}\mathbf{U}$ gives the orientation of the IMU in its inertial reference frame with C indicating the camera frame that the rotation temporarily corresponds to. $^{\mathcal{C}}_{\mathcal{C}'}\mathbf{R}$ is the relative rotation of a current camera frame C' with respect to a keyframe C.

To determine the relative rotation between two frames C and C', we introduce the IMU frame $\mathcal{I}$ and its inertial reference frame $\mathcal{R}$. We assume the rotation between the IMU and camera $^{\mathcal{I}}_{\mathcal{C}}\mathbf{R}$ is known and fixed. This can be computed offline in a number of different IMU to camera calibration methods such as those presented in [74] and [75]. Using Figure 4.24 as a reference, one can easily see that the rotation of the current camera frame C' with respect to a keyframe C can be computed using

$$^{\mathcal{C}}_{\mathcal{C}'}\mathbf{R} = {}^{\mathcal{I}}_{\mathcal{C}}\mathbf{R}^T \cdot {}^{\mathcal{R}}_{\mathcal{C}}\mathbf{U}^T \cdot {}^{\mathcal{R}}_{\mathcal{C}'}\mathbf{U} \cdot {}^{\mathcal{I}}_{\mathcal{C}}\mathbf{R}. \qquad (4.20)$$

In case we know the absolute rotation of $\mathcal{C}$, we can obtain a prior of the absolute rotation of $\mathcal{C}'$ by

$$
{}^{\mathcal{W}}_{\mathcal{C}'}\mathbf{R} = {}^{\mathcal{W}}_{\mathcal{C}}\mathbf{R} \cdot {}^{\mathcal{C}}_{\mathcal{C}'}\mathbf{R} = {}^{\mathcal{W}}_{\mathcal{C}}\mathbf{R} \cdot {}^{\mathcal{I}}_{\mathcal{C}}\mathbf{R}^T \cdot {}^{\mathcal{R}}_{\mathcal{C}}\mathbf{U}^T \cdot {}^{\mathcal{R}}_{\mathcal{C}'}\mathbf{U} \cdot {}^{\mathcal{I}}_{\mathcal{C}}\mathbf{R}. \tag{4.21}
$$

Unrotating Bearing Vectors

Once one has estimated the relative rotation between two camera frames C and C', one can rotate the bearing vectors $\vec{\mathbf{f}}'$ of C' into the frame of C giving ${}^{C}\vec{\mathbf{f}}'$ by

$$
{}^{\mathcal{C}}\vec{\mathbf{f}}' = {}^{\mathcal{C}}_{\mathcal{C}'}\mathbf{R} \cdot \vec{\mathbf{f}}_i \tag{4.22}
$$

Expressing corresponding bearing vectors in the same rotational frame can be used in a number of ways such as to aid feature matching (see section 4.1.2) and help us decide when to trigger new keyframes (see subsection 4.1.9).

4.1.8 Triangulation

One requires 3D points in the world (landmarks) to compute an absolute pose from or to optimise a relative pose.

This subsection introduces triangulation, the process of determining a point in 3D space given its projections onto two frames and a relative transformation between. Intuitively this is done by intersecting the back-projected bearings (rays) of 2D image correspondences. The correspondences come from feature matching as described in section 4.1.2.

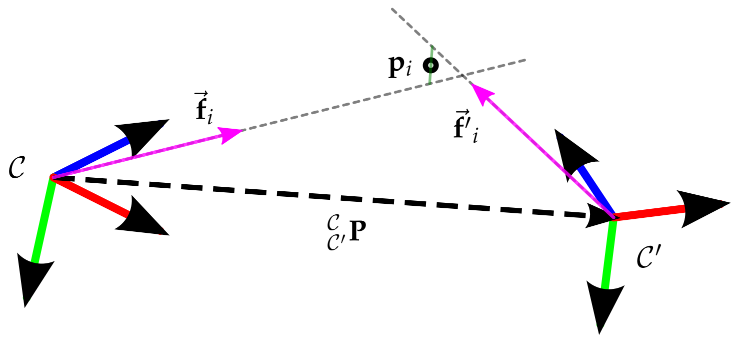


Figure 4.25: Triangulating a point visible from two views

The situation is illustrated in Figure 4.25. It is obvious that the baseline (i.e. the translation) between the two frames $\mathcal{C}$ and $\mathcal{C}'$) must be non-zero (i.e. we must have a triangle). In reality, the bearing vectors never really intersect due to (*a*) image noise, (*b*) camera model/calibration errors, (*c*) keypoint detection accuracy, and (*d*) feature matching uncertainty. Therefore the point at minimal distance from all the intersecting bearings is used as the 3D point position estimate. When one deals with noisy measurements the baseline magnitude corresponds to the uncertainty of the triangulation, as illustrated in Figure 4.26.

Therefore it is desirable to have a sufficiently large baseline before initiating point triangulation. How we decide when to trigger triangulation is discussed in subsection 4.1.9. Once one has triangulated 3D points one can then compute absolute poses and optimise relative poses relative to these points as discussed in subsection 4.1.10.

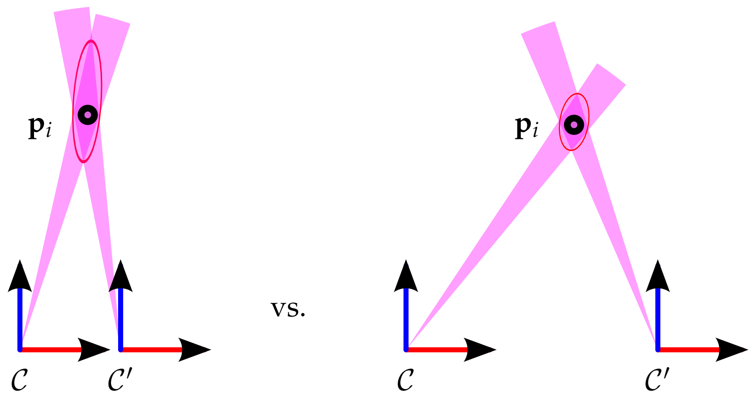


Figure 4.26: The point triangulation uncertainty decreases with a bigger baseline.

Each 3D point $\mathbf{p}_i = \begin{pmatrix} p_{xi} & p_{yi} & p_{zi} \end{pmatrix}$ that can be observed by two frames ${}^{\mathcal{W}}_{\mathcal{C}}\mathbf{P}$ and ${}^{\mathcal{W}}_{\mathcal{C}'}\mathbf{P}$ has two associated unit bearing vectors $\vec{\mathbf{f}}_\mathbf{i}$ and $\vec{\mathbf{f}}'_\mathbf{i}$. If ${}^{\mathcal{C}}_{\mathcal{C}'}\mathbf{P} = \begin{pmatrix} {}^{\mathcal{C}}_{\mathcal{C}'}\mathbf{R}^T & -{}^{\mathcal{C}}_{\mathcal{C}'}\mathbf{R}^T \, {}^{\mathcal{C}}_{\mathcal{C}'}\vec{\mathbf{t}} \end{pmatrix}$ back projects points expressed in camera frame $\mathcal{C}$ to $\mathcal{C}'$, then one can derive

$$\frac{f_{xi}}{f_{zi}} = \frac{{}^{\mathcal{C}}\mathbf{p}_{ix}}{{}^{\mathcal{C}}\mathbf{p}_{iz}}, \quad \frac{f_{yi}}{f_{zi}} = \frac{{}^{\mathcal{C}}\mathbf{p}_{iy}}{{}^{\mathcal{C}}\mathbf{p}_{iz}}, \quad \frac{f'_{xi}}{f'_{zi}} = \frac{{}^{\mathcal{C}'}\mathbf{p}_{ix}}{{}^{\mathcal{C}'}\mathbf{p}_{iz}} \quad \text{and} \quad \frac{f'_{yi}}{f'_{zi}} = \frac{{}^{\mathcal{C}'}\mathbf{p}_{iy}}{{}^{\mathcal{C}'}\mathbf{p}_{iz}} \tag{4.23}$$

from the bearing constraints and using ${}^{\mathcal{C}}\mathbf{p}_i = {}^{\mathcal{C}}_{\mathcal{C}}\mathbf{P} \begin{pmatrix} {}^{\mathcal{C}}\mathbf{p}_i^T & 1 \end{pmatrix}^T$ and ${}^{\mathcal{C}'}\mathbf{p}_i = {}^{\mathcal{C}}_{\mathcal{C}'}\mathbf{P} \begin{pmatrix} {}^{\mathcal{C}}\mathbf{p}_i^T & 1 \end{pmatrix}^T$ formulate the linear least square problem:

$$\begin{pmatrix} f_{xi} \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} {}^{\mathcal{C}}_{\mathcal{C}}\mathbf{P} - f_{zi} \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} {}^{\mathcal{C}}_{\mathcal{C}}\mathbf{P} \\ f_{yi} \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} {}^{\mathcal{C}}_{\mathcal{C}}\mathbf{P} - f_{zi} \begin{pmatrix} 0 & 1 & 0 \end{pmatrix} {}^{\mathcal{C}}_{\mathcal{C}}\mathbf{P} \\ f'_{xi} \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} {}^{\mathcal{C}}_{\mathcal{C}'}\mathbf{P} - f'_{zi} \begin{pmatrix} 1 & 0 & 0 \end{pmatrix} {}^{\mathcal{C}}_{\mathcal{C}'}\mathbf{P} \\ f'_{yi} \begin{pmatrix} 0 & 0 & 1 \end{pmatrix} {}^{\mathcal{C}}_{\mathcal{C}'}\mathbf{P} - f'_{zi} \begin{pmatrix} 0 & 1 & 0 \end{pmatrix} {}^{\mathcal{C}}_{\mathcal{C}'}\mathbf{P} \end{pmatrix} \cdot \begin{pmatrix} {}^{\mathcal{C}}\mathbf{p}_i \\ 1 \end{pmatrix} = \mathbf{0} \tag{4.24}$$

Using singular value decomposition (SVD) one can then derive the coordinate of ${}^{\mathcal{C}}\mathbf{p}_i$.

4.1.9 Angular Disparity

Here we define the concept of angular disparity between camera frames, which we will later use as method to trigger to new keyframes. Disparity is usual meant as the distance between two corresponding pixels in the image plane, but as we represent our 2D measurements as bearing vectors, we can compute an angular disparity instead, saving us the reprojection step. Furthermore one can efficiently and intuitively incorporate IMU rotation priors into the measurement.

We define the angular disparity ${}^{\mathcal{C}'}_{\mathcal{C}}\boldsymbol{\Delta}$ between two frames $\mathcal{C}$ and $\mathcal{C}'$ as the unrotated median difference between all the common observations between both frames.

The IMU is used to rotate all the unit bearings $\vec{\mathbf{f}}'$ from frame $\mathcal{C}'$ into the same rotational frame of $\mathcal{C}$ giving us ${}^{\mathcal{C}}\vec{\mathbf{f}}'$ as explained in section 4.1.7 and the difference between them is defined by equation (4.16). This means that ${}^{\mathcal{C}'}_{\mathcal{C}}\boldsymbol{\Delta}$ is small when the translational distance between both frames is small, regardless of the rotational difference. This procedure can effectively be seen to compensate for disparity caused by rotation. This is desired as it allows us to set an angular disparity threshold ${}^{\mathcal{C}'}_{\mathcal{C}}\boldsymbol{\Delta}_{thresh}$ above which we can assume our baseline is large enough to initiate our first triangulation of the common observations. A similar threshold can be set to trigger new keyframes. Figure 4.27 shows a simplified scenario with and without using the un-rotated bearings.

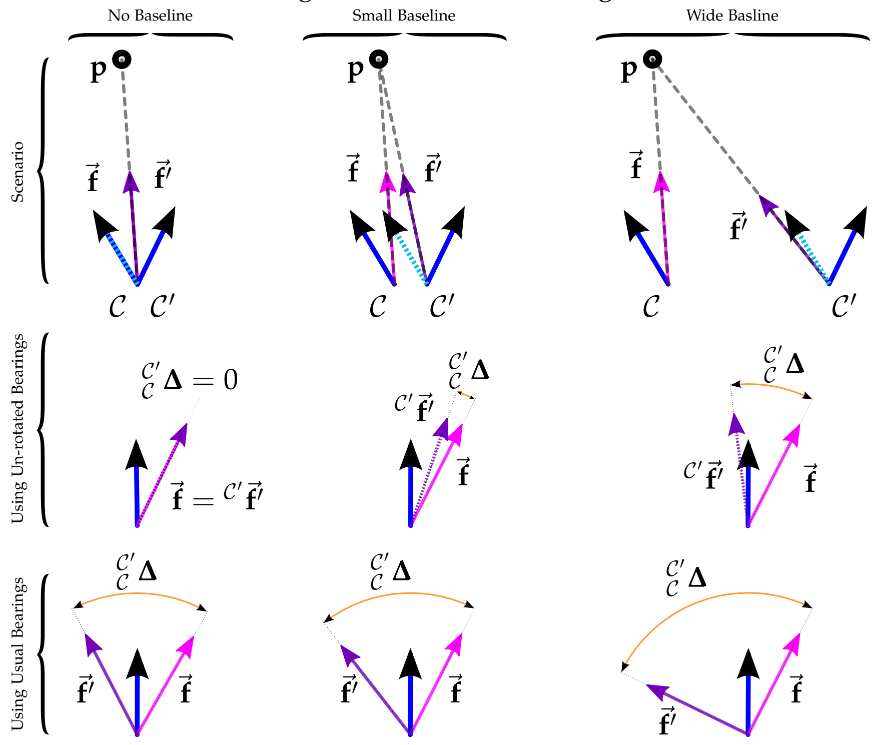

No Baseline
Small Baseline
Wide Basline
Scenario
Using Un-rotated Bearings
Using Usual Bearings
$\mathbf{p}$
$\vec{\mathbf{f}}$
$\vec{\mathbf{f}}'$
$\mathcal{C}$
$\mathcal{C}'$
${}^{\mathcal{C}'}_{\mathcal{C}}\boldsymbol{\Delta} = 0$
$\vec{\mathbf{f}} = {}^{\mathcal{C}'}\vec{\mathbf{f}}'$
${}^{\mathcal{C}'}_{\mathcal{C}}\boldsymbol{\Delta}$
${}^{\mathcal{C}'}\vec{\mathbf{f}}'$

Figure 4.27: A simple 2D example showing how unrotated bearing vectors allow us to reason over the baseline magnitude. In the first row, the scenario is presented: two frames $\mathcal{C}$ and $\mathcal{C}'$ share the common observation of landmark $\mathbf{p}$ and have associated unit bearing vectors. The dashed axis of $\mathcal{C}'$ is parallel to that of $\mathcal{C}$ by means of ${}^{\mathcal{C}'}_{\mathcal{C}}\mathbf{R}$. The left column has a baseline magnitude of zero (i.e. both frames share the same origin), the center column has a small baseline relative to the distance of $\mathbf{p}$, and the right column has a large baseline. The second row shows the angular disparity between both keyframes using the unrotated bearings. Notice that in the case of no baseline it is zero. The last row shows the disparity using the usual bearings, which is completely useless.

4.1.10 Relative and Absolute Camera Pose Estimation

In this section we discuss the heart of any monocular feature based SLAM system: the ability to estimate a relative camera pose given 2D-2D feature point associations and estimate an absolute camera pose relative to some 3D points in the world. We assume that we have a calibrated camera and can compute unit bearing vectors from 2D image plane measurements using Equation 4.15.

Relative Pose Estimation using 2D-2D Correspondences

The goal of the relative pose estimation problem is to recover the relative transformation between two camera frames that observe a common set of associated but unknown 3D landmarks, meaning only corresponding 2D image measurements in both frames are available. The situation is illustrated in Figure 4.29b.

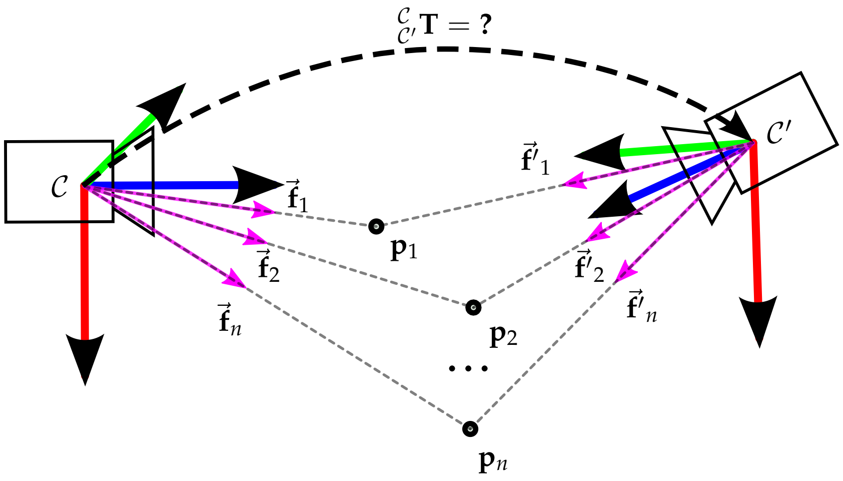


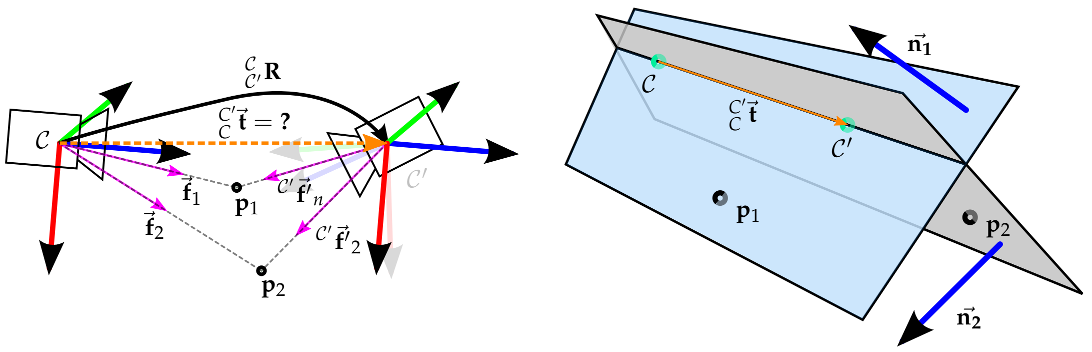
Figure 4.28: Relative Pose Problem: finding the pose of a camera w.r.t another camera given n 2D-2D correspondences between bearing vectors in the camera frames.

This is a well understood problem and multiple solutions exist. Five feature correspondences are required at minimum and therefore the problem is often called the *Five Point Relative Pose Problem* and was first solved in 1913 [76]. Additional solvers including five-point minimal solvers [77] and [78] and non-minimal six point [79], seven point [80], eight point [80] and seventeen point[81] solvers have since been developed. An excellent open source library called *OpenGV - Open Geometric Vision* implements a number of algorithms and is readily available [1].

Using Rotation Priors Of particular interest is the two point solver [73] that only solves for translation using a known rotation. As we can estimate the relative rotation between two frames using an IMU (see subsection 4.1.7) the approach is particularly well suited for our case. The scenario is given in Figure 4.29a and the derivation from[73]. We show it here for completeness.

Using equation (4.20) we can obtain ${}^{\mathcal{C}}_{\mathcal{C}'}\mathbf{T}$, the relative rotation between two frames $\mathcal{C}$ and $\mathcal{C}'$. One can then express the corresponding bearing vectors in frames with identical rotation

[1]OpenGV is available under `http://laurentkneip.github.io/opengv/`



(a) Given the relative rotation between two frames and two corresponding 2D measurements, estimate the translation direction.

(b) We can compute the translation as the intersection between the epipolar planes of each unrotated feature correspondance.

Figure 4.29: The two-point method that computes the translation between two camera frames with a known relative rotation, as derived in [73].

using (4.22). Then the normal vector of the epipolar frame of each feature correspondence is given by

$$\vec{\mathbf{n}}_\mathbf{i} = \vec{\mathbf{f}}_\mathbf{i} \times \left({}^{\mathcal{C}}_{\mathcal{C}'}\mathbf{R} \cdot \vec{\mathbf{f}}'_\mathbf{i} \right). \tag{4.25}$$

If the two epipolar planes are distinct one can then intersect them for the direction of the translation vector, given by

$$^{C'}_{C}\vec{\mathbf{d}} = \vec{\mathbf{n}}_\mathbf{1} \times \vec{\mathbf{n}}_\mathbf{1} \tag{4.26}$$

The scale is not determinable so we normalise it to have unit length

$$^{C'}_{C}\vec{\mathbf{t}} = \pm \frac{^{C'}_{C}\vec{\mathbf{d}}}{\left\| ^{C'}_{C}\vec{\mathbf{d}} \right\|} \tag{4.27}$$

and impose

$$\left(\vec{\mathbf{f}}_\mathbf{i} - {}^{\mathcal{C}}_{\mathcal{C}'}\mathbf{R} \cdot \vec{\mathbf{f}}'_\mathbf{i} \right) \cdot {}^{C'}_{C}\vec{\mathbf{t}} > 0 \tag{4.28}$$

to determine the sign to return a unique solution. When running this algorithm in a RANSAC based hypothesize-and-test procedure one can skip potential degenerate samples identified cross-product magnitudes which are two small. Additionally, if one uses the efficient error function defined in equation (4.18) one can skip the forward and backward projection steps, resulting in a very fast RANSAC iteration, and as the model size is only two very few iterations are required.

This relative pose estimation method using IMU priors gives us our initial point cloud of landmarks that initialises our map. The initial scale is defined by the baseline magnitude $\left\| ^{C'}_{C}\vec{\mathbf{t}} \right\|$ between the two camera frames used. It is then propagated through successive absolute pose estimates against this pointcloud. The baseline is not determinable so we implemented a few heuristics to estimate it in subsection 6.6.4, such as setting the magnitude to an estimated altitude difference obtained from a barometer.

Absolute Pose Estimation using 2D-3D Correspondences

The absolute pose problem consists of estimating the pose of a camera frame given n 2D-3D correspondences between bearing vectors in the camera frame and landmarks in the world frame. It is illustrated in Figure 4.30. This problem is also known as the Perspective-N-Point Problem (PnP). The minimal case is called the Perspective-3-Point(P3P) problem, which only uses $n = 3$ correspondences and return a finite number of solutions. One usually needs a 4th point to disambiguate, generally resulting in a unique solution. In the strive for computationally efficient methods the standard approach has been to solve the P3P problem in a RANSAC based hypothesize-and-test procedure to remove outliers and then solve PnP over all remaining inliers.

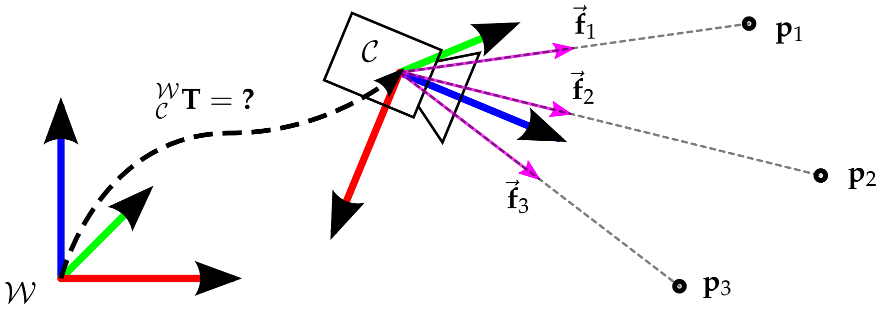


Figure 4.30: The P3P problem: estimating the camera pose, given 2D image measurements and associated 3D landmark positions.

PnP was first coined in 1981 [82] and first studied in 1841 [83]. Once again this is a well investigated problem and and a plethora of linear, non-linear and iterative solutions exist A more recent very robust solution, provided by [84] in 2003 has become popular. It is beyond the scope of this thesis to compare and contrast many of the existing solutions so we refer the reader to [20] where a large collection of related work is given. Furthermore, [20] provides an additional closed form state of the art P3P solver that provides comparable accuracy and precision to [84] at a substantially lower computational cost as it enables the computation of the position of the camera in a single stage.

Using IMU Rotation Priors In [73] the P3P solution is extended to use relative rotation priors given by equation (4.21) resulting in another two point algorithm that returns a unique translation. The solution is non-trival out of scope of this thesis. We refer the reader to [73] for the derivation. Similar to the relative case, this two point solution is execute Equation 4.18 is used as the error function avoiding projections onto the image plane.

4.1.11 Global Optimisation - Bundle Adjustment

Until now we have described methods that operate on correspondence pairs. We initiate a pair of camera poses and triangulated landmarks in the relative estimation method described in section 4.1.10. Until now, the landmarks were only optimised over correspondences from two views only. Once we have our initial landmarks, the absolute pose estimation methods described in section 4.1.10 computes a pose relative to the landmarks that were associated via the current keyframe only. While this might be desirable for performance reasons, better results can be achieved if we consider correspondences over multiple frames. In practise, a landmark is usually tracked over over multiple views, therefore we can achieve a more optimal result by considering all observations of each landmark that have more than one correspondence. Optionally, one can also optimise over the camera intrinsics, but we assume the calibrated case so this is not required. This constitutes a joint optimisation of all 3D point and camera frame poses resulting in a large non-linear optimisation problem usually referred to as bundle-adjustment. The name arose from bundles of light rays originating from each landmark, converging onto each camera's optical center, which are adjusted optimally with respect to the landmarks.

Bundle adjustment is especially useful for handling loop closure. If one can recognise a place one has previously visited, (by using a place recogniser for example, such as those based on *Bag of Word* (BoW) approaches) but one has accumulated enough drift that the camera poses do not converge, bundle adjustment can smoothly 'close the loop' if one can provide landmark correspondences simultaneously to frames at the beginning and end of the loop.

Levenberg Marquardt [85], also known as damped least-squares (DLS), has proven to be one of the more successful non-linear least-squares optimisers due to its use of an effective damping strategy that enabled it to converge quickly from a wide range of initial guesses. Depending on the number of landmarks, camera poses and initial estimates, the optimisation can quickly become computationally expensive. Many optimisations exist, such as exploiting the sparse block structure arising from the the lack of correspondences among parameters for different 3D points and cameras. We refer the reader to [80, 86, 87] for more information. A few open source bundle adjustment libraries exist such as g2o[88] (which we use in our framework), SBA [89], etc.

For our purposes, we employ windowed bundle adjustment - only optimising over a fixed amount of keyframes at most, resulting in constant complexity. We trigger bundle adjustment when adding a new keyframe to the map.

4.2 External Pose Estimation

To facilitate debugging and to have a source of ground truth information, we needed an accurate way to determine the 6D pose of the Crazyflie during flight relative to some fixed world frame. This is usually easier than ego-motion estimation but only works while the observed target is in the field of view of the external system. As this is rarely the case outside of research labs, external tracking is usually not a viable option. Nevertheless it serves as a good debugging and prototyping tool and can be used to validate ego-motion estimation. We implemented two such methods based on different sensors.

4.2.1 Motion Capture System

We used a Qualsys Motion Capture Studio to track infra-red markers attached to the quadrotor with 12 ceiling mounted cameras. All the cameras are connected to a central computer which fuses all the detections and estimates poses of pre-defined marker configurations. Depending on the exposure time and calibration quality the motion capture system can estimate the pose of these marker configurations at up to 500 Hz with mm accuracy. Implementation details are given in subsection 6.5.1.

4.2.2 Kinect

As one does not always have access to a professional motion capture studio, we implemented a way to detect and estimate the pose of the Crazyflie using a Kinect. This allows only user to set-up a small external tracking environment at home for very little cost. However, the accuracy and tracking area is far inferior to a proper motion capture studio. The general set-up is illustrated in Figure 4.31.

The kinect can estimate 640 $\times$480 resolution depth images at 30 Hz using structured light. Therefore, if one knows which pixels on the incoming depth images correspond to the quadrotor, one can determine not only the bearing but also the depth giving us a 3d position ${}^{\mathcal{K}}\mathbf{Q}$ of the quadrotor relative to the sensor We do background segmentation on the incoming depth iamgaes to determine which pixels correspond to the Crazyflie, implementation details are given in subsection 6.5.2.

Usually, the projection of the Crazyflie on the Kinect imaging surface is very small as the quadrotor is so small. This means the number of the segmented pixels on the depth image (e.g. giving us a small point cloud corresponding to the Crazyflie in 3D space) is very limited and we can therefore not determine the quadrotors rotation ${}^{\mathcal{K}}_{Q}\mathbf{R}$. However, we can use the on-board attitude estimate of the Crazyflie to estimate the rotation. In theory, this would not work as the yaw component drifts, but in practise the yaw does not drift enough in the short flight time to cause any problems (see section 7.1.3) and assume there is no yaw drift. If we initially align the IMU rotation estimate with the world frame, we can estimate the pose ${}^{\mathcal{W}}_{Q}\mathbf{T}$ of the quadrotor in the world frame using a combination of the translation ${}^{\mathcal{K}}_{Q}\vec{\mathbf{t}}_{Kinect}$ estimated with the Kinect and the rotation ${}^{\mathcal{W}}_{Q}\mathbf{R}_{IMU}$ estimated from the IMU, giving:

$$
{}^{\mathcal{W}}_{Q}\mathbf{T} = \begin{pmatrix} {}^{\mathcal{W}}_{Q}\mathbf{R}_{IMU} & {}^{\mathcal{W}}_{\mathcal{K}}\mathbf{R}\,{}^{\mathcal{K}}_{Q}\vec{\mathbf{t}} \\ 0_{1\times 3} & 1 \end{pmatrix} \tag{4.29}
$$

Implementation details are covered in subsection 6.5.2 and evaluation details of using the pose estimated by the detector for control is given in subsection 7.4.2.

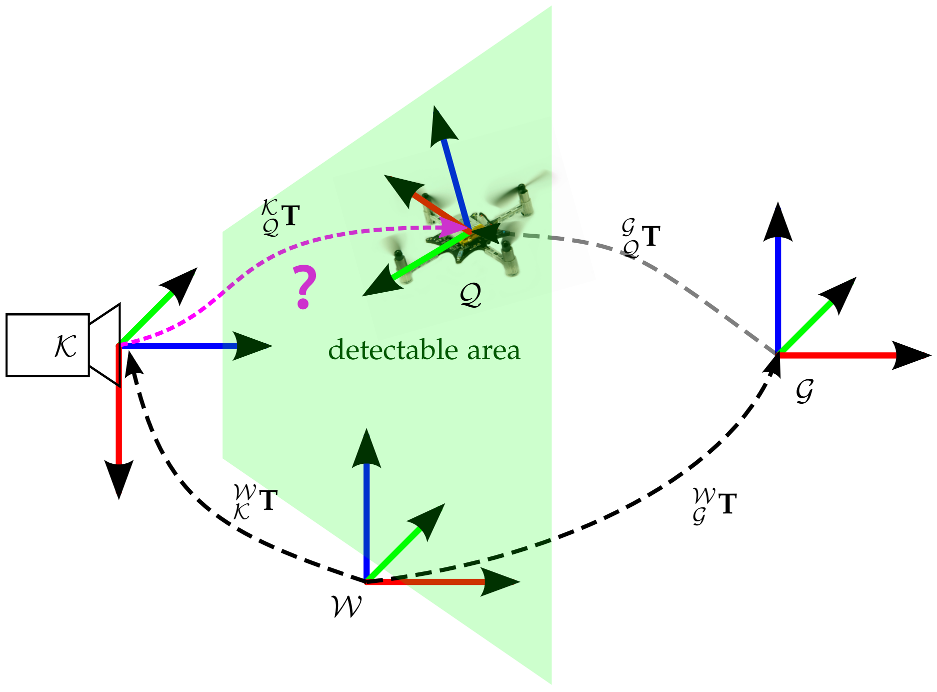

$\mathcal{K}$
$^{\mathcal{K}}_{\mathcal{Q}}\mathbf{T}$
?
$\mathcal{Q}$
$^{\mathcal{G}}_{\mathcal{Q}}\mathbf{T}$
detectable area
$\mathcal{G}$
$^{\mathcal{W}}_{\mathcal{K}}\mathbf{T}$
$^{\mathcal{W}}_{\mathcal{G}}\mathbf{T}$
$\mathcal{W}$

Figure 4.31: The Kinect is used to determine the pose of the quadrotor $^{\mathcal{K}}_{\mathcal{Q}}\mathbf{T}$. If we assume we know the rigid transform $^{\mathcal{W}}_{\mathcal{K}}\mathbf{T}$, we can compute the error between the Crazyflie pose and a goal pose $^{\mathcal{W}}_{\mathcal{G}}\mathbf{T}$ in the world frame using $^{\mathcal{G}}_{\mathcal{Q}}\mathbf{T} = {}^{\mathcal{W}}_{\mathcal{G}}\mathbf{T}^{-1}\,{}^{\mathcal{W}}_{\mathcal{K}}\mathbf{T}\,{}^{\mathcal{K}}_{\mathcal{Q}}\mathbf{T}$. This error can be used to determine the control response required to move the quadrotor to the goal.

4.3 Attitude Estimation

The Crazyflie estimates the attitude on-board in real-time at 250 Hz using gyroscope and accelerometer measurements. The rotation estimate is then fed into the attitude controller so the Crazyflie can remain level or hold target roll and pitch angles. Figure 4.32 illustrates the pipeline from sensor readings to rotation estimation.

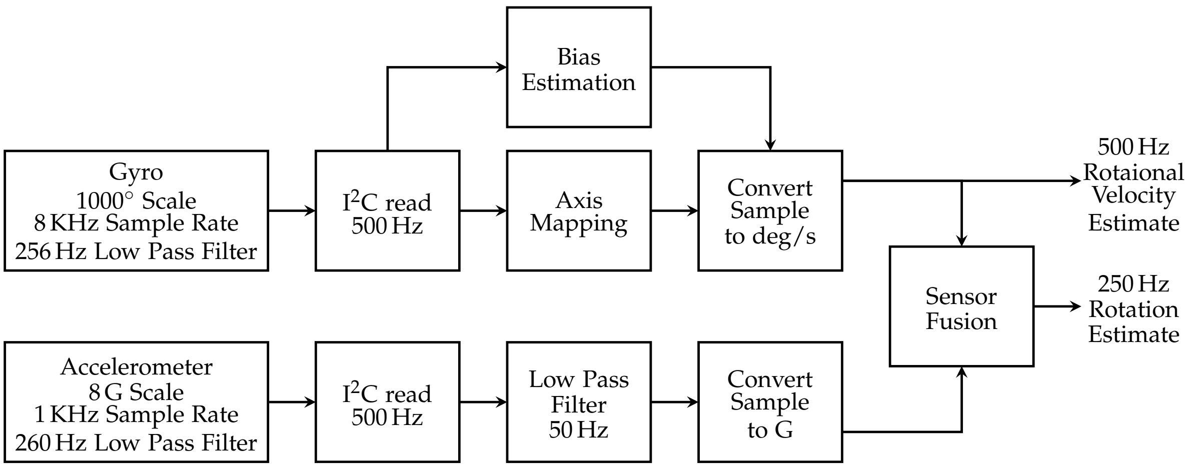


Figure 4.32: The Crazyflie samples IMU data and smooths the signals using a low pass filter. Basic yet effective bias estimation is done when the Crazyflie is an a stable state. The rotation estimation runs at 250 Hz.

When performing attitude estimation Combining data from multiple sensor types can significantly improve attitude estimation as one can combine the advantages and alleviate the disadvantages of each type. For example, gyroscope rotational velocity measurements can be integrated to produce angle estimates that are reliable in the short-term, but tend to drift in the long-term. On the other hand, accelerometers are sensitive to vibration and other non-gravity associated accelerations in the short-term, but do not drift in the long-term and thus provide angular estimates that do not continuously degrade That being said, it is a natural conclusion to combine gyroscope and accelerometer measurements allowing for angle estimates that are resistant to vibration and also immune to long-term angular drift. Note that the heading estimate is a bit of a special case: roll and pitch can be measured by a static accelerometer by looking at the direction of the gravitational pull. However, the yaw rotation axis is defined to be parallel to the gravity vector and is therefore not recoverable from the accelerometer measurements. Unless an additional sensor is added, the yaw component will drift. The Crazyflie has a magnetometer with which one could determine absolute magnetic north, thereby providing a way to estimate yaw directly. However this was not deemed necessary for this thesis as in practise the yaw drift is low enough, as shown in section 7.1.3.

Bitcraze AB opted for Madgwicks implementation of Robert Mahony's *DCM filter* in quaternion form. It is computationally inexpensive requiring only 109 scalar arithmetic operations for each filter update and thus is well suited for real-time operation on an embedded platform with limited computational capacity [90]. Furthermore, it is relatively easy to tune as it has a single adjustable parameter defined by observable system characteristics. Intuitively, the filter resembles a non-linear complementary filter that combines accelerometer measurements for low frequency attitude estimation and integrated gyroscope measurements for high frequency estimation. Note that in this configuration yaw cannot be observed and therefore the yaw estimation drifts over time. However, as mentioned in the previous paragraph, for the short flight time of the Crazyflie this is of little consequence. The filter derivation is not trivial and refer the interested reader to [90] for more details.

5 Control

Control theory studies how dynamical systems react over time when given input and how their behaviour is modified through feedback. In this chapter we will discuss how to implement a basic controller that is used to let a system (i.e. the quadrotor) reach a target state despite not knowing the relationship between the control signal and output behaviour and despite having unknown external disturbances act on the system. A controller is used twice in this thesis: (*a*) a four part position controller is used to set the desired quadrotor roll, pitch, yaw velocity and thrust percentage to move the Crazyflie to a desired 3D position and heading. The output is then fed into (*b*) three individual controllers that control the four quadrotor motors to achieve the desired roll angle, pitch angle and yaw velocity.

First we cover basic control principles, then we introduce the PID controller and explain how to tune it, and finally we explain how one can achieve attitude and position control.

5.1 Control Principles

In this section we cover basic control principles. The general goal in control theory is to control a system, commonly referred to as a *plant*, so that it reaches a (potentially changing) desired set point called the *reference*. With this in mind, one designs a *controller* that computes an *error signal* from the difference between the measured system state and the reference target state. This difference is then applied as feedback to the system input to reduce the error by bringing the actual output closer to the reference. In summary and with Figure 5.4 at hand: we wish to compute input values $\mathbf{u}(t)$ such that the measured error $\mathbf{e}(t)$ between a given target $\mathbf{w}(t)$ and measured system state $\mathbf{y}(t)$, is stably yet quickly minimised over time, while counterbalancing any unknown external disturbances. A good controller will converge to and remain at zero error very quickly and stably without oscillating around the set point.

5.2 PID Control

The proportional-integral-derivative controller (PID controller) is a specific type of controller and has historically been regarded as the best type of controller when one has no knowledge of the underlying process [91]. It emerged during the design of ship autopilots in 1910 and is used in over 90% of today's practical control systems[92].

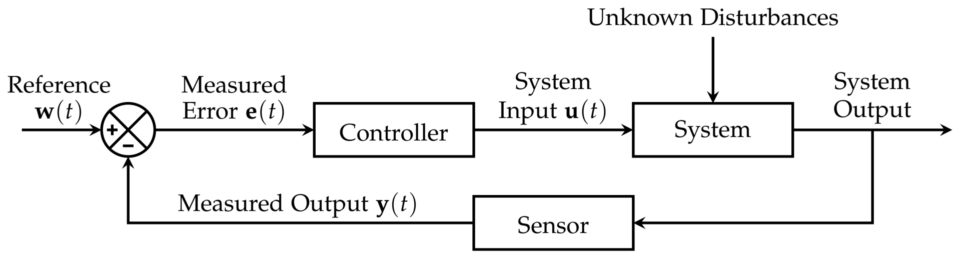

Unknown Disturbances
Reference $\mathbf{w}(t)$
Measured Error $\mathbf{e}(t)$
+
-
Controller
System Input $\mathbf{u}(t)$
System
System Output
Measured Output $\mathbf{y}(t)$
Sensor

Figure 5.1: Schmetic representation of a feedback control loop. The goal is to compute system input values $\mathbf{u}(t)$ that allow the measured error $\mathbf{e}(t)$ = $\mathbf{w}$(t) - $\mathbf{y}(t)$ to quickly yet stably converge to zero.

The PID control scheme is named after its three correcting terms which are based on current rate of change and are summed to calculate the output of the PID controller giving the system input $\mathbf{u}(t)$:

- the **proportional term** which depends on the present error $\mathbf{e}(t)$,
- the **integral term** which depends on the accumulation of past errors $\int_0^t \mathbf{e}(\tau)\,\mathrm{d}\tau$,
- the **derivative term** which depends prediction of future errors $\dot{\mathbf{e}}(t) = \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{e}(t)$

Note that some controlled systems may only need a subset of the control terms which, is achieved by setting the corresponding gains to zero. Such controllers are called P, PD or PI controllers.

Each term has its own user-tunable constant parameter which effects the magnitudes of the respective contributions, resulting in

$$u(t) = \underbrace{K_p \cdot \mathbf{e}(t)}_{\mathrm{P}} + \underbrace{K_i \cdot \int_0^t \mathbf{e}(\tau)\,\mathrm{d}\tau}_{\mathrm{I}} + \underbrace{K_d \cdot \dot{\mathbf{e}}(t)}_{\mathrm{D}} \tag{5.1}$$

where K_p, K_i, and K_d are the respective gains - tuning parameters, e the error, t the instantaneous time and τ, a variable of integration that takes on values from time 0 to the present t. As we are dealing with a discrete system, we approximate the integral and derivative errors by numeric integration and differentiation:

$$\int_0^t \mathbf{e}(\tau)\,\mathrm{d}\tau, \approx \sum_{\tau=0}^{t} \mathbf{e}(\tau) \qquad \frac{\mathrm{d}}{\mathrm{d}t}\mathbf{e}(t) \approx \frac{\mathbf{e}(t) - \mathbf{e}(t-\delta_t)}{\delta_t} \tag{5.2}$$

A typical block diagram of a PID controller is shown in Figure 5.2.

5.2.1 PID Gain Tuning

The three gains of a PID controller are simultaneously its strength and weakness: they can be tuned to provide a control action designed for a wide range of specific process requirements on one hand, but on the other hand finding good values for the gains is

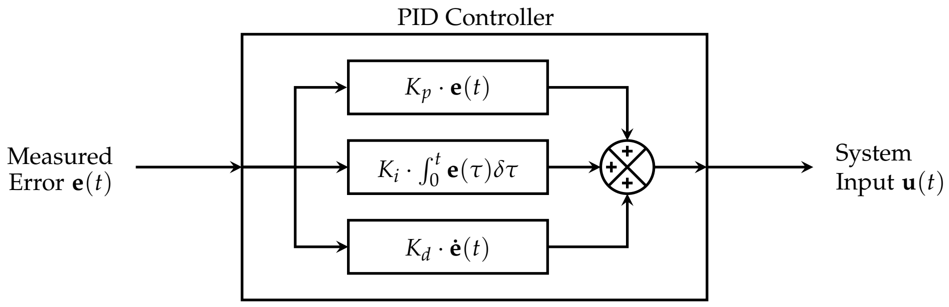

PID Controller
$K_p \cdot \mathbf{e}(t)$
$K_i \cdot \int_0^t \mathbf{e}(\tau)\delta\tau$
$K_d \cdot \dot{\mathbf{e}}(t)$
Measured Error $\mathbf{e}(t)$
System Input $\mathbf{u}(t)$
+
+
+

Figure 5.2: Schematic representation of a **P**roportional-**I**ntegral-**D**erivative (PID) controller. The controller is tuned by manipulating the K_p,K_i and K_d gains and calculates the system input values $\mathbf{u}(t)$ by summing the individual parts.

not trivial; one must usually resort to experimentally hand tuning them through trial and error, despite their conceptually intuitive nature. This usually involves analysing the system output as a function of time after subjecting the system to a step change in input. Experience certainly helps, heuristic methods exist to determine decent initial values (such as the ZieglerNichols method [93]), and software tools are available that can aid the user too [92].

Note there are not optimal gains as a PID controller does not in any way guarantee optimal control of the system or system stability, which can be intuitively explained: as PID control is only a *feedback system* with constant parameters with the assumption that one has no direct knowledge of the process it is controlling, the overall performance is a *reactive* compromise. If one would have knowledge about the system, one could incorporate feed-forward control into the loop and improve performance. Alternative feed-forward controller designs exist, such as the linear-quadratic regulator (LQRs) that requires a model of the underlying system.

Evaluation Metrics

Before we start tuning a PID controller, we need to define our goals and a way to evaluate them Ultimately, one desires a controller that is stable and behaves optimally:

- Stability is the most basic requirement and means the controller eventually converges, e.g. there is no unbounded oscillation. An unstable controller's output diverges (possibly without oscillation), and is limited only by the system itself. Excess gain causes instability, which is often caused by the presence of feedback lag (e.g. if the time to measure the system state is large).
- Optimum behaviour with regard to external disturbances or set point changes is application specific. For example, an application might require very fast responses at the expense of stability while another might desire an energy efficient way to reach a new reference value. Often multiple conflicting objectives such as short response times and high stability are desired, which is where PID tuning becomes difficult and trade-offs occur. Generally one distinguishes between:

- **regulation**, also called disturbance rejection, which refers to how well a controller can keep the controlled variable at a given reference point while it is being subjected to external disturbances, and
- **command tracking**, which refers to how well the controller can let the controlled variable follow set point changes.

To quantise the above, we characterise the response of a controller to a step reference change using the following metrics:

- **Rise Time**, which is the time required for $\mathbf{e}(t)$ to reach less than 10% of the initial error at the time of reference change,
- **Overshoot**, which is the maximum magnitude of $\mathbf{e}(t)$ after the set point has been reached,
- **Convergence Time**, which is the time needed for the system to converge to its steady state, and
- **Steady State Error**, which is the magnitude of $\mathbf{e}(t)$ after the system has converged to its steady state.

How the three weighted terms of a PID controller effect the metrics above is explained in the following subsections, with a summary given in Table 5.1.

The Proportional Gain

The proportional term is usually dominant and is responsible for directly reducing the error: a larger error directly produces a stronger, corrective control signal, with magnitude given by the proportional gain. If the proportional gain is too high, the system can become unstable, leading to overshoot and oscillations. A low gain results in a less sensitive controller and setting it too low would result in a control signal not strong enough to counteract disturbances. Figure 5.3 demonstrates the behaviour of a system after a step reference change for multiple PID-Controllers with different K_p gains.

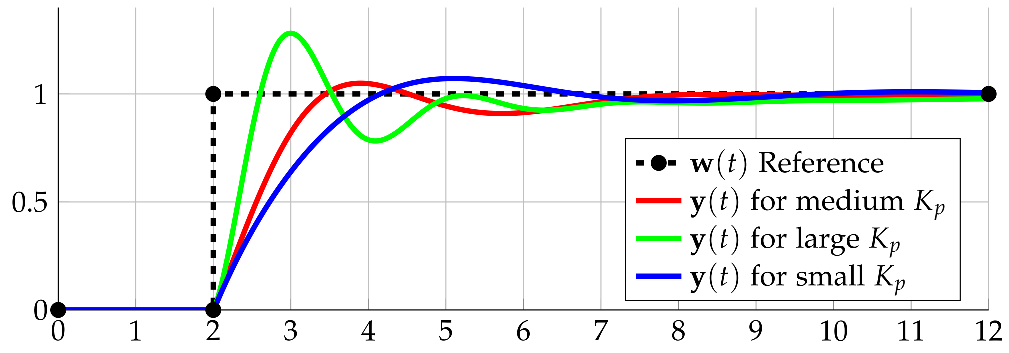


Figure 5.3: The influence of different K_p gains. Large gains lead to a faster rise time, overshoot and oscillations.

The Integral Gain

The contribution of the integral component is governed by the integral gain K_i. The integral term is proportional to both the magnitude and duration of the error. It integrates the error over time and will thereby give the accumulated offset that should have been previously corrected. Residual steady-state errors of biased systems are eliminated as the integral term will increasingly compensate for the bias until the error is zero. However, as the integral term responds to accumulated past error, it reacts slowly to current changes and causes the present value to overshoot the target value, causing low frequency oscilations. Figure 5.5 shows an example of how the integral term compensates for steady state error by comparing a PD and PID controller.

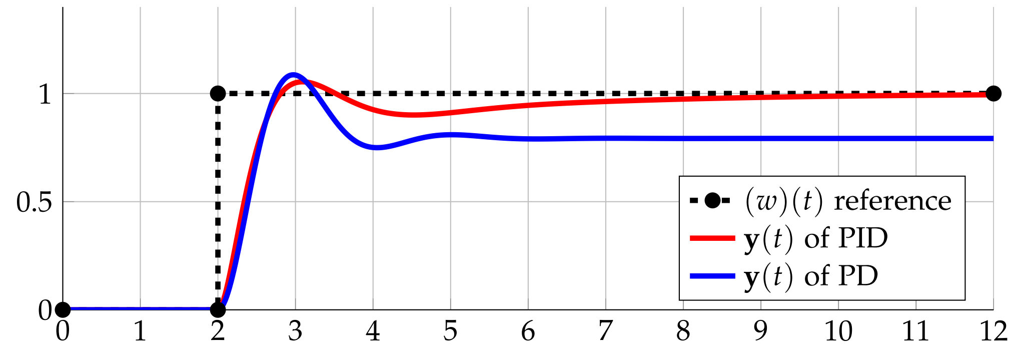


Figure 5.4: Notice that the PD controller never reaches the reference signal. Adding an integral term eliminates the steady state error.

The Derivative Gain

The derivative gain determines the magnitude of the contribution of the derivative term. This term improves convergence time and helps reduce over shoot and eliminate oscillations by anticipating the future behaviour of the error: the faster the error reduces, the more it contributes to slowing down this rate of change, thus dampening the system. However, the derivative response is highly sensitive to noise in the measured output signal $\mathbf{y}(t)$. If this sensor feedback signal is noisy, the derivative component can make the control system unstable. The effect of the derivative gain K_d is shown in Figure 5.5, where a too high value leads to an over dampened system and a too value an under dampened system.

Manual Tuning

Much theory and work has been done concerning PID tuning and we restrict ourselves to the basics, only giving an intuitive, simple guide to obtaining usable gains. We refer the interested reader to [94] for a short tutorial and [95] for a more in depth analysis regarding PID tuning methods and tools.

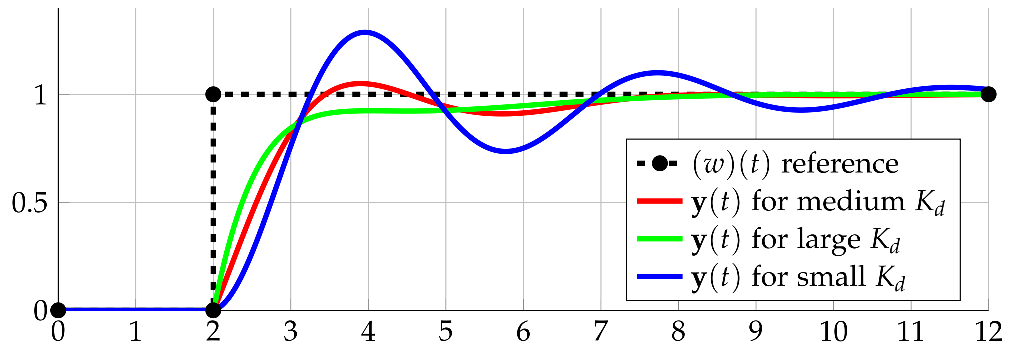


Figure 5.5: Three PID controllers with different K_d gains. Notice how a larger derivative gain reduces oscillations but increases rise time.

To manually tune an online PID controller, one can perform the following steps:

1. focus on the K_p gain fist by setting K_i and K_d to zero and K_p to a small value.
2. while observing the error $\mathbf{e}(t)$, introduce a step in the reference signal $\mathbf{w}(t)$.
3. increase K_p to reduce the rise time until the controller becomes unstable, i.e. does not converge and begins to oscillate.
4. increase K_d, to reduce the overshoot and dampen the oscillations. A responsive PID loop usually overshoots some what before converging.
5. if required, increase K_i until any steady state error is corrected in sufficient time for the process.
   a) However, be conservative, as too much K_i will cause low frequency oscillations.
6. depending on the requirements, repeat the above from point 2. and
   a) either be more conservative on the gains for a more stable system,
   b) or be more aggressive for a more responsive system.
7. if the system oscillates at a low frequency, this could be caused by too much K_i.
8. if the system oscillates at a high frequency, this could be caused by too much K_p and not enough K_d.

Table 5.1: Manaul PID Tuning Guidelines [94][95]. Effects of independently increasing a parameter.

Gain	**Rise Time**	**Overshoot**	**Convergence Time**	**Steady-State Error**	**Stability**
K_p	decrease	increase	small change	decrease	degrade
K_i	decrease	increase	increase	eliminate	degrade
K_d	increase	decrease	decrease	none	Improve if K_d small

5.3 Crazyflie Attitude Control

The Crazyflie performs on-board attitude control so that it can hold desired roll and pitch angles and maintain a desired yaw velocity.

PID controllers can be used together to yield better dynamic performance by cascading them: one can arrange two PID controllers so that the first one controls the setpoint of the next one. The latter controller is set to operate at a higher frequency so that it has time to manipulate the system to reach the it's own desired set point. This can be seen as having two control loops, a slower *outer* control loop, controlling a nested, *inner* loop. For example, on the Crazyflie the outer loop controller controls the roll and pitch angles by feeding the inner loop controller desired rotational velocities.

The right block of Figure 5.6 shows the two on-board nested control loops in action. Each degree of freedom (roll, pitch and yaw velocity) have their own controller, and as the roll and pitch cases are analogous we only consider the roll case. The desired roll $\mathbf{w}_\alpha(t)$ angle is given at 100 Hz, and the error $\mathbf{e}_\alpha(t)$ between it and the estimated angle $\mathbf{y}_\alpha(t)$ determined by the on-board attitude estimation (see Figure 4.32) is fed into the attitude controller at 250 Hz. This feeds the angular rate controller a reference rotational velocity $\mathbf{w}_{\dot\alpha}(t)$ which it attempts to maintain at 500 Hz, using the gyro measurements $\mathbf{y}_{\dot\alpha}(t)$ directly to compute the error $\mathbf{e}_{\dot\alpha}(t)$. Note that the yaw velocity is directly controlled, not the yaw angle. Therefore, the yaw velocity reference signal $\mathbf{w}_{\dot\gamma}(t)$ by-passes the outer control loop. The resulting roll, pitch and yaw velocity system input signals $\mathbf{u}_{\dot\alpha,\dot\beta,\dot\gamma}(t)$ are then used to distribute the available thrust to the four motors using

$$Motor_1 = thrust - \mathbf{u}_{\dot\alpha} + \mathbf{u}_{\dot\beta} + \mathbf{u}_{\dot\gamma} \tag{5.3}$$

$$Motor_2 = thrust - \mathbf{u}_{\dot\alpha} - \mathbf{u}_{\dot\beta} - \mathbf{u}_{\dot\gamma} \tag{5.4}$$

$$Motor_3 = thrust + \mathbf{u}_{\dot\alpha} - \mathbf{u}_{\dot\beta} + \mathbf{u}_{\dot\gamma} \tag{5.5}$$

$$Motor_4 = thrust + \mathbf{u}_{\dot\alpha} + \mathbf{u}_{\dot\beta} - \mathbf{u}_{\dot\gamma} \tag{5.6}$$

which can be derived from subsection 3.1.3.

5.4 Position Control

Positioning the Crazyflie in 3D space is one of the major goals of this thesis. To accomplish this, we need to be able to compute the translational error $\mathbf{e}_{x,y,z}$/heading error $\mathbf{e}_\gamma$ between the measured position $\mathbf{y}_{x,y,z}$/heading $\mathbf{y}_\gamma$ and target position $\mathbf{w}_{x,y,z}$/heading $\mathbf{w}_\gamma$. The majority of this thesis is specifically about 'measuring' this heading and position - either externally or using an on-board camera. See chapter 4 for details.

To move the Crazyflie to a desired position and heading, we add an additional outer control loop (i.e. an initial, additional cascade) that runs at the maximum sensing rate of the used sensor (see Table 4.1). This controller actually consists of four controllers in parallel, each controlling one degree of freedom:

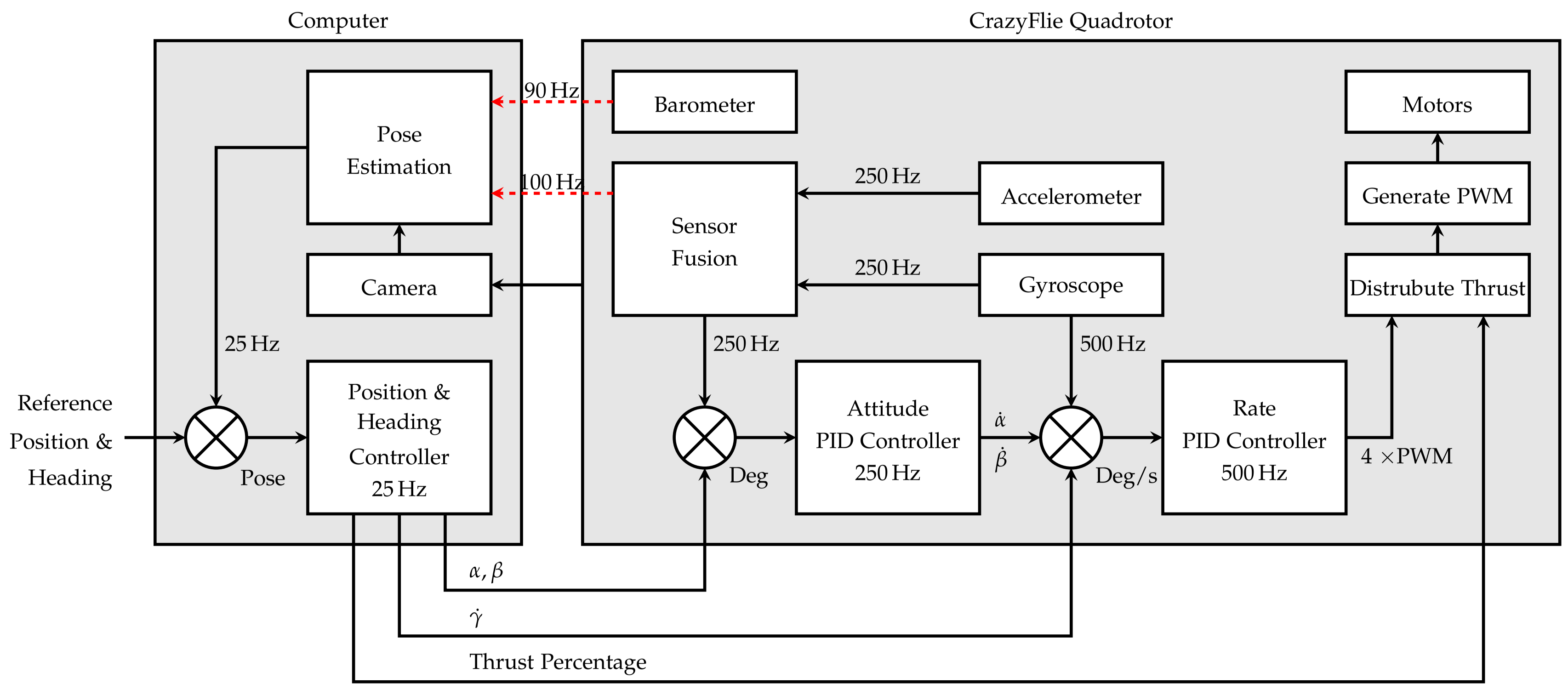


Figure 5.6: Triple cascaded PID controller ultimately controlling the position and heading of the Crazyflie. From inside out: An angular rate controller controls the distribution of thrust to the motors, using the gyroscope to compute the error between the measured angular velocity and the desired angular rates. These desired angular rates originate from the attitude controller, which compares the estimated attitude to the target roll and pitch angles from the position controller. The position controller uses an on-board or external camera to estimate the pose of the Crazyflie, optionally being supported but the on-board pressure and attitude measurements. Additionally, the position controller manipulates the thrust and yawing rate to achieve the desired position and heading.

- the roll angle α - to control lateral motion,
- the pitch angle β - to control forward and backward motion,
- the yaw velocity $\dot{\gamma}$ - to control the heading, and
- the thrust - to control the altitude.

To relate distance to angles, we simply assume a linear relationship. Using a motion model to directly compute which inputs are required to move the Crazyflie to the target with greater efficiency are not in scope of this thesis, as a PID control proved sufficient. Any bias in the Crazyflie, caused by unbalanced motors for example, is compensated for by the attitude control, and therefore the integral term of the position controller used was not required, resulting in a PD-controller.

To control the thrust, we had to add a pole to altitude controller. The Crazyflie requires a certain base thrust to hover and small changes in this result in an altitude increase or decrease. Therefore, a constant output was added to the thrust controller system output. As this bias greatly varies with the condition and current discharge of the battery, a strong integral term was required. See subsection 7.1.1 for details on how the battery discharge affected the required thrust bias.

6 Implementation

In this chapter we will look at the implementation of the various software components, how these components communicate with one another and how the hardware is interfaced, as summarised in Figure 6.1.

The Robot Operating System (ROS) is used as the main 'plumbing', essentially glueing the different software modules and hardware drivers together. For a brief overview of ROS and explanation why it was chosen as an underlying framework, we refer you to section 6.1.

The joystick hardware and drivers are examined in section 6.2 and various camera drivers are compared in section 6.3. A custom driver was written to interface with the Crazyflie, with details provided in section 6.4. Details on the implementation of basic external pose estimation using a motion capture studio as well as a Kinect are provided in section 6.5. Implementation specific notes on the proposed VI-SLAM system and Control Module are covered in 6.6 and 6.7 respectively.

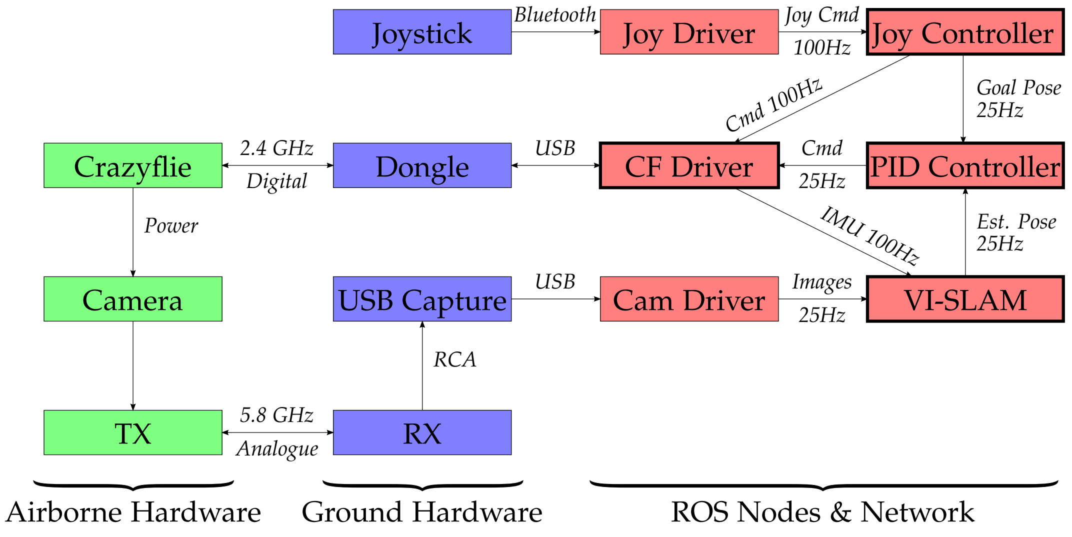


Figure 6.1: Overview of the various hardware and software components used.

6.1 Robot Operating System

The Robot Operating System (ROS)[96] is a flexible, open source (BSD) framework composed of a meta-operating system, a collection of tools, many libraries and defined conventions that facilitates collaborative robotics software development. It provides hardware abstraction, device drivers, a build system, package management, inter-process communication and an implementation of commonly used algorithms and functionality.

ROS was chosen as an underlying framework for numerous reasons. It strongly facilitates a modular design structure as it provides robust and easy to set up communication between different processes, which allows the software to be separated into various standalone components. Many of these components were already implemented, such as the ones required to interface with USB cameras or other input hardware such as joysticks. As each component can be its own process, they can be written in different programming languages. This allowed us to write performance oriented software in C++ and higher level software in Python. Furthermore, it provides many useful tools, such as those for 2D and 3D data visualisation and playing back data recorded during a previous runtime.

6.1.1 ROS Concepts

ROS defines three main levels of concepts: (*a*) the Filesystem level (using Package Resource Names), (*b*) the Computation Graph level (using Graph Resource Names) and (*c*) the Community level. Each will briefly be discussed below.

Filesystem level

This level deals with concepts that cover on disk ROS resources. ROS allows software to be broken up into *packages* and *meta-packages*. Packages are the most atomic build and release item and contains anything that is usefully organised together, such as ROS run-time processes, libraries, datasets, configuration files, etc. Meta-packages serve to represent a group of related packages. For example, a meta-package called *joystick_drivers*[1] contains a set of packages that interface with various joystick devices: *joy*[2] (for generic joysticks), *ps3joy*[3] (for Sony PS3 Controllers) and *wiimote*[4] (for Nintendo Wiimotes). Each package must contain a *manifest* which provides package meta-data, such as it's name, version, license, dependencies, etc. Packages also contain definitions for the messages and services that they provide.

The software accompanying this thesis is divided into two packages: (*a*) *crazyflieROS*[5], which is used to interface with the Crazyflie covered in section 6.4 and (*b*) *ollieRosTools*[6], which contains implements a monocular inertial SLAM system covered in section 6.6.

Computation Graph level

The *Computation Graph* is the peer-to-peer network of concurrently running ROS processes. This graph consists of various entries described below.

Nodes

[1]`http://wiki.ros.org/joystick_drivers`
[2]`http://wiki.ros.org/joy`
[3]`http://wiki.ros.org/ps3joy`
[4]`http://wiki.ros.org/wiimote`
[5]`https://github.com/omwdunkley/crazyflieROS`
[6]`https://github.com/omwdunkley/ollieRosTools`

Nodes are individual processes that perform computation and are written using a ROS client library such as roscpp for C++ and rospy for Python. For example, the Crazyflie driver is a node that exposes its functionality to the ROS network.

Messages
Nodes communicate with each other by sending *messages* over channels called *topics*. Messages are user definable data structures called *msgs* which are much like C structs. Common message types (e.g. for sending images or IMU data) are already defined and additional ones can be specified.

Topics
The third etc ... A *topic* is the name of a communication channel over which messages are transported. They fulfill the many-to-many one-way transport paradigm with publish/subscribe semantics. A node receives messages by *subscribing* to a topic, after which messages published on this topic can be handled by a callback the node implements. Nodes push out *messages* on a topic by *publishing* to one. Each *topic* may have multiple, concurrent publishers and subscribers. Topics should be used for continuous data streams, such as an image stream from a camera or IMU sensor data from a quadrotor.

Services
Services implement the request-reply communication paradigm and used like Remote Procedure Calls. Nodes can provide *services* under a name with a user-definable data structure called *srvs*. A srv is a pair of message structures, one for the request and one for the reply. A client node then sends a request srv to the service name and awaits a reply. Note that requesting a service results in a blocking call, so client code execution cannot continue until the service provider responds. Therefore, services should only be used for quick remote procedure calls that terminate quickly, such as querying a node state or setting some parameters in another node.

Paramter Server
A blackboard communication architecture is implemented using the *Parameter Server*, which serves as the blackboard. It is currently part of the *Master* and therefore only one at a time can exist. Nodes can anonymously store and retrieve data by key on the Parameter Server. This is mainly used to store and look up global configuration options. For example, we store camera model parameters, an enumeration to select the desired joystick/camera, etc.

Master
The ROS *Master* provides a central name registration and look up to the rest of the ROS Computation Graph. Nodes communicate with the Master to report their service and message registration information. Without the ROS Master, nodes would not be able to find each other and exchange messages or invoke services.

Bags
ROS *Bags* and their associated tools provide a very easy way to record, store and play back ROS messages. One could for example record a quadrotors's IMU

data and camera feed and develop nodes that process this information. One can then play the bag file (even at different rates, everything is timestamped) and test, debug and evaluate the nodes performance on exactly the same input data multiple times without needing to access the hardware. Nodes do not distinguish if the incoming data is from a bag file or the real robot, it appears the same way.

Community level

The ROS Community Level concepts are defined ROS resources, facilitating software and knowledge exchange between separate communities. *Distributions* are collections of meta-packages that ensure all dependencies and meta-packages are compatible with eachother thanks to strict versioning. The software for this thesis is based on ROS Hydro Medusa, released 04.09.2013.
Furthermore, the ROS Community level consists of *repositories* that contain software components from various institutions, the *ROS Wiki*[7] which is community driven and provides the main source of documentation, a questions and answers website called *ROS Answers*[8], a bug ticketing system [9] and a blog [10].

6.1.2 ROS Tools

ROS provides many GUI and command line tools that aid the developer in building, maintaining, visualising, configuring and recording complex systems during or after runtime. For a complete listing, please look at the ROS tools wiki entry [11]. A few of the tools commonly used in this thesis are highlighted below.

rqt_graph

ROS computation graph visualisation.

rqt_plot

2D plotting - used to visualise all the data coming in from the Crazyflie such as thrust, IMU and barometric data, voltages, etc. Additionally, the individual PID control signals can be observed which was extremely useful during online tuning.

RViz

3D plotting - used to visualise keyframes with 6D poses, the current pose estimate with the previously flown trajectory, 3D landmarks and observations and the current goal pose. Additionally, RVIZ can be configured to show the raw image stream from the camera, point cloud from the kinect and a debug image stream from the VI-SLAM module.

rqt_reconfigure

Runtime parameter changes - all the settings that can be changed during run-time

[7]`http://wiki.ros.org/Documentation`
[8]urlhttp://answers.ros.org/questions/
[9]`http://wiki.ros.org/Tickets`
[10]`http://www.willowgarage.com/blog`
[11]`http://wiki.ros.org/Tools`

are exposed to this node, which generates a GUI to easily manipulate the available parameters. For example, the user can switch out the detector type, change the number of RANSAC iterations, alter the size of the bundle adjustment window, etc.

Bags
Recording and playing back sensor data, extremely useful for debugging without the quadrotor at hand.

Launch Files
Easily start multiple nodes and their individual parameters with a single command.

6.2 Joystick Driver

The joystick driver is used to send control commands to the Crazyflie. As an input device we opted to use a Sony PS3 game pad controller. It communicates with a computer via USB or Bluetooh and is easy to set up thanks to the availability of the *ps3joy*[12] ROS driver of the *joystick_drivers*[13] package. We configured the ps3joy node to send the current controller axes and button values at a fixed rate of 100 Hz. Our joystick driver then handles these messages in a callback and forwards the resulting commands to the Crazyflie driver, outlined in section 6.4.

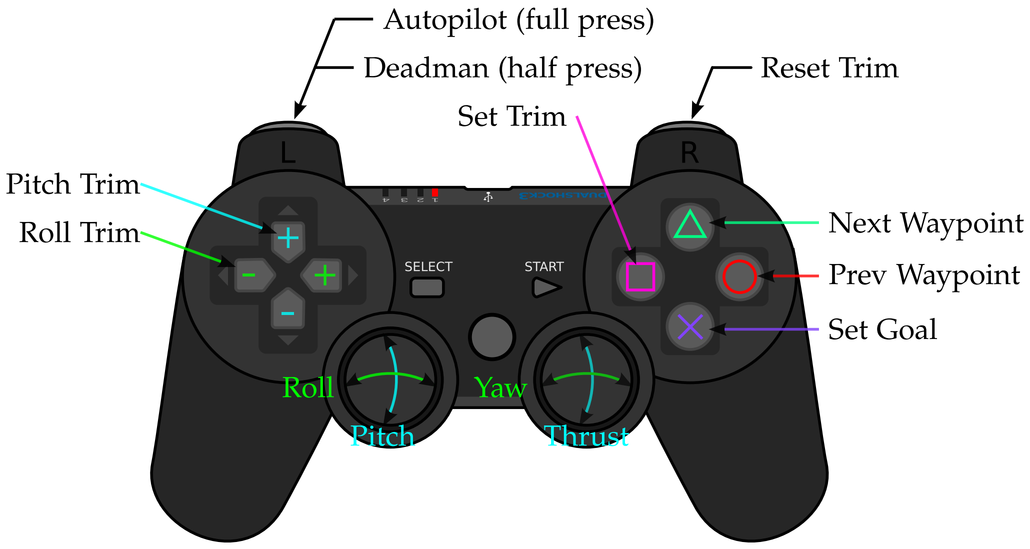


Figure 6.2: The input device used to control the Crazyflie

Four modes of operation exist, which determine what the input buttons and axes do. Refer to Figure 6.2 for an illustration of the game pad with labeled axes and buttons.

Fail Safe This is the default mode and blocks all user input and is activated when the user releases the dead man switch (L1).

[12]http://wiki.ros.org/ps3joy
[13]http://wiki.ros.org/http://wiki.ros.org/joystick_drivers

Manual flight mode To activate this mode, the pilot must press and hold the dead man button (L1) half way. This then enables the pilot to directly control the desired roll and pitch angles, the desired yaw rate and thrust percentage. Some parameters exist to scale the joystick input to angles, such as setting the maximum desired roll/pitch angle and maximum thrust. The node also has some functions that can aid a novice pilot, such as the ability to limit the speed at which thrust can be reduced. This greatly helps novice pilots, as they often kill the throttle when the Crazyflie increases altitude (which cause instability as the Crazyflie has no thrust left to self stabilise with) only to then react too slowly to increase the throttle as the quadrotor begins to plummet into the ground, resulting in an unstable, vertical oscillatory flight path. With the thrust reduction limiting enabled, the Crazyflie still has enough power to self-stabilise and prevent vertical oscillatory behaviours caused.

Altitude hold mode This mode is activated by depressing the dead man switch (L1) all the way. The crazyflie will attempt to maintain its current altitude based from barometric measurements and the pilot does not need to manipulate the throttle. The throttle stick now increases/decreases the target altitude.

Autonomous flight As above, this mode is activated by depressing the dead man switch (L1) all the way and will activate if a a valid 6D pose can be estimated using an on-board or external camera, as described in chapter 4. The Crazyflie will automatically fly to and hold a goal pose (way-point), which the two analogue controller sticks now directly control. The pilot can also cycle between way-points with the triangle and circle buttons.

Trimming the Crazyflie can help increase the attitude stability in case the motors or propellers are not well balanced. The on-board attitude controller will eventually adapt to these constant biases (thanks to a modest integral gain), but manually adding a constant offset can help the controller converge faster. The trim can be reset any time using the R1 button. Two methods can be used to set the roll and pitch trim. The pilot can either

- manually increase or descrease the trim using the D-Pad, or
- hover the Crazyflie manually (using the analogue stick to control roll/pitch keeping the quadrotor still) and then set the trim to the current analogue stick values by pressing the square.

The latter can be successively repeated if required. Trim values can directly be written to the firmware with the supplied flashing utility so that the values remain between power cycles.

6.3 Camera Node

In order to process the analogue image from the Crazyflie, we digitise the image with a Hauppauge USB-Live2. This exposes the incoming images to the Linux kernel as a video device, which can then be accessed by VLC for visual inspection or by various ROS nodes which expose the image stream to the ROS network. To determine which ROS image capture node to use, we devised a way to measure the input lag. The evaluation results are summarised in section 7.1.4.

6.3.1 Measuring Input Lag

We are interested in measuring the time between acquiring an image (measuring the light on the camera's CMOS chip) and receiving the image in a ROS node. This includes

- the capture time (camera induced lag),
- wireless transmission (transmitter/receiver induced lag),
- analogue to digital conversion (Hauppage USB-Live2 induced lag),
- accessing the digital image (kernel/V4L2 induced lag),
- publishing it to the ROS network (ROS capture node induced lag) and
- receiving it in a ROS node (ROS message overhead induced lag).

While it would be interesting to measure the duration of each sub-step, the total time needed from capture to reception suffices for our needs. From now on, we refer to this total time as image input lag. Later on when we process the incoming images, we need to estimate at which point in time they originate from to more accuaretly match an IMU measurement to each frame.

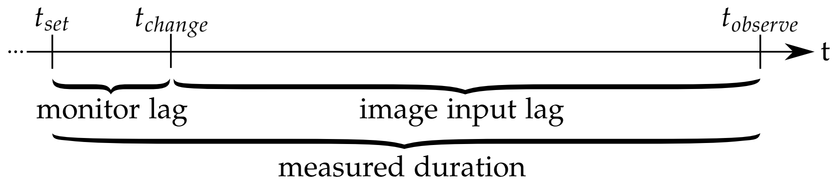


Figure 6.3: Measuring the image input lag. A timer is started the same time a monitor is instructed to change intensity (t_{set}), the monitor then actually changes intensity (t_{change}), and the timer stopped when the intensity change is detected by the camera ($t_{observe}$). Subtracting the time it takes for the monitor to change the pixel intensity from the measured time results in the image input lag.

To measure the image input a lag, we measured the time required to detect a change in the camera's field of view. A ROS node was implemented that listened to the image topic and computed the mean pixel intensity of each incoming image. An additional ROS node was then set-up to control a full screen window that would alternate between being purely white or purely black at known, but random intervals. The camera system was then setup to observe the monitor. We could then measure the time between of setting the screen intensity and an observing transition detection. Note that some additional time is in that measurement, namely the time it takes for the monitor's pixels to actually change their brightness. See Figure 6.3 for an illustration. If one knows the monitor's input lag, one can simply subtract it from the measured time to obtain the total input lag. As this is fully automated procedure, one can rerun the experiment many times.

We considered three ROS image capture nodes: (*a*) gscam [14], (*b*) usb_cam [15], and (*c*) uvc_camera [16] See section 7.1.4 for a brief evaluation of their associated lag and CPU utilisation percentage.

Unfortunately but not surprisingly, our chosen camera is of the rolling shutter type. However, modelling and compensating for it is not addressed further in this thesis and we assume the camera to employ a global shutter.

6.4 Crazyflie Node

The Crazyflie Node can be seen as a ROS driver for the Crazyflie. The node is written in Python and Qt and runs as a full blow GUI application.

The node serves multiple purposes:

Connect to the Crazyflie The node allows the user to scan for available Crazyflies, select one and initiate a connection. Once the connection has been initiated, the Table of Contents (TOC) is downloaded. The previously connected Crazyflie TOC can be cached to reduce the connection initiation time. The node can be configured to connect on start up and automatically reconnect in the case of a connection failure.

Request and receive logging data Upon connection, the Crazyflie transmits its TOC to the node. The TOC logging groups with their entries (an exposed set of variables on the firmware one can request) are then displayed to the user (see Figure 6.5b). The user can select at which rate he/she would like to receive which combination of variable values (for example float gyro.x at 100Hz), which results in *Log Configuration* object. The node then sends this Log Configuration to the Crazyflie. If accepted, the Crazyflie will start pushing data according to the contents of the configuration to the node, and if not, an appropriate error message is displayed and the GUI behaves as expected. Multiple log configurations may exist in parallel. As the GUI components are directly generated from the received TOC, the user can make changes to the Crazyflie firmware (such as removing, changing, or adding logging configurations) without the need to change update the node code. To put the above into ROS terms, the Crazyflie is a node which is configured by service call from the GUI that contains a Logging Configuration. If the call is successful, the Crazyflie will start publishing messages (logs) with the data that was requested.

Monitor the logging rates All incoming logs have associated target rates and actual rates. The GUI measures and displays the latter. Often, if too much information is being request from the Crazyflie the incoming logging rates begin to drop.

Forward all logging messages to ROS The node generates ROS msg types for each full logging configuration, instantiates a publisher of that type and forwards all received logs to the appropriate topic. Once again, this is all generated from the TOC alone

[14]`http://wiki.ros.org/gscam`
[15]`http://wiki.ros.org/usb_cam`
[16]`http://wiki.ros.org/uvc_camera`

so the user does not need to manually make or change ROS msgs or the node code after changing the firmware. For example, if the user added a logging configuration to debug the LED status, the node would generate an appropriate LEDStatus msg and forward the logs to the /crazyflie/LEDStatus topic.

View and change runtime parameters on the firmware The firmware contains variables (called parameters) which are either read-only (R) or read-write (RW). A The node displays all available variables (according to the TOC) and keeps the displayed values in sync with those on the firmware (see Figure 6.5a). Variables with write access can be changed, at which point the node requests a variable change from the Crazyflie and updates the GUI accordingly. If for some reason this fails, an appropriate error message is shown and the GUI continues to accurately reflect the true value of the variable on the firmware.

Synchronising Clocks Just as we needed to estimate the image input delay, we need to be able to estimate the delay of communication with the Crazyflie. Crazyflie logs are timestamped with the Crazyflie clock, but we need a way to relate it to the ROS (CPU) time. To estimate the clock difference Δ_{t0}, the node sends ping messages to the Crazyflie on a specific port. The Crazyflie responds to the ping with its own CPU time ${}^{Q}t_R$ (pong response). The time between the ping request at time ${}^{C}t_{ping}$ and response at time ${}^{C}t_{pong}$ gives us the round trip duration Δ_{packet}. If we then assume the outgoing and incoming durations are equally long, we can use the time-stamp of the packet arrival time ${}^{Q}t_R$ on the quadrotor in the pong response to estimate the offsets between the computer and quadrotor clocks. See Figure 6.4 for a timeline illustration.

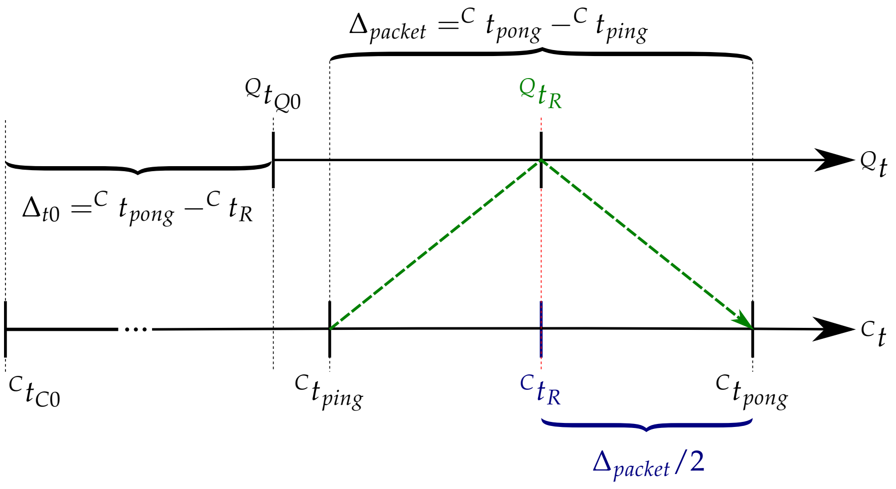


Figure 6.4: By assuming symmetric up and down latencies we can estimate ${}^{C}t_R$ (the time the ping packet was received on the Crazyflie relative to the computer clock) which we can align to ${}^{Q}t_R$ (packet receive time relative to the quadrotor clock) allowing us to estimate Δ_{t0} (offset between the two clocks)

Send control commands The node listens on the command topic to roll, pitch, yaw rate and thrust commands. These either originate from the joystick driver (section 6.2) or from the position controller node (section 6.7). The commands are forwarded to the Crazyflie (unless the disable thrust fail-safe is set in the GUI) on a specific port. The GUI allows the user to see the commands that are being sent, which can be useful for debugging (see Figure 6.5f)

Summarise the current state During field testing, we needed a quick way to visualise the status of the Crazyflie to help avoid recording badly conditioned experiments. An artificial horizon with a colour coded information overlay (red = problem, yellow = warning, green = okay) provides the user with the following information at a quick glance: (*a*) connection quality and status, (*b*) battery percentage, (*c*) battery charge state, (*d*) roll & pitch as an artificial horizon, (*e*) yaw as a compass, (*f*) calibration state, (*g*) which messages are being logged at which frequency (colour coded for target vs actual frequency), (*h*) current vertical acceleration, (*i*) motor thrust contributions, (*j*) kill switch status, (*k*) control commands, (*l*) upstream & downstream radio transmission throughput, (*m*) temperature, and (*n*) barometric pressure. This allowed us to quickly identify if something was going wrong, even while manually piloting the Crazyflie. Furthermore, the camera feed can also be displayed as an additional overlay. See Figure 6.5e for an example.

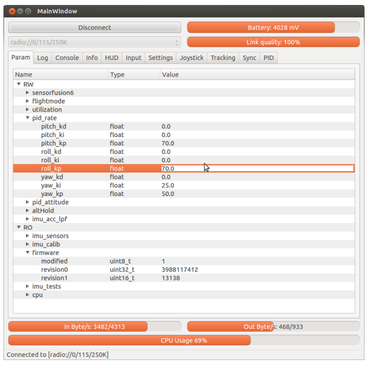
Display console information from the Crazyflie The quadrotor firmware can print messages, which are appropriately picked up by the GUI and displayed for the users convenience.

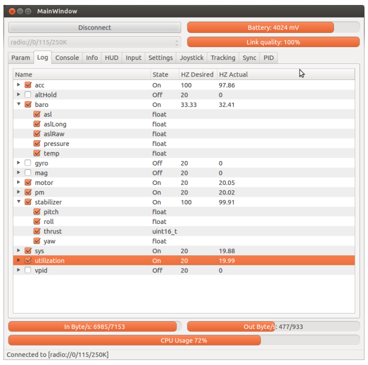
Kinect Pose Estimation The node implements the Kinect-based Crazyflie pose estimation method discussed in subsection 6.5.2 and exposes all the relevant options to the user. Figure 6.5h shows a screen shot.

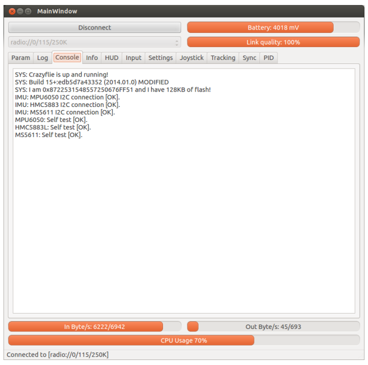
Manage various options and settings Many additional settings are grouped together which control aspects of the GUI (e.g. update/polling rates) and the Crazyflie (e.g. North yaw offset). See Figure 6.5g for a screen shot.

Barometer base station As the barometric pressure is unstable (e.g. due to people opening windows, air conditioning, etc), we allowed multiple instances of the Crazyflie Node to run in parallel, each dedicated to a single quadrotor. This allows us to set-up one quadrotor as a ground station for reference barometric readings. The nodes can communicate with each other to share this information to better estimate pressure differences caused by altitude changes only. See subsection 6.6.5 for more information.

All the settings, including logging configurations, the window size, settings, etc. are saved/read from disk using the Crazyflie URI as an index key when closing/starting the node. This allows the user to automatically recover configurations unique to each Crazyflie.

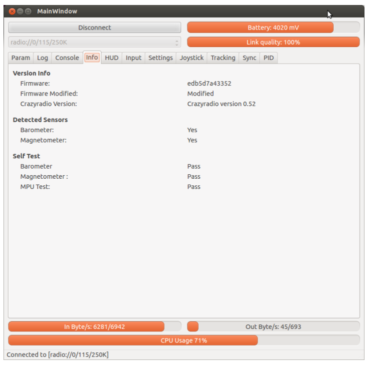

(a) Parameters

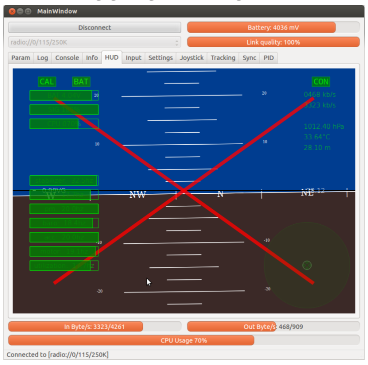

(b) Logging Configuration

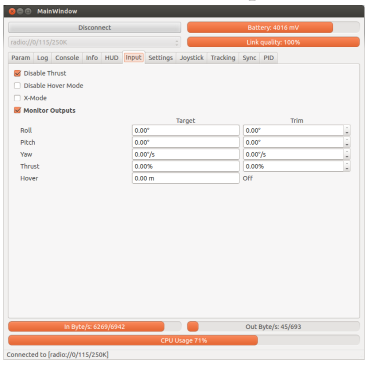


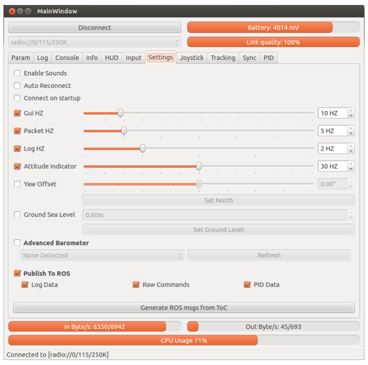
```
SYS: Crazyflie is up and running!
SYS: Build 15+:edb5d7a43352 (2014.01.0) MODIFIED
SYS: I am 0x8722531548557250676FF51 and I have 128KB of flash!
IMU: MPU6050 I2C connection [OK].
IMU: HMC5883 I2C connection [OK].
IMU: MS5611 I2C connection [OK].
MPU6050: Self test [OK].
HMC5883L: Self test [OK].
MS5611: Self test [OK].
```

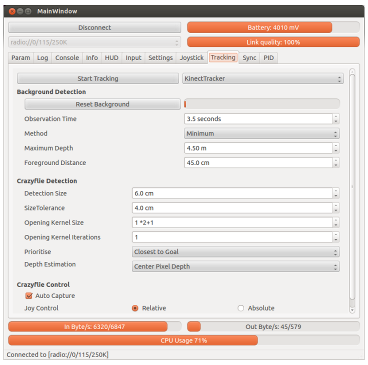
(c) Console Output


(d) Misc. Information


(e) HUD Overview


(f) Control Commands


(g) Various Settings


(h) Tracking Options

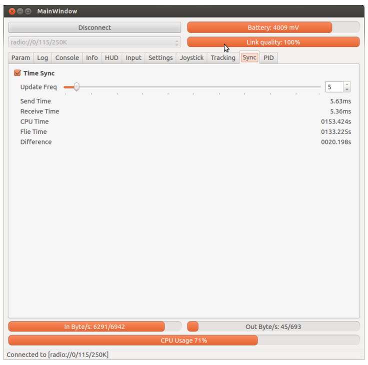

(i) Clock Synchronisation

Figure 6.5: Screenshots from the Crazyflie Node GUI

6.5 External Pose Estimation

For debugging and testing purposes we devised a way to track the pose of the Crazyflie externally. Initially, a motion capture studio was used, but as they are not freely available we implemented a kinect based tracker for home use. This section discusses both approaches briefly.

6.5.1 Qualisys Motion Capture Tracking

A motion capture system, often abbreviated with MoCap system, uses multiple infra-red cameras and illuminators to track the position of round infra-red reflective markers. These markers can be attached to objects and coordinate frames can be defined. Depending on the calibration quality of the cameras, precision around 1 mm is possible. However, in practice the calibration is not perfect and the precision suffers.

We used a Qualisys Motion Capture Studio with 12 cameras for our experiments.

Marker Placement

We used the motion tracker system to track three infra-red markers attached to the back, left and right arm of the quadrotor, about 1 cm in from the motors, under the propellers (which did not effect visibility). The position of the markers is critical for the performance of the motion track:

- We used the smallest available markers (5 mm diameter) to save weight and reduce the impact on flight behaviour.
- Usually one attempts to put the marker in a non-ambiguous, asymmetric configuration but there was not enough space to do so.
- If we put the markers any closer together (such as a configuration attached to the battery), the motion capture system would treat the cluster as a single marker and not correctly detect the quadrotor.
- If we moved the markers further out, they would often be occluded by the motors. The more cameras that can see the markers the better.
- We experimented with attaching markers to the outside of the motor mounts, but they repeatedly fell off on landings and this solution deemed impractical.

The coordinate system of the Crazyflie was defined relative to the markers and was configured as shown in Figure 2.1. Figure 6.6a shows a photo of the quadrotor with the markers attached.

Furthermore, we attached markers to a stick and defined a goal position to be a fixed distance in front of the stick, there by allowing the user to dynamically move the goal position around during flight.Figure 6.6b show a first person view example.

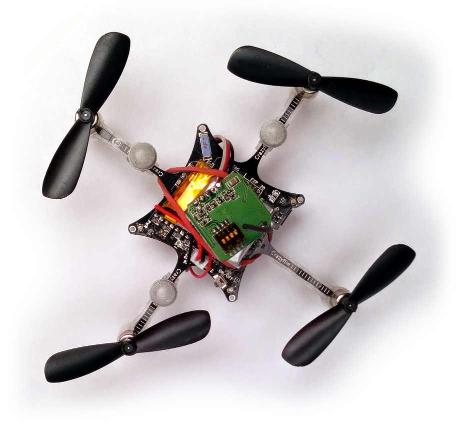
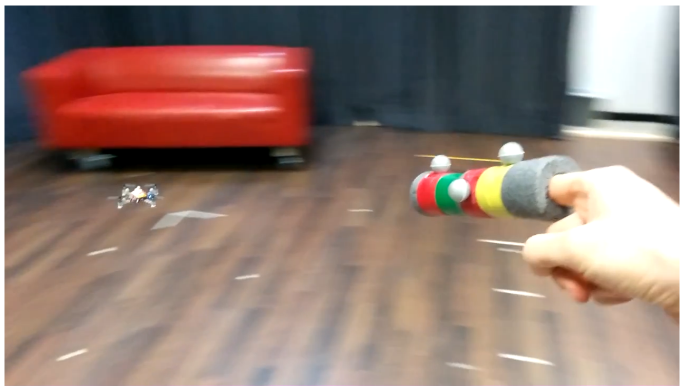


(a) Marker placement **(b)** Dynamic goal set using pointing stick

Figure 6.6: Motion Capture Configuration Photos

Motion Capture Settings

The default timing and exposure configuration of the system was inadequate to detect the fast moving, small markers. Usually the motion capture system runs at 500 Hz, but we reduced it down to 100 Hz so we could ramp up the exposure time, illuminating the markers stronger to allow the camera to see them despite keeping the (noisy producing) imaging sensor gains low.

Qualisys Node Details

A ROS node was written that listened to the TCP/IP socket associated with the output of the dedicated motion capture studio computer. Due to the configuration restriction of the markers, the node performed some post processing on the captured pose to remove the ambiguity of the rotation caused by the symmetric marker placement. The node then time stamped the pose estimate and sent it out over the ROS network as a TF transform, ready for other nodes to query and RVIZ to visualise in real-time.

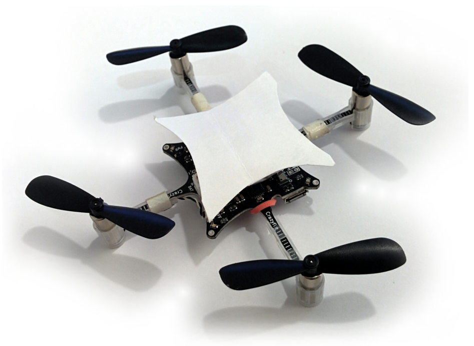
6.5.2 Kinect Based Crayzflie Pose Estimation Node

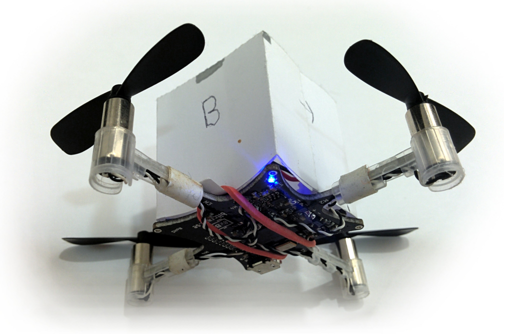
We implemented an external detector that uses a Kinect to estimate the pose. As described in subsection 4.2.2, the on-board Crazyflie attitude estimation is used to determine the heading as the Kinect is only used to detect the position.

The detector was designed with simplicity and rapid production in mind, it is by no means an optimal solution. It outputs its results to RVIZ where the detections can be superimposed over the camera image.

Crazyflie Detection

A ROS node was implemented that listens to incoming depth images supplied by nodes of the ROS freenect_stack[17]. Using depth images, the algorithm first builds a background scene to later segment the Crazyflie against. To improve the detection quality when the Kinect observes the quadrotor from the side, one can add a small piece of paper as shown in Figure 6.7 to increase the silhouette area.


(a) Add paper on the battery to reduce reflections when using a ceiling mounted Kinect.



(b) A little paper cube can improve detection distance.

Figure 6.7: Increasing the silhouette and reducing reflection of the quadrotor improves detection

Background Generation We assume the background is not moving during the background generation phase and that the camera is static. Incoming depth images $\mathbf{D}_1, \ldots, \mathbf{D}_n$ are buffered for a few seconds to create a static scene depth background map $\mathbf{B}$, by using the minimal depth value per pixel over all depth images in the buffer:

$$\mathbf{B}(u,v) = \min_{x=1,\ldots,n} \mathbf{D}_x(u,v) \tag{6.1}$$

Special care is taken to handle inf and nan values correctly by setting them to a maximum distance.

Segmentation To compute a foreground mask $\mathbf{F}$ segmenting parts of the incoming depth image $\mathbf{I}$ from the background $\mathbf{B}$ we perform the following steps:

1. Compute the minimum depth between each pixel of the background and incoming image
$$\mathbf{D}(u,v) = \min(\mathbf{I}(u,v), \mathbf{B}_x(u,v)) \tag{6.2}$$

2. Subtract $\mathbf{D}$ from the incoming depth image, giving us the distance to the background.
$$\mathbf{F}_\delta = \mathbf{I} - \mathbf{D} \tag{6.3}$$

[17]http://wiki.ros.org/freenect_stack

3. Create our foreground mask by thresholding $\mathbf{F}_\delta$ with our minimum required background distance parameter δ_{min}:

$$\mathbf{F}(u,v) = \begin{cases} 1, & \text{if } \mathbf{F}_\delta(u,v) \geq \delta_{min} \\ 0, & \text{otherwise} \end{cases} \tag{6.4}$$

4. Perform some morphological *opening* (*erosion* followed by *dialation*) on foreground mask **F** to fill small gaps and remove noise. The mask might now contain disconnected components, which represent areas in the depth image segmented from the background.

Pose Estimation

1. To identify the center of each component we compute the distance transform (OpenCV's implementation of [97], approximating Euclidean Distance with linear complexity on the number of pixels) **Dist** of our mask foreground mask **F**.
2. For every connected component, we compute the center of mass and approximate pixel size by repeating the following until the distance transform image is blank
   a) Locate the maximum value of the distance transform image, which roughly corresponds to its center of mass. The maximum value corresponds to its radius and if it is too small, we break out of the loop and continue.

   $$u_{max}, v_{max} = \operatorname*{argmax}_{u,v} \mathbf{Dist}(u,v) \tag{6.5}$$

   b) Compute the estimated 3D position of the quadrotor relative to the Kinect camera

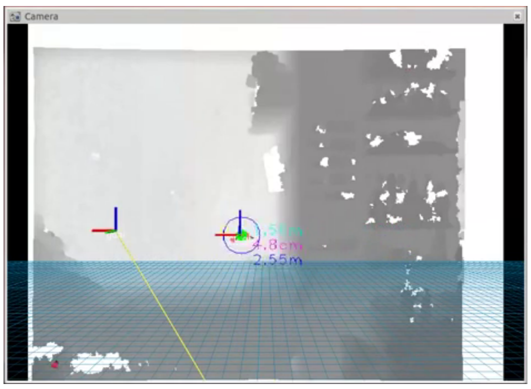
   $${}^{\mathcal{C}}\mathbf{quad} = \begin{pmatrix} (u_{max} - c_x)/f_x \\ (v_{max} - c_y)/f_y \\ 1 \end{pmatrix} * \mathbf{I}(u_{max}, v_{max}) \tag{6.6}$$

   where f_x, f_y, c_x, c_y correspond to the pinhole model parameters (see subsection 4.1.4) of the Kinect.

   c) Flood fill that connected component with zeros using $(u_{max}, v_{max}$ as the seed. This essentially clears that connected component and we can loop to the next one.
3. Optionally filter components by their
   - expected metric size with regard to the diameter in pixels $2 * \mathbf{Dist}(u_{max}, v_{max}$ and depth $\mathbf{I}(u_{max}, v_{max})$,
   - contour shape,
   - maximum depth variation,
   - or other methods. For our needs the above was sufficient.

4. Pick the best remaining component and assume it is the Crazyflie by prioritising the component
   - furthest from the background,
   - closest to the camera from the background,
   - closest to the previously detected 3D position,
   - etc. Once again, for our needs the above was sufficient. Note that motion models, Kalman filters, particle filters, etc. could be used at this point to perform tracking.

In subsection 4.2.2 we explain how to fuse the position estimation from above with the gyro rotation estimate to provide a full 6D pose in the world frame. The above functionality was implemented and integrated into the Crazyflie GUI driver with visualisation being handled by RVIZ. This node can directly be used instead of the Qualisys Motion Capture Node or the VI-SLAM node to provide pose estimations to the Position Controller Node.



(a) Depth image from the kinect with the segmentation and detection overlayed. The numbers correspond to the depth, the estimated radius of the detected object and the distance from the background.

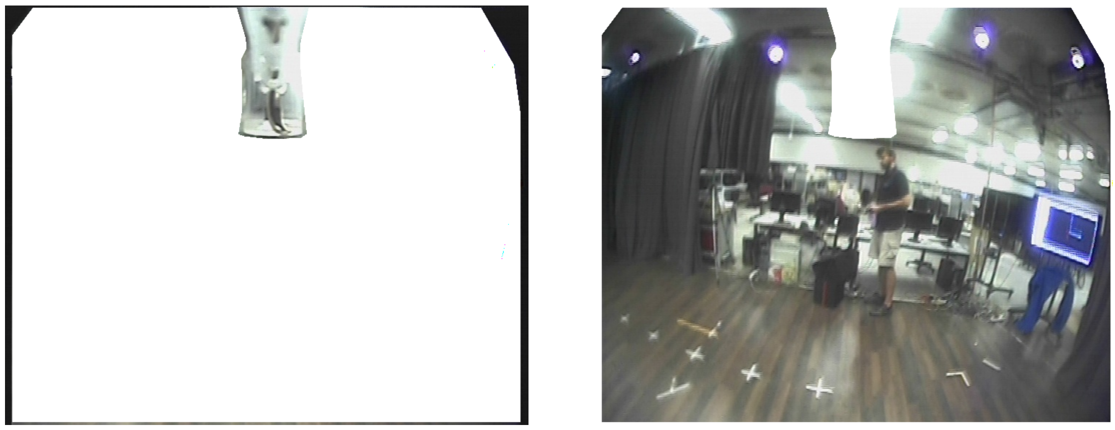
(b) A view from the same scene at the same time from a different perspective. The quadrotor is the small dark blob between the three wall paintings.

Figure 6.8: Using a Kinect one can achieve external position control in small home environments.

6.6 Monocular Inertial SLAM Node

In this section we briefly take a look at the general implementation details of the VI-SLAM node, and specifically how we estimate scale, detect badly transmitted images and remove interlacing artefacts.

The VI-SLAM node listens to images published by the camera driver node as well as telemetric data from the Crazyflie that contains information regarding acceleration, attitude estimation and barometric altitude. The goal of this node is to publish a pose estimation over TF that the position controller can use. In summary, it performs the following steps



(a) The removed regions. **(b)** The remaining regions.

Figure 6.9: Self occlusion is handled by a binary bask that determines the locations where features can be detected.

- Preprocess incoming images,
  - skip the image if the it suffers from too much transmission noise (covered in subsection 6.6.2).
  - apply a binary mask to remove self-occluded image regions from the processing pipeline (example shown in Figure 6.9)
  - deinterlace images to remove the interlacing artefacts associated with PAL (covered in subsection 6.6.4).
- Keep a buffer of attitude estimates from the Crazyflie, so that the incoming images can be time aligned with them.
- For each incoming image and time aligned attitude estimate, perform the VI-SLAM algorithm outlined in Figure 4.2.
- Output the pose estimate result as well as debugging and visualisation information.

The node also works without IMU measurements if the corresponding parameter is set at run-time.

6.6.1 Implementation Specific Details

The node implements the VI-SLAM algorithm outlined in section 4.1. All the parameters are exposed to dynamic reconfigure so they can be changed in real-time, such as

- some controls to force resetting everything, force initialisation and force keyframe generation
- the detinerlacing interpolation method
- an image quality threshold which determines how sensitive we are to dropping images dues to transmission noise
- PTAM camera parameters

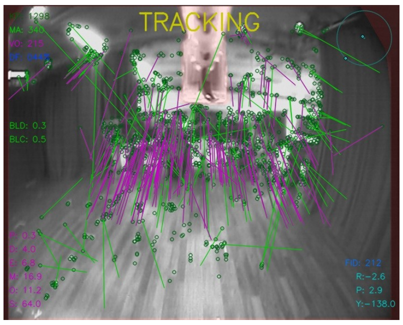
- detector and extractor settings
  - which detector to use (all available ones from opencv)
  - the size of the grid to use when reducing the number of features (see Figure 4.7)
  - the maximum number of key points to detect
  - the detection threshold
  - the number of octaves and octave layers to use for the image pyramid
  - which extractor to use (all available ones from Opencv)
  - if descriptors should be rotated using IMU information
- matching settings (see section 4.1.2)
  - which distance norm to use
  - option to use mutual matches only
  - matching distance threshold
  - minimum ratio required for matches to pass the Lowe ratio test
  - predictive matching angular error thresholds
- VI-SLAM settings
  - the disparity threshold to trigger initialisation
  - the disparity threshold to trigger new keyframe generation
  - the relative and absolute pose estimation method (all methods implemented in OpenGV [71], including translation only using IMU rotation priors [73])
  - ransac thresholds, maximum iterations
  - pose graph optimisation settings (iterations, structure only, window size)

RVIZ can then be used to inspect the transformations of the Crazyflie, camera and keyframes as well as landmark positions, currently matched landmarks and the entire fight path and heading since take-off.

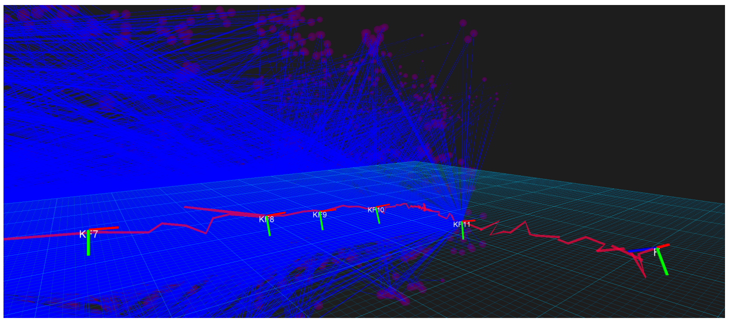
It was implemented in C++ and in a modular way such that sub-modules such as image processing, camera models, landmark, map, frame, auxillary functions, matching, detection and geometric computation related functions were split into different classes. Many of the more math intensive components were originally written in Matlab for easier debugging. The following external libraries were used.

- Eigen [98] was used for all linear algebra,
- OpenCV [51] for image processing and representation,
- OpenGV [71] for geometric vision functions,
- ROS [96] to share transformations, to visualise data, to pass messages between processes, and
- g2o [88] for pose graph optimisation.

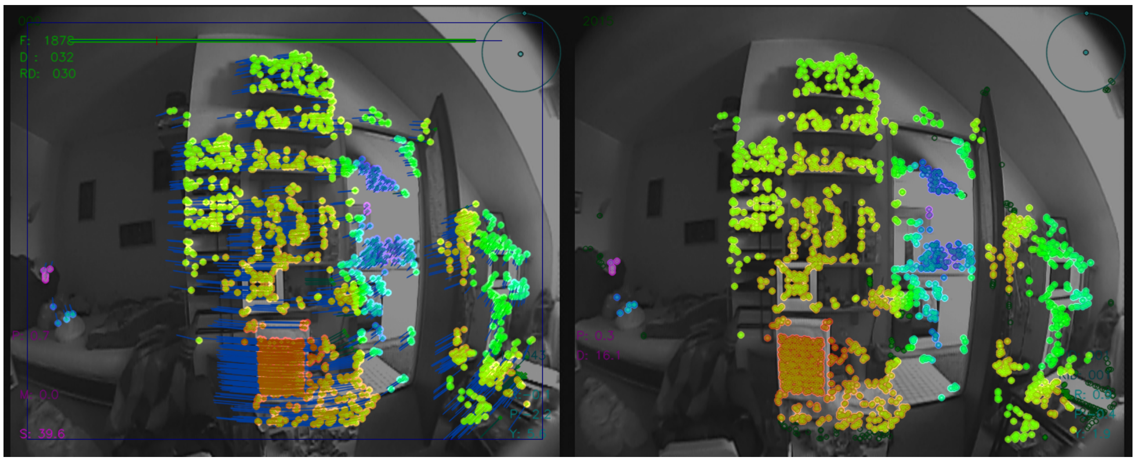
The map was represented as a list of landmarks and keyframes with mutual references: each keyframe has landmark references for each descriptor associated with one; and each landmark had a list of keyframes (and corresponding descriptors ids) from which it was



(a) Feature matching flow visualisation: purple lines represent inliers.



(b) RVIZ debugging infomation: coordinate frames correspond to keyframe poses, purple blobs are landmarks and blue lines associations.



(c) 2D depth visualisation of landmark initialisation between the first two keyframes.

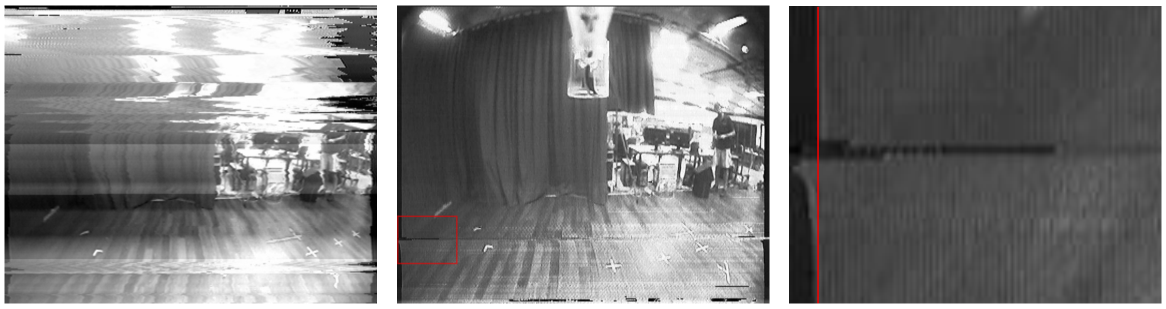
Figure 6.10: Visualisation of the VI-SLAM running and showing debugging output.

visible from. Therefore, the data association is bi-directional, facilitating quicker lookups and traversals, especially when setting up the required structures for RANSAC, keyframe generation and pose graph optimisation.

6.6.2 Corrupt Image Detection

Image transmission quality is dependant on environmental conditions. Depending on interference levels in the 5.8 GHz range, up to 20% of the transmitted images were not usable. Images either suffered from contrast artefacts, or less distinct pixel alignment offsets, usually caused my pixel clock offsets. For example, an image might be shifted multiple rows up or down which completely destroys the relationship between image measurements and the 3D world projected onto the sensor.

Therefore, we required a very fast way to determine if incoming images were accurate and undistorted. Luckily, the imaging sensor does not fully encompass the PAL resolution and therefore the transmitted images have a distinct black border around them where no



(a) A highly corrupted image. **(b)** A less obviously corrupted image. **(c)** Zooming in reveals that some rows have shifted into an image region not used by the sensor.

Figure 6.11: Examples of corrupted images.

measurement was taken. By analysing these borders one can determine if the image has been transmitted correctly.

Figure 6.11b shows an image that suffered from transmission interference and Figure 6.11b shows a close-up of the border region marked in red. Simply by counting the number of pixels beyond a certain intensity threshold (i.e. non-black) within the border region one can obtain an indicator whether or not the image was corrupted. This proved to be extremely reliable and not a single image was falsely classified in our tests. If an image is deemed to be corrupted, it is simply skipped and visually displayed with a red tone.

6.6.3 Image Deinterlacing

As discussed in section 3.3.1, the used camera transports images using the 576i50 video standard, meaning that the images are interlaced. As this causes undesirable artefacts and makes feature detection/extraction imprecise and unreliable we needed a way to preprocessing step to remove the artefacts.

For our purposes, simply interpolating the even rows from the odd rows was sufficient. Effectively, this can be seen as overwriting the even rows with a reconstruction based on its neighbouring rows. We chose to eliminate the even rows as they contain older measurements. Figure 6.12 shows the same image twice, once interlacted and once deinterlaced. A zoom in is provided that highlights the details - one can clearly see even and odd rows. We use the OpenCV interpolation function to provide differerent ways to interpolate with different mask sizes. We defaulted to using bilinear interpolation, which in this case means that pixels on even rows simply take the average value of the pixel directly above and below.

6.6.4 Scale Estimation

Once the scale has been estimated after the intial relative pose estimate, it is implicitly propagated throughout the following absolute pose estimates. However, we cannot deter-

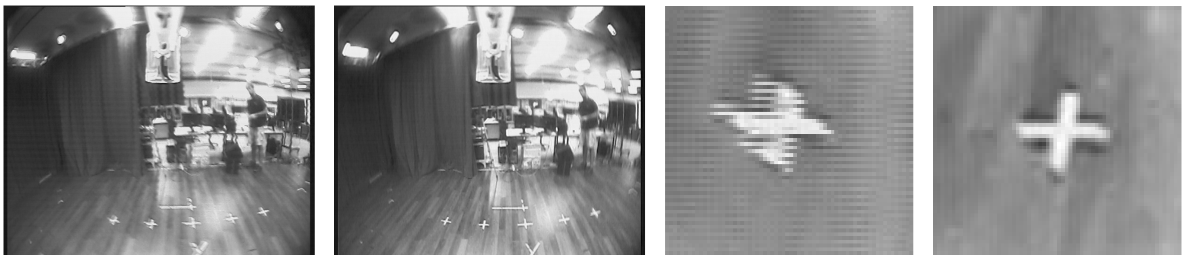

(a) Interlaced (b) Deinterlaced (c) Ghosting Artifact (d) Artifact Removed

Figure 6.12: Comparison between the same image with and without deinterlacing. Details of the cross on the floor demonstrate the interlacing artefact appearance.

mine the initial depth of the triangulated landmarks using a monocular camera alone. As mentioned in section 4.1.1, other scale estimation techniques exist that make use of additional external information such as measured acceleration or information about visible landmarks. However, as shown in section 7.1.3, our accelerometer is too noisy for this purpose due to the vibration of the PCB it is mounted on.

Therefore, we can either set the scale manually by fixing the baseline between the two views used to initialise the map or compute the relative translation magnitude that would result in an averge given scene depth; or attempt to recover the scale automatically using the barometer.

6.6.5 Barometer Based Scale Initialisation

The user can pilot the quadrotor so it takes off in a stable fashion and then active the firmware based hover mode we implemented, which uses barometric pressure to compute relative altitude differences and maintain the current altitude. Once the quadrotor stabilises its altitude autonomously, it can initialise the first keyframe and raise its altitude by an additional meter, at which point it can generate the second keyframe and initialise the map with a baseline computed from the measured altitude difference. Note that if the quadrotor drifts excessively the pilot must still take control of the roll and pitch angles.

As barometers measure air pressure, we can use this to reason over our altitude as the air density decreases with increasing altitude. To compute the altitude above sea level (ASL) in meters at time t_i we use an approximation of the hypsometric formula [99]

$$ASL_t = (T_0/\tau_0) \cdot (1 - (p_t/p_0)^{-R\tau_0/(\mu g)}) \tag{6.7}$$

where p_t is the pressure measurement at time t in mmHg, $p_0 =$ is the pressure at the lower limit of the troposphere, $T_0 = 15\,°\mathrm{C} = 288.15\,\mathrm{K}$ is the temperature at the lower limit of the troposphere, $\tau_0 = -6.5\,°\,\mathrm{km}^{-1}$ is the temperature gradient of the lower troposphere, $R = 8.314\,32\,\mathrm{N\,m\,mol^{-1}\,K^{-1}}$ is the universal gas constant, $\mu = 0.028\,964\,4\,\mathrm{kg\,mol^{-1}}$ is the air molar mass, and $g = 9.806\,65\,\mathrm{m\,s^{-2}}$ is acceleration due to gravity, which when substituted with numerical values becomes

$$ASL_t = 44330.76923 \cdot (1 - (p_t/760)^{0.190263}). \tag{6.8}$$

To compute the relative altitude difference (RAD) between two times t_i and t_{i-1} one simply computes the difference

$$RAD_t^{t-1} = ASL_t - ASL_{t-1}. \tag{6.9}$$

As the barometer is sensitive to micro-climate changes and other sources of noise as discussed in section 7.1.3, we can optionally use the pressure difference between a flying Crazyflie and an additional static Crazyflie to compute relative altitude differences immune to micro climate changes of the environment both quadrotors are situated within. If the reference Crazyflie $\mathcal{Q}_{ref}$ is at ground level, we can estimate the above ground level (AGL) altitude of the flying Crazyflie $\mathcal{Q}_{fly}$ at time t with

$${}^{\mathcal{Q}_{fly}}AGL_t = {}^{\mathcal{Q}_{fly}}ASL_t - {}^{\mathcal{Q}_{ref}}ASL_t \tag{6.10}$$

such that the micro climate change invariant relative altitude difference (RAD) between time t_i and t_{i-1} becomes

$${}^{\mathcal{Q}_{fly}}RAD_t^{t-1} = {}^{\mathcal{Q}_{fly}}AGL_t - {}^{\mathcal{Q}_{fly}}AGL_{t-1} \tag{6.11}$$

which allows us to determine the relative altitude difference between two measurements without being affected by changing micro climate conditions, ultimately enabling automated scale estimation by setting the unit length translational component ${}^{\mathcal{K}_0}_{\mathcal{K}_1}\vec{\mathbf{t}}^{norm}$ of the relative pose estimate between the initial keyframe $\mathcal{K}_0$ and second keyframe $\mathcal{K}_1$ to

$${}^{\mathcal{K}_0}_{\mathcal{K}_1}\vec{\mathbf{t}}^{baro} = {}^{\mathcal{K}_0}_{\mathcal{K}_1}\vec{\mathbf{t}}^{norm} \cdot \frac{{}^{\mathcal{Q}_{fly}}RAD_{t_{\mathcal{K}_1}}^{t_{\mathcal{K}_0}}}{{}^{\mathcal{K}_0}_{\mathcal{K}_1}\vec{\mathbf{t}}_z^{norm}} \tag{6.12}$$

before triangulation.

6.7 Position Control Node

The goal of the position controller is to determine roll, pitch, yaw velocity and thrust percentage commands to send to the Crazyflie, so that the estimated pose of the Crazyflie coincides with a given target position and heading.

The node is written in Python and listens to two TF transformations that represent the current pose and goal pose and forwards commands to the Crazyflie driver node. It implements a PID controller for each degree of freedom as explained in section 5.4. It exposes all the relevant parameters (such as the individual gains and thresholds for the PID of each degree of freedom) to dynamic reconfigure so they can be tuned in realtime. Furthermore, it outputs the individual proportional, integration and derivative contributions as a ROS msg so they can visualised in realtime, which greatly helps tuning.

7 Evaluation

In this chapter we evaluate the effect of the hardware modifications as well as the timing, performance and accuracy of the developed systems on data obtained from a number of experimental test flights with the Crazyflie Nano Quadrotor.

We first focus on the hardware in section 7.1, then look at the developed VI-SLAM system performance in section 7.3 and finish by validating that the various pose estimation methods can be used to achieve position control on the Crazyflie, without requiring any additional filtering or motion models in section 7.4.

7.1 Hardware

7.1.1 Flight Characteristics

In this subsection we consider the effect the modifications required to carry a wireless camera system payload had on the flight characteristics.

Crazyfie and Payload Weight

As Table 7.1 shows, the default, ready to fly Crazyflie weighs just under 19 g. The wireless camera system weighs 6.3 g, which might not sound like much but is over a third of the weight of the quadrotor itself. With additional effort, the flight system could probably be brought down to 5 g if shorter and thinner wire was used. Unfortunately we could not use the lighter and smaller 2.4 GHz transmitter as there was too much interference and the received images were too noisy. The weight of the lighter transmitter is included in the table below for completeness.

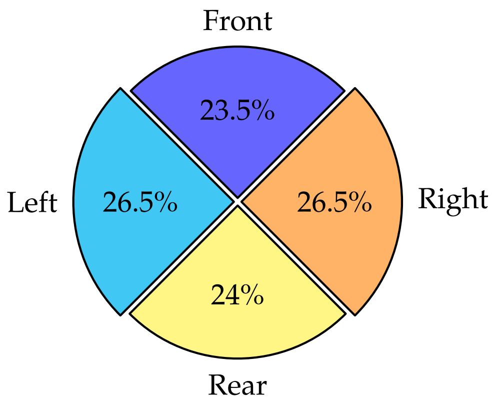
Weight Distribution

Adding a 6.3 g payload to the Crazyflie is only possible if one can keep the center of mass in the middle so that the motors can equally distribute the additionally required thrust. Here we verify that our payload weight distribution and position does not cause an uneven thrust distribution to keep the Crazyflie level.

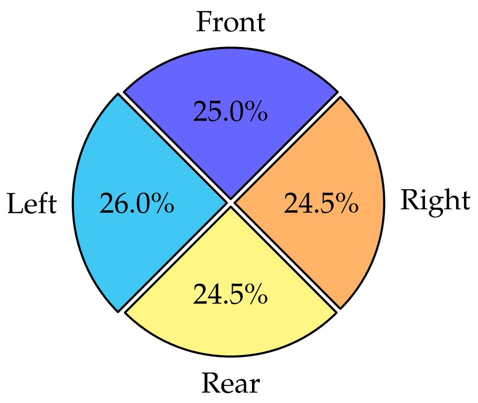
Figure 7.1 shows the contribution of thrust of each of the motors before and after adding the payload. Initially, there was a slight bias towards the left and right motor thrust contribution. Adding the camera system caused a slight center of mass shift to the left,

Table 7.1: Weight of each component used

Component	**Number**	**Weight**	**Σ Weight**
Crazyflie PCB	1	5.20 g	5.20 g
Motors	4	1.64 g	6.56 g
Motor Holders	4	0.21 g	0.84 g
Rotors	4	0.26 g	1.04 g
Battery	1	4.82 g	4.82 g
Camera	1	1.49 g	1.49 g
Wide Angle Lens	1	1.58 g	1.58 g
Transmitter 5.8 GHz (2.4 GHz)	1	1.19 g (0.57 g)	1.19 g (0.57 g)
Voltage Regulator	1	0.45 g	0.45 g
Camera Holder	1	0.75 g	0.75 g
Reflective Markers	3	0.25 g	0.75 g
Cables, solder, etc.	1	0.86 g	0.86 g
		Crazyflie Weight	18.46 g
		Camera System Weight	6.32 g (5.70 g)
		Total Flying Weight	25.53 g (24.91 g)



(a) Without Camera



(b) With Wireless Camera Payload

Figure 7.1: Visualisation of the contribution of thrust per motor required to achieve a stable hover with and without the wireless camera payload.

but the difference is negligible. We can therefore continue knowing that the payload has been placed in such a way that minimises a difference in the flight characteristics. See section 7.1.1 for details on how the thrust required to hover changed by adding the payload.

Throttle and Lift

The amount of thrust the motors generate depend on the state of the battery and the throttle. A throttle of 100% corresponds to running the motors with all available power. To generate the additional thrust to carry the camera payload, the throttle must be increased. Furthermore, if the camera and transmitter are drawing power, even more throttle must be applied to make up for the additional current draw. The amount of throttle used over the entire discharge of the battery is an important indicator of how well the Crazyflie can respond to desired altitude changes, which is especially important when it needs to recover from rapid altitude decreases. Here we evaluate the effect of carrying and powering the wireless camera on the throttle use during hover.

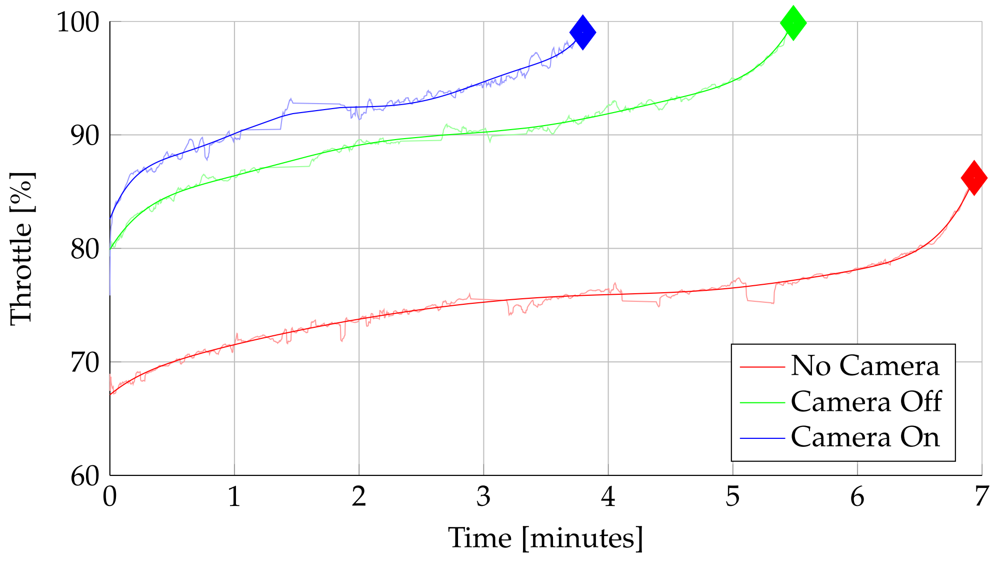


Figure 7.2: A plot showing the amount of throttle required to hover the Crazyflie (*a*) without any payload, (*b*) carrying the camera system but having it turned off, and (*c*) carrying and powering the camera system over the entire battery discharge time. The diamonds represent the time the battery depleted. The thin lines represent the raw measurements.

Looking at item 7.2 it becomes apparent that that the payload weight of the camera is at the practical limit of the Crazyflie. At the three minute mark, the Crazyflie required around 75% throttle to maintain its altitude in the normarl case. This increases to 90% when carrying the payload, and 95% when powering the camera and transmitter. This means that there is only an extra 10% or 5% thrust available respectively. For a stable hover that might be sufficient, but for gaining altitude or for reducing negative vertical velocity safely, it is insufficient. Therefore, the Crazyflie will only be able to safely decrease its altitude at extremely slow speeds if the camera system is used. We conclude that carrying and powering the camera system is therefore barely practical as the Crazyflie simply does not have enough thrust reserves to manoeuvre.

Flight Time

The current voltage which decreases at the LiPo discharges directly affects the amount of thrust generated by the motors, assuming constant throttle. Therefore it is no surprise that Figure 7.3 looks like the inverse of the thrust plot item 7.2.

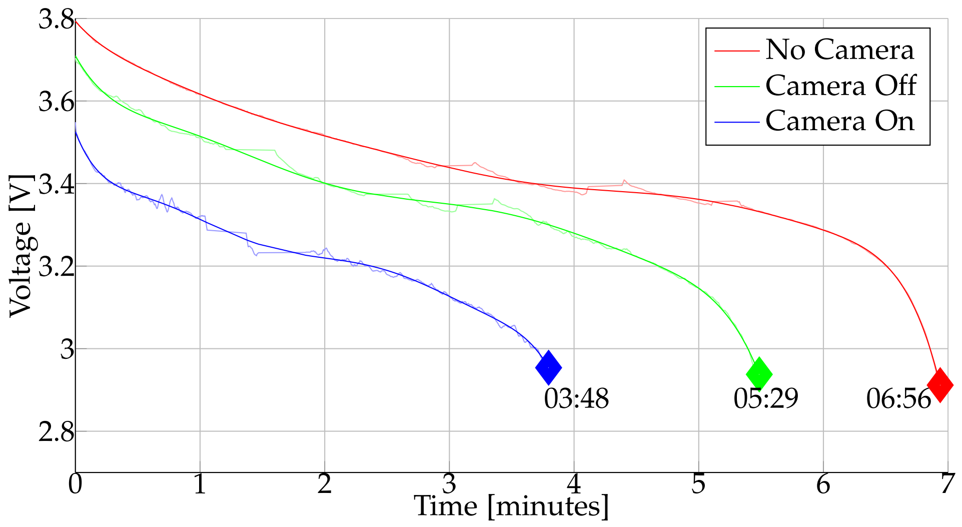


Figure 7.3: Evaluating the discharge of the Crazyflie during a stable hover. Usually, the Crazyflie can remain airborne for around 7 minutes before powering down. However, the time decreases by 90 seconds simply because of the extra power required to generate the additional thrust required to lift the payload. Ultimately, powering the camera system causes additional battery drain reducing the flight time down to 03:48.

Here we evaluate the flight time of a the Crazyflie performing a stable hover in the motion capture studio. The data is taken at the same time of the thrust evaluation. The extra payload reduces the flight time by roughly one and a half minutes and powering the camera reduces it almost an additional two minutes. A flight time of 03:48 is achievable while powering the on-board camera during a stable hover. For scientific purposes this is enough, yet still impractical when running multiple experiments. Therefore we recommend to use multiple batteries so that depleted ones can be recharged (takes around 20 minutes) parallel to further experiments. We noticed the battery performance decreased beyond the 100 charge cycle mark.

7.1.2 Crazyflie Communication Throughput and Timing

We very briefly take a look at the average timing delay of the Crayzflie logging framework. We devised a simple way to get a rough estimate of the timing involved with sending and receiving logged data to and from the Crazyflie in section 6.4. Here we simply show the measured results, as with respect to the current out going bandwidth.

When requesting logging (e.g. sensor measurements) from the Crazyflie, one can chose at which frequency one would like which logs. For example, one may wish to have accelerometer data at 50 Hz and and attitude information at 100 Hz. The throughput varies depending on the log types (e.g. an unsigned 8 bit integer vs a float), the frequency requested and the number of logs/ If one requests too much data, the high throughput can cause significant delays. In Table 7.2 we show the timing results around the throughput threshold region. 1500 byte/s upstream is from sending control commands at 100 Hz.

Table 7.2: Packet Travel Time

Download [byte/s]	**Upload** [byte/s]	$\delta_{packet}/2$ [ms]
1400	1500	6.4 ± 2.2
1400	5900	9.1 ± 4.0
1400	6200	38 ± 3.1

Therefore we conclude that one should not exceed a downstream throughput of ~5900 byte/s, which corresponds to roughly 10 floats at 100 Hz.

7.1.3 Crazyflie Sensors

Accelerometer

As the Crazyflie consists of a single PCB that also serves the purpose of being the quadrotor frame, it and all components soldered to it receive the vibration from the motors directly. Usually one tries to cushion the IMU to isolate it from motor or other aerodynamic induced vibration, but on such a small quadrotor this is simply not feasible.

As a result, the accelerometer measurements are extremely sensitive to the motor usage. We demonstrate this by simply plotting the standard deviation of the acceleration measured in each dimension on a static quadrotor while gently increasing the thrust. The quadrotor was held firmly against the foam padding it came shipped with and the propellers were put on upside down to avoid turbulence induced vibration.

As Figure 7.4 shows, the standard deviation of the accelerometer measurements increases with additional thrust. The Crazyflie requires around 75-90% throttle to hover, at which point the standard deviation of the accelerometer measurements is $\sim 0.1\,\mathrm{m\,s^{-2}}$. Therefore we conclude that using the acceleration measurement magnitudes for some kind of velocity or position estimate is simply not feasible. However, the direction of the measured acceleration vector is good enough to keep roll and pitch from drifting in the sensor fusion algorithm mentioned in section 4.3.

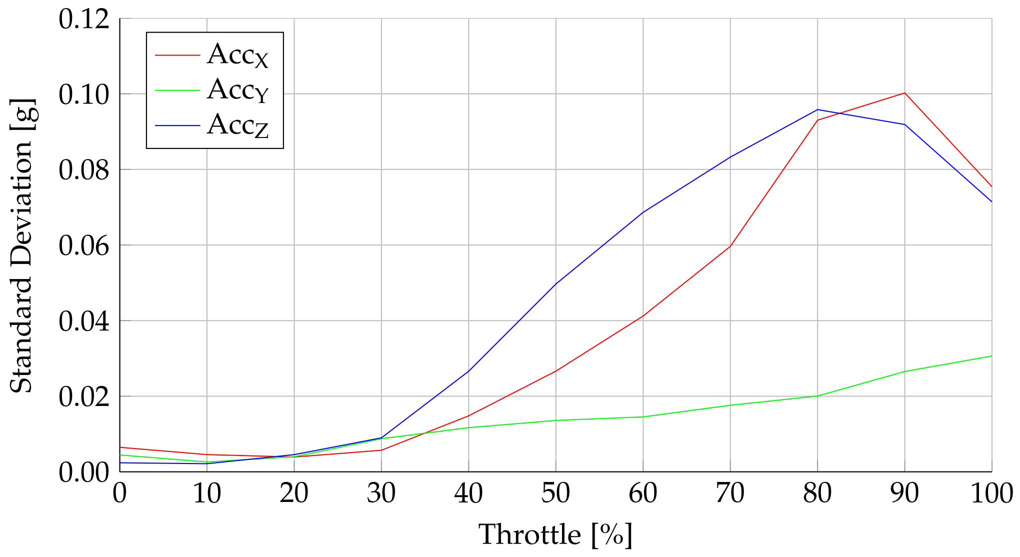


Figure 7.4: Shows the standard deviation of measured acceleration forces of a static Crazyflie as a function of increased throttle. This plot shows how the vibration induced noise increases with increasing motor speed

Gyro Drift

In chapter 4 we assume the frame to keyframe relative rotation estimate provided (see subsection 4.1.7) by the Crazyflie's on-board attitude estimation (see section 4.3) has negligible drift, allowing us to solve for translation only in the the VI-SLAM pose estimation steps. Furthermore, the Kinect based pose estimator relies on a the attitude estimate, as described in subsection 4.2.2.

However, as mentioned in section 3.2.1, the yaw compnant does drift slowly. Here we check that the yaw drift is indeed negligible for the short flight times the Crazyflie can provide. We collected yaw measurements over an 8 minute window and plotted them over time in Figure 7.5. One can observe that the yaw does indeed drift very slowly, around 1.18 °/s. This estimate does correspond to what we observed during the other experiments. However, it is important to note that this drift can indeed be increased through violent acceleration - such as during a crash. We conclude that yaw drift is negligible under normal circumstances for the <7 minute flight time the Crazyflie endures.

Barometer

We use the barometer for scale initialisation in subsection 6.6.4 and obtain measurements at 90 Hz from the custom driver we implemented (details in section 3.2.2). We briefly evaluate the barometer performance in this subsection.

Figure 7.6 plots barometer based altitude over the duration of a typical starting sequence. We plot both the barometer altitude estimate from a single Crazyflie (denoted Baro$_{\text{Orig}}$),

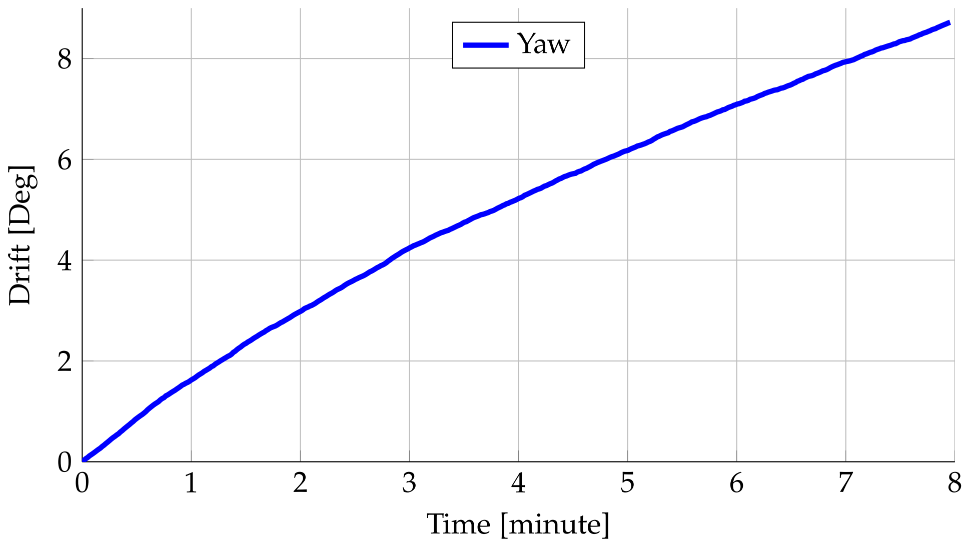


Figure 7.5: Yaw plotted over time from a static Crazyflie. There certainly is drift, but for the short flight time of the Crazyflie it is negligible.

the compensated estimated (which we obtain using barometric readings from another Crazyflie acting as a base-station - see subsection 6.6.4, denoted $\text{Baro}_{\text{Compensated}}$), the raw altitude estimated (denoted Baro_{Raw}) as well as the ground truth altitude as measured by a motion capture system.

Initially, the quadrotor is on the ground. We *zero* the barometer altitude so we can estimate our altitude relative to the ground level instead of sea level. The motors spin up, which causes a huge dip in estimated altitude. This occurs as the air pressure between the floor and rotors increases before the quadrotor eventually generates enough thrust to take-off. The quadrotor autonomously can hold its altitude at 0.5 m and raise its altitude to 1.5 m. This 1.0 m altitude difference can be used to initialise the scale of our VI-SLAM implementation.

We can observe that $\text{Baro}_{\text{Orig}}$ drifts over time, while $\text{Baro}_{\text{Compensated}}$ does not. This is expected as $\text{Baro}_{\text{Compensate}}$ uses relative pressure differences and therefore both the base-station and airborne quadrotor suffer from the same environmental changes.

We conclude that using a relative pressure differences certainly helps in the long run as the drift is successfully eliminated, but for the estimates for the initial 1.0 m altitude increase used by the VI-SLAM system is accurately estimated by both $\text{Baro}_{\text{Compensated}}$ and $\text{Baro}_{\text{Orig}}$, assuming there are no sudden pressure spikes caused by opening windows, doors, etc.

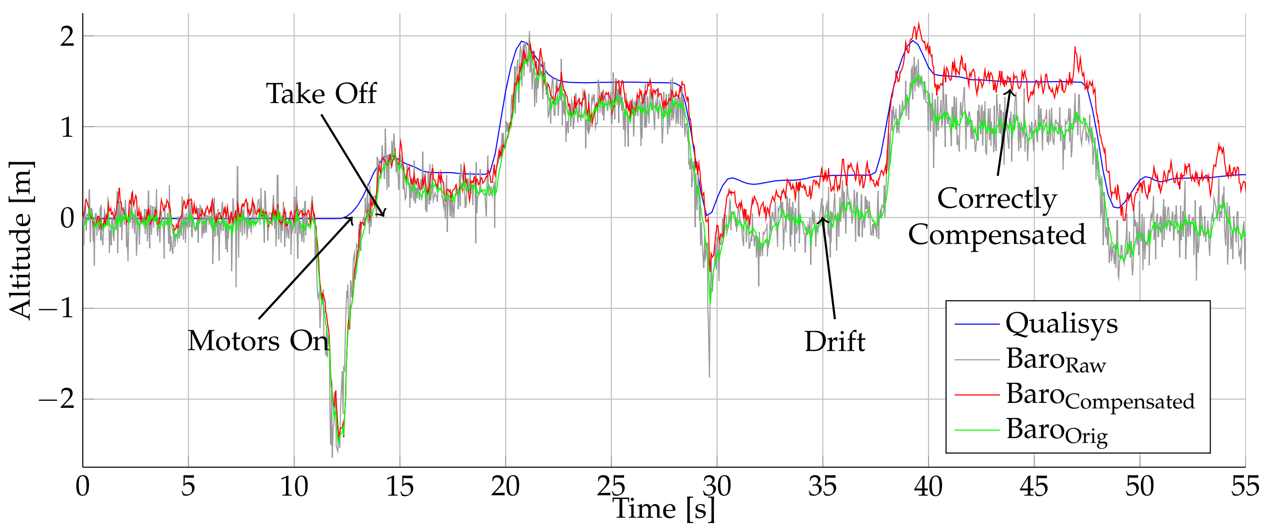


Figure 7.6: Barometer altitude estimates plotted over time from a typical starting sequence. Note that the compensated measurements drift less than the original ones. Also note the incorrect measurements caused by the turbulence right before take-off.

7.1.4 Wireless Image Transmission

Transmission Range

the 5.8 GHz transmitter/receiver combination worked fine for indoor use with a clear line of sight. Ranges up to 15 m were relatively noise free in typical office/lab environments with infrequent frames were classified as corrupt. In sparse environments with better line of sight, the range improves but we restricted our use to the indoors. However, near the end of a battery discharge cycle the number of corrupted frames increased, depending on interference and other environmental factors, up to about one in five. We noticed that that frequency was often crowded and often had to switch channels (e.g. from 5725 Mhz to 5865 Mhz using the dip switch on the transmitter and soldering the appropriate pins to ground on the receiver). The 5.8GHz frequency has troubles penetrating walls and therefore the signal is almost immediately lost when flying into adjacent rooms.

Image Delay

In subsection 6.3.1 we implemented a way to estiamte the camera lag, that is the duration it takes from image acquisition to being able to access the image in a ROS node. For the test, we used a Hazro HZ27WC monitor. This is a bare-bones IPS monitor with no on screen display (OSD) menu or resolution scaler, meaning it has a low and stable input lag of 8.8 ms (as determined by TFTCentral [1], a monitor enthusiast website).

In Table 7.3 we summarise the average recorded lag for various camera drivers as well as their CPU usage (on an Intel Core i5 4650K). Roughly 1200 measurements were taken over

[1]`http://www.tftcentral.co.uk/reviews/content/hazro_hz27wa.htm`

a 3 minute period at a varying frequency between 3 Hz and 13 Hz.

Table 7.3: Camera Driver Evaluation

Camera Driver	Mean	Standard Deviation	CPU Utilisation
gscam [2]	53 ms	16 ms	10%
usb_cam [3]	50 ms	12 ms	7%
uvc_camera [4]	21 ms	11 ms	20%

The uvc_camera node turned out to give us an image with the least lag but using the highest CPU usage while gscam and usb_cam performed suffered from much more lag but with a similar lag to CPU utilisation ratio. As we had enough processing power available, we opted to use the uvc_camera node to provide images to the ROS network. Estimating the lag allows us to better match IMU measurements with camera images which is important for the Visual-Inertial SLAM system.

7.2 Kinect based Pose Estimation

Unfortunately, we could not compare the poses obtained from the Kinect pose estimation node versus the ground truth poses from the motion capture system, as the infra-red (IR) lights required by the motion capture system interfered with the kinect depth sensor. The significant noise and infra-red glare on the floor was substantial enough to make it impossible to estimate the depth of an obstacle as small as the Crazyflie. Background segmentation failed hopelessly - the Crazyflie was not detected at all after one turned up the pre-processing tolerances of the segmentation algorithm to overcome the noise.

However, we overlay the detection results over the RGB camera image to visually inspect the output. We tested the algorithm in three different settings (*a*) a one room apartment with the Kinect pointing forwards, (*b*) a garage and living room with the Kinect mounted on the ceiling pointing downwards, (*c*) an office with the Kinect pointing forwards.

As hinted in subsection 6.5.2, to achieve reliable detection results we had to add a bit of paper to increase the surface area of the Crazyflie. This is due to the small size of the quadrotor and to prevent the IR projection reflecting off some of the quadrotor surfaces.

Using the additional surface area provided by the paper attachment, detection results were stable up to a range of 3 m. Beyond this, the detection would occasionally fail. Beyond 4 m range the detection would usually fail to segment the Crazyflie in the depth image - there we simply no depth values. It takes around 11 ms to process each depth image and return visual feedback, and some times increases up to 20 ms if there are multiple detections or a lot of noise.

False positives only occurred in rare situations, such as when multiple, small objects were in view simultaneously. Occasionally, the system would also detect false positives if no Crazyflie was in view. In subsection 7.4.2 we give an example of a trajectory flown using the estimated pose of the detector for control input, which demonstrates flying waypoints within the Kinect's field of view.

7.3 VI-SLAM Evaluation

To evaluate the performance of the VI-SLAM system, we flew a set of waypoints in a motion capture studio using the ground truth pose as input for the position controller.

7.3.1 Pose Estimation

To compare the the ground truth and VI-SLAM trajectories, we scaled the first 5 seconds of the VI-SLAM system poses to those from the motion capture studio. The scale estimate of the barometer was at 115% percent of the metric scale (usually within $\pm 20\%$ error - highly dependent on environmental conditions), showing that it can be used as a rough estimate.

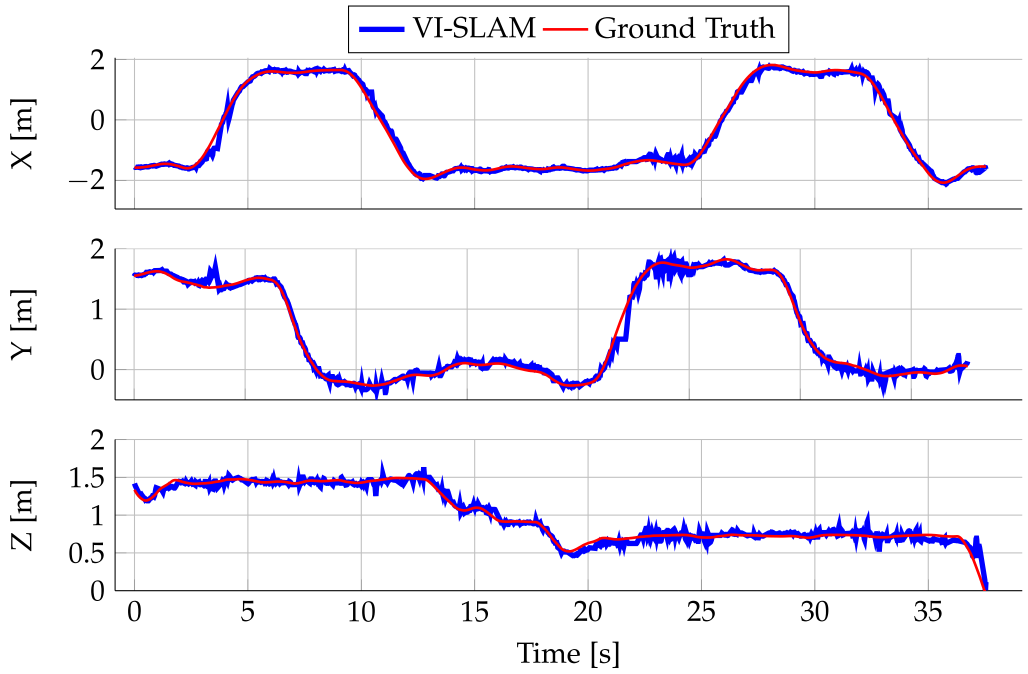


Figure 7.7: 2D visualisation of the VI-SLAM estimated pose versus that of the motion capture system for flying one waypoint circuit

Figure 7.8 shows the flown waypoints, the trajectory our VI-SLAM system estimated and a colour coded ground truth trajectory in 3D, where the colour corresponds in the magnitude of the translational velocity. Figure 7.9 shows the same information 2D.

Figure 7.8 shows the flown waypoints, the trajectory our VI-SLAM system estimated and a colour coded ground truth trajectory in 3D, where the colour corresponds in the magnitude of the translational velocity. Figure 7.9 shows the same information 2D.

We computed the root-mean-square-error (RMSE) between corresponding ground truth and estimated poses for the individual translational dimensions X, Y and Z as well as the 3D euclidean distance between them (denoted XYZ). The results as well as the maximum error in each dimension are summarised in Table 7.4.

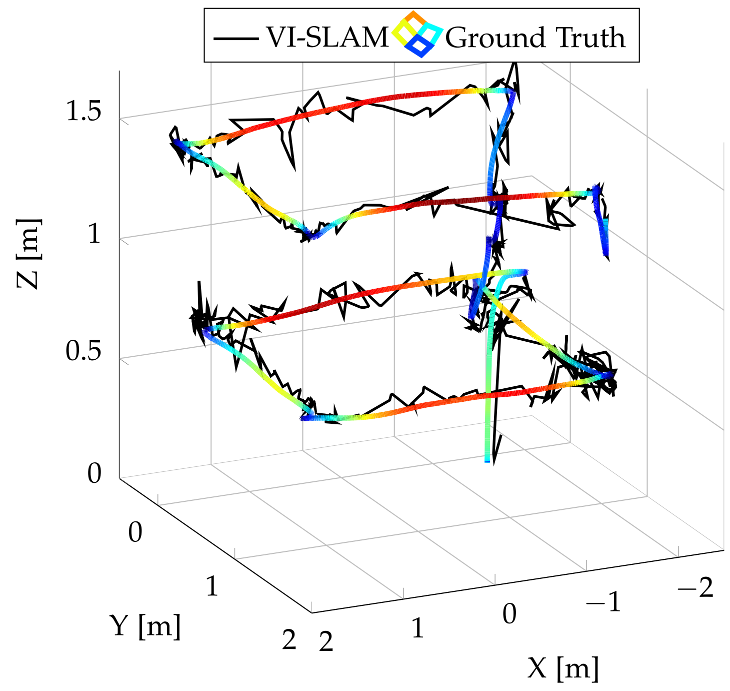


Figure 7.8: 3D plot of the flight trajectory used to evaluate the VI-SLAM system. Black lines denote the VI-SLAM pose estimates and the coloured one is given by a motion capture system running at 100 Hz. The colour corresponds to translational velocity: a hotter colours correspond to faster speeds, with dark red being around 1.2 m/s

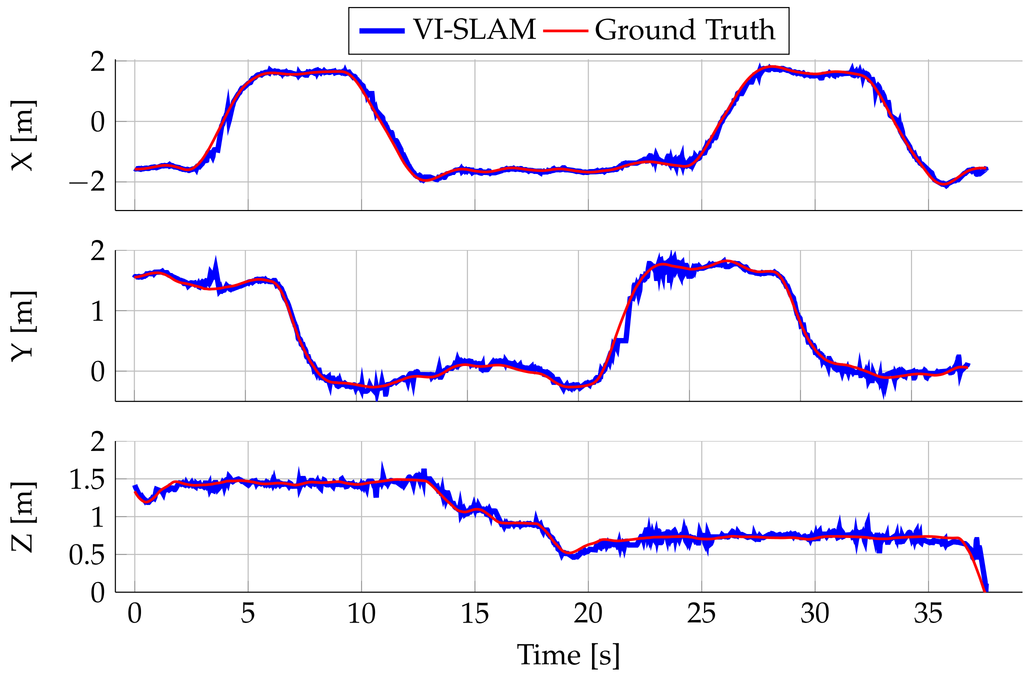


Figure 7.9: 2D visualisation of the VI-SLAM estimated pose versus that of the motion capture system for flying one waypoint circuit

Table 7.4: VI-SLAM Translational Error versus Ground Truth

	X **[cm]**	Y **[cm]**	Z **[cm]**	XYZ **[cm]**
RMSE	9.66	6.69	5.12	8.19
MAX	19.02	24.51	18.81	41.07

Considering the cheap camera quality, rolling shutter and interlaced video transmission, we are very satisfied with the overall RMSE being within 10 cm. Furthermore, the system handled dynamic portions of the same perfectly well, such as when people walked past or when furniture was moved. As long as RANSAC has enough correct matches to build a correct model from, everything that doesnt fit the model (e.g. a moving object) is simply not used. Depending on the number and distribution of features, a new keyframe was successfully generated and integrated into the map every 0.5 m-0.5 m depending on the feature distribution.

7.3.2 Timing

The algorithm runs in real-time, i.e. a pose is computed for every incoming camera frame. In the ideal case, the camera supplies 25 fps, meaning we have 40 ms of compute time per frame. As the image delay is estimated to be around 21 ms (section 7.1.4) and a frame usually takes around 34 ms to process, the pose estimate has a combined delay of roughly 55 ms, which should be sufficient to achieve stable position control according to our tests in section 7.4. However, it is important to note that depending on the environmental conditions, we experienced up to 20% of frame loss due to transmission noise. The corrupt frame detector worked flawlessly. Unfortunately, we did not have enough time to imple-

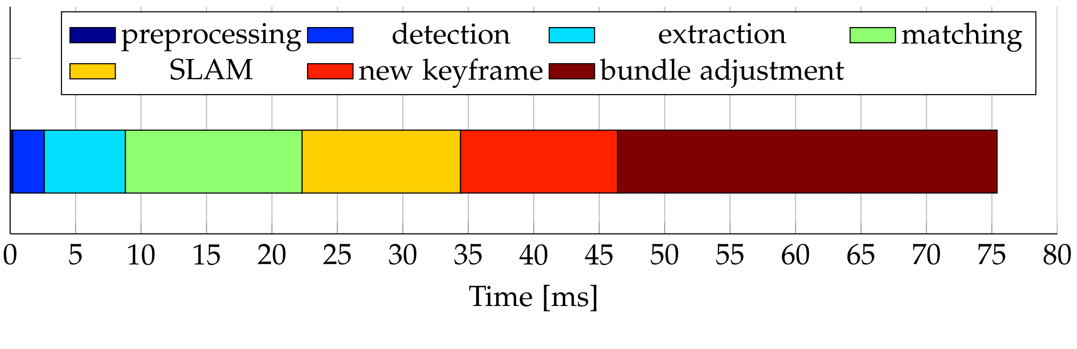


Figure 7.10: Breakdown of the timing of various steps in the VI-SLAM algorithm. Note that red portion of the graph only occurs when keyframes are added, which is not the usual case.

ment a separate mapping thread, so depending on the size of the map and the size of the bundle adjustment window, integrating a new keyframe into the map can cause a frame to be skipped. Typical timing results are summarised in Figure 7.10. The scene had an average of around 650 fast corners detected, of which usually around 70% were matched and 50% were RANSAC pose estimation inliers.

7.4 Control Evaluation

In this section we evaluate how well we can control the Crazyflie given control inputs from different sources. First we consider using external cameras to provide pose estimates and finally conclude by showing that our Visual-Inertial SLAM system is fast and accurate enough for successful position control.

We consider two goals:

Hovering - the ability to remain at a fixed position without drifting.
Here we test how steadily we can hold our position for an entire minute. To evaluate this, we took off manually, started the controller, and allowed the Crazyflie to autonomously stabilise its position before starting to record the translational error in each dimesnion between the reference target position (usually $(0, 0, 1.0)^T m$) and the ground truth position. In Table 7.5 below we summarise the root-mean-square-error (RMSE) and the maximum error over each dimension and the 3D Euclidean distance. We provide 3D plots all with the same scaling to give the reader an intuitive feeling for how well the task was performed. Additionally 2D plot for each dimensions ere provided for closer inspection.

Flying a series of waypoints - the ability to fly to specified goals efficiently.
This goal requires the quadrotor to fly from its current position to a newly specified position, where it should remain until the next waypoint is given. Ideally, it would fly to and converge on the new position directly, quickly, and without overshooting it. Here we estimate the average overshoot, rise-time and convergence time as defined in section 5.2.1 over multiple trials. Once again we provide 3D plots to better visualise waypoint configurations and 2D plots for each dimension.

Table 7.5: Evaluation of Autonomous Drift-Free Hover

Pose Method	Param [ms, Hz, cm]			RMSE [cm]				Max [cm]			
	Lag	Freq	Noise	X	Y	Z	XYZ	X	Y	Z	XYZ
$\text{MoCap}^{3\text{ ms}}_{100\text{ Hz}}$	3	100	0.3	5.9	3.7	0.6	7.0	17.5	12.8	1.7	19.3
$\text{MoCap}^{75\text{ ms}}_{25\text{ Hz}}$	75	25	10	13.8	15.8	7.24	22.2	50.2	44.9	19.3	65.4
$\text{MoCap}^{100\text{ ms}}_{25\text{ Hz}}$	100	25	10	28.9	45.6	32.4	63.0	n/a	n/a	n/a	n/a
Kinect	∼20	30	∼3	4.3	5.4	1.9	7.2	11.7	14.0	6.3	14.1
VI-SLAM	∼55	20-25	∼10	9.9	10.2	3.4	14.6	25.8	20.9	10.3	26.8

7.4.1 Qualisys Tracking Based Control

We first tested the performance of the position controller using pose estimates from a motion capture system. This should be the easiest case as the motion capture system operates accurately at a high frequency with next to no delay.

Hovering Stability with Pose Estimation Lag

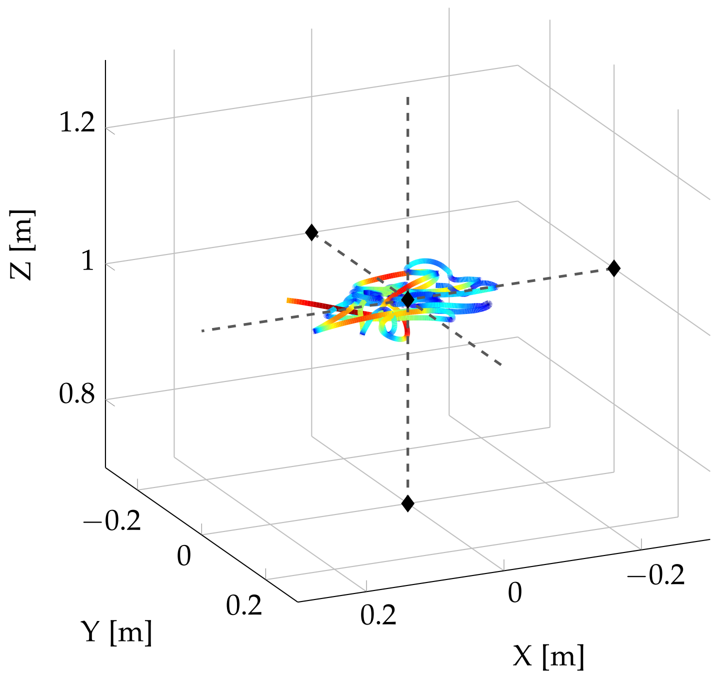


Figure 7.11: 3D visualisation of hovering using the motion capture studio pose estimates. RMSE is ∼7cm.

Figure 7.11 and Figure 7.12 show the same trajectory for a minute of automated hover without carrying the camera payload. Adding the camera actually improved the performance as the quadrotor became more stable with the additional mass.

Instead of simply hovering under ideal conditions, we experimented to observe how the quadrotor reacts to artificially induced delay, noise, and frequency reduction of the pose estimate from the motion capture system. Specifically we were interested in seeing how far we could push the pose estimate delay while keeping the other conditions similar to the expected VI-SLAM system values, i.e. with poses coming in at 25 fps with some additional gaussian noise. This should give us an estimate of the maximal frame processing time available for the VI-SLAM system. The results are summarised in Table 7.5 and visual comparisons can be made in Figure 7.13.

As expected, hovering under the best case conditions resulted in low translational offsets and very stable behaviour. With an artificial 75 ms lag and additional 10 cm of Gaussian noise at 25 fps we were still able to maintain our position but with reduced precision, with the quadrotor eventually converging to a slow, converging oscillation. The limits were exceeded when we attempted to hover with 100 ms lag and 10 cm noise. The quadrotor circled the target position until oscillating out of control. As the VI-SLAM pipeline requires around 55 ms from image acquisition to pose estimation, there is a reserve of 20 ms so it should be fast enough for position control. See Table 7.5 for a summary of the evaluation results.

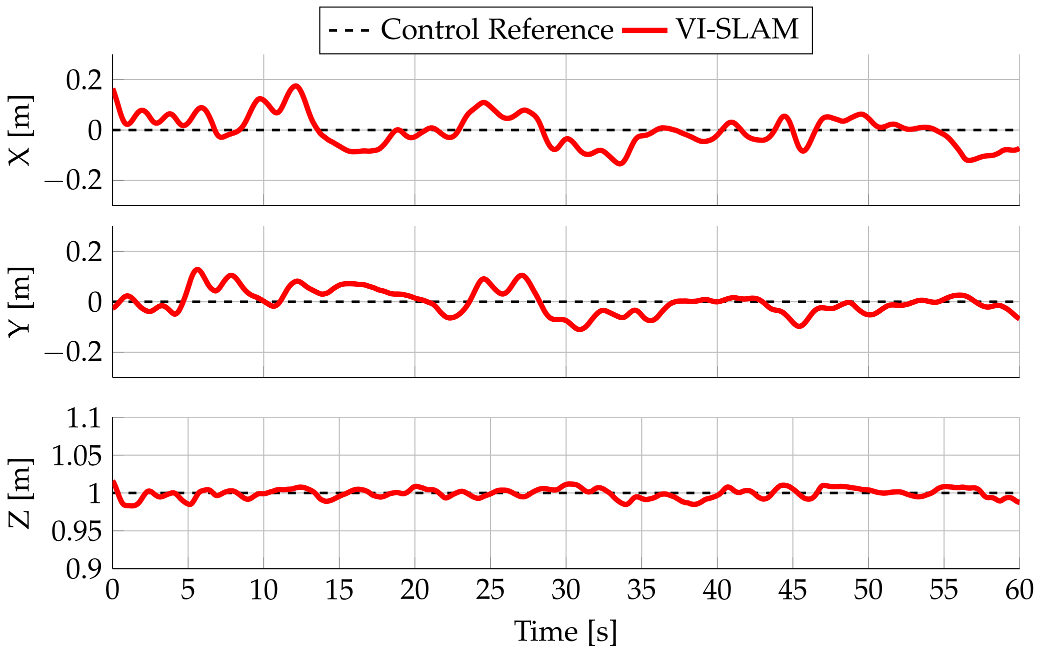
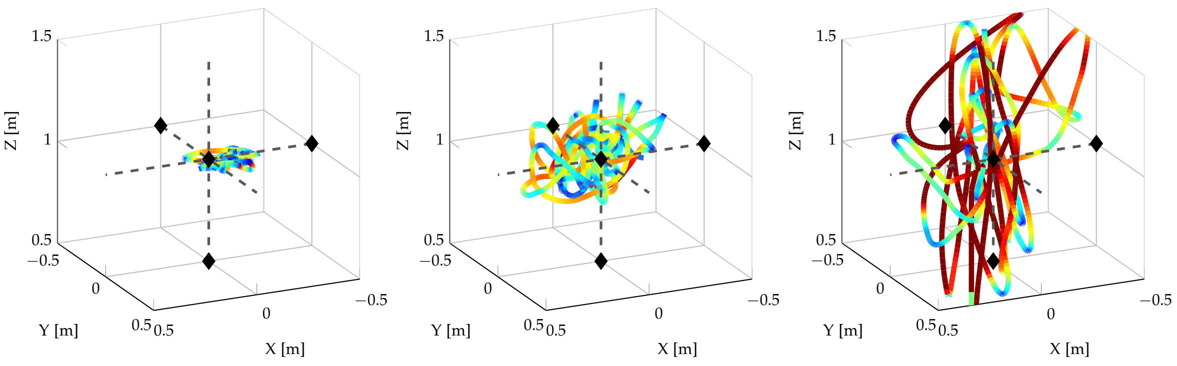


Figure 7.12: Results from hovering using motion capture studio pose estimates. Errors in each dimension are under 7 cm.



(a) MoCap$^{3\,\text{ms}}_{100\,\text{Hz}}$ RMSE=7 cm **(b)** MoCap$^{75\,\text{ms}}_{25\,\text{Hz}}$ RMSE = 22.27 cm **(c)** MoCap$^{100\,\text{ms}}_{25\,\text{Hz}}$ Oscillated out of control

Figure 7.13: Visualisation to compare hovering under different conditions. Lag around 100 ms and over resulted in unstable control, eventually leading to unbounded oscillations and a crash.

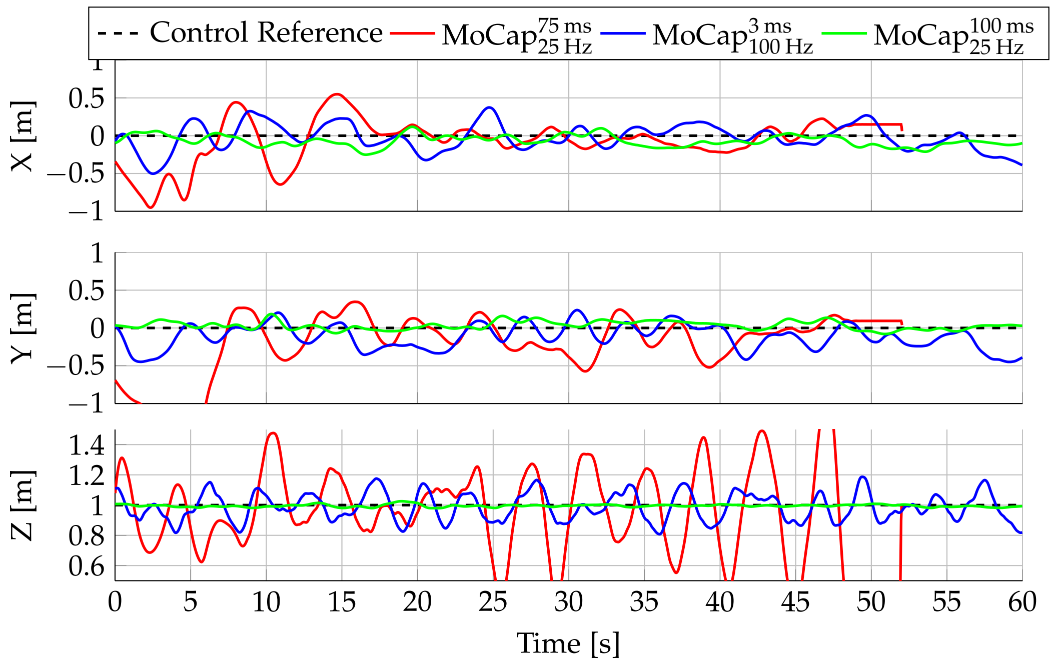


Figure 7.14: 2D visualisation of the translation of the quadrotor during the hover experiment. We can observe that the thrust control is most sensitive to additional delay in the pose estimates.

Waypoint Flying

We setup a few waypoints within the motion capture area such that they covered the maximum space that the Crazyflie could reliably be tracked in. This involved some 2.5 m step sizes in the plane and an altitude increase/decrease of 1 m. The configuration is shown in Figure 7.15 and each translational component is visualised in Figure 7.16.

Table 7.6: Autonomous Waypoint Following: Motion Capture Studio

Step Size [m]		**Rise-Time [s]**			**Overshoot [cm]**			**Convergence [s]**		
x,y	$\pm z$	x,y	$+z$	$-z$	x,y	$+z$	$-z$	x,y	$+z$	$-z$
2.5	1.0	2.1	1.7	1.4	22	12	19	0.6	0.4	1.2

The control evaluation results are given in Table 7.6. Considering the speed at which the quadrotor is traversing the 2.5 m distance, the 22 cm overshoot is completely satisfactory. We conclude that one can achieve reactive control performance by using the pose estimates from the motion capture studio.

7.4.2 Kinect Tracking Based Control

Once again, we first evaluate the hover performance of the controller when using the Kinect based tracker (method explained in subsection 4.2.2 and implementation details in subsection 6.5.2) and then move onto evaluating waypoint flying. We mounted a Kinect to

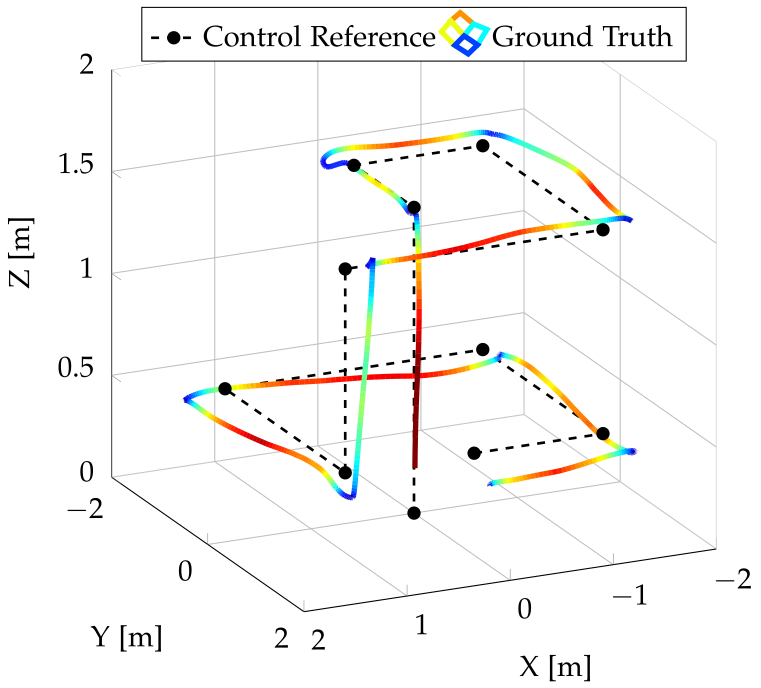


Figure 7.15: Flying waypoints distributed within the motion capture system

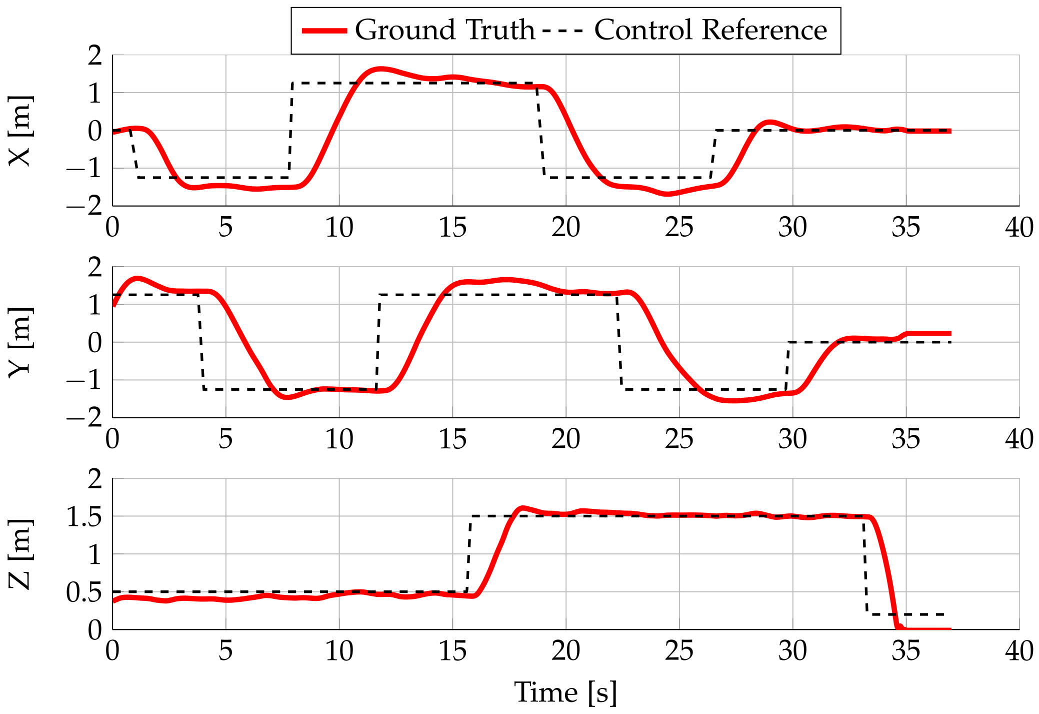


Figure 7.16: 2D visualisation of the different translation components when flying waypoints.

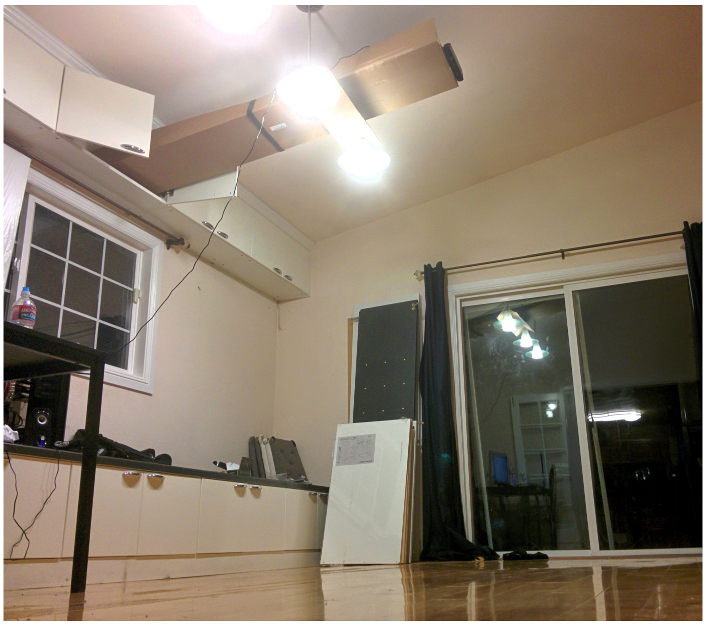


Figure 7.17: Kinect mounted to the ceiling for indoor position control at home.

the ceiling at a height of 2.5 m and had it facing downwards. See Figure 7.17 for a photo of the set-up.

Hover Stability

After generating a background model (which in this case is simply a flat plane - the floor) we started the tracker and let the Crazyflie hover at 1 m altitude in the center of the field of view. The flown trajectory is shown in 3D (Figure 7.18) and 2D (Figure 7.19).

RMSE's and maximum errors are summarised in Table 7.5. The system worked surprisingly well, with RMSE's very close to those from tests within the motion capture studio (Euclidean distance RMSEs of around 7.2 cm, 0.2 cm worse when compared to the motion capture studio based system). Note, that these values are not obtained from the motion tracker studio as the Kinect failed to work there (see section 7.2). However, we believe the results are accurate as we visually observed a very steady hovering performance during the experiment. Therefore, we conclude that one can safely the pose estimates from the Kinect based pose estimator to achieve very good hover control performance.

Flying Waypoints

We specified a list of waypoints within the Kinects field of view. The system tracked the Crazyflie without a single outlier; all pose estimates were reliable and lag free. Unfortunately the field of view is rather limited so the maximal step size we used was 1 m. The flown set of waypoints are visualised in Figure 7.20 and Figure 7.21.

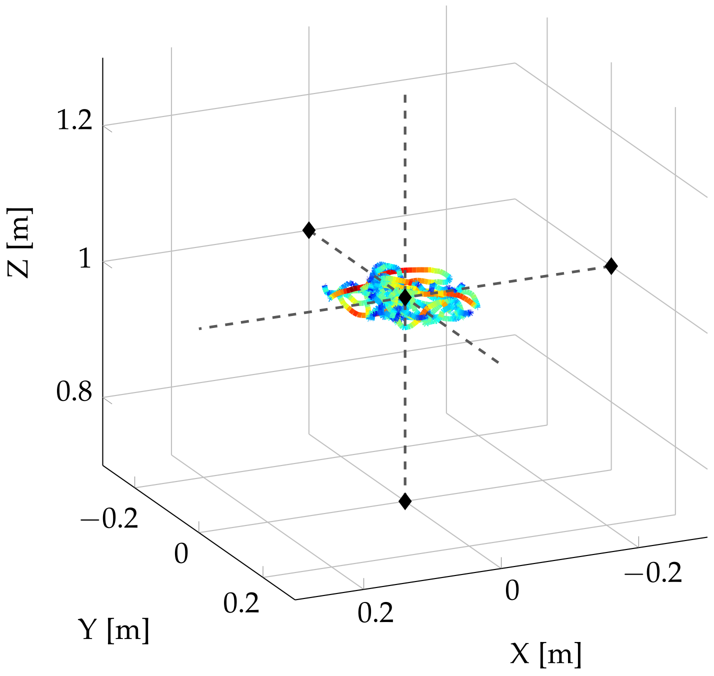


Figure 7.18: 3D visualisation of the flown trajectory when simply attempting to hover.

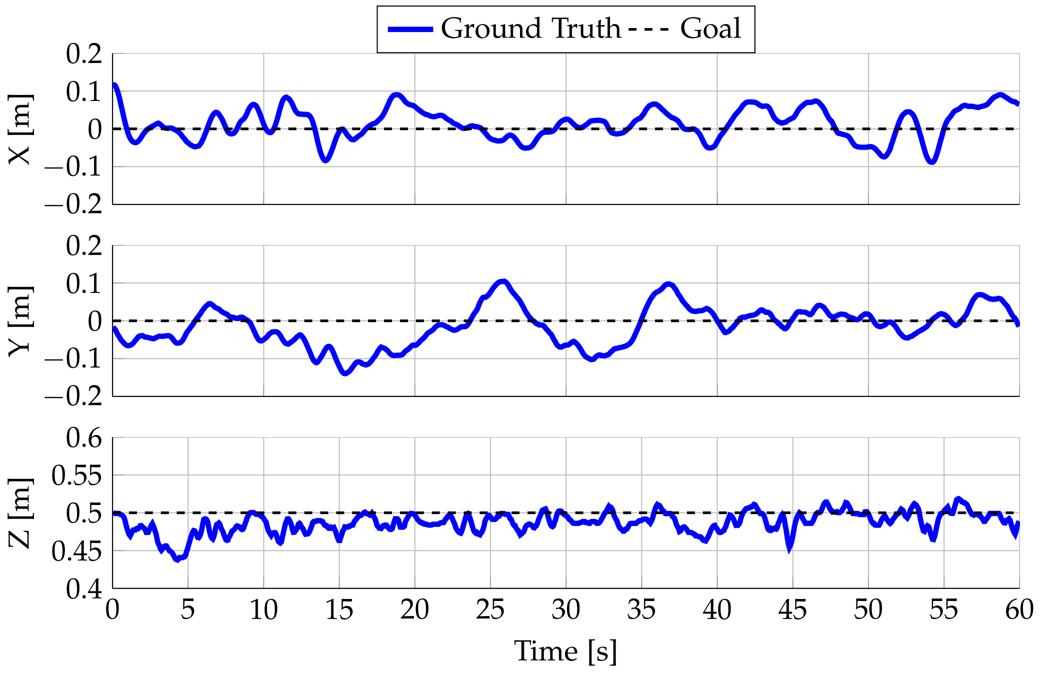


Figure 7.19: Hovering performance using the Kinect pose estimate. RMSEs around 7 cm

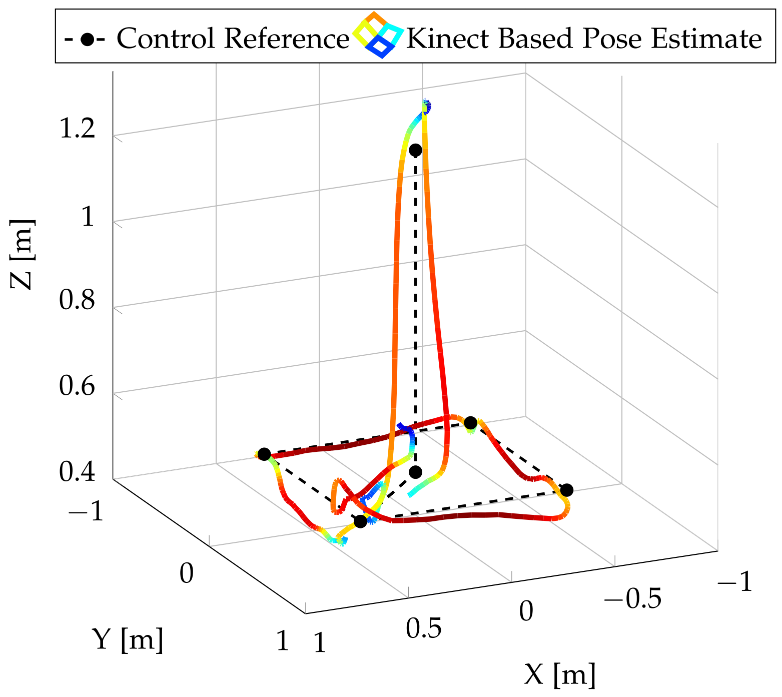


Figure 7.20: We specified waypoints to fly a $1m \times 1$m square followed by an altitude increase and decrease of 0.75 m. This shape was roughly what fit in the Kinect's field of view.

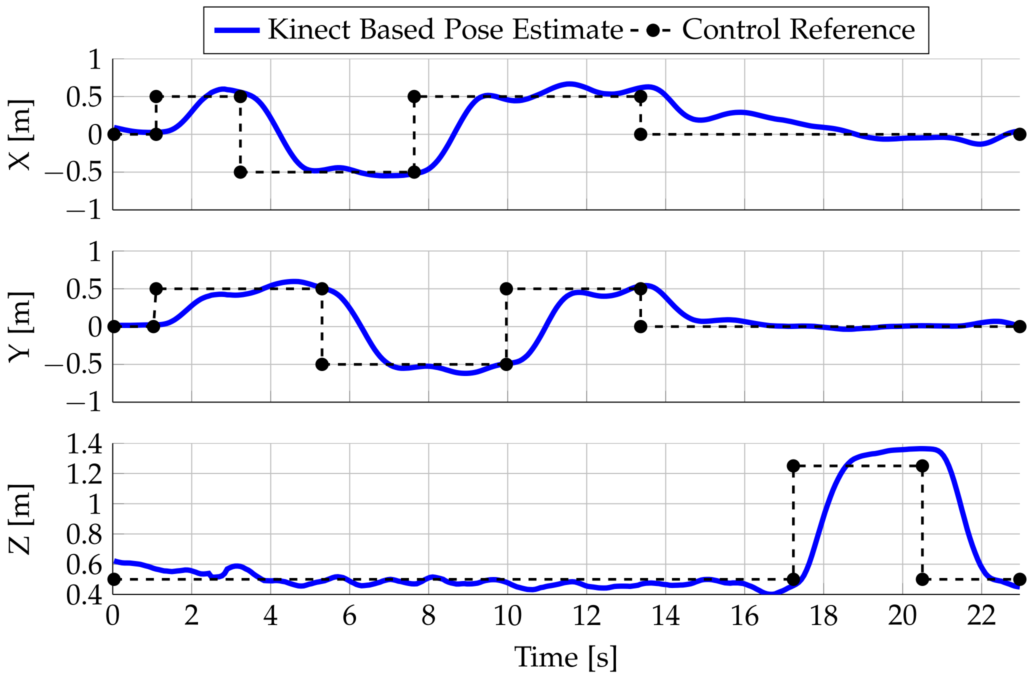


Figure 7.21: 2D visualisation of the goal and flown trajectory using the Kinect based pose estimator.

The results are extremely satisfying, as the quadrotor could rapidly fly from waypoint to waypoint with minimal overshoot and oscillation. The results are summarised in Table 7.7. Minimal overshoot is very important to avoid the quadrotor leaving the limited field of view. We conclude that the Kinect based system is a very attractive way to feed pose estimates in to the position controller. Using this system, uses can achieve accurate and responsive position control from the convenience of their home office.

Table 7.7: Autonomous Waypoint Following: Kinect Poses

Step Size [m]		**Rise-Time [s]**			**Overshoot [cm]**			**Convergence [s]**		
x, y	$\pm z$	x, y	$+z$	$-z$	x, y	$+z$	$-z$	x, y	$+z$	$-z$
1.0	0.75	0.92	1.81	0.89	11.2	16.4	6.8	0.28	0.89	0.66

7.4.3 VI-SLAM Tracking Based Control

In this section we evaluate how viable it is to directly use our unfiltered raw VI-SLAM pose estimates to control the Crazyflie. Aside from using cheap and lightweight hardware for the wireless camera system, we also have the additional disadvantage that we must carry extra weight. On one hand, this does make the control less responsive and therefore the quadrotor does become easier to control, but as we are on the verge of not having enough thrust to lift off (see section 7.1.1), PID control can become pretty unpredictable as it does not take thrust limitations into account.

As discussed in subsection 6.6.4, we use the barometer to estimate the scale. However, we use a different starting procedure to simplify running repetitive experiments. We manually hold the quadrotor in our hands and start the VI-SLAM system. This generates the initial keyframe. We then raise the quadrotor roughly a meter and keep it steady for long enough for the algorithm to detect that the barometer altitude has stabilised and the baseline is large enough. Using the debugging information we then confirm that the computed poses are roughly metric and that the system is online, after which we activate the controller while simultaneously removing our hand. The quadrotor stabilises after a quick altitude dip and we can then move to the desired position where we conduct our hover and waypoint experiments. We primarily use this method to prevent potential crashes in case the system initialisation fails. Generally initialisation was stable and worked roughly 9 out of 10 times.

Hover Stability

As usual, we first start off by verifying the quadrotor can hover. Once we have successfully launched the Crazyflie, we track its position with the motion capture studio. In Figure 7.22 and Figure 7.23 we plot the flown ground truth trajectory and the trajectory estimated by the VI-SLAM system.

The results were very satisfying. The quadrotor held its position, never deviating more than 25 cm from the goal position. RMSEs in the 10 cm-15 cm range are better than we

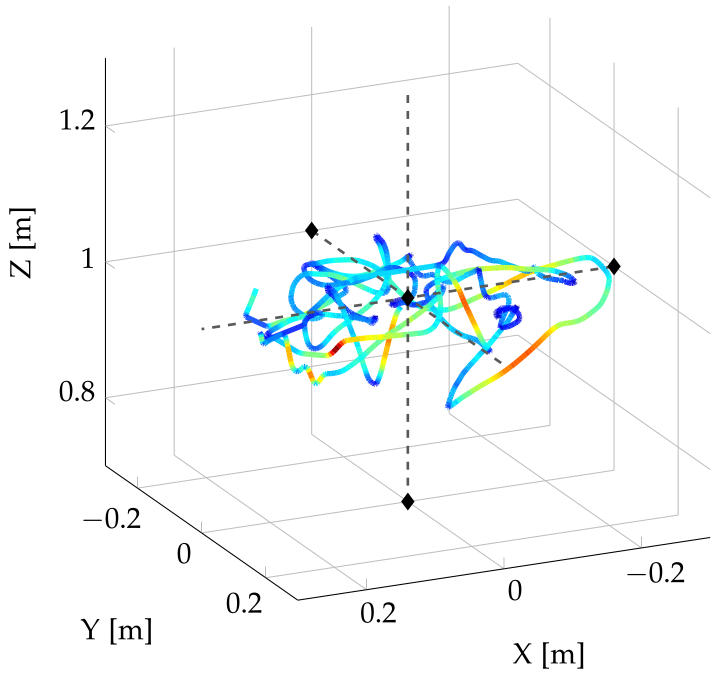


Figure 7.22: 3D visualisation of the Crazyflie hovering using the VI-SLAM pose estimate.

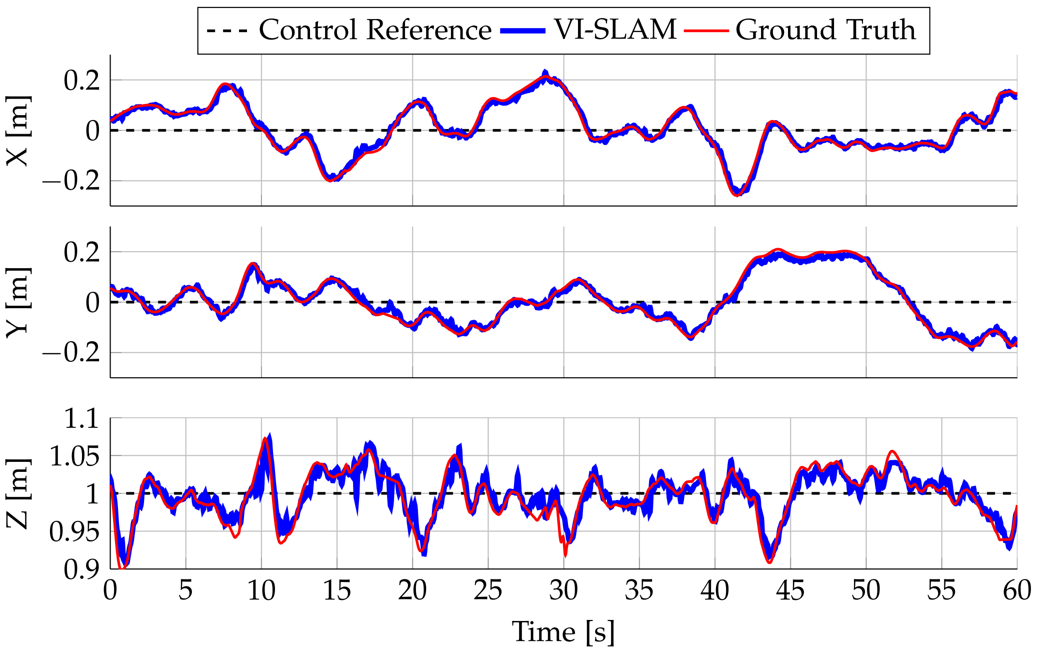


Figure 7.23: 2D visualisation of the Crazyflie hovering using the VI-SLAM pose estimate.

predicted they would be. Occasionally 5 or 6 frames in a row would suffer from transmission corruption which explains some of the spikes. We conclude that using VI-SLAM pose estimates is a viable option for drift free hover

Flying Waypoints

Here we evaluate waypoint flying using the VI-SLAM system for pose estimates. We fly a square shape as indicated in Figure 7.24 and Figure 7.25.

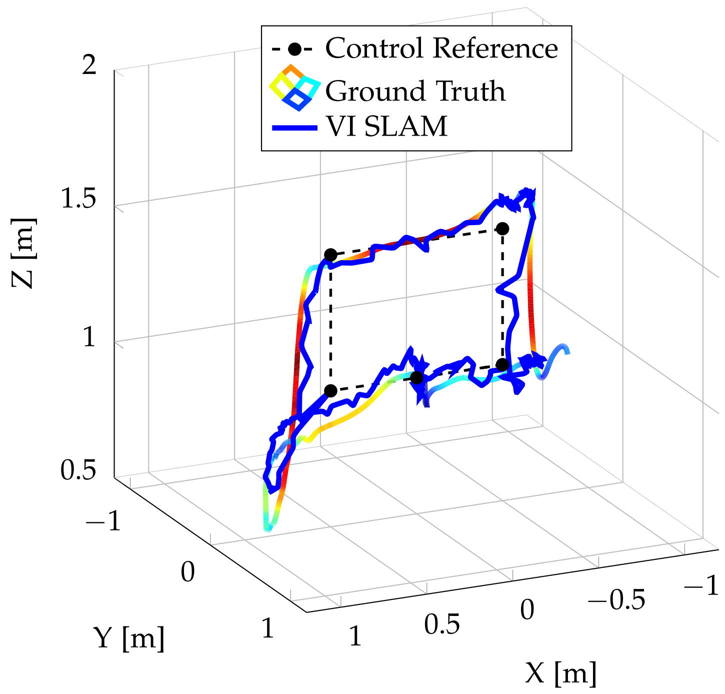


Figure 7.24: 3D visualisation of the estimated trajectory, flown trajectory and goal trajectory. When looking from this angle, the trajectory is counter-clockwise. Notice the large vertical overshoot on the descent.

The control evaluation results are shown in Table 7.8. Generally speaking we were are happy with the results. However, the overshoot on the vertical descent is disappointing but expected. As the Crazyflie was not purpose designed to carry and power a 5 g camera system, it suffers from a reduced thrust to weight ratio. The thrust is simply not enough to compensate for the negative vertical velocity.

We conclude that while position control using VI-SLAM pose estimates is certainly possible, the Crazyflie lacks the required thrust to have controlled descents. As a temporary work around, small step sizes can be used.

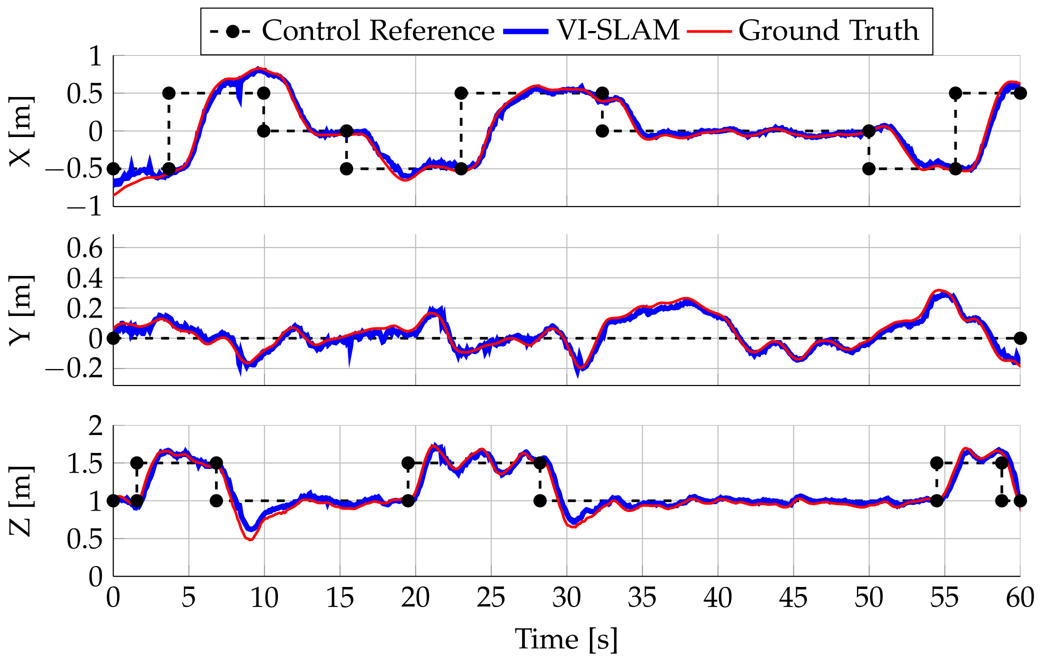


Figure 7.25: Following waypoints using the VI-SLAM estimated poses. The groundtruth and estimated trajectory are visualised on top of the waypoints. Notice that we plot 3 circuits, the first of which corresponds to the 3D visualisation above. The significant overshoot on the descents is clearly visible.

Table 7.8: Autonomous Waypoint Following: VI-SLAM Poses

Step Size [m]		**Rise-Time [s]**			**Overshoot [cm]**			**Convergence [s]**		
x	$\pm z$	x	$+z$	$-z$	x	$+z$	$-z$	x	$+z$	$-z$
1.0	0.5	4.26	2.01	2.09	24.9	31.7	3.89	1.66	1.42	3.41

8 Conclusion

In this thesis we presented an open-source and open-hardware 25 g nano-quadrotor with wireless video capability. We designed and implemented a visual-inertial SLAM system, a position controller, the required drivers and fused them into a single software stack that enables the quadrotor to perform autonomous drift-free hovering and waypoint following. To the best of our knowledge, we demonstrated the smallest and lightest quadrotor with such capability.

Additionally, we implemented a nano-quadrotor pose estimator using a Kinect camera, facilitating exteroceptive control in case a motion capture studio is not available.

Experiments were conducted to (*a*) prove that the systems do indeed work in practice, and (*b*) to evaulate the pose estimation accuracy and position control performance. Hovering RMSEs were in the range of 15 cm when using the on-board camera for pose estimation and half that when using the Kinect based approach. Waypoint following worked flawlessly when using the Kinect, but unfortunately the amount of spare thrust required for recovering from a descent while carrying and powering the camera system was a little too low. This resulted in significant overshoot when using the VI-SLAM based system to perform altitude reduction with step sizes over 50 cm. This problem could probably be mitigated with further controller tuning and by using motors with more thrust.

By providing open and user-friendly, ROS based access to the Crazyflie along with tools for pose estimation and position control, we hope to encourage and ease further research into the field of autonomous *nano*-quadrotors.

8.1 Future Work

There are a number of interesting research directions to augment and build upon the current system, which we will briefly discuss in this chapter. Most were simply not achievable due to time constraints.

Better Motion Control
While PID controllers are simple to implement and achieve average results with ease, they are probably not the best choice. It would be interesting to explore more advanced controllers, possibly incorporating a full motion model.

Add Particle/Kalman Filter to Kinect Tracking
Currently, the Kinect based pose estimator is as simple as it gets. One could do far

better if one would develop a motion model and perform tracking rather than just detection. A Kalman filter or particle filter would make a be a good fit.

Outdoor Tests
Further and more in-depth evaluation of the system is required. It would be especially interesting to see how it copes in the outdoors, especially with respect to longer trajectories and external disturbances such as wind.

Place Recognition
Currently our proposed VI-SLAM system lacks place recognition functionality. This would allow larger loops to be closed when adding keyframes to the map, even if significant drift had been accumulated. Practically speaking, all the required data structures and their contents are in place, so it would be relatively simple to incorporate loop recognition functionality into the propsed system. We would suggest exploring the DLoop place recogniser [31] as it was shown that it works well with binary descriptors we currently employ and could therefore be added with minimal effort.

Crazyflie 2.0
The next generation Crazyflie (photo of an early prototype is shown in Figure 8.1) is close to completion and it promises significantly more thrust at practically the same size. Th estimated 2-4 fold payload increase helps overcome the limited thrust problems we experienced while descending with the camera system of the first Crazyflie generation. Furthermore, the new design will greatly ease the process of adding a camera and the new shape results in far less self occlusion.

More Onboard Computation
Ideally, one would move more and more functionality onto the device itself as computational capabilities increase. This reduces overall complexity and eliminates problems caused by communication delay. Furthermore, sensor measaurements could be used at a higher frequency as the communication throughput bottleneck is removed. The next generation Crazyflie with its STM32F4 is a step in the right direction.

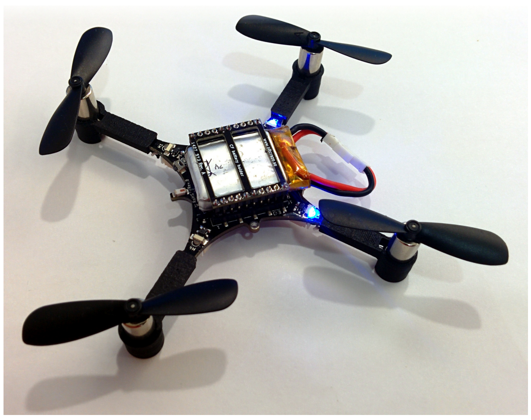


Figure 8.1: A prototype of the next generation Crazyflie. With a revised shape, 20dBm RF power amplifier and much more thrust, it is a significant improvement over the first generation, especially with respect to carrying and powering an additional camera.

Bibliography

[1] D. Mellinger, N. Michael, and V. Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. In *Proc. of the International Symposium on Experimental Robotics (ISER)*, 2010.

[2] D. Mellinger and V. Kumar. Minimum snap trajectory generation and control for quadrotors. In *Proc. of the International Conference on Robotics and Automation (ICRA)*, 2011.

[3] S. Weiss, M. Achtelik, S. Lynen, M. Achtelik, L. Kneip, M. Chli, and R. Siegwart. Monocular vision for long-term micro aerial vehicle state estimation: A compendium. *Journal of Field Robotics*, 30(5):803–831, 2013.

[4] L. Meier, P. Tanskanen, L. Heng, G. Lee, F. Fraundorfer, and M. Pollefeys. Pixhawk: A micro aerial vehicle design for autonomous flight using onboard computer vision. *Autonomous Robots*, 2012.

[5] J. Engel, J. Sturm, and D. Cremers. Scale-aware navigation of a low-cost quadrocopter with a monocular camera. *Robotics and Autonomous Systems (RAS)*, 2014.

[6] A. Briod, J. Zufferey, and D. Floreano. Optic-flow based control of a 46g quadrotor. In *Vision-based Closed-Loop Control and Navigation of Micro Helicopters in GPS-denied Environments, IROS*, 2013.

[7] C. de Wagter, S. Tijmons, B. Remes, , and G. de Croon. Autonomous flight of a 20-gram flapping wing mav with a 4-gram onboard stereo vision system. In *ICRA*, 2014.

[8] Christian Forster, Matia Pizzoli, and Davide Scaramuzza. Svo: Fast semi-direct monocular visual odometry. In *Proc. IEEE Intl. Conf. on Robotics and Automation*, 2014.

[9] R. Moore, K. Dantu, G. Barrows, and R. Nagpal. Autonomous mav guidance with a lightweight omnidirectional vision sensor. In *ICRA*, 2014.

[10] Warren R Young. *Helicopters*. Epic of Flight. Time Life Education, Connecticut, USA, 1982.

[11] Parrot. Parrot establishes itself on the civil drones market, June 2013.

[12] Bitcraze AB. Bitcraze ab web platform, 2011.

[13] J Gordon Leishman. *Principles of Helicopter Aerodynamics with CD Extra*. Cambridge university press, 2006.

[14] Paul Marcel Lambermont. *Helicopters and autogyros of the world*. London: Cassell, 1958.

[15] Chia-Kai Liang, Li-Wen Chang, and Homer H Chen. Analysis and compensation of rolling shutter effect. *Image Processing, IEEE Transactions on*, 17(8):1323–1330, 2008.

[16] Alexandre Karpenko, David Jacobs, Jongmin Baek, and Marc Levoy. Digital video stabilization and rolling shutter correction using gyroscopes. *CSTR*, 1:2, 2011.

[17] Telekommunikation Post und Eisenbahnen Bundesnetzagentur fuer Elektrizitaet, Gas. Schnittstellenbeschreibung für funkanlagen fuer nichtoeffentliche allgemeine funkanwendungen geringer reichweite, 2005.

[18] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam, 2014.

[19] Formlabs. Formlabs form 1, 2014. [`http://formlabs.com/products/form-1/`].

[20] L. Kneip, R.Y. Siegwart, and M. Pollefeys. *Real-time Scalable Structure from Motion: From Fundamental Geometric Vision to Collaborative Mapping*. ETH, 2012.

[21] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping: part i. *Robotics & Automation Magazine, IEEE*, 13(2):99–110, 2006.

[22] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (slam): Part ii. *IEEE Robotics & Automation Magazine*, 13(3):108–117, 2006.

[23] Niklas Karlsson, Enrico Di Bernardo, Jim Ostrowski, Luis Goncalves, Paolo Pirjanian, and Mario E Munich. The vslam algorithm for robust localization and mapping. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 24–29. IEEE, 2005.

[24] Hans P Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical report, DTIC Document, 1980.

[25] Larry Matthies and Steven A Shafer. Error modeling in stereo navigation. *Robotics and Automation, IEEE Journal of*, 3(3):239–248, 1987.

[26] Jack Langelaan and Steve Rock. Passive gps-free navigation for small uavs. In *Aerospace Conference, 2005 IEEE*, pages 1–9. IEEE, 2005.

[27] Georg Klein and David Murray. Parallel tracking and mapping on a camera phone. In *Mixed and Augmented Reality, 2009. ISMAR 2009. 8th IEEE International Symposium on*, pages 83–86. IEEE, 2009.

[28] Gabriel Nützi, Stephan Weiss, Davide Scaramuzza, and Roland Siegwart. Fusion of IMU and vision for absolute scale estimation in monocular slam. *Journal of intelligent & robotic systems*, 61(1-4):287–299, 2011.

[29] M. Achtelik, M. Achtelik, S. Weiss, and R. Siegwart. Onboard IMU and monocular vision based control for MAVs in unknown in- and outdoor environments. In *Proc. of the International Conference on Robotics and Automation (ICRA)*, 2011.

[30] Todd Lupton and Salah Sukkarieh. Removing scale biases and ambiguity from 6dof monocular slam using inertial. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 3698–3703. IEEE, 2008.

[31] Dorian Galvez-Lopez and J. D. Tardos. Bags of binary words for fast place recognition in image sequences. *IEEE Transactions on Robotics*, 28(5):1188–1197, October 2012.

[32] Mark Cummins and Paul Newman. Appearance-only slam at large scale with fab-map 2.0. *The International Journal of Robotics Research*, 30(9):1100–1123, 2011.

[33] G. Klein and D. Murray. Parallel tracking and mapping for small AR workspaces. In *Proc. of the International Symposium on Mixed and Augmented Reality (ISMAR)*, 2007.

[34] JMM Montiel, Javier Civera, and Andrew J Davison. Unified inverse depth parametrization for monocular slam. *analysis*, 9:1, 2006.

[35] David Nistér. An efficient solution to the five-point relative pose problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 26(6):756–770, 2004.

[36] H. Strasdat, A. Davison, J. Montiel, and K. Konolige. Double window optimisation for constant time visual SLAM. In *Proc. of the International Conference on Computer Vision (ICCV)*, 2011.

[37] M. Montemerlo, S. Thrun, D. Koller, and B. Wegbreit. FastSLAM: A factored solution to the simultaneous localization and mapping problem. In *Proc. of the AAAI National Conference on Artificial Intelligence*, 2002.

[38] Andrew J Davison, Ian D Reid, Nicholas D Molton, and Olivier Stasse. Monoslam: Real-time single camera slam. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 29(6):1052–1067, 2007.

[39] Michael Kaess, Hordur Johannsson, Richard Roberts, Viorela Ila, John J Leonard, and Frank Dellaert. isam2: Incremental smoothing and mapping using the bayes tree. *The International Journal of Robotics Research*, page 0278364911430419, 2011.

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

[41] Maxime Lhuillier. Automatic structure and motion using a catadioptric camera. In *Proceedings of the 6th Workshop on Omnidirectional Vision, Camera Networks and Non-Classical Cameras*, 2005.

[42] Davide Scaramuzza and Friedrich Fraundorfer. Visual odometry: Part i: The first 30 years and fundamentals. *Robotics & Automation Magazine, IEEE*, 18(4):80–92, 2011.

[43] Friedrich Fraundorfer and Davide Scaramuzza. Visual odometry: Part ii: Matching, robustness, optimization, and applications. *Robotics & Automation Magazine, IEEE*, 19(2):78–90, 2012.

[44] Berthold K Horn and Brian G Schunck. Determining optical flow. In *1981 Technical Symposium East*, pages 319–331. International Society for Optics and Photonics, 1981.

[45] J. Engel, J. Sturm, and D. Cremers. Semi-dense visual odometry for a monocular camera. In *IEEE International Conference on Computer Vision (ICCV)*, Sydney, Australia, December 2013.

[46] J. Engel, T. Schöps, and D. Cremers. LSD-SLAM: Large-scale direct monocular SLAM. In *European Conference on Computer Vision (ECCV)*, September 2014.

[47] T. Schöps, J. Engel, and D. Cremers. Semi-dense visual odometry for AR on a smartphone. In *International Symposium on Mixed and Augmented Reality (ISMAR)*, September 2014.

[48] Ondrej Miksik and Krystian Mikolajczyk. Evaluation of local detectors and descriptors for fast feature matching. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 2681–2684. IEEE, 2012.

[49] Adam Schmidt, Marek Kraft, and Andrzej Kasiński. An evaluation of image feature detectors and descriptors for robot navigation. In *Computer Vision and Graphics*, pages 251–259. Springer, 2010.

[50] Steffen Gauglitz, Tobias Höllerer, and Matthew Turk. Evaluation of interest point detectors and feature descriptors for visual tracking. *International journal of computer vision*, 94(3):335–360, 2011.

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

[52] H. Bay, T. Tuytelaars, and L.V. Gool. SURF: Speeded-up robust features. In *Proc. of the European Conference on Computer Vision (ECCV)*, 2008.

[53] D. Lowe. Object recognition from local scale-invariant features. In *Proc. of the International Conference on Computer Vision (ICCV)*, 1999.

[54] Pablo F Alcantarilla, Jesús Nuevo, and Adrien Bartoli. Fast explicit diffusion for accelerated features in nonlinear scale spaces. *Trans. Pattern Anal. Machine Intell*, 34(7):1281–1298, 2011.

[55] C. Harris and M. Stephens. A combined corner and edge detector. In *Proc. of the Alvey Vision Conference*, 1988.

[56] Stephen M Smith and J Michael Brady. Susana new approach to low level image processing. *International journal of computer vision*, 23(1):45–78, 1997.

[57] E. Rosten and T. Drummond. Machine learning for high-speed corner detection. In *Proc. of the European Conference on Computer Vision (ECCV)*, 2006.

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

[59] J. Shi and C. Tomasi. Good features to track. In *Proc. of the Conference on Computer Vision and Pattern Recognition (CVPR)*, 1994.

[60] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. Orb: an efficient alternative to sift or surf. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2564–2571. IEEE, 2011.

[61] Edward Rosten and Tom Drummond. Fusing points and lines for high performance tracking. In *IEEE International Conference on Computer Vision*, volume 2, pages 1508–1511, 2005.

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

[63] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In *Computer Vision–ECCV 2010*, pages 778–792. Springer, 2010.

[64] Stefan Leutenegger, Margarita Chli, and Roland Yves Siegwart. Brisk: Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555. IEEE, 2011.

[65] Alexandre Alahi, Raphael Ortiz, and Pierre Vandergheynst. Freak: Fast retina keypoint. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 510–517. Ieee, 2012.

[66] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(10):1615–1630, 2005.

[67] Gang Hua, Matthew Brown, and Simon Winder. Discriminant embedding for local image descriptors. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[68] Tinne Tuytelaars and Cordelia Schmid. Vector quantizing feature space with a regular lattice. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE, 2007.

[69] 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.

[70] Frédréric Devernay and Olivier Faugeras. Straight lines have to be straight: Automatic calibration and removal of distortion from scenes of structured enviroments. *Mach. Vision Appl.*, 13(1):14–24, August 2001.

[71] Laurent Kneip and Paul Furgale. Opengv: A unified and generalized approach to real-time calibrated geometric vision. In *Proc. of the International Conference on Robotics and Automation (ICRA)*, 2014.

[72] Georg Klein and David Murray. Improving the agility of keyframe-based slam. In *Computer Vision–ECCV 2008*, pages 802–815. Springer, 2008.

[73] L Kneip, M Chli, and R Siegwart. Robust real-time visual odometry with a single camera and an imu. In *Proc. of The British Machine Vision Conference (BMVC)*, Dundee, Scotland, August 2011.

[74] Jorge Lobo and Jorge Dias. Relative pose calibration between visual and inertial sensors. *The International Journal of Robotics Research*, 26(6):561–575, 2007.

[75] Dave Zachariah and Magnus Jansson. Joint calibration of an inertial measurement unit and coordinate transformation parameters using a monocular camera. In *Indoor Positioning and Indoor Navigation (IPIN), 2010 International Conference on*, pages 1–7. IEEE, 2010.

[76] E. Kruppa. Zur Ermittlung eines Objektes aus zwei Perspektiven mit innerer Orientierung. *Sitzungsberichte der Mathematisch Naturwissenschaftlichen Kaiserlichen Akademie der Wissenschaften*, 122:1939–1948, 1913.

[77] H. Li and R. Hartley. Five-point motion estimation made easy. In *Proc. of the International Conference on Pattern Recognition (ICPR)*, 2006.

[78] H. Stewénius, C. Engels, and D. Nistér. Recent developments on direct relative orientation. *Journal of Photogrammetry and Remote Sensing (ISPRS)*, 60(4):284 – 294, 2006.

[79] Henrik Stewénius, David Nistér, Magnus Oskarsson, and Kalle Åström. Solutions to minimal generalized relative pose problems. In *Workshop on omnidirectional vision*, 2005.

[80] R. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, 2004.

[81] Hongdong Li, Richard Hartley, and Jae-hak Kim. A linear approach to motion estimation using generalized camera models. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[82] 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.

[83] J. Grunert. Das pothenotische problem in erweiterter gestalt nebst über seine anwendungen in der geodäsie. *Grunerts Archiv für Mathematik und Physik*, 1941.

[84] Xiao-Shan Gao, Xiao-Rong Hou, Jianliang Tang, and Hang-Fei Cheng. Complete solution classification for the perspective-three-point problem. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(8):930–943, 2003.

[85] Donald W Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *Journal of the Society for Industrial & Applied Mathematics*, 11(2):431–441, 1963.

[86] Yi Ma. *An invitation to 3-d vision: from images to geometric models*, volume 26. springer, 2004.

[87] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. Bundle adjustmenta modern synthesis. In *Vision algorithms: theory and practice*, pages 298–372. Springer, 2000.

[88] R. Kümmerle, G. Grisetti, H. Strasdat, K. Konolige, and W. Burgard. g2o: A general framework for graph optimization. In *Proc. of the International Conference on Robotics and Automation (ICRA)*, 2011.

[89] Manolis IA Lourakis and Antonis A Argyros. Sba: A software package for generic sparse bundle adjustment. *ACM Transactions on Mathematical Software (TOMS)*, 36(1):2, 2009.

[90] S Madgwick. An efficient orientation filter for inertial and inertial/magnetic sensor arrays. *Report x-io and University of Bristol (UK)*, 2010.

[91] Stuart Bennett. *A history of control engineering, 1930-1955*. Number 47. IET, 1993.

[92] Yun Li, Kiam Heong Ang, and Gregory CY Chong. Patents, software, and hardware for pid control: an overview and analysis of the current art. *Control Systems, IEEE*, 26(1):42–54, 2006.

[93] JG Ziegler and NB Nichols. Optimum settings for automatic controllers. *trans. ASME*, 64(11), 1942.

[94] Jinghua Zhong. Pid controller tuning: A short tutorial. *class lesson), Purdue University*, 2006.

[95] Kiam Heong Ang, Gregory Chong, and Yun Li. Pid control system analysis, design, and technology. *Control Systems Technology, IEEE Transactions on*, 13(4):559–576, 2005.

[96] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully B. Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.

[97] Gunilla Borgefors. Distance transformations in digital images. *Computer vision, graphics, and image processing*, 34(3):344–371, 1986.

[98] Gaël Guennebaud, Benoît Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[99] Teodor Lucian Grigorie, Liviu Dinca, Jenica-Ileana Corcau, and Otilia Grigorie. Aircrafts' altitude measurement using pressure information: barometric altitude and density altitude. *WSEAS Transactions on Circuits and Systems*, 9(7):503–512, 2010.