Hovedoppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for teknisk kybernetikk

Oskar Gjesdal Veggeland

Real-Time Vision-Aided Inertial Navigation for Vertical Take-Off and Landing of Unmanned Aerial Vehicles

Hovedoppgave i MTTK - Kybernetikk og Robotikk
Veileder: Konstantinos Alexis
Medveileder: Paolo de Petris
Juni 2022

NTNU
Norwegian University of
Science and Technology

Oskar Gjesdal Veggeland

Real-Time Vision-Aided Inertial Navigation for Vertical Take-Off and Landing of Unmanned Aerial Vehicles

Hovedoppgave i MTTK - Kybernetikk og Robotikk
Veileder: Konstantinos Alexis
Medveileder: Paolo de Petris
Juni 2022

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for teknisk kybernetikk

NTNU

Kunnskap for en bedre verden

Preface

This report is the delivery for my master thesis, TTK4900, at the institute of engineering cybernetics at NTNU in the spring of 2022. Throughout the semester I have been supervised by professor Konstantinos Alexis, with help from Ph.D. student Paolo De Petris. With bi-weekly meetings, they have helped me plan my work, identify problems and interpret results.

Most of the software used in the thesis was found online. The exceptions are the ROS node used to synchronize cameras and IMU, the algorithm extension to ROVIO and all software used to produce plots and figures. In addition to this, some modifications have been made to the pre-made software. Any such borderline cases should be clearly indicated in the thesis. All figures, images and other forms of visual representations are created specifically for the thesis.

My visual inertial odometry setup was created with help from the institutes workshop. They made the aluminium bar with screw threads so the camera positions could be easily adjusted. From a base design received by Paolo, the camera mounts was re-designed to fit the aluminium rod. These mounts were 3D-printed at the workshop of MAKE-NTNU.

The paper is written such that anyone with a background from cybernetics and robotics should comprehend the content, although other papers may be referred to for more detailed information or explanations. Good reading.

Table of Contents

List of Figures

Abstract

This thesis presents an analysis of Visual Inertial Odometry (VIO) as a state estimation solution for autonomous fixed wing aircraft during vertical take-off and landing (VTOL). A theoretical foundation of VIO is presented together with some of its properties related to aerial applications. Later, an extensive description of a specific framework, called ROVIO, is given. This framework has been tested on a hardware setup for stereo VIO built for collecting data for the thesis. The setup is presented in detail together with the calibration approach used.

Using this setup, different experiments have been conducted to analyze how the systems performance may be affected by different parameters, motions and configurations. These experiments are mostly related to the long distance to visual features in a typical VTOL scenario. The analysis can conclude, not surprisingly, that the initial depth value given to image features have a significant effect on the system accuracy. It also becomes clear that initializing the system trajectory with sufficient movement can make feature depths converge more quickly, reducing the overall error. Investigating the performance with different camera distances (baselines), revealed no clear correlation. This is likely due to quite apparent calibration issues, probably resulting from frequent camera remounts. Finally, an analysis of the difference between stereo and monocular systems is conducted, showing advantages and disadvantages of either configuration.

The most important contribution of the thesis is a proposed extension to ROVIO, which builds on an assumption of the operating environment in a VTOL scenario. By assuming a flat landing area, the algorithm has been extended to estimate the position of this plane. With this estimate, features can be filtered out if they lie sufficiently far away. Other features can be updated in the direction of the ground estimate, to improve their position accuracy. Details of the extension is presented in the report, together with promising results of its performance. Although this specific extension is tailor-made for ROVIO, the general concept should be applicable for any VIO algorithm.

1 Introduction

This master thesis will investigate a possible state estimator used during vertical take-off and landing (VTOL) for unmanned aerial vehicles (UAVs). The topic is motivated by the Norwegian drone transportation startup Aviant, which transports goods between hospitals in the medical sector of Norway. Medical goods can be transported faster, cheaper, and at shorter notice between medical institutions with such transportation options. There are a few restrictions for small unmanned vehicles that should be imposed on the system. The first thing to think about is that the system's size and weight restrict the total vehicle cargo capacity. This means that a small, lightweight system is preferred to increase the payload. The issue extends to the fact that power consumption should be kept to a minimum, as high-capacity batteries are large and heavy. Lastly, since this is a commercial company with expectations of profit, the price of a system should be kept to a minimum.

The proposed solution is an inertial navigation system (INS) that uses a stereo camera system to perform filter updates. These systems are often referred to as visual-inertial odometry (VIO), a highly researched area for autonomous systems. One of the reasons for this is that it is highly versatile and capable of operating in different working environments. Because VIO does not rely on any external sensors or equipment, it is not restricted to a geographical area. This yields opportunities for exploration underground[23], subsea[39] and even in outer space[21] where global navigation satellite systems (GNSS) are unavailable. Although the application in question for this thesis is not deprived of access to GNSS measurements, there is another motivation to avoid using them. In urban areas with dense construction, GNSS measurements become less reliable. A common issue is that satellite signals can bounce between buildings before reaching the sensor, corrupting the measurements as they are calculated based on a signal's travel time and distance.

Although VIO has the advantage of not relying on external sources, this does not come without a drawback. The introduction of GNSS measurements makes the absolute position of a system observable. In contrast, VIO systems suffer from a drifting position estimate unless corrected by a place recognition or re-localization scheme. Another issue with VIO is the dependency on visual information. The thesis will focus mainly on images in the visible specter, but there is research to support that VIO can work well with thermal imaging as well[14]. The most important thing is that the camera can capture information about the scene, including texture, edges, and different kinds of features. This is a highly relevant issue because lighting conditions in a VTOL scenario can be unpredictable.

As part of the thesis, a VIO system consisting of two cameras and an IMU has been created. With this setup, data has been collected from several scenarios resembling VTOL. Without an actual drone for testing, the problem has been flipped such that big walls simulate the ground in a take-off and landing scenario. By walking up to and back from the wall, one can analyze how the VIO system reacts to a scene with varying depths, stretching to as much as 25 meters in some of the experiments. The open-source framework for VIO, ROVIO[4] has been used to process the collected data. The data is analyzed in different experiments to look at how different configurations and parameters affect the system performance. This includes looking at different initial depth estimates, initial camera movements, different camera baselines, and the difference between monocular and stereo systems. Finally, an adaption to ROVIO is presented where the system assumes that the landing area is flat, yielding the opportunity to disregard or correct outliers that are not predicted to be on the ground plane.

2 Theory

This section will provide some insight into the field of visual inertial odometry. First, a review of the camera sensor and its ability to estimate motion from an image series is given. Next, the fusion of visual and inertial measurement will be presented together with pros and cons of VIO systems. A short introduction to stereo camera triangulation will be given before a thorough review of a state-of-the-art VIO system called ROVIO is presented. ROVIO is the system used to perform experiments in section 6, so a conceptual understanding of the framework is essential to analyze the produced results. Although modifications have been made, large parts of this section is taken from the author's unpublished master project.

2.1 Visual Odometry

Visual odometry (VO) is the problem of estimating the motion between two camera frames by associating changes in the image with a change in the camera perspective. Most VO frameworks can be related to the pipeline illustrated in fig. 1. The first procedure, called feature detection, is based on finding unique features in an image. This is usually points in the image with high gradients such as corners or edges. After detection, the features are tracked, meaning they are associated between images. These tracks can be used to estimate the frame-to-frame motion of the camera. Finally, we can optimize several of these motions jointly, considering that feature tracks can span over multiple camera frames. This procedure is called smoothing and can considerably improve the global consistency of a VO system.

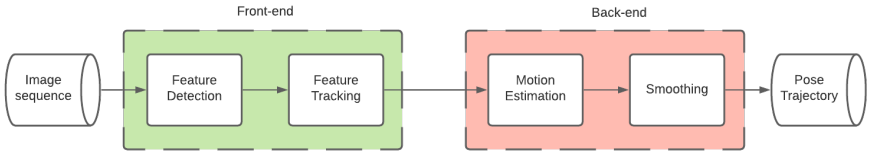


Figure 1: A general VO pipeline, estimating a set of camera poses from a sequence of images.

In visual odometry, it is customary to separate image and feature related processing from the estimation and optimization. The two parts are referred to as the system's front-end and back-end. There exists a wide variety of front-end and back-end schemes and countless combinations of the two. For simplicity, the front-end frameworks will be divided into two different categories: indirect and direct methods. The indirect methods perform feature detection in every new image and try to match these features to those of previous images. On the other hand, direct methods do not perform this feature detection for every new image. These methods assume that previous features can be located in a new image by searching for similar areas based on pixel intensities. The difference between these methods will be further explained in section 2.1.2.

2.1.1 Camera modelling

Like any other sensor, we need to define a model for how a camera interprets its surroundings to create an image. The most commonly used model, which is also used in this thesis, is the perspective camera model. This can be used for front-view cameras with a field of view below 180°. The perspective camera model, often called the pinhole model, assumes that all light that reaches the image sensor goes through a singular point (pinhole) in the camera. It further assumes that light moves in a straight line from any object in a scene through this pinhole. The key idea of modeling the camera is to create a mapping that takes a three-dimensional point from the camera frame $\boldsymbol{X} = [x, y, z]$ and projects it onto the image frame with coordinates [u, v]. If we define the camera frame such that x is positive to the right, y is positive downwards, and z is positive into the scene, the camera projection can be defined as

$$\lambda \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{K}\boldsymbol{X} = \begin{bmatrix} \alpha_u & 0 & u_0 \\ 0 & \alpha_v & v_0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{1}$$

where $(\alpha_u, \alpha_v, u_0, v_0)$ are the camera intrinsic parameters and λ is a scaling parameter used to normalize the image coordinates. (α_u, α_v) are defined as the distance between the camera pinhole and the image sensor. (u_0, v_0) is the coordinates of the image center. Although the intrinsic parameters have clearly defined physical interpretations, these values should be calibrated for each camera because of constructional inaccuracies and modelling errors.

Distortion

Distortion effects result from inaccuracies in the camera model and often appear as morphing effects in the image. The perspective camera model is usually best in the center of an image and becomes worse further away. To apply motion tracking on features in the entire image, it is crucial to account for this deviation. Solutions to the problem rely on mathematical descriptions of the distorting effect and are summarized in a distortion model. Tang et al. define the distortion problem as a mapping f from the undistorted projection of the perspective model (u_u, v_u) to the distorted and faulty projection (u_d, v_d) which we observe in a raw image[37]. If this distortion model is found, the solution to the problem would be to apply the reverse mapping g such that

$$u_u = g_u(u_d, v_d), \quad v_u = g_v(u_d, v_d) \tag{2}$$

In theory, these models can include any mathematical description, but common choice is a radial model. Radial models define the distortion effect based on a pixels distance from a distortion center, (u_0, v_0). I will denote the distance on each axis as $\bar{u}_u = u_u - u_0$, $\bar{v}_u = v_u - v_0$ and $\bar{u}_d = u_d - u_0$, $\bar{v}_d = v_d - v_0$ in the undistorted and distorted cases respectively. The radial assumption reduces the distortion model to a one-dimensional mapping from the distorted radius $r_d = \sqrt{\bar{u}_d^2 + \bar{v}_d^2}$ to the undistorted radius $r_u = \sqrt{\bar{u}_u^2 + \bar{v}_u^2}$. While this is very interpretable, it cannot always account for all the distortion effects in a regular camera. A proposed improvement is to add tangential distortion, which accounts for misalignment between the image sensor and camera lens. This makes up for some of the non-radially symmetric distortion effects, and the undistortion model can be written

$$u_u = u_d + \overbrace{u_d(k_1 r_d^2 + k_2 r_d^4)}^{\text{Radial distortion}} + \overbrace{2p_1 u_d v_d + p_2(r_d^2 + 2u_d^2)}^{\text{Tangential distortion}} \tag{3}$$

$$v_u = v_d + v_d(k_1 r_d^2 + k_2 r_d^4) + 2p_2 u_d v_d + p_1(r_d^2 + 2v_d^2) \tag{4}$$

where k_1, k_2 and p_1, p_2 are distortion parameters and should be calibrated for each individual camera. Higher-order terms can also be included with the addition of more distortion parameters. The combination of radial and tangential distortion models is chosen for experiments with ROVIO in this thesis.

One easy way to spot distortion in images is to look at how straight lines are projected to the image. They tend to appear curved in distorted images. Figure 2 illustrates what happens when we perform undistortion. Note how the straight edges on the door appear bent in the distorted image (left). The undistorted image (right) almost completely removes this unwanted effect.

2.1.2 Front-end

Two main types of front-end frameworks will be considered, indirect and direct methods. Common for both is that they try to associate features from one image to another. Indirect methods perform

Figure 2: Comparing a raw, undistorted image (left) with the effect of applying the radial and tangential undistortion model (right).

feature detections on every single frame. A similarity search over features in a previous image is performed to look for matches for each feature in a new image. The direct methods do however not rely on feature detection for new images. Instead, these methods perform an alignment of the previous features in a new image. By looking at the pixel intensities of a feature, a search in the new image is performed to find the best match of a feature. If the match is good enough, the feature track is accepted. Although direct methods do not generally perform feature detection for every new image, some do this to initialize new features for tracking. These are called semi-direct methods and include, among others, ROVIO.

Feature detection

Feature detection is the procedure of finding points in an image that are easily identifiable and highly likely to be tracked between frames. Many scenes contain corners, edges, and other areas with big changes in pixel intensities. These points are usually easy to associate from image to image because of the big difference in image intensities around the feature, as opposed to uniform surfaces with little texture. Nixon describes two main procedures of extracting local features in his book on feature extraction in computer vision[25]. He argues that finding corners in an image is the same as finding edges with high curvature, meaning areas with a high rate of change in the edge direction. Since the edge itself is defined by a high gradient opposite the edge direction, areas of a high gradient in two opposite directions can be related to a corner. This is the fundamental assumption for the famous Harris corner detector [12], which analyzes the image gradients to extract corners and edges. Nixon also describes a more modern approach based on the analysis of image patches, that is, smaller subsets of an image. These methods can deal with issues that the basic curvature methods are not. For example, they can be created scale-invariant, meaning that a feature can be of any size in the image and still be detected.

An example of a patch-based feature detection scheme is FAST [31], which is used in ROVIO as a feature initialization tool. FAST, or *Features from Accelerated Segmentation Test*, was proposed as a high-speed detector, trading down the feature quality for a better runtime. A test is performed on a query pixel $\boldsymbol{p}$ by looking at 16 pixels surrounding it in a circle. If at least 12 of the 16 pixels have an intensity that differs more than some threshold from the query pixel, the pixel is registered as a feature. The first four pixels to be tested are straight above, to the left, right, and below the query pixel to further increase the speed. If two or more of these pixels are not below or above the threshold, the query pixel is immediately rejected as a corner, avoiding the need to test all 16 values.

Indirect tracking

To perform tracking between images, indirect methods rely on some way to describe the similarity between detected features. This is what feature descriptors are used for. One very simple feature descriptor is simply taking the pixel intensity values of a patch around the feature. A similarity function could then be to calculate the sum of square differences between the pixels of two patches. The problem with this descriptor is that it is neither scale nor orientation invariant, making it unreliable for tracking between images with significant camera motion. More sophisticated descriptors such as SIFT (scale-invariant feature transform) [18] and SURF (speeded up robust features) [3] are examples of descriptors that deal with these issues. The downside to more detailed descriptors and similarity functions is their high computational requirement.

The simplest way to find matches is by comparing every feature in one image to every feature in another. However, the number of computations for this method is quadratic in the number of features, making such a brute force method slow when the feature extraction yields many candidates. Some approaches avoid this by predicting the location of a feature in the next frame and then only searching for matches in the vicinity of this prediction. This motion prediction can be performed using motion sensors, such as an IMU, or by applying a motion model such as a constant velocity model. Based on the uncertainty in these models, one can calculate the predicted mean and covariance of a feature position and use this to specify the search area when looking for matches.

Direct tracking

Direct feature tracks are found by performing a search for previous features in a new image. This is done by finding the new feature location which minimize the photometric error of the feature between the two images. The photometric error is a pixel-by-pixel comparison between image intensity values. An illustration of this is given in fig. 3, where we can see that the errors have high values where the reference and measured patch are different. By visual analysis, we can see that by moving the measured patch downwards and to the right, we can reduce the photometric error until the two white squares are aligned. For some methods, the reference is an entire image. Other methods choose only a subset of the image pixels, for example, those areas of the image with sufficiently high gradients. The last group, which includes ROVIO, minimizes the error of one or more reference patches. We will assume the last problem in this walk-through of direct feature tracking for simplicity.

A well-known approach to the direct feature tracking problem is the optical flow algorithm proposed by Lucas and Kanade [19]. Following their notation, we define the problem mathematically. Let $F(\boldsymbol{x})$ and $G(\boldsymbol{x})$ define the pixel intensities at position $\boldsymbol{x} = [u, v]$ of the reference and measured patch respectively. We further assume that we move our measured patch by a small perturbation, $\delta\boldsymbol{x} = [\delta u,\, \delta v]$, in order to reduce the photometric error. The square of each pixel intensity value is then a way to measure how well aligned the two patches are.

$$E = \sum_{\boldsymbol{x}} [F(\boldsymbol{x} + \delta\boldsymbol{x}) - G(\boldsymbol{x})]^2 \tag{5}$$

We want to find the perturbation $\delta\boldsymbol{x}$ that minimizes this error, yielding the minimization problem

$$\min_{\delta\boldsymbol{x}} \sum_{\boldsymbol{x}} [F(\boldsymbol{x} + \delta\boldsymbol{x}) - G(\boldsymbol{x})]^2 \tag{6}$$

where

$$F(\boldsymbol{x} + \delta\boldsymbol{x}) \approx F(\boldsymbol{x}) + \delta \frac{\partial F(\boldsymbol{x})}{\boldsymbol{x}} \tag{7}$$

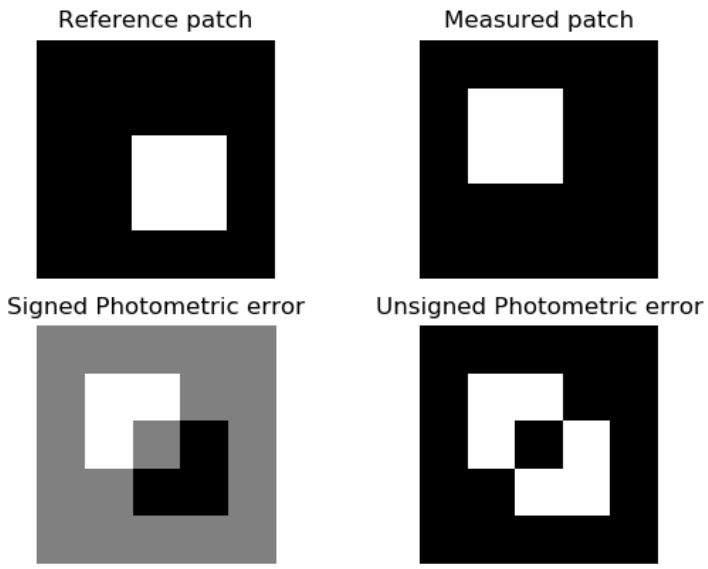


Figure 3: Photometric error example. The photometric errors resemble a pixel-by-pixel intensity difference between the reference and measured patch. In the signed case, gray areas correspond to zero intensity difference while black and white correspond to positive and negative differences. The unsigned photometric error shows the absolute value of differences, so the black areas represent no error, and the white areas represent a big error.

By making a first-order approximation of the form eq. (7), we can solve the minimization problem by differentiating eq. (5) and setting this equal to zero.

$$\begin{aligned} 0 &= \frac{\partial}{\partial(\delta \boldsymbol{x})} E \\ &= \frac{\partial}{\partial(\delta \boldsymbol{x})} \sum_{\boldsymbol{x}} [F(\boldsymbol{x} + \delta \boldsymbol{x}) - G(\boldsymbol{x})]^2 \\ &\approx \sum_{\boldsymbol{x}} 2 \frac{\partial F(\boldsymbol{x})}{\partial \boldsymbol{x}} [F(\boldsymbol{x}) + \delta \boldsymbol{x} \frac{\partial F(\boldsymbol{x})}{\partial(\boldsymbol{x})} - G(\boldsymbol{x})] \end{aligned}$$

Finally, solving for $\delta \boldsymbol{x}$ gives us

$$\delta \boldsymbol{x} = \left[\sum_{\boldsymbol{x}} \left(\frac{\partial F(\boldsymbol{x})}{\partial \boldsymbol{x}} \right)^T [G(\boldsymbol{x}) - F(\boldsymbol{x})] \right] \left[\sum_{\boldsymbol{x}} \left(\frac{\partial F(\boldsymbol{x})}{\partial \boldsymbol{x}} \right)^T \left(\frac{\partial F(\boldsymbol{x})}{\partial \boldsymbol{x}} \right) \right]^{-1} \tag{8}$$

This result shows that by moving the measured patch by the perturbation in eq. (8), the photometric error from eq. (5) will be minimized. That is, assuming the approximation in eq. (7) holds. If this is not the case, the procedure can be performed multiple times until convergence. Many of the proposed feature tracks may be bad due to local minima, changing scenery, and noise. For this reason, it is normal to perform some additional testing to make sure that the track is sufficiently good. This can, for example, be done by placing a threshold on the final photometric error. Other tests used by ROVIO will be presented in section 2.4.

2.1.3 Back-end

Motion estimation

Scaramuzza and Fraundorfer[32] discuss three different kind of motion estimation schemes, all relying on a functioning camera model and feature correspondences. They are:

1. **2D-to-2D:** Motion is estimates based on 2D image correspondences between two camera frames.
2. **3D-to-3D:** Motion is estimated based on correspondences between 3D features given in their respective camera frames.
3. **3D-to-2D:** 3D landmark positions are assumed to be known, and their 2D image correspondences are used to estimate the motion of a camera.

A walkthrough of the 3D-to-2D motion estimation case is given below, as this resembles the back-end of ROVIO. Here, 3D features are kept track of in the filter state, and 2D image observations of these features are used to update the filter.

In 3D-to-2D motion estimation, we assume that we have a prior estimate of the 3D position of a landmark, $\boldsymbol{X}_{k-1}$, and a current 2D image point, $\boldsymbol{p}_k$, associated with that landmark. When estimating the motion of the camera, we want to find the transformation $\boldsymbol{T}_k$ such that the 3D point $\hat{\boldsymbol{X}}_k = \boldsymbol{T}_k \boldsymbol{X}_{k-1}$ projected onto the camera frame, $\hat{\boldsymbol{p}}_k$, is as close as possible to $\boldsymbol{p}_k$. This difference is often referred to as the reprojection error, and the camera motion is found by solving the following minimization problem.

$$\underset{\boldsymbol{T}_k}{\operatorname{argmin}} ||\boldsymbol{p}_k - \hat{\boldsymbol{p}}_k||^2 = \underset{\boldsymbol{T}_k}{\operatorname{argmin}} ||\boldsymbol{p}_k - \boldsymbol{\pi}(\boldsymbol{T}_k \boldsymbol{X}_{k-1})||^2 \tag{9}$$

One needs several correspondences to avoid ambiguous solutions to the problem. In fact, [7] shows that at least three 3D-to-2D correspondences are necessary to calculate the camera pose. An extension of the previous minimization problem can be extended to include n correspondences.

$$\underset{\boldsymbol{T}_k}{\operatorname{argmin}} \sum_{i=0}^{i=n} ||\boldsymbol{p}_k^i - \hat{\boldsymbol{p}}_k^i||^2 = \underset{\boldsymbol{T}_k}{\operatorname{argmin}} \sum_{i=0}^{i=n} ||\boldsymbol{p}_k^i - \boldsymbol{\pi}(\boldsymbol{T}_k \boldsymbol{X}_{k-1}^i)||^2 \tag{10}$$

This problem can be solved with a nonlinear optimization scheme like Levenberg-Marquardt or Gauss-Newton. The outcome of the optimization may depend on the initialization of the parameters in $\boldsymbol{T}_k$. If the frame rate is sufficiently high, the initial transformation $\boldsymbol{T}_k$ can be set to identity. If not, there are solutions to yielding reasonable prior estimates using algorithms such as DLT[13]. The problem of finding reasonable initial estimates for the transformation is one of the reasons why fusion with inertial measurements in VIO can produce a better output. The initial value of $\boldsymbol{T}_k$ can be directly calculated from the inertial measurements between each camera frame.

Filtering vs. smoothing

The procedure just described estimates motion on a frame-to-frame basis, discarding measurements further back in time than the current image. This procedure is often referred to as filtering and can minimize the overhead that comes from keeping track of old and possibly irrelevant frames. This also reduces computational costs and runtime, which is crucial in many applications, especially in real-time operations.

The alternative to filtering is called smoothing and relies on feature tracks from longer series of images. By jointly optimizing over several camera poses at once, it is possible to reduce the overall

error of the trajectory estimation and reduce the amount of drift in the system. It is possible only to perform smoothing on a fixed set of previous frames to keep computational costs feasible in real-time applications. This procedure is called fixed-lag smoothing. Another procedure to reduce the computational costs of smoothing is by introducing keyframes. Instead of smoothing over all the most recent frames, one can choose only to keep track of images that are sufficiently different from each other.

2.2 Visual Inertial Odometry

Scaramuzza and Zhang describes cameras and IMUs as complementary sensor types in their article on visual-inertial odometry of aerial robots[33]. While cameras can extract much information about the scene, the IMU is independent of its surroundings. Camera measurements also contain a lot more information than inertial measurements. A typical gray-scale image is represented by a byte-sized value for each pixel, but IMU measurements only need a couple of bytes for each degree of freedom. The large amount of data in an image makes the processing of measurements much slower than for the IMU. In addition, measurement acquisition is much slower for the camera because of the required exposure time to receive enough light in the photo-sensor. These two properties make it possible to operate the IMU at a much higher rate than the camera. Thus, by including an IMU in the estimation, the filter can extract information about the system in the time between each camera measurement. This information can be used to predict the camera motion, giving a good initial estimate to the motion estimation discussed earlier in section 2.1.3

Camera measurements can be unreliable in many scenarios. When retrieving an image, the camera exposes its image sensor to light from the scene for a period of time. The length of this period, the exposure time, will often depend on properties of the scene, such as lighting conditions, and can have a significant impact on the final image. If the exposure time is too short, the image sensor will receive a small amount of light, and the image becomes very dark, or under-exposed. Likewise, if the exposure time is too long, the image sensor will receive too much light, and the image becomes very bright, or over-exposed. Some scenes can have a high dynamic range (HDR), meaning that they both have very bright and very dark areas. This can cause a problem because capturing a valuable image of the entire scene will be challenging. If the exposure is set to capture a very light image area, it will cause under-exposure in other areas and vice versa. Another issue related to the exposure time is the presence of motion blur. The image scene will change during the exposure during fast motions and cause a blurring effect. Even if there are no exposure issues and sharp images are obtained, there is no guarantee that the image is valuable in a VO context. For example, Low-texture areas with no unique features would make it very difficult to perform tracking.

All the cases above make VO unreliable because feature tracking becomes difficult. In a VIO framework, the same issues would not be as critical because the inertial measurements can be integrated to estimate the motion of a system in the absence of visual information. However, pure integration of inertial measurements is unreliable over a longer period. The presence of biases and noise would lead to a quickly drifting estimate upon integration, making them unfit for motion estimation by itself. In combination with the camera's ability to capture scene details and perform localization based on surrounding features, this drift can be corrected.

An issue related to monocular VO, visual odometry with just one camera, is the ambiguity of scale. Different strategies have been tried to solve this scale ambiguity problem by introducing a scaled constraint to the estimation. For instance, [5] proposes to use the assumption of a planar ground and a fixed distance from the ground to the camera. They argue that this is usually valid for mobile robots and vehicles moving indoors or on roads. These kinds of assumptions are not necessary for VIO systems. IMU measurements include metric measures in acceleration and angular rate, rendering scale observable in the overall state estimation.

Another advantage of using inertial measurements is to gain observability of pitch and roll. Martinelli shows that when gravity is present, these states will be observable modes in the dynamics

[22]. In the case of no movement, the accelerometer will only measure the gravitational pull on the sensor. This gravity vector can be decomposed onto the three accelerometer axis, making it possible to calculate its pitch and roll. However, it will not yield any information about the rotation around the gravity axis, i.e., yaw/heading angle.

2.3 Stereo triangulation and disparity

One of the biggest advantages of stereo systems is that they make it possible to initialize the depth of features from a single image pair. While a monocular camera can only induce the direction of a feature, a perfect stereo system with infinite resolution can, in theory, pinpoint the exact point of a feature. This is done by triangulating the feature position, using knowledge of the camera models and their spatial configurations. Figure 4 illustrates the concept in a simplified 2-dimensional case. The cameras L and R both see a feature P in their image.

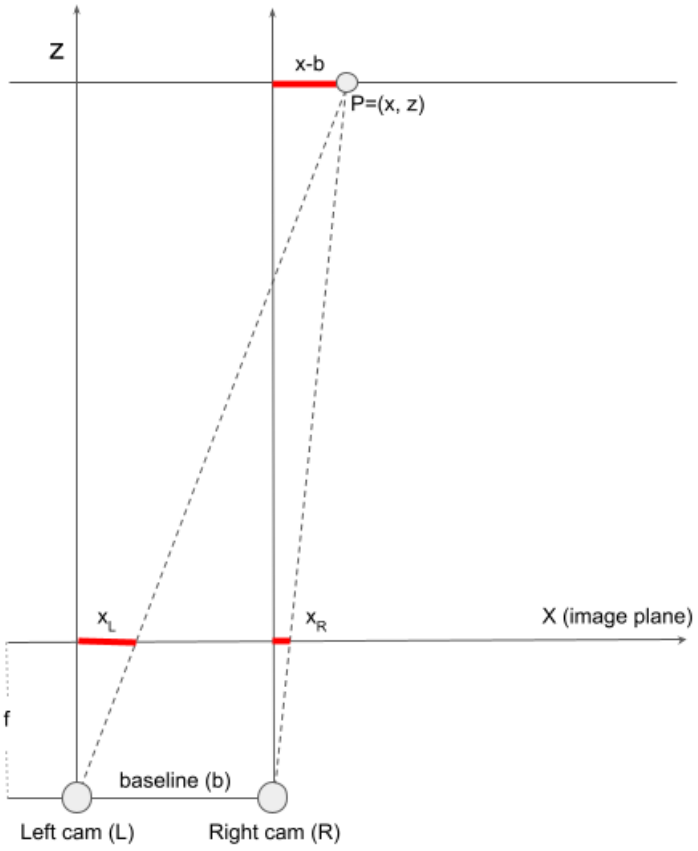


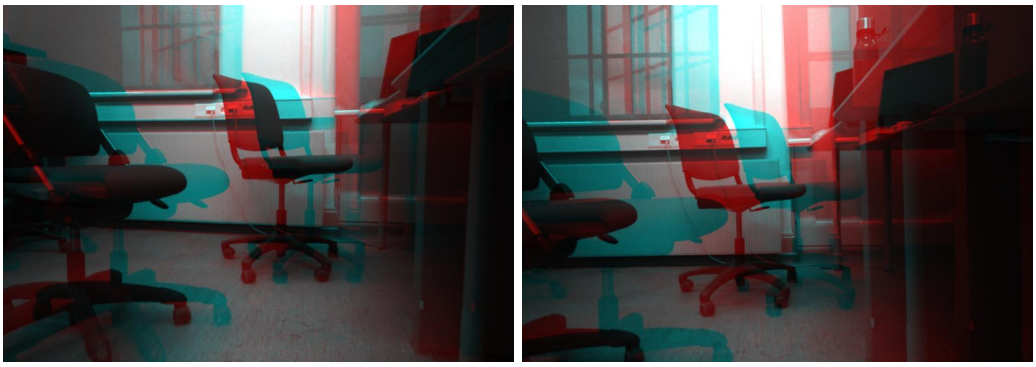
Figure 4: Illustration of a triangulation scheme for a stereo camera system.

The feature, P, is projected to image coordinates x_L and x_R respectively. From the figure, we can identify two sets of similar triangles, one for each camera projection. Since the side ratios between two similar triangles must be constant, we can set up one equality for each camera:

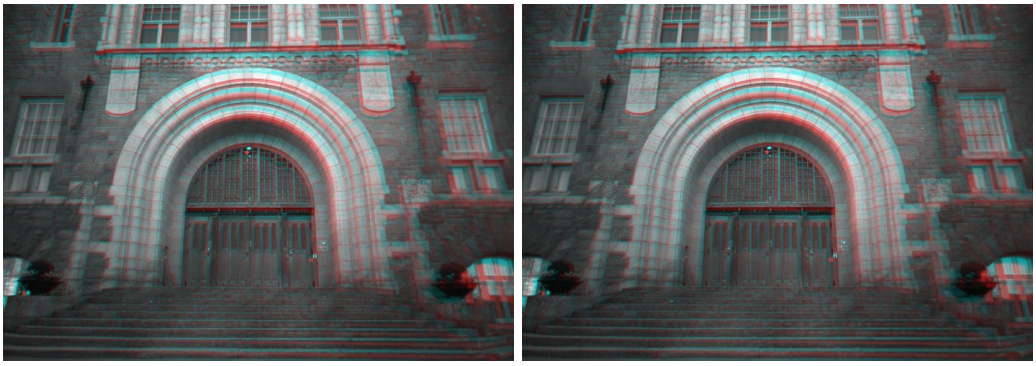
$$\frac{f}{z} = \frac{x_L}{x}, \quad \frac{f}{z} = \frac{x_R}{x-b} \tag{11}$$

With the known quantities f (focal length), x_L, and x_R (image plane coordinates), we have two equations with two unknown variables corresponding to the unknown coordinates of the feature. Simple algebra yields the following closed-loop expression for x and z

$$z = f\frac{b}{x_L - x_R}, \quad x = x_L\frac{b}{x_L - x_R} \tag{12}$$



(a) Baseline=12.5cm (b) Baseline=25cm



(c) Baseline=12.5cm (d) Baseline=25cm

Figure 5: Example of how disparity grows with baseline and decrease with distance. The left and right images are captured with baselines of 12.5cm and 25cm respectively. It is clear from (a) and (b) that the disparity is larger for the images captured with a baseline of 25cm. The effect of scene distance can be seen as the disparity of both (c) and (d) is greatly reduced from (a) and (b).

We can see that the difference between camera pixel coordinates $x_L - x_R$ is inversely proportional to both parameters. This value is often referred to as the stereo disparity, d. From eq. (12) we can see that the disparity is significantly decreased with distance to an object, but also with the baseline of the camera system. Figure 5 illustrates this effect with one set of images from $\sim$ 2m and one set from $\sim$ 10m.

Since the image sensors are discretely divided, the detected disparity accuracy will decrease with lower image resolutions. Other noise can also be introduced by inaccurate camera modeling and calibration. Since the noise will have a relatively bigger impact on small disparities, the triangulation should, in theory, suffer from smaller baselines on large distances where the disparity becomes close to zero. In such cases, increased resolution and baseline should improve the accuracy. Since higher resolution demands higher computational effort, this is not necessarily plausible. High baselines can also become an issue, both regarding the use of space and feature matching quality. When the baseline increase, the change in the viewpoint of a feature can yield a less accurate cross-camera match. This is especially relevant for close-range matching. For these reasons, the best camera resolution and baseline choice will vary with the resources available and feature distances.

2.4 ROVIO

ROVIO [4] is a VIO framework created by the Autonomous Systems Lab at ETH, Zurich. It can be categorized as a filter-based VIO using direct feature tracking. The framework is relatively lightweight, which makes it suitable for aerial robots. Due to its direct front-end with multi-level patches, the method is robust to low texture scenes, an important factor for VTOL applications in changing weather conditions. It is also possible to incorporate multiple cameras, making it possible to utilize disparity to initialize feature depths.

Front-end

Multilevel patches

Every feature is described by a multi-level patch, $P = \{P_0, P_1, ..., P_L\}$ where L is the number of levels in the patch. Every patch level P_l is a $n \times n$ image patch taken at the l'th resolution level. For each level, the resolution image resolution is down-sampled by a factor of 2. This means that P_0 will be a full resolution patch while patch P_n will have a reduced resolution of a factor $\frac{1}{2^n}$. Since the patch size for each level is constant, patches of higher levels will cover a larger area of the scene. Using these patches as feature descriptors has the advantage of taking a large set of pixels into account in the filtering process. In contrast to indirect methods, this approach inherently considers the scene's texture, making it possible to track non-corner-shaped features such as lines. It also makes the tracking more robust to lighting conditions, image blur, and low-texture scenes. During VTOL in an outdoor environment, one can not guarantee anything about scene conditions. Foggy weather could lead to low-texture image frames, and lightning conditions will not be predictable.

Photometric error and jacobian

For an image patch $P = \{P_0, ..., P_L\}$ with coordinate $\boldsymbol{p}$, each patch pixel $\boldsymbol{p}_j$ at image level l has the photometric error given by eq. (13). I_l represents the intensity values of an image at level l. Illumination differences between images are taken care of by the a and b values, and s_l is a scaling factor used to correct for resolution differences in the higher patch levels. The matrix $\boldsymbol{D}$ is used to account for warping effects between images and will be explained later.

$$e_{l,j}(\boldsymbol{p}, P, I, \boldsymbol{D}) = P_l(\boldsymbol{p}_j) - aI_l(\boldsymbol{p}s_l + \boldsymbol{D}\boldsymbol{p}_j) - b \tag{13}$$

By stacking all error terms in a vector $\boldsymbol{b}(\hat{\boldsymbol{p}}, P, I, \boldsymbol{D})$, where $\hat{\boldsymbol{p}}$ is the current estimate of the patch position, we can formulate the minimization of intensity errors as a Gauss-Newton optimization problem.

$$\boldsymbol{b}(\hat{\boldsymbol{p}} + \delta\boldsymbol{p}, P, I, \boldsymbol{D}) = \boldsymbol{A}(\hat{\boldsymbol{p}}, I, \boldsymbol{D})\delta\boldsymbol{p} + \boldsymbol{b}(\hat{\boldsymbol{p}}, P, I, \boldsymbol{D}) \tag{14}$$

$\boldsymbol{A}(\hat{\boldsymbol{p}}, I, \boldsymbol{D})$ denotes the jacobian of the error terms and $\delta\boldsymbol{p}$ a perturbation in the patch position. Although this can be solved numerically by a nonlinear optimizer, the dimensionality of $\boldsymbol{A}$ is way too big for this to be efficient. We shall see later how this problem is solved in the update step of the filter.

Patch warping

When a camera is moved around, objects in the scene will appear warped in different ways depending on the camera's perspective. This warping effect should be accounted for when comparing feature patches between images to achieve optimal results. Figure 6 shows an image of a chessboard that has been warped from the first image to the second. The same feature in the two images should cover the same areas to get the most information from the photometric error. We can see that simply reusing the patch shape from the first image does not work very well in the second. On the other hand, the warped green patch captures the exact area of the original patch from the first image. This warping can be described mathematically by

$$\boldsymbol{D} = \frac{\partial \boldsymbol{p_2}}{\partial \boldsymbol{p_1}} = \frac{\partial \boldsymbol{\pi}(\boldsymbol{\mu_2})}{\partial \boldsymbol{p_1}} = \frac{\partial \boldsymbol{\pi}(\boldsymbol{\mu_2})}{\partial \boldsymbol{\mu_2}} \frac{\partial \boldsymbol{f}(\boldsymbol{\mu_1})}{\partial \boldsymbol{\mu_1}} \frac{\partial \boldsymbol{\pi^{-1}}(\boldsymbol{p_1})}{\partial \boldsymbol{p_1}} \tag{15}$$

where $\boldsymbol{\pi}()$ is the camera projection model, $\boldsymbol{f}()$ is a process model transforming the bearing vector $\boldsymbol{\mu}_i$ of a feature from frame i to $i+1$ and $\boldsymbol{p}_i$ is the pixel coordinate of a feature in frame i. The

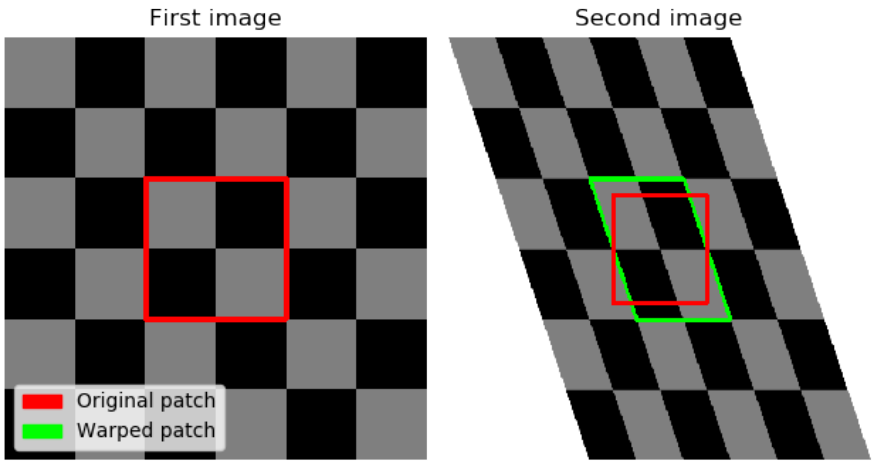


Figure 6: Patch warping example. Note that the two images are scaled differently, such that the original patch (red) is really the same size.

formula can be interpreted as a jacobian matrix describing how a perturbation around a feature position will affect the projection in the next frame.

Feature Management

Detection of new features in ROVIO is based on the FAST corner detector [30] and provides a large number of candidates. After candidates close to current features are removed, the rest is sorted based on a gradient-based score. This score is based on calculating the eigenvalues of the matrix $\boldsymbol{H}$, where g_x and g_y are the image gradients in the x and y directions of the feature. It is similar to the one proposed by [35] but extended to work for multi-level patches. In order to obtain a good distribution of features over the image, a bucketing technique is used to ensure that there are not initialized many features in the same area.

$$\boldsymbol{H} = \begin{bmatrix} g_x^2 & g_x g_y \\ g_x g_y & g_y^2 \end{bmatrix} \tag{16}$$

The original Shi-Tomasi score ranks a feature based on the minimal eigenvalue of $\boldsymbol{H}$. This is motivated by interpreting the eigenvalues as the gradient size in the direction of their corresponding eigenvectors. A good corner feature should therefore have two high eigenvalues. In ROVIO, the sum of both eigenvalues is used to include edges in addition to corners. Even though edges might have a minimal eigenvalue in the direction of the edge, they can introduce much information in the opposite direction. Including such features can increase the system's robustness in low-texture scenarios with few corner points.

To avoid a continuously growing state space with old and useless features, evaluations of the features are performed regularly. The IEKF is not very scalable, so a large state space should be avoided to keep the runtime feasible. After every update step in the filter, all features are subjected to several tests to check their validity. The obtained innovation residuals are compared to the predicted innovation covariance for every feature. If the covariance weighted norm of residuals precedes a certain threshold, the features are rejected. The features are also rejected if the total intensity error is above a certain threshold, regardless of its predicted covariance. Further, a local texture

test is performed by comparing the residuals of four points around each feature with the residual of the actual feature. If the residuals surrounding the feature are not sufficiently larger than the feature residual, the area is considered too uniform, and the feature is rejected.

In addition to these individual feature tests, three additional quality parameters are used to rank features against one another.

1. **Global quality:** How often has the feature track been successful since initialization?
2. **Local quality:** How often has the feature track been successful when predicted to be in the field of view?
3. **Local visibility:** How often has the feature been in the field of view?

These quality measures ensure that an upper bound of features is maintained in the filter state. An adaptive threshold ensures that more old and poor features are filtered out to make room for new ones.

Back-end

The back-end is based on an iterated extended Kalman filter. Starting from the prior state estimate $\boldsymbol{x}_k^-$, the dynamics are linearized about iteratively refined linearization points $\boldsymbol{x}_{k,j}^+$, where k denotes the time and j the iteration number in the current update step. This makes the filter much more robust to poor initial estimates, which becomes essential when estimating newly initialized and uncertain landmarks. Once the filter has converged, the number of iterations will be reduced significantly, making it much more effective.

Formalizing the filter

The propagation step takes the a posteriori state estimate at a previous time step, $\boldsymbol{x}_{k-1}^+$, and estimates the a priori estimate in the current time step, $\boldsymbol{x}_k^-$, based on the model dynamics $\boldsymbol{f}$. The corresponding covariance is also calculated by eq. (18).

$$\boldsymbol{x}_k^- = \boldsymbol{f}(\boldsymbol{x}_{k-1}^+, \boldsymbol{0}) \tag{17}$$

$$\boldsymbol{P}_k^- = \boldsymbol{F}_{k-1}\boldsymbol{P}_{k-1}^+\boldsymbol{F}_{k-1}^T + \boldsymbol{G}_{k-1}\boldsymbol{W}_{k-1}^+\boldsymbol{G}_{k-1}^T \tag{18}$$

Here, $\boldsymbol{F}_{k-1}$ and $\boldsymbol{G}_{k-1}$ are the jacobians of the dynamics with respect to state $\boldsymbol{x}_{k-1}$ and the process noise $\boldsymbol{w} \sim \mathcal{N}(0, \boldsymbol{W})$ respectively.

$$\boldsymbol{F}_{k-1} = \frac{\partial}{\partial \boldsymbol{x}_{k-1}} \boldsymbol{f}(\boldsymbol{x}_{k-1}^+) \tag{19}$$

$$\boldsymbol{G}_{k-1} = \frac{\partial}{\partial \boldsymbol{w}_{k-1}} \boldsymbol{f}(\boldsymbol{x}_{k-1}^+) \tag{20}$$

The update step can be considered iterative in two ways. First of all, it performs update steps for each feature individually. Each feature i neglects the other features in the state and performs the update step considering only the current feature. Secondly, it performs the update step several times for each feature, iteratively refining the linearization points for each step, starting from the first predicted estimate at time k, $\boldsymbol{x}_{k,0}^+ = \boldsymbol{x}_k^-$. The j'th update step will then yield the following jacobians:

$$\boldsymbol{H}_{k,j} = \frac{\partial h}{\partial \boldsymbol{x}_k}(\boldsymbol{x}^+_{k,j}, \boldsymbol{0}) \tag{21}$$

$$\boldsymbol{J}_{k,j} = \frac{\partial h}{\partial \boldsymbol{n}_k}(\boldsymbol{x}^+_{k,j}, \boldsymbol{0}) \tag{22}$$

$$\boldsymbol{L}_{k,j} = \frac{\partial (\boldsymbol{x}^-_k \boxplus \Delta\boldsymbol{x})}{\partial \Delta\boldsymbol{x}}(\boldsymbol{x}^+_{k,j} \boxminus \boldsymbol{x}^-_k) \tag{23}$$

where $\boldsymbol{h}(\boldsymbol{x}^+_{k,j}, \boldsymbol{n}_k)$ defines the innovation term $\boldsymbol{y}_{k,j}$ and $\boldsymbol{L}_{k,j}$ is necessary to account for special linearizations of the rotational state components, and $\boxplus$, $\boxminus$ are addition and subtraction operators adapted to work for rotations. Because of constraints enforced on 3D rotational matrices, they require a special kind of mathematical analysis based on Lie theory. Solà et al. describes this phenomena and it's importance in robotics applications such as state estimation and control[36]. Using eqs. (21) to (23) we can calculate a Kalman gain $\boldsymbol{K}_{k,j}$ and state update terms $\Delta\boldsymbol{x}_{k,j}$ for each iteration j as

$$\boldsymbol{S}_{k,j} = \boldsymbol{H}_{k,j}\boldsymbol{L}^T_{k,j}\boldsymbol{P}^-_{k,j}\boldsymbol{L}_{k,j}\boldsymbol{H}^T_{k,j} + \boldsymbol{J}_{k,j}\boldsymbol{R}_k\boldsymbol{J}^T_{k,j} \tag{24}$$

$$\boldsymbol{K}_{k,j} = \boldsymbol{L}^T_{k,j}\boldsymbol{P}^-_k\boldsymbol{L}_{k,j}\boldsymbol{H}^T_{k,j}\boldsymbol{S}^{-1}_{k,j} \tag{25}$$

$$\Delta\boldsymbol{x}_{k,j} = \boldsymbol{K}_{k,j}\Big(\boldsymbol{H}_{k,j}\boldsymbol{L}_{k,j}(\boldsymbol{x}^+_{k,j} \boxminus \boldsymbol{x}^-_k) - \boldsymbol{h}(\boldsymbol{x}^+_{k,j}, \boldsymbol{0})\Big) - \boldsymbol{L}_{k,j}(\boldsymbol{x}^+_{k,j} \boxminus \boldsymbol{x}^-_k) \tag{26}$$

$$\boldsymbol{x}^+_{k,j+1} = \boldsymbol{x}^+_{k,j} \boxplus \Delta\boldsymbol{x}_{k,j} \tag{27}$$

This update step is repeated until the state estimate converges, i.e., $\Delta\boldsymbol{x}_{k,j}$ is below a specified threshold. The a posteriori covariance is only updated after convergence at the n'th iteration with the formula

$$\boldsymbol{P}^+_k = (\boldsymbol{I} - \boldsymbol{K}_{k,n}\boldsymbol{H}_{k,n})\boldsymbol{L}^T_{k,n}\boldsymbol{P}^-_k\boldsymbol{L}_{k,n} \tag{28}$$

State definition

The filter state $\boldsymbol{x}$ is defined by the following variables.

- $\boldsymbol{r}$:= position of IMU
- $\boldsymbol{v}$:= velocity of IMU
- $\boldsymbol{q}$:= attitude of IMU
- $\boldsymbol{b}_f$:= bias of accelerometer
- $\boldsymbol{b}_\omega$:= bias of gyroscope
- $\boldsymbol{c}$:= translation from IMU to camera
- $\boldsymbol{z}$:= rotational from IMU to camera
- $\boldsymbol{\mu}_i$:= bearing vector to landmark i
- $\boldsymbol{p_i}$:= distance parameter of landmark i

Propagation step

The propagation step is driven by the IMU measurements denoted as $\tilde{\boldsymbol{f}}$ and $\tilde{\boldsymbol{\omega}}$ for accelerometer and gyroscope measurements respectively. After correcting for bias, $\boldsymbol{b}_f, \boldsymbol{b}_\omega$ and noise $\boldsymbol{w}_f, \boldsymbol{w}_\omega$, the estimated accelerations and angular rates, $\hat{\boldsymbol{f}}$ and $\hat{\boldsymbol{\omega}}$, is obtained.

$$\hat{\boldsymbol{f}} = \tilde{\boldsymbol{f}} - \boldsymbol{b}_f - \boldsymbol{w}_f \tag{29}$$

$$\hat{\boldsymbol{\omega}} = \tilde{\boldsymbol{\omega}} - \boldsymbol{b}_\omega - \boldsymbol{w}_\omega \tag{30}$$

We can further calculate the estimated camera velocity and angular rate, where $\boldsymbol{z}()$ denotes a rotation from IMU to the camera frame.

$$\hat{\boldsymbol{v}}_{\mathcal{C}} = \boldsymbol{z}(\boldsymbol{v} + \hat{\boldsymbol{\omega}}^{\times}\boldsymbol{c}) \tag{31}$$
$$\hat{\boldsymbol{\omega}}_{\mathcal{C}} = \boldsymbol{z}(\hat{\boldsymbol{\omega}}) \tag{32}$$

Using these definitions, we can write the dynamics of the system $\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x})$ as follows.

$$\dot{\boldsymbol{r}} = -\hat{\boldsymbol{\omega}}^{\times}\boldsymbol{r} + \boldsymbol{v} + \boldsymbol{w}_r \tag{33}$$
$$\dot{\boldsymbol{v}} = -\hat{\boldsymbol{\omega}}^{\times}\boldsymbol{v} + \hat{\boldsymbol{f}} + \boldsymbol{q}^{-1}(\boldsymbol{g}) \tag{34}$$
$$\dot{\boldsymbol{q}} = -\boldsymbol{q}(\hat{\boldsymbol{\omega}}) \tag{35}$$
$$\dot{\boldsymbol{b}}_f = \boldsymbol{b}_{bf} \tag{36}$$
$$\dot{\boldsymbol{b}}_{\omega} = \boldsymbol{w}_{b\omega} \tag{37}$$
$$\dot{\boldsymbol{c}} = \boldsymbol{w}_c \tag{38}$$
$$\dot{\boldsymbol{z}} = \boldsymbol{w}_z \tag{39}$$
$$\dot{\boldsymbol{\mu}}_i = \boldsymbol{N}(\boldsymbol{\mu}_i)^T \left(\hat{\boldsymbol{\omega}}_{\mathcal{C}} + \boldsymbol{n}(\boldsymbol{\mu}_i)^{\times}\frac{\hat{\boldsymbol{v}}_{\mathcal{C}}}{d(\rho_i)}\right) + \boldsymbol{w}_{\mu,i} \tag{40}$$
$$\dot{\rho}_i = -\boldsymbol{n}(\boldsymbol{\mu}_i)^T\hat{\boldsymbol{v}}_{\mathcal{C}}/d'(\rho_i) + w_{p,i} \tag{41}$$

$\boldsymbol{N}(\boldsymbol{\mu}_i)$ is the projection of a 3D vector onto the 2D tangent space of the bearing vector $\boldsymbol{\mu}_i$, $\boldsymbol{n}(\boldsymbol{\mu}_i)$ is the 3D unit vector corresponding to $\boldsymbol{\mu}_i$ and $\boldsymbol{g}$ is the gravity vector expressed in the world frame. $d(\rho)$ is the distance function and $d'(\rho)$ its derivative. This function takes a distance parameter and produces the actual distance to a feature. By default, ROVIO uses the inverse depth parameterization

$$d(p) = \frac{1}{p} \tag{42}$$

To match the formulation of eq. (17), these equations should be integrated over the specified time-step Δt, which in the case of ROVIO is done using a forward-euler scheme.

$$\boldsymbol{x}_{k+1}^{-} = \boldsymbol{f}(\boldsymbol{x}_k^{+}, \boldsymbol{0}) = \boldsymbol{x}_k^{+} \boxplus \Delta t\dot{\boldsymbol{x}}_k \tag{43}$$

While trivial for most of the states, this becomes a more complicated calculation for the rotational states, $\boldsymbol{q}, \boldsymbol{z}$ and $\boldsymbol{\mu}$ because of the Lie theory related issues mentioned earlier. The equations are given underneath, where the $\otimes$ operator is defined as the quaternion product and $exp()$ is defined as the exponential map from angle to quaternion following the Hamilton convention in eq. (47).

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k \otimes exp\left(\Delta t(\boldsymbol{b}_{\omega,k} + \boldsymbol{w}_{\omega,k} - \tilde{\boldsymbol{\omega}}_k)\right) \tag{44}$$
$$\boldsymbol{z}_{k+1} = exp(\Delta t\boldsymbol{w}_{z,k}) \otimes \boldsymbol{z}_k \tag{45}$$
$$\boldsymbol{\mu}_{i,k+1} = exp\Bigg(\Delta t\Big(\big(\boldsymbol{I} - \boldsymbol{n}(\boldsymbol{\mu}_{i,k})\boldsymbol{n}(\boldsymbol{\mu}_{i,k})^T\big)\hat{\boldsymbol{\omega}}_{\mathcal{C}} + \boldsymbol{n}(\boldsymbol{\mu}_{i,k})^{\times}\frac{\hat{\boldsymbol{v}}_{\mathcal{C}}}{d(\rho_{i,k})} + \boldsymbol{N}(\boldsymbol{\mu}_{i,k})\boldsymbol{w}_{\mu,i}\Big)\Bigg) \otimes \boldsymbol{\mu}_{i,k} \tag{46}$$

$$exp(\boldsymbol{\theta}) = \boldsymbol{I} - \frac{(1 - cos(||\boldsymbol{\theta}||))\boldsymbol{\theta}^{\times}}{||\boldsymbol{\theta}||^2} + \frac{(||\boldsymbol{\theta}|| - sin(||\boldsymbol{\theta}||))\boldsymbol{\theta}^{\times^2}}{||\boldsymbol{\theta}||^3} \tag{47}$$

Update step

ROVIO applies the photometric error as a direct innovation term in their Kalman filter update. The error term is very high-dimensional, which would lead to a high computational cost. However, by observing that eq. (14) is only dependent on the pixel coordinate $\boldsymbol{p}_i$ for patch P_i, it should be possible to reduce it to a two-dimensional error term. This can be done by employing a QR-decomposition of the jacobian matrix.

$$\begin{aligned}\boldsymbol{A}(\boldsymbol{p}_i, \boldsymbol{I}, \boldsymbol{D}_i) &= \boldsymbol{Q}(\boldsymbol{p}_i, \boldsymbol{I}, \boldsymbol{D}_i)\boldsymbol{R}(\boldsymbol{p}_i, \boldsymbol{I}, \boldsymbol{D}_i) \\ &= \begin{bmatrix}\boldsymbol{Q}_1(\boldsymbol{p}_i, \boldsymbol{I}, \boldsymbol{D}_i) & \boldsymbol{Q}_2(\boldsymbol{p}_i, \boldsymbol{I}, \boldsymbol{D}_i)\end{bmatrix} \begin{bmatrix}\boldsymbol{R}_1(\boldsymbol{p}_i, \boldsymbol{I}, \boldsymbol{D}_i) \\ \boldsymbol{0}\end{bmatrix}\end{aligned} \tag{48}$$

Where the innovation term $\boldsymbol{y}_i$ and its jacobian $\boldsymbol{H}_i$ for feature i becomes

$$\boldsymbol{y}_i = \boldsymbol{Q}_1(\boldsymbol{p}_i, \boldsymbol{I}, \boldsymbol{D}_i)^T \boldsymbol{b}(\boldsymbol{p}_i, P_i, \boldsymbol{I}, \boldsymbol{D}_i) \tag{49}$$

$$\boldsymbol{H}_i = \boldsymbol{R}_1(\boldsymbol{p}_i, \boldsymbol{I}, \boldsymbol{D}_i) \frac{d\boldsymbol{\pi}}{d\boldsymbol{\mu}}(\boldsymbol{\mu}_i^+) \tag{50}$$

These results can be used directly in the update step equations defined from eq. (24).

Multiple camera extension

The filter description above is based on a monocular VIO setup, although ROVIO does have the ability to integrate multiple cameras in the filter.

In the case of multiple cameras, the filter state becomes more or less the same. Extrinsic parameters for the extra cameras must be added, and the features must be parameterized for a specific camera. If features are used in the filter update of a different camera than the one used for parameterization, a transformation of the bearing vector to the correct frame has to be performed. Since this step is highly reliant on the transformation between cameras, the performance of stereo systems will depend heavily on accurate calibration.

3 Related Work

In a paper comparing several different state-of-the-art VIO systems, Delmerico and Scaramuzza score the systems on their per-frame processing time, CPU and memory usage[6]. Their results show a clear positive correlation between these properties and the accuracy of a system. Across several experiments, they found that VINS-mono[27] was the most consistently accurate system. This method applies fixed-lag smoothing, which improves consistency over the previous frames. In addition to this, the algorithm includes temporal calibration of the sensors online, making the method less affected by synchronization issues[28]. Another promising system that could show high accuracy despite the long per-frame processing time was OKVIS[17]. Delmerico and Scaramuzza argues that this shows that the algorithm is robust but points out that the data collected did not include high-velocity motions, which can be challenging to handle with low frame rates. This method, like VINS-mono, applies fixed-lag smoothing to improve consistency across camera frames. Although this can increase the accuracy of the systems, it does not come without its price. The smoothing significantly increases computational costs and memory usage. In contrast to this, ROVIO[4] applies a much cheaper pure filter method for the image updates. Although it is evaluated to have lower accuracy than VINS-mono and OKVIS, its lightweight nature can make it a better choice as resources can be freed up for other tasks such as motion planning and control.

In VIO, the quality of the input data is essential for the system's performance. The actual quality of measurements does not only rely on the quality of the sensor. Another critical factor is how we associate the measurement to a point in time. If measurements are given wrong timestamps, it can corrupt the estimation process and lead to significant inaccuracies. In a multi-sensor system such as VIO, relating all sensors to a common reference clock can significantly impact the system's quality. An approach used by Nikolic et al. is to have an onboard MCU to oversee the timestamps. This keeps track of the reference time and periodically triggers the IMU and cameras to start data acquisition[24]. They also point out a critical difference between the IMU and camera acquisition. Because the camera needs some exposure time to retrieve data, the measurement will suffer from an exposure-dependent offset, assuming that the ideal measurement timestamp is in the middle of the exposure. They propose a strategy where the camera trigger signal is sent slightly earlier to account for the exposure offset.

Although the method mentioned above can reduce significant errors related to time offsets between different sensors, there still may be constant offsets related to hardware delays. The signal used to trigger a camera can have significant delays during transmission, e.g., due to internal signal filtering in the camera. To account for this, Furgale et al. suggests a software solution to estimating the temporal offset between sensors in continuous time[8]. The method proposed jointly estimates both temporal and spatial differences between the IMU and camera sensors by including the temporal offset as a variable in the optimization process. The procedure is available in the open-source calibration toolbox, Kalibr[9].

Varying lighting conditions can be challenging in many VIO applications, including outdoor VTOL. With bad lighting, the cameras will struggle to extract and track features. They will be rendered useless, and we are left with a dead reckoning IMU system. Mascarich et al. deals with this challenge in an underground environment by attaching LED lights to their drone[23]. The LEDs are only triggered to flash when the camera shutter is open to save energy. Whether this is a possible solution in VTOL is not necessarily given. The experiments were conducted in a much more confined space, making it easier to illuminate with a restricted light source. If more powerful equipment is needed, it could become too heavy, big, or power demanding to be applicable. Another solution to the light problem is to move away from the visual light specter, for example, applying thermal imaging instead. Khattak et al. concluded nighttime experiments at an height of 11 meters above a parking lot[14]. Their key-frame-based thermal VIO system showed good accuracy, indicating that a thermal solution to VIO might be a good option for VTOL at night. In another paper, Khattak et al. presents a thermal inertial odometry system that is resilient not only to dark environments but visually degraded scenes in the presence of smoke and dust[15]. This robustness to weather conditions can be of high value to a company such as Aviant, which

may need to transport medical goods despite bad conditions.

When it comes to solving the problem of VTOL, much research suggests the use of a target pattern on the landing site. Both Araar et al. and Lange et al. use patterns designed such that they are detectable at different distances[2][16]. This is done by utilizing sub-patterns in different scales, such that different sub-patterns can be recognized at different distances. They both show that the problem of landing a multi-rotor UAV with visual detection of a landing platform is plausible, even with a moving target, as in the case of Araar et al. A shortcoming of this type of solution is the need for a specific platform pattern at the place of landing, making the system less versatile. Also, occluded scenes may make the pattern invisible, ruining the chance of detecting the target. They also perform their experiments indoors, with good lighting conditions and low distances. This is not comparable with an outdoor scenario with unstable weather conditions and much larger ranges.

The large feature ranges are perhaps on of the main challenges related to VTOL. As discussed by Warren et al., the stereo disparity is reduced with distance to the scene. For higher altitudes, a stereo camera system is thus reduced to a monocular system which makes feature depth estimation more difficult [38]. In their research on fixed-wing aircraft, Warren et al. perform visual odometry up to heights of 120m. Due to the wrong initial depth estimates from ordinary stereo triangulation, they suggest a solution where scale is optimized over a bundle of stereo images instead. This yields a bigger disparity due to the movement of the aircraft between images. Although this effect might not be as significant in a VTOL case as opposed to a fixed-wing aircraft with much higher horizontal speeds, it indicates that horizontal excitation of the cameras will improve the observability of a UAV's height.

An analysis from Alvertos [1] shows that increasing either the baseline or resolution of the cameras will improve the triangulation of features at large distances. However, the overlapping field of view can restrict the stereo's ability to detect cross-camera features at close distances. To account for this, Gallup et al. suggests a setup with a dynamic baseline and camera-resolution which can be changed online[10]. By adjusting the baseline and camera resolution, their system can maintain sufficient disparity between the cameras across different scene depths. The result is an overall accuracy less affected by varying feature depths than a regular VIO system.

Another solution to altitude estimation for fixed-wing aircraft is proposed by [40]. Their solution is based on using a deep neural network to evaluate the height from the ground, based on images of the runway on which it is landing. With supporting inertial measurements, their network can predict accurate altitude estimates that significantly outperform standard VIO methods. An issue related to this type of solution is the need for training data for the network. If a UAV is to perform VTOL in unknown areas, there is no guarantee that the scene is relatable to the training data for the network.

4 System implementation

The implemented VIO system consists of two cameras and an IMU, all rigidly mounted on an aluminum rod as shown in fig. 7. All three sensors are connected through a USB interface to provide sensor data and meta information to an external host computer. The host computer runs serial drivers to read and process the data.

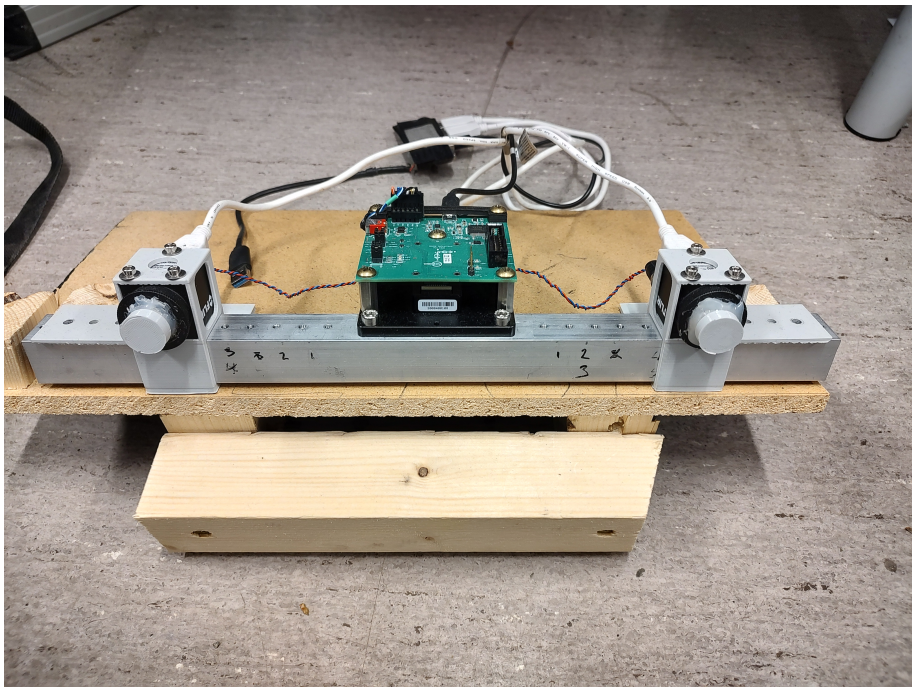


Figure 7: Prototype created for data collection

4.1 Hardware

Hardware synchronization

In VIO, synchronized data acquisition is key to high-performance systems. All measurements should be related to the same clock reference, as their timestamps relative to system startup are used in the estimation process. All measurements are triggered from the same microcontroller unit (MCU) to ensure this. The MCU is triggered to store a time stamp on an internal microsecond counter at the time of IMU acquisition. A trigger signal to the cameras is timed with the same counter to ensure that the cameras and IMU use the same clock reference. The setup is visualized in the block diagram of fig. 8. Note that the actual images are transferred by a direct serial interface between the computer and cameras, while the corresponding timestamps are transferred from the MCU. Images are associated with the correct timestamps in a synchronization node discussed in section 4.2.

Cameras

The system incorporates two Blackfly S USB3 cameras from Teledyne FLIR. These are color cameras with a resolution of 1.6 megapixels and a maximum frame rate of 226 frames per second. The camera is equipped with a USB connector used to transfer image data to a host computer and control commands to the cameras. The control signal is, among other things, used to reduce the resolution from 1440x1080 to 720x540 to reduce computational costs and set up the camera to be triggered by an external source. This trigger signal uses the general-purpose input/output (GPIO) connections available on the camera. Two such pins are used in the implementation, the

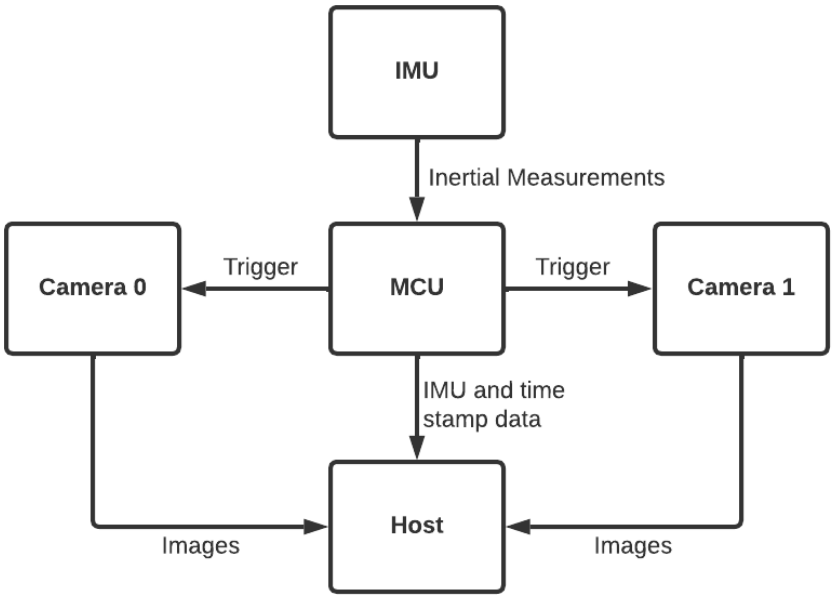


Figure 8: High-level diagram of hardware setup

opto-isolated input, and opto-isolated ground pins. The former is connected to a trigger GPIO pin on the MCU, the latter to MCU GND to ensure a common voltage reference between the two systems.

Evaluation board and IMU

The IMU and MCU depicted in fig. 8 are tightly integrated on a single circuit board developed by Aceinna. This is called an evaluation board, and it is produced to make system integration with the IMU easier for developers. It handles the low-level communication with the IMU and prepares data for transmission over a UART interface. The onboard MCU is programmed to adjust the data acquisition process. It is configured to read and transmit IMU measurements at 200Hz. At every 10th cycle, the MCU sets one of the GPIO pins high to trigger the cameras, yielding a camera frame rate of 20Hz. It also records the time of triggering, which is sent together with IMU measurements to the host computer over UART.

4.2 Driver

To collect VIO data, a host computer is needed to run the drivers for the setup. This is implemented using ROS (Robot Operating System) and consists of three different nodes, the *Blackfly Nodelet*, *OpenIMU ROS driver* and the *Synchronizer* node.

The *Blackfly Nodelet* is responsible for communication with the cameras. At system startup, the cameras are configured, and an image stream is set up between the cameras and the host computer. Some functionality has been added to the node, but the main content is taken from a Github repository from NTNU-ARL[26]. Images are read with their frame number, which is used to associate the image with a timestamp. An exposure-dependent offset is added to the timestamp after retrieval. By adding half the exposure time, the adjusted timestamp is placed in the middle of acquisition instead of the very beginning. This timing offset is illustrated in fig. 9, which also shows the temporal offset (TO). The temporal offset is not accounted for in the driver but in the ROVIO node and will be discussed further in section 4.3. The exposure-dependent offset is an essential adjustment in cases with fast-moving scenes where images are blurred due to the motion.

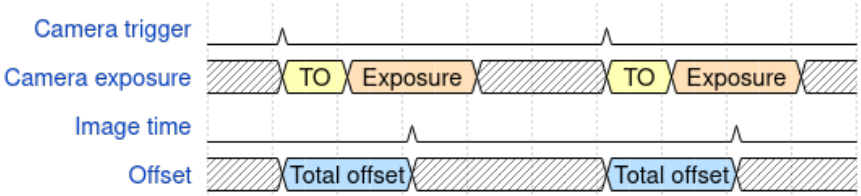


Figure 9: Camera timing diagram. The image measurement timing is affected by two offsets, the temporal offset, and an exposure offset. The temporal offset (TO in the diagram) accounts for delays between when the trigger signal is sent, and the camera starts exposing. The exposure offset is equal to half the exposure time of the camera.

Once the correct image stamp is obtained, it is published together with the image on a ROS topic.

The *OpenIMU ROS driver*[29] is a driver published by Aceinna, used to read IMU data in real-time with ROS. Some adjustments have been made to handle communication with the other driver nodes and signal the MCU control acquisition and triggering frequency. The node communicates with the evaluation board MCU over UART and collects both IMU data and timestamp information. During an initialization face, the driver waits for signals from the blackfly nodelet, indicating that a connection to the cameras has been established. Once all connections are established, the driver signals the MCU to start data acquisition and camera triggering. Together with their corresponding timestamps, IMU data are sent directly to a ROS topic. Camera timestamps and trigger count are also received. These are sent to the *Synchronizer* server, which handles the association between timestamps and camera measurements.

The last is the mentioned *Synchronizer* node, which consists of a server storing timestamps and associating them to camera measurements. This node is created to handle communication between the IMU and camera drivers. Whenever the camera driver receives an image, it requests its corresponding timestamp from the synchronizer node, using the image count as a key. These image stamps are stored in the synchronizer node when received by the IMU driver.

4.3 Calibration

Calibration is the process of finding system-specific parameters that should be used in the mathematical description of the system. Since the underlying calibration methods is considered out of scope for the thesis, they will not be explained in detail. A short summary of how calibration was performed will however be given.

1. IMU noise parameters
2. Camera intrinsic parameters
3. Camera extrinsic parameters
4. Temporal offset between cameras and IMU

The IMU noise parameters include white noise density and bias random walk of the accelerometer and gyroscope. The two different noise categories can be considered high-frequency noise and low-frequency drift that occurs in a signal. IMU measurements were recorded overnight for 18 hours without moving the sensor to estimate these values. The collected data was analyzed with the Allan Variance method discussed in [34] using the open-source library from [11].

The camera model uses intrinsic parameters to project 3D points onto the image plane. The intrinsic parameters describe focal length, optical center, and distortion for the pinhole model used in this project. Intrinsic calibration was performed using the calibration toolbox, Kalibr [9]. The intrinsic calibration procedure requires a set of images with a specific type of checkered pattern in the frame. A regular choice is the aprilgrid shown in fig. 10. With knowledge of the spacing and

size of the pattern, the toolbox can optimize the parameters by minimizing the reprojection errors of recognized edges from the target. It is essential to use a set of images such that the calibration target covers the entire image frame. This makes the distortion calibration robust in all parts of the image.

The extrinsic parameters describe the spatial transformations between the cameras and IMU. These transformations are essential to predict the movement of features in a camera frame. Without the spatial information of the system, one can not predict how the camera moves based on IMU measurements. The temporal offset calibration is closely related to the extrinsic parameters, as this indicates how much the time source for each camera deviates from the IMU. In our case, all sensors are stamped with the same clock, but there are still delays in the camera triggering process that need to be accounted for, such as internal signal filtering in the camera inputs. This temporal offset is illustrated in the timing diagram in fig. 9, denoted as TO. Both extrinsic parameters and temporal offset were calibrated using Kalibr[9]. A video sequence is required for this calibration, containing a target pattern as shown in fig. 10. It is vital that all IMU axis is adequately exited and that the movements are smooth.

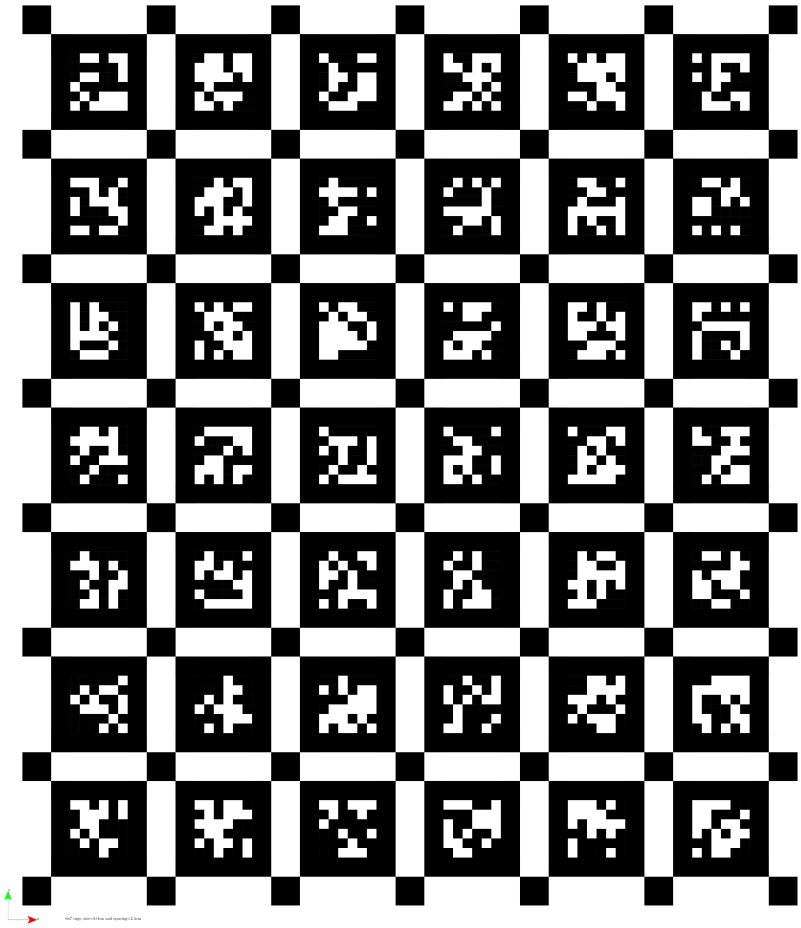


Figure 10: Aprilgrid target pattern (created with Kalibr[9])

5 ROVIO extension

Part of the thesis has been devoted to extending the ROVIO algorithm to take advantage of the specific problem of VTOL. By default, the method is a general-purpose algorithm, and so it does not apply prior knowledge about the pose trajectory or camera scenes. In VTOL scenarios, however, we can make a few assumptions about the geometry of the surroundings.

The extension is built around the assumption of a flat ground. Relative to the descending or ascending direction, usually the direction of gravity, the ground should appear as a normal plane. In addition to this, the extension assumes that there can be multiple such planes. This can, for example, be a flat platform on top of a flat rooftop and even include the flat ground on which the building stands. With this assumption in mind, we can utilize the fact that the vertical distance from the drone to each point has to correspond with one of the planes. This is illustrated in fig. 11, where we can see that the red outliers fail to fit either of the two heights. A thorough mathematical description of how this is done will be given, but some notation has to be defined first. In the rest of this section, ${}^{A}_{PQ}\boldsymbol{r}$ refers to a vector from P to Q, defined in a reference frame A. Furthermore, the rotation from a reference frame B to the reference frame A is defined as ${}^{A}_{AB}\boldsymbol{q}$. Using this formulation a vector can be transformed between reference frames by eq. (51).

$$ {}^{A}_{PQ}\boldsymbol{r} = {}^{A}_{AB}\boldsymbol{q}({}^{B}_{PQ}\boldsymbol{r}) \tag{51} $$

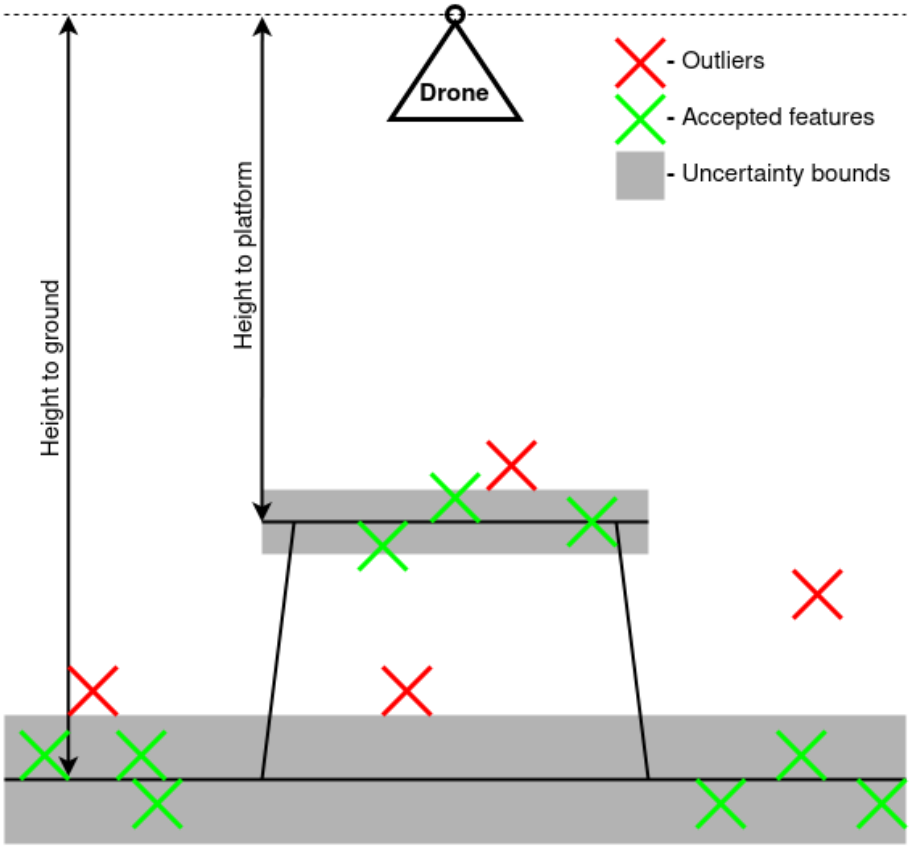


Figure 11: Visualizing the cluster filter. Features are filtered out based on their vertical distances from the system. Uncertainties in feature positions are neglected in this illustration, although they are accounted for in the actual implementation.

A general way to calculate the vertical distance between the drone and a feature P is necessary. Because of the robocentric formulation, the position of a feature P is given by its bearing vector and distance parameter. This is parameterized in the camera frame and the position of the feature can thus be written as a vector on the form ${}^{C}_{CP}\boldsymbol{r} = [{}^{C}x, {}^{C}y, {}^{C}z]$. Because the camera extrinsics are known, we can easily find the vector from the IMU to P in the IMU frame M as

$$ {}^{M}_{MP}\boldsymbol{r} = {}^{M}_{MC}\boldsymbol{q}({}^{C}_{CP}\boldsymbol{r}) + {}^{M}_{MC}\boldsymbol{r} \tag{52} $$

This vector can be transformed into the world frame W by applying a rotation with the robot orientation available in the filter state, denoted here as ${}^{W}_{WM}\boldsymbol{q}$

$$ {}^{W}_{MP}\boldsymbol{r} = {}^{W}_{WM}\boldsymbol{q}({}^{M}_{MP}\boldsymbol{r}) = {}^{W}_{WM}\boldsymbol{q}({}^{M}_{MC}\boldsymbol{q}({}^{C}_{CP}\boldsymbol{r}) + {}^{M}_{MC}\boldsymbol{r}) \tag{53} $$

With this description, the vertical distance h_P from IMU to a feature P can easily be extracted from the third component of ${}^{W}_{MP}\boldsymbol{r}$. This is because the world frame z-axis ${}^{W}\boldsymbol{z} = [0, 0, 1]$ is enforced to be aligned with the direction of gravity upon initialization.

With a formulation of the vertical distance of all the points, we can take the set of distances and perform a one-dimensional clustering algorithm to find the most reasonable distances for each plane. Only features with sufficiently low uncertainty-to-depth ratios are considered to make the clustering more reliable. The clustering is performed with a k-means algorithm which iteratively refines cluster positions and assigns points to the closest one[20]. This procedure is illustrated for the one-dimensional distance problem in fig. 12. The clustering procedure requires a predetermined number of clusters k for which a center value is assigned. With the position of these k centers, we have our best guess of where the different planes are located. The output from the procedure also includes an uncertainty measure of each cluster, defined as the standard deviation of all points assigned to each center. This is illustrated by the red and blue error-bars in fig. 12. Features whose depth is not close enough to the k values are filtered out. In this step, the uncertainty of clusters and feature positions are accounted for. If the scene contains a high spread of feature distances, the cluster filter will be less likely to neglect outlying features. For a feature P with a vertical distance h_P from the drone, the criteria for matching a cluster i with vertical distance h_i is given by

$$ ||h_P - h_i|| - 3\text{std}(h_P) - \text{std}(h_i) < 0 \tag{54} $$

where std() is the standard deviation of a variable. The equation shows that features with high variance have a low chance of being filtered away, giving newly initialized features a chance to converge in the filter. Hence, the feature is only considered an outlier if it converges to an outlying distance. Subtraction of the cluster deviation is used to penalize uncertain clusters. This gives some slack to the assumption of a completely flat surface and accounts for inconsistencies in the plane.

Updating the feature depth

In addition to detecting outliers, the cluster centers can provide a "ground truth" for the feature depths. After a feature is assigned to a cluster i, its depth could be readjusted to perfectly match the cluster center with a depth d_i. However, a correction step is implemented instead of completely adjusting the depth to give some slack to the planar assumption. The correction is based on a one-dimensional Kalman filter update, where the observed cluster depth d_i corresponds to the state measurement, and the current depth estimate d_p^- is analogous to the prior state estimate. The procedure is illustrated in fig. 13. To update the feature, we thus need to find the cluster depth estimate and its covariance. The depth can be found by scaling the feature bearing vector such that the vertical feature distance h_p equals the vertical cluster distance h_i. This gives the following expression for the cluster depth

$$ d_i = \frac{h_i}{h_p} d_p \tag{55} $$

Its standard deviation $\text{std}(d_i) = \sqrt{P_i}$ can be calculated by performing a measured depth calculation for a hypothetical cluster with center at a distance of one standard deviation from the original

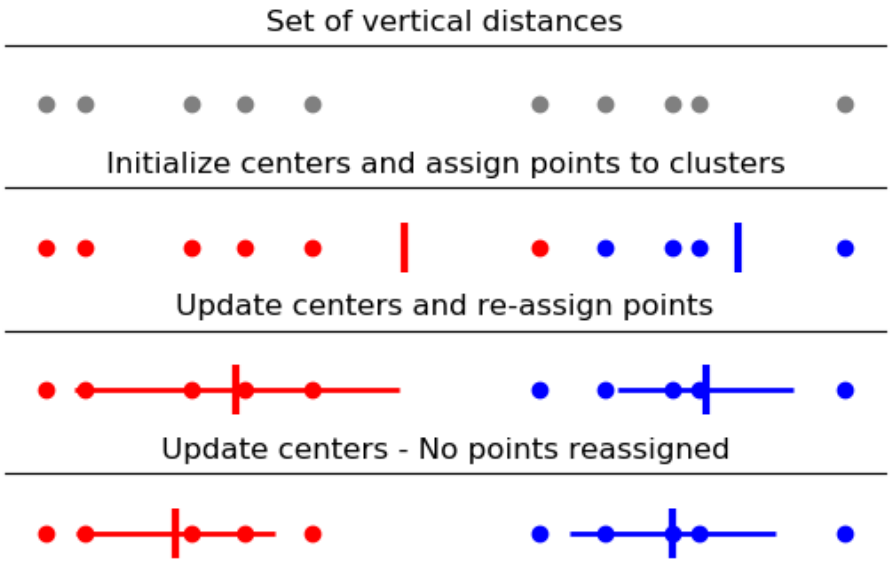


Figure 12: Illustration of k-means clustering for the 1 dimensional distance separation problem. Circles correspond to features vertical distances used in the clustering process. Error-bars indicate centers, together with their standard deviation.

center, lets call this d_{i+}. The difference between the two calculated depths is equal to the standard deviation of the measurement, so

$$P_i = \text{std}(d_i)^2 = (d_{i+} - d_i)^2 = \left(\frac{h_i + std(h_i)}{h_p} d_p - \frac{h_i}{h_p} d_p \right)^2 \tag{56}$$

Since the prior feature depth d_p^- and its covariance P_p^- is already known in the filter, we have everything we need to calculate the adjusted a-posteriori depth d_p^+. With these values in place, and the prediction value and uncertainty present in the filter, we can calculate the corrected depth by a standard Kalman update, given in eq. (57).

$$K = \frac{P_p^-}{P_i + P_p^-} \tag{57}$$

$$d_p^+ = d_p^- + K(d_i - d_p^-) \tag{58}$$

$$P_p^+ = (1 - K)P_p^- \tag{59}$$

To control the strength of the update extension, a positive scaling factor s is introduced to the calculation of the Kalman gain, making the gain less corrective for $s < 1$ and more aggressive for values of $s > 1$. The Kalman gain equation then becomes

$$K = s\frac{P_p^-}{P_i + P_p^-}, \quad s \geq 0 \tag{60}$$

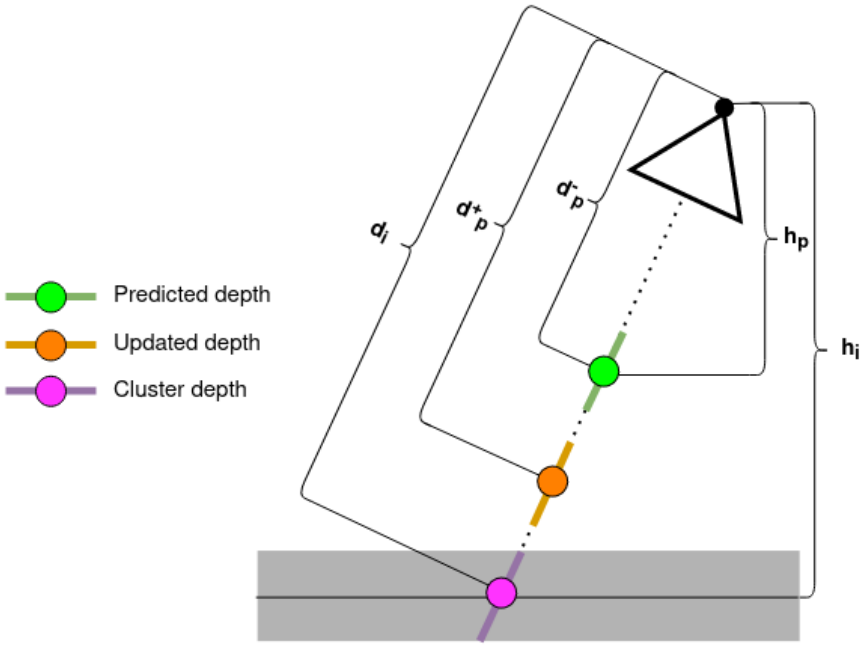


Figure 13: Depth update based on clustering measurement. Colored circles and lines indicate the respective depths and standard deviations. After a cluster detection, features assigned to the cluster can be updated in direction of the center. This is done by adjusting the predicted depth in direction of the cluster depth illustrated in the figure.

Adjustment for non-VTOL experiments

The formulation described above only works when the distance of interest is aligned with the direction of gravity. In experiments performed for this thesis, the system has been carried up to and away from vertical walls to simulate VTOL, so the calculations should be adjusted to find the distance in the direction of the wall instead. To make this adjustment, a change in the ROVIO initialization is necessary. The original method use a measurement from the accelerometer ${}^{M}\boldsymbol{f}_0 = [f_x, f_y, f_z]$ to initialize the rotation ${}^{W}_{WM}\boldsymbol{q}_0$ of the robot. ROVIO requires a z-axis ${}^{W}\boldsymbol{z} = [0, 0, 1]$ aligned with the direction of gravity, induced by the initial measurement ${}^{M}\boldsymbol{f}_0$ with the following equality

$$ {}^{W}_{WM}\boldsymbol{q}_0({}^{M}\boldsymbol{f}_0) = \| {}^{M}\boldsymbol{f}_0 \| {}^{W}\boldsymbol{z} \tag{61} $$

The problem with this in the experiments is that there is no way to tell how the robot pose will be aligned with the other axis ${}^{W}\boldsymbol{x}$ and ${}^{W}\boldsymbol{y}$. Small changes in roll and pitch upon initialization will therefore affect the initial yaw angle of the robot, meaning that there is no general way to say anything about a global distance in these directions. To account for this, an adjustment was made to the initialization process. This is essentially done by concatenating the original initial rotation ${}^{W}_{WM}\boldsymbol{q}_0$ with an alignment rotation ${}^{W}_{WM}\boldsymbol{q}_{align}$ defined by aligning the robot y-axis in the world frame ${}^{W}\boldsymbol{y}_m = {}^{W}_{WM}\boldsymbol{q}_0({}^{M}\boldsymbol{y}_m)$ to the world-frame x-axis.

$$ {}^{W}_{WM}\boldsymbol{q}_{align}({}^{W}\boldsymbol{y}_m) = {}^{W}\boldsymbol{x} \tag{62} $$

This results in an initial pose ${}^{W}_{WM}\boldsymbol{q}'_0$ defined as

$$ {}^{W}_{WM}\boldsymbol{q}'_0 = {}^{W}_{WM}\boldsymbol{q}_{align} \otimes {}^{W}_{WM}\boldsymbol{q}_0 \tag{63} $$

where $\otimes$ is defined as the quaternion product. With this adjustment, we can use the filter described earlier as long as we align the prototype with the direction of the wall from initialization. As long

as the IMU y-direction points against the wall, we can choose the first component of ${}^{W}_{MP}\boldsymbol{r}$ as the distance of interest for feature P.

6 Results and discussion

All results have been divided into a series of experiments. The goal is to keep the findings as tidy as possible so that it is easier to look at how individual changes affect system performance. The experiments test out various features to see how they affect the accuracy of the estimated trajectories. All collected data is meant to simulate a VTOL scenario, where different walls represent the ground. Experiments include trajectories where the system is carried up the wall to simulate a landing and back to the starting point, similar to a take-off.[1] The data should compare well with VTOL because most or all features appear on a flat surface at considerable depths. Although it should be possible to draw some conclusions on relevant issues related to VTOL, further research and testing should be performed with an actual aerial vehicle to retry the results.

Each of the following experiments will give a brief introduction to the motivation of the experiment, present some results, and discuss the observations. Since no ground truth is available, the evaluation metrics are somewhat limited. Therefore, most results are derived from internal estimation values and compared with real-life measurements in the testing area. Underneath is a collection of frequently used evaluation methods which will be used in several experiments. These are explained here to avoid repeating their intuition every time they are used.

Total position error

The first is a measure of the overall accuracy of a trajectory. Utilizing the fact that all trajectories have the same start and ending point, we can calculate the total position error by finding the position difference between the initial and final position estimates. Typically, these results will be presented as a scatter plot, where the x-value represents the baseline used for that particular run and the y-value represents the total position error. An example of this is given in fig. 14a.

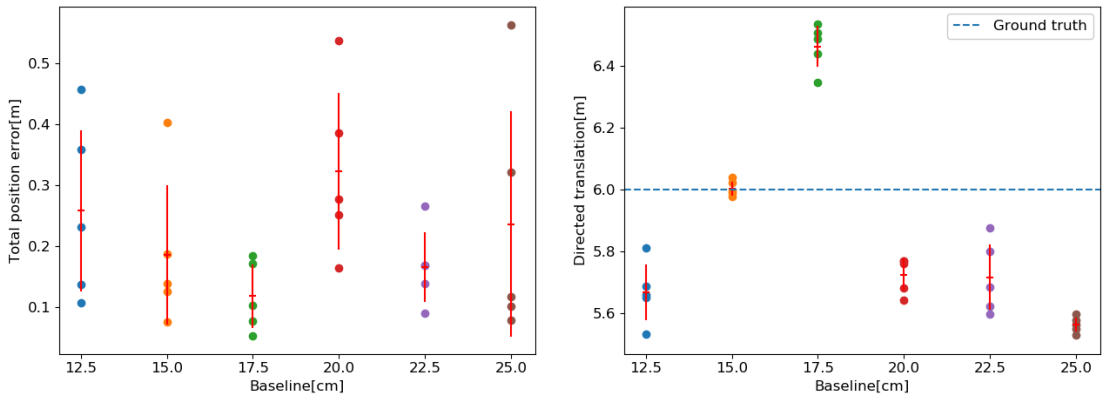
Directed translation plot

The second trajectory measure looks at how far the system is estimated to move in a specified direction, defined here as the *directed translation* of a trajectory. Most interestingly is the direction of the wall, which corresponds to the height trajectory of an actual VTOL scenario. This is illustrated to the left in fig. 15. Since all trajectories in a dataset follow approximately the same path, the point where the system turns around to go back to its origin is the same for all runs of the same trajectory. The distance from this point to the initial position has been measured physically, and can provide an approximate "ground truth" for evaluating the estimates. This information can be collected in a scatter-plot for directed translation as shown in fig. 14b. A typical use-case for this type of metric is to see how a trajectory scale is captured by the system. Even though a particular estimate has a very low total position error, it can have a very bad scale estimate, which indicates that the trajectory estimate is in fact quite bad.

Directed feature distance

Some of the results and discussions will be referring to *directed feature distances*. This is simply the feature depth, decomposed in direction of the wall. A visual illustration of this can be found in fig. 15. This is a useful way to measure feature positions in the following experiments because the distance between the cameras and the wall is usually known, albeit with limited accuracy due to texture in the wall an inaccurate measurements. Nevertheless, this information can produce approximate ground truths when analyzing the accuracy of a features estimated position.

[1] For a more visual representation of the collected data, please visit the video playlist on youtube at https://youtube.com/playlist?list=PLGvCS2eO6FFSZ15_l0nN_-fzCSyCDjXLt for video examples of the different experiments.



(a) Total position error example

(b) Directed translation example

Figure 14: Example figure, illustrating how typical figures might look like for (a) Total position error, and (b) Directed translation.

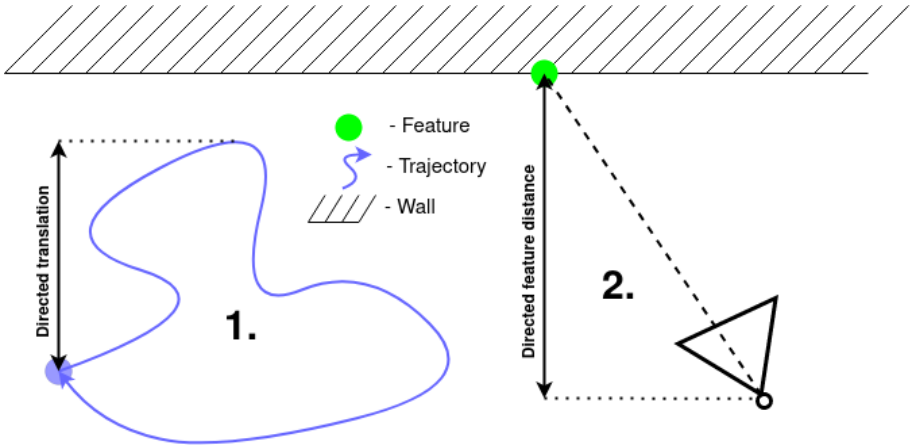


Figure 15: Illustration of the directed translation and directed feature distance metrics. The directed translation (**1.**) is a measure of how far a trajectory has moved in direction of the wall. The directed feature distance (**2.**) is a features depth, decomposed in direction of the wall.

6.1 Experiment 1 - On the effect of initial depth estimates

In general, it is always preferable to have as accurate initial conditions as possible when it comes to navigation. This also includes initial estimates of the feature in a VIO system. From the perspective of a camera, the bearing vector to a feature is well-defined on initialization, as it can be extracted from its detected pixel coordinate. The depth however is not observable based on a single image observation. In ROVIO, this is solved by giving a newly initialized feature either a fixed initial depth value or the median value of the other camera features, depending on how well the filter has converged. On initialization, the filter has not had time to converge, making the initial depth parameter the only viable option. In stereo systems, it is possible to adjust the value, however. If the feature can be found in both camera frames, it is possible to triangulate the depth of the feature as discussed in section 2.3. To match features between cameras, a feature bearing vector is transformed from one camera frame to the other. From this initial prediction, a local search for the feature is performed in the second camera. If the initial depth estimate is bad, we should expect the cross-camera transformation to be less accurate, negatively affecting the triangulation accuracy. Figure 16 illustrates how the initial depth estimates affects this cross-camera transformation.

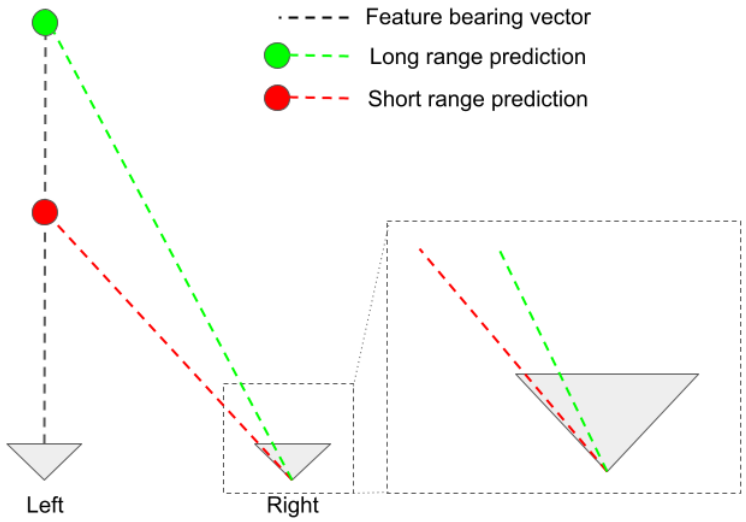


Figure 16: Illustration of how depth estimates affect the predicted bearing vector of a secondary camera. Depending on the left cameras depth estimate, the features bearing vector in the right camera frame changes.

To test this theory, data have been sampled from three different types of trajectories in front of the main building. All trajectories follow a straight path in direction of the main building wall. The system is carried up to the wall and back to the same starting point, with cameras constantly facing the building. The main difference between the three trajectories is their starting point. They are initialized from a distance of 10, 20 and 25 meters away from the wall. An example of a typical starting point for the 25m trajectory can be seen in fig. 17. Each of the three types of trajectories has been recorded five times with six different baseline configurations. Each run was evaluated with different initial depth estimates, ranging from 2 to 99 meters. The values tested are indicated on the x-axis of figs. 18 and 19.

An overview of how the initial depth parameter affect the triangulation can be found in fig. 18. Figure 18a shows how the number of triangulated features vary with the initial depth parameter while fig. 18b show the average directed distance of triangulated features. As a motivation for why the depth estimate matters, fig. 20 show how the predicted location of a feature after a cross-camera transformation is affected by the features depth estimate. Finally, an analysis of how the system's overall accuracy is affected can be seen in fig. 19, where the total position errors are plotted.

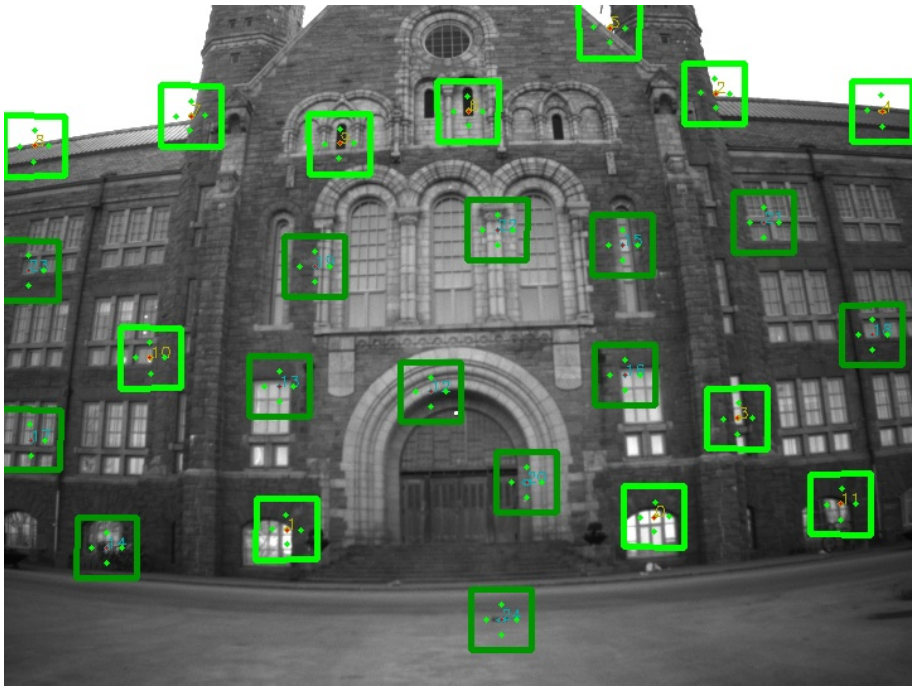


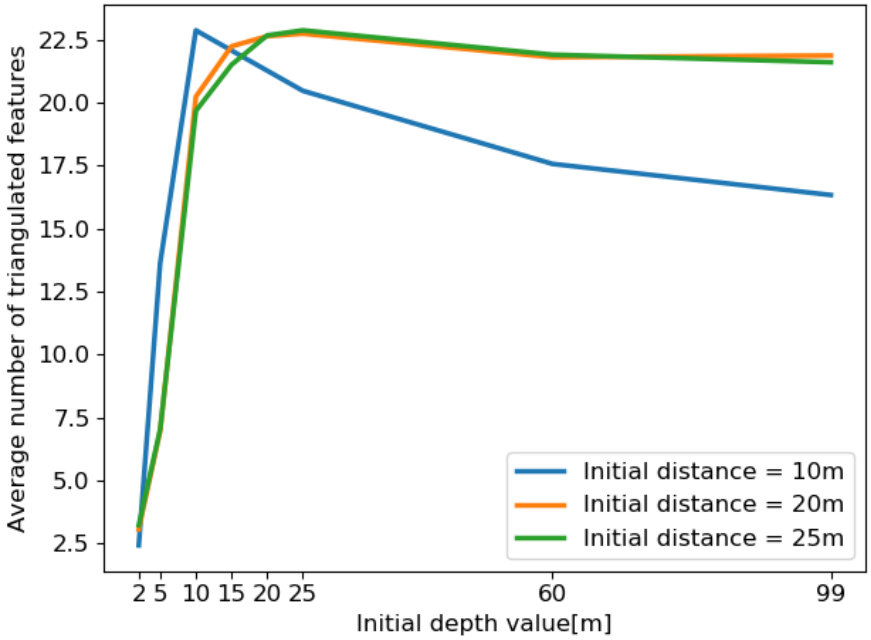
Figure 17: Main building from 25m distance. Each square represents the location of a multilevel patch in the image, and its color indicates which camera frame the feature is represented in.

Discussion

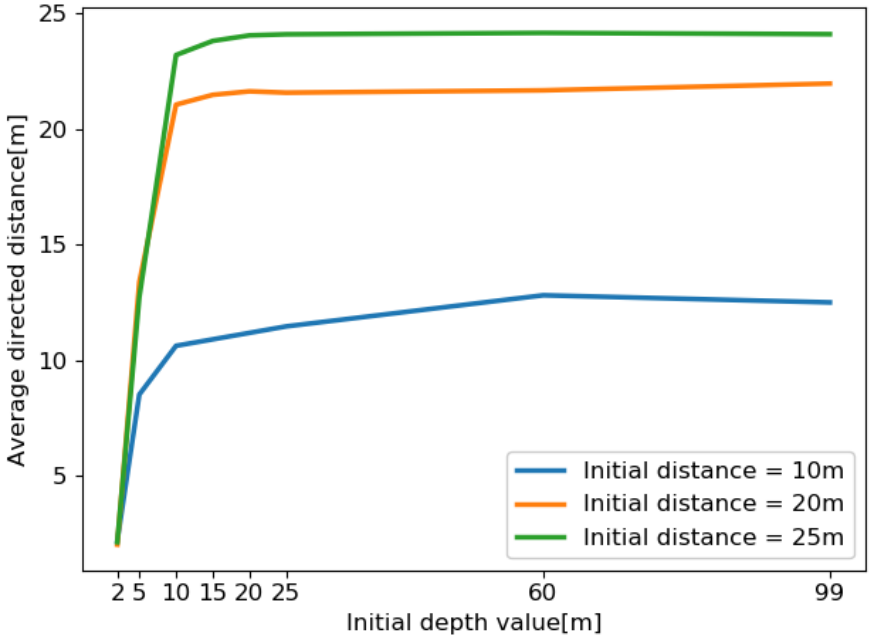
A visual analysis of fig. 20 clearly shows that a bad depth estimate negatively affects the predicted location of a left camera feature in the right camera. Running a patch alignment algorithm on the two predicted patch locations, using the red patch as a reference, it should be clear that the green prediction has the best chance of aligning correctly due to a superior initial condition. This implies that a good initial depth estimate yields a higher chance of finding a correct feature match and performing an accurate triangulation.

Support for this argument can be found in fig. 18, where it is clear that too low initial depth estimates lead to much fewer and less accurate triangulations. The triangulated depths are very low for low initial depth values, indicating that the patch alignment fails to find the correct patch and matches two different points in the scene. Interestingly, too high initial depth values do not significantly affect the triangulation accuracy. There is no apparent change in the number of triangulated features or their depths for the initial wall distances of 20 and 25 meters. This indicates that overestimating the initial depth estimates from large distances will have little effect on the triangulation procedure. This is in line with predictions from section 2.3, saying that the disparity approaches zero as the distance to features increases. With growing depth estimates, the predicted disparity also approaches zero, so the difference between the two becomes very small and a patch alignment is able to find the correct match. On the other hand, when the actual distance gets closer, the observed disparity increases, so the difference between predicted and observed disparity increases as well. This is a probable reason for why we see a decline in the number of triangulated features from 10 meters, as the initial depth estimate increases in fig. 18a.

If we further analyze the trajectory errors in fig. 19, we see that the average position errors more or less converge with increasing initial depth values. Since the trajectories starting from 20 and 25 meters can triangulate with the same accuracy for increasing depth values, this is not surprising. What may seem odd at first is that the trajectory from an initial distance of 10 meters also performs equally well with increasing initial depth values. One reason for this could be that ROVIO initializes the covariance of a feature based on its depth. A high initial depth value will yield low faith in the features from the start. The high covariance leads to a bigger trust in IMU measurements in the period before features can converge. Since this experiment is performed with



(a) Number of initially triangulated features as a function of the initial depth value.



(b) Average directed distances (as illustrated in fig. 15) of initially triangulated features as a function of the initial depth value.

Figure 18: Analysis of the how the initial depth value affects the triangulation accuracy from for the first set of images in a set of VIO datasets. The three lines correspond to three different types of trajectories, starting from 10, 20 and 25 meters away from the wall. Ticks on x-axis is only present for the tested initial depth values.

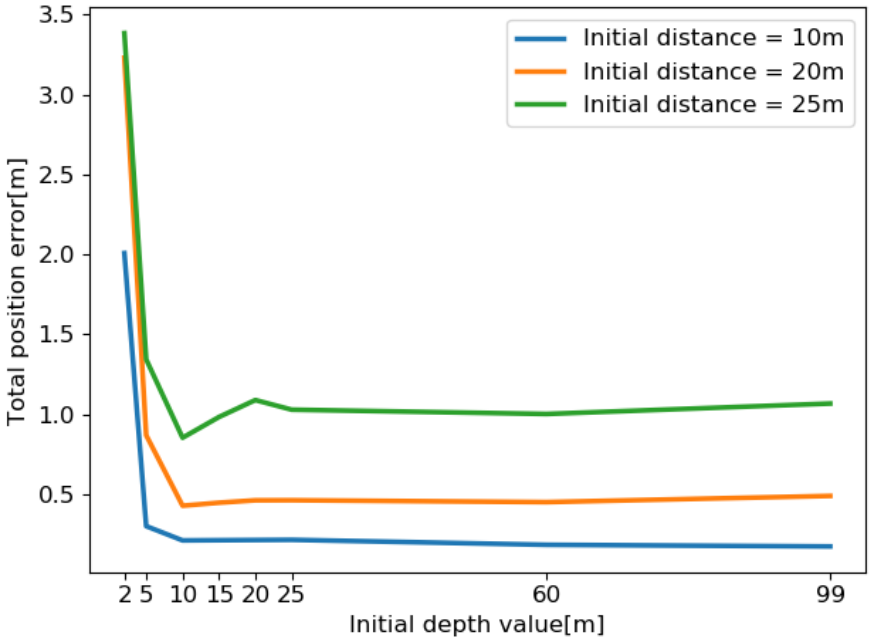


Figure 19: Average total position error as a function of the initial depth value. The three lines correspond to three different types of trajectories, starting from 10, 20 and 25 meters away from the wall. Ticks on x-axis is only present for the tested initial depth values.

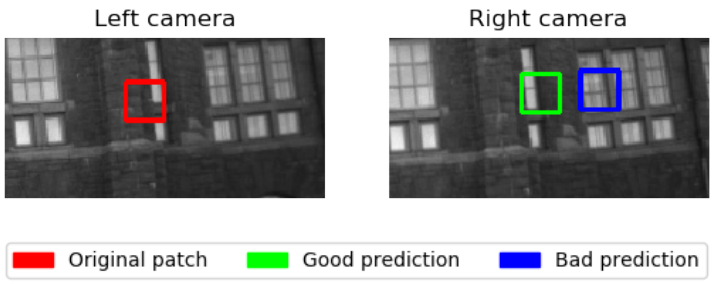


Figure 20: Visualization of the effect of depth estimates on cross-camera predictions from a wall distance of $\sim$ 25m. The colored square represent the correct size of the multilevel patches used in my experiments. The green and blue patches are predicted with feature depth estimates of 25m, and 2m, respectively.

an initialization maneuver (which will be explained further in the next experiment), the features converge very quickly despite their bad initialization, so the propagated errors are not very big. Similarly, we can see that the errors are very big when a low initial depth value is given. This is very likely because the depth covariance is assigned a proportionally small value. Bad initial values with low covariance is certainly not preferable, and introduce large errors into the filter. This is however a very specific problem to the algorithm. Slight changes in the source code of ROVIO should resolve some of these issues.

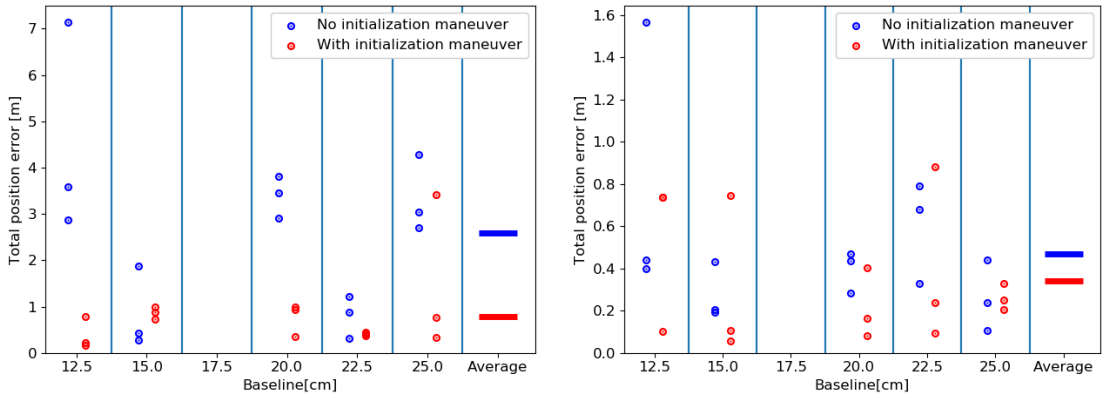
To summarize, it seems that the initial depth estimate indeed does matter. The results indicate that overestimating does not negatively impact performance too much, while underestimation can cause larger problems. At least, this seems to be the case with ROVIO, which is not customized to this type of trajectories. Two main problems occur with this algorithm related to VTOL. The first is the fact that the system perform local searches for patch alignment upon triangulating feature depths. This works well with good initial estimates of the features depth, but it may end in horrible feature matching depending on scene depth and depth estimate. The local feature matching alignment is justified by the fact that it reduces computational costs significantly to enable the system to run at a high frequency in real-time scenario. A suggestion to the specific problem of VTOL could however be to perform only the initial triangulation procedure with a wider, or even a global image search to ensure proper depth initialization of the features. The second problem related to ROVIO and VTOL seems to be the fact that the features depth covariance is heavily underestimated when given low initial depth values. A simple solution to this would be to provide a more conservative covariance to all features, not necessarily depending on the initial depth value.

6.2 Experiment 2 - On the effect of initialization maneuvers

From the previous experiment, we have seen that the initial depth estimates of features can significantly impact the accuracy of ROVIO. Unfortunately, it will not always be possible to have reasonable initial depth estimates. This experiment looks at introducing initialization maneuvers as an alternative way to quickly observe the depth of features and increase the accuracy and robustness of the VIO system.

The initialization maneuver is a procedure where the system is moved around such that the detected features are moved around in the image frame. This serves to converge the IMU biases and feature depth estimates in the filter. By moving the system, the filter can correct the prediction errors of feature locations by adjusting these parameters. As discussed in section 3, the observability of depth for stereo systems is reduced when the baseline-to-depth ratio becomes too small, as is the case in VTOL scenarios far from the ground. In this case, the only way to observe the depth of features is by moving and rotating the cameras, trusting that the filter can adjust the depths based on prediction errors. From large distances, small translations will have a relatively small effect on the images. Rotating the setup will however yield more movement of features in an image, and hopefully make them converge faster. The maneuver does not have to be very big, but sufficient enough to significantly move features around in the image.

To investigate the effects of initialization maneuvers, trajectories have been recorded from the same location with and without the maneuver. These trajectories resemble the data from experiment 1 in the way they approach a wall and go back to the same location, while facing the wall throughout the trajectory. The initial distance from the system to the wall was approximately 20 meters, and the turning point approximately 10 meters from the wall. This should yield an expected directed translation, as described in fig. 15, of ∼10 meters. Three such runs was collected for five different baselines (It was six until the data for baseline=17.5cm was corrupted). The total position errors for each run is shown in fig. 21, and the directed translation values can be seen in fig. 22.



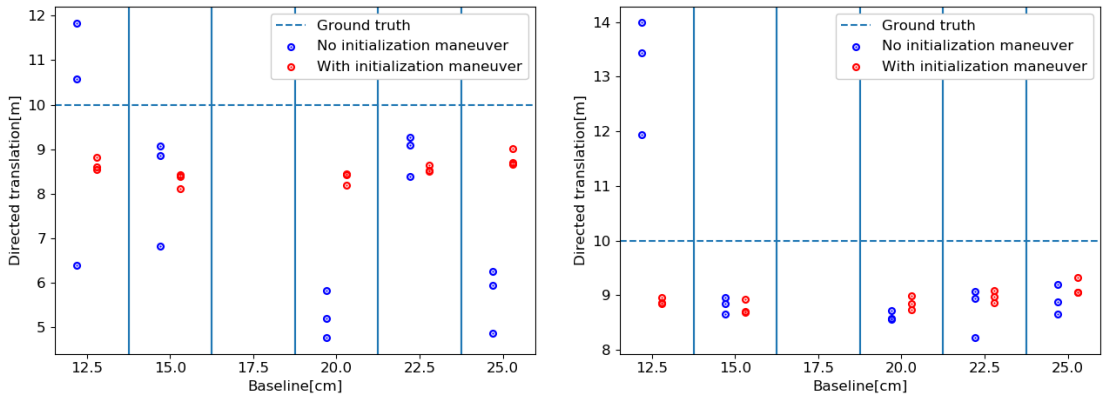
(a) 2m initial depth estimate (bad)

(b) 20m initial depth estimate (good)

Figure 21: Position errors for different initial depth values. Red and blue dots correspond to with and without initialization maneuver respectively. Each dot represents the total position error of a trajectory in the dataset. The right column marked *Average* visualize the average total position error over all trajectories.

Discussion

Looking at the position errors in fig. 21a, there should be no doubt that the initialization maneuver has indeed had a big impact on the accuracy of the system, with an initial depth value of 2 meters. The average total position error is reduced by a factor 3 across all the baselines. Comparing this with fig. 21b, we see that the effect is not as big when the initial depth value is set to 20 meters, which is approximately the distance to the wall. In this case, the systems position errors are comparatively low either way. The fact that we observe such a significant improvement when



(a) 2m initial depth estimate (bad)

(b) 20m initial depth estimate (good)

Figure 22: Directed translation values for different initial depth values as explained in the beginning of this chapter. Red and blue dots correspond to with and without initialization maneuver respectively. Each dot represents the total position error of a trajectory in the dataset.

the initial depth value is bad, and not with an appropriate value indicates that the initialization maneuver indeed has the advantage of compensating for bad initialization by updating and quickly converging the features after startup. The slight improved performance with good initial estimates can probably be explained by two factors: i) The initialization maneuver makes estimation of IMU biases possible, yielding higher accuracy IMU measurements from an early stage in the trajectory. ii) Although the initial depth value is good, it is not perfect for all features. Therefore, the initialization maneuver will still have a job of correcting for, albeit small, initial value errors.

The directed translation values in fig. 22 show how well the trajectory scale is captured, at least in the direction of the wall, which is the most interesting as it relates to the height of a drone in VTOL. In the case with an initial depth value of 2m (fig. 22a), some baseline configurations strongly underestimate the trajectory length. This is probably the result of low depth estimates, which corrupts the filter by asserting that the world is smaller than it is. The same scale errors are corrected when the maneuver is applied. In the case with 20m init depth, the difference is not as big, probably due to proper depth initialization of the features. However, we notice that the 12.5cm baseline strongly overestimates the scale when the initialization maneuver is not performed. There is a good chance that this data is erroneous and that the observed improvement is not a result of the maneuver. It seems highly unlikely that such significant errors should be observed when the initial estimate is this good.

A general observation from fig. 22 is that most of the trajectories seem to underestimate the trajectory. Even for trajectories with good initial depth estimates and initialization maneuvers, this is the case. We will see later, in experiment 4, that this is an effect that occurs only in the stereo camera case. The problem will be discussed further then, so it will not be discussed here.

From the results presented, it should be clear that applying an initialization maneuver can improve the performance of a VIO system. This has been shown, both as a reduction in overall position error, and from a more correct scale estimate of the trajectory. A probable cause for the improvement is the enhanced ability to quickly converge feature depths. Since this assumption does not rely on special attributes of ROVIO, it can be assumed that the conclusion is general for most VIO systems.

6.3 Experiment 3 - On the effect of different baselines

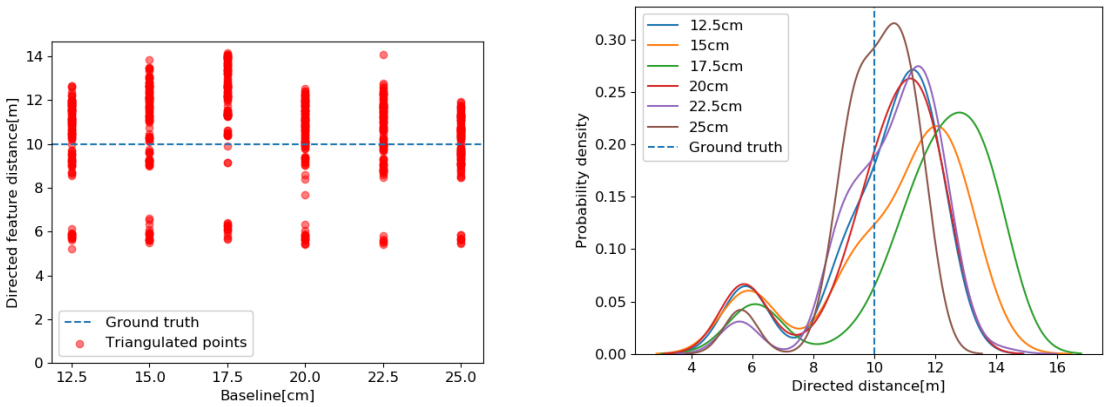
This experiment will look at how changing the baseline of a system might affect estimation accuracy. In theory, the baseline of the cameras restricts the ability to triangulate at higher depths due to the disparity restrictions discussed in section 2.3. The idea is to see if evidence can be found of these restrictions in the collected data and see how well this fits with theory. This analysis will be performed with the same dataset as used in experiment 1, with three different initial wall distances of 10, 20, and 25 meters.

The initially triangulated *directed feature distances*, as illustrated in fig. 15, for three different wall distances is shown in fig. 23. Only triangulated features from the very first frame is used. With 5 runs per baseline, and approximately 25 triangulated features for each run, the total set of triangulated features amount to almost 125 per baseline. This should be sufficient to see if there are clear trends in the data. From top to bottom, the triangulations from 10, 20, and 25 meters are visualized using two different plots. The left ones are scatter-plots over the directed distances, sorted by which baseline produced them. The fact that they are directed, means that we can use the measured distance from the system start to the wall as an approximated ground truth. On the right, these directed distances are converted into probability densities to compare the mean and spread more easily across different baselines. In addition to the analysis on feature triangulation, trajectory metrics for total position error and directed translation is included in fig. 24. These visualizations give accurate descriptions of the separate runs, but they are a bit noisy. To get a more general view of the evaluations, a plot of the average position errors and their corresponding standard deviations are shown in fig. 25.

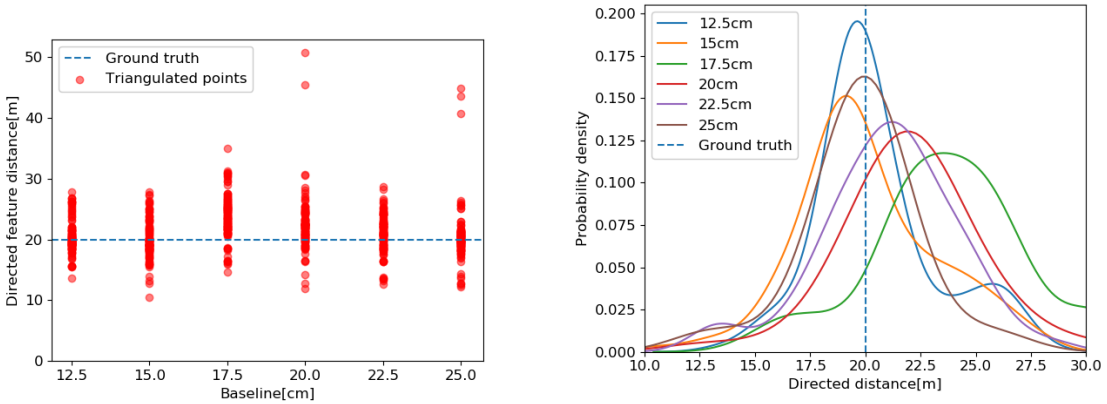
Discussion

At first glance, the different baseline performances in fig. 24 look quite sporadic. There is no clear pattern, at least not for the trajectories starting from 10 and 20 meters. It may be argued that the lower baseline in fig. 24c perform somewhat worse than the others, but it is really hard to tell the baselines apart from this data. Looking at the average error plots in fig. 25c, it does indeed seem to be a slight trend of improved performance with increasing baselines when the wall distance becomes sufficiently high. Since the two other average plots in figs. 25a and 25b seem to show no correlation, it is tempting to conclude that increasing the distance to 25 meters has made a baseline based performance effect appear. If we assume that the lower baseline is poorly affected by a lack of disparity, this should also be possible to observe in the triangulation plots in fig. 23c. From a distance of 25 meters, we can see that the smallest baseline of 12.5cm indeed underestimates the distances quite a lot compared with the other baselines. The same effect does not appear at closer distances in figs. 23a and 23b, making it plausible to assume that the increased wall distance has degraded the triangulation accuracy of the lowest baseline. A big problem with this assumption is the fact that there is no evidence of correlation between baseline and triangulation distance for the larger baselines. In lack of a more clear trend including the entire set of baselines, it is difficult to conclude on the effect based on the results presented here. Although the evidence is not quite adequate, it may work as a weak indicator. Since we only observe a slight tendency to deprecation for the lowest baseline from the most extended range, it may be the case that the experiments are just able to scratch the surface of where the effect starts to appear. For this reason, the study should be repeated with even lower baselines and longer feature ranges. This should be able to give more clear evidence of the effect if it is even present at all.

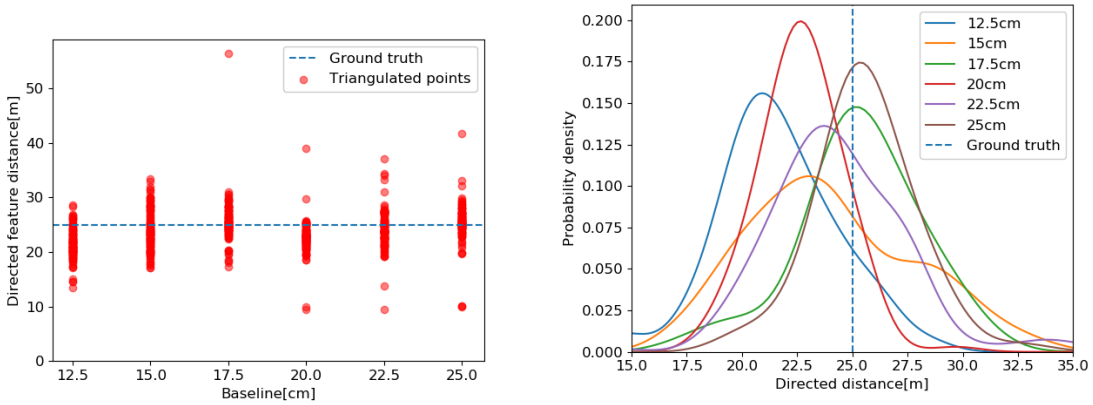
From fig. 24 we can see that the scale observed by the different baselines varies a lot. Furthermore, there is no clear correlation between the observed scale and a change in baseline. Since everything but the baselines remains equal between each trajectory, this may indicate an issue related to the extrinsic calibration. Slight changes in the camera mounts between calibration and data collection could lead to errors in the cross-camera transformations, resulting in unpredictable behaviour. Resolving this problem should yield more consistent results, and may even make the presence of a potential baseline effect easier to observe.



(a) Directed distance distribution from 10m

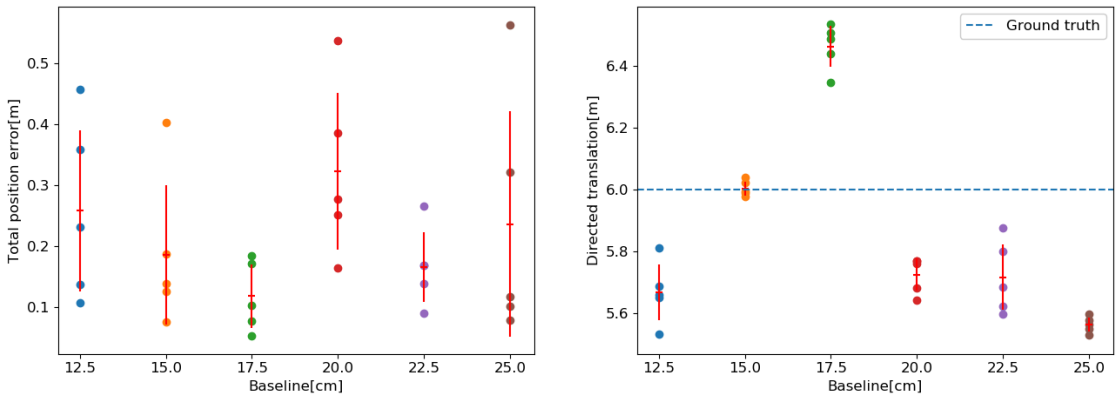


(b) Directed distance distribution from 20m

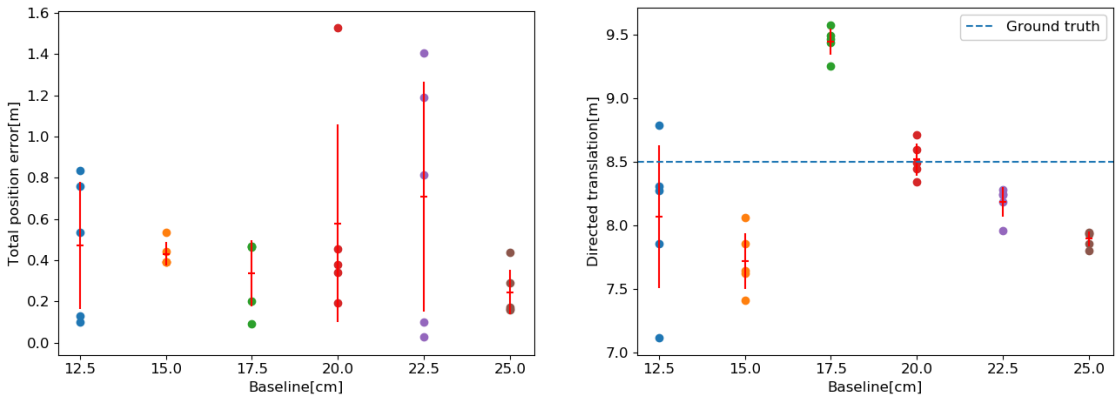


(c) Directed distance distribution from 25m

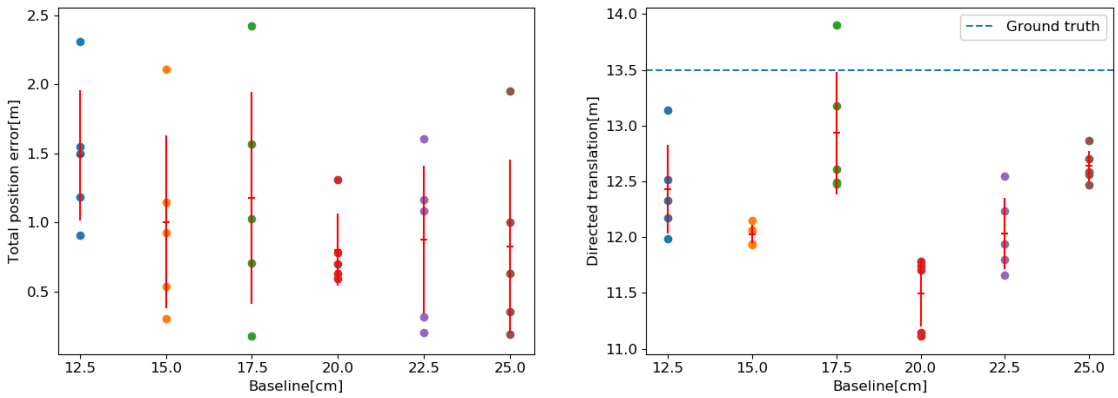
Figure 23: Initial directed distances to the wall for all triangulated features. To the left, the directed distance of each triangulated feature is scattered for every baseline. To the right, the distribution of directed distances is shown for each of the baselines configurations. Both figures include a ground truth marker, which is the measured as the shortest distance from the initial system position to the wall.



(a) Initial wall distance = 10m

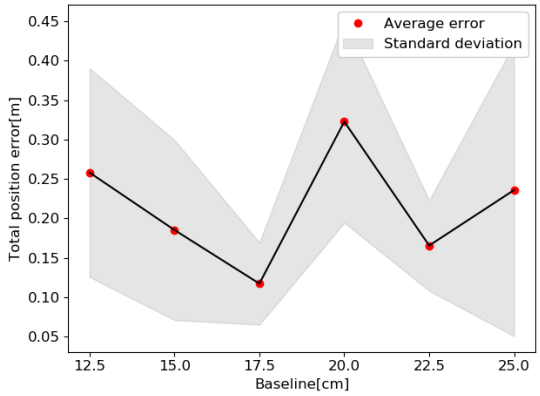


(b) Initial wall distance = 20m

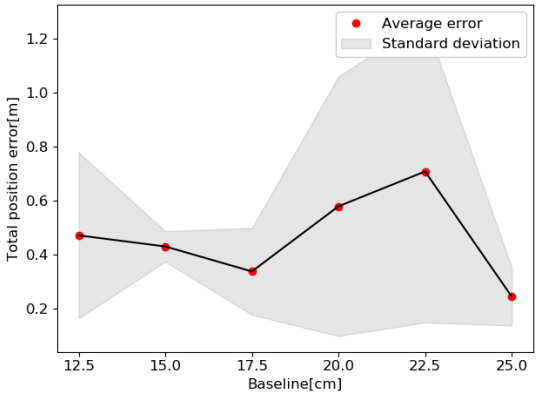


(c) Initial wall distance = 25m

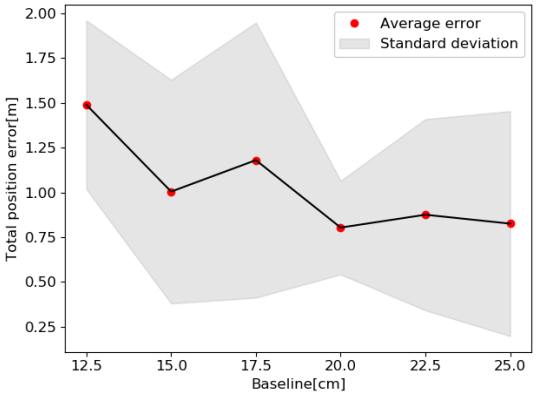
Figure 24: Trajectory evaluations for three different initial wall distance trajectories. Left plots show the total position errors of each run according to baseline. Right plots show the directed translation in direction of the wall for each run. The reader is referred to the beginning of the chapter for a more thorough explanation of these metrics.



(a) Average errors per baseline from 10m



(b) Average errors per baseline from 20m



(c) Average errors per baseline from 25m

Figure 25: Average accuracy of the system as a function of the camera baseline. Red dots indicate the average total position error for a specific baseline. The gray fill covers an area of one standard deviation away from the average value.

6.4 Experiment 4 - Monocular and stereo performance

This experiment will investigate the performance of monocular systems against stereo systems. In addition, evaluations of a stereo system where the cameras are treated separately as two monocular cameras will be included. This means that features are not mapped from one camera to the other during the filter update. The cameras rather keep track of individual set of features, and so the camera update is treated as two separate monocular updates. This type of configuration will be referred to as a non-cross-camera stereo system. The three configurations are all investigated using the same dataset. This is done by ignoring one of the cameras in the monocular case. The datasets is the same as in experiment 1 and 3, from the front of the main building at Gløshaugen and include trajectories starting from 10, 20, and 25 meters away from the building.

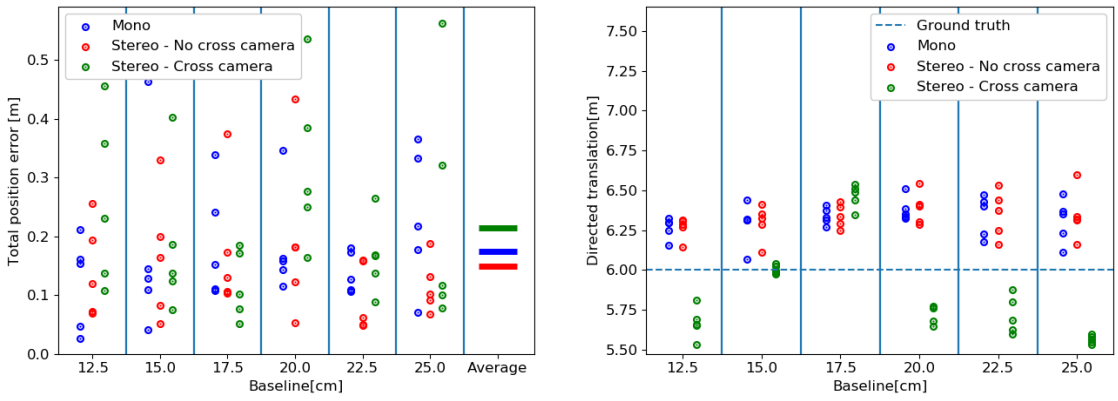
Trajectory data from this experiment can be found in figs. 26 and 27, where the initial depth value is respectively set to 2 meters and the initial distance to the wall, respectively. The left plots in both figures correspond to the final position error of the trajectories. Right plots illustrate the directed translation, as described in fig. 15, compared to the ground truth marked as a dashed line. All figures include a color-coded system where each color corresponds to a different camera configuration specified in the caption. In fig. 28, one example run for each of the trajectories is used to illustrate the perceived wall distance for each of the camera configurations. This wall distance is calculated as the average directed feature distance (as in fig. 15) for each image frame in the trajectory.

Discussion

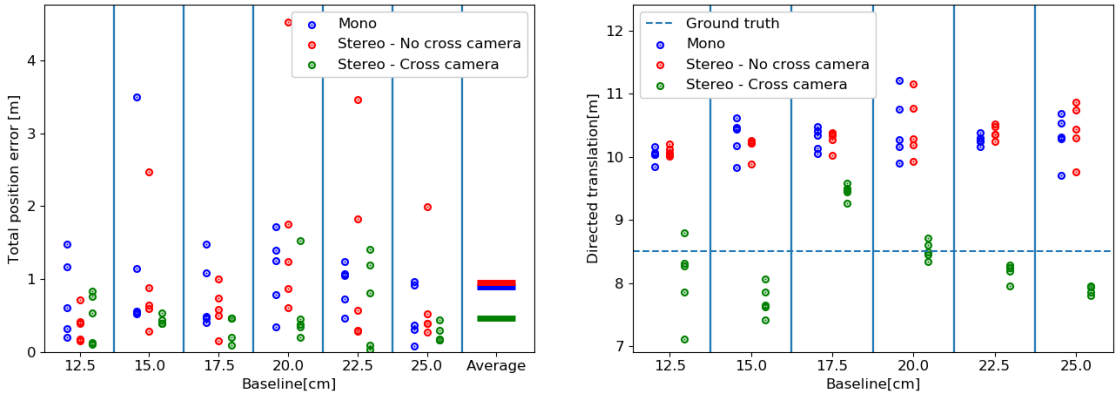
From the directed translation plots in figs. 26 and 27, we can see a pretty clear trend where monocular and non-cross-camera estimations repeatedly overestimate the trajectory length, while the cross-camera stereo configuration underestimates it. This is probably a result of wrongful estimation of feature distances, as observed in fig. 28. We see here that the stereo system in general estimates a lower directed features distance than the monocular and non-cross systems. One reason for the high predicted depths in the monocular and non-cross case can be a lack of feature motion. When the scene has low motion in the camera frame, it is difficult for the filter to correct the feature depths. In the stereo case, each feature is constantly updated by a disparity observation between the cameras, making it less dependent on movement in the scene. It does however seem from the consistently underestimated directed translation values in figs. 26 and 27 that this perpetual stereo correction is slightly harsh, and ends up underestimating the feature depths.

Interestingly, it is challenging to see any significant difference between the monocular and non-cross-camera stereo performance. This is likely because the stereo pair is reduced to a system with two monocular cameras when cross-camera measurements are not used. The two cameras do not complement each other, so the information gain from using both cameras is very low. Especially considering that their fields of view are very similar.

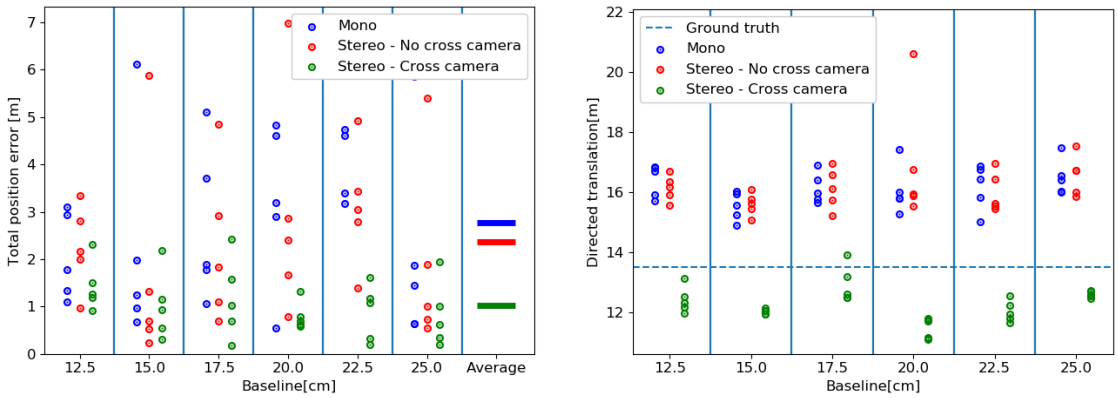
When it comes to accuracy, the better configuration choice seems to depend strongly on the initial depth estimate and distance to the ground. As shown in fig. 26, the stereo system with cross-camera measurements outperforms the others with accurate initial depth estimates when it comes to total position errors. This is not obvious from the trajectory starting at a 10m distance in fig. 26a, but becomes apparent with the longer-range trajectories in figs. 26b and 26c. This indicates that the stereo advantage is higher with large distances, possibly because the depth is more accurately estimated. Since long-range objects move less in the scene than closer ones, the updates of long-range features will be less robust to the presence of image noise and thus less accurate. Introducing a second camera with cross-camera measurements can counteract this effect. If the first camera performs a bad update, the second camera can correct for this damage. This dynamic also makes it possible for the cameras to determine the depth of features with little movement. Studying the wall distance estimates in fig. 28 we find supporting evidence of this. It seems that the green lines, corresponding to stereo estimated wall distances, is less noisy than the others. It is not unlikely that this is a result of reduced noise as a side-effect of having two cameras observe the same feature.



(a) Trajectory starting 10 meters away from the wall

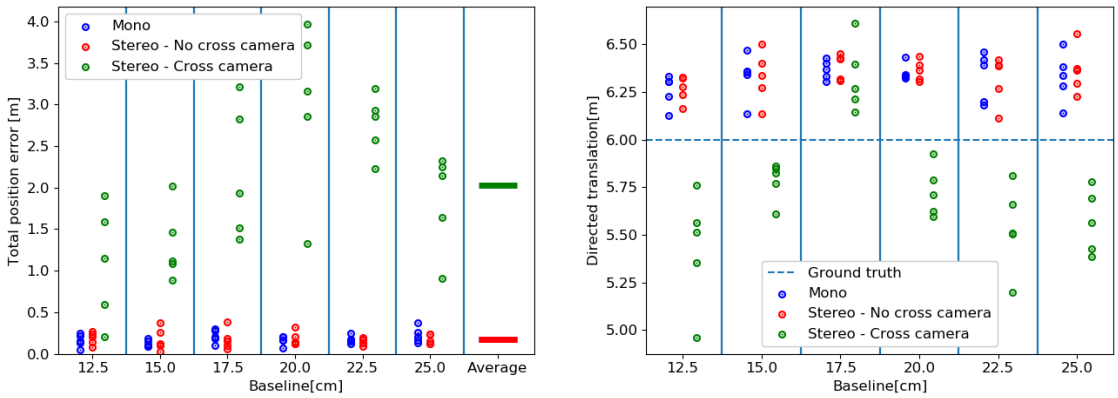


(b) Trajectory starting 20 meters away from the wall

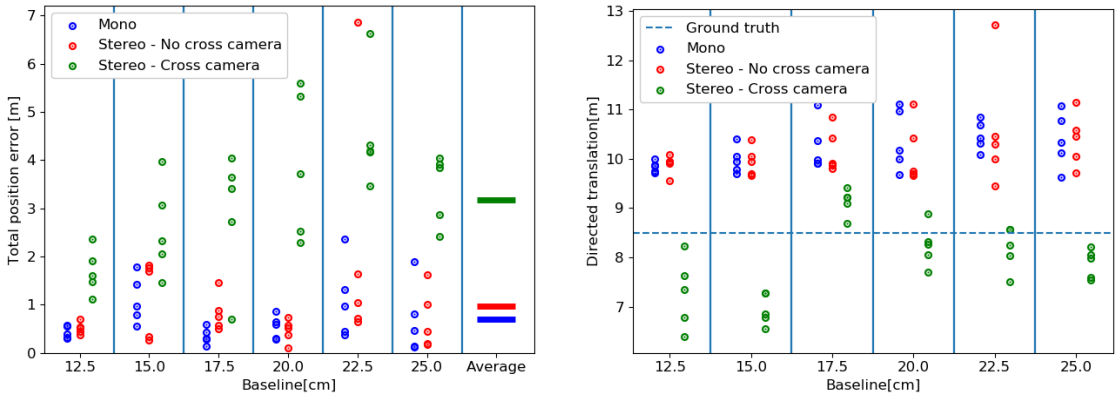


(c) Trajectory starting 25 meters away from the wall

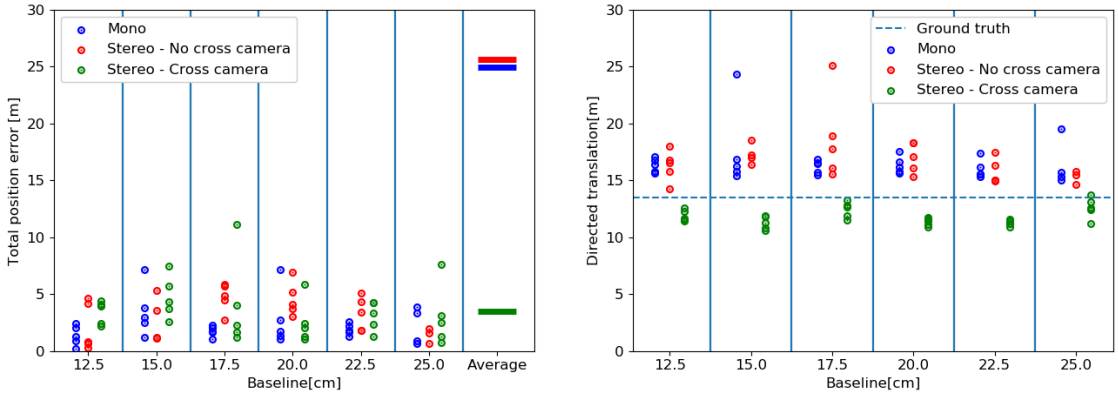
Figure 26: Position errors and directed trajectory lengths with good initial depth estimates corresponding to the actual distance to the wall. The colors represent three different camera configurations. Blue dots represent values from a mono-camera evaluation, red dots from a stereo camera evaluation without cross-camera measurements, and green dots represent the classic stereo system with cross-camera measurements.



(a) Trajectory starting 10 meters away from the wall

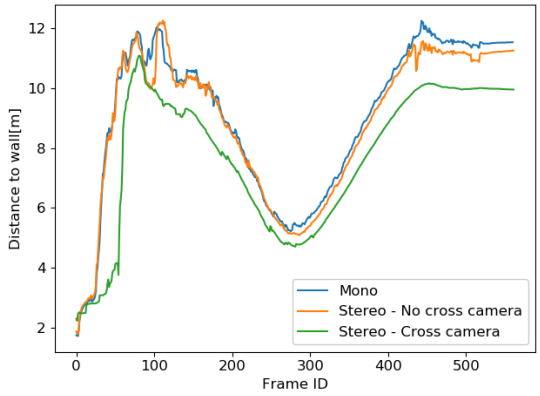


(b) Trajectory starting 20 meters away from the wall

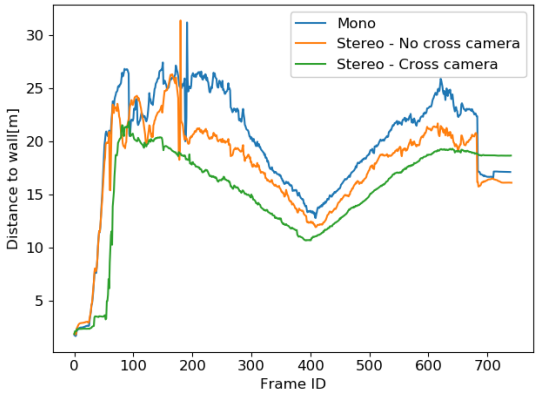


(c) Trajectory starting 25 meters away from the wall

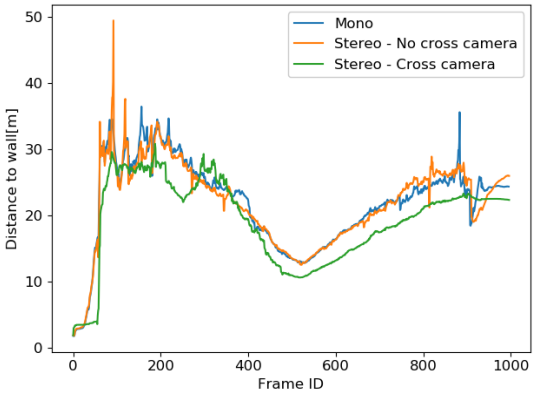
Figure 27: Position errors and directed trajectory lengths with bad initial depth estimates of 2m. The colors represent three different camera configurations. Blue dots represent values from a mono-camera evaluation, red dots from a stereo camera evaluation without cross-camera measurements, and green dots represent the classic stereo system with cross-camera measurements.



(a) Initial wall distance = 10m



(b) Initial wall distance = 20m



(c) Initial wall distance = 25m

Figure 28: Estimated distance to the wall for example trajectories from 10, 20 and 25 meters. This estimate is found by taking the average directed distance of all features in each frame. In these plots, the initial depth estimate of all features is set to 2 meters.

Moving to the scenario of bad initial depth values, we see an apparent negative effect of having cross-camera measurements in fig. 27. The total position error for the stereo system is quite big, even for the trajectory starting from only 10 meters. One reason for this may lie in the erroneous cross-camera feature transformation. With bad depth estimates, the features are transformed into a secondary camera frame at completely wrong pixel coordinates. When the direct multi-patch search is performed, there is no way to guarantee that it finds the correct feature but instead updates the filter after a completely wrong patch alignment. We can also see from fig. 28 that the stereo system use longer time to converge than the other configurations. This is a probable cause for high errors in the stereo systems, if bad initial depth values are given. However, the stereo system seems to have a much lower total position error than the other configurations, even with bad initial depth estimates, when the initial wall distance becomes large. This is seen in fig. 27c, where the average position errors for monocular and non-cross systems are extraordinary high. This is caused by diverged trajectories, outside the scope of the plot. This indicates that even though the stereo system is slower to converge with bad initial estimates, it is more robust to diverging filter states due to their cross-camera measurements.

An important thing to consider from these results is the fact that the stereo system with cross-camera measurements show sporadic and highly varying difference between the different baselines. This is for example apparent in fig. 26b, where the directed translation is greatly overestimated with a baseline of 17.5cm. Since this difference is completely removed without cross-camera measurements, it is reasonably to assume that the error stems from a discrepancy between the camera mounting and calibrated extrinsic parameters. This discrepancy has probably occurred because the camera mounts have been frequently readjusted between baseline configurations after the initial calibration was completed. Only the slightest difference in remounting the cameras can cause significant model errors. If the camera mount is skew, with respect to the calibrated orientation, the error introduced will increase with the distance to features as illustrated in fig. 29. Accurate calibration is therefore crucial in a VTOL scenario with large feature depths. The observed calibration issues must have been present in all experiments presented in the thesis. It is probably most relevant in this comparison and in the analysis of different baselines because the experiments are inherently built around assumptions of accurate cross-camera measurements. However, with knowledge of this systematic error, all results should be interpreted accordingly.

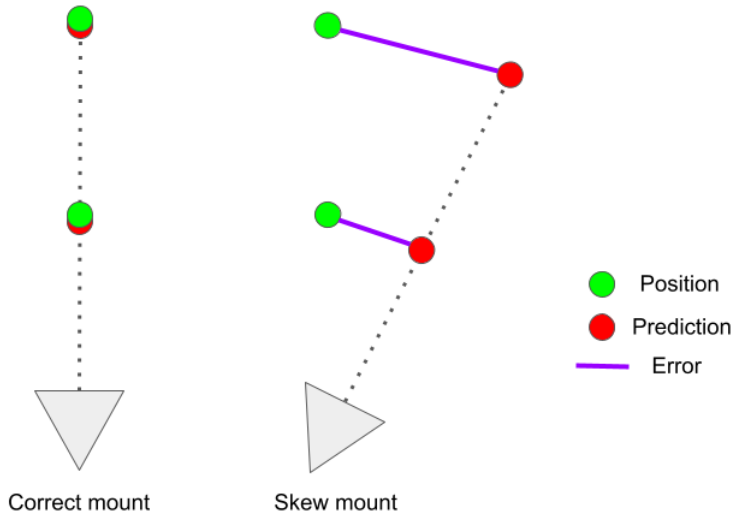


Figure 29: Illustration of how mounting errors can increase with distance to a feature. When a camera is mounted with an error, such as the skew illustrated, the predicted bearing vector to a feature after a cross-camera transformation does not correspond with the actual bearing to the feature. The prediction error of the feature location grows with the distance to the feature. This illustrates the importance of accurate calibration when performing cross-camera measurements.

6.5 Experiment 5 - ROVIO extension analysis

This experiment has the purpose of evaluating the implemented ROVIO extension described in section 5. The extension is based on the assumption that all features lie on a set of planes, orthogonal to the direction of gravity. Since this direction is not well-defined in the datasets, the IMU's initial pose is used to define the direction of interest. In the following data, two versions of the extension have been tested. The first is a simple outlier detection scheme that detects and removes features that do not fit the plane assumption. The second version perform the same outlier detection, but goes on step further by updating the feature depths in direction of its closest cluster.

To analyze the assumption of a single plane, the trajectories in front of the main building starting from initial distances of 10, 20, and 25 meters are used. This is the same dataset as used in experiments 1,3 and 4. Since the extension is made to handle an arbitrary amount of planes, a couple of runs have also been collected from a scenario where there are two distinct planes at different distances. These trajectories contain a set of containers up close at $\sim$10 meters from the initial position, and a big wall in the background at a distance of $\sim$25 meters. Some example images from both the single plane and double plane trajectories are shown in fig. 30. In the single plane visualization, shown in fig. 30a, yellow patches are used to indicate the features that are included in the cluster. With two planes as seen in fig. 30b, yellow and cyan patches are used to indicate a features assignment to the respective clusters. In all images, other colors indicate a lack of cluster assignment.

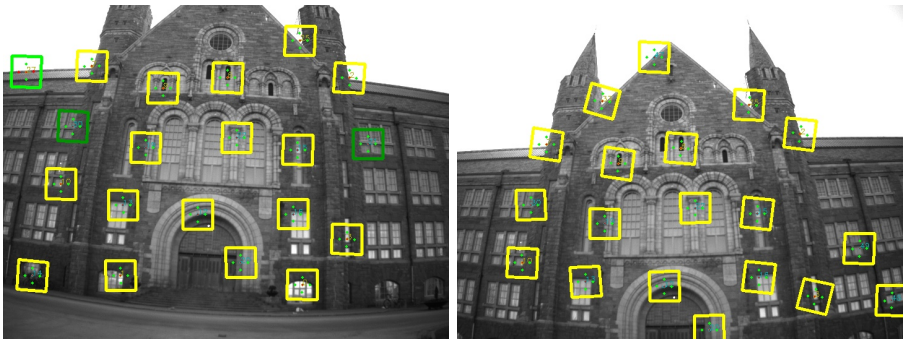
A summary of the total position errors in the single plane case is summarized in fig. 31. Trajectory evaluations are sorted into different baseline configurations to make it more interpretable how the different trajectories are affected by the extensions. An extra column showing the overall average is shown to the far right of each figure as well. Some of the markers are difficult to see in this column because they lie on top of each other, meaning their average performance is close to identical. Accuracy results for the trajectories with two planes are visualized in fig. 32. Since this experiment contains less data, each run is shown separately to clarify how the extensions affected each run. As in fig. 31, the average performance is summarized in the rightmost column.

Discussion

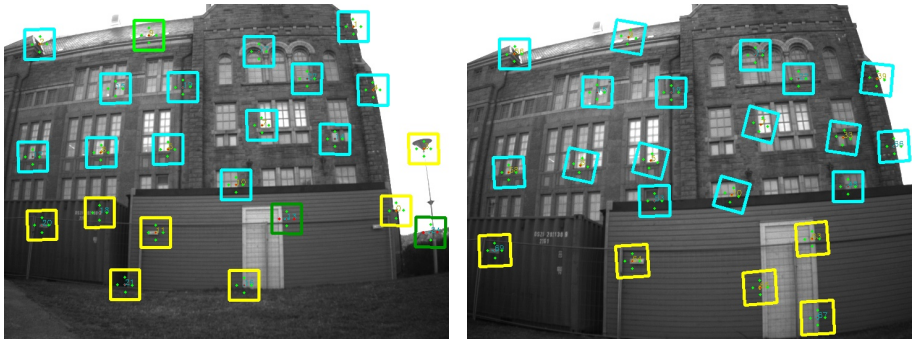
Figure 30 can be used to understand some of the properties of the cluster extension. We can see that not all features are assigned to a cluster, yet they are not discarded as outliers. This is the case for some of the points in the left images of figs. 30a and 30b. The reason why the features are not immediately discarded for not being assigned to a cluster is because their uncertainty-to-depth ratio is too high. This is an important property of the extension, as newly initialized features are by nature highly uncertain, and labeling them as outliers immediately is therefore not preferable. They should only be considered outliers if they do not match a cluster center after converging in the filter. We can for example see that one of the non-assigned features to the left in fig. 30a indeed is included in the cluster after converging in the right image.

From fig. 30b, it is clear that segmentation between more than one plane is possible with the extension. Both images show that the filter has no problem separating features on the back wall from features on the container. We can also see here, in the left image, that a feature on the lamp-post is included in the front cluster. Although this may seem strange, it is mathematically sensible, because the lamp-post is positioned approximately as far away as the containers.

Quantitative results from fig. 31 show an interesting tendency. The extended algorithm performs worse than the original from an initial distance of 10m (fig. 31a). From the 20 meters there are no significant changes (fig. 31b), and from 25 meters, the extension seems to be performing somewhat better (fig. 31c). A possible explanation for this is the fact that the underlying assumption of the extension breaks down when the system is closer to the wall. Significant indents, windows and other texture has a relatively bigger effect on the directed feature distance from close-range than from a long-range. For example, the doorway of the building is indented by a meter or so. From a distance of 25 meters, this corresponds to an error of $\frac{1}{25} = 4\%$, but from a range of 10 meters, the same error is $\frac{1}{10} = 10\%$.



(a) Two images from the same trajectory with a single plane.



(b) Two images from the same trajectory with two planes.

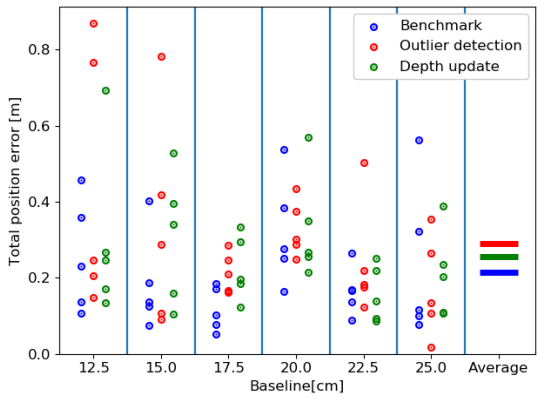
Figure 30: Visualizing the cluster assignments for (a) A single plane trajectory and (b) A double plane trajectory. Left images are taken in the early stage of a trajectory, while the right images are taken at later times, after features have had time to converge. Yellow and cyan are color-codes, indicating what cluster a feature is assigned to. Green patches indicate that the feature is not assigned to any of the clusters.

Analysis of the double plane case in fig. 32 shows that the extensions indeed perform well when introducing a second plane. This is supported by the accurate segmentation ability seen in fig. 30b. In this case, the closer plane of 10 meters is relatively flat compared with the wall in fig. 30a. The background wall is a bit noisier, but this is sufficiently far away for the plane assumption to hold. Overall, the extensions perform well in this scenario, although more data should be collected from different environments to investigate this further.

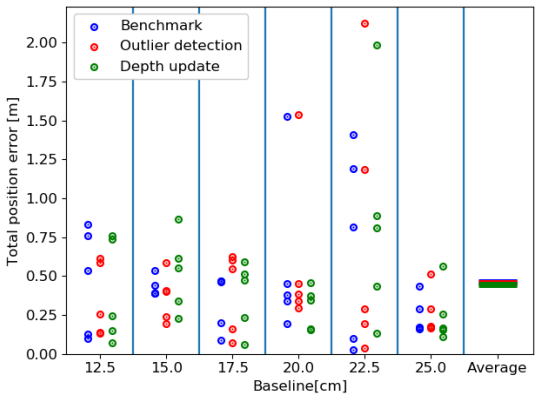
We can see from fig. 32 that the depth update extension performs worse than the benchmark on *run 2*. This motivates for a further analysis of the spesific trajectory. The estimated cluster centers in fig. 33c show that there is a point in the trajectory where the plane segmentation fails, around frame 350. An image from this area, is shown in fig. 33a, where the source of error can be identified. All detected features lie on the wall in the background, so the assumption of two distinct planes break down. Since the extension works with a predetermined number of cluster, in this case two, the algorithm fails if features are only detected in a single plane. We can see from the rest of the graph in fig. 33c that the issue is only temporary, and that the cluster segmentation successfully recovers after a few hundred frames. This is supported by visual inspection of fig. 33b, where the two planes are clearly separated again. This effect motivates the idea of extending the algorithm to automatically detect the number of clusters, or in the very least detect when such scenarios occur. This can for example be done by asserting a sufficient separation between the two clusters, such that the filter extension is disregarded if the clustering algorithm returns two centers with a difference below a certain threshold.

Another interesting observation from fig. 33c is the fact that the cluster centers are only valid when their deviations are sufficiently low. This is because the cluster considered too uncertain to be taken into account, and so the filter extension is not performed.

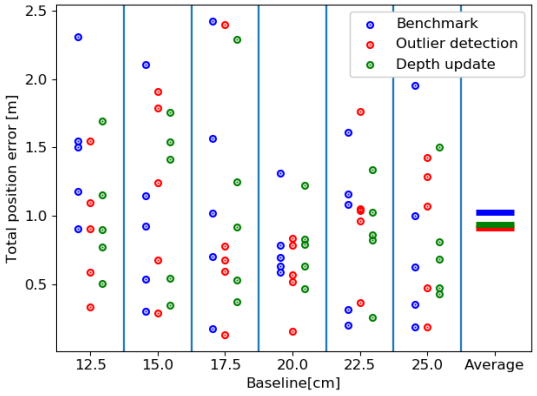
In conclusion, the extension shows some promising results, at least with sufficient feature depths



(a) Initial wall distance = 10m



(b) Initial wall distance = 20m



(c) Initial wall distance = 25m

Figure 31: Comparing accuracy's for the standard ROVIO algorithm with my two extensions. Every point corresponds to the position error of a single run. The average errors are summarized in the rightmost column.

when the underlying assumption remains accurate. It is important to keep in mind here that the procedure can be somewhat negatively impacted because the assumed planes' direction is not well-defined. Since the direction is calculated based on the initial pose of the system, some error is bound to be introduced. Even if the system was initialized perfectly, the direction would drift over time with the yaw angle, which is not observable in VIO as discussed in section 2.2. In the case of VTOL, this would not be a problem. Due to gravity, the direction of the planes will always be easily observable. It should also be mentioned that the results presented here stem from a very limited dataset, in terms of variety and size. To further conclude on the extension performance, larger datasets should be collected from a much more diversified set of scenes.

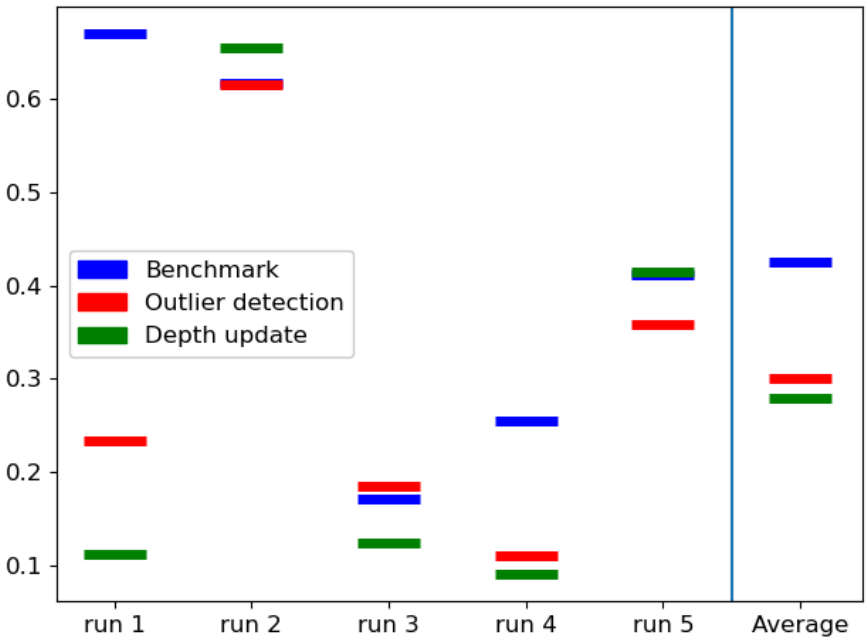
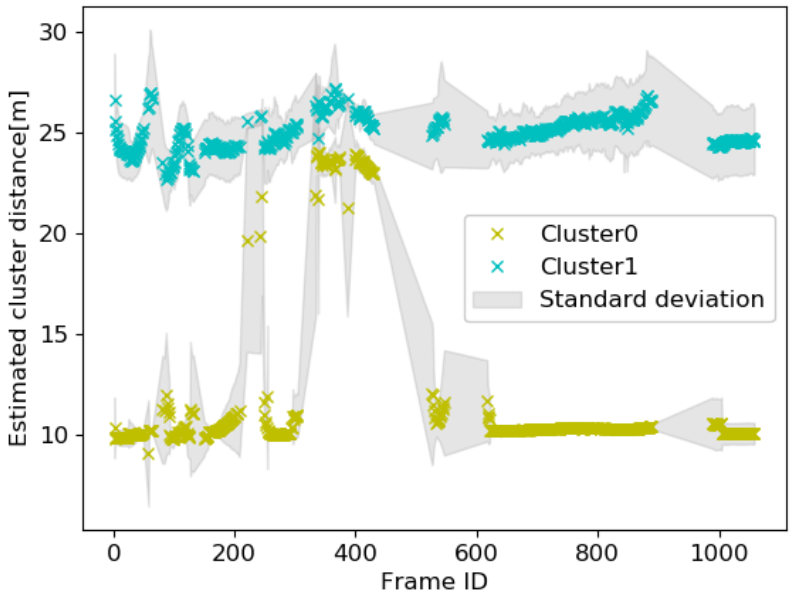


Figure 32: Comparing accuracy's for the standard ROVIO algorithm with my two extensions on a dataset containing two planes at different distances. Each column shows how a single run was evaluated by ROVIO with and without the implemented extensions. All runs was collected with the same baseline of 25cm. The average errors over all five runs are summarized in the rightmost column.

(a) Image frame 351

(b) Image frame 823



(c) Estimation of cluster centers

Figure 33: The images in (a) and (b) are taken from the run that produced the graph in (c). This is *run 2*, as defined in fig. 32, where we can see that the depth update fails and performs slightly worse than the classical approach. The graph in (c) illustrates the estimated cluster centers throughout the trajectory, together with their standard deviations. When no cross is marked, it means that the clustering algorithm failed, due to high variance in the cluster or too few features.

7 Conclusion

The first thing we observed was the importance of a good initial depth estimate when it comes to the system's accuracy. Underestimating the depth gives bad stereo matching and results in an erroneous depth triangulation. Since this results from the local search that ROVIO performs, other methods may not have the same issue. It also seems that overestimating the initial depth is better than underestimating it in case of doubt. If the system cannot extract good depths from triangulation, a possible solution may be to perform an initialization movement before landing. This allows the filter to update and converge the feature depths.

There is no clear sign from the results presented that the baselines significantly impact the system's accuracy. Weak tendencies to better performance for high baselines are present in the trajectory from 25 meters, but certainly not enough to conclude anything. From the same experiment, it was clear that there was a big difference in the observed scale between the baselines, but this did not seem to correlate with the baseline distance. A possible reason for this is calibration errors caused by the frequent remounting of the cameras.

After comparing the performance of monocular systems to stereo systems with and without cross camera measurements, more evidence of poor calibration was presented. The observed trajectory scales were much more consistent over the baselines in the monocular case, indicating that something was off with the cross-camera projections. Despite this, the cross camera stereo system seemed to outperform the monocular and non-cross camera systems in the case where the initial depth was appropriately tuned. However, when wrong initial depth estimates were used, the stereo system gave significant errors, even in the shortest range scenario. Due to the mounting errors, it is difficult to say whether this result has much value. Therefore, the experiment should be re-tried with a more rigid mount, which should not be remounted between calibration and data collection.

The presented ROVIO extension showed improved accuracy for some of the collected data. The fact that the extension worked poorly from a distance of 10 meters and better from further ranges is supported by the underlying assumption of the extension. Since wall indents, windows, and other textures are relatively big from short ranges, the flat ground assumption is expected to break down at these distances. From the results with two planes, it seems that the algorithm is able to separate different clusters from each other, and the overall errors was in fact reduced when applying the extension. Although more data should be collected to say anything sure about the accuracy improvement, the preliminary results are promising.

8 Further work

The effect of baselines on the accuracy of long-range initialization of features should be further explored by extending the range of baselines. The system had a lower limit of 12.5cm and an upper limit of 25cm. Expanding these limits in both directions could make it easier to observe its effect on disparity and accuracy. On the same topic, changing the cameras' resolutions have not been investigated, although this, in theory, should have a similar effect. This can be done in two ways: adjusting the camera settings to sample images at different resolutions or playing with the pyramid levels of the multilevel patches in ROVIO. Neglecting the lowest pyramid level in a patch should have a similar effect as lowering the camera resolution by half on both axes. In addition, even higher ranges or altitudes can be tested to further analyze the interaction between baseline, resolution, feature depth, and disparity.

Another topic that has not been examined in the thesis is the robustness to changing and varying light conditions. While investigating the different attributes of VTOL and VIO, data has only been collected in good conditions to isolate the effects of the different experiments. If VIO is to be implemented as a solution for VTOL in a commercial system, care has to be taken concerning how it reacts to smog, glare, and other weather phenomena that might affect the visual quality of the images.

Since the clustering extension needs to know the number of planes in its view, it is not robust to emergency landings in areas where the terrain is unknown. It could be interesting to investigate the possibility of automatically detecting the number of planes. This could be extended to the problem of turning on and off the cluster filter based on the scene. Furthermore, the clustering algorithm should be tested with more data for multi-plane scenarios. The five runs used in this thesis may indicate good performance, but this is not enough to conclude on anything. Several diversified scenarios should be tested and evaluated to give a broader perspective.

As a more general note, two things need to be considered for all experiments performed in the thesis. First of all, the constant issues related to inaccuracies in extrinsic calibration need to be resolved. In further work, it should be noted that calibrating the system only to remount the cameras frequently is not a viable solution. Even the slightest change in camera orientation can throw off the cross-camera measurements. A more systematic approach where baseline configurations are calibrated and not changed until after the data is collected might be necessary. Secondly, all results are based on horizontal movements against walls to simulate a VTOL. While similar in the sense that the visual information is the same, the fact that gravity points in another direction can impact the results. For this reason, the experiments should be replicated in an actual VTOL scenario to make them more trustworthy.

Bibliography

[1] N. Alvertos. 'Resolution limitations and error analysis for stereo camera models'. In: *Conference Proceedings '88., IEEE Southeastcon*. 1988, pp. 220–224. DOI: 10.1109/SECON.1988.194847.

[2] Oualid Araar, Nabil Aouf and Ivan Vitanov. 'Vision Based Autonomous Landing of Multirotor UAV on Moving Platform'. eng. In: *Journal of intelligent & robotic systems* 85.2 (2016), pp. 369–384. ISSN: 0921-0296.

[3] Herbert Bay, Tinne Tuytelaars and Luc Van Gool. 'SURF: Speeded Up Robust Features'. In: *Computer Vision – ECCV 2006*. Ed. by Aleš Leonardis, Horst Bischof and Axel Pinz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 404–417. ISBN: 978-3-540-33833-8.

[4] Michael Bloesch et al. 'Iterated extended Kalman filter based visual-inertial odometry using direct photometric feedback'. In: *The International Journal of Robotics Research* 36.10 (2017), pp. 1053–1072. DOI: 10.1177/0278364917728574.

[5] Sunglok Choi, Jaehyun Park and Wonpil Yu. 'Resolving scale ambiguity for monocular visual odometry'. In: *2013 10th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI)*. 2013, pp. 604–608. DOI: 10.1109/URAI.2013.6677403.

[6] Jeffrey Delmerico and Davide Scaramuzza. 'A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots'. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, pp. 2502–2509. DOI: 10.1109/ICRA.2018.8460664.

[7] M. Fischler and R. Bolles. 'Random Sample Consensus: A Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography'. In: *Communications of the ACM* 24.6 (1981), pp. 381–395. URL: /brokenurl#%20http://publication.wilsonwong.me/load.php?id=233282275.

[8] Paul Furgale, Joern Rehder and Roland Siegwart. 'Unified temporal and spatial calibration for multi-sensor systems'. In: *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2013, pp. 1280–1286. DOI: 10.1109/IROS.2013.6696514.

[9] Paul Furgale et al. *Kalibr*. https://github.com/ethz-asl/kalibr. 2021.

[10] David Gallup et al. 'Variable baseline/resolution stereo'. In: *2008 IEEE Conference on Computer Vision and Pattern Recognition* (2008), pp. 1–8.

[11] Martin Günther et al. *Allan Variance ROS*. https://github.com/ori-drs/allan_variance_ros/. 2021.

[12] Christopher G. Harris and M. J. Stephens. 'A Combined Corner and Edge Detector'. In: *Alvey Vision Conference*. 1988.

[13] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. Cambridge University Press, 2004. DOI: 10.1017/CBO9780511811685.

[14] Shehryar Khattak, Christos Papachristos and Kostas Alexis. 'Keyframe-based thermal–inertial odometry'. In: *Journal of Field Robotics* 37.4 (2020), pp. 552–579. DOI: https://doi.org/10.1002/rob.21932. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.21932. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.21932.

[15] Shehryar Khattak, Christos Papachristos and Kostas Alexis. 'Keyframe-based thermal–inertial odometry'. In: *Journal of Field Robotics* 37 (Dec. 2019). DOI: 10.1002/rob.21932.

[16] Sven Lange, Niko Sunderhauf and Peter Protzel. 'A vision based onboard approach for landing and position control of an autonomous multirotor UAV in GPS-denied environments'. In: *2009 International Conference on Advanced Robotics*. 2009, pp. 1–6.

[17] Stefan Leutenegger et al. 'Keyframe-Based Visual-Inertial Odometry Using Nonlinear Optimization'. In: *The International Journal of Robotics Research* 34 (Feb. 2014). DOI: 10.1177/0278364914554813.

[18] D.G. Lowe. 'Object recognition from local scale-invariant features'. In: *Proceedings of the Seventh IEEE International Conference on Computer Vision*. Vol. 2. 1999, 1150–1157 vol.2. DOI: 10.1109/ICCV.1999.790410.

[19] Bruce D. Lucas and Takeo Kanade. 'An Iterative Image Registration Technique with an Application to Stereo Vision'. In: *Proceedings of the 7th International Joint Conference on Artificial Intelligence - Volume 2*. IJCAI'81. Vancouver, BC, Canada: Morgan Kaufmann Publishers Inc., 1981, pp. 674–679.

[20] J. MacQueen. 'Some methods for classification and analysis of multivariate observations'. In: 1967.

[21] Mark Maimone, Yang Cheng and Larry Matthies. 'Two years of Visual Odometry on the Mars Exploration Rovers'. In: *Journal of Field Robotics* 24.3 (2007), pp. 169–186. DOI: https://doi.org/10.1002/rob.20184. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.20184. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/rob.20184.

[22] Agostino Martinelli. 'Vision and IMU Data Fusion: Closed-Form Solutions for Attitude, Speed, Absolute Scale and Bias Determination'. In: *IEEE Transactions on Robotics* (July 2011), Volume 28 (2012), Issue 1 (February), pp 44–60. URL: https://hal.archives-ouvertes.fr/hal-00743262.

[23] Frank Mascarich et al. 'A multi-modal mapping unit for autonomous exploration and mapping of underground tunnels'. In: *2018 IEEE Aerospace Conference*. 2018, pp. 1–7. DOI: 10.1109/AERO.2018.8396595.

[24] Janosch Nikolic et al. 'A synchronized visual-inertial sensor system with FPGA pre-processing for accurate real-time SLAM'. In: *2014 IEEE International Conference on Robotics and Automation (ICRA)*. 2014, pp. 431–437. DOI: 10.1109/ICRA.2014.6906892.

[25] Mark Nixon. *Feature Extraction and Image Processing for Computer Vision*. Vol. 3rd ed. Academic Press, 2012, pp. 180–199. ISBN: 9780123965493. URL: https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=477505&site=ehost-live.

[26] NTNU-ARL. *blackfly_nodelet*. https://github.com/ntnu-arl/blackfly_nodelet. 2021.

[27] Tong Qin, Peiliang Li and Shaojie Shen. 'VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator'. In: *IEEE Transactions on Robotics* 34.4 (2018), pp. 1004–1020. DOI: 10.1109/TRO.2018.2853729.

[28] Tong Qin and Shaojie Shen. 'Online Temporal Calibration for Monocular Visual-Inertial Systems'. In: *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE. 2018, pp. 3662–3669.

[29] *ROS Driver for OpenIMU*. https://github.com/ROS-Aceinna/ros_openimu. 2021.

[30] Edward Rosten and Tom Drummond. 'Fusing points and lines for high performance tracking.' In: *IEEE International Conference on Computer Vision*. Vol. 2. Oct. 2005, pp. 1508–1511. DOI: 10.1109/ICCV.2005.104. URL: http://www.edwardrosten.com/work/rosten_2005_tracking.pdf.

[31] Edward Rosten and Tom Drummond. 'Machine learning for high-speed corner detection'. In: *European Conference on Computer Vision*. Vol. 1. May 2006, pp. 430–443. DOI: 10.1007/11744023_34. URL: http://www.edwardrosten.com/work/rosten_2006_machine.pdf.

[32] Davide Scaramuzza and Friedrich Fraundorfer. 'Visual Odometry [Tutorial]'. In: *IEEE Robot. Automat. Mag.* 18 (Dec. 2011), pp. 80–92. DOI: 10.1109/MRA.2011.943233.

[33] Davide Scaramuzza and Zichao Zhang. 'Visual-Inertial Odometry of Aerial Robots'. In: (June 2019).

[34] Naser El-Sheimy, Haiying Hou and Xiaoji Niu. 'Analysis and Modeling of Inertial Sensors Using Allan Variance'. In: *Instrumentation and Measurement, IEEE Transactions on* 57 (Feb. 2008), pp. 140–149. DOI: 10.1109/TIM.2007.908635.

[35] Jianbo Shi and Tomasi. 'Good features to track'. In: *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*. 1994, pp. 593–600. DOI: 10.1109/CVPR.1994.323794.

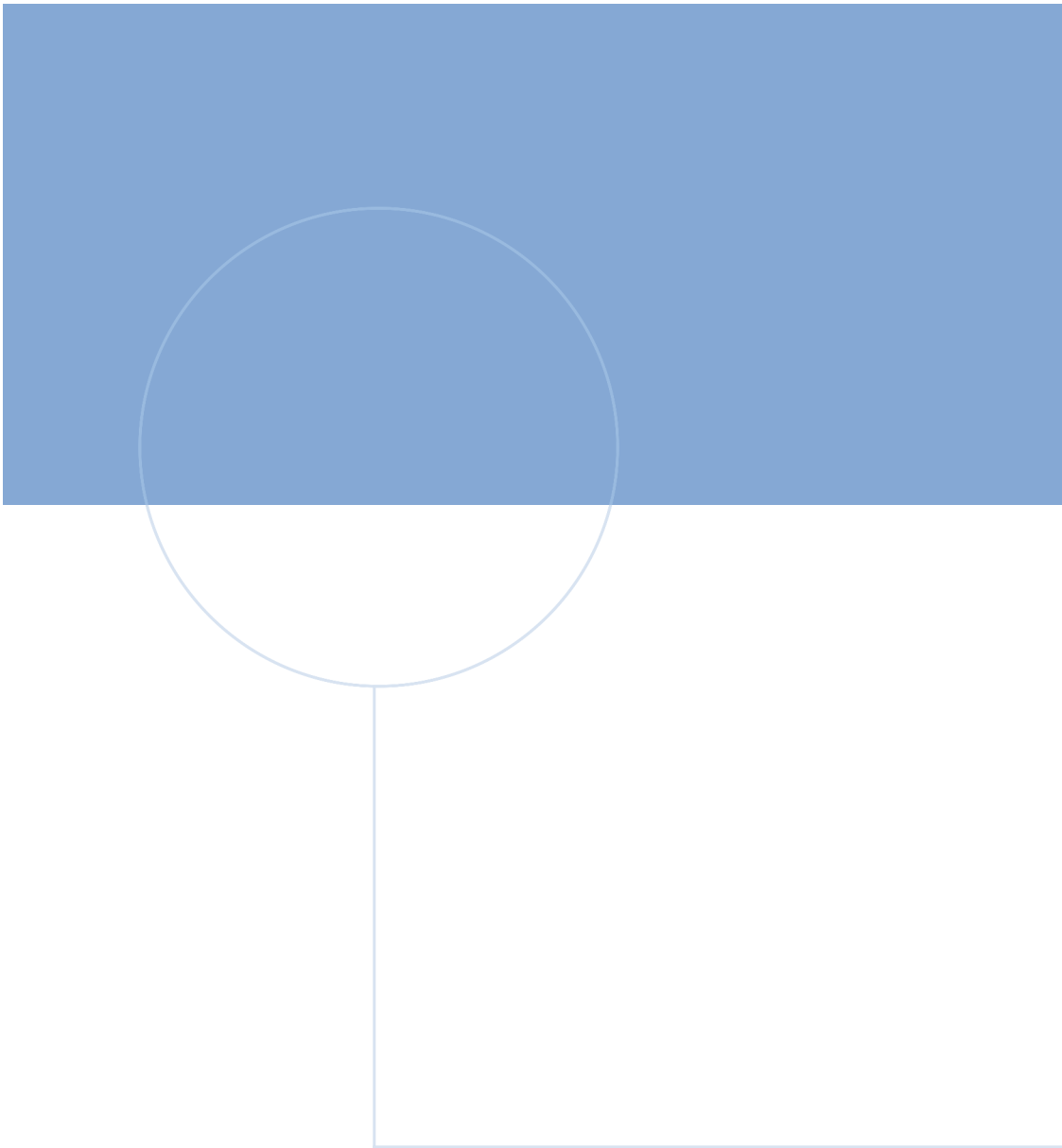
[36] Joan Solà, Jeremie Deray and Dinesh Atchuthan. *A micro Lie theory for state estimation in robotics*. 2021. arXiv: 1812.01537 [cs.RO].

[37] Zhongwei Tang et al. 'A Precision Analysis of Camera Distortion Models'. In: *IEEE Transactions on Image Processing* 26.6 (2017), pp. 2694–2704. DOI: 10.1109/TIP.2017.2686001.

[38] Michael Warren, Peter Corke and Ben Upcroft. 'Long-range stereo visual odometry for extended altitude flight of unmanned aerial vehicles'. In: *The International Journal of Robotics Research* 35.4 (2016), pp. 381–403. DOI: 10.1177/0278364915581194. eprint: https://doi.org/10.1177/0278364915581194. URL: https://doi.org/10.1177/0278364915581194.

[39] Zhizun Xu et al. 'An Integrated Visual Odometry System for Underwater Vehicles'. In: *IEEE Journal of Oceanic Engineering* 46.3 (2021), pp. 848–863. DOI: 10.1109/JOE.2020.3036710.

[40] Xupei Zhang et al. 'VIAE-Net: An End-to-End Altitude Estimation through Monocular Vision and Inertial Feature Fusion Neural Networks for UAV Autonomous Landing'. In: *Sensors* 21 (Sept. 2021), p. 6302. DOI: 10.3390/s21186302.



NTNU

Norwegian University of
Science and Technology