Precise localization of a UAV using visual odometry

T.H. (Tjark) Post

MSc Report

Committee:
Prof.dr.ir. S. Stramigioli
Dr.ir. M. Fumagalli
Dr.ir. F. van der Heijden
Dr. F.C. Nex

December 2015

037RAM2015
Robotics and Mechatronics
EE-Math-CS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Contents

Chapter 1

Introduction

Nowadays a lot of research is done on the topic of multirotor Unmanned Aerial Vehicles (UAV's). These UAV's can reach places which are hardly reachable for humans. At the University of Twente we are researching aerial manipulation. This research is part of the AEROWORKS project. The AEROWORKS project aims at researching and developing Aerial Robotic Workers (ARW's) who can collaboratively perform maintenance tasks. One of the use cases envisioned by AEROWORKS is the inspection and maintenance of lightning rods on windmills. These can corrode over time and need yearly inspection. This is currently done by shutting down the windmill and a man going up to do the inspection. AEROWORKS believes that this task can also be executed by ARW's. To achieve this different fields are further researched such as mapping, collaboration, manipulation and localization.

The field of operation of our currently researched aerial manipulation will be outdoors eventually. At the UT however we have no UAV's which are able to fly outdoors. The goal of this project is to setup a UAV which is capable of outdoor operation. This UAV can then function as a platform for further research on aerial manipulation.

A multirotor UAV from itself is a very unstable system. It requires proper high bandwidth control to stabilize the UAV. Therefore almost every UAV is equipped with an Inertial Measurement Unit (IMU). This IMU consists of a accelerometer and a gyroscope providing high bandwidth measurements of the linear acceleration and the angular velocity of the UAV. To obtain position and attitude (orientation) the accelerometer measurement has to be integrated twice and the gyroscope measurement has to be integrated once. A bias on the measurement readings will result in a drift on the position and attitude. This results in the UAV not being able to stabilize itself. The drift of the pitch and roll can be compensated using the gravity vector provided by the IMU. To compensate for the position drift extra position measurements have to be incorporated. These position measurements can be low bandwidth and have to be fused with the IMU measurements.

In the outdoor field GPS is often used as a position sensor. The position measurement provided by GPS however is accurate within a few meters. Performing

a maintenance task on for example a lightning rod requires a higher accuracy than GPS can provide. Also GPS is not always available which is an extra disadvantage. Other methods for position sensing have been subject of research. For example LIDAR, laser scanning, stereo vision or monocular vision can be used for position sensing. These sensors have to be carried onboard with the UAV. This makes that size and weight become important factors since it affects flight performance and flight duration. Also it is important that the position sensor can provide position measurements under a wide variety of circumstances. Stereo vision can provide a lot of information about the environment while the mass and size of cameras is small. It however requires a lot of computational power to extract this information from the camera images. But with the exponentially growing computation power of computers this is already feasible to run the necessary algorithms real time onboard of the UAV. Furthermore in the future the required computers will become more light-weight and also more computational expensive algorithms can be used to estimate the position from the camera images. Stereo cameras have the advantage over mono cameras that they provide an accurate depth map on a metric scale while with a mono camera the depth information requires a scaling factor to map it on a metric scale.

In this project the Skybotix VI-sensor is used. This VI-sensor consists of a stereo camera and an IMU. The stereo camera will be used to measure the pose of the VI-sensor. This pose will be fused with the IMU of the VI-sensor to obtain a high bandwidth pose measurement. The flight controller will use this pose measurement in its own sensor fusion to obtain an accurate high-bandwidth pose estimation which is necessary to stabilize and control the UAV.

First a literature study is performed to explore methods for localization based on vision and to explore sensor fusion methods. Then the chosen sensor fusion method is elaborated. This results in a high-bandwidth pose estimation from the VI-sensor. The chapter Realization describes how the setup is designed in which the VI-sensor is used as an external pose measurement. The chapter Experiments and discussion shows the experiments which have been performed to validate the performance of the setup and the performance will be discussed. Then in the chapter Conclusions and recommendations conclusions will be drawn and recommendations for further research will be made.

Chapter 2

Literature

This chapter describes the literature study. First the motion capture system and the VI-sensor are explained. Then the previous work in our group is shortly addressed. This is followed by a comparison of a SLAM and a VO+IMU system. Furthermore previous work of others on visual odometry and sensor fusion is described after which a choice is made what is going to be used in our setup.

2.1 External motion capture system

For localization of the UAV often a motion capture system like VICON or OptiTrack is used. This system consists of multiple cameras observing the flying area. Retroreflective markers are rigidly attached to the UAV and are detected by the cameras of the motion capture system. With multiple cameras observing the same marker the position of that marker can be measured with sub-millimeter accuracy. The system requires a calibration procedure before it can be used. At the University of Twente are two OptiTrack motion capture systems available for research.

Figure 2.1: UAVs flying using a motion capture system for localization

2.2 VI-sensor

For pose estimation based on vision the VI-sensor[8] is used. The VI-sensor combines two Aptina MT9V034 global shutter cameras with a ADIS 16488 IMU in a time-synchronized setup. The cameras provide 8 bits monochrome images with a resolution of 752x480 pixels at 20 Hz using the ROS interface.

The VI-sensor comes with a factory calibration containing camera calibration and inter sensor dimensions. The base line of the stereo camera is 11 cm.

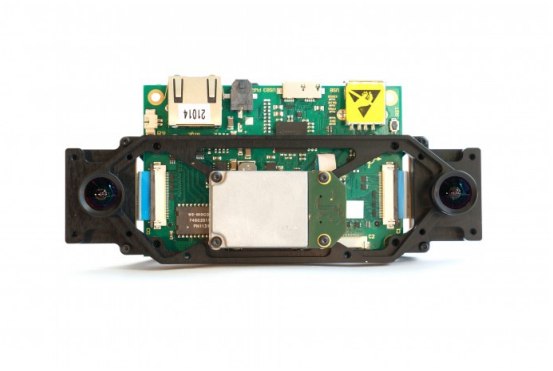

Figure 2.2: VI-sensor

2.3 EuRoC

This project is a follow-up of the EuRoC project at the University of Twente. The EuRoC project aimed at developing a platform for localization and mapping in an unknown environment with the help of two webcams and an IMU and lateron also a VI-sensor. An important recommendation from this project was to implement sensor fusion to increase the accuracy of the platform.

2.4 SLAM vs VO+IMU sensor fusion

Two approaches can be taken to solve the localization problem. The first solution is Simultaneous Localization And Mapping (SLAM). The SLAM algorithm keeps track of the pose of the observer and of the pose of the recognized landmarks. The other solution is sensor fusion of Visual Odometry and an IMU. This approach only keeps track of the pose of the observer and not the landmarks. This makes that the pose estimation of the VO+IMU sensor fusion approach can drift over time since estimation errors are accumulating and can't be corrected. The SLAM approach is not drifting since the drift of the observer can be observed by keeping track of the pose of the landmarks. A SLAM algorithm is therefore more complicated than a VO+IMU sensor fusion algorithm. SLAM also has a higher computational burden. The drift of the VO+IMU sensor fusion is not very big and can be corrected easily by the user. The disadvantage of the temporal drift does not weigh up to the advantage of easier implementation of the VO+IMU sensor fusion therefore this method will be chosen.

2.5 Visual odometry

Determining the motion of a camera by analysing the associated camera images is called visual odometry. Visual odometry algorithms determine the motion based on two consecutive images. The algorithm basically consists of the following steps:

1. Feature extraction
2. Feature matching
3. Motion estimation

The motion of the camera is calculated using only two consecutive frames so small errors in the estimation will accumulate and cause the estimate to drift. A pose estimation from visual odometry can be obtained by integrating the motion estimate. A comparison of different odometry methods for RGB-D cameras has been performed by Fang and Scherer[1]. They describe image based methods and depth based methods. Image based methods are found more faster and more accurate than depth based methods in feature rich environments. Since depth information from a stereo camera also relies on the number and quality of features it is better to look at the image based methods. Three image based methods are described of which only FOVIS[5] and VISO2[3] are applicable for monochrome stereo cameras. Both algorithms have a ROS implementation. The paper concludes that FOVIS is the best choice for accuracy and speed so FOVIS is chosen as visual odometry method.

2.6 Comparison of sensor fusion methods

The sensor fusion algorithm fuses visual odometry with IMU measurements. FOVIS provides a twist measurement including a covariance matrix. It can also provide a pose measurement which is the integration of the twist measurements. This pose measurement does not provide a covariance matrix. The IMU measurement consists of a linear acceleration and an angular velocity measurement. Since we are not aiming at improving the state of the art we are looking for already implemented packages. This resulted in three packages: Robot pose EKF[1], Robot localization[2] and ETHZ ASL Sensor fusion[3].

2.6.1 Comparison

Important properties of the sensor fusion is what it accepts as input measurement and which states are tracked. An overview of this can be seen in table 2.1.

2.6.2 Remarks

Not everything can be put in a table so a few remarks on the different sensor fusion algorithms are made here.

2.6.2.1 Robot pose EKF

The sensor fusion algorithm uses only the orientation data provided in a ROS IMU message. This means that an extra filter is needed to estimate the orientation of the IMU based on the IMU measurements.

[1] http://wiki.ros.org/robot_pose_ekf
[2] http://wiki.ros.org/robot_localization
[3] http://wiki.ros.org/ethzasl_sensor_fusion

	Robot pose EKF	Robot localization	ETHZ ASL Sensor fusion
Input:			
IMU	✓*	✓(multiple)	✓(one)
Pose sensor	✓	✓	✓
Twist sensor		✓	Only linear velocity
Tracked states:			
Position	✓	✓	✓
Linear velocity		✓	✓
Linear acceleration		✓	✓
Orientation	✓	✓	✓
Angular velocity		✓	✓
IMU bias			✓
Sensor-IMU configuration			✓
Pose scale			✓
Roll and pitch drift			✓

*Only accelerometer. Requires an extra filter to obtain attitude from IMU.

Table 2.1: Overview of the inputs and the tracked states of the different sensor fusion algorithms

2.6.2.2 Robot localization

This sensor fusion is set up very general. It has an Extended Kalman Filter and an Unscented Kalman Filter implementation. The input measurements have to match the states such that the observation matrix H is identity. The process noise can be tuned by the user but it is not clear how this should be done.

2.6.2.3 ETHZ ASL Sensor fusion

This sensor fusion is designed for use with Micro Air Vehicles (MAV). It is based on an Extended Kalman Filter. The IMU is used in the propagation step. This makes that the linear acceleration and the angular velocity states can not be updated by other sensors than the IMU. The sensor fusion framework compensates for time delay in the measurements. The sensor-IMU configuration and the pose scale are also estimated but those are already known on beforehand. The process noise covariance is already determined in this implementation.

2.6.3 Choice of sensor fusion implementation

Comparing the different sensor fusion implementations it is clear that Robot pose EKF is not good enough compared to the other two methods. Robot localization and ETHZ ASL Sensor fusion have both their pro's and cons but the ETHZ ASL Sensor fusion is chosen since it has more features and is already designed focused on use with MAV's.

Chapter 3

Sensor fusion with VI-sensor

This chapter details how the sensor fusion algorithm works. This sensor fusion algorithm is published by Weiss[12]. This sensor fusion algorithm has been implemented to work with the VI-sensor. The IMU measurements are fused with the pose estimation obtained by the visual odometry to obtain a high-bandwidth pose estimation from the VI-sensor. The sensor fusion is able to detect pitch and roll drift and can compensate for the time delay caused by the calculation time of the visual odometry. First the general functioning of the sensor fusion is elaborated. Then it will be described how the visual odometry pose has to be incorporated in the algorithm. The VI-sensor can also be used to create a map of the area. This will be described at the end of this chapter.

3.1 Software setup

Figure 3.1 shows the software setup which is used to obtain a high-bandwidth pose from the VI-sensor.

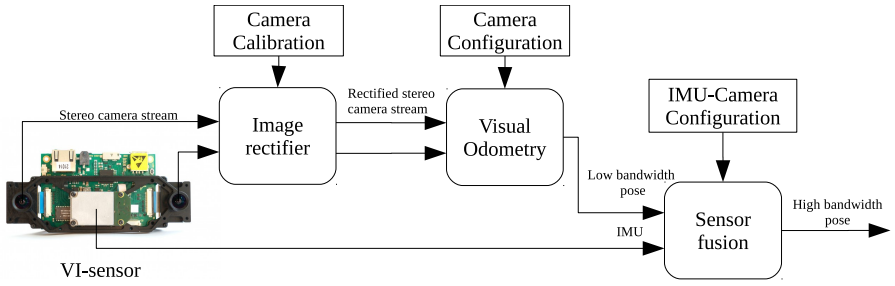

Figure 3.1: Software setup

The VI-sensor provides a stereo image stream. The image rectifier will remove distortion from the images using the provided camera calibration parameters which results in rectified camera images. The visual odometry block will

calculate the pose using the rectified camera images and the known transformation between both cameras. This pose is published at the frame rate of the camera which is 20 Hz in the case of the VI-sensor. The low bandwidth pose is fed to the sensor fusion together with the IMU data. The sensor fusion uses the known configuration of the IMU with respect to the camera to calculate a high bandwidth pose.

3.2 Sensor fusion

The sensor fusion algorithm used is an Error State Extended Kalman Filter (ESEKF) or Indirect Extended Kalman Filter. It uses three states: the true state $\mathbf{x}$ representing the physical state in which the vehicle is, the estimated state $\hat{\mathbf{x}}$ which is our estimation of the true state and the error state $\tilde{\mathbf{x}}$ which is the difference between the true state and the error state $\tilde{\mathbf{x}} = \mathbf{x} - \hat{\mathbf{x}}$. Ideally we want our estimated state to be the same as our true state so we want the error state to go to zero. The ESEKF measures the difference between our measurement and our estimation, which is the error, and uses this to calculate the corrections which has to be performed on the estimated state. This is elaborated more in the Kalman procedure. Recent work has been published which provides a very detailed description of the working of an ESEKF[10].

3.2.1 IMU as propagation sensor

The IMU is used as a propagation sensor in the sensor fusion. The bandwidth of the IMU is already high enough for the desired output bandwidth. Therefore it is unnecessary to compute extra prediction steps inbetween the IMU measurements. The IMU is affected by bias and noise. The bias is estimated in the filter and can therefore be compensated.

3.2.2 Coordinate frames

Four frames can be identified in the ESEKF model which can be seen in figure 3.2.

The world frame is always aligned with gravity. The IMU frame is the frame in which the IMU measurements are done. The camera frame is the origin of the reference camera. The vision frame is the world as observed by the cameras. The transformations between these frames are indicated using p and q. p_c^i describes the three-dimensional position of the origin of the camera frame of reference expressed in the IMU frame of reference. q_c^i is the quaternion representation of the orientation of the camera frame of reference expressed in the IMU frame of reference. The same holds for the other transformations where c describes the camera frame of reference, v describes the vision frame of reference, w describes the world frame of reference and i describes the IMU frame of reference.

The VI-sensor has an initial orientation in the filter. The visual odometry defines this initial orientation as a unity rotation. The filter however uses a different rotation. The initial rotation is known since it is defined by how it is mounted on the UAV (assumed the UAV is leveled on initialization). This

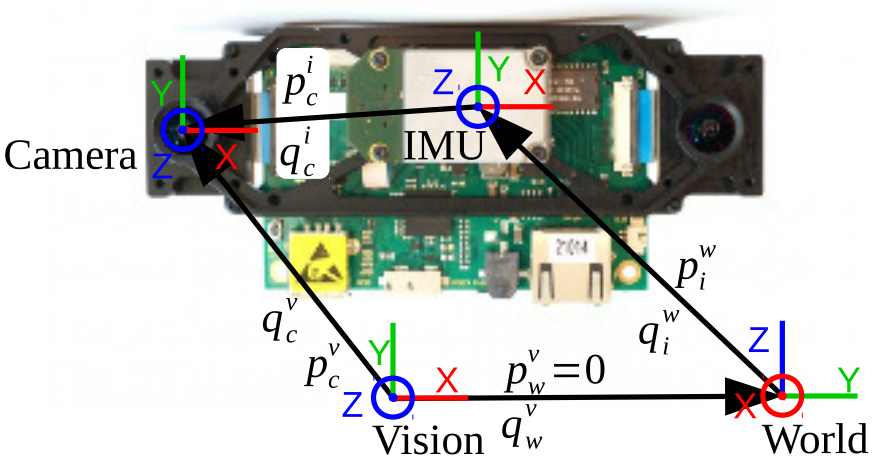

Figure 3.2: Model used in EKF

initial rotation is incorporated in the rotation of the vision frame with respect to the world frame.

3.2.3 Drift observability

Since the pose measurements from visual odometry can drift the vision frame is defined separately from the world frame. The position drift of the visual odometry is unobservable in this setup. By forcing $p_w^v = 0$ the position of the vision frame equals the position of the world frame which means that the world frame is drifting in position along with the vision frame.

A drift in roll or pitch is observable using the gravity vector which can be obtained from the accelerometer. Therefore the rotation of the world frame in the vision frame can become non-zero. This non-zero rotation represents the roll and pitch drift of the visual odometry attitude measurement. A yaw drift is still unobservable. Therefore the yaw of the world frame is aligned with the yaw of the vision frame. This is done and further detailed in the update step of the ESEKF.

3.2.4 Kalman procedure

This section will detail the Kalman procedure which is used. A Kalman procedure consists of two steps: a propagation step and an update step. The IMU measurements are used in the propagation step and the visual odometry pose measurement will be used in the update step.

3.2.4.1 State vector

The state vector of the ESEKF is:

$$\hat{\mathbf{x}} = \begin{bmatrix} p_i^w & v_i^w & q_i^w & b_\omega & b_a & \lambda & q_w^v & q_c^i & p_c^i \end{bmatrix}^T \tag{3.1}$$

with p_i^w and v_i^w describing respectively the three-dimensional position and velocity and q_i^w describing the quaternion representation of the orientation of the IMU frame expressed in the world frame. The three-dimensional vectors b_ω and b_a describe the bias terms on respectively the gyroscope and the accelerometer. The scalar λ describes the scaling factor which is needed when scaled position measurements are done for example when using a mono camera setup. The quaternion q_w^v describes the orientation of the world frame expressed in the vision frame and the quaternion q_c^i and the three-dimensional position p_c^i describe respectively the orientation and the position of the camera expressed in the IMU frame.

The error state is defined as:

$$\tilde{\mathbf{x}} = \begin{bmatrix} \tilde{p}_i^w & \tilde{v}_i^w & \delta\theta_i^w & \tilde{b}_\omega & \tilde{b}_a & \tilde{\lambda} & \delta\theta_w^v & \delta\theta_c^i & \tilde{p}_c^i \end{bmatrix}^T \tag{3.2}$$

where $\tilde{p}_i^w$ represents the difference between the real position p_i^w and the estimated position $\hat{p}_i^w$: $\tilde{p}_i^w = p_i^w - \hat{p}_i^w$. Similarly for velocity, scale and bias terms. The error in the orientation cannot be expressed by simple subtraction. The error in the orientation can be described by introducing the error quaternion:

$$\delta q_i^w = \hat{q}_i^{w-1} \otimes q_i^w \tag{3.3}$$

The $\otimes$ operator is the quaternion multiplication operator. This equation describes the same as:

$$\delta R_i^w = \hat{R}_i^{w-1} R_i^w \tag{3.4}$$

In the working area of the ESEKF the estimation of the orientation will be close to the real value of the estimation. This means that the error quaternion δq is close to identity. This allows to use a small angle approximation of the error quaternion:

$$\delta R_i^w = I + \lfloor \delta\theta_i^w \rfloor_\times \tag{3.5}$$

In which $\lfloor x \rfloor_\times$ is the skew-symmetric matrix composed from the vector x. The relationship between δq and $\delta\theta$ is given by:

$$\delta q \approx \begin{bmatrix} 1 & \frac{1}{2}\delta\theta_x & \frac{1}{2}\delta\theta_y & \frac{1}{2}\delta\theta_z \end{bmatrix}^T = \begin{bmatrix} 1 & \frac{1}{2}\delta\theta \end{bmatrix}^T \tag{3.6}$$

The small angle approximation of the error allows to express the error using three terms in stead of four. This small angle approximation $\delta\theta$ is used in the error state.

3.2.4.2 Propagation step

During the propagation step the states of the ESEKF are propagated using the IMU measurements. For every IMU measurement the following steps are performed:

1. Propagate the state variables according to their differential equations. For quaternions the first order quaternion integration is used[11].
2. Calculate F_d and Q_d which describe the propagation of the error state covariance matrix and the additional measurement noise.
3. Compute the propagated error state covariance matrix according to

$$P_{k+1|k} = F_d P_{k|k} F_d^T + Q_d \tag{3.7}$$

3.2.4.2.1 Propagate using differential equations

The differential equations used to propagate the state variables are:

$$\begin{aligned}
\hat{\dot{p}}_i^w &= \hat{v}_i^w \\
\hat{\dot{v}}_i^w &= R_i^w(a_m - \hat{b}_a) - g \\
\hat{\dot{q}}_i^w &= \frac{1}{2}\Omega(\omega_m - \hat{b}_\omega)\hat{q}_i^w \\
\hat{\dot{b}}_\omega &= 0, \hat{\dot{b}}_a = 0, \hat{\dot{\lambda}} = 0 \\
\hat{\dot{p}}_c^i &= 0, \hat{\dot{q}}_c^i = 0, \hat{\dot{p}}_w^v = 0, \hat{\dot{q}}_w^v = 0
\end{aligned}$$

where a_m and ω_m are respectively the measurements of the accelerometer and the gyroscope and $\Omega(\omega)$ is the quaternion-multiplication matrix of ω. It is constructed as:

$$\Omega(\omega) = \begin{bmatrix} -\omega_x & -\omega_y & -\omega_z & 0 \\ 0 & \omega_z & -\omega_y & \omega_x \\ -\omega_z & 0 & \omega_x & \omega_y \\ \omega_y & -\omega_x & 0 & \omega_z \end{bmatrix} \tag{3.8}$$

when the quaternion is defined as $q = \begin{bmatrix} q_w & q_x & q_y & q_z \end{bmatrix}^T$

The position and velocity are updated according to

$$\begin{aligned}
\hat{p}_{i_{k+1|k}}^w &= \hat{p}_{i_{k|k}}^w + \hat{\dot{p}}_i^w \Delta t \\
\hat{v}_{i_{k+1|k}}^w &= \hat{v}_{i_{k|k}}^w + \hat{\dot{v}}_i^w \Delta t
\end{aligned}$$

For the quaternion a first order quaternion integrator is used to integrate the quaternions[11].

3.2.4.2.2 Calculate propagation of error state covariance matrix and additional measurement noise

To calculate the propagation of the error state covariance matrix and the additional measurement first the dynamics of the error state are needed[12]. The error state dynamics in continuous time are:

$$
\begin{aligned}
\Delta\dot{p}_i^w &= \Delta v_i^w \\
\Delta\dot{v}_i^w &= -R_i^w \lfloor a_m - \hat{b_a} \rfloor_\times \delta\theta_i^w - R_i^w \Delta b_a - R_i^w n_a \\
\delta\dot{\theta}_i^w &= -\lfloor \omega_m - \hat{b_\omega} \rfloor_\times \delta\theta_i^w - \Delta b_\omega - n_\omega \\
\Delta\dot{b}_\omega &= n_{b_\omega} \quad \Delta\dot{b}_a = n_{b_a} \quad \Delta\dot{\lambda} = 0 \\
\Delta\dot{p}_c^i &= 0 \quad \delta\dot{q}_c^i = 0 \quad \Delta\dot{p}_w^v = 0 \quad \delta\dot{q}_w^v = 0
\end{aligned}
$$

The error state dynamics can be described in the linearized continuous time error state equation

$$\dot{\tilde{x}} = F_c \tilde{x} + G_c n \tag{3.9}$$

with n being the noise vector $n = \left[n_a^T, n_{b_a}^T, n_\omega^T, n_{b_\omega}^T\right]^T$. The continuous-time error state covariance matrix F_c can be discretized:

$$F_d = exp(F_c \Delta t) = I_d + F_c \Delta t + \frac{1}{2!} F_c^2 (\Delta t)^2 + \dots \tag{3.10}$$

A repetitive and sparse structure can be found in the matrix exponents which allows to express F_d exactly as

$$F_d = \begin{bmatrix}
I_3 & \Delta t & A & B & -\hat{R}_i^w \frac{(\Delta t)^2}{2} & 0_{3\times 13} \\
0_3 & I_3 & C & D & -\hat{R}_i^w \Delta t & 0_{3\times 13} \\
0_3 & 0_3 & E & F & 0_3 & 0_{3\times 13} \\
0_3 & 0_3 & 0_3 & I_3 & 0_3 & 0_{3\times 13} \\
0_3 & 0_3 & 0_3 & 0_3 & I_3 & 0_{3\times 13} \\
0_{13\times 3} & 0_{13\times 3} & 0_{13\times 3} & 0_{13\times 3} & 0_{13\times 3} & I_{13}
\end{bmatrix} \tag{3.11}$$

with

$$A = -\hat{R}_i^w \lfloor \hat{a} \rfloor_\times \left(\frac{(\Delta t)^2}{2} - \frac{(\Delta t)^3}{3!} \lfloor \omega \rfloor_\times + \frac{(\Delta t)^4}{4!} (\lfloor \omega \rfloor_\times)^2 \right) \tag{3.12}$$

$$B = -\hat{R}_i^w \lfloor \hat{a} \rfloor_\times \left(\frac{-(\Delta t)^3}{3!} + \frac{(\Delta t)^4}{4!} \lfloor \omega \rfloor_\times - \frac{(\Delta t)^5}{5!} (\lfloor \omega \rfloor_\times)^2 \right) \tag{3.13}$$

$$C = -\hat{R}_i^w \lfloor \hat{a} \rfloor_\times \left(\Delta t - \frac{(\Delta t)^2}{2} \lfloor \omega \rfloor_\times + \frac{(\Delta t)^3}{3!} (\lfloor \omega \rfloor_\times)^2 \right) \tag{3.14}$$

$$D = -A \tag{3.15}$$

$$E = I_3 - \Delta t \lfloor \omega \rfloor_\times + \frac{(\Delta t)^2}{2!} (\lfloor \omega \rfloor_\times)^2 \tag{3.16}$$

$$F = -\Delta t + \frac{(\Delta t)^2}{2!} \lfloor \omega \rfloor_\times - \frac{(\Delta t)^3}{3!} (\lfloor \omega \rfloor_\times)^2 \tag{3.17}$$

The matrix G_c describes the propagation of the noise. It can be written as

$$G_c = \begin{bmatrix} 0_3 & 0_3 & 0_3 & 0_3 \\ -\hat{R}_i^w & 0_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & -I_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & I_3 \\ 0_3 & I_3 & 0_3 & 0_3 \\ 0_{13\times3} & 0_{13\times3} & 0_{13\times3} & 0_{13\times3} \end{bmatrix} \tag{3.18}$$

The discrete time system noise covariance matrix Q_d can now be calculated as[6]:

$$Q_d = \int_{\Delta t} F_d(\tau) G_c Q_c G_c^T F_d(\tau)^T d\tau \tag{3.19}$$

with

$$Q_c = \begin{bmatrix} \sigma_{n_a}^2 & 0_3 & 0_3 & 0_3 \\ 0_3 & \sigma_{n_{b_a}}^2 & 0_3 & 0_3 \\ 0_3 & 0_3 & \sigma_{n_\omega}^2 & 0_3 \\ 0_3 & 0_3 & 0_3 & \sigma_{n_{b_\omega}}^2 \end{bmatrix} \tag{3.20}$$

For the VI-sensor IMU, ADIS 16488, this matrix is

$$Q_c = \begin{bmatrix} 0.022563^2 \cdot I_3 & 0_3 & 0_3 & 0_3 \\ 0_3 & 0.00008^2 \cdot I_3 & 0_3 & 0_3 \\ 0_3 & 0_3 & 0.004^2 \cdot I_3 & 0_3 \\ 0_3 & 0_3 & 0_3 & 0.000003^2 \cdot I_3 \end{bmatrix} \tag{3.21}$$

3.2.4.3 Measurement update

With only the propagation the error state covariance matrix will continue to grow i.e. the estimation becomes more uncertain. A measurement from another sensor can provide extra information to update the estimate and reduce the uncertainty. The Kalman procedure for the measurement update is as follows:

1. Compute the residual $\tilde{z} = z - \hat{z}$
2. Compute the innovation $S = HPH^T + R$
3. Compute the Kalman gain $K = PH^T S^{-1}$
4. Compute the correction $\hat{\tilde{x}} = K\tilde{z}$
5. Update states $\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + \hat{\tilde{x}}$
6. Update error state covariance matrix
   $P_{k+1|k+1} = (I_d - KH)P_{k+1|k}(I_d - KH)^T + KRK^T$

in which z is the performed measurement, $\hat{z}$ is the estimated measurement based on the estimated states, H is the observation matrix relating the measurement error $\tilde{z}$ with the error state $\tilde{x}$ as $\tilde{z} = H\tilde{x}$. P is the error state covariance matrix and R is the measurement noise covariance matrix.

To process the measurement the residual $\tilde{z}$, the observation matrix H and the measurement noise covariance matrix R have to be determined. The visual odometry pose is used as a measurement. This measurement can be split in three parts: the position, the attitude and an artificial yaw measurement. The artificial yaw measurement is included since the yaw drift is unobservable. By including an artificial yaw measurement it can be assured that the yaw of the vision frame is aligned with the yaw of the world frame. The three parts of the pose measurement can be computed separately and at the end combined in one observation matrix, one residual and one measurement noise covariance matrix.

3.2.4.3.1 Position measurement

For the position measurement the measurement function is:

$$h_p(\tilde{x}) = \tilde{z}_p = R_w^v(p_i^w + R_i^w p_c^i)\lambda + n_p - \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda} \tag{3.22}$$

To obtain the observation matrix H the measurement function has to be linearized around the estimated state:

$$H_p = \frac{\partial h_p(\tilde{x})}{\partial \tilde{x}}|_{x=\hat{x}} \tag{3.23}$$

This results in the following observation matrix:

$$H_p = \begin{bmatrix} \hat{R}_w^v\hat{\lambda} \\ \mathbf{0_{3x3}} \\ -\hat{R}_w^v\hat{R}_i^w\lfloor\hat{p}_c^i\times\rfloor\hat{\lambda} \\ \mathbf{0_{3x3}} \\ \mathbf{0_{3x3}} \\ \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w\hat{p}_c^i) \\ -\hat{R}_w^v\lfloor(\hat{p}_i^w + \hat{R}_i^w\hat{p}_c^i)\hat{\lambda}\times\rfloor \\ \mathbf{0_{3x3}} \\ \hat{R}_w^v\hat{R}_i^w\hat{\lambda} \end{bmatrix}^T \tag{3.24}$$

How to derive this observation matrix is detailed in the appendix. The measurement noise covariance matrix was not given by the visual odometry so this is estimated at:

$$R = \begin{bmatrix} 0.01 & 0 & 0 \\ 0 & 0.01 & 0 \\ 0 & 0 & 0.01 \end{bmatrix} \tag{3.25}$$

The residual is

$$\tilde{z}_p = z_p - \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w\hat{p}_c^i)\hat{\lambda} \tag{3.26}$$

3.2.4.3.2 Attitude measurement

For the attitude measurement the measurement function is:

$$h_q(\tilde{x}) = \tilde{z}_q = (\hat{q}_c^v)^{-1} \otimes q_c^v = (\hat{q}_c^i)^{-1} \otimes (\hat{q}_i^w)^{-1} \otimes (\hat{q}_w^v)^{-1} \otimes q_w^v \otimes q_i^w \otimes q_c^i \quad (3.27)$$

Linearizing this function around the estimated state results in the following observation matrix:

$$H_q = \begin{bmatrix} \mathbf{0_{3x3}} \\ \mathbf{0_{3x3}} \\ \frac{1}{2}R_i^c \\ \mathbf{0_{3x3}} \\ \mathbf{0_{3x3}} \\ 0 \\ \frac{1}{2}R_i^c R_w^i \\ \frac{1}{2}\mathbf{I_3} \\ \mathbf{0_{3x3}} \end{bmatrix}^T \quad (3.28)$$

The measurement covariance matrix R is also not given for the attitude so it is also estimated. The estimation used is:

$$R = \begin{bmatrix} 0.02 & 0 & 0 \\ 0 & 0.02 & 0 \\ 0 & 0 & 0.02 \end{bmatrix} \quad (3.29)$$

The residual of the attitude measurement is:

$$\tilde{z}_q = (\hat{q}_c^v)^{-1} \otimes z_q \quad (3.30)$$

3.2.4.3.3 Artificial yaw measurement

The artificial yaw measurement is to align the yaw of the vision frame with the yaw of the world frame. This can be achieved by using an observation scalar $H = 1$, a measurement noise covariance of $1 \cdot 10^{-6}$ and a residual

$$y = yaw((\hat{q}_w^v)^{-1} \otimes q_w^v) \quad (3.31)$$

where the yaw of a quaternion is calculated as

$$yaw(q) = atan2(2(q_w q_z + q_x q_y), (1 - 2(q_y^2 + q_z^2))) \quad (3.32)$$

3.2.4.4 Complete measurement

The three parts can be combined together. This will result for the observation matrix in

$$\begin{bmatrix} \tilde{z}_p \\ \tilde{z}_q \\ \tilde{z}_{yaw} \end{bmatrix} = \begin{bmatrix} H_p \\ H_q \\ 1 \end{bmatrix} \tilde{x} \quad (3.33)$$

The total measurement error covariance matrix is

$$R = \begin{bmatrix} 0.01 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0.01 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0.01 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0.02 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.02 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0.02 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \cdot 10^{-6} \end{bmatrix} \tag{3.34}$$

And the total residual is

$$\tilde{z} = \begin{bmatrix} z_p - \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda} \\ (\hat{q}_c^v)^{-1} \otimes z_q \\ yaw((\hat{q}_w^v)^{-1} \otimes q_w^v) \end{bmatrix} \tag{3.35}$$

3.2.5 Fixed and known configuration

The EKF was designed for configurations in which the scale and the camera-IMU configuration is not known on beforehand. That means that those values are incorporated in the state vector. In our configuration however the camera-IMU configuration is known since the stereo cameras were calibrated in the factory and this data is provided with the cameras. Also the scale is known and fixed on 1 since a stereo camera is used. The states p_c^i, $\delta\theta_c^i$ and λ could be removed from the state vector since their values do not need to be estimated. But the software implementation also allows to fix the scale and the configuration. In addition the initial error state covariance matrix has to be put to zero at the rows and columns corresponding to the known states. This makes the filter know that these states are fully known.

3.2.6 Lost visual odometry

It can happen that the visual odometry algorithm is not able to estimate the pose. This can be caused for example when it can't find enough features in the image. The algorithm will produce a faulty measurement which can be recognized by all zeros. In such a situation the measurement should not be taken into account in the sensor fusion. This can be done by setting the H-matrix and the residual to zero. The state estimation of the sensor fusion will now only be determined based on the IMU.

When the visual odometry has recovered it will continue to provide measurement data. The motion estimation is again valid but the pose estimation, which is an integration of the motion estimation, will continue from the pose it had before the visual odometry was lost. This is not correct. It should start from the pose estimation given by the sensor fusion at the moment of recovery. This error can be compensated by introducing a bias on the pose. This bias will be calculated at the moment of recovery.

$$p_{bias} = \hat{z}_p - z_p$$
$$q_{bias} = \hat{z}_q \otimes z_q^{-1}$$

Where $\hat{z}$ is the expected measurement value using the estimated states of the sensor fusion and z is the obtained measurement value. The calculated bias should now be added to every new measurement value.

$$z_p = p_{bias} + z_p$$
$$z_q = q_{bias} \otimes z_q$$

Next to an update of the measurement using a bias term the uncertainty of the measurement value has also increased.

3.2.7 Time delay compensation

A useful property of the ESEKF implementation of Weiss[12] is that it allows for time delay compensation. The visual odometry algorithm which extracts the pose from the camera images takes some time and meanwhile the IMU has already propagated further. Timestamps of the camera images and the IMU are used to update the ESEKF at the time where the measurement took place. Then the state is further propagated from the time where the measurement took place using buffered IMU-measurements. This is shown in figure 3.3.

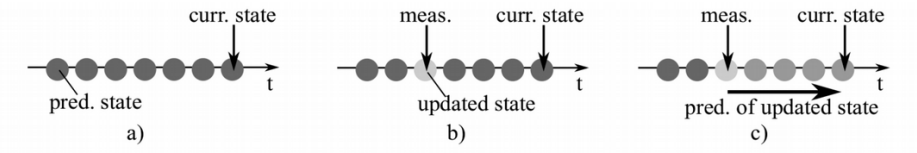

Figure 3.3: Time delay compensation in the ESEKF

In a) the UAV is using the current state as reference. In b) a delayed measurement update arrives and the corresponding buffered state is corrected using the measurement. In c) the states following the corrected states are corrected by propagating the buffered IMU measurements using the propagation step of the ESEKF.

3.3 Mapping

A map is also created during the flight. To create a map a point cloud and a corresponding pose estimate are needed. The point cloud can be transformed in to the world frame using the estimated pose of the UAV. The point cloud is generated using Stereo Block matching on the stereo camera images. The map is build using an octomap[4] structure to reduce memory load. In the chapter Experiments and discussion a map built with this method is shown.

Chapter 4

Realisation

Chapter 3 showed the theoretical approach toward the use of the VI-sensor as a high bandwidth pose sensor. The next step is to set up a system in which the external pose estimation can be used to control a UAV. This chapter first describes how the UAV is setup. Then the setup used to control the UAV is elaborated. Since the first test flights are performed inside an OptiTrack area a fail-safe has been implemented which switches back to OptiTrack in case visual odometry fails. This will described in the end of this chapter.

4.1 System setup

The UAV used is the ArduCopter Hexa-B. Six AC2830-358 850Kv motors are mounted on the frame. The ESCs are from 3D Robotics and can deliver 20 A of current and 6-16.8 V DC voltage. A FPV Radio Telemetry Ground Module is also mounted which allows wireless Mavlink communication. The flight controller used is a Pixhawk.

Mounted on the UAV are the VI-sensor and an Intel NUC 5i5RYK computer. The computer is running Ubuntu 14.04 LTS. ROS (Robot Operating System) Jade is used as framework for the software which is used. Communication between the NUC and the Pixhawk is done using the Mavlink protocol. Since the NUC and the Pixhawk are both mounted on the drone the communication link is a USB cable. The whole system is designed such that it can operate on a LiPo 4S battery. Further details about the design can be found in the appendix. A picture of the UAV can be seen in figure 4.1.

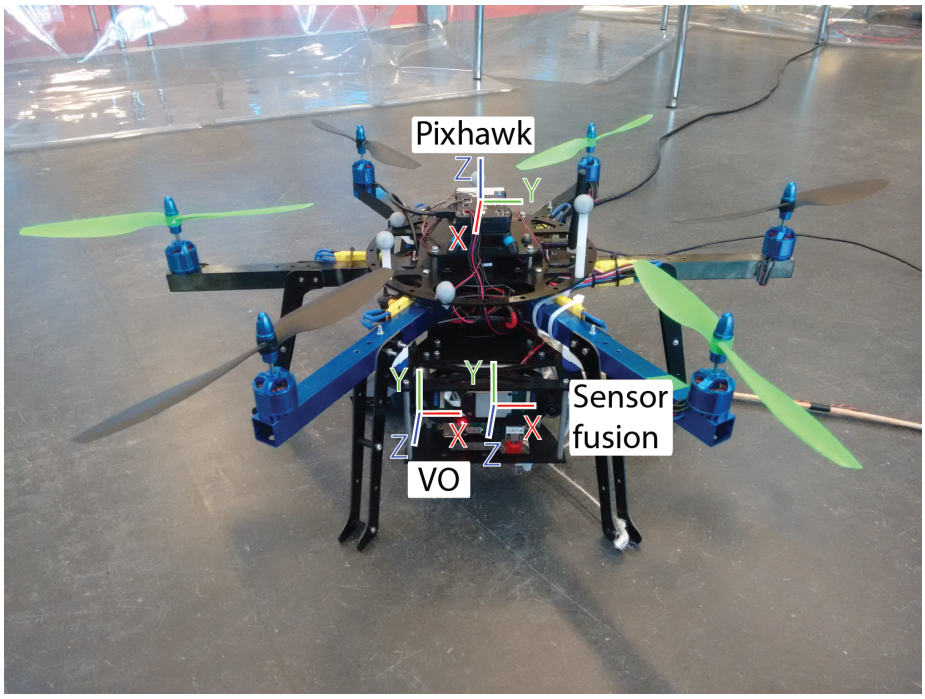

Figure 4.1: Hexacopter

The picture shows the UAV. On top and in the center of the UAV the Pixhawk flight controller is located. The grey balls are the OptiTrack markers used for the OptiTrack pose estimation. The blue arms of the UAV show the front of the UAV. At the front the VI-sensor is mounted. At the back the Intel NUC is mounted. Both units weigh approximately the same and are placed such that the center of mass of the UAV stays in the center of the UAV.

In this picture also relevant frames are shown. The visual odometry (VO) frame is located in the reference camera and is the frame in which the visual odometry algorithm expresses its pose. The sensor fusion frame is located in the IMU of the VI-sensor and is the frame in which the sensor fusion expresses its pose. The Pixhawk frame is located in the IMU of the Pixhawk and is the frame in which the Pixhawk expresses its pose. Based on this pose the UAV is controlled. The OptiTrack pose is not shown for clarity of the image but the frame is in approximately the same position and orientation as the Pixhawk frame.

4.2 Control setup

An external controller is used to control the UAV. This means that part of the controller is run outside the Pixhawk which is called offboard. The Pixhawk can be put in an offboard control mode. In this mode the setpoints for the onboard controller are computed offboard. The offboard control mode accepts attitude, position, velocity and acceleration setpoints. It also allows to directly control the actuators. The control setup can be seen in figure 4.2.

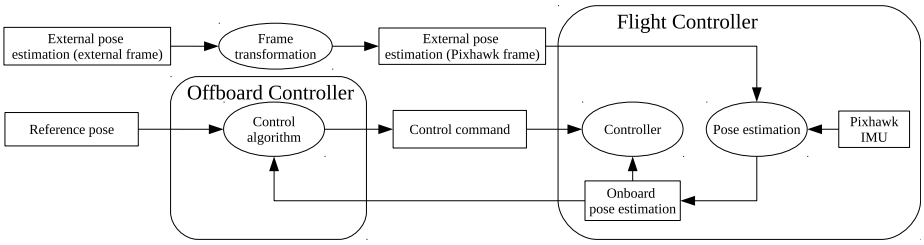

Figure 4.2: Control setup

Two external poses can be seen in figure 4.2. One is the external pose expressed in its own frame. The location of these frames can be seen in figure 4.1. The other is the external pose expressed in the Pixhawk frame. Inbetween is a transformation which transforms the external pose from its own frame to the Pixhawk frame. This external pose can be OptiTrack, visual odometry, the pose output of the sensor fusion or another pose. In the Pixhawk the external pose is used in the pose estimation to compensate for drift of the Pixhawk IMU. The onboard pose estimation consists of two parts: an attitude estimation and a position estimation. Both estimators estimate position and attitude based on IMU readings from the Pixhawk IMU which are corrected by the external pose. The onboard pose estimator publishes its pose which is used in the offboard controller. The control algorithm compares the onboard pose estimation with a reference pose and publishes a control command based on this difference. This control command is sent to the controller onboard the Pixhawk.

4.2.1 Frame transformation

Figure 4.2 shows that a frame transformation is required to transform the pose expressed in the external frame such that it is expressed in the Pixhawk frame. For the visual odometry and the sensor fusion this pose is estimated using a ruler. The position of the VO frame expressed in the Pixhawk frame is estimated at

$$p^P_{VO} = \begin{bmatrix} 0.15 \\ -0.055 \\ -0.1 \end{bmatrix} \tag{4.1}$$

The rotation of the VO frame expressed in the Pixhawk frame is estimated at

$$R^P_{VO} = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \end{bmatrix} \tag{4.2}$$

For the sensor fusion the rotation is estimated the same as VO $R^P_{SF} = R^P_{VO}$. The position is estimated at

$$p^P_{SF} = \begin{bmatrix} 0.15 \\ 0 \\ -0.1 \end{bmatrix} \tag{4.3}$$

For OptiTrack it is assumed that the center of the markers is at the same position as the Pixhawk IMU. The OptiTrack pose estimation is initialized with the

Pixhawk frame aligned with the OptiTrack frame. This makes that no transformation matrix is required since the OptiTrack frame is aligned in position and orientation with the Pixhawk frame.

4.2.2 Offboard controller

The offboard controller used is a PD position+yaw controller with velocity setpoints as a control command to the Pixhawk. Velocity setpoints as an output is chosen because of compatibility with other UAVs. Also an advantage of using an offboard control algorithm is that it is easier to tune and understand what is going on. The reference position and yaw can be set by a Graphical User Interface (GUI). This GUI also allows to set the gains of the controller. A screenshot of the GUI can be seen in figure 4.3.

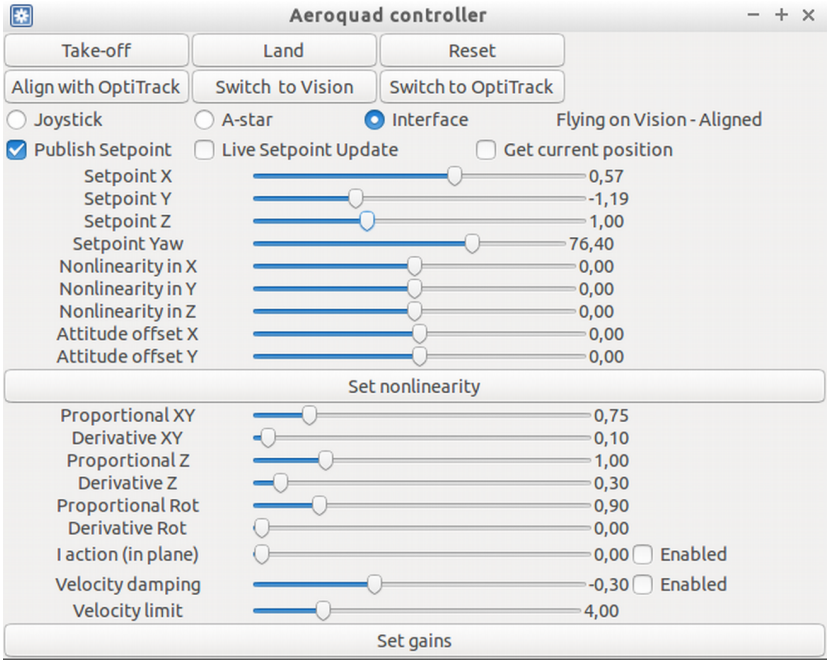

Figure 4.3: Graphical User Interface

4.3 Fail-safe

When flying on visual odometry it is possible that the visual odometry algorithm fails. This can be caused by too little features inside the image (for example flying in front of a evenly white wall) or motion blur caused by rapid movements. When visual odometry fails the position and attitude estimator are not able to estimate the pose of the UAV correctly which will cause the UAV to crash. Since most of the test flights are still performed inside an OptiTrack arena a fail-safe is implemented which switches back to OptiTrack if the visual odometry fails. This can be seen in figure 4.4.

Figure 4.4: Fail-safe

The input poses are not aligned with each other. For continuity in the output of the fail-safe during the switch it is necessary that both poses are aligned with each other. The OptiTrack pose is sent as the external pose when the fail-safe is activated. For safety it is best that the OptiTrack pose is not changed. Therefor the VO pose has to modified to align both input poses. How to align these frames is elaborated in subsection 4.3.1. With the input poses aligned the fail-safe also has to decide whether it should be activated. This is further elaborated in subsection 4.3.2.

4.3.1 Frame alignment

The vision pose and the OptiTrack pose are not aligned with each other. Both poses are presented in their own frame of reference. The vision pose can be expressed as homogeneous matrix H_S^V. A homogeneous matrix describes the orientation and translation in one matrix:

$$H_S^V = \begin{bmatrix} R_S^V & p_S^V \\ 0 & 1 \end{bmatrix} \tag{4.4}$$

where R_S^V is the orientation of the sensor expressed in the visual odometry frame of reference and p_S^V is the position of the sensor expressed in the VO frame of reference. In the same way the OptiTrack pose can be expressed as H_S^O which is the pose of the sensor expressed in the OptiTrack reference frame.

The OptiTrack frame is used as reference frame so the sensor pose expressed in the VO frame has to be expressed in the OptiTrack reference frame. To express the sensor pose in the OptiTrack reference frame the following transformation is needed:

$$H_S^O = H_V^O H_S^V \tag{4.5}$$

This requires the transformation matrix H_V^O. This transformation matrix can be calculated during a frame alignment procedure:

$$H_S^O = H_V^O H_S^V \tag{4.6}$$
$$H_V^O = H_S^O (H_S^V)^{-1} \tag{4.7}$$

The resulting transformation matrix H_V^O can now be used to transform new sensor pose measurements from the VO reference frame into the OptiTrack reference frame. Note that the frames are only aligned at the moment of frame alignment. After the alignment procedure the sensor pose in the VO reference frame can drift again. Continuously aligning the VO frame with the OptiTrack frame would mean that while flying on vision actually the OptiTrack pose is

used. Aligning the OptiTrack reference frame with the VO reference frame would require to transform the OptiTrack pose which is not desirable since the OptiTrack is the backup and transforming this pose might cause the backup to fail.

4.3.2 Switching procedure

There are two situations in which the fail-safe should switch to OptiTrack. The most obvious one is when the visual odometry fails to estimate a pose. This can be caused by insufficient inliers or a reprojection error. This is easy to detect since an error is produced by the VO algorithm. The other situation is when the visual odometry provides a bad estimation which is way off the actual value. This can happen when the visual odometry has little features but still finds a solution (which is an incorrect solution). To detect this situation the velocity of the visual odometry is compared with the velocity of the OptiTrack pose measurement. When the difference exceeds a threshold value, the fail-safe switch is activated. The velocity of the VO can be obtained directly from the velocity measurement of the VO. For OptiTrack only a pose measurement is received. To estimate the OptiTrack velocity a first order fixed window approach is used with a window of 10 samples. This calculates the velocity as

$$v(n) = \frac{p(n) - p(n-10)}{t(n) - t(n-10)} \tag{4.8}$$

The fail-safe detector has been implemented as a state machine. This can be seen in figure 4.5.

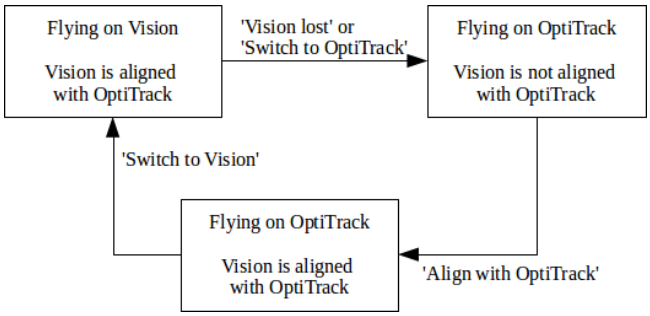

Figure 4.5: Fail-safe detector state machine

Information of the state in which the state machine is is provided to the user through the GUI. This can also be seen in figure 4.3. The system starts up in the 'Flying on OptiTrack - not aligned' state. By pressing the 'align' button the vision pose is aligned with the OptiTrack pose according to the procedure described above. This brings the state machine in the 'Flying on OptiTrack - aligned' state. Still the OptiTrack pose is sent to the UAV. Now the system is ready to make the switch to vision. No sudden changes are expected since both poses are aligned. By pressing the 'switch to vision' button the system will switch to vision which means that now the vision pose is sent to the UAV as external pose. The state machine is now in the 'Flying on Vision - aligned' state. For every state holds that when the switching conditions of the fail-safe

are met the state machine will go to the 'Flying on OptiTrack - not aligned' state. Flying on OptiTrack since this is the safest mode now and not aligned since it is not sure that vision and OptiTrack are still aligned. Alignment and switching to vision however can be performed in air. Additionally switching to OptiTrack can also be done by pressing the 'switch to OptiTrack' button in the GUI.

Chapter 5

Experiments and discussion

Three experiments have been performed to test the performance of the algorithms presented before. The first experiment compares the different position and yaw estimation algorithms during a flight. It appeared that motor vibrations affect the sensor fusion a lot therefore a second experiment has been performed in which the motors are not armed. The third experiment shows the results of the mapping algorithm described in chapter 2. At the end of the chapter there is a discussion on the experiments and the presented work.

5.1 Evaluation of position and yaw estimation during flight

The goal of this experiment is to evaluate and compare the position and yaw estimations of the different pose estimation algorithms. The experiment setup is shown in figure 5.1.

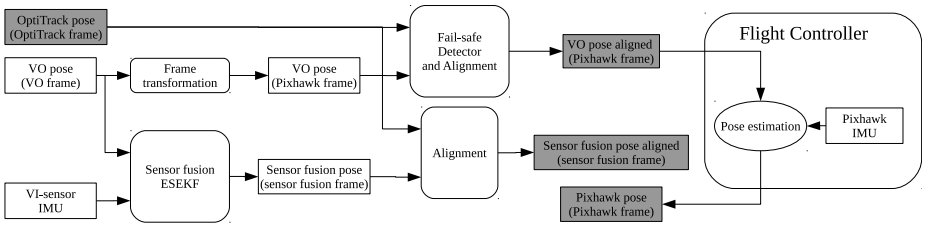

Figure 5.1: Experiment setup

This frame shows at the left the OptiTrack pose and the visual odometry (VO) pose obtained from the stereo camera images. The VO pose is first transformed to the Pixhawk frame using the frame transformation described in subsection 4.2.1. Then it is used as an input for the fail-safe detector together with the OptiTrack pose. The fail-safe detector also aligns both frames. The output of the fail-safe detector is the VO pose expressed in the Pixhawk frame aligned with the OptiTrack pose. This pose is sent to the UAV where the onboard pose estimation fuses this pose with the Pixhawk IMU measurements. This results in the Pixhawk pose which is the pose used for control of the UAV.

The other path in the figure shows the VO pose fused with the VI-sensor IMU using the ESEKF. This results in the sensor fusion pose expressed in the sensor fusion frame. Unfortunately the frame transformation to express the pose in the Pixhawk frame is forgotten to include here. The sensor fusion pose is also aligned with the OptiTrack pose such that the poses are easy to compare. This results in the sensor fusion pose expressed in the sensor fusion frame aligned with OptiTrack. The position and yaw of the pose estimations in the grey boxes are compared in the experiments.

The motor vibrations have a big effect on the IMU measurements. This is shown in figure 5.2

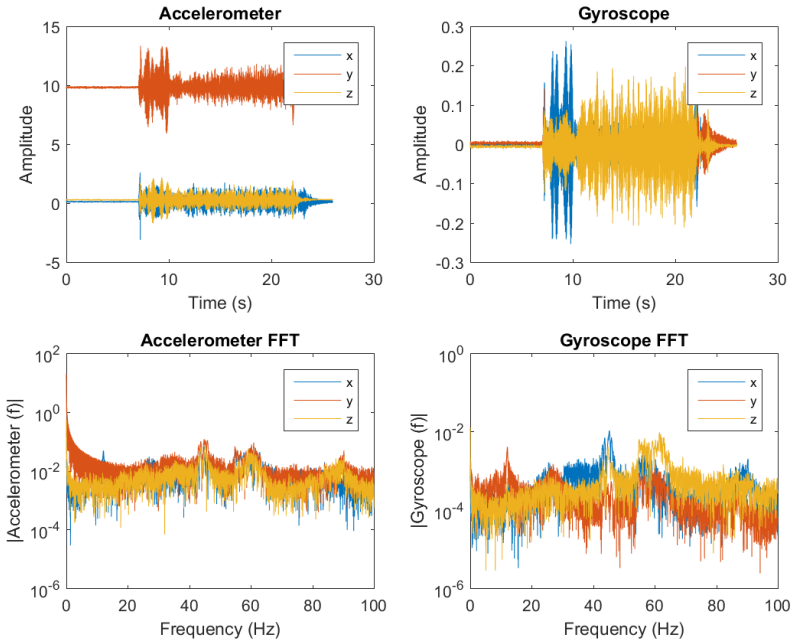

Figure 5.2: Effect of motor vibrations on IMU measurements

The noise on the accelerometer and gyroscope increases a lot when the motors are armed at around $t = 8$. This effect will be taken into account in the sensor fusion by increasing the noise parameters of the accelerometer and the gyroscope by a factor 100. The bias parameters remain the same.

To compare the different algorithms a flight is performed in which all the three linear directions and the yaw have been excited. It appeared from the experiment that the Pixhawk pose had a slight delay of 0.14 s. It is not clear what caused this delay. This has been compensated in the post-processing of the experiment. Also all the frames have been aligned at $t = 0$. The results of the flight can be seen in figure 5.3 and figure 5.4.

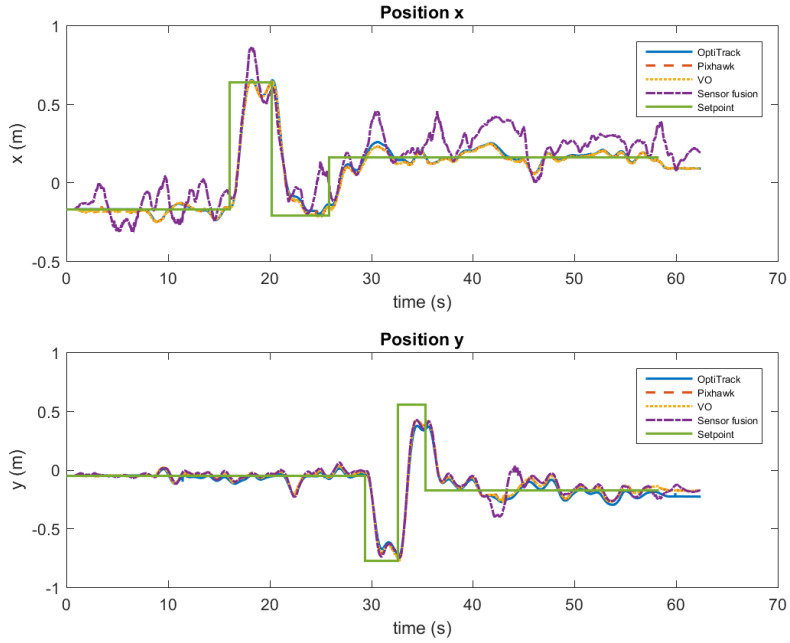

Figure 5.3: Comparison of pose estimation algorithms, x and y.

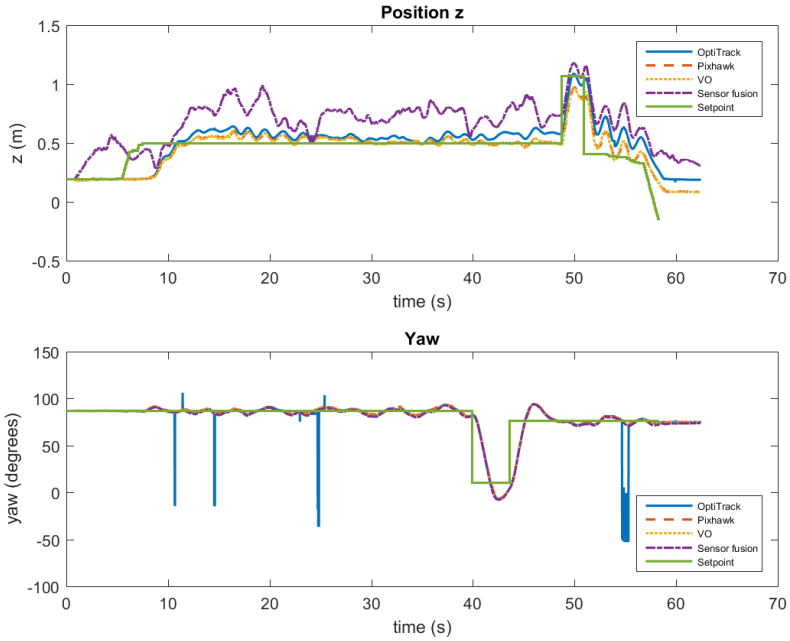

Figure 5.4: Comparison of pose estimation algorithms, z and yaw.

It can be seen that the UAV follows the setpoint. How well it follows this setpoint is a matter of control. In this experiment we are only interested in the accuracy of the pose estimation. It can be seen that for x, y and z the VO, the Pixhawk and the OptiTrack estimations are similar. For x and z the Sensor fusion estimation clearly shows a low accuracy. For y however it seems to follow the OptiTrack estimation. A remark has to be made for the Sensor fusion. The frame transformation shown in figure 4.2 has not been included. The alignment procedure compensates for this error when the yaw does not change but at $t = 40 - 48$ the yaw changes which causes an error in x and y at that time. The yaw estimation of OptiTrack shows short peaks. OptiTrack estimates the pose based on the location of multiple markers. If these markers are close to a rotation-symmetric configuration OptiTrack can measure a different orientation of the UAV. This could be the cause of these short peaks. In these figures it is hard to see what the differences between the different pose estimation algorithms are. In figure 5.5 and figure 5.6 these differences are better shown. In these figures OptiTrack is used as ground truth and for every pose estimation the difference with OptiTrack is shown. The absolute position error in figure 5.6 is calculated by $e_p = \sqrt{e_x^2 + e_y^2 + e_z^2}$.

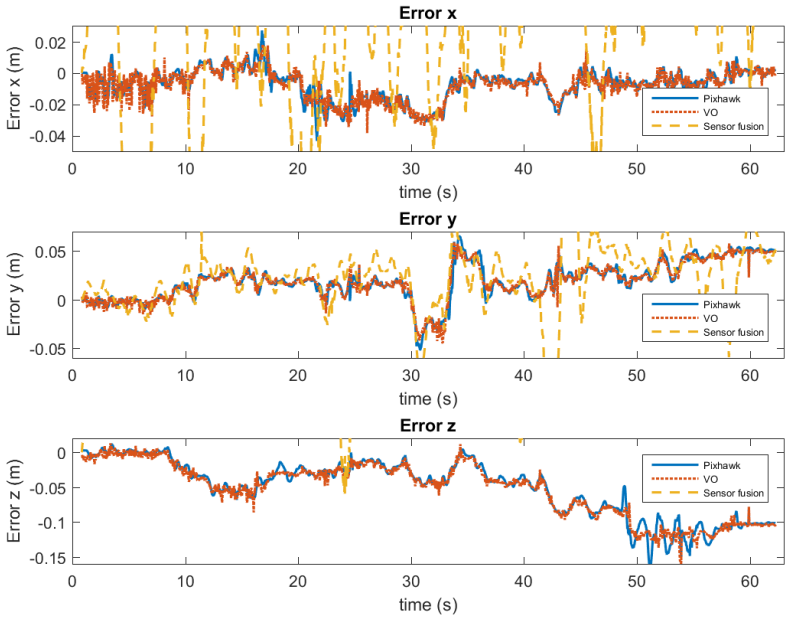

Figure 5.5: Pose estimation error for x, y, and z.

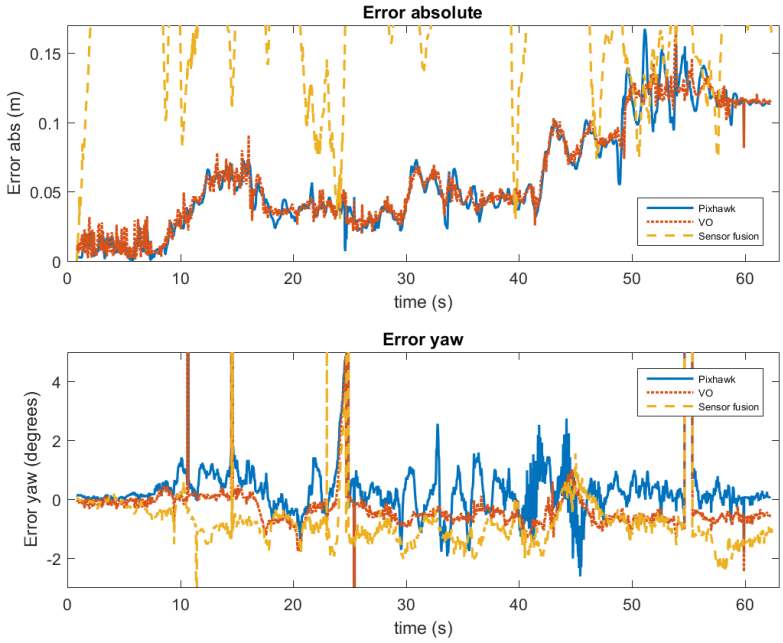

Figure 5.6: Pose estimation error for yaw and absolute position error.

Figure 5.5 shows that the error of the VO position is within 3 cm for x and within 6 cm for y. The position error in z shows a drift of approximately 13 cm after 60 seconds. The absolute position error shown in figure 5.6 is also approximately 13 cm which is mostly caused by the position error in z. For the error in yaw it can be seen that the error stays within 3 degrees. It was expected that the Pixhawk error would follow the VO error but apparently they are not the same. It is not clear what the reason is why they are not the same.

5.2 Evaluation of sensor fusion

As seen in figure 5.2 the motor vibrations have a big effect on the IMU measurements. Therefore also a 'flight' has been performed without the motors armed. The UAV is moved around in the OptiTrack area and the estimation of the sensor fusion is recorded. The same experiment setup as in figure 5.1 is used. The original IMU noise values are used. The result of this experiment can be seen in figure 5.7 and 5.8.

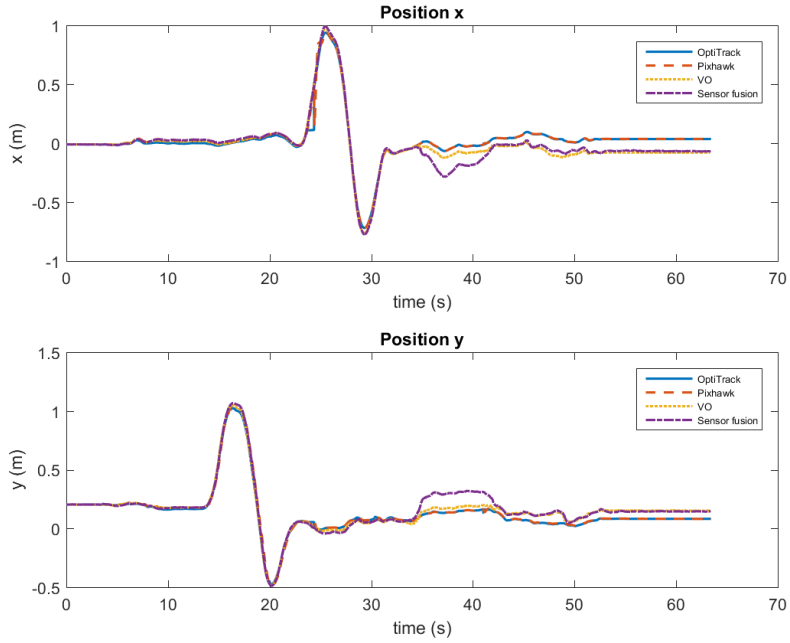

Figure 5.7: Evaluation of sensor fusion, x and y.

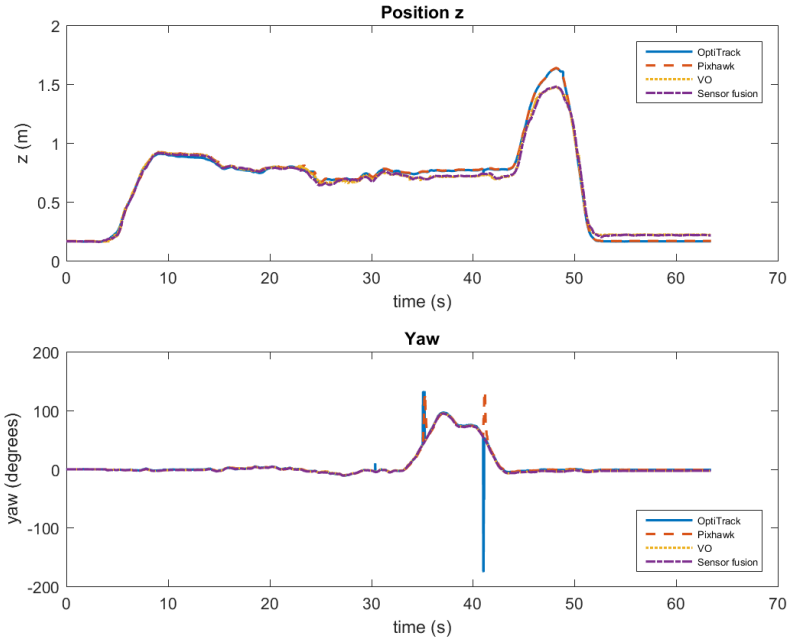

Figure 5.8: Evaluation of sensor fusion, z and yaw.

In this figure it can also be seen that an extra error in the sensor fusion position x and y is introduced when the UAV is yawing from $t = 35-42$. Around $t = 25$ a short OptiTrack error occurred. Since the fail-safe algorithm checks the difference in velocity of OptiTrack and VO the fail-safe is also activated when OptiTrack generates a position error. That is the case at around $t = 25$. From this moment the Pixhawk pose estimation receives OptiTrack as external pose. The peaks in the yaw plot have the same reason as with the previous experiment. In these figures the performance of the sensor fusion is better now. Still it is hard to see how well it is performing. Therefore also an error plot is made. This can be seen in figure 5.9 and figure 5.10.

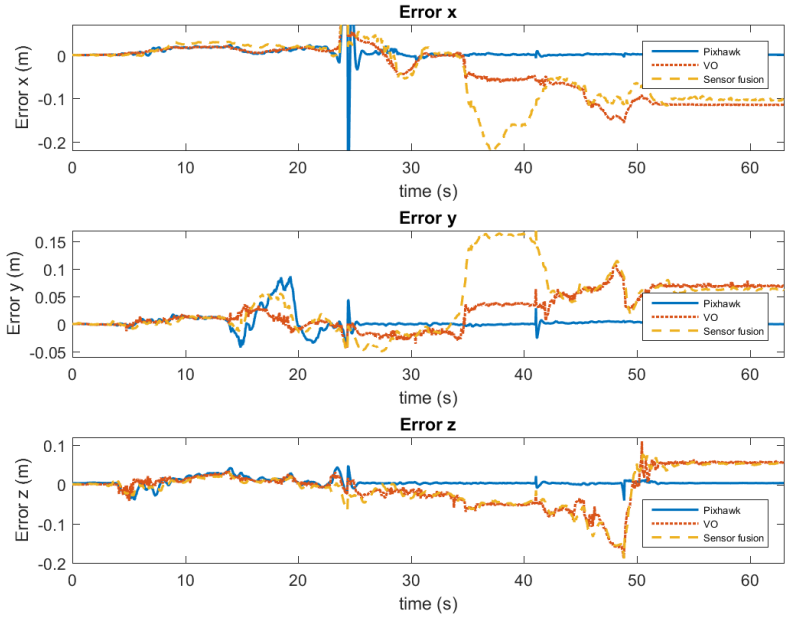

Figure 5.9: Evaluation of sensor fusion for x, y and z.

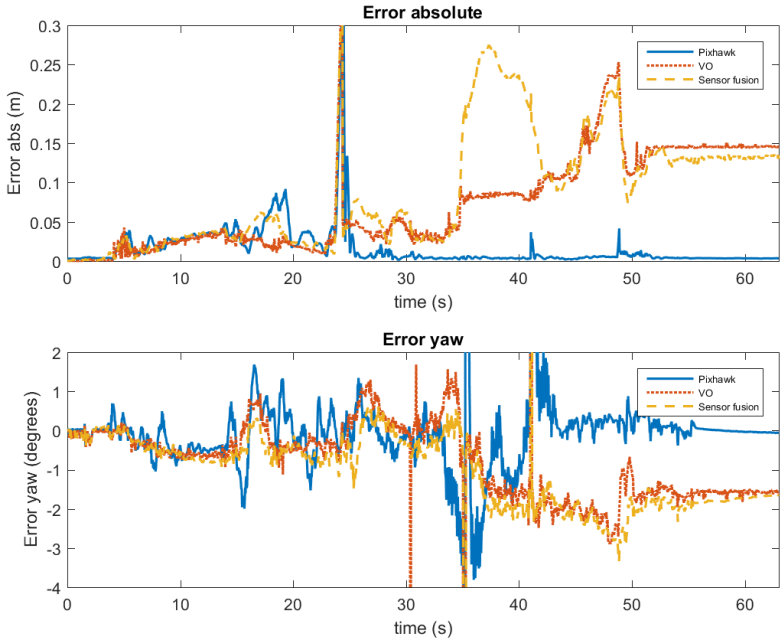

Figure 5.10: Evaluation of sensor fusion for yaw and absolute position error.

Still the error due to the missing transformation matrix can be seen. Apart from that the sensor fusion seems to follow the VO. The absolute position error is approximately 25 cm after 50 seconds but then reduces to approximately 15 cm.

5.3 Mapping

In chapter 2 the mapping algorithm is described. A flight has been performed with the VI-sensor mounted on the UAV to map the environment. The Pixhawk pose is used as reference to project the point clouds in the map. The Pixhawk is flying using visual odometry. Figure 5.11 shows the total map of the small flying arena of the RAM group at the University of Twente created by the UAV. Figure 5.12 shows the map compared to a photo from approximately the same point of view.

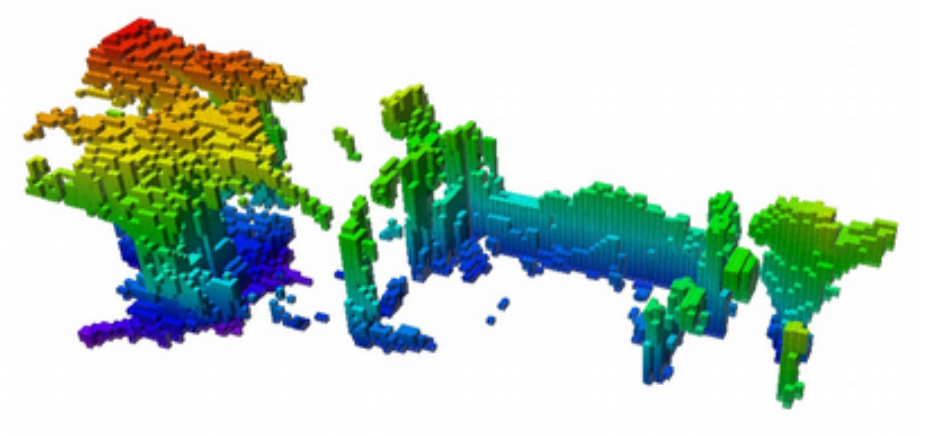

Figure 5.11: Octomap of the small fly arena of the RAM group at the University of Twente

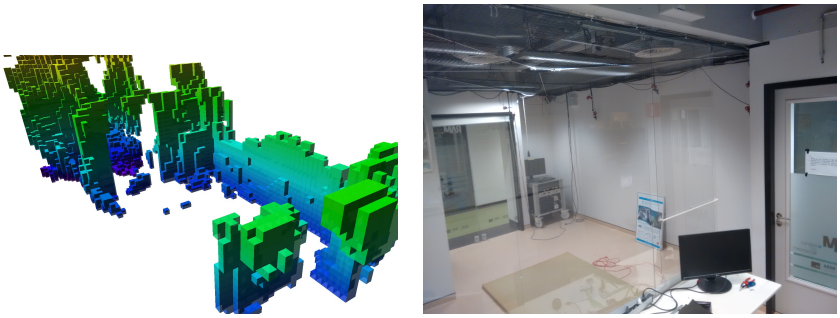

Figure 5.12: Comparison of map and photo.

The comparison of the map and the photo shows that walls closeby are well observed. Also the glass door can be seen in the map as a rectangular hole in the wall. Below in the image also the desk with the computer can be observed. Further away from the UAV the map does not look that accurate anymore. This could be caused by the Stereo Block matching algorithm not being accurate at long distances. Maybe other point cloud generation techniques will provide better results.

5.4 Discussion

This section will discuss the experimental results and the presented work.

5.4.1 Frame transformation

The final pose estimation which is used for control is expressed in the Pixhawk frame of reference. This makes that all the external pose estimations which are sent to the Pixhawk also have to be expressed in the Pixhawk frame of reference.

The experiments shown in section 5.1 and 5.2 show that not incorporating this transformation leads to additional errors. For VO this transformation between VO and Pixhawk has been estimated using a ruler and for OptiTrack it is tried to align the OptiTrack frame as good as possible with the Pixhawk frame such that no transformation is needed. The better this transformation is estimated, the lower the error caused by misalignment will affect the pose estimation. This would result in better flight performance. The sensor fusion described in chapter 3 is able to estimate the transformation matrix between an external pose sensor and an IMU. This could be used to estimate these frame transformations.

5.4.2 Motor vibrations

As shown in figure 5.2 the motor vibrations have a big effect on the IMU measurements of the VI-sensor IMU. The experiments show that this deteriorates the performance of the sensor fusion. Already damping between the UAV and the VI-sensor has been applied to reduce the vibrations but this is still not enough. Aligning the motors could help to reduce the vibrations of the motor. The motor vibrations are now tried to be compensated by modeling it as an increase of noise in the sensor fusion. However, the source of the vibrations is clear and since the motors are spinning at high speed the vibrations could also be reduced by applying a low-pass filter on the IMU measurements. This has been researched a little bit with Matlab on the same IMU data as figure 5.2. An averaging filter where the last five measurements are averaged gives the following result:

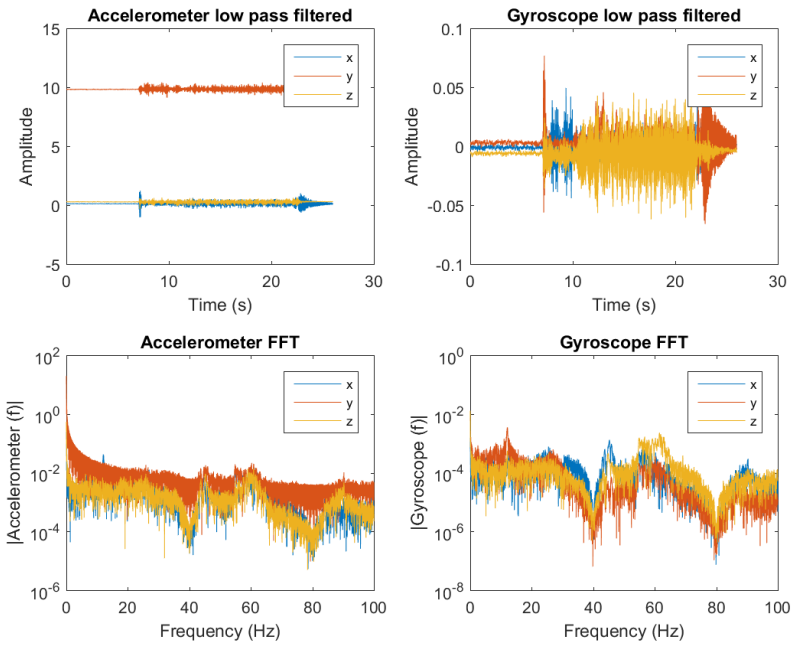

Figure 5.13: Effect of motor vibrations on IMU measurements

It can be seen that the vibrations are reduced significantly compared to figure 5.2. But this has to be tested to evaluate the effects in the real setup.

5.4.3 Sensor fusion

The ESEKF is implemented to obtain a high-bandwidth pose estimation from the VI-sensor. This is useful when the VI-sensor is used separately. However in the UAV setup also a sensor fusion algorithm is implemented onboard of the UAV. This made it possible to fly the UAV using only the VO measurement. The proposed setup had two sensor fusion algorithms cascaded. When the sensor fusion performs properly with the motors armed it should be tested if this cascaded sensor fusion will have better performance than using only one sensor fusion algorithm. Bypassing the Pixhawk sensor fusion however requires that the control of the UAV also has to be done offboard.

5.4.4 VO-IMU coupling

The visual odometry pose estimation and the IMU are loosely-coupled in the ESEKF meaning that both are treated as separate parts not interacting with each other. This makes that when visual odometry is lost it re-initializes from its last known visual odometry pose and not the sensor fusion pose. An attempt has been done to re-initialize at the sensor fusion pose but this hasn't been tested. It may be better to use a tightly-coupled sensor fusion for fusing the visual odometry pose with the IMU.

5.4.5 Point cloud generation

The map is generated based on the point clouds which are generated from the stereo camera images. The algorithm used for this is Stereo Block matching. According to the KITTI benchmark[7] there are algorithms with better results which can also run at real time. A more accurate point cloud results in a more accurate map.

5.4.6 Outdoor

The setup has now only been tested indoor. It is designed for outdoor operation so additional experiments should be performed to test the performance in an outdoor environment.

5.4.7 Multiple sensors

The drawback of visual odometry is that when the cameras can't find features in the images the algorithm is not capable of estimation the pose of the UAV. This means that the pose estimator in the flight controller is not able to estimate the pose accurately since it only has IMU measurements as input. Position control is then not possible anymore. Adding more sensors to the framework like GPS lowers the chance that no pose can be estimated.

Chapter 6

Conclusion and recommendations

6.1 Conclusion

This project aimed at obtaining an accurate localization from vision. The VI-sensor was used to provide stereo camera images and IMU measurements. The visual odometry algorithm FOVIS obtained a pose estimation from the stereo camera images. The sensor fusion algorithm published by Weiss has been used to fuse the VO pose with the IMU measurements. This provides a high-bandwidth pose estimation from the VI-sensor. This high-bandwidth pose estimation was aimed to be used in a UAV setup providing the localization of the UAV. However the motor vibrations affected the sensor fusion too much to be used in the setup. Therefore only the low-bandwidth VO pose has been used for providing the localization of the UAV. A working setup has been created with a hexacopter equipped with the VI-sensor and an Intel NUC which can fly using visual odometry pose estimation. A fail-safe has been designed and implemented to prevent the UAV from crashing when visual odometry fails and flying inside an Opti-Track area. Experiments show a good accuracy and drifts up to 15 cm/min. During the flight also a map of the environment can be created.

6.2 Recommendations

6.2.1 Robust pose from vision

A problem encountered in this project is that when visual odometry is lost once an error is introduced between the estimated pose and the actual pose. This has been tried to solve by adding a bias to the VO pose which should align both frames. Better would be to make a tightly-coupled sensor fusion which fuses visual odometry with IMU. This will make the pose estimation more robust. Time delay compensation and roll and pitch drift compensation should be included in the new sensor fusion.

6.2.2 Motor vibrations

The effect of motor vibrations on the IMU measurements has to be reduced. This should be done preferably mechanically. Using proper damping between the motors and the VI-sensor can reduce the damping. This has to be investigated more. Another mechanical solution is aligning the motors to reduce the eccentricity. The effect of vibrations can also be reduced in the software by implementing a low-pass filter on the IMU measurements.

6.2.3 Frame transformation estimation

Using incorrect frame transformations between the Pixhawk and the external pose sensor leads to additional errors in the localization. These frames have to be estimated more accurately. The sensor fusion algorithm of Weiss described in this project can likely be used to estimate these frame transformations.

6.2.4 Cascaded sensor fusion

When the ESEKF provides accurate enough pose estimations to be used in flight the system consists of two sensor fusion algorithms: the ESEKF and the onboard pose estimation. It should be investigated how much benefit is gained from using only VO and onboard pose estimation to using ESEKF and onboard pose estimation.

6.2.5 Outdoor testing

The setup has been designed for outdoor operation but it has never been tested outdoor. This should be done to verify that the setup can be used outdoor.

6.2.6 Additional sensors

More sensors should be added to reduce the dependency on the visual odometry pose. When VO is lost this shouldn't lead to uncontrollability of the UAV. For example including GPS would still provide pose estimation when VO is lost even if this GPS pose is less accurate. It should be considered however that extra sensors will add extra mass to the UAV.

6.2.7 SLAM

The VO pose estimation is subject to drift. Implementing a visual SLAM (Simultaneous Localization And Mapping) algorithm would eliminate the drift in the pose estimation. Next to a more accurate pose estimation elimination of the drift will also result in more accurate maps. A visual SLAM algorithm however is more computational intensive than VO. If VO and visual SLAM are used next to each other then VO can provide real-time pose estimation and the visual SLAM can correct for the drift of the VO. Then it is not necessary that the visual SLAM algorithm runs in real-time.

6.2.8 Multi sensor fusion

In this project the single sensor fusion (SSF) framework has been used. From the same authors and with the same principle also exists the multi sensor fusion (MSF) framework. This allows to fuse more than one sensor with the IMU measurements. Also when only one sensor is used next to the IMU it is good to use the MSF since it seems that this framework is maintained.

6.2.9 Point cloud generation

The map is built using point clouds generated by Stereo Block Matching. As shown in the discussion better point cloud generation techniques exist. Implementing these newer techniques would increase the accuracy of the generated point cloud and therefore also the accuracy of the created map.

6.2.10 VIKI integration

VIKI is a ROS GUI currently developed within the AEROWORKS group. Its goal is to make it easier for the user to use ROS packages and connecting different packages into one system. This requires a modular design of the system. The current implementation of this project has to be made more modular. The fail-safe, control and GUI are now integrated in one module while it would be better to separate these modules. Then the modules have to be integrated in the VIKI framework.

6.2.11 Haptic interface

Currently the UAV is operated by moving sliders in a GUI on a computer screen. A haptic interface provides a more natural way of controlling the UAV. Also information from the environment can be def to the pilot as a haptic feedback.

Appendix A

Observation matrix of ESEKF

This appendix describes how to derive the observation matrix used in the ESEKF. The observation matrix H relates the measurement function $\tilde{z}$ to the error state used in the ESEKF $\tilde{x}$ as

$$\tilde{z} = H\tilde{x} \tag{A.1}$$

A.1 Position

For the position measurement part the following measurement function is used:

$$h_p(\tilde{x}) = \tilde{z}_p = R_w^v(p_i^w + R_i^w p_c^i)\lambda + n_p - \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda} \tag{A.2}$$

To obtain the observation matrix, this function has to be linearized around the estimated state:

$$H_p = \frac{\partial h_p(\tilde{x})}{\partial \tilde{x}}|_{x=\hat{x}} \tag{A.3}$$

This equation can be solved for every state apart and then be concatenated at the end. Let's start with solving it for $\tilde{p}_i^w$. First we fill in the equation:

$$\frac{\partial h_p(\tilde{x})}{\partial \tilde{p}_i^w}|_{\mathbf{x}=\hat{\mathbf{x}}} = \frac{\partial(R_w^v(p_i^w + R_i^w p_c^i)\lambda + n_p - \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda})}{\partial \tilde{p}_i^w}|_{\mathbf{x}=\hat{\mathbf{x}}} \tag{A.4}$$

We want to find the derivative with respect to $\tilde{p}_i^w$. It can be seen that p_i^w can be rewritten as $p_i^w = \tilde{p}_i^w + \hat{p}_i^w$. This will result in the following equation:

$$\frac{\partial h_p(\tilde{x})}{\partial \tilde{p}_i^w}|_{\mathbf{x}=\hat{\mathbf{x}}} = \frac{\partial(R_w^v((\tilde{p}_i^w + \hat{p}_i^w) + R_i^w p_c^i)\lambda + n_p - \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda})}{\partial \tilde{p}_i^w}|_{\mathbf{x}=\hat{\mathbf{x}}} \tag{A.5}$$

The next step is to evaluate the equation at $x = \hat{x}$. This will result that all the variables referring to the real values (without a hat) can be substituted with variables referring to the estimate (with a hat).

$$\frac{\partial h_p(\tilde{x})}{\partial \tilde{p}_i^w}|_{\mathbf{x}=\hat{\mathbf{x}}} = \frac{\partial(\hat{R}_w^v((\tilde{p}_i^w + \hat{p}_i^w) + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda} + n_p - \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda})}{\partial \tilde{p}_i^w} \tag{A.6}$$

Analysing this equation shows that a lot of variables are cancelled out. Removing this variables will result in:

$$\frac{\partial h_p(\tilde{x})}{\partial \tilde{p}_i^w}|_{\mathbf{x}=\hat{\mathbf{x}}} = \frac{\partial(\hat{R}_w^v \tilde{p}_i^w \hat{\lambda} + n_p)}{\partial \tilde{p}_i^w} \tag{A.7}$$

Since $\hat{\lambda}$ is a scalar it is easy to solve the equation now resulting in

$$\frac{\partial h_p(\tilde{x})}{\partial \tilde{p}_i^w}|_{\mathbf{x}=\hat{\mathbf{x}}} = \hat{R}_w^v \hat{\lambda} \tag{A.8}$$

The same procedure can be repeated for the other states. For calculating $\frac{\partial h(\tilde{x})}{\partial \tilde{v}_i^w}|_{\mathbf{x}=\hat{\mathbf{x}}}$ it is easy to see that

$$\frac{\partial h_p(\tilde{x})}{\partial \tilde{v}_i^w}|_{\mathbf{x}=\hat{\mathbf{x}}} = \frac{\partial(R_w^v(p_i^w + R_i^w p_c^i)\lambda + n_p - \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda})}{\partial \tilde{v}_i^w}|_{\mathbf{x}=\hat{\mathbf{x}}} \tag{A.9}$$

$$= \mathbf{0_{3x3}} \tag{A.10}$$

For determining the derivative with respect to $\delta\theta_i^w$ the derivation is slightly different. We first start with writing down the equation

$$\frac{\partial h_p(\tilde{x})}{\partial \delta\theta_i^w}|_{\mathbf{x}=\hat{\mathbf{x}}} = \frac{\partial(R_w^v(p_i^w + R_i^w p_c^i)\lambda + n_p - \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda})}{\partial \delta\theta_i^w}|_{\mathbf{x}=\hat{\mathbf{x}}} \tag{A.11}$$

Now we want to substitute R_i^w with a term containing the estimate $\hat{R}_i^w$ and the error $\delta\theta_i^w$. This relation can be derived from the definition of the error quaternion.

$$q_i^w = \hat{q}_i^w \otimes \delta q_i^w \tag{A.12}$$

$$R_i^w = R_{\hat{i}}^w R_i^{\hat{i}} \tag{A.13}$$

$$= \hat{R}_i^w(\mathbf{I_3} + \lfloor \delta\theta_i^w \times \rfloor) \tag{A.14}$$

This relation can now be used to substitute R_i^w.

$$\frac{\partial h_p(\tilde{x})}{\partial \delta\theta_i^w}|_{\mathbf{x}=\hat{\mathbf{x}}} = \frac{\partial(R_w^v(p_i^w + \hat{R}_i^w(\mathbf{I_3} + \lfloor \delta\theta_i^w \times \rfloor)p_c^i)\lambda + n_p - \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda})}{\partial \delta\theta_i^w}|_{\mathbf{x}=\hat{\mathbf{x}}} \tag{A.15}$$

From here the same procedure is used by evaluating the equation at $x = \hat{x}$ and cancelling out terms which are the same.

$$\frac{\partial h_p(\tilde{x})}{\partial \delta\theta_i^w}|_{\mathbf{x}=\hat{\mathbf{x}}} = \frac{\partial(\hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w(\mathbf{I_3} + \lfloor \delta\theta_i^w \times \rfloor)\hat{p}_c^i)\hat{\lambda} + n_p - \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w\hat{p}_c^i)\hat{\lambda})}{\partial \delta\theta_i^w} \tag{A.16}$$

$$= \frac{\partial(\hat{R}_w^v\hat{R}_i^w\lfloor \delta\theta_i^w \times \rfloor \hat{p}_c^i\hat{\lambda} + n_p)}{\partial \delta\theta_i^w} \tag{A.17}$$

The final step is to rewrite the equation such that $\delta\theta_i^w$ is moved to the end of the equation. This can be done by using the relation

$$A \times B = -B \times A \tag{A.18}$$

When this is applied on the equation this would make

$$\lfloor \delta\theta_i^w \times \rfloor \hat{p}_c^i = -\lfloor \hat{p}_c^i \times \rfloor \delta\theta_i^w \tag{A.19}$$

This will continue the derivation with

$$\frac{\partial h_p(\tilde{x})}{\partial \delta\theta_i^w}|_{\mathbf{x}=\hat{\mathbf{x}}} = \frac{\partial(\hat{R}_w^v\hat{R}_i^w\lfloor \delta\theta_i^w \times \rfloor \hat{p}_c^i\hat{\lambda} + n_p)}{\partial \delta\theta_i^w} \tag{A.20}$$

$$= \frac{\partial(-\hat{R}_w^v\hat{R}_i^w\lfloor \hat{p}_c^i \times \rfloor \delta\theta_i^w\hat{\lambda} + n_p)}{\partial \delta\theta_i^w} \tag{A.21}$$

$$= -\hat{R}_w^v\hat{R}_i^w\lfloor \hat{p}_c^i \times \rfloor \hat{\lambda} \tag{A.22}$$

The derivation of the other states is similar to the derivations described above. The derivations are listed below.

The derivative with respect to $\tilde{b}_\omega$:

$$\frac{\partial h_p(\tilde{x})}{\partial \tilde{b}_\omega}|_{\mathbf{x}=\hat{\mathbf{x}}} = \frac{\partial(R_w^v(p_i^w + R_i^w p_c^i)\lambda + n_p - \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w\hat{p}_c^i)\hat{\lambda})}{\partial \tilde{b}_\omega}|_{\mathbf{x}=\hat{\mathbf{x}}}$$
$$= \mathbf{0_{3x3}}$$

The derivative with respect to $\tilde{b}_a$:

$$\frac{\partial h_p(\tilde{x})}{\partial \tilde{b}_a}|_{\mathbf{x}=\hat{\mathbf{x}}} = \frac{\partial(R_w^v(p_i^w + R_i^w p_c^i)\lambda + n_p - \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w\hat{p}_c^i)\hat{\lambda})}{\partial \tilde{b}_a}|_{\mathbf{x}=\hat{\mathbf{x}}}$$
$$= \mathbf{0_{3x3}}$$

The derivative with respect to $\tilde{\lambda}$:

$$
\begin{aligned}
\frac{\partial h_p(\tilde{x})}{\partial \tilde{\lambda}}|_{\mathbf{x}=\hat{\mathbf{x}}} &= \frac{\partial (R_w^v(p_i^w + R_i^w p_c^i)\lambda + n_p - \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda})}{\partial \tilde{\lambda}}|_{\mathbf{x}=\hat{\mathbf{x}}} \\
&= \frac{\partial (R_w^v(p_i^w + R_i^w p_c^i)(\tilde{\lambda} + \hat{\lambda}) + n_p - \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda})}{\partial \tilde{\lambda}}|_{\mathbf{x}=\hat{\mathbf{x}}} \\
&= \frac{\partial (\hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)(\tilde{\lambda} + \hat{\lambda}) + n_p - \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda})}{\partial \tilde{\lambda}} \\
&= \frac{\partial (\hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\tilde{\lambda} + n_p}{\partial \tilde{\lambda}} \\
&= \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)
\end{aligned}
$$

The derivative with respect to $\delta\theta_w^v$:

$$
\begin{aligned}
\frac{\partial h_p(\tilde{x})}{\partial \delta\theta_w^v}|_{\mathbf{x}=\hat{\mathbf{x}}} &= \frac{\partial (R_w^v(p_i^w + R_i^w p_c^i)\lambda + n_p - \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda})}{\partial \delta\theta_w^v}|_{\mathbf{x}=\hat{\mathbf{x}}} \\
&= \frac{\partial (\hat{R}_w^v(\mathbf{I_3} + \lfloor \delta\theta_w^v \times \rfloor)(p_i^w + R_i^w p_c^i)\lambda + n_p - \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda})}{\partial \delta\theta_w^v}|_{\mathbf{x}=\hat{\mathbf{x}}} \\
&= \frac{\partial (\hat{R}_w^v(\mathbf{I_3} + \lfloor \delta\theta_w^v \times \rfloor)(\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda} + n_p - \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda})}{\partial \delta\theta_w^v} \\
&= \frac{\partial (\hat{R}_w^v \lfloor \delta\theta_w^v \times \rfloor (\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda} + n_p)}{\partial \delta\theta_w^v} \\
&= \frac{\partial (-\hat{R}_w^v \lfloor (\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda} \times \rfloor \delta\theta_w^v + n_p)}{\partial \delta\theta_w^v} \\
&= -\hat{R}_w^v \lfloor (\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda} \times \rfloor
\end{aligned}
$$

The derivative with respect to $\delta\theta_c^i$:

$$
\begin{aligned}
\frac{\partial h_p(\tilde{x})}{\partial \delta\theta_c^i}|_{\mathbf{x}=\hat{\mathbf{x}}} &= \frac{\partial (R_w^v(p_i^w + R_i^w p_c^i)\lambda + n_p - \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda})}{\partial \delta\theta_c^i}|_{\mathbf{x}=\hat{\mathbf{x}}} \\
&= \mathbf{0_{3x3}}
\end{aligned}
$$

The derivative with respect to $\tilde{p}_c^i$:

$$
\begin{aligned}
\frac{\partial h_p(\tilde{x})}{\partial \tilde{p}_c^i}|_{\mathbf{x}=\hat{\mathbf{x}}} &= \frac{\partial (R_w^v(p_i^w + R_i^w p_c^i)\lambda + n_p - \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda})}{\partial \tilde{p}_c^i}|_{\mathbf{x}=\hat{\mathbf{x}}} \\
&= \frac{\partial (R_w^v(p_i^w + R_i^w(\tilde{p}_c^i + \hat{p}_c^i))\lambda + n_p - \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda})}{\partial \tilde{p}_c^i}|_{\mathbf{x}=\hat{\mathbf{x}}} \\
&= \frac{\partial (\hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w(\tilde{p}_c^i + \hat{p}_c^i))\hat{\lambda} + n_p - \hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i)\hat{\lambda})}{\partial \tilde{p}_c^i} \\
&= \frac{\partial (\hat{R}_w^v(\hat{p}_i^w + \hat{R}_i^w \tilde{p}_c^i)\hat{\lambda} + n_p)}{\partial \tilde{p}_c^i} \\
&= \hat{R}_w^v \hat{R}_i^w \hat{\lambda}
\end{aligned}
$$

This will result in the following observation matrix:

$$H_p = \begin{bmatrix} \hat{R}_w^v \hat{\lambda} \\ \mathbf{0_{3x3}} \\ -\hat{R}_w^v \hat{R}_i^w \lfloor \hat{p}_c^i \times \rfloor \hat{\lambda} \\ \mathbf{0_{3x3}} \\ \mathbf{0_{3x3}} \\ \hat{R}_w^v (\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i) \\ -\hat{R}_w^v \lfloor (\hat{p}_i^w + \hat{R}_i^w \hat{p}_c^i) \hat{\lambda} \times \rfloor \\ \mathbf{0_{3x3}} \\ \hat{R}_w^v \hat{R}_i^w \hat{\lambda} \end{bmatrix} \tag{A.23}$$

A.2 Attitude

For the position measurement part the following measurement function is used:

$$h_q(\tilde{x}) = \tilde{z}_q = (\hat{q}_c^v)^{-1} \otimes q_c^v = (\hat{q}_c^i)^{-1} \otimes (\hat{q}_i^w)^{-1} \otimes (\hat{q}_w^v)^{-1} \otimes q_w^v \otimes q_i^w \otimes q_c^i \tag{A.24}$$

To obtain the observation matrix, this function has to be linearized around the estimated state:

$$H_q = \frac{\partial h_q(\tilde{x})}{\partial \tilde{x}}|_{x=\hat{x}} \tag{A.25}$$

This derivation can also be split up in three parts containing. The first part is the derivative with respect to $\delta\theta_c^i$. The derivative is calculated at $x = \hat{x}$ so every quaternion except q_c^i can be substituted by its estimated variant:

$$h_q(\tilde{\mathbf{x}}) = (\hat{q}_c^i)^{-1} \otimes (\hat{q}_i^w)^{-1} \otimes (\hat{q}_w^v)^{-1} \otimes \hat{q}_w^v \otimes \hat{q}_i^w \otimes q_c^i \tag{A.26}$$

The equation $\hat{q}^{-1} \otimes \hat{q}$ results in the unit quaternion and can then be removed from the equation. This results in the following equation:

$$h_q(\tilde{\mathbf{x}}) = (\hat{q}_c^i)^{-1} \otimes q_c^i \tag{A.27}$$

This equation can be recognized as the definition of the error quaternion so it can also be written as

$$h_q(\tilde{\mathbf{x}}) = \delta q_c^i \tag{A.28}$$

The error quaternion can also be written as its small angle approximation:

$$h_q(\tilde{\mathbf{x}}) = \begin{bmatrix} 1 \\ \frac{1}{2}\delta\theta_c^i \end{bmatrix} \tag{A.29}$$

When we now look at the derivative $\frac{\partial h_q(\tilde{\mathbf{x}})}{\partial \delta\theta_c^i}$ it can be solved as

$$\frac{\partial h_q(\tilde{\mathbf{x}})}{\partial \delta\theta_c^i} = \frac{1}{2}\mathbf{I_3} \tag{A.30}$$

For $\delta\theta_i^w$ the measurement function reduces to:

$$
\begin{aligned}
h_q(\tilde{\mathbf{x}}) &= (\hat{q}_c^i)^{-1} \otimes (\hat{q}_i^w)^{-1} \otimes (\hat{q}_w^v)^{-1} \otimes \hat{q}_w^v \otimes q_i^w \otimes \hat{q}_c^i \\
&= (\hat{q}_c^i)^{-1} \otimes (\hat{q}_i^w)^{-1} \otimes q_i^w \otimes \hat{q}_c^i \\
&= (\hat{q}_c^i)^{-1} \otimes \delta q_i^w \otimes \hat{q}_c^i
\end{aligned}
$$

Now we need the following quaternion property:

$$
q \otimes \begin{bmatrix} 0 \\ x \end{bmatrix} \otimes q^{-1} = \begin{bmatrix} 0 \\ Rx \end{bmatrix} \tag{A.31}
$$

where q and R describe the same rotation.
Now the equation can be rewritten as

$$
\begin{aligned}
h_q(\tilde{\mathbf{x}}) &= \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & R_i^c \end{bmatrix} \delta q_i^w \\
&= \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & R_i^c \end{bmatrix} \begin{bmatrix} 1 \\ \frac{1}{2}\delta\theta_i^w \end{bmatrix}
\end{aligned}
$$

The partial derivative can now be solved:

$$
\begin{aligned}
\frac{\partial h(\tilde{\mathbf{x}})}{\partial \delta\theta_i^w} &= \frac{\partial (R_i^c \frac{1}{2}\delta\theta_i^w)}{\partial \delta\theta_i^w} \\
&= \frac{1}{2} R_i^c
\end{aligned}
$$

And last for $\delta\theta_w^v$ the same procedure can be followed to find the solution:

$$
\begin{aligned}
h(\tilde{\mathbf{x}}) &= (\hat{q}_c^v)^{-1} \otimes q_c^v = (\hat{q}_c^i)^{-1} \otimes (\hat{q}_i^w)^{-1} \otimes (\hat{q}_w^v)^{-1} \otimes q_w^v \otimes \hat{q}_i^w \otimes \hat{q}_c^i \\
&= (\hat{q}_c^i)^{-1} \otimes (\hat{q}_i^w)^{-1} \otimes \delta\theta_w^v \otimes \hat{q}_i^w \otimes \hat{q}_c^i \\
&= \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & R_i^c \end{bmatrix} \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & R_w^i \end{bmatrix} \delta q_w^v \\
&= \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & R_i^c \end{bmatrix} \begin{bmatrix} 1 & \mathbf{0} \\ \mathbf{0} & R_w^i \end{bmatrix} \begin{bmatrix} 1 \\ \frac{1}{2}\delta\theta_w^v \end{bmatrix} \\
\frac{\partial h(\tilde{\mathbf{x}})}{\partial \delta\theta_w^v} &= \frac{\partial (R_i^c R_w^i \frac{1}{2}\delta\theta_w^v)}{\partial \delta\theta_w^v} \\
&= \frac{1}{2} R_i^c R_w^i
\end{aligned}
$$

This makes the full observation matrix

$$
H_q = \begin{bmatrix} \mathbf{0_{3x3}} \\ \mathbf{0_{3x3}} \\ \frac{1}{2}R_i^c \\ \mathbf{0_{3x3}} \\ \mathbf{0_{3x3}} \\ 0 \\ \frac{1}{2}R_i^c R_w^i \\ \frac{1}{2}\mathbf{I_3} \\ \mathbf{0_{3x3}} \end{bmatrix} \tag{A.32}
$$

Appendix B

Physical design

The hexacopter was already assembled and available. A few mounts have been made for the Pixhawk, VI-sensor and Intel NUC.

B.1 Mount of VI-sensor and NUC

Mounting the VI-sensor and the Intel NUC required some more effort. The first designs which were made were passing too much vibrations of the motors to the IMU of the VI-sensor. A few iterations have been made to end up with the design presented in figure B.1.

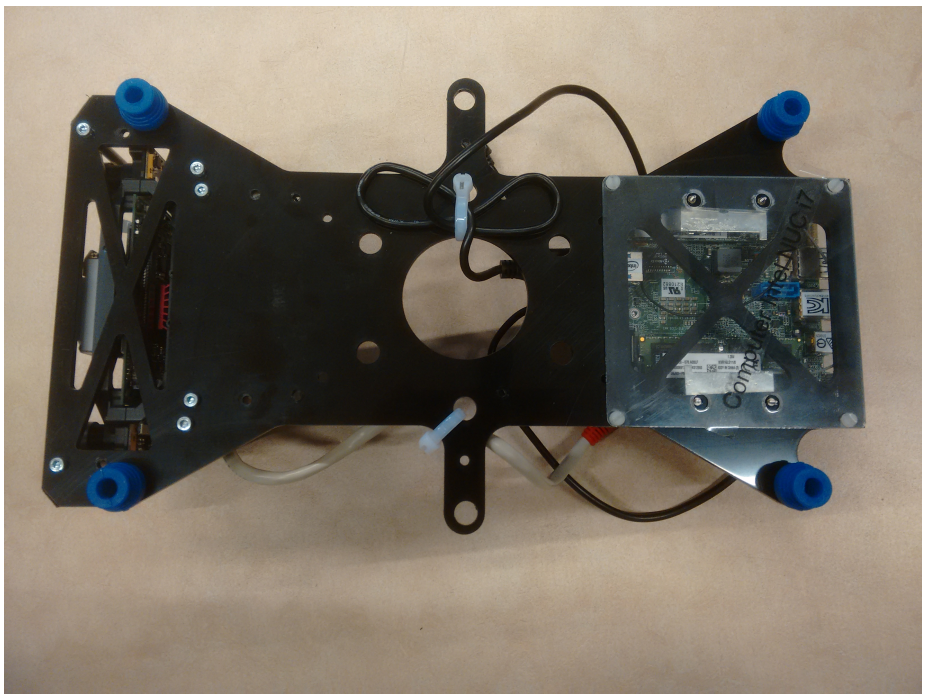

Figure B.1: Mount for VI-sensor and Intel NUC

On the left side of the image the VI-sensor can be recognized and on the right side the Intel NUC can be recognized. The VI-sensor is put at the front of the UAV. The mount is connected to the UAV with the four blue dampers. These dampers reduce the vibrations of the motors on the mount.

B.2 Electronics

A few components on the UAV need to be powered. That is the Pixhawk, the VI-sensor, the Intel NUC and the motors. The Pixhawk came with its own power module which ensured correct powering of the Pixhawk. The following table lists the voltage specifications of the different components.

Component	Voltage
VI-sensor	10-13 V
Intel NUC	12-19 V
Motors	6-16.8 V

The system is powered by a LiPo battery. The LiPo battery is a series of LiPo cells which determine the voltage. The following table lists the voltage provided by different LiPo batteries.

Number of cells	Minimum voltage	Maximum voltage
1S	3.2	4.2
2S	6.4	8.4
3S	9.6	12.6
4S	12.8	16.8
5S	16	21

It can be seen that the Intel NUC requires a 4S battery, the motors can operate with a 2S, 3S or 4S battery. The VI-sensor requires a 3S battery. A 4S battery is chosen as power source. A voltage regulator is used to provide the correct voltage for the VI-sensor. The circuit of the voltage divider can be seen in figure B.2.

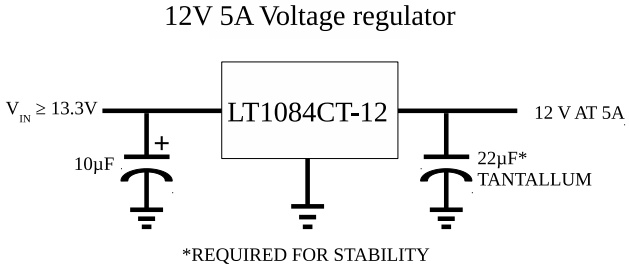

Figure B.2: Voltage regulator

The LT1085CT-12 is a 12V 5A voltage regulator which is commercially available. Two capacitors had to be added according to the datasheet.

Appendix C

Software setup

This appendix will elaborate the implementation of the software structure. A block diagram of the software setup is given in figure C.1. Different colors are used to distinguish the different elements of the software. The blue color represents the visual odometry. The red color represents the mapping. The green color represents the sensor fusion. The purple color represents the Pixhawk pose estimation. The yellow color represents the control algorithms and the grey color represents OptiTrack. How this software structure is implemented in ROS is given in figure C.2. The ellipses represent nodes and the rectangles represent topics. In the rest of this appendix every ROS node will be explained.

C.1 vi_sensor_node

The package `vi_sensor_node` interfaces with the VI-sensor. It will publish the camera streams of both the cameras on the topics `cam0/image_raw` and `cam1/image_raw`. It also provides a camera info message which contains intrinsic and extrinsic parameters of the cameras[1]. These messages are published together with every camera image.

Next to two cameras, the VI-sensor also has an IMU. The values of the IMU are published on the topic `imu0`. The relation between the IMU and the two cameras can be found in the topic `cam0/calibration` and `cam1/calibration`.

Sometimes it can happen that the sensor is connected to the PC and you can ping the sensor(10.0.0.1) but autodiscovery can't find the sensor. It is possible that the embedded software got locked due to unstable power at boot-up. The symptom would be that the FPGA LEDs continue to blink alternating while booting instead of having only one blinking LED when the sensor is ready. This can be solved by doing the following steps:

```
ssh root@10.0.0.1
mount -o remount,rw /
rm fpga/FPGA_BITSTREAM_BROKEN
reboot && exit
```

[1]Message definition can be found at http://docs.ros.org/indigo/api/sensor_msgs/html/msg/CameraInfo.html

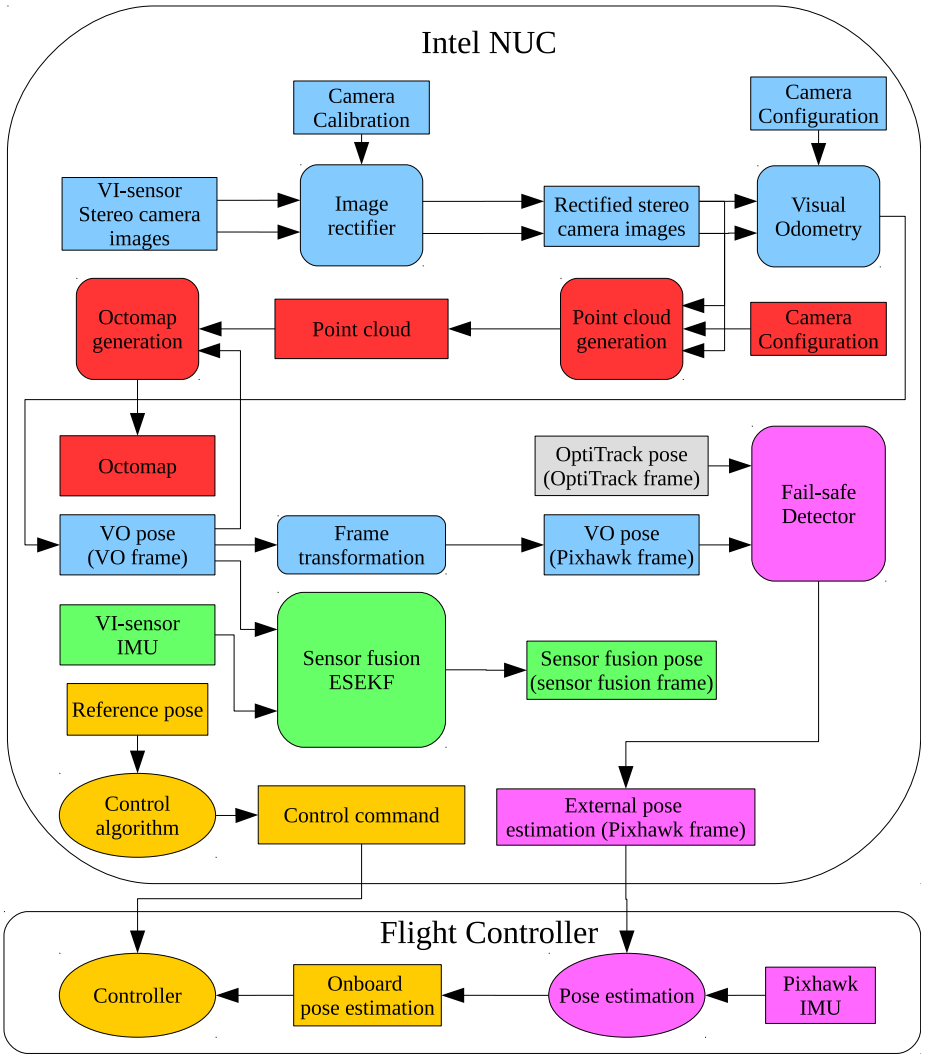

Figure C.1: Software setup

For other issues refer to https://github.com/ethz-asl/visensor_node/wiki/FAQ.

C.2 stereo_image_proc

The package `stereo_image_proc` performs image rectification of the stereo images and calculates the disparity between both image streams. For image rectification it needs the raw image streams and the intrinsic and extrinsic parameters located in the camera info message. The rectified images are published on `left/image_rect` and `right/image_rect`. For calculating the disparity the node uses OpenCV's block matching algorithm. The node will also generate a point cloud which can be used to create a map.

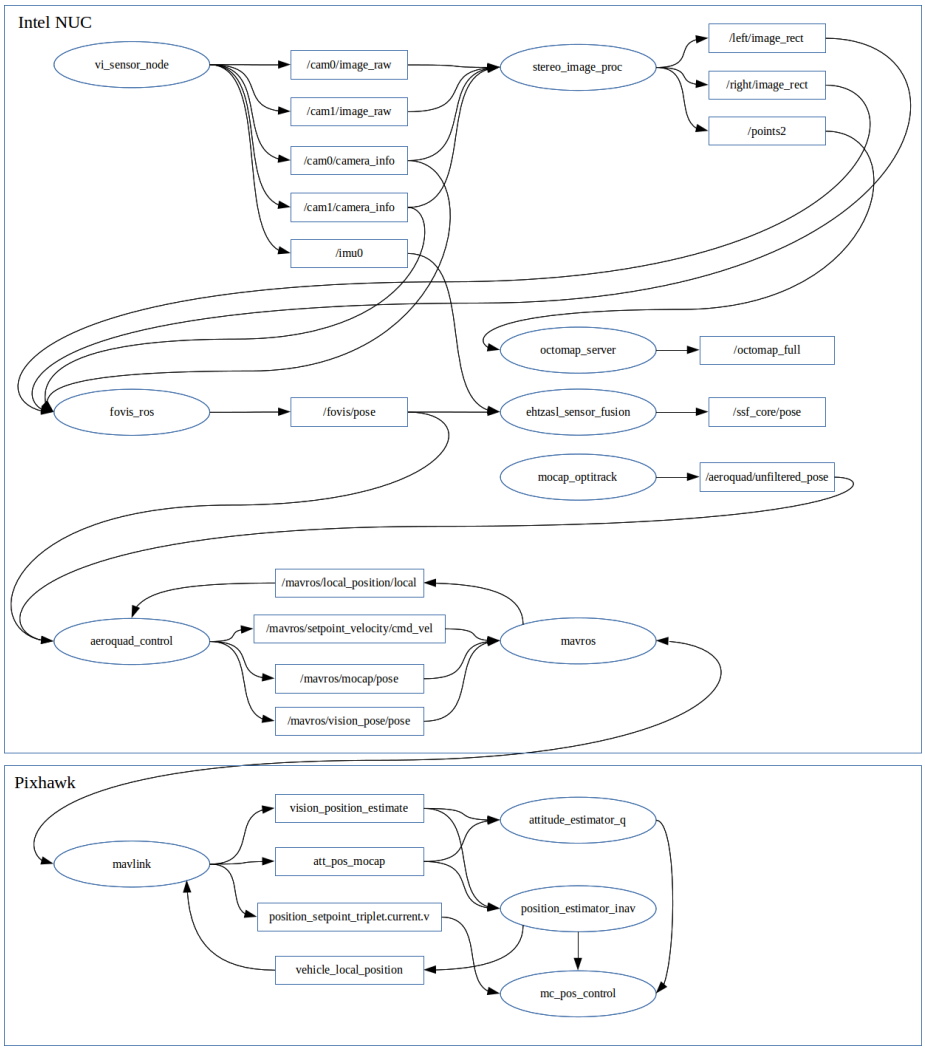

Figure C.2: Software setup

C.3 fovis_ros

The package `fovis_ros` is the ROS implementation of the FOVIS algorithm. It requires the rectified stereo images and the camera info message to calculate the visual odometry. It will publish the odometry on the topic `fovis/odometry`. The pose is published on `fovis/pose`.

C.4 ethzasl_sensor_fusion

The package `ethzasl_sensor_fusion` fuses the visual odometry pose with the IMU values. The output of this sensor fusion is a pose which is published on `ssf_core/pose`.

The package is divided in two main parts: `ssf_core` and `ssf_updates`. The core of the EKF is meant to be unchanged. The updates part is where the sensor specific files are located. Inside the `src` folder of the package three files have to be created: `vi_sensor_measurements.h`, `vi_sensor_sensor.cpp` and `vi_sensor_sensor.h`.

The file `vi_sensor_measurements.h` provides the initialization of the EKF. The files `vi_sensor_sensor.cpp` and `vi_sensor_sensor.h` implement the measurement callback. Inside this function the H-matrix, measurement residual and measurement noise covariance matrix are determined and used to process the measurement such that the core can correct the state vector using this measurement.

A configuration file is used which contains information about the setup. This file `vi_sensor_fix.yaml` contains the noise parameters of the IMU, the transformation between camera and IMU, the initial orientation of the vision frame with respect to the world frame.

The scale of a stereo camera is always 1 so the scale is initialized on 1 and kept fixed. The bias has to be estimated by the filter so is not fixed. The IMU-camera calibration is known and is therefor fixed. Here the noise of the IMU can be provided for the sensor fusion. The following noise values are used:

```
noise_acc: 0.022563
noise_accbias: 0.00008
noise_gyro: 0.004
noise_gyrobias: 0.000003
```

The noise on the scale, q_w^v, q_c^i and p_c^i are all zero. The delay of the measurement is also set to zero. The measurement noise is given as the standard deviation. The measurement noise of the position is set at $\sigma_p = 0.01$ m. The measurement noise of the orientation is set at $\sigma_\theta = 0.02$ rad.

The IMU-camera calibration can be obtained from the `cam0/calibration` topic. The initialization the orientation of the world frame in the vision frame is $q_w^v = \begin{bmatrix} 0 & 0 & \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \end{bmatrix}^T$.

C.5 mocap_optitrack

The package `mocap_optitrack` publishes Optitrack data which is streamed by Natural Point Tracking Tools on the topic `aeroquad/unfiltered_pose`. The file `mavros_mocap.yaml` relates the ID's of the trackables to the topics on which the node should publish.

C.6 aeroquad_control

The `aeroquad_control` package is the offboard controller of the UAV. The controller was designed for AR.Drone Parrots. It has been adjusted to fly with Pixhawk controlled UAV's. It is a PD controller where the reference position

can be set through a graphical user interface. This reference position is compared with the local position which is obtained from the Pixhawk. The control output is a velocity command which is sent to the Pixhawk.

C.7 mavros

The package `mavros` creates a bridge between ROS and the Mavlink protocol. The package consists of multiple plugins which converts ROS topics into Mavlink messages and vice versa. For example to send the optitrack pose to the Pixhawk the pose can be published on the topic `mavros/mocap/pose` and the `mavros` package will take care that the pose is streamed to the Pixhawk. The `mavros` package is used to stream the velocity commands, the optitrack pose and the vision pose from the NUC to the Pixhawk and to stream the local position of the Pixhawk from the Pixhawk to the NUC.

C.8 octomap_server

The package `octomap_server` is used to create a map of the environment. This map is created in an octomap structure[4].

C.9 Pixhawk firmware

During the project a few changes have been made to the firmware. When the firmware was downloaded the external heading in the attitude estimator was not yet incorporated in the master version. Also there was a scaling problem in position setpoints when flying on velocity setpoints. These issues have been solved by the community and are now incorporated in the master version of the Firmware. The only difference between the local version and the master version should be the mavlink transmission rate of the local position of the UAV. This local position is needed in the control. The original firmware transmits the local position at 3 Hz. This is too low for control so it has been increased to 30 Hz. It should also be possible to set the stream rate through via Mavlink. If that is implemented would mean that our software can be operated on the original firmware of the Pixhawk. Another point of interest is that our implementation of the external attitude in the attitude estimator is slightly different than the implementation in the original firmware although the functionality is the same.

Appendix D

Coordinate frames

For control it is very important to have every coordinate frame well defined. This appendix will elaborate the coordinate frames which are used in the hexacopter setup.

D.1 Frame naming

Frames are often named by using North, East, South, West, Up and Down. The frame will be expressed using three letters which relate XYZ to three of the six directions i.e. NED means X is pointing North, Y is pointing East and Z is pointing Down. When a magnetometer is used in the setup the North has to correspond to the real North of the earth. In this setup no magnetometer is used so the North can be set arbitrarily. The ROS convention for a coordinate frame in relation to a body is X forward, Y left and Z up or NWU[2]. The Pixhawk coordinate convention is NED[9].

The different coordinate frames are showed in figure D.1.

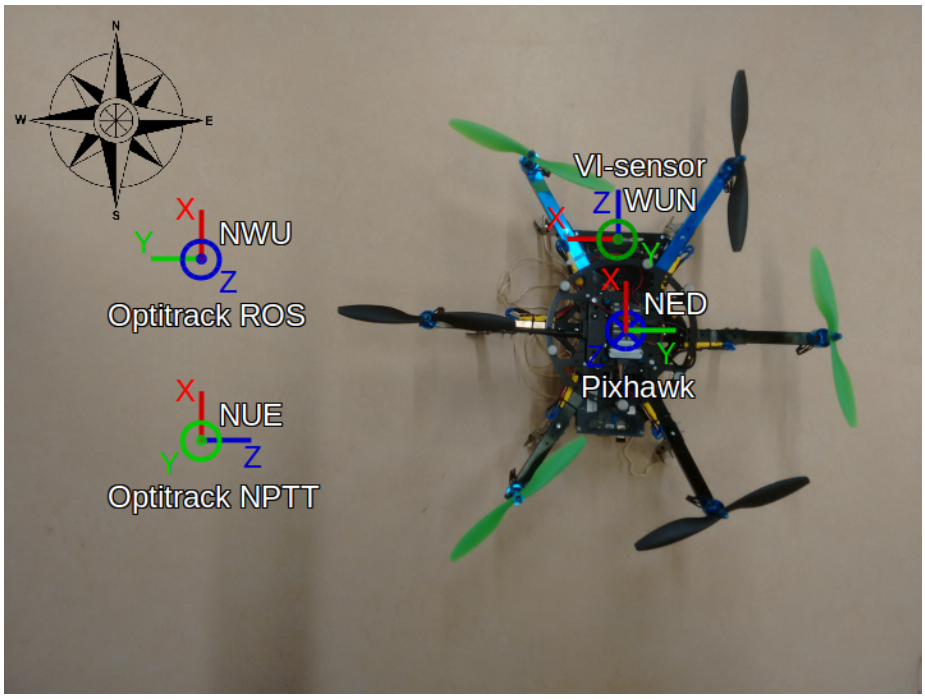

Figure D.1: Coordinate frames

The software of OptiTrack, Natural Point Tracking Tools, uses another convention in its graphical user interface. The coordinate frame is presented there as showed in figure D.1 by Optitrack NPTT. The Optitrack ROS frame is the frame as it is represented by the ROS mocap node. Note that on the image the origin of Optitrack ROS and Optitrack NPTT is not drawn at the same location but in reality the origin is in the same location. In NPTT a trackable can be created from the Optitrack markers. The orientation of the UAV during this initialization is defined as the unity rotation. It is necessary that this trackable is oriented such that the X-axis of the trackable coincides with the X-axis of the Pixhawk as shown in the image. The mavros package will convert the ROS coordinate frame to the Pixhawk coordinate frame. This is done using the function `UAS::transform_frame_enu_ned`. This function will do a frame transformation which is a 180 degrees rotation around the x-axis. The naming of this function is not very convenient since the transformation it really does is a NWU-NED transformation.

Bibliography

[1] Zheng Fang and Sebastian Scherer. Experimental study of odometry estimation methods using rgb-d cameras. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, September 2014.

[2] Tully Foote and Mike Purvis. Standard units of measure and coordinate conventions. http://www.ros.org/reps/rep-0103.html, December 2014.

[3] Andreas Geiger, Julius Ziegler, and Christoph Stiller. Stereoscan: Dense 3d reconstruction in real-time. In *IEEE Intelligent Vehicles Symposium*, Baden-Baden, Germany, June 2011.

[4] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013. Software available at http://octomap.github.com.

[5] Albert S. Huang, Abraham Bachrach, Peter Henry, Michael Krainin, Daniel Maturana, Dieter Fox, and Nicholas Roy. Visual odometry and mapping for autonomous flight using an rgb-d camera. In *Int. Symposium on Robotics Research (ISRR)*, Flagstaff, Arizona, USA, Aug. 2011.

[6] Peter S. Maybeck. *Stochastic models, estimation, and control*, volume 141 of *Mathematics in Science and Engineering*. 1979.

[7] Moritz Menze and Andreas Geiger. Object scene flow for autonomous vehicles. In *Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. http://www.cvlibs.net/datasets/kitti/eval_scene_flow.php?benchmark=stereo.

[8] Janosch Nikolic, Joern Rehder, Michael Burri, Pascal Gohl, Stefan Leutenegger, Paul T Furgale, and Roland Siegwart. A synchronized visual-inertial sensor system with fpga pre-processing for accurate real-time slam. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 431–437. IEEE, 2014.

[9] Pixhawk. Frames of reference. https://pixhawk.org/dev/know-how/frames_of_reference.

[10] Joan Solà. Quaternion kinematics for the error-state kf. Technical report, Institut de Robòtica i Informàtica Industrial of Technical University of Catalonia and Spanish Council for Scientific Research, 2015.

http://www.iri.upc.edu/people/jsola/JoanSola/objectes/notes/kinematics.pdf.

[11] Nikolas Trawny and Stergios I. Roumeliotis. Indirect Kalman filter for 3D attitude estimation. Technical Report 2005-002, University of Minnesota, Dept. of Comp. Sci. & Eng., March 2005.

[12] Stephan M. Weiss. *Vision Based Navigation for Micro Helicopters.* PhD thesis, ETH Zrich, 2012. http://e-collection.library.ethz.ch/eserv/eth:5889/eth-5889-02.pdf.