Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Alexander Johansen

Fast Optical Flow Estimation for End-Effector Stabilization

Completed at the Autnomous Systems Lab at ETH Zürich.

Supervised by:
Karen Bodie
Michael Pantic
Roland Siegwart

Master's thesis in Cybernetics and Robotics
Supervisor: Tor Arne Johansen
August 2020

NTNU
Kunnskap for en bedre verden

Alexander Johansen

Fast Optical Flow Estimation for End-Effector Stabilization

Completed at the Autnomous Systems Lab at ETH Zürich.

Supervised by:
Karen Bodie
Michael Pantic
Roland Siegwart

Master's thesis in Cybernetics and Robotics
Supervisor: Tor Arne Johansen
August 2020

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Motivation

Unmanned aerial vehicles (UAV) have since their inception been used for aerial photography introducing a new method to inspect industrial installations in hard-to-reach areas. As technology has progressed, it has become possible to achieve more complicated maneuvers and the quality of service delivered using UAVs has drastically increased. The natural next step in this process is to perform physical inspection as done in Bodie et al. (2019). This inspection uses a rigid arm mounted on an omnidirectional miniaturized aerial vehicle (OMAV) in order to perform contact based inspection of concrete infrastructure. This method does not provide much flexibility as the entire system must move in order to move the tool used to inspect the surface. The next iteration of this platform introduces an independently controlled end-effector where the dynamics are significantly faster then the OMAV allowing for better stabilization and more precise interaction with the target. The addition of a precise end-effector pushes the capabilities of the system beyond only inspection, and enables simple interaction such as manipulation and repair of the target.

In order to perform these types of operations, the position of the end-effector in reference to the surface it is interacting with, must be know to a very high degree of accuracy. For the end-effector to be maneuverable it is critical that the attached sensors are lightweight and that the state estimation is at a similar frequency as that of the control loop.

This thesis aims to go beyond the current state of the art by building a highly integrated system that maximizes framerate and accuracy on sensor, firmware and front-end algorithm levels.

1.2 Problem description

The OMAV used in this was initally introduced as the VoliroX in Kamel et al. (2018) and was further improved in Bodie et al. (2018) and Bodie et al. (2019). The current iteration

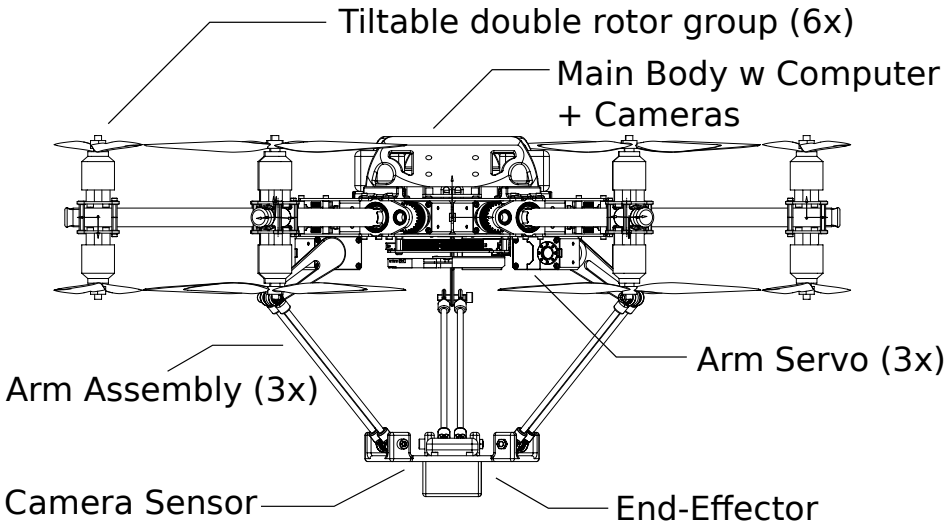


Figure 1.1: An annotated sketch of the final system

of the platform is called Ouzel and comprises 6 independently tiltable arms with 2 motors per arm capable of delivering 12.7N of force per motor. The over-actuation of the system enables flight in any orientation.

The current odometry algorithm run on the platform is called Rovio, introduced in Bloesch et al. (2015). While this odometry method provides global stability in 6-dof it does not provide high enough estimation accuracy or rates to stabilize the end-effector to the desired degree of precision.

The UAV has the capability of flying in any orientation and has a delta-arm that can move translationally independent from the main body, which should be sufficient in order to perform the intended operations. This means that the end-effector will be rigidly mounted in regard to the rotation axes and free in the translational axes. A rendering of the final system is shown in fig. 1.1

1.3 Project scope

While the physical specifications of the arm is of course critical for the performance of the final system, another key aspect is the measurement and estimation of the end-effector state. This project aims to research possible solutions to the state estimation problem, as well as developing a sensor system that is capable of estimating the end-effector state with mm precision.

The physical arm is being developed concurrently with this project and will therefore not be available to perform tests on. However the development and testing of the state estimation can be performed independently from the arm assembly.

The approach used in this project was using a camera and range sensor in order to

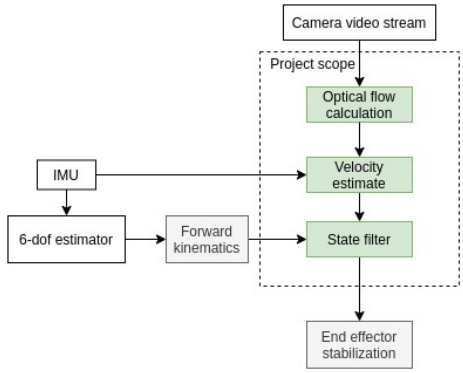


Figure 1.2: An overview of the proposed solution, the green blocks are the main contributions of this project. The white blocks are external inputs and the grey blocks are worked on concurrently with this project.

calculate the velocity of the end-effector. This can be done at very high frequencies and fusing the measurement with odometry from the global estimator using forwards kinematics provides a full state estimate in the desired reference frame at high frequencies. A diagram of a high level system overview is shown in fig. 1.2

Chapter 2

Literature review and existing solutions

Solving the problem of egocentric positioning in 6-dof is a complex and computationally intensive problem. Current state of the art visual-inertial odometry(VIO) algorithms have a maximum frequency av approximately 40-60Hz and are the results of decades of research. Though odometry methods provide a global reference position and work well at the large scale navigation, they are not developed with the intention of stabilizing something locally at very high speeds and accuracy.

This project seeks to implement a visual-inertial state estimation method that has a high enough speed and precision to control an end-effector mounted on the UAV with mm precision. The proposed solution does not attempt to implement another 6-dof estimator and relies on external odometery to relate the current position to the world frame.

The following chapter will explore the existing solutions as well as various methods that are currently used for aerial stabilization and navigation.

2.1 Rovio

Rovio(RObust Visual Inertial Odometry) is the egocentric state estimation algorithm currently deployed on the platform developed in Bloesch et al. (2015). Rovio is a visual-inertial state estimator based on an iterated EKF. Rovio uses FAST corner features Rosten and Drummond (2005) parameterized in a robocentric view with a bearing vector and distance associated with every feature. Multi level patches are extracted around each of these features and warped according to the IMU-measured motion in order to minimize the patch error between two frames. Extracting patches around each feature gives much richer information about the environment and increases tracking robustness, and enables tracking of linear features in order to gain information along the perpendicular axis.

The photometric error between a given patch and the next image is calculated by estimating the position and warping of the patch in the image and calculating it, using equa-

tion 35 in Bloesch et al. (2017). This takes inter-frame illumination changes and the higher level patch features increase robustness with respect to image blur or alignment issues into account, while the lower level features ensure precise feature tracking. The FAST corner detector provides a large amount of candidate features to be tracked and in order to reduce computational burden a subset of these are chosen. The score of each feature is calculated and coupled with a bucketing technique and the best features with a good distribution within the camera frame are chosen.

In visual-inertial sensor setups IMU measurements are typically acquired between 5-50 faster than images. Forster et al. (2017) solve this computational burden with IMU preintegration. The proposed solution in Rovio is a simpler preintegration method in which the Jacobian is evaluated based on the mean of the IMU measurements. This is in order to only calculate the a-priori covariance once. It is stated that this method does not suffer any performance loss compared to the regular method, and this might be the case for large scale accuracy over time, but it needs to be examined more in detail for pinpoint accuracy in low speed environments.

Various other Visual-Inertial-Odometry (VIO) algorithms exist and combining processing methods from these might improve estimation results. Some open source methods that achieve similar precision levels are VINS-Mono by Qin et al. (2018), Okvis Leutenegger et al. (2013) and methods suggested by Forster et al. (2017).

2.2 Optical flow based estimation and control

2.2.1 Definition

Optical flow is the apparent motion of objects, surfaces and edges caused by the 2d projection of a relative 3D motion between an observer and scene. The optical flow is formulated as a 2D vector field of a size corresponding to the image input with vectors $\in \mathbb{R}^2$ where each vector describes the direction and magnitude that each point has moved from one frame to another.

Optical flow is created by the translation and rotation of a 3D point in the world **P**(X,Y,Z) in the image frame $\boldsymbol{x}$(x,y), the full definition of variables are illustrated in fig. 2.1 from Raudies (2013). The position of the point in the image frame can be expressed by P and the focal length f as

$$\begin{bmatrix} x \\ y \end{bmatrix} = \frac{f}{Z} \begin{bmatrix} X \\ Y \end{bmatrix} \tag{2.1}$$

Note that optical flow occurs both if the observation frame or the point moves. The model equation for instantaneous or differential motion disregarding the movement of the scene is:

$$\begin{bmatrix} X_{OF} \\ Y_{OF} \end{bmatrix} = \underbrace{\frac{1}{Z} \begin{bmatrix} -f & 0 & x \\ 0 & .-f & y \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}}_{translational\ flow} + \underbrace{\frac{1}{f} \begin{bmatrix} x \cdot y & -(f^2+x^2) & f \cdot y \\ (f^2+y^2) & -x \cdot y & -f \cdot x \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}}_{rotational\ flow} \tag{2.2}$$

Equation (2.2) shows some important characteristics, translational flow superimposes linearly with rotational flow. The depth Z of the 3D point only influences the translational component while the rotational component is only dependent on the focal length. Figure 2.2 illustrates optical flow generated from various types of motion.

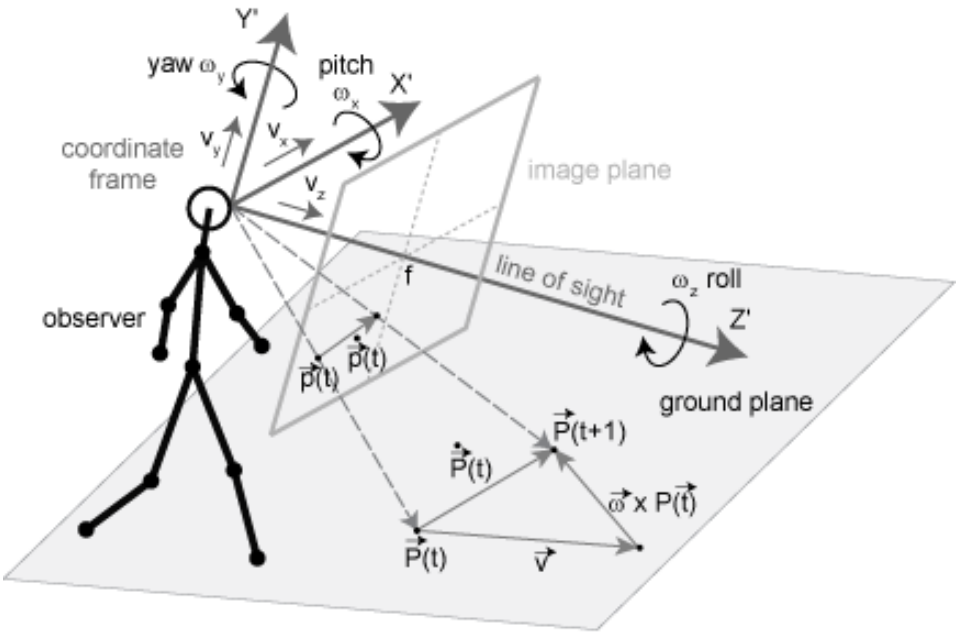


Figure 2.1: Illustration of how translational and rotational velocity of an observer generate appearant motion in the scene, from Raudies (2013)

2.2.2 Methods of computation

Determining optical flow can be classified into the following categories

- Patch based matching
- Feature matching
- Phase correlation
- Deep neural networks

Feature based techniques provide very good accuracy even when there are large displacements. However, they are computationally expensive, less accurate in case of deformation and sub pixel displacements are not detectable. This is essentially what most other state of the art VIO algorithms try to optimize by reducing the amount of features to track and only tracking the best features in an image rather than generating the full flow field.

Patch-based matching is very similar to feature based methods, only instead of selecting features to track between frames, a patch-based method solve an optimization problem

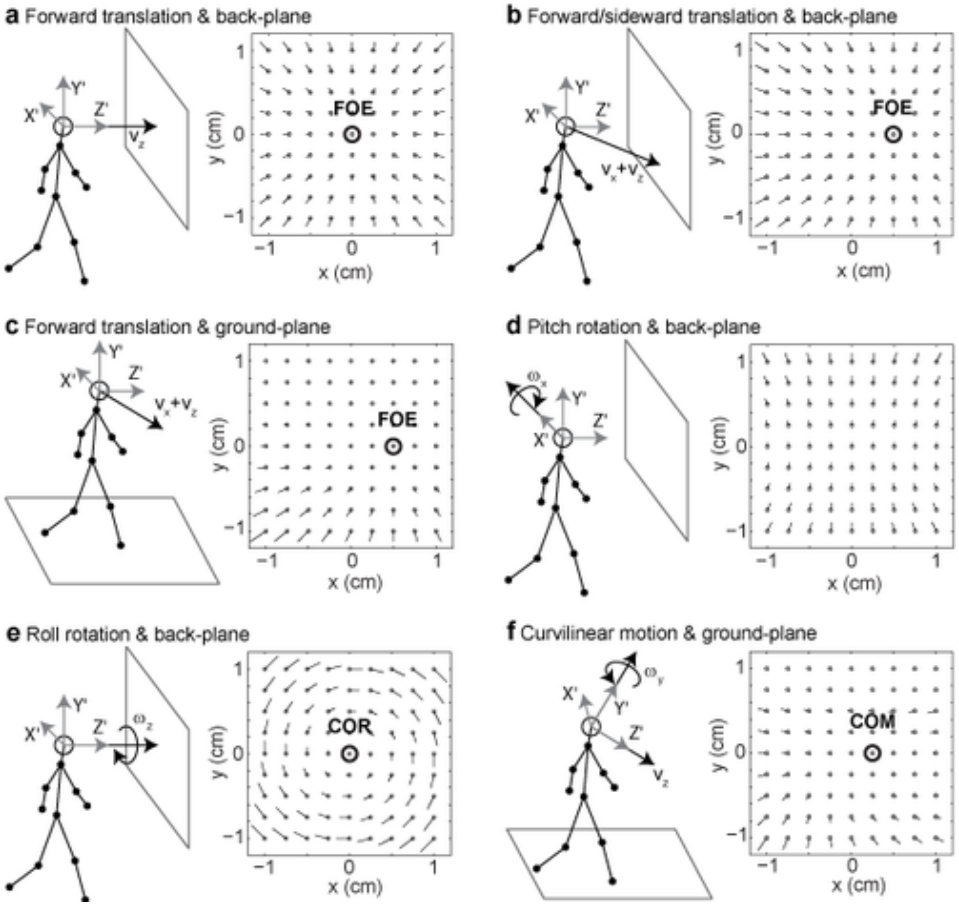


Figure 2.2: Illustration of how various movements impact the observed optical flow from Raudies (2013)

to determine the displacement of an arbitrary region in the original image. This type of method is also referred to as a differential method. The patch based method is generally faster then feature tracking as no feature selection is required, however feature based methods are more robust to warping and regions that have featureless interiors. Patch- or gradient-based methods suffer from sensitivity issues due to changing lighting and linearization. They are generally quite precise when the inter-frame motion is small and are fast to compute.

These methods might be usable in order to provide a high frequency pose estimation to augment the existing state estimation when the MAV is trying to maintain a stable position in space.

Recently there has been much development in developing neural networks for optical flow estimation with some of them relatively lightweight and fast such as LiteFlowNet Hui et al. (2018) inspired by FlowNet2 Ilg et al. (2016). FlowNet2 is capable of frame-

rates between 8-140fps, however most CNN methods are run on high-end GPUs and are therfore not suited for on board aerial processing. Most CNNs are based on the block and feature matching methods and are often trained on publicly available data-sets such as SINTEL Butler et al. (2012), which is a good resource for comparing various methods. Many of these algorithms are designed to perform extremely well on these datasets and can be over-engineered for the task and accuracy that is required in this project providing little customization and slow speeds.

If it proves hard to decide what method to use for this application, it is possible to use one of the more advanced methods as a ground truth flow and quantitatively compare the candidates to each other.

A more exhaustive survey of the existing methods of computing optical flow is done in Fortun et al. (2015)

2.2.3 Issues with optical flow

Many flow methods are based on the assumption that the lighting is consistent from one frame to another, while some robustness might be gained by normalizing frames or feature areas, finding proper correspondents between frames will be harder with lighting variations. Challenging lighting conditions might occur in artificially lit scenes where the flicker from one frame to the next or in situations where the observer is between the light source in the scene causing moving shadows to appear in the image.

Another known problem that is shared between most vision based algorithms is that texture is needed in order to calculate displacement from one frame to another. Dense optical flow methods are particularly vulnerable as they do not use information gained outside of these texture-less areas to estimate the internal flow. If for example a texture-less square was moving across a textured scene a feature based method that uses all information available in the image could interpolate the speed of the internal points of the square, while a patch based method would provide ambiguous flow estimates in the texture-less regions.

A more complicated version of this problem is known as the aperture problem. This occurs when the ends of an edge are not visible in the image, in this scenario it becomes impossible to estimate the motion along this edge. This problem is demonstrated in fig. 2.3.

2.2.4 Navigation using OF

Zufferey and Floreano (2005) explore the use of optical flow to navigate with ultra lightweight UAV. They demonstrate usage of optical flow for obstacle avoidance and altitude hold with very limited weight and computation power.

Ruffier and Franceschini (2005) go more in depth on how optical flow can be used as a control strategy.

There are many more optical flow experiments done using aircraft that are attempting to maintain altitude or follow a physical path, commonly for these is the goal of maintaining a constant optical flow or predetermined flow sequence. In the case of a stable hover the goal is to minimize all flow.

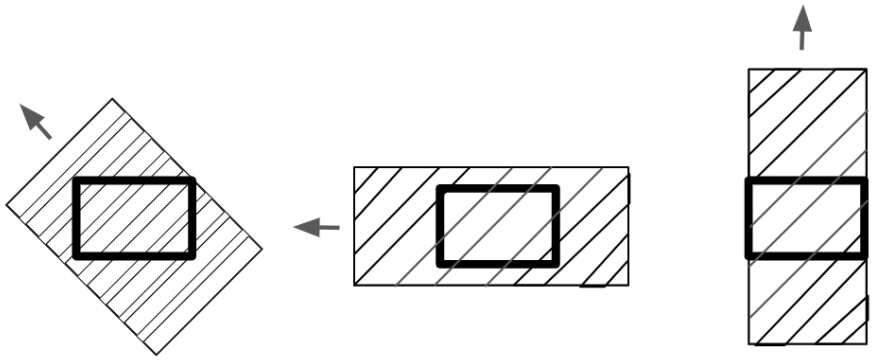


Figure 2.3: Illustration of the aperture problem. If the observer can only detect movement inside the bold rectangle the motion of the lines will look identical.

Chapter 3

Fundamentals

3.1 Coordinate Frames

The project mainly uses three different coordinate frames, frames with the same origin and orientation are written together:

- Body/IMU
- World/Vicon
- Camera/End-Effector

During testing the external video capture system measures the position and orientation of the body frame using ENU coordinates, while the internal state estimation is expressed with NED coordinates.

In this report the transformation from body to camera frame is a rigid transformation. However when the arm is mounted on the system the transformation from body to camera frame can be estimated using the forward kinematics.

3.2 Description of Ouzel

Ouzel the OMAV used that the delta arm will be mounted on. It is a hexacopter with dual KDE 885 motors on each arm, where each arm can rotate separately in order to preform complex aerial maneuvers. The current system is controlled with a low level embedded flight controller, a Pixhawk 4. The controller is custom written software that is capable of determining the motor rates and arm angles in order to stabilize the system.

As a higher level controller a Intel NUC i7 running linux is used to interface with the flight controller and preform higher level tasks. ROS is run on the NUC to preform high level estimation of the system state and communicate with the ground station.

Ouzel was most recently used to inspect concrete surfaces with a fixed arm mounted on the main body, this configuration is shown in fig. 3.1.

Figure 3.1: Ouzel the hardware necessary to preform contact inspection.

3.3 Visual inertial odometry

Visual inertial odometry (VIO) is the process of estimating the pose and velocity of a system using only the input from on board cameras and IMUs. In GPS denied environments VIO(along with lidar based navigation) is the only viable option for state estimation.

With the development of autonomous aerial platforms accurate VIO is an essential element in the performance of these systems. Combining accurate VIO algorithms with simultaneous localization and mapping (SLAM) enables long missions to be carried out with minimal error.

While this project does not aim at developing an odometry method, it is important to understand the existing odometry method on the platform, why it is not the correct solution to the problem we are solving and what other methods exits and their strengths.

3.4 Optical flow

This project aims to select the most suited optical flow method and will not perform extensive tests of the state of the art of optical flow algorithms. In this section the underlying theory of optical flow will be introduced as well as the workings of some select algorithms.

In order to find and select a well-suited algorithm for this project three main points were considered:

- Run time for the full algorithm including image prepossessing
- Processing power needed to run the algorithm
- Accuracy

The goal of the project is to provide a high frequency velocity estimation, in order to do this the underlying algorithm must also operate at a high speed. Many of the cutting edge algorithms are evaluated on the end point error(EPE) with little or no regard for run time with top performers such as RAFT Teed and Deng (2020) and STaRFlow Godet et al.

(2020) running at 5-10Hz on GPUs and high preforming CPU algorithms such as SfM-PMMaurer et al. (2018) taking 32s per frame.

As processing-power onboard the drone is limited it is essential that the algorithm performs well on a CPU and not run on dedicated hardware.

Finally it must be accurate in the environments where expected usage will occur. The expected use case of the optical flow is to detect ego motion while observing planar surfaces. In this scenario the relative accuracy of the algorithms might vary significantly compared to what the reported EPE indicates. In practice most modern algorithms should provide sufficiently good results in order to determine the speed of the observer.

3.4.1 Lucas–Kanades method

Many of the modern algorithms are in some way based on the Lucas-Kanade algorithm introduced in Lucas and Kanade (1981). The method describes the general method of computing optical flow and it will briefly be summarised here.

Optical flow can be simplified as solving a registration problem characterized by having two functions F(**x**) and G(**x**) which give the pixel value of the location **x** in two images. The goal of the optical flow estimation is to find a vector ***h*** that minimises the difference between F(**x** + **h**) and G(**x**) in some region (*R*) around **x**. A metric typically used to calculate the disparity is the L_2 norm as a cheap and accurate measurement of disparity.

$$L_2 = \sum_{x \in R} \left[F(\mathbf{x} + \mathbf{h}) - G(\mathbf{x}) \right]^2 \tag{3.1}$$

By minimizing L_2 **h** is the resulting flow of the pixel **x**. This is done by Gauss-Newton gradient decent of a non-linear optimization problem.

Applying this notation to a more practical example; consider two consecutive images in a image stream I_t, I_{t+1} one can extract a region centered around a pixel **x** from I_t and call this region T(**x**). The optimization problem using the L_2 norm eq. (3.1) then becomes

$$\mathbf{h} = argmin_{\mathbf{h}} \sum_{x} \left[I_{t+1}(\mathbf{x} + \mathbf{h}) - T(\mathbf{x}) \right]^2 \tag{3.2}$$

Minimizing this is a non-linear optimization task as pixel intensity is not correlated with position. The Lucas-Kanade method solves this by assuming **h** is known, and iterably solves for increments to the parameter $\Delta\mathbf{h}$

$$\Delta\mathbf{h} = argmin_{\Delta\mathbf{h}} \sum_{x} \left[I_{t+1}(\mathbf{x} + \mathbf{h} + \Delta\mathbf{h}) - T(\mathbf{x}) \right]^2 \tag{3.3}$$

Assuming that the image is linear in an area around **x**, eq. (3.3) is linearized by preforming a first order Taylor expansion

$$\Delta\mathbf{h} = argmin_{\Delta\mathbf{h}} \sum_{x} \left[I_{t+1}(\mathbf{x} + \mathbf{h}) + \nabla I_{t+1} \frac{\partial(\mathbf{h} + \Delta\mathbf{h})}{\partial\mathbf{h}} \Delta\mathbf{h} - T(\mathbf{x}) \right]^2 \tag{3.4}$$

Where the gradient ΔI_{t+1} is evaluated at $\mathbf{x} + \mathbf{h} + \Delta\mathbf{h}$. The steepest decent direction is calculated by taking the partial derivative

$$\sum_x \left[\nabla I_{t+1} \frac{\partial(\mathbf{h}+\Delta\mathbf{h})}{\partial \mathbf{h}}\right]^T \left[I_{t+1}(\mathbf{x}+\mathbf{h}) + \nabla I_{t+1} \frac{\partial(\mathbf{h}+\Delta\mathbf{h})}{\partial \mathbf{h}} \Delta\mathbf{h} - T(\mathbf{x})\right]^2 \tag{3.5}$$

From this expression the closed form formula for $\Delta\mathbf{h}$ is is given as:

$$\Delta\mathbf{h} = \mathbf{H}^{-1} \sum_x \left[\nabla I_{t+1} \frac{\partial(\mathbf{h}+\Delta\mathbf{h})}{\partial \mathbf{h}}\right]^T \left[I_{t+1}(\mathbf{x}+\mathbf{h}) + \nabla I_{t+1} \frac{\partial(\mathbf{h}+\Delta\mathbf{h})}{\partial \mathbf{h}} \Delta\mathbf{h} - T(\mathbf{x})\right]^2 \tag{3.6}$$

where ***H*** is the hessian matrix evaluated at **x+h**.
Using this expression **h** is updated until $\Delta\mathbf{h}$ reaches sufficiently small values.

3.4.2 PCA-flow

Principal component analyses flow is an optical flow algorithm introduced by Wulff and Black (2015). It utilizes a sparse to dense approach where *K* feature points between two images are matched. The correspondence of these feature points $\{\mathbf{p}_{k,t}, \mathbf{p}_{k,t+1}\}$ induce a displacement vector $\mathbf{h} = \mathbf{p}_{k,t+1} - \mathbf{p}_{k,t}$. This provides a sparse flow estimate from one frame to the next, this is beneficial as using fewer points to calculate flow results in a cheaper algorithm. In addition feature matching is a robust method of tracking large displacements where the linearization around a pixel is no longer a good approximation.

Feature detection

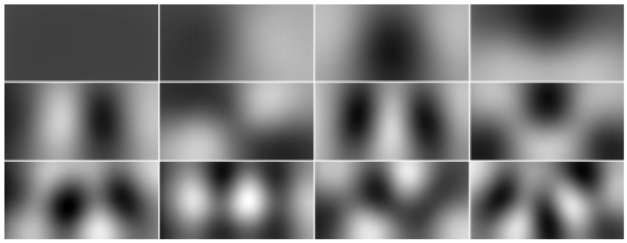
Various methods to track features from frame to frame have been developed; such as SIFT Lindeberg (2012), SURF Bay et al. (2006) and FAST Rosten and Drummond (2005). PCA-flow utilizes a separate feature detection from Geiger et al. (2011) which is designed for visual odometry applications, while this choice is effective, more optimal solutions are evaluated in Otsu et al. (2013). While the results in Otsu et al. (2013) indicats that FAST features might be a better choice for this application note that the current odometry algorithm on the platform uses FAST features, so in order to increase robustness it might be beneficial to utilize a separate algorithm.

Densification

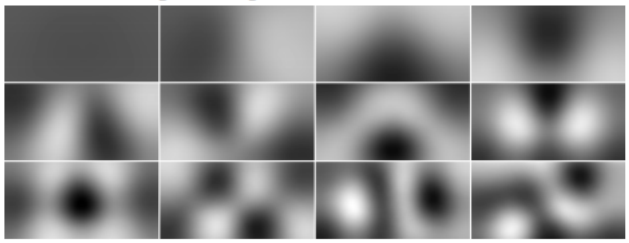
Computing a dense flow field using the sparse is done using a linear combination of a set of basis vectors. In order to compute the basis vectors GPUFlow Werlberger et al. (2009) is used to calculate the flow of four Hollywood movies. 500 principal components of the resulting flow are calculated using the robust PCA method from Hauberg et al. (2014). A selection of the resulting basis is shown in fig. 3.2

3.4.3 Coarse 2 fine optical flow

In Liu (2009) a method based on the Lucas-Kanade algorithm is implemented. However it differs from Lucas-Kanade in that the conjugate gradient method is used to iteratively



(a) Principal components for horizontal motion



(b) Principal components for vertical motion

Figure 3.2: Selected principal components used for flow densification, image from Wulff and Black (2015)

compute the pixel flow values, which should increase the speed of the algorithm. This method utilizes a coarse to fine approach which computes the image flow at coarser resolutions and uses this to initialize flow at higher resolution. This helps avoid poor local minima and improve estimation for large movements.

3.4.4 Dense Inverse search

Fast optical flow using dense inverse search proposed by Kroeger et al. (2016) is an algorithm that achieves average results on both the KITTI Geiger et al. (2013) and SINTEL Butler et al. (2012) data-sets in terms of accuracy. This performance is very impressive considering that it runs orders of magnitude faster than other algorithms with accuracy in this category. According to the original paper it can achieve run speeds of up to 300Hz while maintaining high image quality. However this does not include image preprocessing which, when included, significantly reduces the speed of the algorithm.

DIS is a patch based, multi-layered optical flow algorithm. The underlying patch matching is done using the Lucas-Kanade method. However it uses methods introduced in Baker and Matthews (2004) to optimize the search. The main takeaway is how it utilizes a inverse search function instead of eq. (3.3)

$$\Delta\mathbf{h} = \sum_{\mathbf{x}} \left[T(\mathbf{x} - \Delta\mathbf{h}) - I_{t+1}(\mathbf{x} + \mathbf{h})\right]^2 \tag{3.7}$$

This removes the need to recompute the Hessian and Jacobian of the images every iteration which significantly speeds up the minimization problem.

Explanation of the algorithm

An image pyramide is created by down-scaling the original image by a factor of 2 until the desired height *h* is reached. In order to complete the variational refinement step the gradients of the image at each scale must be calculated. For each scale *s* a dense flow field $\mathbf{U}_s$ is created starting from the coarsest scale. The resolution of $\mathbf{U}_s$ is equal to the resolution of the image at the scale *s*.

Creating grid: A patch size and patch overlap coefficient are selected and a grid is generated uniformly over the image domain and the overlap is floored to an integer pixel value. The number of patches N_s needed to be matched between frames is determined by the density of the patches and image size.

Initializing flow: At the coarsest scale the flow is initialized to 0 and for each consecutive scale *s* the flow is initialized from the previous scale *s-1* as visualized in fig. 3.3.

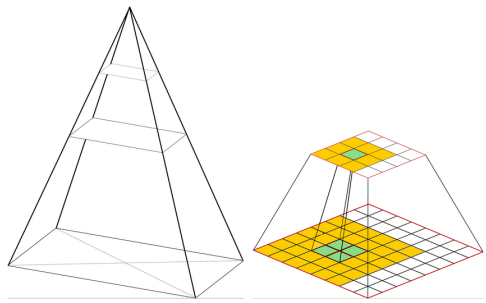


Figure 3.3: Visual representation of image pyramid and initialization from previous layers

Inverse search: The optimal displacement is calculated for each patch in N_s. If the flow of a patch is found to be larger than the patch size it is reset to the initial flow in order to increase robustness.

Flow densification: The central pixel $\mathbf{x}$ in every patch *i* has now been assigned a displacement vector $\mathbf{u}_i$, the surrounding pixels displacement vectors are decided by the weighted average of the surrounding patches:

$$\mathbf{U}_s(\mathbf{x}) = \frac{1}{Z}\sum_{i}^{N_s} \frac{\lambda_{i,\mathbf{x}}}{max(\epsilon, d_i(\mathbf{x})}\mathbf{u}_i \tag{3.8}$$

Where $\lambda_{i,\mathbf{x}}$ is 1 when the pixel $\mathbf{x}$ is in the patch *i* and 0 otherwise, $d_i(\mathbf{x})$ is the intensity difference of the two images being compared $d_i(\mathbf{x}) = I_{t+1}(\mathbf{x}+\mathbf{u}_i) - I_t(\mathbf{x})$, ϵ is the smallest possible intensity difference and Z is the aggregated weight used to normalize the flow.

Variational refinement might be not be necessary for the scope of the project since it is not expected to be large motion boundaries, hence this step is optional. The variational refinement is an iterative process in which the energy in the image is minimized, it is defined as a weighted sum of intensity, gradient and smoothness terms. The magnitude of the intensity and gradient terms relate how similar the variation of the displacement vector is to the intensity and gradient image, while the smoothness term penalizes the total gradient of displacements.

3.5 Ego-motion estimation

Calculating the ego-motion of the observer from optical flow is a non-trivial problem with research being done on many possible options Chhaniyara et al. (2008); Ho et al. (2017); Honegger et al. (2012). This section will briefly review the methods implemented in this project.

3.5.1 4-point algorithm

In Ma et al. (2003) the 4-point algorithm is introduced. This section will give a brief introduction to the algorithm. It is quite technical so for proofs or further explanation please refer to chapter 5 of Ma et al. (2003). Here the *continues homography matrix* **H** is given as a simplification of eq. (2.2) which assumes that points in the scene are located on a common plane such that $\mathbf{n}^T\mathbf{X} = d$ where **n** is the unit normal vector to the plane and **X** is the position of a point in 3D space and *d* is the distance to the plane.

$$\mathbf{H} = [\boldsymbol{\omega}]_\times + \frac{1}{d}\mathbf{v}\mathbf{n}^T \tag{3.9}$$

such that

$$\dot{\mathbf{X}} = \mathbf{H}\mathbf{X} \tag{3.10}$$

here $[\boldsymbol{\omega}]_\times$ is the skew symmetric matrix associated with $\boldsymbol{\omega}$. The *continues homography constraint* is given below

$$[\mathbf{x}]_\times \mathbf{H}\mathbf{x} = [\mathbf{x}]_\times \mathbf{u} \tag{3.11}$$

Which involves a set of *N* image pairs $(\mathbf{x}_i, \mathbf{h}_i)$ where **h** is the displacement from one image to the next. In order to retrieve the elements of **H** it is re-stacked into a vector $\mathbf{H}_s$ such that it can be solved as a linear set of equations.

$$\mathbf{a}_i^T \mathbf{H} = [\mathbf{x}_i]_\times \mathbf{u}_i \tag{3.12}$$

Where $\mathbf{a}_i = \mathbf{x}_i \otimes [\mathbf{x}_i]_\times$ and $\otimes$ is the Kroneker product operator and $i \in N$. By stacking all the equations into one equation the result is

$$\chi \mathbf{H}^s = \mathbf{B} \tag{3.13}$$

Where $\chi = [\mathbf{a}_1, ..., \mathbf{a}_N]^T \in \mathbb{R}^{3n\times 9}$ and $\mathbf{B} = [[\mathbf{x}_1]_\times \mathbf{u}_1, ..., [\mathbf{x}_N]_\times \mathbf{u}_N]^T \mathbb{R}^{3n}$ in order to solve this system uniquely rank(χ) = 8. Since each pair of image points and flow values impose two constraints at least four optical flow pairs that are not colinnear are necessary.

Since χ has a one dimensional null space, the resulting matrix is not a perfect representation of **H** and will therefore be denoted $\mathbf{H}_L$. In order to normalize this matrix the eigenvalues $[\lambda_1, \lambda_2, \lambda_3]$, in ascending value, of $\mathbf{H}_L^T + \mathbf{H}_L$ are computed and the final **H** matrix is given by

$$\mathbf{H} = \mathbf{H}_L - \frac{1}{2}\lambda_2 \mathbf{I} \tag{3.14}$$

Finally in order to extract the velocity and surface information from **H** the 4 possible solutions are given in table 3.1. Where $[\lambda_1, \lambda_2, \lambda_3]$ are the sorted eigenvalues of $\mathbf{H} + H^T$ and $[\mathbf{u}_1, \mathbf{u}_2, \mathbf{u}_3]$ are the corresponding eigenvectors.

Solution 1	$\frac{1}{d}\mathbf{v}_1 = \sqrt{\alpha}\tilde{\mathbf{v}}_1$ $\mathbf{n}_1 = \frac{1}{\sqrt{\alpha}}\tilde{\mathbf{n}}_1$ $\boldsymbol{\omega} = \mathbf{H} - \tilde{\mathbf{v}}_1\tilde{\mathbf{n}}_1^T$	Solution 3	$\mathbf{v}_3 = -\mathbf{v}_1$ $\mathbf{n}_3 = -\mathbf{n}_1$ $\omega_3 = \omega_1$
Solution 2	$\frac{1}{d}\mathbf{v}_2 = \sqrt{\alpha}\tilde{\mathbf{v}}_2$ $\mathbf{n}_2 = \frac{1}{\sqrt{\alpha}}\tilde{\mathbf{n}}_2$ $\boldsymbol{\omega} = \mathbf{H} - \tilde{\mathbf{v}}_2\tilde{\mathbf{n}}_2^T$	Solution 4	$\mathbf{v}_4 = -\mathbf{v}_2$ $\mathbf{n}_4 = -\mathbf{n}_2$ $\omega_4 = \omega_2$

Table 3.1: Solutions for planar homography decomposition

Where

$$\alpha = \frac{1}{2}(\lambda_1 - \lambda_3) \tag{3.15}$$

$$\tilde{\mathbf{v}}_1 = \frac{1}{2}(\sqrt{2\lambda_1}\mathbf{u}_1 + \sqrt{-2\lambda_3}\mathbf{u}_3) \tag{3.16}$$

$$\tilde{\mathbf{v}}_2 = \frac{1}{2}(\sqrt{2\lambda_1}\mathbf{u}_1 - \sqrt{-2\lambda_3}\mathbf{u}_3) \tag{3.17}$$

$$\tilde{\mathbf{n}}_1 = \frac{1}{2}(\sqrt{2\lambda_1}\mathbf{u}_1 - \sqrt{-2\lambda_3}\mathbf{u}_3) \tag{3.18}$$

$$\tilde{\mathbf{n}}_2 = \frac{1}{2}(\sqrt{2\lambda_1}\mathbf{u}_1 + \sqrt{-2\lambda_3}\mathbf{u}_3) \tag{3.19}$$

Of the four possible solutions show in table 3.1 two can be easily be removed by imposing the positive depth constraint $\mathbf{n}^T > 0$ as the camera can only see points infront of it. However there is still ambiguity in deciding the final solution which might be decided using additional measurements.

3.5.2 Explicit velocity calculation with aiding measurements

From eq. (2.2) the following equations can be extracted

$$X_{of} = \frac{v_z x - v_x f}{Z} - \omega_y f + \omega_z y + \frac{\omega_x xy - \omega_y x^2}{f} \tag{3.20}$$

$$Y_{of} = \frac{v_z y - v_y f}{Z} - \omega_x f + \omega_z x + \frac{\omega_x y^2 - \omega_y xy}{f} \tag{3.21}$$

Solving for velocity leads to

$$v_x = \frac{1}{f}(v_z x + Z(-X_{OF} + \omega_z y + \frac{\omega_x xy - \omega_y x^2}{f}) - \omega_y Z \tag{3.22}$$

$$v_y = \frac{1}{f}(v_z y + Z(-Y_{OF} + \omega_z x + \frac{\omega_x y^2 - \omega_y xy}{f}) - \omega_x Z \tag{3.23}$$

$$\tag{3.24}$$

This equation must be preformed individually for each pixel in the the image, however using the assumption that the camera is perpendicular to the surface of interaction the terms that include the pixel location will cancel out by symmetry due to the distance Z distance being equal for opposing pixels. This leads to the following equation for velocity:

$$v_x = Z(X_{OF}/f - \omega_y) \tag{3.25}$$

$$v_y = Z(Y_{OF}/f - \omega_x) \tag{3.26}$$

Using this expression it is only necessary to use the average flow value of the flow field.

When using a feature based algorithm the feature is normally pinpointed at a exact pixel location. However using a camera with a focal distance of 2.8mm and pixel size of 6.9um at a distance of 1m from the scene means that moving one pixel in the image is equivalent of a 2.4mm movement in the scene. The nature of the optical flow calculation allows for a movement to be detected in the sub pixel region meaning that even intensity changes in the target pixel should result in estimated motion.

3.6 Kalman filter

Throughout the project various filters were developed in order to use the provided sensor data to estimate the camera frame state. The classic Kalman filter in Kalman (1960) relies on the assumptions that:

- Linearity of process and observation models
- Disturbances are modeled as Gaussian noise

The Kalman filter state update is a recursive state estimate, meaning that the state estimation is only derived from the previous state and the new measurement. The estimation process is divided into two steps: prediction and update. The state of the filter is represented by the following two variables.

- $\hat{\mathbf{x}}$ is the filter state estimate with dimension *n* equal to the number of states
- **P**, the filter estimate covariance matrix. With the dimension n x n. This is a measure of the accuracy of the current state estimate.

In order to properly estimate a process with measurement noise and unmodeled process noise the following matrices are specified:

- **A** The state-transition matrix
- **H** The observation model
- **Q** The process noise covariance
- **R** The observation noise covariance

The observation of the system true state is therefore according to

$$\mathbf{z} = \mathbf{H}\mathbf{x} + \mathbf{v} \tag{3.27}$$

where **v** is the noise vector with covariance **R**.

Note that additional information of the control input can be included, but is omitted here as this was not the case in this project.

Prediction

The prediction step is a propagation of the filter state from the previous to the current time. This is done according to the state transition matrix as follows, note subscript *k* indicates the current time step.

$$\mathbf{x}_k = \mathbf{A}_k\hat{\mathbf{x}}_{k-1} \tag{3.28}$$

Every time the state is updated, the corresponding covariance matrix is updated accordingly.

$$\mathbf{P}_k = \mathbf{A}_k\mathbf{P}_{k-1}\mathbf{A}_k + \mathbf{Q}_k \tag{3.29}$$

Update

The update step incorporates the measurement into the current estimate. First the innovation **y** and the innovation covariance **S** are calculated as follow

$$\mathbf{y}_k = \mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_k \tag{3.30}$$

$$\mathbf{S}_k = \mathbf{R} + \mathbf{H}\mathbf{P}\mathbf{H}^T \tag{3.31}$$

The innovation measures the difference between the measured state and the estimated state. Since **S** and **P** give accurate measures of the respective accuracy the optimal Kalman gain is calculated by

$$\mathbf{K}_k = \mathbf{P}\mathbf{H}^T\mathbf{S}^{-1} \tag{3.32}$$

Intuitively the Kalman gain is larger if the confidence in the measurement is larger than the estimated state. Finally the state and state covariance are updated.

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k + \mathbf{K}_k\mathbf{y}_k \tag{3.33}$$

3.6.1 Extended Kalman filter

Usage of the Kalman filter can be extended to encompass non-linear systems by use of the aptly named Extended Kalman Filter (EKF). The EKF is based on the same equations as the KF, however the state prediction is completed directly using the model non-linear function *f*, while the state covariance is estimated using the Jacobian of f:

$$\hat{\mathbf{x}}_k = f(\hat{\mathbf{x}}_{k-1}) \tag{3.34}$$

$$\mathbf{P}_k = \mathbf{A}\mathbf{P}\mathbf{A}^T + \mathbf{Q}^T \tag{3.35}$$

Similarly the updated equations become

$$\mathbf{y}_k = \mathbf{z}_k - h(\hat{\mathbf{x}}_k) \tag{3.36}$$

$$\mathbf{S} = \mathbf{R} + \mathbf{H}\mathbf{P}\mathbf{H}^T \tag{3.37}$$

Here A is the Jacobian of f evaluated at state $\boldsymbol{x}$ and H the Jacobian of h at state $\mathbf{x}$.

$$\mathbf{A} = \left.\frac{\partial f}{\partial \hat{\mathbf{x}}}\right|_{\hat{\mathbf{x}}_k} \tag{3.38}$$

$$\mathbf{H} = \left.\frac{\partial h}{\partial \hat{\mathbf{x}}}\right|_{\hat{\mathbf{x}}_k} \tag{3.39}$$

The rest of the equations are identical to the linear Kalman filter.

As the state covariance can only be approximated the filter loses its optimality, the performance of the filter is therefore highly dependant on if the linearization is a good approximation of the underlying model.

3.6.2 Error state Kalman filter

A large portion of the time spent during this project was spent developing an Error State Kalman Filter (ESKF). While this filter was not used during the evaluation of the system due to large noise in the IMU and lack of time for validation, the theory is still included for reference and the implementation is included in the digital attachments. For more in depth reading on error state Kalman filters and the quaternions kinematics see Solà (2017).

In an error state filter one must consider three different values: the true-, nominal- and error-state of the system. The true value is a composition of the nominal and error state. The idea of the ESKF is to integrate the high rate IMU data using strapdown equations into a nominal state $\mathbf{x}$. Due to inaccuracies due to noise and biases this integration will accumulate small errors summarised by the filters error state $\delta\mathbf{x}$. This error state incorporates all errors, noise and perturbations, and is estimated in parallel with the nominal state integration. When additional sensor data is available (such as Rovio or Optical flow) the error state is corrected and a posterior Gaussian estimate is injected into the nominal state and reset to zero.

The benefits of such an estimator is that since the error state is always small and operating close to the origin we can use a minimal set of parameters to represent it and avoid

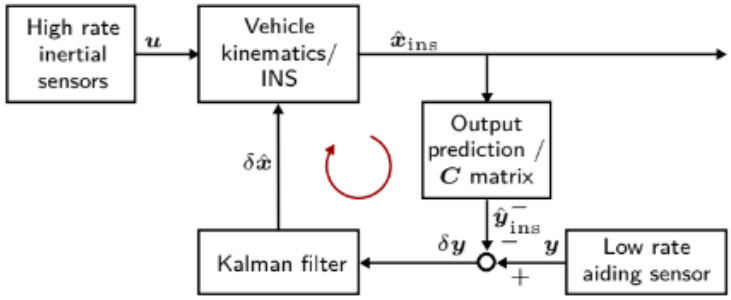


Figure 3.4: Block diagrams of an error state Kalman filter

issues related to singularities. In addition as the system dynamics are generally captured by the nominal state the error state dynamics are slow and can be updated at a lower frequency.

Another benefit is that the error state can be updated at the time the measurement occurred and propagated forward to the current state, allowing a mathematically consistent way of fusing measurements with different time delays, while still maintaining a high frequency estimation of the current state. This feature is very useful when dealing with images that have a processing delay.

3.6.3 Kinematics

The true state is a composition of the nominal state and error state such that $\mathbf{x}_t = \mathbf{x} + \delta\mathbf{x}$. The true state kinematics are as follows

$$\dot{\mathbf{p}}_t = \mathbf{v}_t \tag{3.40a}$$
$$\dot{\mathbf{v}}_t = \mathbf{R}_t(\mathbf{a}_m - \mathbf{a}_{bt} - \mathbf{a}_n) + \mathbf{g} \tag{3.40b}$$
$$\dot{\mathbf{q}}_t = \frac{1}{2}\mathbf{q}_t \otimes (\boldsymbol{\omega}_m - \boldsymbol{\omega}_{bt} - \boldsymbol{\omega}_n) \tag{3.40c}$$
$$\dot{\mathbf{a}}_t = \mathbf{a}_w \tag{3.40d}$$
$$\dot{\boldsymbol{\omega}}_t = \boldsymbol{\omega}_w \tag{3.40e}$$
$$\tag{3.40f}$$

where the subscript *t* indicates the true state value, *m* are values measured by the IMU and *bt* and *n* indicate the true bias and noise respectively. $\mathbf{R}_t$ is the rotation matrix between the body frame and earth frame.

The nominal state kinematics are derived from the modeled system by removing the

noises,

$$\dot{\mathbf{p}} = \mathbf{v} \tag{3.41a}$$
$$\dot{\mathbf{v}} = \mathbf{R}(\mathbf{a}_m - \mathbf{a}_b) + \mathbf{g} \tag{3.41b}$$
$$\dot{\mathbf{q}} = \frac{1}{2}\mathbf{q} \otimes (\boldsymbol{\omega}_m - \boldsymbol{\omega}_b) \tag{3.41c}$$
$$\dot{\mathbf{a}} = 0 \tag{3.41d}$$
$$\dot{\boldsymbol{\omega}} = 0 \tag{3.41e}$$
$$\tag{3.41f}$$

The error state kinematics can then be derived from the true- and nominal state,

$$\delta\dot{\mathbf{p}} = \delta\mathbf{v} \tag{3.42a}$$
$$\delta\dot{\mathbf{v}} = -\mathbf{R}[\mathbf{a}_m - \mathbf{a}_b]_\times \delta\boldsymbol{\theta} - \mathbf{R}\delta\mathbf{a}_b - \mathbf{R}\mathbf{a}_n \tag{3.42b}$$
$$\delta\dot{\mathbf{q}} = -[\boldsymbol{\omega}_m - \boldsymbol{\omega}_b]_\times \delta\boldsymbol{\theta} - \delta\boldsymbol{\omega}_b - \boldsymbol{\omega}_n \tag{3.42c}$$
$$\delta\dot{\mathbf{a}} = \mathbf{a}_w \tag{3.42d}$$
$$\delta\dot{\boldsymbol{\omega}} = \boldsymbol{\omega}_w \tag{3.42e}$$
$$\tag{3.42f}$$

where $[\mathbf{x}]_\times$ represents the skew symmetric operator. Here equations eq. (3.42b) and eq. (3.42c) require some none trivial manipulations of the non-linear equations eq. (3.41b) and eq. (3.41c) to obtain the linearized dynamics, for a full proof refer to Solà (2017).

In order to implement the state kinematics they are converted to discrete time and the nominal state kinematics become

$$\mathbf{p}_{k+1} = \mathbf{p}_k + \mathbf{v}_k\Delta t + \frac{1}{2}(\mathbf{R}_k(\mathbf{a}_{m,k} - \mathbf{a}_{b,k}) + \mathbf{g})\Delta t^2 \tag{3.43a}$$
$$\mathbf{v}_{k+1} = \mathbf{v}_k + (\mathbf{R}_k(\mathbf{a}_{m,k} - \mathbf{a}_{b,k}) - \mathbf{g})\Delta t \tag{3.43b}$$
$$\mathbf{q}_{k+1} = \mathbf{q}_k \otimes \mathbf{q}\{(\boldsymbol{\omega}_{m,k} - \boldsymbol{\omega}_{b,k})\Delta t\} \tag{3.43c}$$
$$\mathbf{a}_{b,k+1} = \mathbf{a}_{b,k} \tag{3.43d}$$
$$\boldsymbol{\omega}_{b,k+1} = \boldsymbol{\omega}_{b,k} \tag{3.43e}$$

Here the notation $q\{\mathbf{x}\}$ represents the *rotation vector to quaternion* formula.

$$\mathbf{q}\{\mathbf{v}\} = \phi\mathbf{u} = \begin{bmatrix} cos(\phi/2) \\ \mathbf{u}sin(\phi/2) \end{bmatrix} \tag{3.44}$$

Where $\mathbf{v}$ represents a rotation of ϕ rad around the axis $\mathbf{u}$

3.6.4 Error state transition

The prediction step carried out for the error state is similar to a normal EKF and are written as,

$$\delta\hat{\mathbf{x}}_k = f(\mathbf{x}_{k-1}, \mathbf{u}_k) \cdot \delta\hat{\mathbf{x}} \tag{3.45}$$
$$\mathbf{P}_k = \mathbf{F}_k\mathbf{P}_{k-1}\mathbf{F}_k^T + \mathbf{Q}_k \tag{3.46}$$

$\mathbf{F}_k$ is the Jacobian of f() with respect to the error, and Q is the covariance of the IMU.

$$\mathbf{F}_k = \left.\frac{\partial f}{\partial \delta \mathbf{x}}\right|_{\mathbf{x}_k \mathbf{u}_k} \tag{3.47}$$

Of course as the error state is not observable by the IMU measurements, eq. (3.45) will always return zero as the Jacobian is linearized around the nominal state and the mean of the error is zero. While the predicted error state is zero, the associated covariance increase with every prediction step.

3.6.5 Fusing external state measurements

Three stages.

- Observation of the error state
- Inject error state into nominal state
- Reset error state

Observation of the error state

Sensor data is observed following:

$$\mathbf{y} = h(\mathbf{x}_t) + v \tag{3.48}$$

where *h*() is a differentiable function and *v* is noise.

In order to calculate the correction equations the observation model Jacbian **H** must be defined with respect to the error state. However, as the error state has not been observed yet the true state estimate is simply the nominal state, $\hat{\mathbf{x}}_t = \mathbf{x}$.

$$\mathbf{H}_k = \left.\frac{\partial f}{\partial \delta \mathbf{x}}\right|_{\mathbf{x}_k} \tag{3.49}$$

For most measurements **H** is trivially calculated as standard Jacobian of *h*(). However if the measurement is of the state attitude **q** the measurement must be converted to delta angles in order to integrate into the error state.

Injection of the observed error and reset

This is easily done by using the appropriate composition

$$\mathbf{x} = \mathbf{x} \oplus \delta \hat{\mathbf{x}} \tag{3.50}$$

After injection the observed error state is reset to 0.

Chapter 4

Evaluation

4.1 Experimental setup

During the development phase of the project the lab was closed due to coronavirus. This led to the need to have a test platform that was usable in a home office environment. For this purpose the camera, IMU and versavis board were removed from the MAV and rigidly mounted to a testing board. Tests with this board was done with the full camera resolution at 40Hz.

When mounted on the MAV the same camera that was used for Rovio was used for optical flow measurements. This camera is mounted at a 45 degree angle to the drone body frame. As applications in the future intend to have the camera pointing perpendicular to the ground the MAV was flown at a 45 degree angle in order to compensate for its mounting offset. Tests in this configuration was done at 120hz and with a region of interest of 256x256 set in the center of the image.

For ground truth (GT) measurements an external motion capture system was used, dubbed vicon by which the measurements are received using a vrpn client (Virtual-Reality Peripheral Network). The notation in the plots uses the notation vrpn_client, vicon and GT interchangeably.

4.1.1 Components

The critical components that are needed for the estimation to be viable are a camera and IMU. This project used a versavis board in order to hardware synchronize the camera and IMU measurements. The IMU used is the ADIS16448 and the camera is the FLIR Firefly S FFY-U3-04S2M. In addition some method of measuring the scene distance is necessary, here tis was accomplished using the height measurement from vicon.

4.1.2 Software

ROS was used as a middle ware layer in order to facilitate communication between various modules and over the network. The FLIR firefly is provided with a ROS driver called the spinnaker SDK which was used to set the region of interest of 256x256 pixels and setup such that it could be manually triggered with the versavis board.

4.1.3 Calibration

For each setup an initial calibration of the platform was performed in order to determine the exact transformation matrix between the IMU and camera, the distortion parameters and intrinsic matrix of the camera. The tool used to complete this was kalibr developed in Furgale et al. (2013), and a new calibration was preformed each time the testing configuration was adjusted. Rovio will rapidly diverge with even small errors in calibration. Additionally, in order to properly compensate angular flow as discussed in section 3.5 the precise transformation matrix between camera and IMU is necessary in order to transform the measured angular velocity to the camera frame.

4.1.4 Camera setup

The camera is capable of capturing frames at 720x540 @ 120Hz. However this is with a significant amount of distortion at the edges of the frame. This distortion complicates the calculation of optical flow and can be compensated using the distortion parameters calculated by kalibr as seen in fig. 4.1 and fig. 4.2 The rectified image was tested in

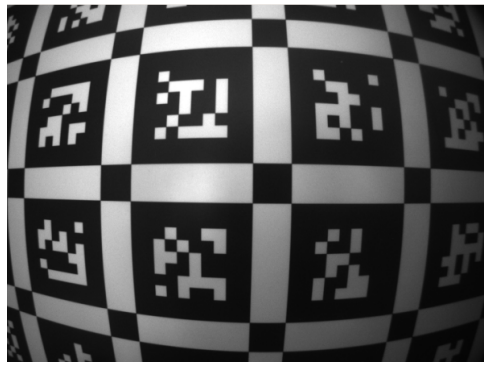


Figure 4.1: Image without undistoring

various environments and in lower light, the corners in the image appear very dark and cause a significant amount of noise. In order to both reduce the run-time of the optical flow algorithm and eliminate the large distortion and dimming effects at the edges of the frame it was decided to create a region of interest(ROI) in the center of the image. The DIS-flow algorithm requires a resolution that is a power of two, if this is not done it will pad the image which sometimes leads to artifacts at the image edges. Therefore, a natural choice for the ROI was a 256x256 area in the center of each frame, here the distortion is negligible and an even amount of light is hitting the sensor.

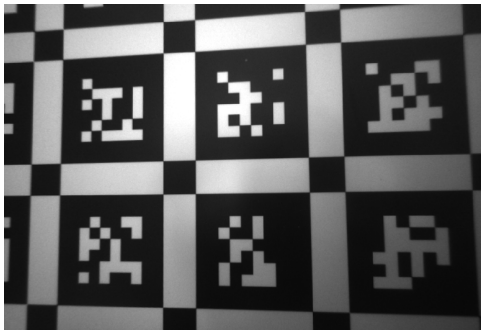


Figure 4.2: Image after undistoring

4.1.5 Data collection

As the camera mounted on the drone is angled 45 degrees, data collection was collected in three main ways

- Handheld movement
- Takeoff to landing with the drone flying horizontally
- Takeoff to landing where the drone is flying at a 45 degree angle such that the camera is perpendicular to the ground

The handheld tests give a good indication of the algorithm performance, however the flight tests give a more realistic example including vibrations and oscillations. The data sets where the camera is mainly facing perpendicular to a surface are the most realistic for the intended usage of the system, and these are the results that will be presented in this section.

4.2 Optical flow

In this section the three main options: DiSFlow, PCA-flow and coarse2fine, will be evaluated in order to decide which should be used as the underlying flow calculation method. In order to evaluate the accuracy of the methods a combination of the reported accuracy and a qualitative assessment were considered. Coarse2fine was not evaluated on any standardized data set and will only be evaluated qualitatively. In order to visualize the flow, a color image is used where the intensity of the color is related to the magnitude of the flow while the color indicates the direction according to the color wheel in fig. 4.3.

The accuracy of an optical flow method is described by the Endpoint Error (EPE). This is calculated by comparing the estimated displacement vector $\mathbf{h}_e$ with the ground truth optical flow vector $\mathbf{h}_{gt}$ where the endpoint error is the Euclidian distance between the two. Typically the most prominent metric is the frame average EPE, however, some specialized algorithms might be better suited for large or small displacements in which case the EPE for the pixels that have moved more or less than some benchmark value are evaluated.

Figure 4.3: Color wheel used to represent pixel movement

4.2.1 PCA-flow

PCA-flow has an average EPE of 8.6 according to Butler et al. (2012) and despite using a feature based method does not seem to have any particular advantage at large displacements.

PCA flow was evaluated on the handheld test rig, meaning that the testing process was done with the full camera resolution of 720x540. At this resolution it was able to achieve an average speed of 1.5Hz with a flow as seen in fig. 4.4.

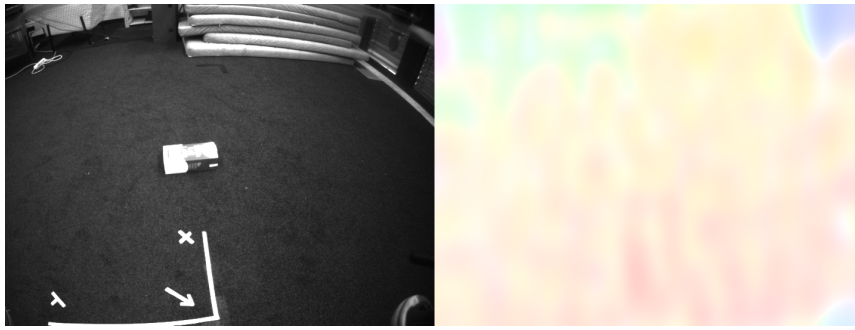


Figure 4.4: Flowfield of PCA-flow at full resolutio n

As can be seen in the flow output there is a slight oscillation to the flow. This occurs due to the interpolation using biases between matched feature points, showing that principal components used to segment motion in media is not a great fit for this application.

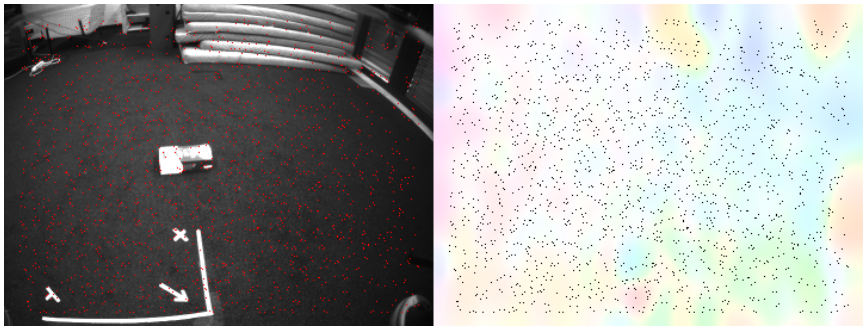


Figure 4.5: Feature points of a frame

In an attempt to give the algorithm the best possible run time, various parameters were adjusted. The highest speed was achieved by halving the image resolution and changing the feature detector to use FAST features. After tuning the mean frequency it was 4.5Hz and due to this still not being sufficient for the intended application the algorithm was disregarded from further tests.

4.3 Coarse 2 fine

Coarse2fine produced high quality results when tested on stock footage taken from a high altitude drone seen in fig. 4.6. However, the rendering time per image was 22s. As this method uses many of the same concepts as DISflow but at a much slower speed it was not explored further in this report.

4.3.1 DIS flow

Dense inverse search has an EPE of 10.13, which is more then double the error of the highest preforming algorithms and almost 20% worse than PCA-flow. The main attribute of DISflow was that it was developed with minimizing time complexity in mind.

When discussing the speed of the algorithm it is important to mention preprocessing times. While the paper lays out that the frequency of the algorithm can reach 600Hz this excludes the preprocessing time needed to access the disc and calculate the gradients of the image. When this is included the frequency decreases to a maximum of 46Hz.

In a scenario where a video must be processed in real time, these preprocessing steps are essential to the overall performance of the algorithm. Using the ROS image transport library, the disc access time can be eliminated, some overhead still persists due to the need to serialize the image. However, this can be further improved by setting up the flow calculation node and the camera driver to be managed by the same nodlet manager.The image gradients must be calculated for each incoming image, but by ensuring that this is only done once for each image the preprocessing time can be further reduced by half. Finally rescaling images before the flow calculation is necessary in order to build the multi scale pyramid as discussed in section 3.4.4. The algorithm also must pad the image in order

Figure 4.6: An example frame of the estimated flow during a pure translational movement

to ensure that it restlessly divisible on all scales of the pyramid, by carefully selecting a region of interest in the camera driver that corresponds to this resolution, initial padding and removal of padding is not needed. The final preprocessing for each new image frame using the final 256x256 resolution was thereby reduced to approximately 2ms from 20ms. While this is a significant improvement the entire flow calculation must take less than 8ms in order to run at the cameras frame rate limit of 120Hz. When conducting tests the processing times of a single frame could sometimes spike to 12-15ms causing a dropped frame. These spikes are most likely due to a preemption of the process, however these dropped frames did not significantly cause any loss in accuracy so it was decided to keep the quality at this level and have a frequency of around 110Hz on the 120Hz stream.

The initial tests of the algorithm were performed on the test rig so parameter exploration would only provide approximate results of what would be seen on the final system, particularly in regard to timing. The laptop that carried out these tests completed the preprocessing at approximately 10ms for the full resolution image after the optimization steps were implemented. The system was almost able to keep up with the camera at full resolution running at 40Hz, dropping frames occasionally but with an average calculation time of 30ms. An example of the produced flow at this stage is shown in fig. 4.7.

To see what the result would look like when running at very high frequencies the input resolution was reduced by half and patch overlap lowered to 0.3. Using this setup the average calculation time was reduced to 10ms at a significant cost to the quality as seen in fig. 4.8.

These initial results were promising and provided significantly better overall results than any other method. Moving forward this method would become foundation for further results.

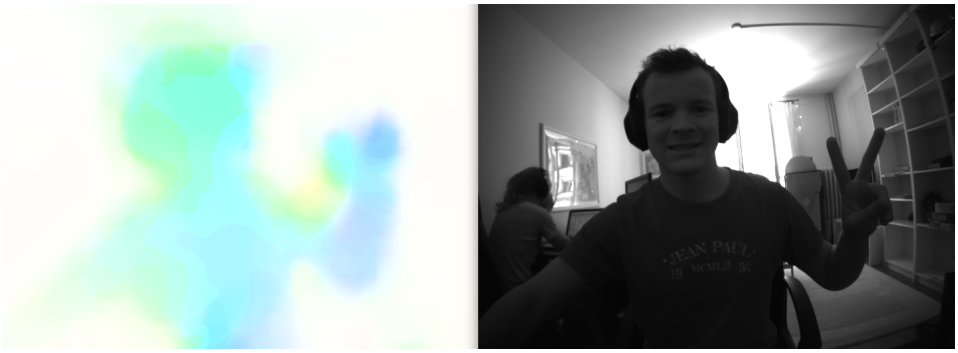


Figure 4.7: Initial testing with full resolution image

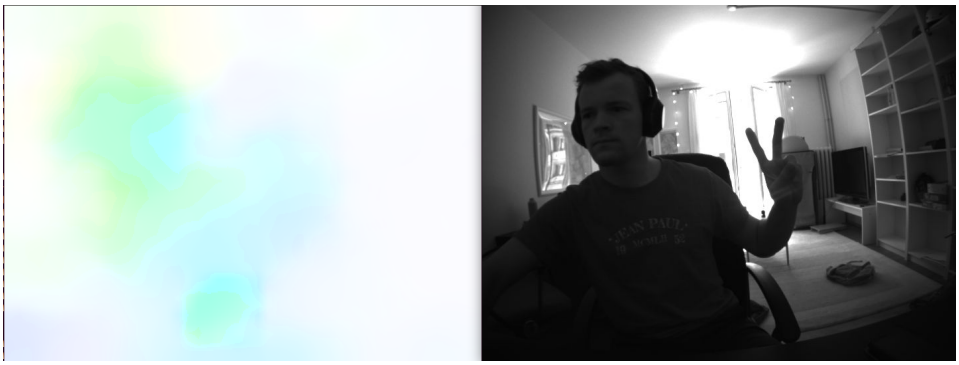


Figure 4.8: Flow output when average run time is 10ms

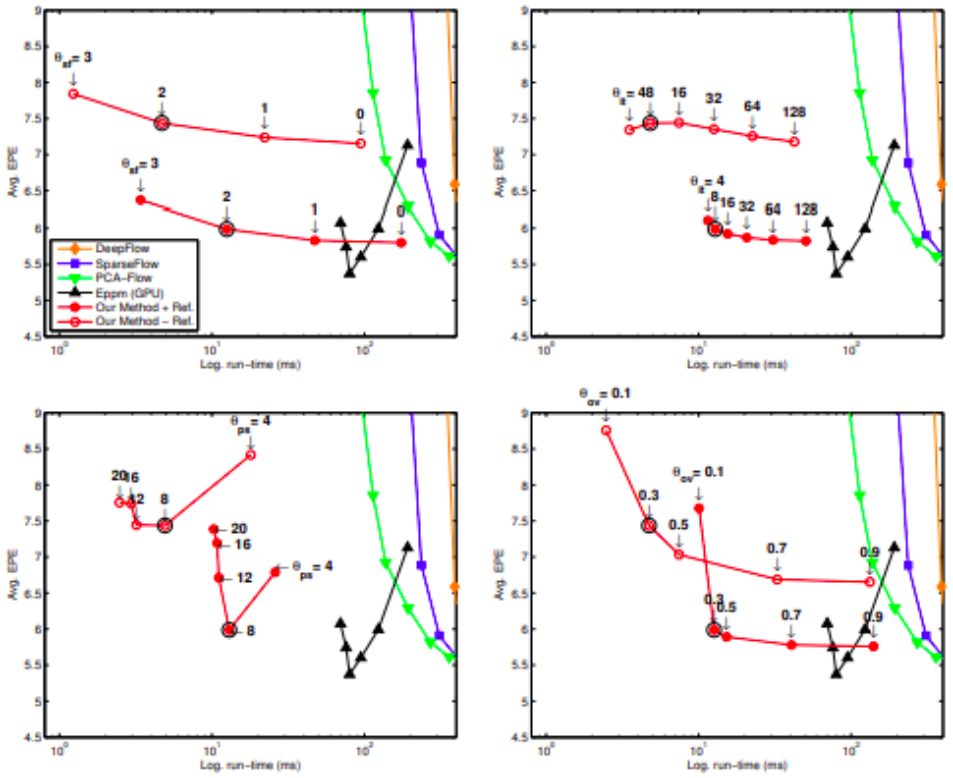


Figure 4.9: Optical flow results while individually varying parameters. Top left to bottom right shows variation of: Finest scale in pyramid, gradient decent iterations, patch size, patch overlap. Image from Kroeger et al. (2016)

Parameter exploration

In Kroeger et al. (2016) various parameters are explored in order to optimize run speed accuracy trade off show in fig. 4.9.

These results were verified on the target system and the final parameters used were:

Parameter	Value
Finest scale in pyramid	1 (0 indexed)
Number of iterations	12
Patch size	8x8
Patch overlap	0.4
Intensity weight	10
Gradient weight	10
Smoothness weight	20

Note that the finest scale in the image pyramid is by far the factor that has the largest in-

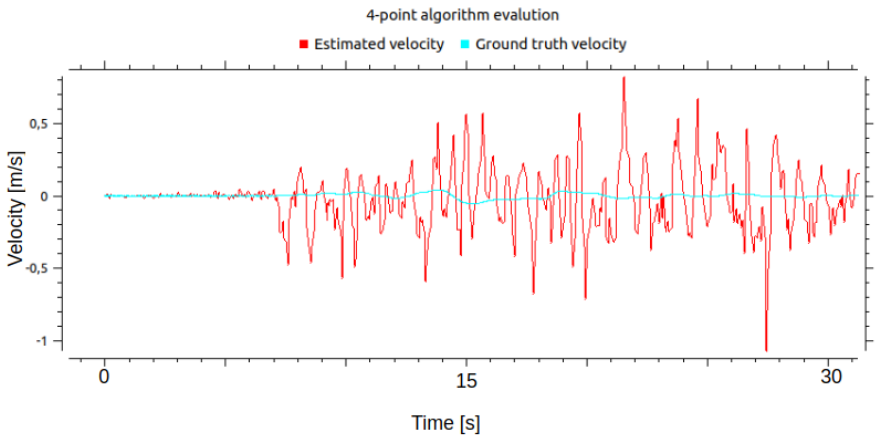


Figure 4.10: Caption

fluence on the speed of the algorithm. During testing this was the main parameter that was tuned in order to achieve speeds in the desired range, and was what was most rigorously explored.

4.4 Ego-motion estimation

4.4.1 4-point algorithm

Testing was done using the 4-point algorithm in order to determine the camera velocity and plane orientation, however it became apparent that without including further information such as in Grabe et al. (2015) this method is not a viable option for ego-motion calculation. Typical velocity estimations can be seen in fig. 4.10

It is an interesting option to explore in the future as the scene normal vector should become observable when two frames are captured from different locations, with further development this method should be able to provide usable results.

4.4.2 RANSAC

In the OpenCV library an option is available to filter outlying points when using various integrated 3D reconstruction methods using RANSAC.

RANSAC is short for RAndom SAmple Consensus and is an iterative method for filtering outliers. In areas with very few features, such as over- and underexposed surfaces, optical flow methods suffer from ambiguous flow values. In some cases it might be desirable to not consider these values at all when calculating the ego-motion of the MAV. Unfortunately these integrated methods proved too slow as they are applied within other functions, running at an average of 65ms per 128x128 image and slower the more outliers

there are in the image. The produced results are quite promising, and if the flow calculation allows, this should be integrated in the future. Some qualitative results are displayed in fig. 4.11 and fig. 4.12 where the MAV was flying above a featureless white table with a piece of wood mounted on top, the wood allowed for good flow results, while the table produced very noisy results.

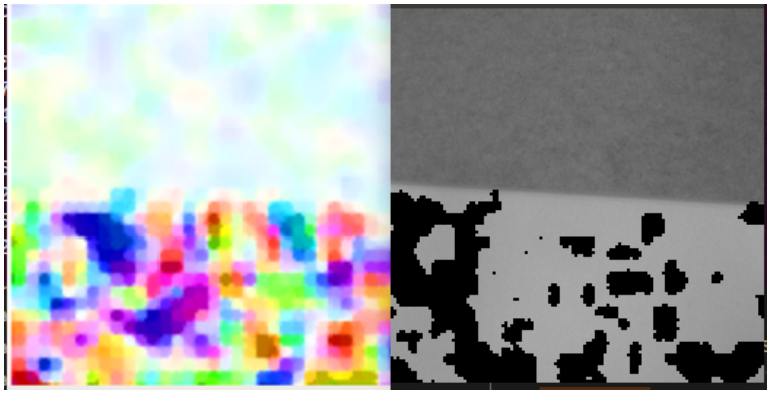


Figure 4.11: Image showing which regions are filtered using RANSAC with a reprojection error threshold of 1. The right image shows the original flow while the in the left image shows the input image with the occluded regions marked as outliers based on the flow.

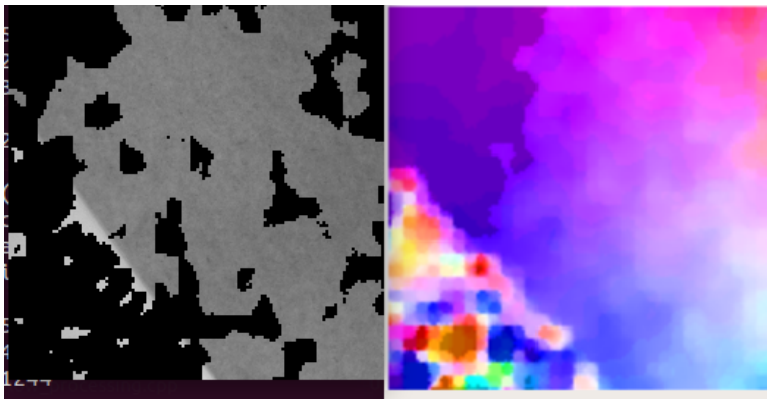


Figure 4.12: Image showing which regions are filtered using RANSAC with a reprojection error threshold of 0.1. The right image shows the original flow while the in the left image shows the input image with the occluded regions marked as outliers based on the flow.

4.4.3 IMU rotation correction

The optical flow is received as a matrix of pixel displacements from one frame to another. In order to calculate the relative velocity of the observer it is first necessary to calculate

the flow speed. This is done by calculating the focal length in pixels according to

$$f_{pix} = f_{mm}/(s_i/s_f * d) \tag{4.1}$$

Here f_{mm} is the camera focal length in mm, s_i is the original image size and s_f is the size of the optical flow field and d is the pixel size in mm. Note that this assumes the flow has been scaled equally in the x and y axis, if this is not the case the focal length must be computed individually in the x and y axis. Dividing the average flow rate by the integration period results in the flow velocity. Applying eq. (3.25) the relative velocity $\frac{v}{Z}$ is found.

IMU noise

Rejection of angular flow was first tested on the handheld system. This provided a good metric in order to compare the average flow with the measured angular velocity. However when the same tests were run on the system during flight, the IMU noise had increased dramatically from a standard deviation of approximately 0.005 rad/s to 0.15 rad/s as seen in fig. 4.13.

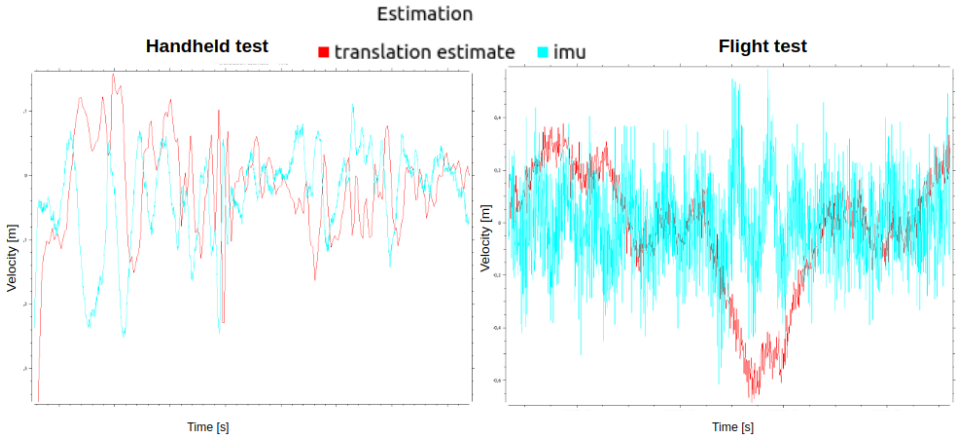


Figure 4.13: A comparison in IMU-noise between handheld and flight tests with the resulting relative velocity estimate

As the estimated relative velocity is a sum of the measured angular velocity, noisy IMU measurements directly result in noisy estimations. In fig. 4.14 this problem is illustrated. T

During the project, tests were conducted to see if increasing the IMU trigger speed from the versavis board had any effect. The default frequency of the IMU was 200Hz, but the ADIS IMU supports speeds of up to 700Hz. When attempting to run the camera at 120Hz and the IMU at 700Hz bandwidth became a large issue and messages were throttled. After reducing the IMU to 400Hz everything worked as expected, however the test done to determine if this change caused any improvement was carried out using handheld tests. This caused the trajectory to be inconsistent between tests making them hard to compare and more importantly the noise present during flight was not detected. This test should

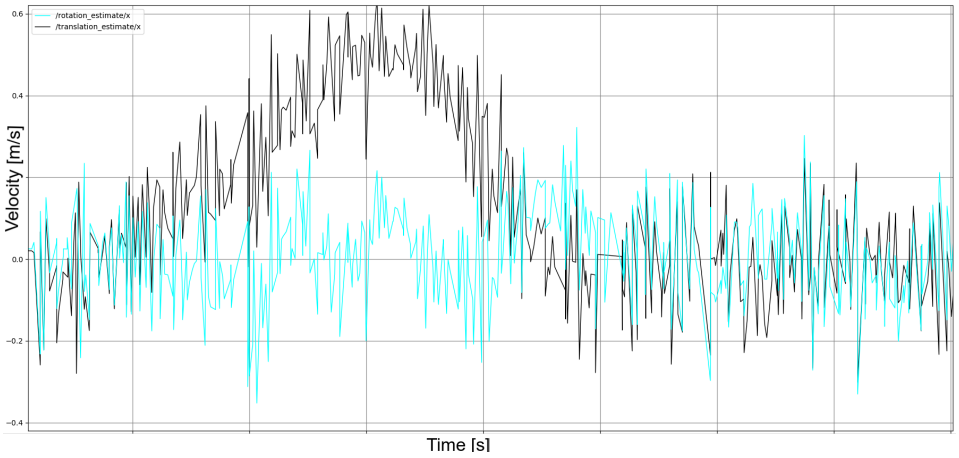


Figure 4.14: Estimated rotational estimate and resulting translation estimate. Observe how the rotational noise is directly transferred to the estimated velocity

be redone in the future to better determine if an increase in IMU speed could be used to filter more effectively the measured rotation as the handheld showed no improvement when running the IMU at higher frequencies and the IMU-rate was kept at 200Hz.

4.5 Estimating scene distance

If a flow field is captured while the MAV is moving while the camera is not perpendicular to the ground the assumption that the difference between the Z-distance between the central pixel and the the pixel at the edges of the frame is negligible might break. In order to account for scenarios with large amounts of parallax the average distance distance from the camera to the scene can be calculated. As the range sensor was not mounted yet the vicon ground truth measurement was used as the Z parameter to estimate the scene depth.

The average distance to the scene can be calculated as

$$\hat{Z}_i = Z \frac{1}{(\theta_i - \frac{\alpha}{2}) - (\theta_i + \frac{\alpha}{2})} \int_{\theta_i + \frac{\alpha}{2}}^{\theta_i - \frac{\alpha}{2}} cos(x)dx \tag{4.2}$$

$$= Z \frac{1}{\alpha}(sin(\theta_i + \frac{\alpha}{2}) - sin(\theta_i - \frac{\alpha}{2})) \tag{4.3}$$

$$= Z \frac{2}{\alpha} cos(\theta_i) sin(\frac{\alpha}{2}) \tag{4.4}$$

As velocity is calculated separately for x and y, θ_i is the rotation about the major axis that the Z measurement was taken $i \in [x, y]$. α is the angle of view the camera calculated by

$$\alpha = 2arctan\frac{d}{2f} \tag{4.5}$$

Here d is the size of the sensor and f is the camera focal length.

When the range sensor is employed, the above equations still hold, however note that these equations require the orientation of the drone with respect to the surface it is measuring. In order to get an accurate estimate of an arbitrarily angled surface it would require either additional depth information or flight in a linearizable region perpendicular to the surface.

4.6 Kalman filter

The Kalman filter used in the final evaluation was a standard EKF. As outlined in section 3.6.2 the ESKF uses the IMU value in order to propagate the filter however to keep the final results as unbiased as possible and showcase purely the accuracy the calculated flow the EKF was selected. The implementation of the ESKF is given as an attachment to this report, however it has not been rigorously validated.

The basic block diagram of the the filter is shown in fig. 4.15

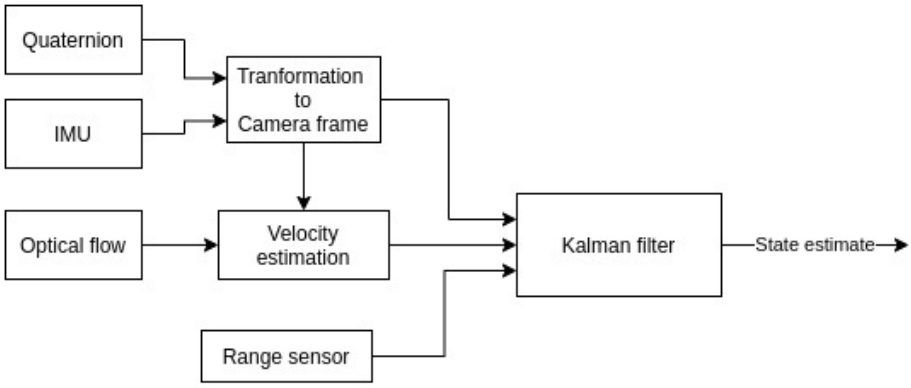


Figure 4.15: The Kalman filter inputs and outputs

Note here that the quaternion associated with the MAVs orientation is used as an input to the Kalman filter. This is done in order to integrate the velocity of the camera over time in order to get an accurate position in world frame. However if the only goal is to stabilize the end effector with reference to a planar surface this is not necessary and the end effector system can function independently of the external odometry.

4.7 Existing error

A few test data-sets were gathered in order to assess the current accuracy of Rovio in various situations where we would like to increase estimation precision.

In fig. 4.16 the platform is stationary on the ground with the motors switched off. In fig. 4.17 the platform is lifted manually and held motionless above the ground.

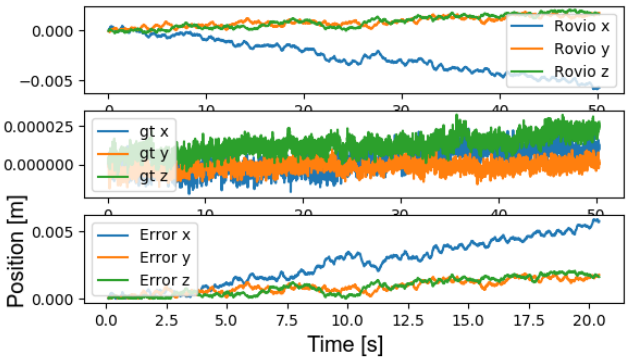


Figure 4.16: Estimation results with motors switched off, the Y-axis is the displacement in meters and the X-axis is time in seconds.

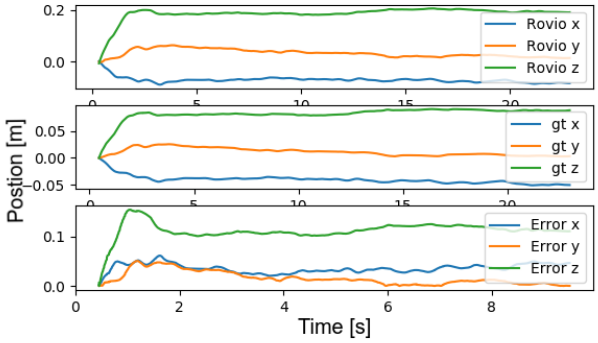


Figure 4.17: Estimation results where the platform is manually lifted off the ground and held in a stable position, the Y-axis is the displacement in meters and the X-axis is time in seconds.

From the figures it is evident that the estimation error is already quite small in these relevant cases indicating a challenging project. After the initial drift where Rovio estimates the depth of the tracked features the drift is typically in the sub-cm range.

4.8 Final experiments

Data was collected from various experiments where the OMAV was flown at an angle such that the camera was facing perpendicular to the floor. Unlike experiments done handheld these results include a large amount of noise both from the IMU due to air frame vibration and the OMAV tends to oscillate around the x-axis as some control authority is lost when doing non-level flights. As such, these experiments show how effective the algorithm is at rejecting rotational flow when computing velocity in a realistic setting.

During these tests Rovio was run concurrently with the flow calculation in order to

provide a comparison between the suggested method and the already existing odometry algorithm. However, when analyzing the results it was found that Rovio performed terribly under the conditions presented and would not provide a fair benchmark comparison. Typically Rovio will very quickly diverge when the features provided are insufficient or the provided camera calibration is bad. However during many of these tests Rovio often was able to track the movement for a while before diverging or be off by some scalar factor. While this was frustrating, it further showed the importance of an algorithm that is able to perform well in regions very close to surfaces when a feature may very quickly come in and out of frame. The plots of Rovio can be found in appendix B.

4.8.1 Hovering

This experiment represents the nominal use-case of the system where the main body attempts to hold a single position in the air. Here the MAV is hovering half a meter above the ground and attempting to stay centered at the origin. The filter state is initialized mid flight at the position estimated by vicon, after the initialization only the quaternion is used as an input in order to integrate the velocity over time to view the error over time in reference to vicon.

Figure 4.18 shows the general trajectory of the MAV over time. At approximately 6s there was a significant dip in the Z axis, as this is an unmodeled disturbance it is expected that some error is accumulated which can be seen in the Y-axis.

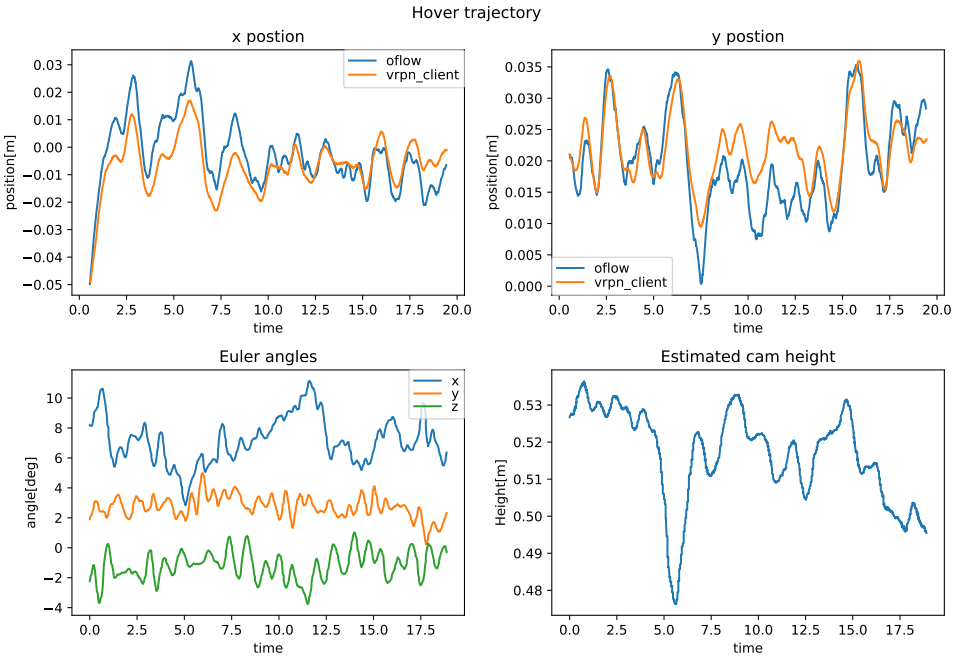


Figure 4.18: Hover trajectory

The measured velocity is measured in the camera frame so in order to compare the ground truth velocity to the measurement, both velocities are transformed to the world frame velocities. The resulting velocity is then an average over a 10 frame time period or approximately 0.1s for visualization purposes, as the raw velocity measurements is very noisy. The resulting velocity is shown in fig. 4.19.

From fig. 4.19 it seems like the main error source occurs during velocity peaks where the flow consistently overshoots the estimated absolute velocity. It is hard to pinpoint the origin of the error, but this is most likely due to noise generated in the image output. However, looking at the position in fig. 4.18 the absolute position error is quite similar during large and small displacements suggesting that the error is zero mean. The velocity error over time is shown in fig. 4.20.

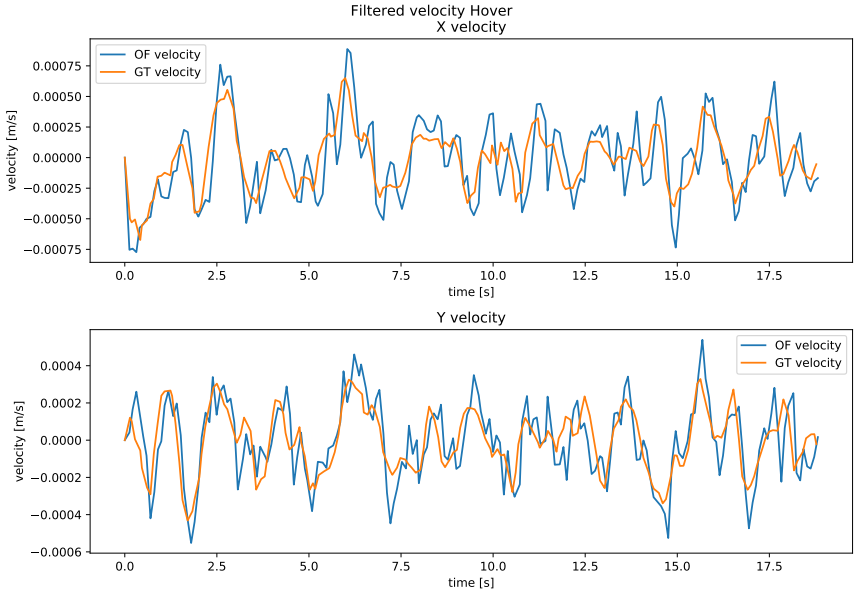


Figure 4.19: Hover velocity filtered

In order to show a clear plot when evaluating velocities the average velocity over a 10 frame period(approximately 0.1s) is shown as the raw data is quite noisy.

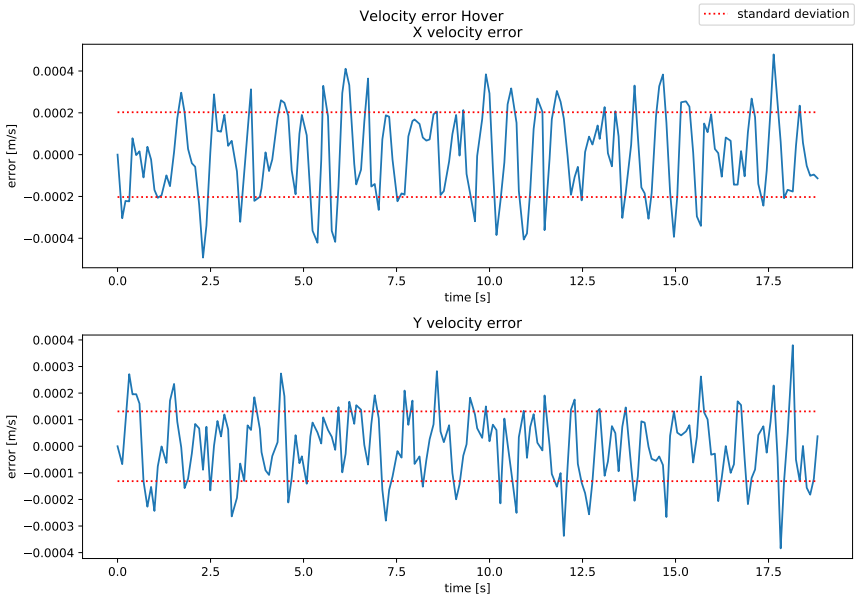


Figure 4.20: Hover trajectory

In fig. 4.20 it is clear that the error frequency is much higher than the velocity oscillation indicating that the error is not correlated with the current velocity of the drone. Much of the error is likely due to the camera not being perfectly aligned with the ground and oscillations drone attitude. In general the measured error is very small with a standard deviation of around 0.2mm/s.

4.8.2 Compound movement

After the nominal use-case was tested it was desired to see how well the velocity estimation performed during a more complicated use-case with translational motion. Rotation around the Z-axis and a combination of these movements, the full mission including way points for the trajectory is given appendix A.

The motivation preforming these tests was to be evaluate the possibility of giving the end-effector velocity commands and use the velocity measured using optical flow to enable high speed velocity tracking in reference to the surface being interacted with. This enables the arm to perform movements with high precision with respect to the surface and not the body.

Again the filter state is initialized to the currently estimated position of the system and from there is only updated with the body quaternion and optical flow measurement. Figure 4.21 shows how the position drifts over time. This shows that the current velocity has a tendency to drift quite significantly during angular velocities around the Z-axis. This is a string indication that the assumption of the camera being perpendicular to the ground can cause significant drift during rotations. The reason that the y position is affected more by this rotation is due to the MAV being more off axis around the X-axis than the Y-axis.

Otherwise the poorly translational movements are tracked quite well, again the y-position has drifted due to the offset axis.

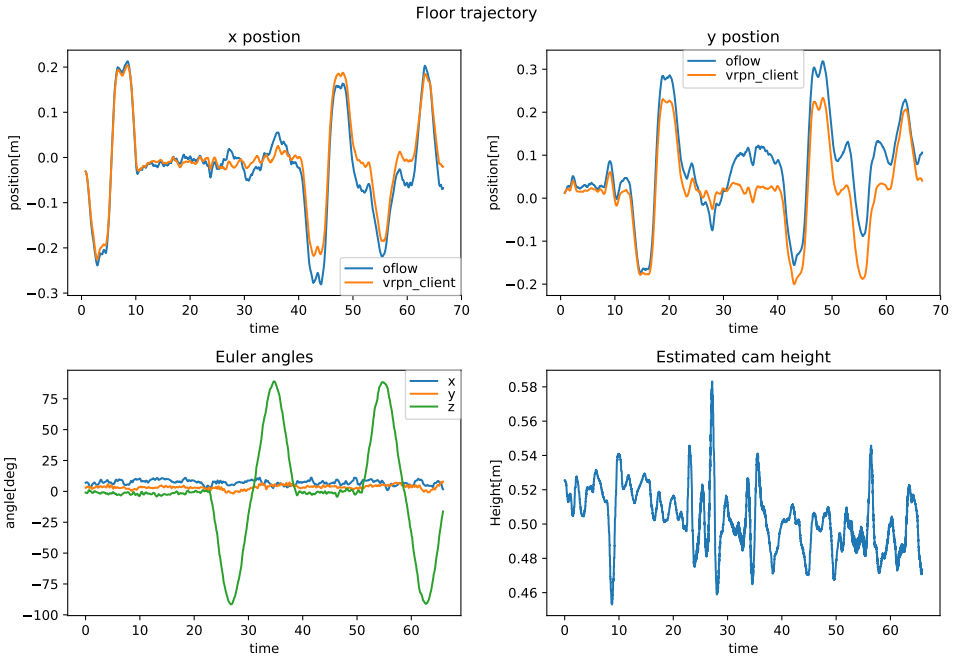


Figure 4.21: Floor trajectory

When inspecting the velocity in fig. 4.22 and fig. 4.23 the tracked velocity looks very accurate, the main drifts occur during the translational motion, as was expected. The standard deviation of the error is in the same range as earlier at 0.2mm/s and should provide sufficient accuracy to stabilize the end-effector.

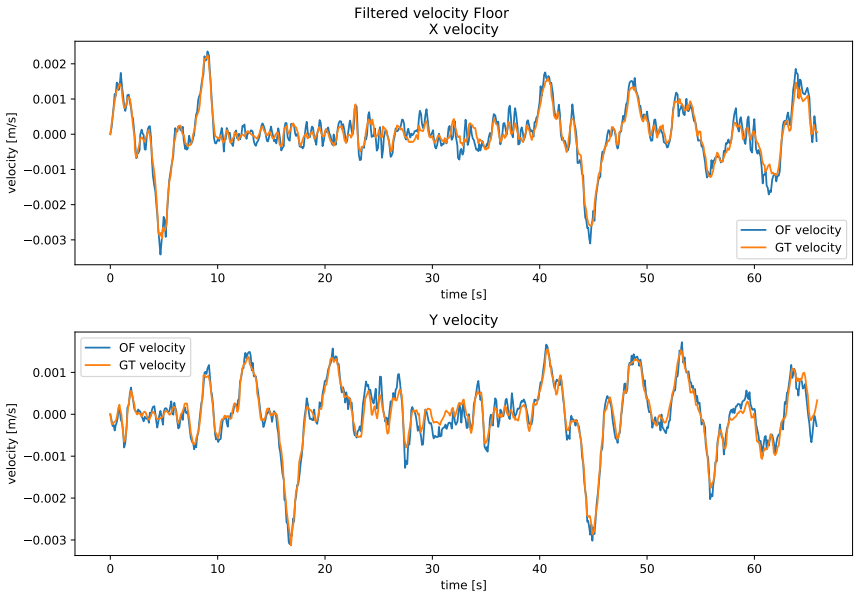


Figure 4.22: Floor trajectory

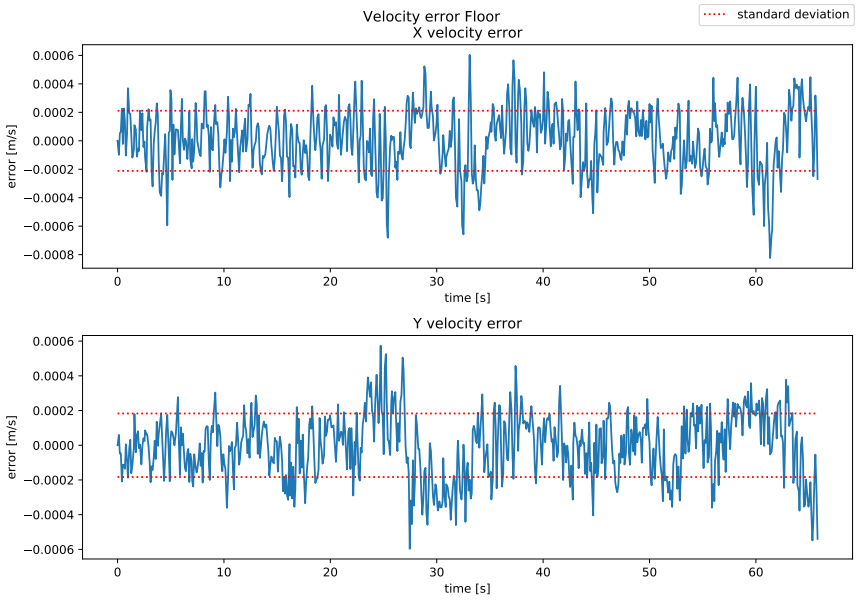


Figure 4.23: Hover trajectory

These small drifts could be minimized by fusing the velocity measurement with the forward kinematics from the on-board state estimation algorithm.

4.8.3 Tabletop movement

This demonstrates the ability of the optical flow to measure the velocity during the same compound movements described in the previous section. This test however was conducted while hovering over a table that was extremely featureless. This environment proved exceptionally difficult for Rovio which diverged almost immediately, some examples of flow output is shown in fig. 4.24 and fig. 4.25. Testing showed that the noise generated on texture-less surfaces was mainly zero mean, which should not affect the results during translation significantly, however during rotation this will cause one part of the image to become zero mean which will therefore cause the assumption used in section 3.5 - that pixels on opposing sides of the image will cancel out - to fail.

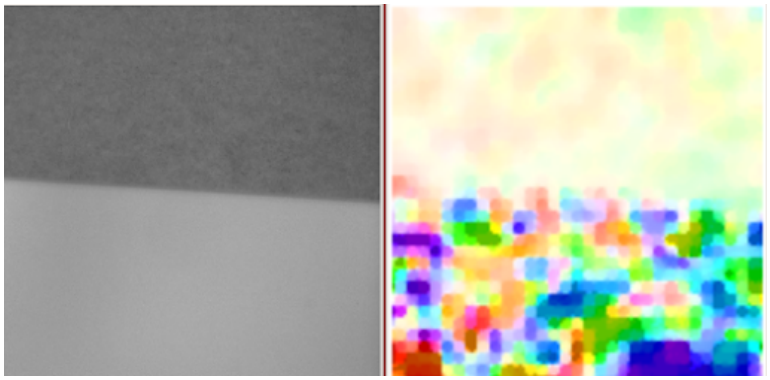


Figure 4.24: The flow output while hovering in place above table

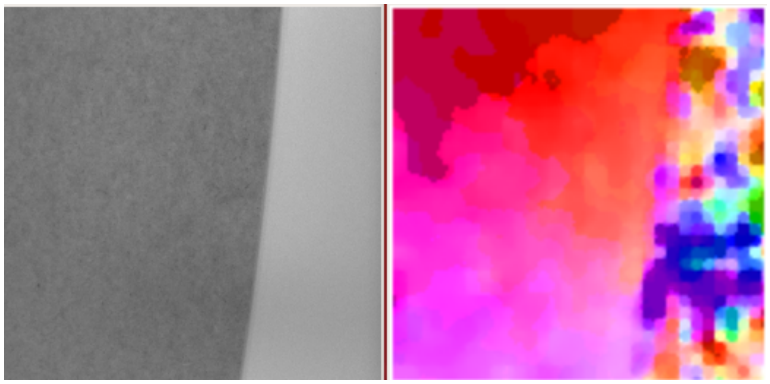


Figure 4.25: Flow output while preforming a rotation above table

While flying, the camera was alternating between observing the table, a wooden board mounted on top of the table and occasionally the floor. The wooden board was placed at the origin with the table and floor mainly entering around the edges of the frame. The Z-value used to disambiguate the velocity was the height above the table, this means that

in frames where the floor was also observable, the estimated velocity would be slightly smaller than the actual velocity.

It is clear from this test how part of the frame being featureless causes drift during rotations. At 35s the table is obscuring the edge of the frame as seen in fig. 4.25 causing a drift of up to 10 cm compared to the the vicon measurement.

In this test the high reflection of the table occasionally also caused the vicon measurement to be extra noisy which shows up as spikes in the velocity estimate.

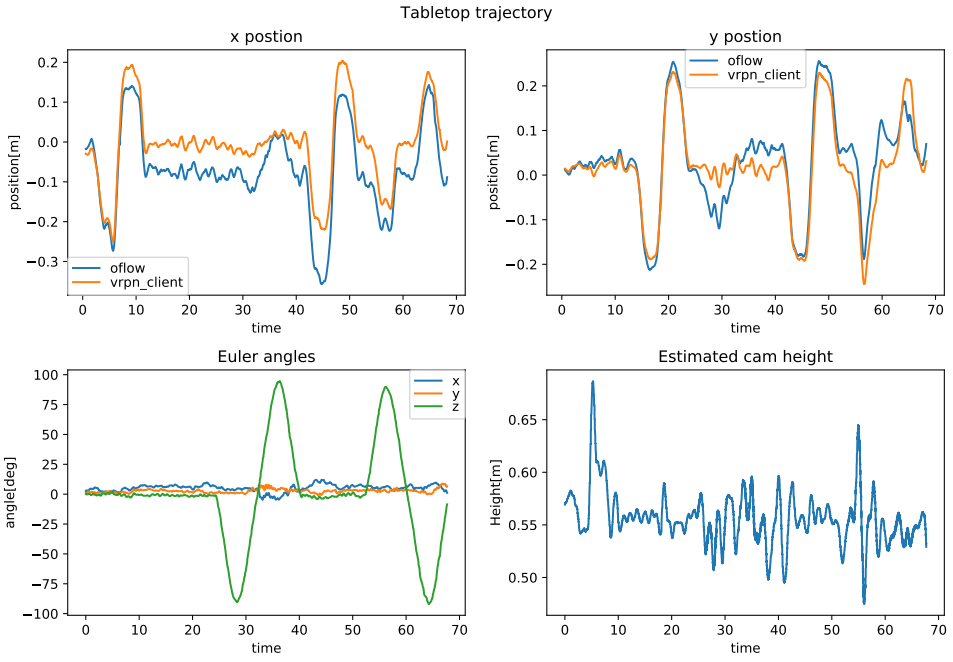


Figure 4.26: Tabletop trajectory

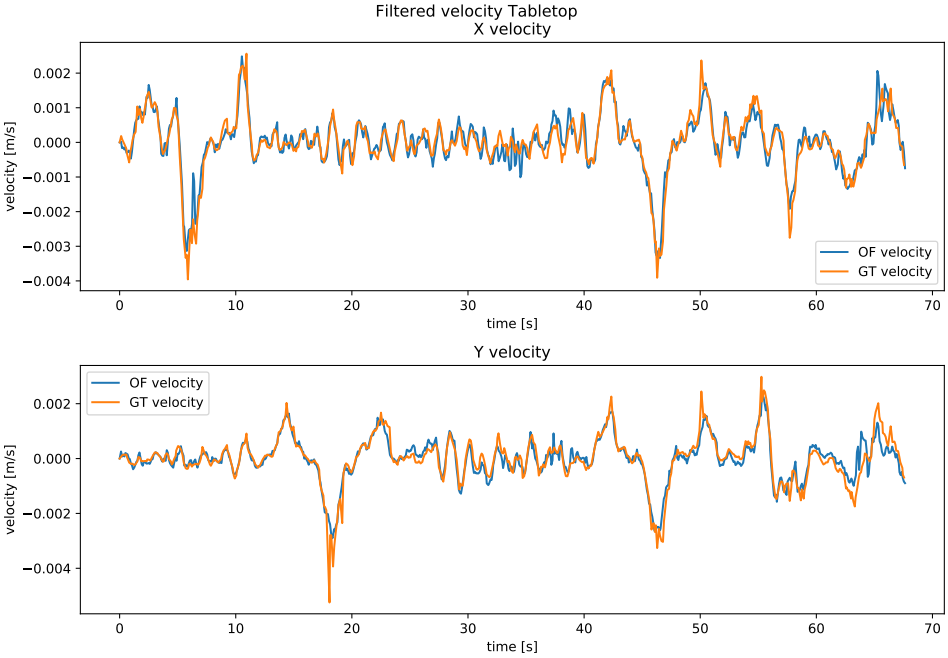


Figure 4.27: Tabletop trajectory

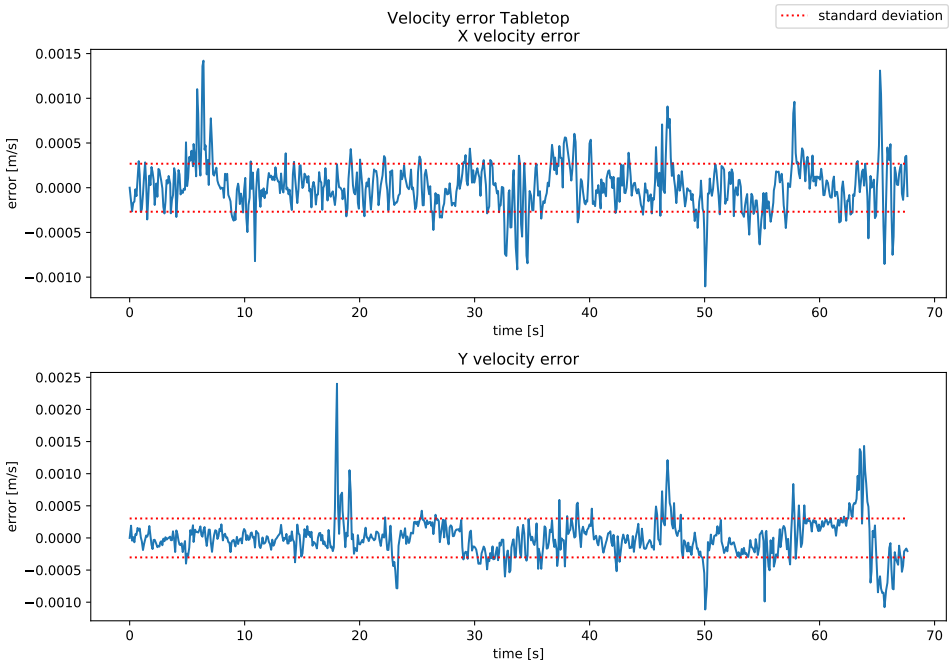


Figure 4.28: Tabletop trajectory

In this test it is clear that the estimation is not accurate during large rotations, this can be solved using a more robust mathematical model combined with outlier rejection. The mathematical should be easily implementable and only incur slight computational overhead. The outlier rejection is a more complicated problem as it must identify outliers very fast.

Chapter 5

Conclusion and future work

As seen in the results, the overall error of the optical flow is very small when compared with the ground truth. Having a standard deviation of only 0.2mm/s in operation during the expected use case achieved the project goals of high accuracy, high speed estimation. This is very promising moving forward and shows the capabilities of this contribution. However the overall assumption that the camera is facing perpendicular to the surface and the following simplifications to the movement model removes a significant degree of robustness from the system. This assumption can be easily remedied provided that the surface normal is aligned with the gravity vector. The gravity vector can then be estimated in the filter model and the offset between the camera axis and the surface normal can be calculated. However for a more general solution it would be necessary to estimate the surface normal given the optical flow. This should be possible to do using a recursive filter and the homographic constraint, while this was attempted during the project it was not finished due to time restraints. During the project, implementation of structure from motion was discussed, which would provide the ability to handle more complicated surfaces and circumvent the problem of estimating the surface vector all together, as this would estimate the depth of each pixel. However as the intended use case of the system is primarily on surfaces, this would cause unnecessary complications, calculating structure from motion would preform much of the same calculation as Rovio and the robustness of having an independent measurement would decrease if it is handled the same way as the existing odometry method.

A method to estimate the surface normal would be to use a distance sensor that can measure more than one dimension. In theory it would be sufficient to implement three measurements of the distance to the surface in order to estimate the normal, a more accurate solution is using ToF cameras capable of densely measuring distance in 2D. These 3D sensors are becoming increasingly lightweight and easy to use and would most be use full sensor for other reasons than just optical flow. If it were possible to mount and calibrate a depth sensor parallel to the camera and map the the depth pixel by pixel to the image frame it would provide the ability to very precisely estimate the velocity on an individual pixel basis. Regardless of the method used to estimate pixel distance, whether it is us-

ing the surface normal or individually measuring the distance to each pixel, using the full equation of motion is easily implementable once the distance is known.

Using outlier filtering on the resulting velocity would provide an additional degree of robustness. Filtering outlier velocities would be simpler than directly filtering the flow field. During rotations the estimated flow has values pointing in every direction while the estimated velocity for each pixel should be distributed around the true velocity of the drone. By calculating a standard deviation of the estimated velocities outliers can simply be omitted in one pass without the need for iterative solution.

To summarize the following steps should be taken to improve the velocity estimation:

- Estimate the distance to each pixel in the frame by:
  - Estimating the surface normal
  - Using a 3D sensor to measure the approximate pixel distance
- Calculate the velocity individually per pixel
- Filter outlier velocities and calculate mean

While the implementation of the optical flow in the system does not need much revision the ego motion estimation has several points that can still be improved. Overall the method showed great promise and performing tests on the system after the delta arm has been mounted and uses the optical flow estimation as a control input will be very exciting.

First and foremost I would like to thank my supervisors Karen Bodie and Michael Pantic, both for giving me the opportunity to work on this fascinating project and for their supervision and support throughout the whole project. I would also like to express my gratitude to the Autonomous Systems Lab for allowing me to complete my thesis at such an esteemed lab.
Lastly, I would like to express a special gratitude to Linnea Gidner for her support and understanding during the long days of home office during the project.

Bibliography

Baker, S., Matthews, I., 2004. Lucas-kanade 20 years on: A unifying framework. International journal of computer vision 56, 221–255.

Bay, H., Tuytelaars, T., Van Gool, L., 2006. Surf: Speeded up robust features, in: European conference on computer vision, Springer. pp. 404–417.

Bloesch, M., Burri, M., Omari, S., Hutter, M., Siegwart, R., 2017. Iterated extended kalman filter based visual-inertial odometry using direct photometric feedback. The International Journal of Robotics Research 36, 1053–1072. URL: https://doi.org/10.1177/0278364917728574, doi:10.1177/0278364917728574, arXiv:https://doi.org/10.1177/0278364917728574.

Bloesch, M., Omari, S., Hutter, M., Siegwart, R., 2015. Robust visual inertial odometry using a direct ekf-based approach, in: 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pp. 298–304. doi:10.1109/IROS.2015.7353389.

Bodie, K., Brunner, M., Pantic, M., Walser, S., Pfändler, P., Angst, U., Siegwart, R., Nieto, J., 2019. An omnidirectional aerial manipulation platform for contact-based inspection. arXiv:1905.03502.

Bodie, K., Taylor, Z., Kamel, M., Siegwart, R., 2018. Towards efficient full pose omnidirectionality with overactuated mavs. arXiv:1810.06258.

Butler, D.J., Wulff, J., Stanley, G.B., Black, M.J., 2012. A naturalistic open source movie for optical flow evaluation, in: ECCV.

Chhaniyara, S., Bunnun, P., Seneviratne, L., Althoefer, K., 2008. Optical flow algorithm for velocity estimation of ground vehicles: A feasibility study. International Journal on Smart Sensing and Intelligent Systems 1. doi:10.21307/ijssis-2017-289.

Forster, C., Carlone, L., Dellaert, F., Scaramuzza, D., 2017. On-manifold preintegration for real-time visual–inertial odometry. IEEE Transactions on Robotics 33, 1–21. URL: http://dx.doi.org/10.1109/TRO.2016.2597321, doi:10.1109/tro.2016.2597321.

Fortun, D., Bouthemy, P., Kervrann, C., 2015. Optical flow modeling and computation: A survey. Computer Vision and Image Understanding 134. doi:10.1016/j.cviu.2015.02.008.

Furgale, P., Rehder, J., Siegwart, R., 2013. Unified temporal and spatial calibration for multi-sensor systems, in: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 1280–1286.

Geiger, A., Lenz, P., Stiller, C., Urtasun, R., 2013. Vision meets robotics: The kitti dataset. The International Journal of Robotics Research 32, 1231–1237.

Geiger, A., Ziegler, J., Stiller, C., 2011. Stereoscan: Dense 3d reconstruction in real-time, in: 2011 IEEE Intelligent Vehicles Symposium (IV), pp. 963–968.

Godet, P., Boulch, A., Plyer, A., Besnerais, G.L., 2020. Starflow: A spatiotemporal recurrent cell for lightweight multi-frame optical flow estimation. arXiv:2007.05481.

Grabe, V., Bülthoff, H.H., Scaramuzza, D., Giordano, P.R., 2015. Nonlinear ego-motion estimation from optical flow for online control of a quadrotor uav. The International Journal of Robotics Research 34, 1114–1135.

Hauberg, S., Feragen, A., Black, M.J., 2014. Grassmann averages for scalable robust pca, in: 2014 IEEE Conference on Computer Vision and Pattern Recognition, pp. 3810–3817.

Ho, H.W., de Croon, G.C., Chu, Q., 2017. Distance and velocity estimation using optical flow from a monocular camera. International Journal of Micro Air Vehicles 9, 198–208. URL: https://doi.org/10.1177/1756829317695566, doi:10.1177/1756829317695566, arXiv:https://doi.org/10.1177/1756829317695566.

Honegger, D., Greisen, P., Meier, L., Tanskanen, P., Pollefeys, M., 2012. Real-time velocity estimation based on optical flow and disparity matching, in: 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems, pp. 5177–5182.

Hui, T.W., Tang, X., Loy, C.C., 2018. Liteflownet: A lightweight convolutional neural network for optical flow estimation. arXiv:1805.07036.

Ilg, E., Mayer, N., Saikia, T., Keuper, M., Dosovitskiy, A., Brox, T., 2016. Flownet 2.0: Evolution of optical flow estimation with deep networks. arXiv:1612.01925.

Kalman, R.E., 1960. A new approach to linear filtering and prediction problems .

Kamel, M., Verling, S., Elkhatib, O., Sprecher, C., Wulkop, P., Taylor, Z., Siegwart, R., Gilitschenski, I., 2018. The voliro omniorientational hexacopter: An agile and maneuverable tiltable-rotor aerial vehicle. IEEE Robotics & Automation Magazine 25, 34–44. URL: http://dx.doi.org/10.1109/MRA.2018.2866758, doi:10.1109/mra.2018.2866758.

Kroeger, T., Timofte, R., Dai, D., Gool, L.V., 2016. Fast optical flow using dense inverse search. arXiv:1603.03590.

Leutenegger, S., Furgale, P., Rabaud, V., Chli, M., Konolige, K., Siegwart, R., 2013. Keyframe-based visual-inertial slam using nonlinear optimization. doi:10.15607/RSS.2013.IX.037.

Lindeberg, T., 2012. Scale invariant feature transform .

Liu, C., 2009. Beyond Pixels: Exploring New Representations and Applications for Motion Analysis. Ph.D. thesis. Massachusetts Institute of Technology.

Lucas, B., Kanade, T., 1981. An iterative image registration technique with an application to stereo vision (ijcai).

Ma, Y., Soatto, S., Kosecka, J., Sastry, S.S., 2003. An Invitation to 3-D Vision: From Images to Geometric Models. SpringerVerlag.

Maurer, D., Marniok, N., Goldluecke, B., Bruhn, A., 2018. Structure-from-motion-aware patchmatch for adaptive optical flow estimation, in: Proceedings of the European Conference on Computer Vision (ECCV), pp. 565–581.

Otsu, K., Otsuki, M., Ishigami, G., Kubota, T., 2013. An Examination of Feature Detection for Real-Time Visual Odometry in Untextured Natural Terrain. volume 208. pp. 405–414. doi:10.1007/978-3-642-37374-9_39.

Qin, T., Li, P., Shen, S., 2018. Vins-mono: A robust and versatile monocular visual-inertial state estimator. IEEE Transactions on Robotics 34, 1004–1020. doi:10.1109/TRO.2018.2853729.

Raudies, F., 2013. Optic flow. Scholarpedia 8, 30724. doi:10.4249/scholarpedia.30724. revision #149632.

Rosten, E., Drummond, T., 2005. Fusing points and lines for high performance tracking, in: Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1, pp. 1508–1515 Vol. 2.

Ruffier, F., Franceschini, N., 2005. Optic flow regulation: The key to aircraft automatic guidance. Robotics and Autonomous Systems 50, 177–194. doi:10.1016/j.robot.2004.09.016.

Solà, J., 2017. Quaternion kinematics for the error-state kalman filter. arXiv:1711.02508.

Teed, Z., Deng, J., 2020. Raft: Recurrent all-pairs field transforms for optical flow .

Werlberger, M., Trobin, W., Pock, T., Wedel, A., Cremers, D., Bischof, H., 2009. Anisotropic huber-l1 optical flow. doi:10.5244/C.23.108.

Wulff, J., Black, M.J., 2015. Efficient sparse-to-dense optical flow estimation using a learned basis and layers, in: IEEE Conf. on Computer Vision and Pattern Recognition (CVPR 2015), pp. 120–130.

Zufferey, J.., Floreano, D., 2005. Toward 30-gram autonomous indoor aircraft: Vision-based obstacle avoidance and altitude control, in: Proceedings of the 2005 IEEE International Conference on Robotics and Automation, pp. 2594–2599. doi:10.1109/ROBOT.2005.1570504.

Appendix

Appendix A

- pos: [0.0, 0.0, 0.6], att: [0.0, 0.785, 0.0], force: [0.0, 0.0, 0.0], stop: True, time: 4.0
- pos: [0.2, 0.0, 0.6], att: [0.0, 0.785, 0.0], force: [0.0, 0.0, 0.0], stop: True, time: 4.0
- pos: [-0.2, 0.0, 0.6], att: [0.0, 0.785, 0.0], force: [0.0, 0.0, 0.0], stop: True, time: 4.0
- pos: [0.0, 0.0, 0.6], att: [0.0, 0.785, 0.0], force: [0.0, 0.0, 0.0], stop: True, time: 4.0
- pos: [0.0, 0.2, 0.6], att: [0.0, 0.785, 0.0], force: [0.0, 0.0, 0.0], stop: True, time: 4.0
- pos: [0.0, -0.2, 0.6], att: [0.0, 0.785, 0.0], force: [0.0, 0.0, 0.0], stop: True, time: 4.0
- pos: [0.0, 0.0, 0.6], att: [0.0, 0.785, 0.0], force: [0.0, 0.0, 0.0], stop: True, time: 4.0
- pos: [0.0, 0.0, 0.6], att: [1.57, 0.785, 0.0], force: [0.0, 0.0, 0.0], stop: True, time: 4.0
- pos: [0.0, 0.0, 0.6], att: [0.0, 0.785, 0.0], force: [0.0, 0.0, 0.0], stop: True, time: 4.0
- pos: [0.0, 0.0, 0.6], att: [-1.57, 0.785, 0.0], force: [0.0, 0.0, 0.0], stop: True, time: 4.0
- pos: [0.0, 0.0, 0.6], att: [0.0, 0.785, 0.0], force: [0.0, 0.0, 0.0], stop: True, time: 4.0
- pos: [0.2, 0.2, 0.6], att: [0.0, 0.785, 0.0], force: [0.0, 0.0, 0.0], stop: True, time: 4.0
- pos: [-0.2, -0.2, 0.6], att: [0.0, 0.785, 0.0], force: [0.0, 0.0, 0.0], stop: True, time: 4.0
- pos: [0.0, 0.0, 0.6], att: [0.0, 0.785, 0.0], force: [0.0, 0.0, 0.0], stop: True, time: 4.0
- pos: [0.2, 0.2, 0.6], att: [-1.57, 0.785, 0.0], force: [0.0, 0.0, 0.0], stop: False, time: 4.0
- pos: [0.0, 0.0, 0.6], att: [0.0, 0.785, 0.0], force: [0.0, 0.0, 0.0], stop: True, time: 4.0
- pos: [-0.2, -0.2, 0.6], att: [1.57, 0.785, 0.0], force: [0.0, 0.0, 0.0], stop: False, time: 4.0
- pos: [0.0, 0.0, 0.6], att: [0.0, 0.785, 0.0], force: [0.0, 0.0, 0.0], stop: True, time: 4.0

Appendix B

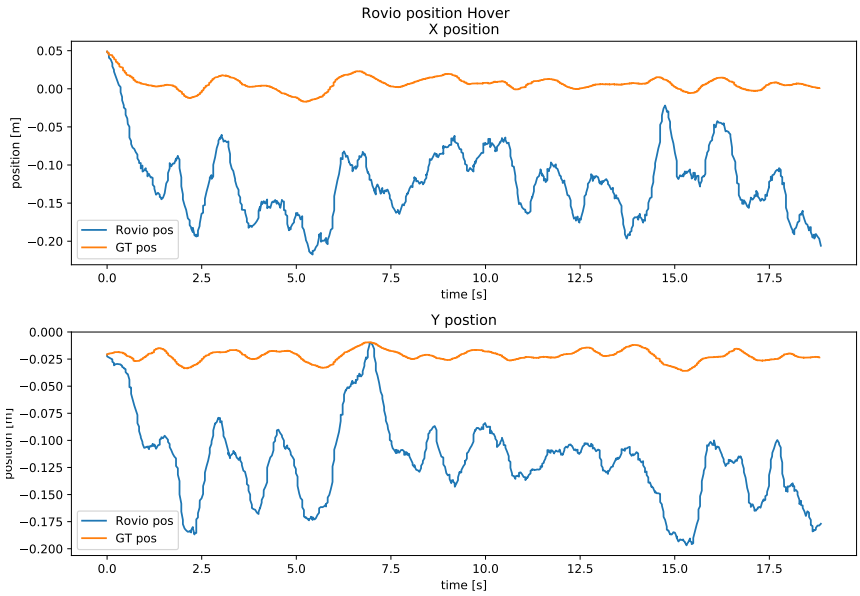


Figure 5.1: Rovio position compared to the ground truth during the hover test

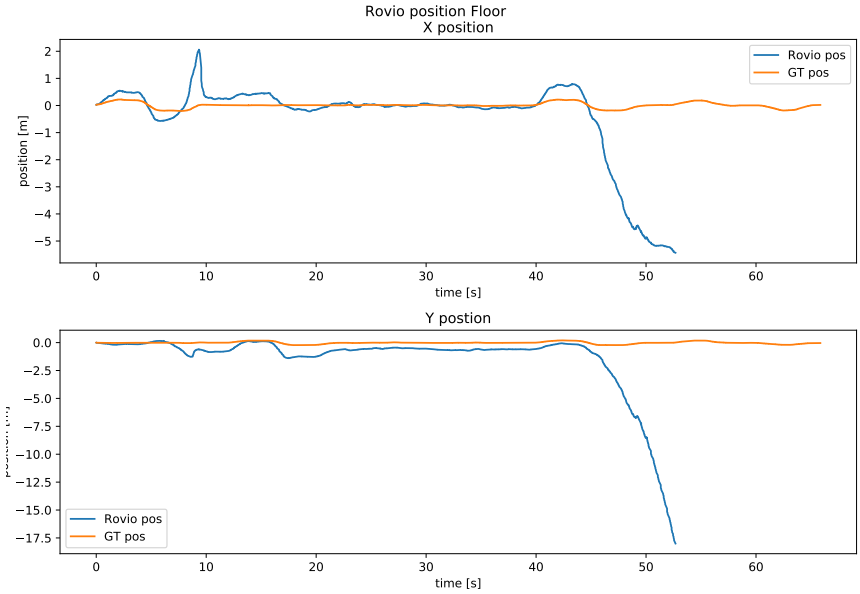


Figure 5.2: Rovio position compared to the ground truth during the trajectory test in the floor

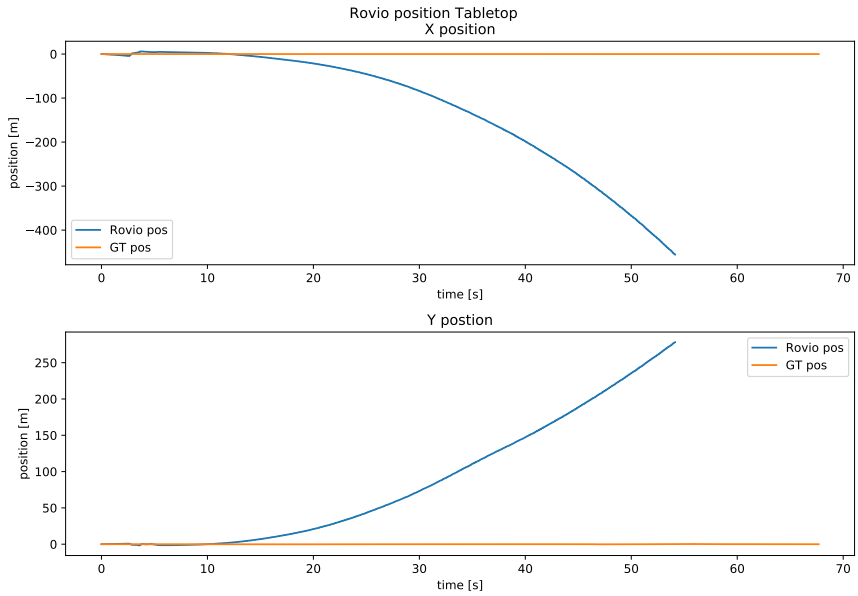
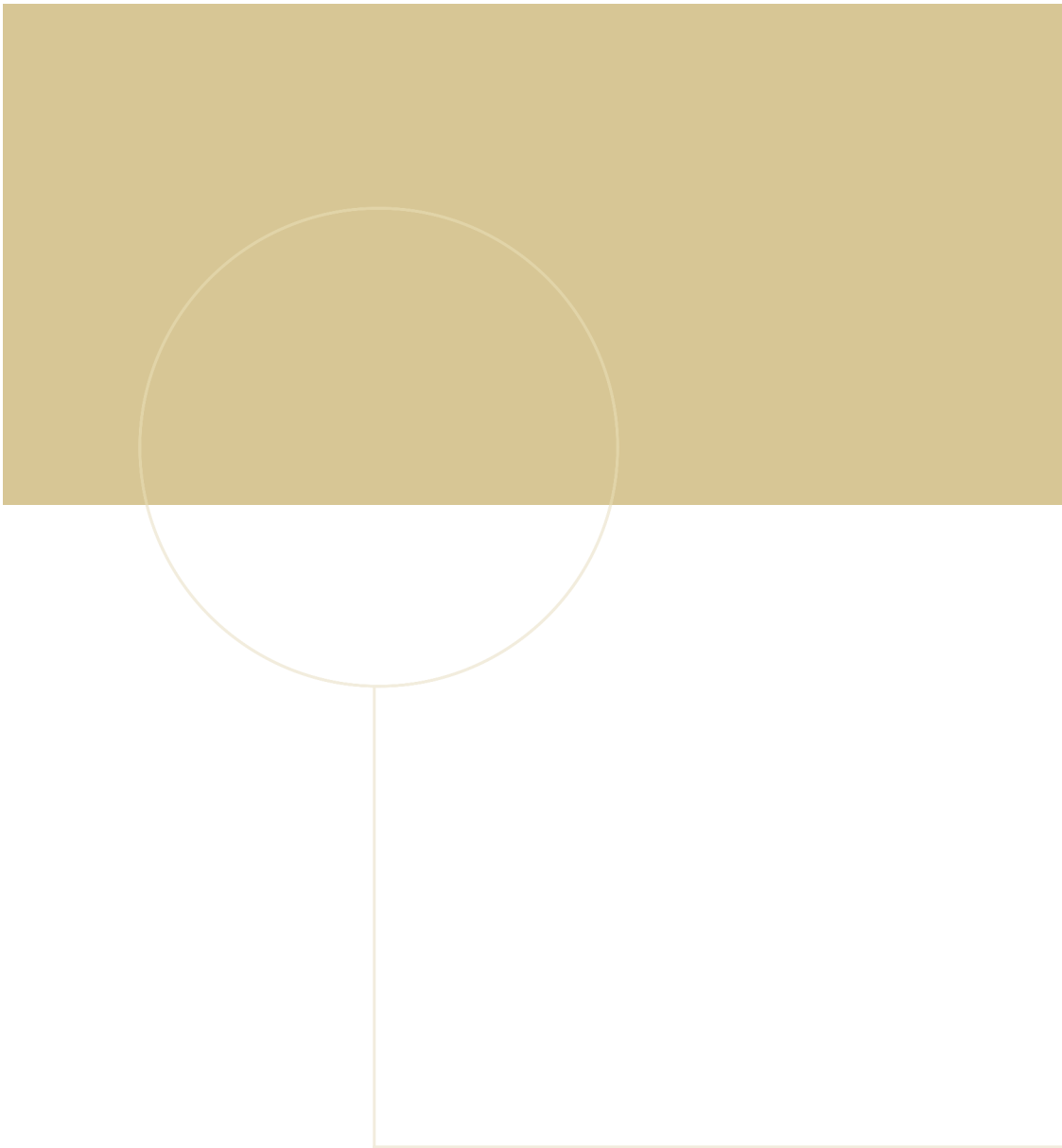


Figure 5.3: Rovio position compared to the ground truth during the trajectory test preformed over the table



NTNU

Kunnskap for en bedre verden