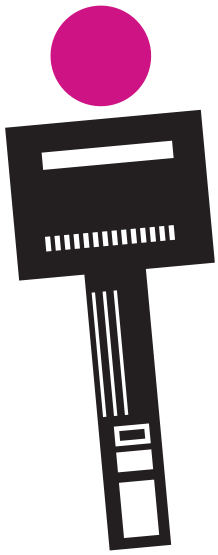


RAM ROBOTICS AND MECHATRONICS

Geometric state estimator tightly-coupling force and pose estimation for interaction in vision-denied environments

M. (Michiel) Bongertman

MSC ASSIGNMENT

Committee:
A. Franchi, Ph.D
R.A.M. Rashad Hashem, MSc
dr. D. Bicego
dr. ir. R.G.K.M. Aarts

September, 2020

038RaM2020
Robotics and Mechatronics
EEMathCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

UNIVERSITY OF TWENTE. | TECHMED CENTRE UNIVERSITY OF TWENTE. | DIGITAL SOCIETY INSTITUTE

Abstract

In recent years, UAVs showed promising capabilities for different applications. In the public, UAVs proved to be useful in applications such as logistics and inspection. As new advancements are being made by the robotics community, UAVs can also be used for interaction tasks. For such tasks, the employment of UAVs establishes 'safe for humans' working environments in critical operations. Examples of such physical tasks and environments are urgent or regular maintenance tasks on wind turbines, assistance at nuclear plants in case of a catastrophe, safety assessments of newly-blast generated voids into mines, or even assistance during natural disasters such as earth-quakes.

To accomplish stable and controlled interaction, both a controller guaranteeing stability and knowledge about the interaction is needed. The interaction force and torque could be measured using a sensor. However, this limits the applicability by size, costs, and weight, which consequently limits the battery time. Besides, force and torque sensors can only measure contacts at the point where they are mounted.

A common alternative is to use observers based on the nonlinear dynamics of the robot. This allows for estimating external forces and torques. Nonetheless, these methods only work well for practical situations if the forces are large and the noise is small. State estimators suffer less from this problem as they are designed to consider process and measurement noise.
Recently a work has been published which proposes to tightly-couple dynamics and pose estimations into a state estimator. Intuitively, this adds information to the pose estimation resulting in an accuracy increase for the estimated pose.

This thesis proposes a tightly coupled pose wrench estimator intended for physical interaction using an aerial robot. The state estimator can estimate the pose and external force by using known actuation inputs and the robot's dynamics. In previous work, the benefits of such an approach have been shown for disturbances instead of controlled contact interactions. This thesis uses the estimated external force for interaction scenarios where a UAV engages in physical contact with its environment.
Furthermore, the state estimator exploits measurements from a Global Positioning System (GPS) and Inertial Measurement Unit (IMU). Also, the estimator is able to estimate states belonging to the Lie group SO(3). This makes the state estimator geometric, thus invariant of changes of inertial frame. Besides, parametrization of orientation in SO(3) avoids the well-known singularity called gimbal lock.

The estimator is validated through realistic Gazebo simulations and a dataset of an experiment where a UAV physically interacts with a static object. The estimator has been tuned and estimations have reached an accuracy in the same order of magnitude as the simulated accelerometer and gyroscope. Also, the simulations and experiments show that the UAV is capable of accurately estimating the contact force while simultaneously estimating the UAVs pose. Recommendations on future work are to integrate Visual Inertial Odometry (VIO) to realize GPS-Aided VIO and to make use of an automated tuning scheme, as the quality of the estimation is highly dependent on the set of tuning parameters given to the estimator.

Acronyms

DOF Degree of freedom
EKF Extended Kalman filter
ECEF Earth-centered earth fixed coordinate frame
ECI Earth-centered inertial coordinate frame
IMU Inertial measurement unit
GNSS Global navigation satellite system
GPS Global positioning system
GUI Graphical User Interface
NED North East Down coordinate frame
NGDC Nation Geophysical Data Center
MoCap Motion-capture system
ROS Robot Operating System
RMS Root Mean Square (error)
SLAM Simultaneous localization and mapping
UAV Unmanned aerial Vehicle
UKF Unscented Kalman filter
VIO Visual inertial odometry
VO Visual odometry
V-SLAM Visual simultaneous localization and mapping
WGS-84 World geodetic system 1984
WMM World Magnetic Model

Nomenclature

$\bar{(.)}$	Raw measurement
$\breve{(.)}$	Predicted variable
$\hat{(.)}$	Estimated variable
$\mathcal{M}$	General Manifold
$\mathcal{X}$	State belonging to a Manifold $\mathcal{X} \in \mathcal{M}$
ω	Angular velocity $[\frac{rad}{s}]$
ψ^b	Body fixed frame, defined in the center of mass of the UAV
ψ^i	Inertial- or world fixed frame
a	Linear Acceleration $[\frac{m}{s^2}]$
$f(x, u)$	State function, denotes a non-linear system model used by the state estimator to predict the next state
$h(\breve{x})$	Observation mapping, function used to map states to the observation space of the corresponding observation
m	Magnetic field
u	Measurement input, sensor measurement used in prediction step of state estimator
x	State vector in real space $x \in \mathcal{R}^n$
y	Observation vector, sensor measurement used in correction step of state estimator

Contents

Chapter 1

Introduction

In past years Unmanned Aerial Vehicles (UAVs) have gained lots of attention due to their high flexibility. UAVs have been used in logistics to deliver packages and medicine and they have also shown promising capabilities for inspection. For example, they can be used to patrol areas such as harbors or help to find wildfire hot-spots.
Recently, UAVs are also starting to be employed for applications that involve physical contact of the aerial robot with its environment. Interactive UAVs can be of significant assistance, as they can perform physical tasks remotely with a maximum range and minimum reaching time. Besides, the employment of UAVs establishes 'safe for humans' working environments in critical operations [David et al., 2017]. Examples of such physical tasks and environments are urgent or regular maintenance tasks on wind turbines, assistance at nuclear plants in case of a catastrophe, safety assessments of newly-blast generated voids into mines, or even assistance during natural disasters such as earth-quakes.

Nevertheless, the employment of UAVs for physical contact interaction remains very challenging considering that aerial robots can not rely on a fixed-base but have to deal with a floating one. Additionally, most multi-rotors suffer from inherent under-actuation complicating their control, since exerting horizontal forces can only be accomplished with additional tilting motion.
However, in recent years controllers are emerging that allow exerting contact forces while also achieving stable flight during interaction. In many of these control schemes, the controller uses a combination of motion control and force/torque control to achieve a controlled interaction. To close the loop, knowledge has to be collected about how much force the UAV exerts into its environment. Naturally, one could use force and torque sensors to obtain knowledge about the interaction. However, integrating such sensors for controlled interactions comes at the expense of size, costs, and weight which consequently limits the battery time.
A more sophisticated solution would be to use an algorithm estimating the forces and torques between the UAV and the object of interaction. This also comes at the benefit of achieving a more versatile solution, as estimators can estimate general external forces and torques, instead of contact forces and torques at the point where the sensor is mounted.

Recently, a work has been published [Nisar et al., 2019] which proposes to tightly-couple wrench estimation with pose estimation for scenarios with interactions, contacts, or disturbances. This method allows both pose estimation as the perception of external forces acting on the aerial robot.
Commonly, external forces are recovered using loosely-coupled estimators in conjunction with an odometry system. Adding dynamics to pose estimation using a tightly coupled state estimator aids both the odometry and the external force estimation. Intuitively, the knowledge from the external forces adds information to the estimator, leading to an increase in odometry

accuracy [Nisar et al., 2019].
Also, loosely-coupled estimators estimate external forces based on a deterministic formulation. This results in that these approaches only work well when the forces are large and the sensor noise is small [McKinnon and Schoellig, 2016]. State estimators are designed to properly take process noise and measurement noise into account. Therefore, state estimators will be able to adequately handle noisy measurement when estimation external forces.

Furthermore, this thesis will give special attention to the sensor suite used by the state estimator. In general, UAVs contain Inertial Measurement Units (IMUs), which are cheap, lightweights sensors capable of making accurate measurements. However, using them for pose estimation as stand-alone inevitably leads to drift caused by integration. A common solution is to use them side-by-side with sensors measuring positional information.
A common choice is to fuse IMU measurements with Visual Odometry, achieving Visual Inertial Odometry (VIO). Global Positioning System (GPS) signals are usually ignored as GPS signals are notorious to be lost during operation. Besides, GPS is not available indoors or in dense urban environments, so-called GPS-denied environments.
Nevertheless, UAVs can operate in a wide range of environments, which also poses challenges to VIO algorithms, such as drastically varying lighting conditions, uneven illumination, low texture scenes, and abrupt changes in attitude due to wind gusts or aggressive maneuvering [Sun et al., 2018]. These conditions ask for a robust VIO algorithm.
Though GPS is prone to signal losses, it is freely available in outdoor environments. Considering GPS in conjunction with VIO will increase the robustness of the solution, especially in environments challenging for vision-aided estimation algorithms. Therefore, this thesis will work towards GPS-aided VIO.

1.1 Related Work

This section will review two fields related to this thesis. First, advancements in the field of pose estimation for flying robots will be presented. Next, solutions able to estimate external forces and torques are shown. Later on, both the topics will come together. Both subsections will first examine the literature and next summarize the works using a table.

1.1.1 Pose estimation

The aim of [Duflos et al., 2006] is to develop a GPS/IMU multisensor fusion algorithm for unmanned land vehicles. The method achieves the estimation of both position and velocity using a Kalman Filter. Next to this, the algorithm considers so-called 'contextual information' that can be used to favor certain measurements or minimize the importance of others depending on the context. This method increases the reliability in case of bad data or signal loses. Besides, the algorithm is presented for GPS and IMU fusion, but it is able to integrate a high number of sensors.
Another work using IMU and Global Navigation Satellite Systems for position estimation is [Grip et al., 2012]. Instead of only estimating position and velocity, the authors present an observer that can also estimate orientation and gyroscope biases. The observer represents the estimated orientation by a rotation matrix $R \in SO(3)$, which prevents well-known obstructions caused by Euler angles.
The observer has been verified in simulation, but has not been tested for potential errors such as: accelerometer bias, magnetic disturbances and GNSS failure. The latter is a well-known

drawback of GNSS-based estimation. GNSS measurements, such as GPS, are notorious for signal losses or their unavailability in so-called GPS-denied environments (specifically indoors or in dense urban environments).

To avoid the unreliability of GPS, [Mourikis and Roumeliotis, 2007] proposes to use visual Inertial Odometry (VIO). Their method, called the multi-state Constraint Kalman Filter (MSCKF), is one of the first VIO algorithms to achieve real-time vision aided inertial navigation. The algorithm uses an EKF where the observation model can express the geometric constraints that appear when a static feature is observed from multiple camera poses.
In [Li and Mourikis, 2012] an observability analysis was performed and it was shown that the standard way of computing Jacobians in the filter inevitably results in inconsistency and therefore a loss of accuracy. The performance of the MSCKF was improved by ensuring correct observability properties without incurring additional computational cost.
Though the MSCKF is a relatively more outdated VIO algorithm, the method still achieves a respectable accuracy and computation time, as shown in [Delmerico and Scaramuzza, 2018]. Furthermore, the MSCKF forms the basis for many of the modern VIO algorithms.

The work [Lynen et al., 2013] presents a generic framework, dubbed the Multi Sensor-Fusion Extended Kalman Filter (MSF-EKF). The MSF-EKF is able to process delayed, relative and absolute measurements from a theoretically unlimited number of different sensors and sensor types while allowing self-calibration of the sensor-suite online.
The modularity of this sensor fusion algorithm allows seamless handling of additional/lost sensor signals during operation. Due to the modularity of MSF-EKF, GPS can still be considered even in GPS-denied environment. This comes with the benefit of more robustness and accuracy.

In [Bloesch et al., 2015] a highly robust monocular VIO is achieved by using pixel intensity errors of image patches directly. After detection, the tracking of the multilevel patch features is closely coupled to an EKF by using the intensity error as innovation term. The method achieves accurate tracking performance and high robustness, specifically in difficult situations with very fast motions.
The work [Sun et al., 2018] also considers robustness for VIO. The authors acknowledge that UAVs operate in a wide range of environments that pose challenges to VIO algorithms, such as drastically varying lighting conditions, uneven illumination, low texture scenes, and abrupt changes in attitude due to wind gusts or aggressive maneuvering. To increase the robustness of VIO for these challenging environments, the authors propose to use a stereo-vision configuration. Besides, the stereo-vision VIO is achieved using filtering-based estimation, as these approaches are generally much more efficient over optimization-based algorithms.

Opposed to the previously called algorithms, the work [Brossard et al., 2018] presents a Unscented Kalman Filter (UKF) implementation of VIO. The method is based on a stereo configuration of the MSCKF which is called the S-MSCKF. The use of the unscented transform, in the state estimator, spares the computation of Jacobian. This makes the method more versatile and allows for fast prototyping.
Next to this, the method makes use of a UKF on Lie groups, which was presented earlier in [Brossard et al., 2017]. Instead of making use of quaternions, as the formerly seen algorithms do, the UKF on Lie groups allows VIO to estimate orientations expressed as rotation matrix.

Nonlinear optimization is able to estimate states while retaining high accuracy. However, real-time optimization quickly becomes infeasible as the trajectory grows over time. This problem gets even worse since IMU measurements come in at a high rate, which leads to a fast growth

of the number of variables in the optimization.
In [Forster et al., 2016] this issue is addressed by pre-integrating inertial measurements between selected keyframes into a single relative motion constraint.
This paper brings the preintegration theory to maturity by properly addressing the manifold structure of the rotation group.
Furthermore, it is also shown that the preintegrated IMU model can be seamlessly integrated into a visual-inertial pipeline, under the unifying assumption of factor graphs. The method is evaluated on simulated and real data-sets. The evaluation shows that the method leads to accurate state estimation in real-time.

A VIO state estimator that uses the pre-integration technique is called VINS-MONO [Qin et al., 2018]. VINS-MONO achieves highly accurate VIO by using a tightly coupled, non-linear optimization-based method that fuses pre-integrated IMU measurements and feature observations. The approach is said to be reliable, complete, and versatile for different applications that require high accuracy in localization.
In [Nisar et al., 2019] it is acknowledged that the external forces and torques can also serve a purpose in VIO through additional information. The work adds the robot's dynamics to a VIO estimation problem effectively providing more information for the pose estimation, which increases the accuracy of the odometry. They apply the theory of preintegration [Forster et al., 2016] to formulate a tightly coupled, sliding-window estimator suitable for real-time applications. The additional external force has been implemented in the VIO pipeline VINS-MONO [Qin et al., 2018]. It was shown that the tightly-coupled approach increases the accuracy of the estimator up to 29 % as compared to the original pipeline VINS-MONO.

Table 1.1: Overview literature pose estimation. First column cites the publication, second column denotes the sensor used in conjunction with the IMU, third column denotes what estimation algorithm is used, fourth column denotes how orientation is expressed.

Publication	Sensor Suite (excl. IMU)	Estimation Algorithm	Orientation parameterization
[Duflos et al., 2006]	GPS	KF	-
[Grip et al., 2012]	GPS	Observer	SO(3)
[Li and Mourikis, 2012]	VIO	EKF	SU(2)
[Lynen et al., 2013]	Fusion	EKF	SU(2)
[Bloesch et al., 2015]	VIO	UKF	SU(2)
[Sun et al., 2018]	VIO	EKF	SU(2)
[Brossard et al., 2018]	VIO	UKF	SO(3)
[Qin et al., 2018]	VIO	Optimization	SO(3)

1.1.2 Force and torque feedback for interaction

This subsection looks into how force and torque feedback is achieved in the literature for physical interacting using UAVs. The methods encountered in the literature can be organized in three categories. The first are direct methods, which use force and torque sensors to directly measure contact forces and torques. The second category of indirect methods, exploit the UAV's dynamics in order to estimate external forces and torques. After estimation, the external forces and torques are used as a measure for the interaction. Next to this, methods combining force and torque sensors with estimation algorithms will be examined. In such cases, the sensors measure the interaction forces and torque and the algorithm is used for other purposes.

Instead of utilizing force and torque sensors, the work [Yüksel et al., 2014] proposes a non-linear force observer that can be used as indirectly to estimate contact forces and torques. A non-linear wrench observer is based on the Langrangian dynamics of a quadrotor. The estimated force is exploited to achieve contact interaction. The force observer gives a good approximation if the external forces are not rapidly varying.
A benefit of the indirect force feedback is, that the avoidance of force and torque sensors attached to the UAV reduces equipment costs and weight carried by the robot. Besides, a sensor can only measure the force/torque applied to the point it is mounted in, while the observer is able to estimate general external contacts of the UAV.
Another work exploiting estimated force feedback is [Ryll et al., 2017]. Instead of using an underacted quadrotor, the paper presents a fully-actuated UAV intended for physically interactive tasks. By utilization of a tilted propeller design, full-actuated is achieved. The full-actuation allows full and dexterous 6D force control. The control is achieved by, using pose estimations, from a state estimator, in conjunction with a momentum-based wrench observer. A benefit of this system is that due to the full-actuation the UAV can counteract any wrench during the contact with the environment

Another work [Tomić and Haddadin, 2014] uses external wrench estimation for collision detection. The authors make use of a hybrid estimation method combining acceleration-based estimation with momentum-based estimation. The acceleration-based estimation allows to estimate translational velocity in a drift-less manner. However, for the rotational domain, angular acceleration can only be obtained through numerical differentiation. Since this is undesired, a momentum-based estimation is used for the rotational domain in conjunction with the acceleration-based estimation for the translational domain.
This work also realizes a limitation of indirect force/torque feedback. The estimated external wrench will not only consist of contributions due to the interaction, but it will also contain modeling errors and disturbances, such as aerodynamic effects. This problem is addressed in [Tomić and Haddadin, 2015], where the collision detection is extended by aerodynamic models. The models allow to simultaneously estimate aerodynamic forces and contact forces online. The discrimination between the two is achieved by identifying the natural contact frequency characteristics for both "interaction" cases. This can be used to filter the external wrench and separate it into the contribution by the collision and by aerodynamic effects.
They mention that collisions show high frequent effects and the aerodynamic forces are mostly dominant in the low frequency area. Because of this, the discrimination of the two can be achieved with well-designed filters. However, controlled contact interactions are not exclusively dominated by high frequent behavior. This means that for those scenarios, the discrimination might not be as simple as for collisions.
In [Rajappa et al., 2017], the momentum-based estimation from [Tomić and Haddadin, 2014] is used in conjunction with a sensor. The authors deploy a sensor ring to separate human contributions from disturbances, such as wind and parameter uncertainties. This allows the system to reject disturbances that are not simply measured by sensors, while also being able to let humans change the desired trajectory by simply applying forces on the UAV.
The inability to discriminate interaction forces is also acknowledged in [Nava et al., 2020]. The work presents an optimization-based method for controlling aerial manipulators in physical contact with the environment. The authors mention that indirect feedback methods, exploiting the robot dynamics, are prone to errors in case of parameter uncertainties. Furthermore, when the system is affected by external disturbances, it might not be possible to discriminate them from interaction forces. These drawbacks are resolved by using an onboard force/torque sensor mounted at the end effector. Naturally, this comes at the price of weight, size and equipment costs, but when using an onboard manipulator this is already of consideration.

In some previous works, the estimation of interaction forces has been accomplished by utilizing non-linear observers. This works well in cases where forces are large and sensor noise is small [McKinnon and Schoellig, 2016]. Otherwise, inputs and outputs of the nonlinear observers must be carefully filtered, as the algorithm is based on a deterministic formulation that does not account for sensor noise or process noise. It is mentioned that nonlinear stochastic state estimation algorithms are designed to properly handle sensor and process noise. Therefore, they present an external force and torque estimator which specifically takes sensor noise and model imperfections into account. This is achieved by using an Unscented Kalman Filter (UKF). The force estimations have been used in conjunction with an admittance controller to enable a quadrotor to hold position relative to a wind source.

Table 1.2: Overview literature force/torque feedback. First column cites publication, second column denotes how feedback is achieved, third column denotes UAV actuation and fourth column for what interaction forces method is used.

Publication	Force/Torque feedback	UAV actuation	Sensed Signal
[Gioioso et al., 2014]	Off-board sensor	Underactuated	Physical contact
[Wopereis et al., 2017]	Off-board sensor	Underactuated	Physical contact
[Yüksel et al., 2014]	Observer	Underactuated	Physical contact
[Ryll et al., 2017]	Observer	Fully actuated	Physical contact
[Tomić and Haddadin, 2015]	Observer	Underactuated	Collisions and Aerodynamic
[Rajappa et al., 2017]	On-board sensor & observer	Underactuated	Human contact and disturbances
[Nava et al., 2020]	On-board sensor	Fully actuated	Physical contact
[McKinnon and Schoellig, 2016]	State estimator	Underactuated	Wind
[Nisar et al., 2019]	State estimator	Underactuated	Disturbances

Table 1.2 compares how force and torque feedback is achieved in the reviewed literature. It can be seen that none of the encountered works use a state estimator to engage in physical contact interaction with a fully actuated UAV. Works that estimate contact forces and torques mostly use deterministic formulated observers, which only works well in cases of high signal-to-noise ratios. As seen in [McKinnon and Schoellig, 2016], stochastic formulations, such as state estimators, are designed to properly handle measurement noise and process noise.
The works considering state estimators for estimating external forces and torques, present their work for wind disturbances or programmatically introduced disturbances. However, neither of them engage in physical contact with the environment. Besides, the nonlinear dynamics of a quadrotor have been exploited. Fully actuated aerial robots can exert a full wrench without the need to tilt. This is beneficial for contact interaction, as horizontal forces can easily be exerted into objects and surfaces.

1.2 Research Goal

This thesis aims to design a state estimator which tightly couples a fully-actuated hexarotor's dynamics into pose estimation. Besides, the estimations should comply with the UAV's nonlinear geometric structure. Therefore, the estimation problem should be formulated geometrically

and the state estimator should be able to deliver estimates which are subject to the mathematical constraints of their underlying Lie group. Furthermore, the performance of the tightly coupled state estimator will be validated for an interaction scenario. Additionally, special attention will be given to the effects of a loss of signal. The goals of this thesis result into the following research questions:

1. How can force estimation be tightly-coupled with pose estimation for a geometric state estimator?
2. What are the limitations of external force estimation for the tightly-coupled state estimator for physical contact interaction using a UAV?
3. How does a state estimator tightly-coupling force estimates into the pose estimation compare against a state estimator exclusively estimating the pose for physical contact interaction using a UAV?

Note that the aim of this research has similarities with [Nisar et al., 2019]. The most important difference between this thesis and the literature is that this thesis will consider the tightly-coupled estimation for physical interaction, instead of a self-introduced disturbance. It is much simpler to estimate a self-introduced disturbance, as the characteristics of the external force can be altered such that they are easily estimated. A real-world interaction however can be a more complicated signal more challenging to estimate. Thus, applying a tightly-coupled state estimator on a real-world interaction will give more insight into the true limitations of the method for realistic scenarios.

Moreover, this thesis proposes to utilize a GPS-aided VIO. The use of VIO remains challenging in environments with drastically varying lighting conditions, uneven illumination, low texture scenes, and abrupt changes in attitude [Sun et al., 2018]. Besides, GPS becomes freely available outdoors. The employment of GPS in addition to VIO will improve the overall robustness of the state estimator.

Also, VIO has four unobservable degrees of freedom (DOFs), namely the three DOFs corresponding to the global position, and one corresponding to the rotation about the gravity axis. Utilization of GPS for state estimation does achieve full-observability for the global position [Chagas and Waldmann, 2015]. Therefore, the incorporation of GPS will lead to improvements in the observability of the solution.

1.3 Proposed method

In this work, a state estimator tightly-coupling external force estimation with pose estimation will be proposed. To achieve this coupling, a model for the motion prediction will be formulated which specifically takes the forces acting on the body into account. Figure 1.1 shows an overview of the proposed state estimator. This section will first explain how the proposed method estimates both the pose and the external force. Next, the choices made for the proposed method will be elaborated.

The State estimator uses two measurement systems, the UAV's IMU, and a GPS. The IMU contains acceleration measurements $\bar{a}_b^{b,i}$ and angular velocity measurements $\omega_b^{b,i}$. The GPS measures the position of the UAV ξ_b^i and its velocity $v_b^{b,i}$.

In addition to these sensors, an Electronic Speed Controller is needed for the solution. As described in [Rashad et al., 2019a] by using each propeller's spinning velocity, the drag-to-thrust ratio and the internal configuration of all propellers, one can calculate the wrench inserted into the UAV by the controller via the propellers. In order to obtain knowledge about the propeller's spinning velocity, the ESC is needed.

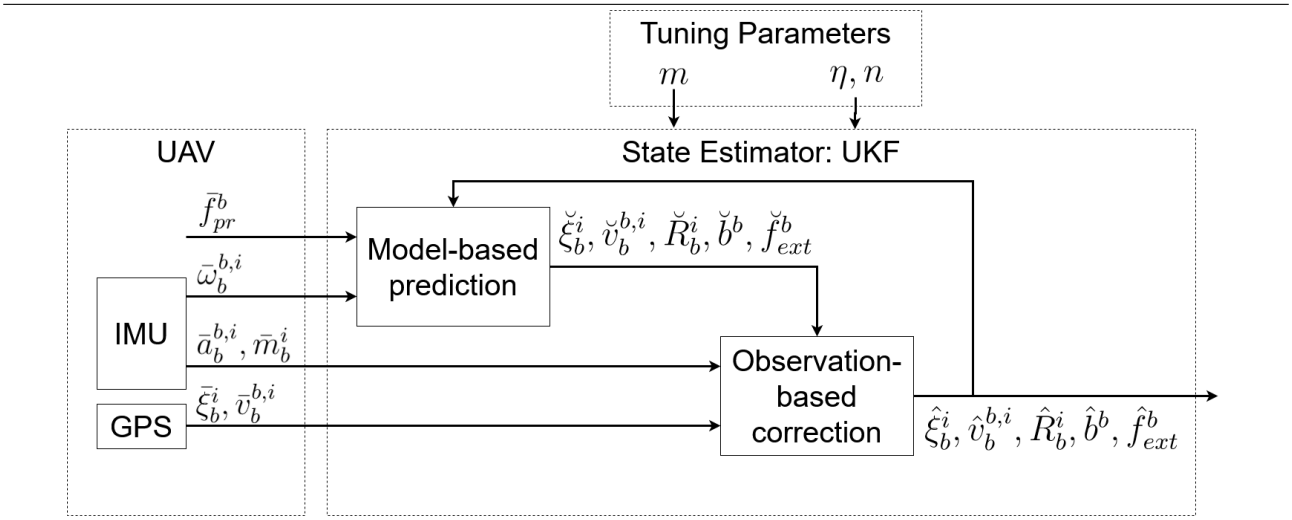
The proposed method uses an UKF to estimate both the UAV's pose and the external force acting on the body $\hat{f}_{ext}^b$. The UKF uses a system model to predict each state and it corrects

the states using observations obtained from sensors. The external force can be predicted by using a model and it can be correction by mapping the forces acting on the body towards the expected acceleration. Subsequently, $\hat{f}^b_{ext}$ and $\bar{f}^b_{pr}$ will be used to predict the pose of the UAV. This method achieves a state estimator where the external force estimation is tightly-coupled with the pose estimation. Next to the tightly-coupled estimator, a estimator exclusively estimating the robot's pose will be formulated as well. The benefits of having two estimators are twofold. First, the geometrical pose estimator will contain a very similar approach, but without the external force estimation, allowing to more easily verify the estimator. Besides, the geometrical pose estimator allows to compare with the estimator tightly-coupling the robot's dynamics.

The tightly coupled state estimator will be implemented by using a filtering-based estimator. Optimization-based estimators are more accurate but in general filtering-based estimators are computationally more efficient than competing optimization-based methods [Sun et al., 2018]. Besides formulating a filtering-based estimator provides an alternative research direction which can later be compared with [Nisar et al., 2019].
The filtering-based algorithm chosen is the UKF. The EKF is a bit more lightweight computationally but the UKF shows superior performance for highly nonlinear problems, as the EKF is only optimal up to first-order non-linearities. Besides, the UKF spares the error-prone computation of Jacobians which makes the method more versatile.

Next to this, the state estimator makes use of a GPS to correct the position. Most of the literature, such as [Nisar et al., 2019], uses VIO odometry to estimate the UAV's pose. However, as acknowledged by [Sun et al., 2018] using VIO is challenging in environments with drastically changing lighting conditions, uneven illumination, low texture scenes and, abrupt changes in attitude due to wind gusts. Their solution contains a stereo-vision setup for VIO, which complicates the UAV system design by size, weight, and computational costs. Besides, outdoors GPS is available freely. To not ignore GPS signals once they are available, this thesis proposes to use a GPS-aided VIO. However, to keep the work tractable for the time of a Master thesis, this thesis will use a GPS/IMU fusing as a first step.

Figure 1.1 General overview of the proposed state estimator. The 'UAV' block denotes signals needed from the aerial robot, and the 'Tuning Parameters' block denotes the parameters to be set by the user.



Another goal of the project is to obtain geometric estimations. To accomplish this, it is first needed to formulate the state estimator's models such that they comply with rigid body kine-

matics and dynamics. This causes the predictions, made by the state estimator, to be geometric. Secondly, a geometric state estimator outputs states belonging to a Lie group. For a state to be an element of a Lie group, it will be subject to certain mathematical constraints. Traditional state estimators will use operations that break the conditions imposed by the Group. Therefore, an alternative state estimator will be used which ensures states belonging to a Lie group will be estimated such that they satisfy the imposed constraints.

The purpose of this state estimator is to be used for aerial physical interaction applications. This can be accomplished by using the external force, estimated by the algorithm, as the interaction force and feeding the estimated interaction force back to the control scheme.
The interaction estimation can be achieved using the fully-actuated Hexarotor named BetaX, which can be seen in figure 1.2. BetaX uses an energy tank-based wrench/impedance controller to accomplish closed-loop position- and wrench control [Rashad et al., 2019b]. The concept of energy tanks is used to guarantee the system's overall contact stability to arbitrary passive environments. Besides, BetaX has a tilted-propeller design that achieves full-actuation. Under-actuated UAVs, such as quadrotors, need to tilt before moving horizontally. This complicates interactions as horizontal forces can only be exerted into the environment with the additional tilting. Fully-actuated UAVs don't suffer from this complication, as they can exert the full 6d wrench without the need for additional motion. Nevertheless, the fully-actuated UAV is employed solely to simplify the interaction. The proposed state estimator is independent of the full-actuation of the UAV.
To find the limitations of the tightly coupled state estimator a software implementation will be made in C++. The code will be executable as a ROS node such that a Gazebo simulation of BetaX can be used to experiment with.

1.4 Thesis Structure

The rest of this thesis will be organized as follows. Chapter 2 will give the background theory needed to understand the core chapters of this thesis. The chapter will be divided into two sections. The first section shares relevant basics on state estimation and the second section presents some mathematical concepts of Lie theory. Chapter 3 describes the used sensors and their models. Chapter 4 will use the concepts taught in chapter 2 to formulate operations that can be used to redefine state estimators. These operations will accomplish that the redefined state estimators can estimate states belonging to a Lie group. Chapter 5 will formulate two geometric state estimators. The first one fuses information from an IMU with a GPS. The second one also couples the UAV's dynamics into the estimator. In chapter 6 simulations are being done to validate estimator. Chapter 7 will experiment with the state estimator using (manipulated) real-world data. And lastly, chapter 8 will come to a conclusion.

Figure 1.2 Fully-actuated hexarotor hovering at a non-zero pitch and roll angle [Rashad et al., 2019a]

Chapter 2

Background

Sensor fusion is the process of combining measurements from sensors or algorithms into a combined estimation. Sensor fusion aims to improve measurement accuracy or precision. For instance, IMUs are typically very accurate but suffer from drift or bias if used for pose estimation. Therefore, these sensors are typically fused with measurement systems providing less accurate but bias-free pose information.

Sensor fusion can be achieved by state estimation. In general, two state estimation methods are considered: Filtering-based estimation or optimization-based estimation. Filtering-based methods make use of (non-linear) Kalman Filters. The states are predicted using models and corrected by observations. Optimization-based algorithms are mainly gradient-based, such as Newton's method or Gauss-Newton's method [Gui et al., 2015].
In general optimization-based estimation is more accurate but comes with higher computational demands. Filtering approaches are more suited to real-time applications [Brossard et al., 2018]. Nevertheless, methods to consider optimization-based estimation for real-time applications are emerging. One of these methods is called pre-integration [Forster et al., 2016].

One can recognize specific names denoting which sensor suite is used in a state estimator. Visual Odometry (VO) denotes the motion estimation of a system using only the input of a single or multiple cameras attached to it [Scaramuzza and Fraundorfer, 2011]. A fusion between IMU data and VO is called Visual Inertial Odometry (VIO). Sensor fusion can also be encountered in synchronous localization and mapping (SLAM) literature. The SLAM problem is concerned with estimating a robot's pose while also building a map of the environment. In SLAM, state estimation or sensor fusion strategies can be applied to improve the estimation of the robot's pose which consequently also improves the quality of the map. SLAM algorithms such as visual-SLAM (V-SLAM) can also be coupled in the state estimator. In such cases SLAM provides the robot's pose which can be used to correct biases and counteract drift caused by integration.

Depending on how the sensors or algorithms are fused, the fusion is either loosely-coupled or tightly coupled. For loosely-coupled systems, the incorporated estimators can estimate states as stand-alone, but the information is fused to improve the performance. In tightly-coupled estimators, the estimation in one step is dependent on variables estimated in another step.

In state estimation for robotics, typical states to be estimated are the robot's: position, velocity, orientation and the sensor bias. For states belonging to real space, the estimation can simply be achieved by traditional state estimation algorithms. However, orientation is parameterized on a manifold. This is done because parameterization in real space leads to singularities known as gimbal lock [Hertzberg et al., 2011]. States expressed on a manifold can not be handled

similarly to states expressed in real space. Therefore, redefined operations will be needed to correctly estimate the states expressed on a manifold.

The remainder of this chapter will be used to further explain filtering based techniques for state estimation and Lie theory.

2.1 State Estimation

A kalman filter is a discrete estimation process. The filter uses two steps to estimate the state vector $x_k \in \mathcal{R}^n$, where n denotes the number of states and k the current sample. The first step is to predict the state $\breve{x}_k$ by using knowledge of the system and measurement inputs. The second step is to correct the state prediction using observations obtained by sensors.

The knowledge of the system during the prediction can be obtained by a linear state space model:

$$x_k = Ax_{k-1} + Bu_k + \eta_k \tag{2.1}$$

Where x_{k+1} denotes the next state, A is the system matrix, u is a measurement input, B is the input matrix, and η_k is measurement noise. The measurement noise is injected via the input and can be modeled as additive zero mean Gaussian noise $\eta_k \sim \mathcal{N}(0, \sigma_\eta^2)$.

The correction step is formulated as follows:

$$y_k = Cx_k + n_k \tag{2.2}$$

Where $y_k \in \mathcal{R}^m$ denotes the observation vector with m observations,$C \in \mathcal{R}^{m \times n}$ is a noiseless observation mapping from the (predicted) states to the observations and n_k denotes observation noise coming from sensors used during the correction step.

The Kalman filter uses operations on vectors and matrices. Therefore, the noise will be formulated using covariance matrices. The covariance matrices are defined by the expectation $E(.)$:

$$\begin{aligned} Q &= E[\eta_n \eta_n^T] \\ R &= E[n_k n_k^T] \end{aligned} \tag{2.3}$$

Where Q is the measurement covariance matrix and R is the observation covariance matrix. Both covariance matrices are similar in the sense that both denote noise injected via sensors. However, Q is a measure for the sensor noise in the prediction step and R for the correction step.

Finally, the Kalman Filter uses the Kalman Gain $K \in \mathcal{R}^{n \times m}$ to estimate the value of the states:

$$\hat{x}_k = \breve{x}_k + K(y_k - C\breve{x}) \tag{2.4}$$

Where $\breve{x}$ denotes a predicted state. The Kalman gain is an optimal gain as it tries to minimize the process noise in the estimator. The Kalman gain and the equations of the filtering process are derived in appendix A.
Depending on whether equation (2.1) is formulated continuously or discrete, $\breve{x}$ is obtained with or without integration.

The process error e_k denotes the differences between the actual state and the estimated state: $e_k = x_k - \hat{x_k}$. The error can be used to define the process covariance matrix, which is a measure for the uncertainty in the estimation:

$$P = E[e_k e_k^T] = E[(x_k - \hat{x_k})(x_k - \hat{x_k})^T] = \begin{bmatrix} E\left[e_{k-1}e_{k-1}^T\right] & E\left[e_k e_{k-1}^T\right] & E\left[e_{k+1}e_{k-1}^T\right] \\ E\left[e_{k-1}e_k^T\right] & E\left[e_k e_k^T\right] & E\left[e_{k+1}e_k^T\right] \\ E\left[e_{k-1}e_{k+1}^T\right] & E\left[e_k e_{k+1}^T\right] & E\left[e_{k+1}e_{k+1}^T\right] \end{bmatrix} \tag{2.5}$$

In the last equality, it can be recognized that the process covariance matrix describes the expectation or variance between the previous- current- and next error sample.

The remainder of this section will be used to explain the general process of the Kalman Filter in 2.1.1. The Kalman Filter will be derived for a linear case. Naturally, there is no promise that models needed for real-world applications are linear. Therefore 2.1.2 will show some nonlinear alternatives which are commonplace in filtering literature. Lastly, some final matters will be explained regarding consistency and observability. These are topics frequently discussed in literature since they are associated with sufficient estimation accuracy.

2.1.1 Linear Kalman Filter

The Kalman Gain and the process covariance functions have been derived in appendix A.
The linear Kalman Filter needs the following equations to make (optimal) estimations: State prediction (2.1), Process Covariance prediction (A.13), Kalman gain (A.9), state correction (2.4) and process covariance update (A.10). This leads to the following algorithm:

Algorithm 1 Linear Kalman Filter $(x_{k-1}, P_{k-1}, u_k, y_k)$

1: $\breve{x}_k = Ax_{k-1} + Bu_k + \eta_k$
2: $\breve{P}_k = AP_{k-1}A^T + Q$
3: $K_k = \breve{P}_k C^T (C\breve{P}_k C^T + R)^{-1}$
4: $\hat{x}_k = \breve{x}_k + K_k(y_k - C\breve{x}_k)$
5: $\hat{P}_k = (I - K_k C)\breve{P}_k$
6: return $\hat{x}_k$,$\hat{P}_k$

The algorithm returns $\hat{x}_{k+1}$ and $\hat{P}_{k+1}$ which can be used as the next input. By doing so, the algorithm iteratively returns new estimates. Of course, the user should choose an initial x_k and P_k for the first iteration. The choice of P_k will influence the Kalman gain K. Inspection of the estimation equation (2.4) shows that when K is close to 1, the prediction cancels and the estimator puts lots of trust in the observation y_k. As the Kalman filter proceeds, P will have less uncertainty and therefore smaller values. Consequently, a small P will result in a K close to 0. This cancels the residual such that more trust is put into the prediction $\breve{x}$. When this happens the filter is said to be "converging".

Formulating, the Kalman Filter is a matter of choosing the appropriate system model. Thus, A,B, and u_k will depend upon the model the user chooses. Furthermore, depending on the sensors used for the correction step y_k, the observation mapping H should also change. The measurement covariance matrices Q and R will contain the noise or uncertainty in u and y.

2.1.2 Nonlinear Filtering Techniques

As seen in subsection 2.1.1, the linear Kalman filter relies on linear system models for the state prediction and observation mapping. Needless to say, there is no promise that the states

to be predicted behave linearly for real-world situations. For instance, rotations are notorious nonlinear states. Nonlinear system models and observation mappings can be denoted as follows:

$$\breve{x}_k = f(x_{k-1}, u_k + \eta_k) \tag{2.6}$$

$$y_k = h(\breve{x}_k) + n_k \tag{2.7}$$

Note that the noise in the system model comes from u_k. Since $f(x_k, u_k + \eta_k)$ is nonlinear the noise will also be non-additive. For the observation mapping, the noise n_k can be captured using an additive model.

The nonlinear functions pose a problem for propagating the process noise. The linear Kalman filter assumes that the noise is Gaussian distributed. This is true because the filter handles the estimation (or prediction) as mean and the involved noise as the variance. However, the assumption does not hold for non-linear system models, since the noise η_k is mapped through $f(x_k, u_k + \eta_k)$ resulting in colored noise. This means that the predicted process noise can not be calculated with equation A.13

Two common solutions to the non-Gaussian distributions are linearization of the states, as done by the extended Kalman Filter, or approximation of the distribution which is done for the Unscented Kalman Filter.

Extended Kalman Filter (EKF)

The nonlinear functions (2.1) and (2.2) can be linearized by a first order taylor expansion with respect to the states [Stachniss, 2013a]:

$$f(x_{k-1}, u_k) \approx f(\hat{x}_{k-1}, u_k) + \underbrace{\frac{\partial f(\hat{x}_{k-1}, u_k)}{\partial x_{k-1}}}_{:=F_k}(x_{k-1} - \hat{x}_{k-1}) \tag{2.8}$$

$$h(x_k) \approx h(\breve{x}_k) + \underbrace{\frac{\partial h(\breve{x}_k)}{\partial x_k}}_{:=H_k}(x_k - \breve{x}_k) \tag{2.9}$$

Where F_k and H_k are the Jacobian matrices of the functions. The previous expansion disregards the effects the non-additive noise. In most cases the contribution due to noise can be separated: $\breve{x}_k = f(x_{k-1}, u_k) + g(x_{k-1}, \eta_k)$. This allows to introduce a Jacobian for the noise:

$$G_k := \frac{\partial g(\hat{x}_{k-1}, \eta_k)}{\partial \eta_k} \tag{2.10}$$

The separation might not be possible for every function. In such a case the partial derivatives can also be taken from $f(\hat{x}_{k-1}, u_k + \eta_k)$, since the purpose of the separation is mostly to simplify the derivation of the derivatives.

Using the Jacobian matrices the algorithm for the extended kalman filter is as follows [Chadaporn Keatmanee, 2015]:
An error state representation of the EKF also exists [Gui et al., 2015]. The error state representation is similar to the standard EKF, but the difference is that it aims to estimate the error of state instead of the full state $\tilde{x} = x - \hat{x}$. In general, this method is a bit more precise since the magnitude of the estimated quantity is decreased consequently decreasing the linearization error is well.

Algorithm 2 Extended Kalman Filter $(x_{k-1}, P_{k-1}, u_k, y_k)$

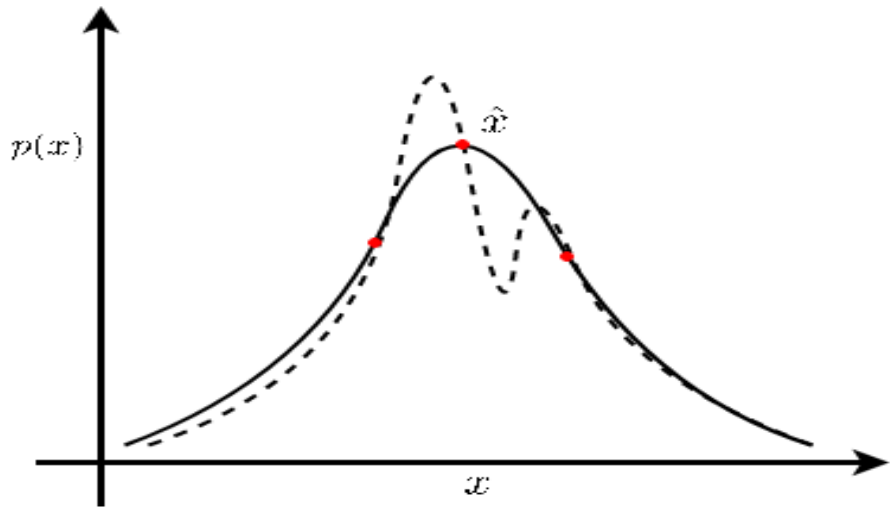
1: $\breve{x}_k = f(\hat{x}_{k-1}, u_k + \eta_k)$
2: $\breve{P}_k = F_k P_{k-1} F_k^T + G_k Q G_k^T$
3: $K_k = \breve{P}_k H_k^T (H_k \breve{P}_k H_k^T + R)^{-1}$
4: $\hat{x}_k = \breve{x}_k + K_k(y_k - H_k \breve{x}_k)$
5: $\hat{P}_k = (I - K_k H_k)\breve{P}_k$
6: return $\hat{x}_k$,$\hat{P}_k$

Since the error state Extended Kalman Filter doesn't come at the cost of complexity or computational power, it is advised to always use this formulation when considering an EKF.

Unscented Kalman Filter (UKF)

The UKF aims to approximate a non-Gaussian distribution by means of the Unscented transform. The unscented transform uses a number i of sigma points $\mathcal{S}^{[i]}$ and a weighting $w^{[}i]$ for the approximation. A sketch is shown in figure 2.1.

Figure 2.1 The Non Gaussian Distribution (Dashed line) is approximated to a Gaussian distribution (solid line) by using sigma points (red dots). X-axis denotes the possible value of state x, y-axis denotes the probability of the value $p(x)$



The sigma points can be collected as follows [Stachniss, 2013b]:

$$\begin{aligned} \mathcal{S}^{[0]} &= x \\ \mathcal{S}^{[i]} &= x + (\sqrt{(n+\lambda)P})_i \quad \text{for i = 1,..,n} \\ \mathcal{S}^{[i]} &= x - (\sqrt{(n+\lambda)P})_{i-1} \quad \text{for i = 1+1,..,2n} \end{aligned} \tag{2.11}$$

Where n denotes the dimensionality of the state $x \in \mathcal{R}^n$, λ can be used as scaling parameter, i denotes the column in the vector $\mathcal{S}^{[i]}$ and $\sqrt{(.)}$ is the matrix square root. A common definition of the matrix square root is the Cholesky Matrix square root.
The first sigma point is the mean of the distribution. The mean is always chosen as the most-likely state. For every dimension in the state vector, two additional sigma points are needed: a sigma point left of the mean and a sigma point to the right of the mean. The three total sigma points per dimension are enough to fit the Gaussian distribution.

Not every sigma point should be emphasized equally in the approximation. For instance, the first sigma points, denoting the estimated mean, is most important for the fit. Therefore, a weighting $w^{[i]}$ is introduced for each sigma points. In principle, the weighting could be chosen arbitrarily as long as it doesn't amplify the approximation. This can be accomplished if the sum equals one:

$$\sum_{i}^{2n} w^{[i]} = 1 \tag{2.12}$$

Using the weights and the collected sigma points an approximated mean $\hat{x}$ and covariance matrix $\hat{P}$ can be obtained as follows:

$$\hat{x} \approx \sum_{i=1}^{2n} w^{[i]} \mathcal{S}^{[i]} \tag{2.13}$$

$$\hat{P} \approx \sum_{i=1}^{2n} w^{[i]} (\mathcal{S}^{[i]} - \hat{x})(\mathcal{S}^{[i]} - \hat{x})^T \tag{2.14}$$

A common choice for the weighting is the following [Stachniss, 2013b]:

$$\begin{aligned} w_m^{[0]} &= \hat{x} \\ w_c^{[i]} &= w_m^{[0]} + (1 - \alpha^2 + \beta) \\ w_m^{[i]} &= w_c^{[i]} = w_m^{[0]} + (1 - \alpha^2 + \beta) \quad \text{for i = 1,..,2n} \end{aligned} \tag{2.15}$$

Where $w_m^{[i]}$ denotes a weighting used for estimating the mean or state and $w_c^{[i]}$ is used for estimation the covariance matrix. The tuning parameter α and β can be chosen as follows:

$$\begin{aligned} &\kappa \geq 0 \quad \text{Influences how far the sigma points are away from the mean} \\ &\alpha \in (0, 1] \\ &\lambda = \alpha^2(n + \kappa) - n \\ &\beta = 2 \quad \text{Optimal choice for Gaussian} \end{aligned} \tag{2.16}$$

Now the algorithm for the UKF is as follows:

Algorithm 3 Unscented Kalman Filter $(x_{k-1}, P_{k-1}, u_k, y_k)$

$\mathcal{S}_{k-1}^{[i]} = (x_{k-1} \quad x_{k-1} + (\sqrt{(n+\lambda)\Sigma})_i \quad x_{k-1} + (\sqrt{(n-\lambda)\Sigma})_i$

$\breve{\mathcal{S}}_k^{[i]} = f(\mathcal{S}_{k-1}^{[i]}, u_k)$
$\breve{x}_k = \sum_{i=1}^{2n} w_m^{[i]} \breve{\mathcal{S}}_k^{[i]}$
$\breve{P}_k = \sum_{i=1}^{2n} w_m^{[i]} (\breve{\mathcal{S}}_k^{[i]} - \breve{x}_k)(\breve{\mathcal{S}}_k^{[i]} - \breve{x}_k)^T + Q$

$\breve{\mathcal{Y}}_k^{[i]} = h(\breve{\mathcal{S}}_k^{[i]})$
$\breve{y}_k = \sum_{i=1}^{2n} w_m^{[i]} \breve{\mathcal{Y}}_k^{[i]}$
$S_k = \sum_{i=1}^{2n} w_m^{[i]} (\breve{\mathcal{Y}}_k^{[i]} - \breve{y}_k)(\breve{\mathcal{Y}}_k^{[i]} - \breve{y}_k)^T + R$
$\breve{P}_k^{x,y} = \sum_{i=1}^{2n} w_m^{[i]} (\breve{\mathcal{S}}_k^{[i]} - \breve{x}_k)(\breve{\mathcal{Y}}_k^{[i]} - \breve{y}_k)^T$
$K_k = \breve{P}_k^{x,y} S_k^{-1}$

$\hat{x}_k = \breve{x}_k + K_k(y_k - \breve{y}_k)$
$\hat{P}_k = \breve{P} - K_k S_k K_k^T$
return $\hat{x}_{k+1}$,$\hat{P}_{k+1}$

In the first line, the sigma points are collected using the previous states. The prediction (lines 2 to 4) is based on the traditional prediction but it is implemented for Sigma point. Note that the correction (lines 5 to 9) is more extensive.
Line 5 denotes the states mapped to the observation space. This is done in a separate line conversely to the previous algorithms. In line 6 the predicted observations are estimated by means of the unscented transform. Line 7 introduces S_k as the residual covariance and line 8 introduces $\breve{P}_k^{x,y}$ as the predicted cross-covariance between observation space and the states. Line 9 shows that these two are introduced to obtain the Kalman Gain. Using the Kalman Gain equation (A.9), one can see why line 9 holds:

$$K_k = \underbrace{\breve{P}_{k+1} C^T}_{P_k^{x,y}} \underbrace{(C \breve{P}_{k+1} C^T + R)^{-1}}_{S_k^{-1}} \tag{2.17}$$

The estimation is made straightforwardly in line 10. The process covariance matrix is updated using an alternative equation. This equation can be proven as follows:

$$\begin{aligned} P_k &= (I - K_k C_k)\, \breve{P}_k \\ &= \breve{P}_k - K_k C_k \breve{P}_k \\ &= \breve{P}_k - K_k (\breve{P}_k^{x,y})^T \\ &= \breve{P}_k - K_k (\breve{P}_k^{x,z} S_k^{-1} S_k)^T \\ &= \breve{P}_k - K_k (K_k S_k)^T \\ &= \breve{P}_k - K_k S_k^T K_k^T \\ &= \breve{P}_k - K_k S_k K_k^T \end{aligned} \tag{2.18}$$

2.1.3 Formulating the state estimator

Using a state estimator contains a couple of choices to be made. One should choose what type of state estimator to use and which sensors to involve.

Previously one linear and two non-linear kalman filter have been shows. Using such a filter involves formulation of system model (2.1) or (2.6) and an observation mapping (2.2) or (2.9). The models shown have been formulated for discrete system, but one can also formulate for continuous systems. Naturally this would involve discrete integration.

Next to the formulation either the EKF or UKF can be chosen for the filter implementation. In general, the EKF is easier to implement (on the algorithmic side) and converges faster. The UKF is slightly more accurate and reliable [Qasem and Reindl, 2007]. Also, the UKF spares the error-prone computation of Jacobians. This also makes the UKF a more modular choice since a different sensor suite can be handled without any further calculations.

An important consideration for state estimation is the observability of the states. Observability is the possibility to estimate the state based on input/output data [Gui et al., 2015]. The observability of a linear system can be analyzed straightforwardly. However, for non-linear systems the observability should be found using observability analysis.
Well known unobservable states for VIO are the orientation around the gravity axis and the global position. The intuition behind this is that the visual odometry only provides bearing information and the IMU is a double integrator for the position. This causes error for the unobservable states to grow without bounds.
Another problem for the state linearization of the EKF is that the yaw is unobservable but

it appears to be observable for the linearized system [Li and Mourikis, 2012]. This causes the estimator to believe that is has more information than it actually does have. Therefore it will return a covariance matrix for the state that underestimates the actual uncertainty.

In conclusion, when choosing a state estimation strategy it is important to consider which states might be unobservable. If a state proves to be unobservable adding a sensor to correct for this state is beneficial to bound the otherwise unrestrained error.

2.2 Lie Theory

The use of Lie theory shows many advantages for parametrization of motion in robotics. Applying Lie theory results into systems which are geometric, meaning that they are invariant for a change of coordinates of the inertial frame.
Next to this, expressing orientation in a Lie group avoids singularities that arise when using local parametrizations such as Euler angles. These singularities appear in the form of gimbal lock.

Some Lie groups commonly encountered in robotics are the: the rotation group $SO(3) \in \mathcal{R}^{3\times3}$, the group of rigid body motion $SE(3) \in \mathcal{R}^{4\times4}$ and the quaternion group $SU(2) \in \mathcal{R}^4$.
Similar to $SO(3)$, Quaternions can be used to express rotation. However, $SO(3)$ is considered to be the true parametrization of rotations, and $SU(2)$ is the minimal singularity-free parametrization of $SO(3)$. This results into $SU(2)$ being a double covering of $SO(3)$. That means that for every single rotation there are two different Quaternions with different values.

Now that some groups used in robotics have been examined, the remainder of this section will be used to: Define Manifolds & Lie groups, define group actions, and show how to map between different spaces. Next $SO(3)$ and $SE(3)$ will be examined more closely and twist and wrenches will be examined.

2.2.1 Manifolds & Lie Groups

A smooth manifold is a topological space that locally resembles linear space [Joan Solá, 2019]. The state or variable moves on the surface of the manifold. Since the manifold is smooth there should exist a tangent space for every point on the manifold.
Next a group $\mathcal{G}$ is a set $\mathcal{G}$ and an (composition) operation $\circ$ that satisfies the following axioms[Stramigioli, 2018]:

$$Associativity \quad : \quad (\chi_1 \circ \chi_2) \circ \chi_3 = \chi_1 \circ (\chi_2 \circ \chi_3) \in \mathcal{G} \tag{2.19}$$

$$Identity \quad : \quad \chi \circ I = I \circ \chi = \chi \tag{2.20}$$

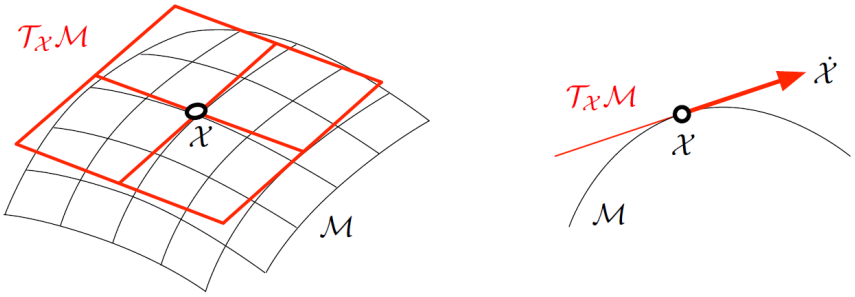
$$Inverse \quad : \quad \chi^{-1} \circ \chi = \chi \circ \chi^{-1} = I \tag{2.21}$$

For elements $\chi, \chi_1, \chi_2, \chi_3 \in \mathcal{G}$.
Now a lie group $\mathcal{M}$ can be defined as both a smooth manifold and a group.
Figure 2.2 illustrates what this would look like graphically. Do note that this is a simplification since most manifolds are not bounded by a dimension of three. The figure shows a manifold $\mathcal{M}$ where a tangent space $T_{\mathcal{X}}\mathcal{M}$ is drawn at point $\mathcal{X}$ on the surface of the manifold.

Due to the smoothness of the Manifold structure, a local linear space on the Lie group can be defined. This flat space is called the "Lie Algebra" and it is tangent to the Lie group at the identity of $\mathcal{T}_{\chi}\mathcal{M}$. The linear nature of the lie algebra allows one to do calculus while respecting

the nature of the Manifolds. This shows one important benefit of the Lie group structure. One can associate a point expressed in the linear space to the much more complex non-linear space on the manifold's surface.

Figure 2.2 Manifold $\mathcal{M}$ with tangent space $T_{\mathcal{X}}\mathcal{M}$ at $\mathcal{X}$. Sidecut visualises that velocity element $\dot{\mathcal{X}}$ belongs to tangent space $T_{\mathcal{X}}\mathcal{M}$ and not to the manifold $\mathcal{M}$ [Joan Solá, 2019].



2.2.2 Group Actions

Different Lie groups all have different actions in order to make transformations such as rotation and translation. These transformations are also known as group actions. For a Lie group $\mathcal{M}$ and a set $\mathcal{V}$, the action $\mathcal{X} \cdot v$ can be defined as the action of $\mathcal{X} \in \mathcal{M}$ on $v \in \mathcal{V}$ [Joan Solá, 2019]:

$$\cdot : \mathcal{M} \times \mathcal{V} \to \mathcal{V}; (\mathcal{X}, v) \mapsto \mathcal{X} \cdot v \tag{2.22}$$

The action $\cdot$ can be considered as a group actions if it satisfies the following axioms:

$$\begin{aligned} \textit{Identity}: &\quad I \cdot v = v \\ \textit{Compatibility}: &\quad (\mathcal{X}_1 \circ \mathcal{X}_2) \cdot v = \mathcal{X}_1 \cdot (\mathcal{X}_2 \cdot v) \end{aligned} \tag{2.23}$$

For the commonly encountered Lie group in robotics, the groups actions are defined as follows:

$$\begin{aligned} &SO(n): \text{Rotation Matrix} && R \cdot x \triangleq Rx \\ &SE(n): \text{Euclidean matrix} && H \cdot x \triangleq Rx + \xi \\ &SU(2): \text{unit quaternion} && q \cdot x \triangleq qx\mathrm{q}^* \end{aligned} \tag{2.24}$$

Where: $R \in SO(n)$ denotes a rotation matrix, $x \in \mathcal{R}^n$ represents a state or variable, H is a homogeneous matrix (Further explained in subsection 2.2.5), ξ denotes a position, q a quaternion and q^* is the quaternion inverse.

2.2.3 Mapping to spaces

As previously described, the manifold structure gives the ability to map from and to different spaces. A variable or state living in real space $x \in \mathcal{R}$ can be associated to the Tangent Space (Lie algebra) using the tilde map $\tilde{(.)}$. In order to map from a Lie algebra to real space, the vee map $(.)^V$ will be used. Both maps are defined as follows:

$$\begin{aligned} \text{Tilde} \ &: \mathcal{R}^n \to T_{\mathcal{X}}\mathcal{M}; \quad x \mapsto \tilde{x} = \sum_{i=1}^{n} x_i E_i \\ \text{Vee} \ &: \quad T_{\mathcal{X}}\mathcal{M} \to \mathcal{R}^n; \quad \tilde{x} \mapsto (\tilde{x})^{\vee} = x = \sum_{i=1}^{n} \tilde{x}_i e_i \end{aligned} \tag{2.25}$$

Where E_i and e_i are the vector bases which map to the corresponding vector space.
In order to map from the tangent space (Lie algebra) to the Lie group and back, the exponential map *exp* and logarithmic map *log* can be used. These operations provide mapping as follows:

$$\exp: \quad \mathcal{T}_{\chi}\mathcal{M} \to \mathcal{M} \quad ; \quad \tilde{x} \mapsto \chi = \exp(\tilde{x}) \tag{2.26}$$

$$\log: \quad \mathcal{M} \to \quad \mathcal{T}_{\chi}\mathcal{M}; \quad \chi \mapsto \tilde{x} = \log(\chi) \tag{2.27}$$

By using the Taylor series, closed forms of the exponential map can be obtained [Joan Solá, 2019]:

$$exp(\tilde{x}) = I + \tilde{x} + \frac{1}{2}\tilde{x}^2 + \frac{1}{3!}\tilde{x}^3 + ... \tag{2.28}$$

In appendix B it is shown that the Taylor expansion of the expontial map can be used to derive the following identities:

$$\begin{aligned} &1. \ \lim_{\varepsilon \to 0} exp(\tilde{\varepsilon}) \approx I + \tilde{\varepsilon} \\ &2. \ \lim_{\varepsilon \to 0} exp(\tilde{-\varepsilon}) \approx I - \tilde{\varepsilon} \\ &3. \ exp(\mathcal{X}\tilde{x}\mathcal{X}^{-1}) = \mathcal{X} exp(\tilde{x}) \mathcal{X}^{-1} \end{aligned} \tag{2.29}$$

2.2.4 Rotation group SO(3)

The special orthogonal group $SO(n)$ is a group whose elements are rotation matrices $R \in SO(n)$. A rotation matrix R_2^1 changes the orientation of a coordinate system ψ^1 towards the orientation of ψ^2. This group is also known as the rotation group. The number n denotes the dimension of the rotation. This thesis will be concerned with three-dimensional motion. therefore rotation matrices from $SO(3)$ will be used.

The elements of a rotation matrix R_2^1 are the inner product $\langle , \rangle$ between coordinate system ψ^1 and ψ^2 for the unit vectors $\hat{x}, \hat{y}, \hat{z}$ [S. Stramigioli, 2001]:

$$R_2^1 = \begin{array}{c|ccc} & \hat{x}_2 & \hat{y}_2 & \hat{z}_2 \\ \hline \hat{x}_1 & \langle \hat{x}_1, \hat{x}_2 \rangle & \langle \hat{x}_1, \hat{y}_2 \rangle & \langle \hat{x}_1, \hat{z}_2 \rangle \\ \hat{y}_1 & \langle \hat{y}_1, \hat{x}_2 \rangle & \langle \hat{y}_1, \hat{y}_2 \rangle & \langle \hat{y}_1, \hat{z}_2 \rangle \\ \hat{z}_1 & \langle \hat{z}_1, \hat{x}_2 \rangle & \langle \hat{z}_1, \hat{y}_2 \rangle & \langle \hat{z}_1, \hat{z}_2 \rangle \end{array} \tag{2.30}$$

Element of the special orthogonal group have some important properties. These properties are as follows [S. Stramigioli, 2001]:

$$\begin{aligned} &1. \ det(R_1^2) = 1 \\ &2. \ R_1^2 = (R_2^1)^{-1} = (R_1^2)^T \\ &3. \ \text{The columns and rows vector of } R_1^2 \text{ have length 1 and are orthogonal} \end{aligned} \tag{2.31}$$

The first and third properties are important in order to use an element of $SO(3)$ as rotation. The first property ensures that the distance in rotated vectors is preserved. The third element preserves the inner configuration between the axes of the rotated vectors (the angles between the axes will remain). The second property states that the transpose can be used as inverse. This is a beneficial property since transposing is computationally more efficient than inversion.

As seen in the previous sections, Lie groups have tangent spaces and operations in order to map towards or from the tangent spaces. In the case of $SO(3)$ the Lie algebra is a space of skew symmetric matrices $\tilde{\omega} \in so(3)$. The tilde map $\tilde{(.)}$ maps an angular velocity vector $\omega \in \mathcal{R}^3$ to a skew-symmetric matrix and the Vee map maps back to $\mathcal{R}^3$:

$$\begin{aligned} \text{Tilde} \ : \quad & \mathcal{R}^3 \to so(3); \quad \omega \mapsto \tilde{\omega} \\ \text{Vee} \ : \quad & so(3) \to \mathcal{R}^3; \quad \tilde{\omega} \mapsto (\tilde{\omega})^{\vee} = \omega \end{aligned} \tag{2.32}$$

where the skew-symmetric form $\tilde{(.)}$ of an arbitrary vector in real space $x \in \mathcal{R}^3$ is defined as follows:

$$\tilde{x} = x^{\sim} := \begin{bmatrix} 0 & -x_z & x_y \\ x_z & 0 & -x_x \\ -x_y & x_x & 0 \end{bmatrix} \tag{2.33}$$

Throughout this thesis either $\tilde{(.)}$ or $(.)^{\sim}$ will be used. The second notation is more convenient terms containing more than one variable.
A help-full property of the skew-symmetric matrix is that is equals the outer product or cross product $\wedge$. For two arbitrary vector $x_1 x_2 \in \mathcal{R}^3$:

$$x_1 \wedge x_2 = \tilde{x_1} x_2 = -x_1 \tilde{x_2} = -x_2 \wedge x_1 \tag{2.34}$$

The exponential map of $SO(3)$ maps a skew-symmetric matrix of unit length $\tilde{\hat{\omega}} \in so(3)$ by an angle $\theta \in \mathcal{R}^3$ towards the corresponding rotation matrix $R = exp(\theta \tilde{\hat{\omega}}) \in SO(3)$. This map is well known as the Rodriquez formula [S. Stramigioli, 2001]:

$$exp(\theta \tilde{\hat{\omega}}) = I + \tilde{\hat{\omega}} sin(\theta) + \tilde{\hat{\omega}}^2 (1 - cos(\theta)) \tag{2.35}$$

2.2.5 Homogeneous Matrices

General motion does not exclusively contain rotation. In order to consider the full motion, translation will also be considered. The Special Euclidean group $SE(n)$ is a group of Homogeneous matrices or H-Matrices in short. This group can be used as general transformation for rigid body motion.
Element of $SE(3)$ can transform vectors expressed in homogeneous coordinates towards different coordinate frames. For a position vector p^1 expressed in ψ^1, the homogeneous coordinates can be obtained by introducing the number 1 in the fourth index [S. Stramigioli, 2001]:

$$p^1 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \end{bmatrix} \Leftrightarrow P^1 = \begin{bmatrix} x_1 \\ y_1 \\ z_1 \\ 1 \end{bmatrix} \tag{2.36}$$

Where the capital letter of the vector denotes the homogeneous coordinates. Now the general transformation $H_1^2 \in SE(3)$ can be used to map towards ψ^2:

$$P^2 = H_1^2 P^1 = \begin{bmatrix} R_1^2 & \xi_1^2 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} p^1 \\ 1 \end{bmatrix} = \begin{bmatrix} R_1^2 p^1 + \xi_1^2 \\ 1 \end{bmatrix} \tag{2.37}$$

Where $R_1^2 \in SO(3)$ denotes the rotation and $\xi_1^2 \in \mathcal{R}^3$ the distance from ψ^1 to ψ^2.

consequently due to the fact that H-matrices can be used to transform between coordinate frames, the transformations can be stacked using the chain rule:

$$H_n^0 = H_1^0 H_2^1 ... H_n^{n-1} \tag{2.38}$$

2.2.6 Twists and Wrenches

Velocity Vectors can also be considered geometrically. The geometrical vector for (angular) velocity is called "twist" and it contains the following components:

$$T = \begin{bmatrix} \omega \\ v \end{bmatrix} \tag{2.39}$$

Where $\omega, v \in \mathcal{R}^3$. Using the formerly defined H-matrices the twist can be defined. The twists can be considered in the Lie Algebra of $SE(3)$. Depending on the transformation either a left twist or a right twist can be defined:

$$\begin{aligned} \text{Left twist: } \tilde{T}_1^{1,2} &:= H_2^1 \dot{H}_1^2 \\ \text{Right twist: } \tilde{T}_1^{2,2} &:= \dot{H}_1^2 H_2^1 \end{aligned} \tag{2.40}$$

Where: $\tilde{T} = \begin{bmatrix} \tilde{\omega} & v \\ 0 & 1 \end{bmatrix}$. Similar to the transformations seen in the subsection 2.2.5 H-matrices can be used transform twists $\tilde{T}_1^{2,2} = H_1^2 \tilde{T}_2^{1,2}$.
In some cases it is simpler to transform the twist in vector form. Hence, the Adjoint matrix Adj_H of a H-matrix can used used to execute these transformations [S. Stramigioli, 2001]. The Adjoint matrix is square and contain six rows and columns $Adj_H \in \mathcal{R}^{6\times6}$. The transformation of ψ^1 to ψ^2 using adjoint matrices can be achieved as follows:

$$Adj_{H_1^2} = \begin{bmatrix} R_1^2 & 0 \\ \tilde{\xi}_1^2 R_1^2 & R_1^2 \end{bmatrix} \tag{2.41}$$

The transformation between twists T^2 and T^1 can now be considered using the Adjoint:

$$T^2 = Adj_{H_1^2} T^1 \tag{2.42}$$

Similar to the twist, a special vector can also be defined for forces f and torques τ. A wrench W is a sixdimensional co-vector with the following elements:

$$W = \begin{bmatrix} \tau \\ f \end{bmatrix} \tag{2.43}$$

A wrench is called a co-vector because it transforms differently. For instance, a change of coordinates for wrenches using the adjoint can only be obtained as follows:

$$W^2 = Adj^T_{(H_1^2)^T} W^1 \tag{2.44}$$

Chapter 3

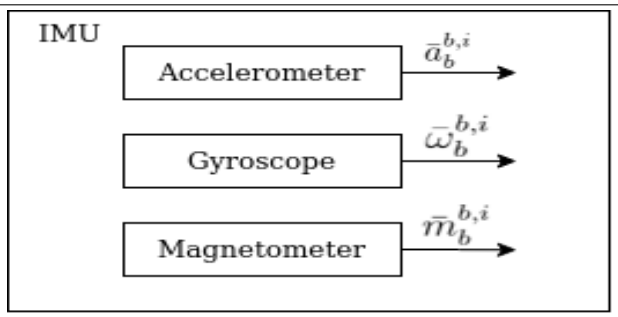
Sensors & Navigations Systems

The most commonly encountered sensors in state estimation for UAVs is the inertial measurement unit (IMU). IMUs are cheap sensors capable of making accurate measurements. However, IMUs generally suffer from an inevitable bias or drift. In order to compensate for this bias, the IMU measurements are usually considered together with a sensor measuring positional information of the system. Examples of such systems are: global navigation satellite systems (GNSS) and visual odometry (VO).

3.1 Inertial Measurement Unit (IMU)

Typical IMUs contain the following sensors: Accelerometers, gyroscopes and magnetometers. These sensor correspondingly measure the: acceleration ($\bar{a}$), angular velocity ($\bar{\omega}$) and magnetic field ($\bar{m}$). It will be assumed that the IMU is perfectly fixed to the body frame ψ^b. By means of this assumption, all measurements captured by the IMU are measured in the ψ^b with respect to the inertial frame ψ^i expressed in ψ^b. Furthermore, the bar notation $\bar{(.)}$ is used to denotes a raw sensor measurement. Figure 3.1 shows a block diagram representation of an IMU and its measurements.

Figure 3.1 Block Diagram showing IMU assembly and its measurements [Nøkland, 2011]



3.1.1 Accelerometer Measurement Model

Using an accelerometer one can measure the linear acceleration $a_b^{b,i} \in \mathcal{R}^3$ of the body. However, this measurement is affected by gravity in the vertical axis. By incorporating this constant deviation, the accelerometer measurements can be modelled as follows:

$$\bar{a}_b^{b,i} = R_i^b(\ddot{\xi}_b^i - g_b^i e_3) + b_a^b + \eta_a^b \tag{3.1}$$

The measured acceleration $\bar{a}_b^i$ is modelled as the linear acceleration vector $\ddot{\xi}_b^i$ corrected by the gravity g_b^i acting on ψ^b expressed in ψ^i and rotated by $R_i^b \in SO(3)$ from ψ^i to ψ^b. Furthermore

the measured acceleration is corrupted by an additive bias $b_a^b \in \mathcal{R}^3$ expressed in ψ^b and an additive white noise $\eta_a \sim \mathcal{N}(0, \sigma_a^2) \in \mathcal{R}^3$.

The bias can be assumed to be slowly time-varying [Kok et al., 2017]. In that case the bias can be modelled as a random walk:

$$\dot{b}_a^b = \eta_{b_a} \tag{3.2}$$

Where $\eta_{b_a} \sim \mathcal{N}(0, \sigma_{b_a}^2)$. By manipulating the variance $\sigma_{b_a}^2$ one can manipulate how quickly the bias is deviating. By using larger values for the variance, more quickly changing random walks are expected. Vice versa, by using smaller variances more slowly changing random walks are expected.

3.1.2 Gyroscope Measurement Model

Gyroscopes measure the angular velocity of the rigid body it is attached to. Since these measurements do not feel the pul by gravity, the measurements can be modelled straightforwardly:

$$\bar{\omega}_b^{b,i} = \omega_b^{b,i} + b_\omega^b + \eta_\omega^b \tag{3.3}$$

The measured angular velocity $\bar{\omega}_b^{b,i}$ is modelled as the actual angular velocity $\omega_b^{b,i}$ with additive bias b_ω^b expressed in ψ^b and additive white noise $\eta_\omega \sim \mathcal{N}(0, \sigma_\omega^2)$. Similar to the accelerometer the additive bias of the gyroscope can be modelled as a random walk:

$$\dot{b}_\omega^b = \eta_{b_\omega} \tag{3.4}$$

Where the bias noise η_{b_ω} is assumed to be Gaussian $\eta_{b_\omega} \sim \mathcal{N}(0, \sigma_{b_\omega}^2)$.

3.1.3 Magnetometer

According to [Lim et al., 2012] a magnetometer is a sensor commonly encountered in UAV flight avionics. Most UAVs posses magnetometers to measure the three directional magnetic field intensity in a body frame m^b. This measurement can be used to obtain the orientation, in yaw direction, of a rigid body. If one knows the magnetic field intensity in an inertial frame m^i, the difference between the m^i and m^b is caused by a rotation of the body. This can be modelled as follows: [Wang et al., 2018]:

$$\bar{m}_b^{b,i} = R_i^b m^i \tag{3.5}$$

The earth's magnetic field intensity differs per region. According to [Wang et al., 2018] the magnetic field intensity m^i in most regions of the earth can be calculated according to World Magnetic Model (WMM). The Nation Geophysical Data Center (NGDC) has a *online calculator* which can be used to calculate the magnetic field intensity for different regions. For the University of Twente "Drienerlolaan 5" in Enschede the magnetic field is calculated to be: $m^i = \begin{bmatrix} 18952.2 & 775.2 & 45580.2 \end{bmatrix} nT$.

It will be assumed that the magnetometer can be modelled with an additive measurement model. By means of this assumption equation (3.5) can be rewritten:

$$\bar{m}^b = R_i^b m^i + \eta_{mag} \tag{3.6}$$

Where an additive white noise $\eta_{mag} \sim \mathcal{N}(0, \sigma_{mag}^2)$ is considered.

3.2 Global Positioning System (GPS)

GPS has the advantage that it can be implemented in a simple manner. Though the measurements are not as accurate as IMU measurements, GPS does show to have a limited drift. This characteristic makes it a good companion for the IMU.
A disadvantage of GPS is that is has a limited reliability [Duflos et al., 2006]. It is possible that GPS loses its signal for a certain amount of time. The quality for GPS data suffers most indoors or in dense urban environments. Due to this potential loss of information, it is still advantageous to use an additional sensor system capable of correcting IMU biases next to the GPS for certain applications.

The rest of this section will be concerned with: the coordinate systems used in GPS navigation and the measurement model.

3.2.1 Reference Frames

In Satellite navigation systems different coordinate frames can be encountered. Some of these frames are fixed to references in space while others are fixed to the earth. The following are examples of frames which be used to express coordinates received by satellites [Nøkland, 2011]:

- Earth-Centered Inertial (ECI) frame: is an inertial frame where the origin is fixed to the center of the earth. The x-axis is pointed towards the vernal equinox (vector denoting the earth's position relative to the sun when Spring arrives for the northern hemisphere). The z-axis points along the Earth's rotation axis.
- Earth-Centered Earth Fixed (ECEF) frame: is also an inertial frame where the origin is fixed to the Earth. However, conversely to the ECI frame, the ECEF frame's orientation is also fixed to the earth. The x-axis is fixed to 0°longitude (Greenwich meridian) and the y-axis is fixed to 0 °latitude (equator). This makes that transforming from the ECI frame to the ECEF frame involves rotation by the angular velocity of the earth.
- North East Down (NED) frame: denotes a frame fixed to the surface of the earth. The x-axis points to the true North of the earth, the y-axis points eastwards and the z-axis points down to the center of the earth.
  One can also use the East North Up frame, which is the same as the NED frame but the z-axis points up.

The World geodetic system 1984 (WGS84) is an ECEF frame which is considered as a standard for working with GPS messages. The coordinate system uses modeled and defined parameters of the earth, such as its angular velocity and radius. The defined parameters are used to make the coordinate changes for different frames. For example, the earth's angular velocity is used to transform from the ECI to the ECEF frame.

For local navigational applications, it is most convenient to work with the NED frame. This is true since this frame can be fixed to the initial position of a system. The following rotation can be applied to transform to the NED frame from an ECEF frame: [Nøkland, 2011]:

$$R(\theta_{ned}^{ecef}) = \begin{bmatrix} -sin(\mu)cos(l) & -sin(l) & -cos(\mu)cos(l) \\ -sin(\mu)sin(l) & cos(l) & -cos(\mu)sin(l) \\ cos(\mu) & 0 & -sin(\mu) \end{bmatrix} \tag{3.7}$$

Where $\theta_{ned}^{ecef} = \begin{bmatrix} l & \mu \end{bmatrix}^T$ denotes the longitude l and latitude μ.

3.2.2 Measurement Model

GPS is capable of tracking the position ξ_b^i and the linear velocity $v_b^{b,i}$ of a rigid body. The GPS signal is measured from an antenna placed on the system to track. However, it is no promise that the antenna is fixed to ψ^b. For that reason the antenna will be considered in the measurement model:

$$\bar{\xi}_b^i = \xi_b^{,i} + R_b^i r_{ant}^b + \eta_{gps,p} \tag{3.8}$$

$$\bar{v}_b^{i,i} = v_b^{i,i} + R_b^i \tilde{\omega}_b^{b,i} r_{ant}^b + \eta_{gps,v} \tag{3.9}$$

Where r_{ant}^b is the distance from the GPS antenna to the body frame, R_b^i denotes the rotation from ψ^b to ψ^i, $\tilde{\omega}_b^{b,i}$ denotes the tilde mapped angular velocity from ψ^b to ψ^i expressed in ψ^b. An additive white noise is considered for the positional measurement $\eta_{gps,p} \sim (0, \sigma_{gps,p}^2)$ and the velocity measurement $\eta_{gps,v} \sim (0, \sigma_{gps,v}^2)$.

Chapter 4

State Estimation on Manifold

This chapter will be concerned with estimating states belonging to a Lie group. In order to make use of state estimators for states on a manifold, traditional state estimators should be redefined. This must be done because state estimator use operators which 'break' the mathematical constraint imposed by the Lie group.

To demonstrate this, an example will be given for a estimating $R \in SO(3)$. One of the constraints for $SO(3)$ can be seen in equation 2.31: $det(R) = 1$. The Kalman filter corrects predictions using equation 2.4. In this equation the innovation $K(y_k - C\breve{x})$ will be replaced with $\delta R \in SO(3)$ for the sake of simplicity. Now if one would naively use equation 2.31 for estimation R, this would lead to the following

$$\hat{x}_k = \breve{R}_k + \delta R_k \tag{4.1}$$

Naturally $det(\breve{R}_k) = 1$ and $det(\delta R_k) = 1$ thus:

$$det(\hat{x}_k) = det(\breve{R}_k + \delta R_k) \neq 1 \tag{4.2}$$

Which shows for this approach $\hat{x}_k \notin SO(3)$.

In order to define state estimators which do respect the constraints, the plus "+" should be replaced by an operator that maps to a Lie group. Also in order to calculate residuals $K(y_k - C\breve{x})$ for states on a manifold, the "$-$" should be replaced as well.

In section 4.1 the plus- and minus operator will be introduced. These are operators composed of the mappings encountered in section 2.2.6. This allows doing calculus while respecting the structure of the Lie group one would like to use. Section 4.2 shows how these operators can be used to handle uncertainty on a manifold. Section 4.3 will redefine partial derivatives for elements of manifolds. This can be used in case one wants to implement a state estimator which linearizes the state, such as an EKF. Section 4.4 will present an UKF which uses section 4.1 and 4.2 to estimate states belonging to a Lie group.

4.1 The plus and minus operator

State estimation requires operations to map perturbations onto the space a state belongs to. This becomes more involved when considering states belong to a non-additive space such as $SO(n)$ or $SE(n)$. If one uses states belonging a manifold, there is the need to add perturbations from real space: $\mathcal{M} \circ \mathcal{R}^n \to \mathcal{M}$.

The plus operator $\oplus$ and minus operator $\ominus$ allows to introduce increments between elements of a (curved) manifold and express them in its (flat) tangent vector space. Both $\oplus$ and $\ominus$ are introduced in [Joan Solá, 2019]. The operators can be defined by using the exponential map $exp(.)$ (2.26) and logarithmic map $log(.)$ (2.27). Depending on the order of operation, either right- or left operators can be defined:

$$\text{right-}\oplus: \quad \mathcal{X}_2 = \mathcal{X}_1 \oplus \tau^{\mathcal{X}_1} \triangleq \mathcal{X}_1 \circ \exp\left(\tau^{\mathcal{X}_1}\right) \in \mathcal{M} \tag{4.3}$$

$$\text{right-}\ominus: \quad x^{\tau} = \mathcal{X}_2 \ominus \mathcal{X}_1 \triangleq \log\left((\mathcal{X}_2)^{-1} \circ \mathcal{X}_1\right) \in T_{\mathcal{X}}\mathcal{M} \tag{4.4}$$

Note that for (4.3) the exponential map $exp(\tau^{\mathcal{X}_1})$ is composed from the right and that the argument is the tangent space at $\mathcal{X}$. This should be considered as the frame in which τ is expressed. Since $\tau^{\mathcal{X}}$ is at expressed at $\mathcal{X}$, it said to be in the local frame.
Now for the left operators:

$$\text{left-}\oplus: \mathcal{X}_2 = \tau^{I} \oplus \mathcal{X}_1 \triangleq \text{Exp}\left(I^{\tau}\right) \circ \mathcal{X} \in \mathcal{M} \tag{4.5}$$

$$\text{left-}\ominus: \tau^{I} = \mathcal{X}_2 \ominus \mathcal{X}_1 \triangleq \log\left(\mathcal{X}_1 \circ \mathcal{X}_2^{-1}\right) \in T_I\mathcal{M} \tag{4.6}$$

In (4.6) there result obtained is at the tangent space at Identity. Therefore the $\ominus$ is said to be in a global frame. It is most convenient to increment locally and subtract globally, thus (4.3) and (4.6) will be used as default.

The definition of $\oplus$ and $\ominus$ holds for general Lie groups. Therefore one can use the exponential- and logarithmic functions of a specific Lie group, such as SO(n), to define $\oplus$ and $\ominus$ respecting the constraints of the corresponding group.

4.2 Probability

Sensor fusion/state estimation relies on the use of probability distributions to represent uncertainty and noisy sensor data. When expressing states in lie groups the uncertainty and noise should also respect the constraint of the groups. Therefore this section will redefine probabilistic measures using $\oplus$ and $\ominus$.

For state estimation processes one always considers a state $x \in \mathcal{R}^n$ with uncertainty η assumed as white noise $\eta \sim \mathcal{N}(0, \sigma^2)$. This uncertainty is commonly being referred to as process noise. The estimatio with its process noise can be described as follows:

$$\mathcal{N}(x, \sigma^2) = \hat{x} + \mathcal{N}(0, \sigma^2) \tag{4.7}$$

Where $\hat{x}$ denotes the (estimated) mean and x denotes the full state.

The general approach for expressing uncertainty in Lie groups is to lift the uncertainty to the Lie group. This method is advocated similarly in [Hertzberg et al., 2011] and [Joan Solá, 2019]. A similar approach is taken in [Brossard et al., 2017] but in their work, the perturbation is expressed globally (increment from the left).
In order to accomplish the uncertainty for a state in a Manifold $\mathcal{X} \in \mathcal{M}$, the uncertainty η should be expressed in the tangent space of $\mathcal{X}$:

$$\tau = \tilde{\eta} \in \mathcal{T}_{\mathcal{X}} \tag{4.8}$$

Now the plus operator can be used to add the uncertainty:

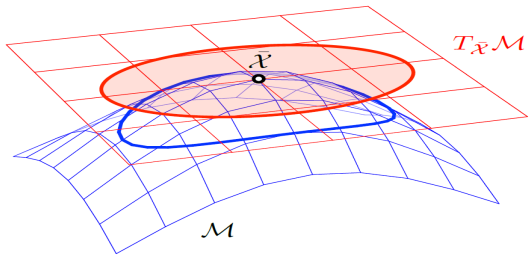
$$\mathcal{N}(\mathcal{X}, P) = \mathcal{X} \oplus \tau \in \mathcal{M} \tag{4.9}$$

The lifting approach is visualized in figure 4.1.
Similar to errors in real space, the errors in the tangent space can be found by considering the difference between the true state $\mathcal{X}$ and estimated mean $\hat{\mathcal{X}}$: $\tau = \mathcal{X} \ominus \hat{\mathcal{X}}$. This can be used to define a covariance matrix in the tangent space:

$$\mathcal{P} = E[\tau\tau^T] = E[(\mathcal{X} \ominus \hat{\mathcal{X}})(\mathcal{X} \ominus \hat{\mathcal{X}})^T] \in \mathcal{T}_{\mathcal{X}} \tag{4.10}$$

The covariance matrix definition from (4.10) can be used for both states and measurements.

Figure 4.1 Uncertainty around $\mathcal{X} \in \mathcal{M}$. The uncertainty is expressed in the tangent space $T_{\mathcal{X}}\mathcal{M}$ (red) and lifted to the group (blue) [Joan Solá, 2019]



.

4.3 Partial derivatives

As explained in subsection 2.1 the EKF solves the non-linearity by means of linearizing the state. In order to accomplish this a Jacobian matrix with respect to the states and noise is applied. In other to define the Jacobian partial dirivatives for Lie groups should be found. The well known partial derivative for real numbers is as follows:

$$\frac{\partial f(x)}{x} = \lim_{\varepsilon \to 0} \frac{f(x+\varepsilon) - f(x)}{\varepsilon} \tag{4.11}$$

where $f(x), x \in \mathcal{R}^n$. As is known by many, the idea of differentiation is to introduce a small increment in the function $f(x+\varepsilon)$ an calculate the rate of change with respect to the original function f(x).

Whenever the state is an element on a manifold the standard definition doesn't hold. One cannot add a perturbation in real space $\varepsilon \in \mathcal{R}^n$ to a state on a manifold $\mathcal{X} \in \mathcal{M}$. However, using $\oplus$ the perturbation can be lifted in order to still considering small differences on the manifold:

$$\frac{\partial f(\mathcal{X})}{\mathcal{X}} = \lim_{\varepsilon \to 0} \frac{f(\mathcal{X} \oplus \varepsilon) - f(\mathcal{X})}{\varepsilon} \tag{4.12}$$

By inspecting of the definition one can see that the idea of differentiation has not changed for elements on manifolds.A small increment is still added to the original function $f(\mathcal{X} \oplus \varepsilon)$, but now it is added in the Lie group.

If one of the states in the state estimator is a manifold element, then there probably also exists a function: $\mathcal{X} \to \mathcal{X}_2 = f(\mathcal{X})$. In such a case, the difference in the perturbed function should be

considered on the manifold. Thus the partial derivative can be altered to consider difference on manifold by using $\ominus$:

$$\frac{\partial f(\mathcal{X})}{\mathcal{X}} = \lim_{\varepsilon \to 0} \frac{f(\mathcal{X} \oplus \varepsilon) \ominus f(\mathcal{X})}{\varepsilon} \tag{4.13}$$

Lastly for state estimation one can also encounter mappings from real space to the manifold $x \to \mathcal{X} = f(x)$. In such a case the result of $f(x)$ is a manifold element but small changes are still defined relative to x. For situation the following partial derivative can be used:

$$\frac{\partial f(x)}{x} = \lim_{\varepsilon \to 0} \frac{f(x + \varepsilon) \ominus f(x)}{\varepsilon} \tag{4.14}$$

4.4 Unscented Kalman Filter on Manifold

Using the operators from section 4.1 and 4.2, the UKF can now be formulated to handle states in Lie groups $\mathcal{X} \in \mathcal{M}$. The algorithm is as follows [Hertzberg et al., 2011]:

Algorithm 4 Unscented Kalman Filter $(\mathcal{X}_{k-1}, P_{k-1}, u_k, y_k)$

1: $\mathcal{S}_{k-1}^{[i]} = (\mathcal{X}_{k-1} \quad \mathcal{X}_{k-1} \oplus (\sqrt{(n+\lambda)P_{k-1}})_i \quad \mathcal{X}_{k-1} \oplus (\sqrt{(n-\lambda)P_{k-1}})_i$

2: $\breve{\mathcal{S}}_k^{[i]} = f(\mathcal{S}_{k-1}^{[i]}, u_k)$
3: $\breve{\mathcal{X}}_k = \sum_{i=1}^{2n} w_m^{[i]} \breve{\mathcal{S}}_k^{[i]}$
4: $\breve{P}_k = \sum_{i=1}^{2n} w_m^{[i]} (\breve{\mathcal{S}}_k^{[i]} \ominus \breve{\mathcal{X}}_k)(\breve{\mathcal{S}}_k^{[i]} \ominus \breve{\mathcal{X}}_k)^T + Q$

5: $\breve{\mathcal{S}}\breve{\mathcal{Y}}_k^{[i]} = h(\breve{\mathcal{S}}_k^{[i]})$
6: $\breve{\mathcal{Y}}_k = \sum_{i=1}^{2n} w_m^{[i]} \breve{\mathcal{S}}\breve{\mathcal{Y}}_k^{[i]}$
7: $S_k = \sum_{i=1}^{2n} w_m^{[i]} (\breve{\mathcal{Y}}_k^{[i]} \ominus \breve{y}_k)(\breve{\mathcal{Y}}_k^{[i]} \ominus \breve{y}_k)^T + R$
8: $\breve{P}_k^{x,y} = \sum_{i=1}^{2n} w_m^{[i]} (\breve{\mathcal{S}}_k^{[i]} \ominus \breve{\mathcal{X}}_k)(\breve{\mathcal{Y}}_k^{[i]} \ominus \breve{y}_k)^T$
9: $K_k = \breve{P}_k^{x,y} S_k^{-1}$

10: $\delta = K_k(y_k \ominus \breve{y}_k)$
11: $P'_k = \breve{P} - K_k S_k K_k^T$
12: $\mathcal{X}'_t = \Big(\mathcal{X}_k \oplus \delta \quad \mathcal{X}_k \oplus (\delta + (\sqrt{(n+\lambda)P'_k}) \quad \mathcal{X}_k \oplus (\delta - (\sqrt{(n+\lambda)P'_k})\Big)$
13: $\hat{\mathcal{X}}_k = \sum_{i=1}^{2n} w_m^{[i]} (\mathcal{X}'_t)$
14: $\hat{P}_k = \sum_{i=0}^{2n} w_m^{[i]} (\mathcal{X}'_t \ominus \hat{\mathcal{X}})(\mathcal{X}'_t \ominus \hat{\mathcal{X}})^T$
15: return $\hat{\mathcal{X}}_k$,$\hat{P}_k$

Some differences can be observed relative to the UKF from 2.1. First of all, the sigma points gathered in line 1 are now obtained using $\oplus$. Line 4 shows that the process covariance matrix is now obtained using equation 4.10. This equation is also used in line 7 and line 8. Line 10 shows that the innovation is now calculated using $\ominus$. This allows the observation to be real $y_k \in \mathcal{R}^n$ and obtain a real number for the innovation. Line 12 adds the innovation from line 10 to the prediction and gather the sigma points. In line 13, these sigma point are summed with the weight to obtain the final estimation. Lastly the $\hat{P}_k$ is obtained in line 14.

Chapter 5

Filter Formulation

This chapter will formulate two estimators. As is explained in chapter 2 this is a matter of formulating two sets of functions. The first set are the state equations $f(x, u)$ that will be used to predict the next value of the states. The second set will be the observation mapping $h(\breve{x})$ to map the predicted states towards the observation space.

Before the estimators are formulated, equations of motion will be derived. Section 5.1 will derive equations of motion purely based on kinematics. Section 5.2 derives similar equations but these equations are based on dynamics instead of exclusively kinematics. Both sets of equations will be derived geometrically and in accordance with chapter 2. This allows the estimators to be free of coordinate changes and avoids singularities.
Section 5.3 will formulate a geometrical state estimator where the prediction is driven by accelerometer and gyroscope measurements. Section 5.4 will build upon this estimator but the prediction will be formulated such that it tightly-couples an estimated external force.

5.1 Rigid Body Kinematics

The motion of a UAV can be tracked by using two frames: An inertial frame ψ^i and a body frame ψ^b. The body frame considered is the frame fixed to the center of mass of the UAV. The rigid body motion can be described by considering the twist between the frames. For the left twist, this looks as follows:

$$\tilde{T}_b^{b,i} = H_i^b \dot{H}_b^i = \begin{bmatrix} R_i^b & \xi_i^b \\ 0 & 1 \end{bmatrix} \begin{bmatrix} \dot{R}_b^i & \dot{\xi}_b^i \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} R_i^b \dot{R}_b^i & R_i^b \dot{\xi}_b^i \\ 0 & 0 \end{bmatrix} = \begin{bmatrix} \tilde{\omega}_b^{b,i} & v_b^{b,i} \\ 0 & 0 \end{bmatrix} \tag{5.1}$$

This definition can be worked out component-wise:

$$v_b^{b,i} = R_i^b \dot{\xi}_{ib}^{\ i} \tag{5.2}$$

$$\tilde{\omega}_b^{b,i} = R_i^b \dot{R}_b^i \tag{5.3}$$

By straightforward algebraic manipulation, the equation can be written as follows:

$$\dot{\xi}_b^i = R_b^i v_b^{b,i} \tag{5.4}$$

$$\dot{R}_b^i = R_b^i \tilde{\omega}_b^{b,i} \tag{5.5}$$

By taking the time derivative of $v_b^{b,i}$ (5.2), an analytic expression for linear acceleration can be derived.

$$\begin{aligned}\dot{v}_b^{b,i} &= \frac{d}{dt}(R_i^b \dot{\xi}_{ib}^{\ i}) \\ &= \dot{R}_i^b \dot{\xi}_{ib}^{\ i} + R_i^b \ddot{\xi}_{ib}^{\ i} \\ &= \tilde{\omega}_i^{b,b} R_i^b \dot{\xi}_{i\xi i}^{\ i} + R_i^b \ddot{\xi}_{ib}^{\ i} \\ &= -\tilde{\omega}_b^{b,i} v_b^{b,i} + (R_b^i)^{-1} \ddot{\xi}_{ib}^{\ i}\end{aligned} \tag{5.6}$$

The equations of motion (5.4),(5.5) and (5.6) can be used to fully describe the motion of a rigid body.

5.2 Rigid Body Dynamics

To track motion of a rigid body, one can also relate dynamic effects on the rigid body motion. This thesis will consider three wrenches acting on the body:

$$W^b = W_g^b + W_{pr}^b + W_{ext}^b \tag{5.7}$$

Where w_g^b denotes a wrench caused by gravity, w_{pr}^b a wrench injected by the controller via the propellers and w_{ext}^b is an external wrench.

The gravity wrench can be modelled straightforwardly, since it is a wrench oriented in the same direction as ψ^i: $R_b^g = R_b^i$. However, the origin of this frame is fixed to ψ^b. Thus, considering gravity in the body frame is to transform the gravity contribution $-mg$ to the body frame:

$$W_g^b = Ad_{R_b^g}^T \begin{bmatrix} 0_5 \\ -mge \end{bmatrix} = \begin{bmatrix} 0_3 \\ -R_b^i(mge_3) \end{bmatrix} \tag{5.8}$$

The wrench caused by the propellers is acting in the body frame W_{pr}^b. This effect can be modelled by considering the thrust from each propeller $\lambda = \begin{bmatrix}\lambda_1 & ... & \lambda_6\end{bmatrix}^T$ and a control allocation matrix M mapping the propeller thrust to the wrench acting on the body [Rashad et al., 2019b].

$$W_{pr}^b = M\lambda \tag{5.9}$$

External effects acting on the model will be captured within the external wrench W_{ext}^b. Since the behaviour of w_{ext}^b is generally not known beforehand, no underlying dynamics are assumed. In order to still capture the behaviour of the external wrench, it will be modelled as a random walk [McKinnon and Schoellig, 2016]:

$$\dot{W}_{ext}^b = \eta_{W_{ext}} \tag{5.10}$$

Where $\eta_{W_{ext}} \sim \mathcal{N}(0, \sigma_{W_{ext}}^2)$ is a White noise. This method leaves $\sigma_{W_ext}^2$ as a tweaking parameter. By choosing larger values the external wrench will change more quickly and smaller values result into a more slowly varying external wrench.
The wrench W_b acting on the body will be a sum of the previously described wrenches, which components look as follows:

$$\begin{aligned} f^b &= f_g^b + f_{pr}^b + f_{ext}^b \\ \tau^b &= \tau_{pr}^b + \tau_{ext}^b \end{aligned} \tag{5.11}$$

The effects of a wrench on the motion of a rigid body can be described by the Euler Equation for rigid body's [S. Stramigioli, 2001]:

$$\mathcal{I}^k \dot{T}_b^{k,i} = \begin{pmatrix} -\tilde{\omega}_k^{k,i} & -\tilde{v}_{k,i}^{k,i} \\ 0 & -\tilde{\omega}_k^{k,i} \end{pmatrix} \mathcal{I}^k T_k^{k,i} + \left(W^k\right)^T \tag{5.12}$$

Where $\mathcal{I}^k$ denotes the inertia matrix in the principal inertial frame ψ^k. Due to the fact that it is expressed in the principal inertial frame, the inertia matrix is a diagonal matrix dependent on the moment of inertia $J \in \mathcal{R}^3$ and mass m in the following manner: $\mathcal{I} = diag\begin{bmatrix} J & mI_{3x3} \end{bmatrix}$. For simplification purposes it will be assumed that the body frame aligns with the principal inertia frame $\psi^b = \psi^k$. Now the terms will be written component-wise. First the upper row will be considered:

$$\begin{aligned} J\dot{\omega}_b^{b,i} &= \tilde{\omega}_b^{b,i} J\omega_b^{b,i} + \tau^b \\ J\dot{\omega}_b^{b,i} &= J\tilde{\omega}_b^{b,i}\omega_b^{b,i} + \tau^b \\ \dot{\omega}_b^{b,i} &= \tilde{\omega}_b^{b,i}\omega_b^{b,i} + J^{-1}\tau^b \\ \dot{\omega}_b^{b,i} &= \tilde{\omega}_b^{b,i}\omega_b^{b,i} + J^{-1}(\tau_{pr}^b + \tau_{ext}^b) \end{aligned} \tag{5.13}$$

The first step is to move the inertia matrix J forward. This is allows as the J is diagonal. Lastly the torque can be expanded by the torque contributions considered.
Now for the lower row:

$$\begin{aligned} m\dot{v}_b^{b,i} &= -\tilde{\omega_b^{b,i}} m v_b^{b,i} + f^b \\ m\dot{v}_b^{b,i} &= -m\tilde{\omega_b^{b,i}} v_b^{b,i} + f^b \\ \dot{v}_b^{b,i} &= -\tilde{\omega}_b^{b,i} v_b^{b,i} + m^{-1} f^b \\ \dot{v}_b^{b,i} &= -\tilde{\omega}_b^{b,i} v_b^{b,i} + m^{-1}\left(f_g^b + f_{pr}^b + f_{ext}^b\right) \end{aligned} \tag{5.14}$$

Similar to the rotational case, first the mass m is moved, then divided and last the force contribution is expanded.
Both equations (5.13) and (5.14) result into an expression which can be used the describe the motion of a rigid body. Conversely the previous section, these equations are based on dynamics instead of exclusively kinematics.

5.3 Geometrical pose estimator

In order to track the UAV's pose, the following states will be considered:

$$x = \begin{bmatrix} \xi_b^i \\ v_b^{b,i} \\ R_b^i \\ b^b \end{bmatrix} \tag{5.15}$$

Where $\xi_b^i \in \mathcal{R}^3$ denotes the position, $v_b^{b,i} \in \mathcal{R}^3$ denotes the velocity and a rotation matrix $R_b^i \in SO(3)$ is considered. In section 3.1 it is seen that accelerometer- and gyroscope measurements are often affected by biases. Generally, state estimators can correct for these distortions if they are added to the state vector. For that reason, depending on the number of sensors s, a bias vector $b^b \in \mathcal{R}^{3s}$ will be added. It is expected that the biases are expressed in the body frame.
Furthermore, note that the rotation matrix belongs to the Lie group SO(3). therefore, the state estimator should be redefined as shown in chapter 4.

5.3.1 IMU based Prediction

A common encountered state estimator uses IMU measurements to predict the UAV's motion. This can be accomplished by using the measured acceleration $\bar{a}_b^{b,i}$ and the measured angular velocity $\bar{\omega}_b^{b,i}$ as measurement input. Equation 3.1 and 3.3 show the measurement models of the accelerometer and the gyroscope. The measurement can be incorporated into the state estimator as follows:

$$u = \begin{bmatrix} \bar{a}_b^{b,i} \\ \bar{\omega}_b^{b,i} \end{bmatrix} = \begin{bmatrix} R_i^b(\ddot{\xi}_b^i - (g_b^i e_3) + b_a^b + \eta_a^b \\ \omega_b^{b,i} + b_\omega^b + \eta_\omega^b \end{bmatrix} \tag{5.16}$$

The first three states from the state estimator can be predicted by using the kinematic relations (5.4),(5.5) and (5.6). The biases involved in this estimator are the accelerometer bias b_a^b and the gyroscope bias b_ω^b. The bias vector will be redefined to capture these effects: $b^b = \begin{bmatrix} b_a^b & b_\omega^b \end{bmatrix}^T$. The biases are predicted by means of their model (3.2) and (3.4). Incorporating all of these equations, the states can be predicted as follows:

$$\begin{bmatrix} \dot{\xi}_b^i \\ \dot{v}_b^{b,i} \\ \dot{R}_b^i \\ \dot{b}_a^b \\ \dot{b}_\omega^b \end{bmatrix} = \begin{bmatrix} R_b^i v_b^{b,i} \\ -\tilde{\omega}_b^{b,i} v_b^{b,i} + R_i^b \ddot{\xi}_b^i \\ R_b^i \tilde{\omega}_b^{b,i} \\ \eta_{ba} \\ \eta_{b\omega} \end{bmatrix} \tag{5.17}$$

In order to incorporate the measurements correctly, a distinction will be made between actual variables and their measured value. Using that distinction, variables can be restored from for instance biases. The measurement inputs u can be manipulated as follows:

$$\begin{aligned} \bar{a}_b^{b,i} &= R_i^b(\ddot{\xi}_b^i - (g_b^i e_3) + b_a^b + \eta_a^b \\ &= R_i^b \ddot{\xi}_b^i - R_i^b g_b^i e_3 + b_a^b + \eta_a^b \\ R_i^b \ddot{\xi}_b^i &= \bar{a}_b^{b,i} + R_i^b g_b^i e_3 - b_a^b - \eta_a^b \end{aligned} \tag{5.18}$$

And for the gyroscope:

$$\omega_b^{b,i} = \bar{\omega}_b^{b,i} - b_\omega^b - \eta_\omega^b \tag{5.19}$$

Now these equations can be substituted in (5.17) resulting into the state equations:

$$\begin{bmatrix} \dot{\xi}_b^i \\ \dot{v}_b^{b,i} \\ \dot{R}_b^i \\ \dot{b}_a^b \\ \dot{b}_\omega^b \end{bmatrix} \begin{bmatrix} R_b^i v_b^{b,i} \\ -(\bar{\omega}_b^{b,i} - b_\omega^b - \eta_\omega)^\sim v_b^{b,i} + \bar{a}_b^{b,i} + R_i^b g^i - b_a - \eta_a \\ R_b^i(\bar{\omega}_b^i - b_\omega^b - \eta_\omega)^\sim \\ \eta_{b_a} \\ \eta_{b_\omega} \end{bmatrix} \tag{5.20}$$

It can be seen that the state equations are non-linear and contain non-additive noise. The measurement noise involved in the state equations will be defined within the noise vector: $\eta = \begin{bmatrix} \eta_a^b & \eta_\omega^b & \eta_{b_a} & \eta_{b_\omega} \end{bmatrix} \in \mathcal{R}^{12}$. Naturally there is no way one can estimate the value of the noise. Hence a distinction will be made between a contribution due to noise and a contribution due to the state changes.

$$\underbrace{\begin{bmatrix} \dot{\xi}_b^i \\ \dot{v}_b^{b,i} \\ \dot{R}_b^i \\ \dot{b}_a^b \\ \dot{b}_\omega^b \end{bmatrix}}_{\dot{x}} = \underbrace{\begin{bmatrix} R_b^i v_b^{b,i} \\ -(\bar{\omega}_b^{b,i} - b_\omega^b)^\sim v_b^{b,i} + \bar{a}_b^{b,i} + (R_b^i)^T g^i - b_a^b \\ R_b^i(\bar{\omega}_b^i - b_\omega^b)^\sim \\ 0 \\ 0 \end{bmatrix}}_{f(x,u)} + \underbrace{\begin{bmatrix} 0 \\ \tilde{\eta_\omega} v_b^{b,i} - \eta_a \\ -R_b^i \tilde{\eta_\omega} \\ \eta_{b_a} \\ \eta_{b_\omega} \end{bmatrix}}_{g(x,\eta)} \tag{5.21}$$

The correctness of the formulation can be checked by rewriting it in terms of the states and inputs. The state vector will be rewritten to: $x = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 \end{bmatrix}^T$, the measurement input to: $u = \begin{bmatrix} u_1 & u_2 \end{bmatrix}^T$ and the noise vector to $\eta = \begin{bmatrix} \eta_1 & \eta_2 & \eta_3 & \eta_4 \end{bmatrix}$. Now the state functions can be rewritten as follows:

$$\underbrace{\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \\ \dot{x_5} \end{bmatrix}}_{\dot{x}} = \underbrace{\begin{bmatrix} x_3 x_2 \\ -(u_2 - x_5)^{\sim} x_2 + u_1 + (x_3)^T g^i - x_4 \\ x_3 (u_2 - x_5)^{\sim} \\ 0 \\ 0 \end{bmatrix}}_{f(x,u)} + \underbrace{\begin{bmatrix} 0 \\ \tilde{\eta_2} x_2 - \eta_1 \\ -x_3 \tilde{\eta_2} \\ \eta_3 \\ \eta_4 \end{bmatrix}}_{g(x,\eta)} \tag{5.22}$$

This shows that every predicted state is purely based on other states, measurement input or noise. The only parameter still written as a variable is g^i. However, this is a known constant that can be predefined. Therefore it can be concluded that the state equations are formulated correctly.

5.3.2 GPS based correction

In the case of this estimator, the accumulated errors caused by integration drift will be contained by using GPS measurements.

As seen in section 3.2 a GPS is able to provide the position ξ_b^i and the velocity $v_b^{i,i}$ of a rigid body. Involving especially the position into the correction helps to contain most of the accumulated drift since the position integrates measurement noise twice. Furthermore, the velocity and the magnetometer will be used to limit drift for the velocity and for the rotational domain. As seen in subsection 3.1.3 the magnetometer measures the magnetic field intensity in the body frame.

Using all of these measurements, the observation vector y becomes the following:

$$y = \begin{bmatrix} \bar{\xi}_b^i \\ \bar{v}_b^{i,i} \\ \bar{m}^b \end{bmatrix} \tag{5.23}$$

As seen in chapter 3 the measurement models (3.6), (3.8) and (3.9) are as follows:

$$\begin{aligned} \bar{\xi}_b^i &= \xi_b^{,i} + R_b^i r_{ant}^b + \eta_{gps,p} \\ \bar{v}_b^{i,i} &= v_b^{i,i} + R_b^i \tilde{\omega}_b^{b,i} r_{ant}^b + \eta_{gps,v} \\ \bar{m}^b &= R_i^b m^i + \eta_{mag} \end{aligned} \tag{5.24}$$

The antenna distance r_{ant}^b and the magnetic field intensity in the inertial frame m^i are constants which can be known beforehand. By considering these as parameters the measurements can be reconstructed by only using the states of the estimator. Thus, for each observations, the observation mapping can be formulated as follows:

$$\underbrace{\begin{bmatrix} \bar{\xi}_b^i \\ \bar{v}_b^{i,i} \\ \bar{m}^b \end{bmatrix}}_{y} = \underbrace{\begin{bmatrix} \xi_b^i + R_b^i r_{ant}^b \\ R_b^i (v_b^{b,i} + \tilde{\omega}_b^{b,i} r_{ant}^b) \\ (R_b^i)^T m^i \end{bmatrix}}_{h(\check{x})} + \underbrace{\begin{bmatrix} n_{gps,p} \\ n_{gps,v} \\ n_{mag} \end{bmatrix}}_{n} \tag{5.25}$$

Note that every state encountered in the observation mapping $h(\breve{x})$ are predicted states.
The first observation $\bar{\xi}_b^i$ can be reconstructed straightforwardly since all variables in the measurement model are already in the states.
Mapping of the second observations $\bar{v}_b^{i,i}$ can be accomplished by transforming to ψ^i: $v_b^{i,i} = R_b^i v_b^{b,i}$.
To simplify, the rotation matrix can be factored out: $R_b^i v_b^{b,i} + R_b^i \tilde{\omega}_b^{b,i} r_{ant}^b = R_b^i (v_b^{b,i} + \tilde{\omega}_b^{b,i} r_{ant}^b)$.
The third observation $\bar{m}^b$ already contains only a state if the rotation matrix is transposed.
Lastly, note that all noise is additive noise. Therefore no functions are needed to deal with the noise.

Similar to the prediction step in subsection 5.3.1, the correctness of the formulation will be checked by rewriting such that the mapping is only dependent on state parameters and inputs. The observations will be rewritten to $y = \begin{bmatrix} y_1 & y_2 & y_3 \end{bmatrix}$ and the observation noise $n = \begin{bmatrix} n_1 & n_2 & b_3 \end{bmatrix}$:

$$\underbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix}}_{y} = \underbrace{\begin{bmatrix} x_1 + x_3 r_{ant}^b \\ x_3(x_2 + \tilde{u_2} r_{ant}^b) \\ (x_3)^T m^i \end{bmatrix}}_{h(\breve{x})} + \underbrace{\begin{bmatrix} n_1 \\ n_2 \\ n_3 \end{bmatrix}}_{n} \tag{5.26}$$

Where r_{ant}^b and m_i can be found beforehand.

5.4 Geometrical pose wrench estimator

The second estimator to be formulated in this chapter is the state estimator tightly coupled with forces and torques. This estimator will use the same sensor suite as the geometrical pose estimator from section 5.3, but it will use the equations of motion based on dynamics derived in section 5.2. For this estimator the states are similar to the previous but extended:

$$x = \begin{bmatrix} \xi_b^i \\ v_b^{b,i} \\ R_b^i \\ b^b \\ f_{ext}^b \end{bmatrix} \tag{5.27}$$

The state vector is extended by f_{ext}^b. This enables the estimator to also estimate external influences and to involve this estimation into the pose estimation as well.

5.4.1 Dynamics based Prediction

In order to couple the dynamics it is assumed that the components of the wrench caused by propellers τ_{pr}^b and f_{pr}^b are known. This assumption is reasonable if the UAV contains an Electronic Speed Controller (ESC). Equation 5.9 shows that the wrench contribution can be found by considering the thrust per propeller and the internal mapping M. The assumption can be met by performing an identification procedure on the propellers to obtain the trust.

Next to τ_{pr}^b and f_{pr}^b the angular velocity $\omega_b^{b,i}$ will be involved as well. The measurement inputs becomes:

$$u = \begin{bmatrix} \bar{f}_{pr}^b \\ \bar{\tau}_{pr}^b \end{bmatrix} = \begin{bmatrix} f_{pr}^b + \eta_{f_{pr}}^b \\ \tau_{pr}^b + \eta_{\tau_{pr}}^b \end{bmatrix} \tag{5.28}$$

f^b_{pr} is the force component from the propeller wrench, τ^b_{pr} is the torque component. Next to this noise is added as well. The noise represents the uncertainty for the (identified) force and torque and they can be modelled as zero mean Gaussian noise: $\eta_{f_{pr}} \sim (0, \sigma^2_{f_{pr}})$ and $\eta_{\tau_{pr}} \sim (0, \sigma^2_{\tau_{pr}})$

Using the component-wise Euler equations for rigid bodies (5.13), (5.14) and the force component of the modelled external wrench (5.10) ,the state prediction becomes:

$$\begin{bmatrix} \dot{\xi}^i_b \\ \dot{v}^{b,i}_b \\ \dot{R}^i_b \\ \dot{b}_a \\ \dot{b}_\omega \\ \dot{f}^b_{ext} \end{bmatrix} = \begin{bmatrix} R^i_b v^{b,i}_b \\ -\tilde{\omega}^{b,i}_b v^{b,i}_b + m^{-1}(f^b_{pr} + f^b_g + f^b_{ext}) \\ R^i_b \tilde{\omega}^{b,i}_b \\ \eta_{ba} \\ \eta_{b\omega} \\ \eta_{f_{ext}} \end{bmatrix} \tag{5.29}$$

The state equations for the position $\dot{\xi}^i_b$ and orientation $\dot{R}^i_b$ and the biases remain unaffected relative to the geometrical pose estimator from section 5.3. The state equations for $\dot{v}^{b,i}_b$ is now set to the equation of motion (5.14).
Now the distinction between actual variables and their measurement value will be made:

$$\begin{bmatrix} \dot{\xi}^i_b \\ \dot{v}^{b,i}_b \\ \dot{R}^i_b \\ \dot{b}_a \\ \dot{b}_\omega \\ \dot{f}^b_{ext} \end{bmatrix} = \begin{bmatrix} R^i_b v^{b,i}_b \\ -(\omega^{b,i}_b - b^b_\omega - \eta_\omega)^\sim v^{b,i}_b - Rge_3 + m^{-1}(\bar{f}^b_{pr} - \eta^b_{f_{pr}} + f^b_{ext}) \\ R^i_b(\omega^{b,i}_b - b^b_\omega - \eta^b_\omega)^\sim \\ \eta_{ba} \\ \eta_{b\omega} \\ \eta_{f_{ext}} \end{bmatrix} \tag{5.30}$$

In order to derive the final state equations, the noise contribution $\eta = \begin{bmatrix} \eta^b_{f_{pr}} & \eta^b_\omega & \eta^b_{b_a} & \eta^b_{b_\omega} & \eta^b_{f_{ext}} \end{bmatrix}^T$ will be decoupled from the state contribution:

$$\underbrace{\begin{bmatrix} \dot{\xi}^i_b \\ \dot{v}^{b,i}_b \\ \dot{R}^i_b \\ \dot{b}_a \\ \dot{b}_\omega \\ \dot{f}^b_{ext} \end{bmatrix}}_{\dot{x}} = \underbrace{\begin{bmatrix} R^i_b v^{b,i}_b \\ -(\omega^{b,i}_b - b^b_\omega)^\sim v^{b,i}_b - Rge_3 + m^{-1}(\bar{f}^b_{pr} + f^b_{ext}) \\ R^i_b(\omega^{b,i}_b - b^b_\omega)^\sim \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{f(x,u)} + \underbrace{\begin{bmatrix} 0 \\ \tilde{\eta}^b_\omega - m^{-1}\eta_{f_{pr}} \\ -R^i_b \tilde{\eta}^b_\omega \\ \eta_{ba} \\ \eta_{b\omega} \\ \eta_{f_{ext}} \end{bmatrix}}_{g(\eta)} \tag{5.31}$$

As standard procedure that last step remaining is to check the correctness of the formulation. The vectors encountered will be rewritten their numbered version. The state vector becomes: $x = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 & x_5 & x_6 \end{bmatrix}^T$, the measurement input will be $u = \begin{bmatrix} u_1 & u_2 \end{bmatrix}^T$ and the noise vector is $\eta = \begin{bmatrix} \eta_1 & \eta_2 & \eta_3 & \eta_4 & \eta_5 \end{bmatrix}^T$. Now the state equations become:

$$\underbrace{\begin{bmatrix} \dot{x_1} \\ \dot{x_2} \\ \dot{x_3} \\ \dot{x_4} \\ \dot{x_5} \\ \dot{x_6} \end{bmatrix}}_{\dot{x}} = \underbrace{\begin{bmatrix} x_3 x_2 \\ -(u_2 - x_5)^\sim x_2 - x_3 g e_3 + m^{-1}(u_1 + x_8) \\ x_3 \tilde{u_2} \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{f(x,u)} + \underbrace{\begin{bmatrix} 0 \\ \tilde{\eta}_2 - m^{-1}\eta_1 \\ -x_3 \tilde{\eta}_2 \\ \eta_3 \\ \eta_4 \\ \eta_5 \end{bmatrix}}_{g(\eta)} \tag{5.32}$$

Expect the mass m, which can be known beforehand, equations can be formulated using states, noise or measurement inputs. Therefore it will be concluded that the formulation is correct.

5.4.2 Dynamics based Correction

Now the accelerometer can be used to correct the external force. By involving the linear acceleration the observation vector y becomes:

$$y = \begin{bmatrix} \bar{\xi}_b^i \\ \bar{v}_b^{i,i} \\ \bar{m}^b \\ \bar{a}_b^{b.i} \end{bmatrix} \tag{5.33}$$

The GPS measurements $\bar{\xi}_b^i$, $\bar{v}_b^{i,i}$ and magnetometer measurements $\bar{m}^b$ can be involved equally to the correction in section 5.3. In order to map towards the acceleration, Newton's second law will be used:

$$\begin{aligned} a_b^{b,i} &= m^{-1}(f^b) \\ &= m^{-1}(f_{pr}^b + f_g^b + f_{ext}^b) \end{aligned} \tag{5.34}$$

Now the accelerometer measurement model (3.1) can be used to obtain $a_b^{b,i}$. But first the model will be algebraically manipulated as follows: $\bar{a}_b^{b,i} = R_i^b(\ddot{\bar{\xi}}_b^i - g_b^i e_3) + b_a^b + \eta_a^b = a_b^{b,i} - R_i^b g_b^i e_3 + b_a^b + \eta_a^b$. This allows to rewrite equation 5.34 to consider the raw measurements, biases and noise:

$$\begin{aligned} \bar{a}_b^{b,i} + R_i^b g_b^i e_3 - b_a^b - \eta_a^b &= m^{-1}(\bar{f}_{pr}^b + \eta_{f_{pr}} + R_i^b(m g_b^i e_3) + f_{ext}^b) \\ \bar{a}_b^{b,i} - b_a^b - \eta_a^b &= m^{-1}(\bar{f}_{pr}^b + \eta_{f_{pr}} + f_{ext}^b) \\ \bar{a}_b^{b,i} &= m^{-1}(\bar{f}_{pr}^b + f_{ext}^b) + b_a^b + m^{-1}\eta_a^b + \eta_{f_{pr}} \end{aligned} \tag{5.35}$$

This formulation allows to correct the external force. This enables the estimator to estimate both the external force and the pose of the UAV.
Note that the gyroscope measures the angular velocity and not the angular acceleration. Since the current sensor configuration doesn't allow to measure the angular acceleration directly, there can be no correction for the external torque. The observation mapping will now become the following:

$$\underbrace{\begin{bmatrix} \bar{\xi}_b^i \\ \bar{v}_b^{i,i} \\ \bar{m}^b \\ \bar{a}_b^{b,i} \end{bmatrix}}_{y} = \underbrace{\begin{bmatrix} \xi_b^i + R_b^i r_{ant}^b \\ R_b^i(v_b^{b,i} + \tilde{\omega}_b^{b,i} r_{ant}^b) \\ (R_b^i)^T m^i \\ m^{-1}(f_{pr}^b + f_{ext}^b) + b_a^b \end{bmatrix}}_{h(\breve{x})} + \underbrace{\begin{bmatrix} n_{gps,p} \\ n_{gps,v} \\ n_{mag} \\ m^{-1}\eta_a^b + \eta_{f_{pr}} \end{bmatrix}}_{n} \tag{5.36}$$

The correction noise for the accelerometer is now affected by m and $\eta_{f_{pr}}$. Thus $n_a = m^{-1}\eta_a^b + \eta_{f_{pr}}$. But note that in this case n_a doesn't denote the accelerometer noise, but the mixture.
As is common practise by now, the correctness will be checked. The observation vector is rewritten to: $y = \begin{bmatrix} y_1 & y_2 & y_3 & y_4 \end{bmatrix}$, the observation noise to $n = \begin{bmatrix} n_1 & n_2 & n_3 & n_4 \end{bmatrix}$ and the state vector is rewritten the same as for the prediction step.

$$\underbrace{\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{bmatrix}}_{y} = \underbrace{\begin{bmatrix} x_1 + x_3 r_{ant}^b \\ x_3(x_2 + \tilde{u_2} r_{ant}^b) \\ (x_3)^T m^i \\ m^{-1}(u_1 + x_6) + x_4 \end{bmatrix}}_{h(\breve{x})} + \underbrace{\begin{bmatrix} n_1 \\ n_2 \\ n_3 \\ m^{-1}\eta_1 + \eta_5 \end{bmatrix}}_{n} \tag{5.37}$$

Since r_{ant}^b,m and m^i are parameters known beforehand, the observation mapping $h(\breve{x})$ is also formulated correctly in terms of only the states and known parameters.

Chapter 6

Simulation: Tuning state estimators

This chapter will show all results obtained from validating the state estimator using Gazebo simulations. The simulation setup is explained in appendix E.
For each simulation, the scenario will be described first. Next, the parameters relevant to the simulation scenario will be analyzed. After that, the plots and graphs of the results are shared and examined. Lastly, results will be discussed and a conclusion will be made.

The simulation environment used is Gazebo with the RotorS simulator[Furrer et al., 2016]. In Gazebo, the fully-actuated hexarotor called BetaX is simulated as seen in figure E.1. To interface all needed software, the middleware Robot Operating System (ROS) has been used. ROS interfaces the simulation with the: state estimation software, simulated sensors, and the controller. The state estimators have been implemented in C++ using the Manifold Toolkit (MTK) [Hertzberg et al., 2011]. The software implementation of the state estimators is being described in D. Both the geometrical pose estimator and the geometrical pose wrench estimator have their node, which can be launched for each simulation. Moreover, both ROS nodes contain two source files. One source file implements the algorithmic part of the software, while the other file interfaces the state estimator to ROS.
The simulation setup is described in more detail in appendix E.

The first simulation, shown in section 6.1, aims to tune the geometrical pose estimator. This simulation only considers pose estimation to reduce the complexity in tuning parameters. The found set of parameters will serve as a baseline for tuning the following simulations. However, it is important to realize that some variables are used differently for the geometrical pose wrench estimator than for the geometrical pose estimator.
During the second simulation, in section 6.2, the UAV will engage in a static interaction with a wall. The interaction is static in the sense that the UAV will not move. However, the interaction wrench will change while performing the interaction.

6.1 Simulation 1: Tuning geometrical pose estimator

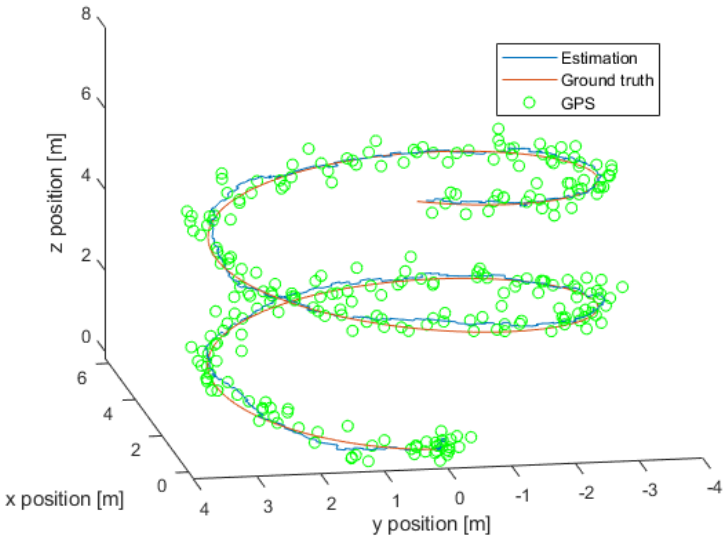
This simulation Will aim to validate the geometrical pose estimator. The estimator will be tuned and it will be checked in the estimator show a higher accuracy and precision than the raw measurements (especially, the GPS measurements).
The geometrical pose estimator is validated first, as this also simplifies the validation for the geometrical pose wrench estimator. This is true because the geometrical pose wrench estimator shows similarities with the geometrical pose estimator but also considers the dynamics additionally. However, note that the accelerometer is used differently for both estimators.

6.1.1 Description of the Scenario

In this scenario, the geometrical pose estimator will be tuned in an empty world. To fully explore the pose estimation every state should be triggered by the trajectory. That means that next to translation the UAV should also rotate.
Furthermore, notice the fictitious force term $\tilde{\omega}_b^{b,i}$ from equation 5.6. In order to also obtain effects from the fictitious force, the UAV should rotate while translating.

A trajectory that satisfies the above-mentioned requirements is a helix. The helix trajectory with one of the simulation results, which will be discussed later, is shown in figure 6.1 (The parameters in this figure are the favorite parameters for a GPS with 1 Hz). The UAV will move in the shape of a helix while pointing inwards. This makes sure the UAV translates while simultaneously rotating in the z-axis.
The helix can be automated using the GUI and the parameters are set as follows: Radius is 3, freq is 0.5, climbR is 0.25. The flight mode to trigger the helix is "Neg. Helix". The parameters aim to resemble the size of an operating space which could be encountered for real-world interactions in small environments.

Figure 6.1 Estimation, ground truth and GPS measurement of the helix trajectory.



6.1.2 Tuning parameters analysis

In section 5.3 the geometrical pose estimator has been formulated. It was shown that equation 6.1 can be used to predict the position deterministically and equation 6.2 corrects the position

using the GPS:

$$\dot{\xi}_b^i = R_b^i v_b^{b,i} \tag{6.1}$$

$$\bar{\xi}_b^i = \xi_b^i + R_b^i r_{ant}^b + n_{gps} \tag{6.2}$$

The velocity is obtained by using IMU measurements (6.3). This injects sensor bias and white noise of both the gyroscope and the accelerometer:

$$\dot{v}_b^{b,i} = -(\omega_b^{b,i} - b_\omega^b - \eta_\omega^b)^\sim v_b^{b,i} + \bar{a}_b^{b,i} + R_i^b g^i - b_a^b - \eta_a \tag{6.3}$$

The orientation can be predicted based on gyroscopic measurements (6.4) and the magnetometer can correct it (6.5):

$$\dot{R}_b^i = R_b^i \tilde{\omega}_b^{b,i} \tag{6.4}$$

$$\bar{m}^b = (R_b^i)^T m^i + n_{mag} \tag{6.5}$$

The challenge for the tuning is that some tuning parameters affect multiple states. This could lead to difficult situations since influencing a tuning parameter could lead to a performance increase in one state while decreasing the performance for another. For instance, if η_ω is influenced the ratio between n_{mag} and η_ω will change while also changing the ratio between η_a and η_ω. This affects both the estimation performance of R_b^i and $v_b^{b,i}$

In other to tune while avoiding the tuning parameter coupling as much as possible, R_b^i will be tuned first. This is done since R_b^i is not influenced by any other states. This allows to only focus on changing n_{mag} and η_ω. The position correction is only affected by $n_{gps,p}$. Therefore, the parameter $n_{gps,p}$ will be tuned secondly.
Lastly, η_a will be tuned. This affects both ξ_b^i and $v_b^{b,i}$. It is expected that a proper tuning for R_b^i and η_ω^b is already found. This suggests that the best ξ_b^i and $v_b^{b,i}$ should be found by solely tuning η_a.

6.1.3 Description of the results

As previously mentioned the tuning strategy is to first tune R_b^i by altering η_ω^b and n_{mag}. This will be done within case 1. Next in case 2 n_{gps} will be changed to find the best ξ_b^i error. Lastly η_a^b will be tuned in case 3 which should result into a good tuning for ξ_b^i and $v_b^{b,i}$.
This subsection will present the results using plots and error bar graphs contain the mean error and variance in the error. The exact values can be found in F.1.

In all cases. the error is obtained from subtraction with ground truth. However, the orientation error is calculated using quaternion and then transformed to Euler angles. This is done to avoid large peak in the error due to a change from $-\pi$ to π.
All last remark is that sometimes the duration of the simulation is longer than others. This is the case because when the simulation is started some parameters have to be set, such as the trajectory parameters. However, preferably errors are calculated for the same simulation time.
This is solved by finding the first local maximum and second local minimum for the y-position. As seen in

Figure 6.2 Region of the error is defined by finding the first local maximum and second local minimum for the y-position.

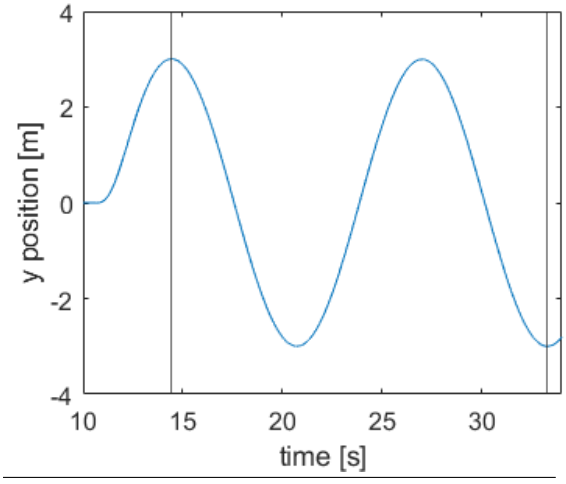


figure 6.2, the y position behaves like a sine function. This makes is simple to find a relatively large region for the error calculations. The vertical lines in the presented plots will denote the region used to calculate the error.

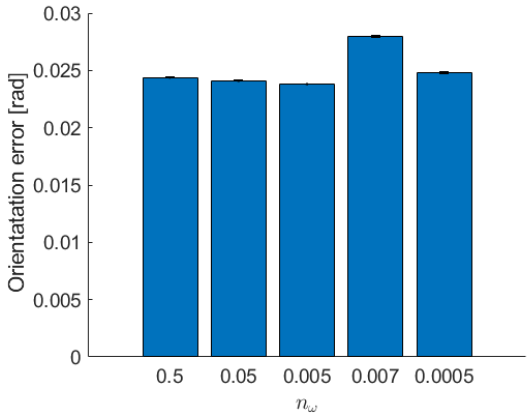
Case 1: tuning Rotation

Figure 6.3 shows the simulation result of tuning the angular velocity measurement noise η_ω (gyroscope) and orientation observation noise n_{mag} (magnetometer). Each bar represents a simulation for a certain tuning parameter. Figure 6.3a shows an error bar graph for tuning the gyroscope, 6.3c shows the same plot but showing standard deviation instead of variance, 6.3b is the error bar graph of the magnetometer tuning, and 6.3d is again similar but shows the standard deviation.
The reason that two similar error bar graphs are shown is that the mean error is a lot larger than the variance, resulting into that differences among the variances are hard to observe. Therefore, plots using the standard deviation (square root of variance) are shown as well.
Lastly, all error bar graphs shown for case 1 are of all axes summed. The only axis which is excited is the z-axis. Therefore, the z-axis shows larger errors than the x-axis and y-axis, as can be seen in table F. In order to keep the number of plots low but still involve the other axis, the values are summed.

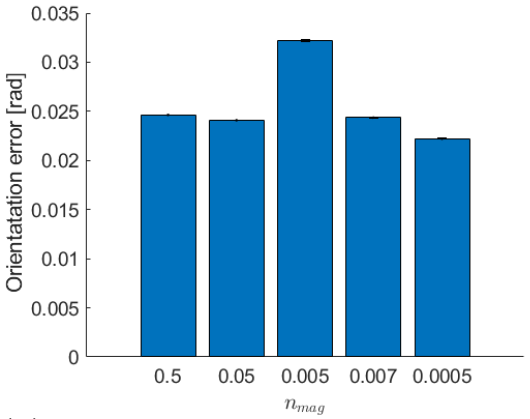
Figure 6.3a shows that the lowest mean error is observed for the $\eta_\omega = 0.005$ with a value of $e_{R,\mu} = 0.0238rad$. Note that the differences among the simulations are small. Furthermore, it shows that all mean errors are relatively close to each other, indicating a static error.
In order to compare the variation of the error data, one can look at figure 6.3c. This figure also shows that $\eta_\omega = 0.005$ is the simulation with the best performance. For this simulation the variance showed to be $e_{R,\sigma^2} = 0.000032rad$. Similar to the mean error, do note that the differences among simulations are small.
Figure 6.3b shows the error bar graph for tuning the magnetometer used for correcting the orientation. For tuning the magnetometer, the η_ω was fixed to 0.005. In this case, the value $n_{mag} = 0.0005$ shows the lowest mean error, with the value of $e_{R,\mu} = 0.0222rad$. Figure 6.3d shows that the lowest standard deviation is not observed for $n_{mag} = 0.0005$ ($e_{R,\sigma^2} = 0.000039$), but for $n_{mag} = 0.5$ ($e_{R,\sigma^2} = 0.000026$). Nevertheless, since $e_{R,\mu}$ is significantly lower, the upper bound of $n_{mag} = 0.0005$ is still the lowest. Therefore, $n_{mag} = 0.0005$ is the preferred value for the orientation observation noise.

In order to observe what causes the standard error, figure 6.4a shows the error for the yaw angle. It can be observed that is a standard error for the yaw estimation around the mean $e_{R,\mu} = 0.0222$. The estimated yaw versus the actual yaw is shown in 6.4b, and a zoomed-in version of the first jump is shown in 6.4c. It this version, it becomes clearly visible that the estimation is delayed. This delay causes the estimation to always have an error, while the rotation axis is changing.

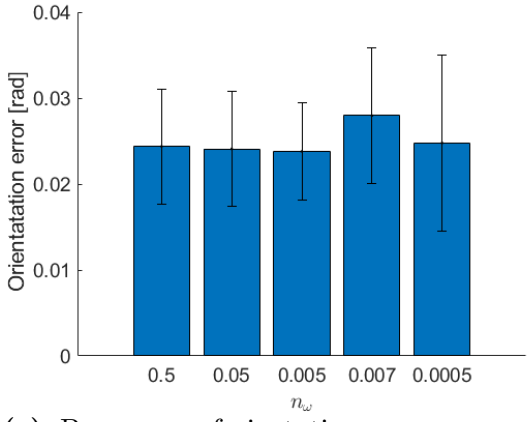
Figure 6.3 Simulation 1: case 1. Results of tuning rotation prediction and correction. Bar error plots show mean error between ground truth and estimation and variance of the error for each noise parameter value.



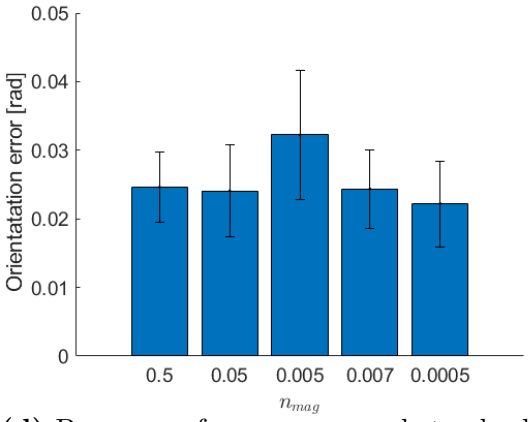
(a) Bar error of orientation mean error and variance for tuning η_ω with $n_{mag} = 0.05$.



(b) Bar error of mean error and variance for tuning n_{mag} with $\eta_\omega = 0.005$.

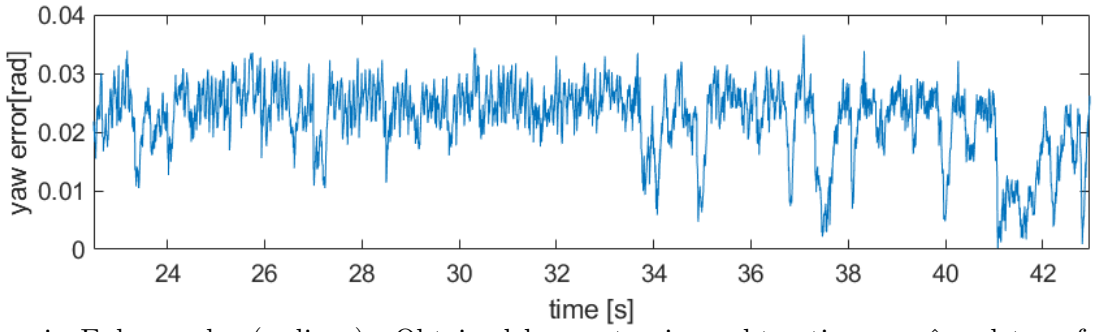


(c) Bar error oforientation mean error and standard deviation for tuning η_ω with $n_{mag} = 0.05$.

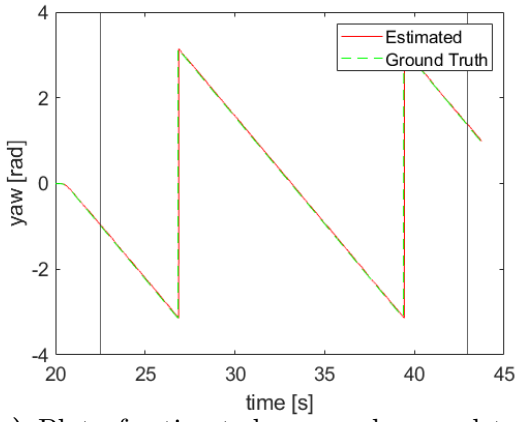


(d) Bar error of mean error and standard deviation for tuning n_{mag} with $\eta_\omega = 0.005$.

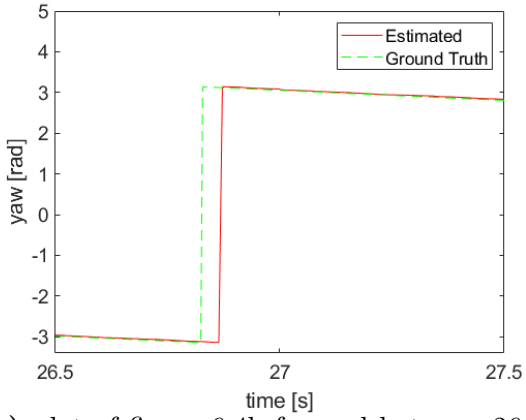
Figure 6.4 Yaw error, and estimation plotted with ground truth (obtained from simulation). The error is calculated



(a) yaw in Euler angles (radians). Obtained by quaternion subtraction $q \ominus \hat{q}$ and transforming to Euler angles.



(b) Plot of estimated yaw and ground truth in Euler angles (radians).



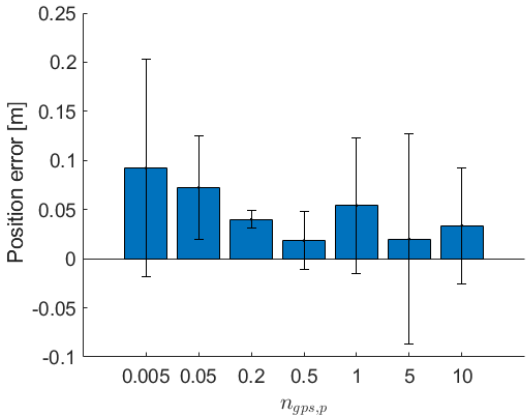
(c) plot of figure 6.4b focused between $26.5s$ and $27.5s$.

Case 2: Tuning the GPS correction

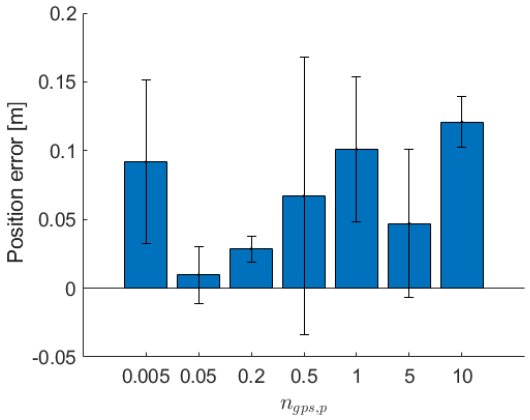
In this case the GPS position correction is tuned. The results are shown in figure 6.5.
For all axes, there is a considerable variation for the mean error. This is likely because the update rate of the GPS is 1 Hz and the simulation time is approximately 30 seconds. As the noise variance of the GPS is simulated to be $0.2m$, it is likely the mean errors deviate a lot for the low amount of samples.
Nevertheless, for all axes the variance is lowest for $n_{gps,p} = 0.2m$. Besides, there seems to be a large improvement on the variance of the simulated GPS.

In order to check that the variance should be trusted more than the mean error, figure 6.5d and 6.5e are added. It can be seen that the estimation for $n_{gps,p} = 0.2m$ shows less deviation from ground truth than $n_{gps,p} = 0.5m$. However, the deviations for $n_{gps,p} = 0.5m$ are both above and under the ground truth. As there is an error for both the plus and minus direction, the mean will result into a lower value than if both deviations would be in the same direction.

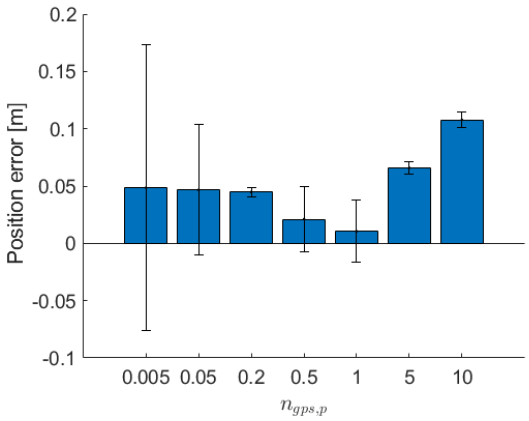
Figure 6.5 Simulation 1: case 2. Result of tuning position correction $n_{gps} = 0.5m$



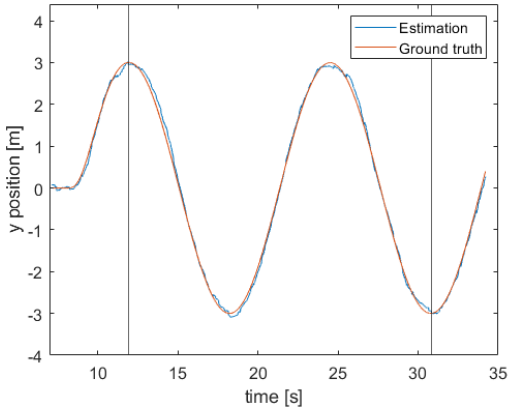
(a) Error bar graph of x-position.



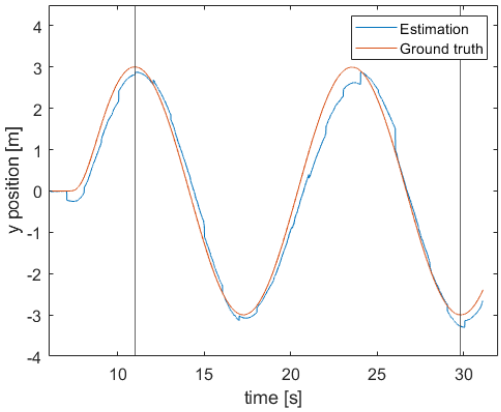
(b) Error bar graph of y-position.



(c) Error bar graph of z-position.



(d) Plot of y-position for $n_{gps} = 0.2m$



(e) Plot of y-position.

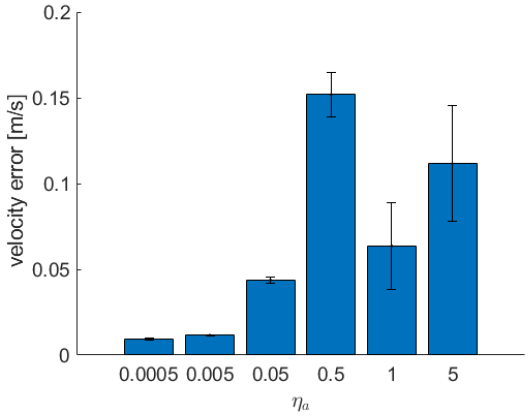
Tuning the accelerometer

The accelerometer affects the velocity directly. As all other parameters are tuned now, the best η_a should also result in the best position estimation. The results are shown in figure 6.6.

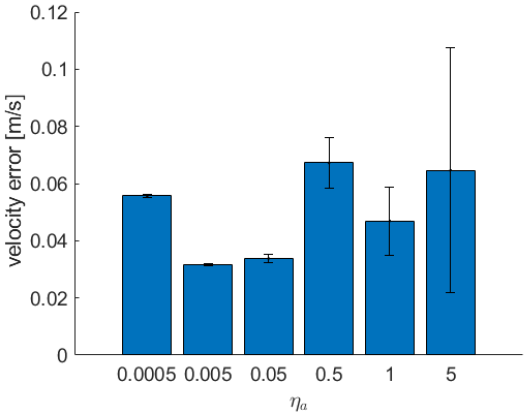
Figure 6.6a, 6.6b and 6.6c show the error bar graphs of the velocity for this case. The lowest value seems to appear for $\eta_a = 0.05$ for the mean error. The lowest variance is found for $\eta_a = 0.005, 0.05$ and 0.5. The RMS seems to be affected by both the mean and variance. The best value is found for $\eta_a = 0.05$

Figure 6.6a, 6.6b and 6.6c show the velocity estimation errors for tuning with η_a. First of all, for the velocity, the lower values seem to lead to fewer estimation errors than the high values for the x-axis and the y-axis. Conversely to the horizontal axes, the z-axis shows the lower mean error for the higher values of η_a. The differences can be explained as the trajectory of z is linear, while the trajectory of the horizontal axes is sinusoidal. The lowest variance if found for $\eta_a = 0.005m/s^2$.
Opposed to the velocity, the lowest variance found for the position is $\eta_a = 0.05m/s^2$. Note that the jump to the second best estimation ($\eta_a = 0.005m/s^2$) is relatively large. As the difference from $\eta_a = 0.005$ to $\eta_a = 0.05$ is relatively smaller, $\eta_a = 0.05$ is considered to be the best estimation.

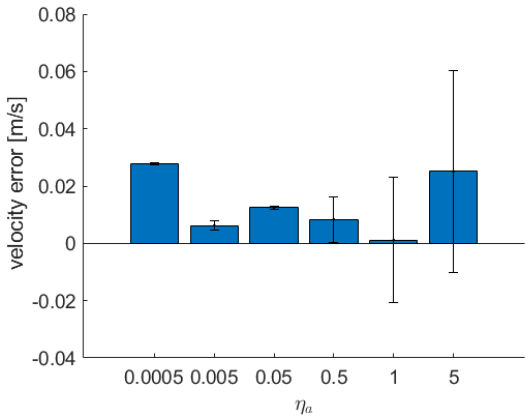
Figure 6.6 Simulation 1: case 3. Results of tuning η_a for both the velocity and position estimation.



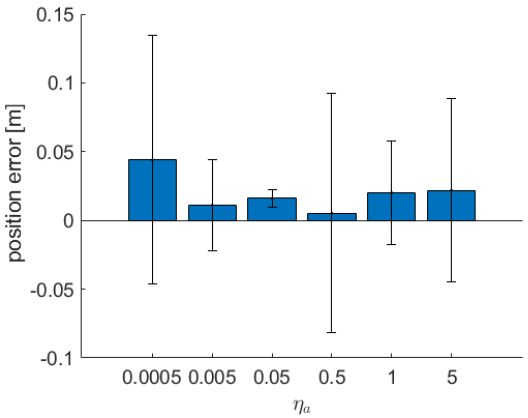
(a) Error bar graph for x velocity.



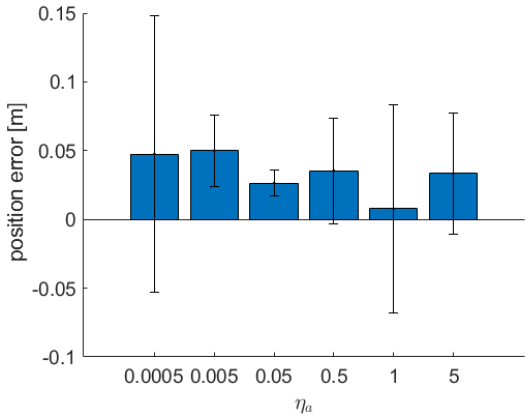
(b) Error bar graph for y velocity.



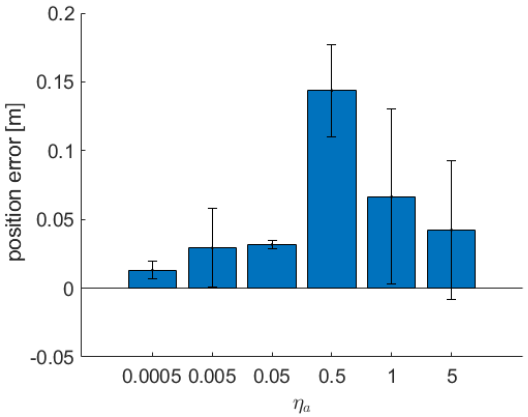
(c) Error bar graph for z velocity.



(d) Error bar graph for x position.



(e) Error bar graph for y position.



(f) Error bar graph for z position

6.1.4 Discussion on the results

In this subsection, it will be discussed what the best tuning parameters where. If it differs from the simulated variance, it will be analyzed what causes the difference.

For case 1, the best tuning is found for $\eta_\omega = 0.005 rad/s$ and $n_{mag} = 0.0005 rad$. This is a slight deviation from the simulated variance of $0.007 rad/s$ for the accelerometer and $0.007 rad$ for the magnetometer.

The noise function of the orientation prediction is $g(\eta) = -R_b^i \tilde{\eta}\omega$, while the UKF on a manifold algorithm presented in section 4.4 is as follows: $\check{P}_k = \sum_{i=1}^{2n} w_m^{[i]} (\check{\mathcal{S}}_k^{[i]} \ominus \check{\mathcal{X}}_k)(\check{\mathcal{S}}_k^{[i]} \ominus \check{\mathcal{X}}_k)^T + Q$. As seen, the measurement noise is added using the diagonal measurement covarance matrix Q. However, the function $g(\eta)$ does not result into a diagonal matrix in real space.
The fact that $g(\eta)$ is not implemented directly into the UKF is no large issue, as the UKF solves the non-linearity by means of the sigma points. Nevertheless, it could still be that due to the constant addition of the measurement covariance matrix Q, the estimation accuracy can be higher for deviating values.
Nevertheless, for the simulation where $\eta_\omega = 0.005 rad/s$ and $n_{mag} = 0.0005 rad$, the mean error for the z-axis is $0.0240 rad$ and the variance is $3.2314e(-5) rad$ (This can be found in appendix F). The z-axis shows the largest errors and the variance is substantially lower than the simulated noise for the magnetometer $0.007 rad$. Therefore, it will be concluded that the implementation and tuning parameters lead to a proper estimation of the orientation

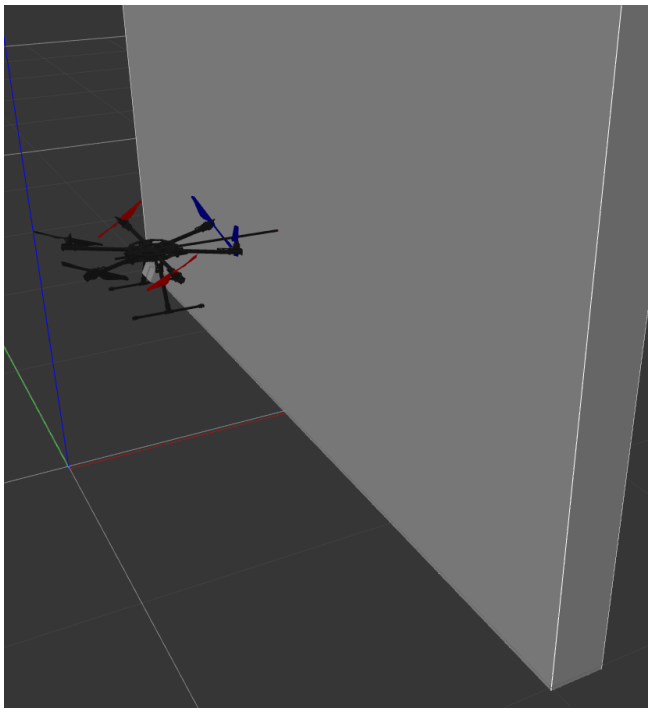
For case 2, the optimal value was found for the true simulated variance. This conclusion had been based on the variance, as the variance was the lowest for $n_{gps,p}$. The mean error was not lowest for this axis. It is expected that the mean errors are not a trustworthy indicator for the best estimation. In order to check this expectation, one plot of the favorite $n_{gps,p} = 0.2m$ and a plot for $n_{gps,p} = 0.5m$ was added. This last plot had a lower mean error but a larger variance compared to $n_{gps,p} = 0.2m$. The plot shows that the estimations for $n_{gps,p} = 0.5m$ deviate more from ground truth, but in both the plus- and minus direction. Therefore, the mean resulting mean error is lower than it should be.
The mean errors found for the $n_{gps,p} = 0.2$ are 0.04, 0.0286 and 0.0448 m correspondingly for x,y and z. The variance found is $0.0093m$, $0.0093m$ and $0.0039m$ for x,y and z. Especially the variance is considerably lower than the simulated GPS signal.

For case 3, the optimal value was found differently for the velocity and the position. The value, resulting in the lowest error variance for the velocity is $\eta_a = 0.005 m/s^2$ and for the position $\eta_a = 0.05 m/s^2$. This indicates that another value might not be optimal. It is suspected that η_ω is not the optimal value, as that value deviates from the true value.
Nevertheless, the difference between the variance for $\eta_a = 0.005 m/s^2$ and $\eta_a = 0.05 m/s^2$ are small
Lastly, it should be remarked that the GPS velocity signal has not been used for this simulation. When using a magnetometer and GPS position signal, the full pose can be corrected for integration drift. However, in real-world applications magnetometer are not very robust, since they are easily disturbed by for instance magnetic distortions. Since the GPS velocity is expressed in the inertial frame $\bar{v}_b^{i,i}$ and the velocity in the state is expressed in the body frame $v_b^{b,i}$, the GPS velocity measurement could be used to correct the orientation.
However, in simulation it was observed that the estimation performance decreased whenever GPS velocity signals were considered. It was later found that this was due to a mistake in the observation mapping. It turned out that the rotation matrix in the velocity part of equation 5.25 was implemented as $(R_b^i)^T$ instead of R_b^i. This mistake was corrected when discovered, but this was done after all simulations where performed.

6.2 Simulation 2: Force estimation for a static interaction

In this scenario, the UAV will statically interact with a wall. The UAV will not move but the interaction force will increase. The estimator to tune will be the geometrical pose wrench

Figure 6.7 BetaX in gazebo simulation environment. The world used for this scenario is called "wall_of_int", which is an empty world with a wall. The side of the wall is placed at $x = 0.45$ and the wall is equipped with a force sensor which measurements can be used as ground truth for the interaction scenario.



estimator.

6.2.1 Description of the scenario

Figure 6.7 shows the betaX in the gazebo simulation environment. The object of interactions is a flat wall. The wall is equipped with a force sensor from the gazebo library. As soon as an object interacts with the wall, the force sensor provides the location of the interaction and the associated force and torque.
A side note is that the force measurements are twice as large as the actual force. Also, the measurements jump between zero and the measured value. In order to work with the measurements more easily, a measurement is set to the previous measurement if it equals zero. Also, the measurements are multiplied with 0.5 in order to obtain the true force.

During this scenario the UAV will go to a position right in front of the wall. In this case that is $\begin{bmatrix} x & y & z & yaw \end{bmatrix} = \begin{bmatrix} 0.45 & 0 & 1 & 0 \end{bmatrix}$. The rosbag recording will start as soon as the UAV stands still at that position. From then the setpoint will be placed behind the wall. This will let the UAV apply a force against the wall. The setpoints chosen are $x = \begin{bmatrix} 0.55 & 0.65 & 0.85 & 1.05 \end{bmatrix}$ and the other coordinates stay the same. After 10 seconds wall clock time the next setpoint will be chosen untill none are left.

6.2.2 Analysis of tuning parameters

Section 5.4 formulates the pose wrench estimator. It was shown that the following equations are used for force estimation:

$$\dot{f}_{ext}^{b} = \eta_{f_{ext}} \tag{6.6}$$

$$\bar{a}_{b}^{b,i} = f_{ext} + f_{pr}^{b} + b_{a}^{b} + n_{a}^{b} \tag{6.7}$$

Equation 6.6 predicts the external force based on $\eta_{f_{ext}}$, where $\eta_{f_{ext}}$ can be chosen to tell how quickly the external force is expected to change. The external force is corrected using (6.7). The accelerometer measurements are used as observation. The propeller force f_{pr}^{b} is used as measurement input. In this case f_{pr}^{b} already has gravity compensation. Therefore the previously encountered gravity term can be neglected. f_{pr}^{b} and f_{ext}^{b} can be used to map towards the accelerometer measurements.

The equations show that the tuning parameters involved in the force estimation are: $\eta_{f_{ext}}$, η_{pr}^{b} and n_{a}^{b}. These parameters will be changed in this scenario to study their effects on the force estimates and to find a favorite set.
Note that b_{a}^{b} is estimated similarly to f_{ext}. Both use similar models (of course with a different tuning) and both are corrected by the same observation. This complicates their estimation as both estimates can easily be mistaken for each other. In order to simplify the estimation process b_{a}^{b} will not be considered for this scenario.

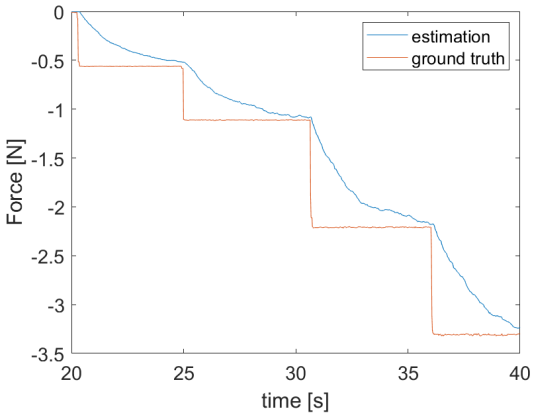
6.2.3 Description of the results

For this scenario, results are obtained for two cases. In the first case, the effects of tuning parameter $\eta_{f_{ext}}$ are studied. Next, the parameter n_a will be tuned to investigate effects of the accelerometer noise tuning on the wrench estimation.

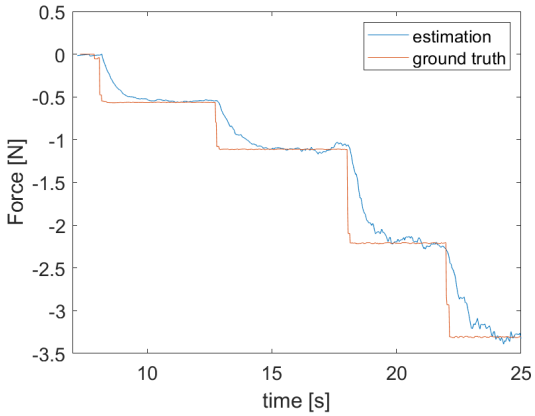
Case 1: Effects of $\eta_{f_{ext}}$

Figure 6.8 shows the external force estimations for different values of $\eta_{f_{ext}}$.
In figure 6.10a and 6.10b it is very clear to observe that the estimation can't follow the ground truth quick enough. As the value for $\eta_{f_{ext}}$ is too low, the estimator expects the external force to change more slowly than it does. This causes large deviations relative to ground truth, especially during the interaction force change.
In figure 6.8c and 6.8d the estimation seems to change more quickly. However, for $\eta_{f_{ext}} = 0.05N/s$, deviations are still observed whenever the interaction force changes. Furthermore, it can be seen that the estimation is more noisy than for the lower values.
In figure 6.8e and 6.8f most noise can be observed. This shows that as $\eta_{f_{ext}}$ increases, the estimation indeed changes more quickly, but this also causes the estimation to capture noise instead of only the force contribution.

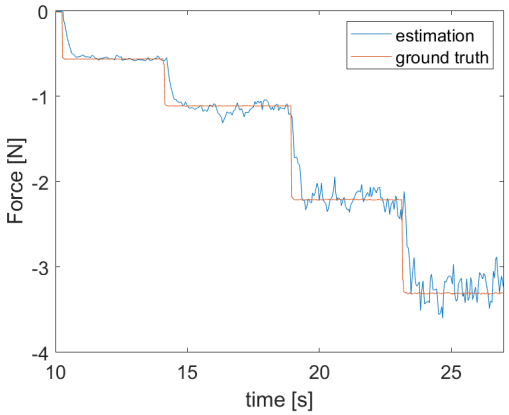
Figure 6.8 Simulation result for simulation 2 case 1. Plots show the estimated external force against ground truth.



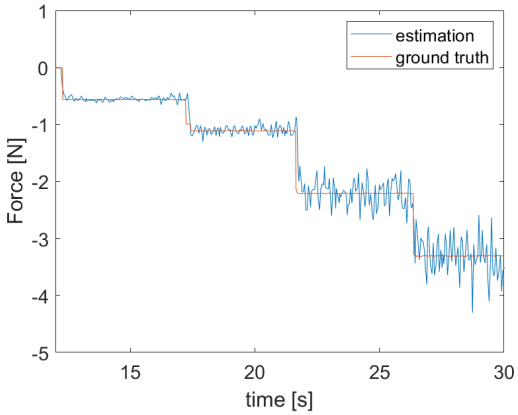
(a) $\eta_{f_{ext}} = 0.0005N/s$



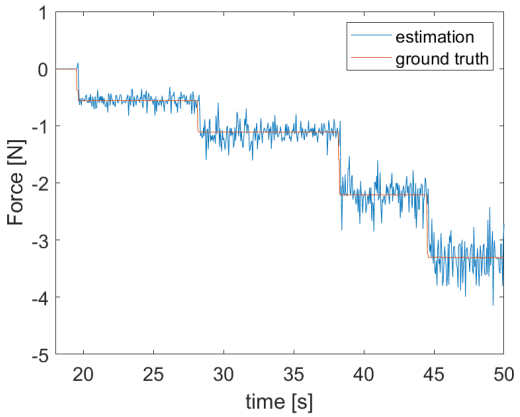
(b) $\eta_{f_{ext}} = 0.005N/s$



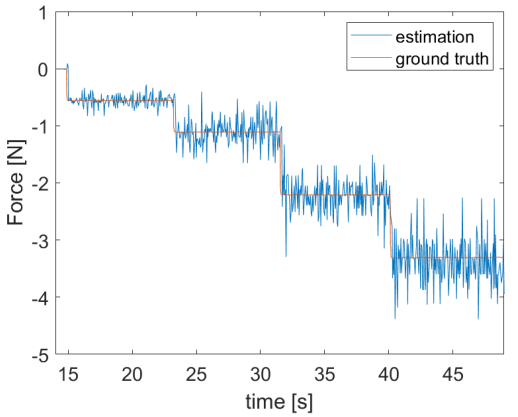
(c) $\eta_{f_{ext}} = 0.05N/s$



(d) $\eta_{f_{ext}} = 0.5N/s$



(e) $\eta_{f_{ext}} = 5N/s$



(f) $\eta_{f_{ext}} = 50N/s$

Case 2: Effects of η_a

Figure 6.9 shows a the bar graphs for the tuning of n_a. The first bar graph contains the mean error, the second one contain the variance in the noise and the last one contains the *root mean square* error (RMS)

For smaller values of n_a there is a relatively low mean error. $n_a = 0.005m/s^2$ and $n_a = 0.5m/s^2$ shows the lowest mean error. For $n_a = 0.5$ the mean error seems to show a local peak. This peak

is approximately equal to the mean error observed for $n_a = 1$ $n_a = 5m/s^2$ and $n_a = 10m/s^2$. For $n_a = 50m/s^2$ the mean error is very high. A glance at the plot in figure 6.10 shows that for the high value the estimation shows a slow following behaviour similar to those observed for low $\eta_{f_{ext}}$.
The bar graph for the error variance shows that higher values for n_a result into less variance. Specifically for $n_a = 5m/s^2$, $10m/s^2$ and $50m/s^2$ the variance is relatively low. The variance is a bit higher for $n_a = 0.05m/s^2$ and $1m/s^2$. A smaller peak is observed for $n_a = 0.5m/s^2$ and a high peak is observed for $n_a = 0.005m/s^2$. The RMS shows to mimic the variance.
Figure 6.10 shows two plots, where one is given for $n_a = 1m/s^2$ and one for $50m/s^2$. As suggested before, the lower value shows more noise but quickly reacts to a change of interaction signal. The second plot contains less noise but shows a small error whenever the interaction force changes.

Figure 6.9 Error Bar graph of result for simulation 2 case 2. $\eta_{f_{ext}}$ is chosen The static interaction has been performed for different values of n_a.

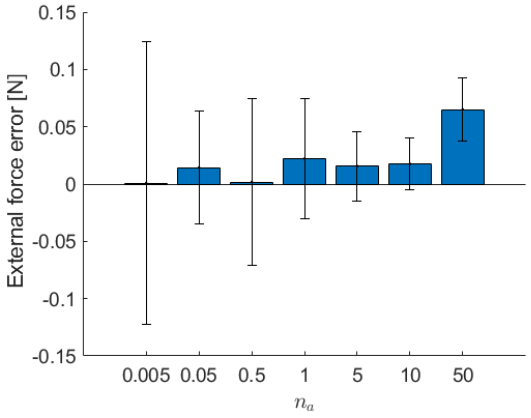
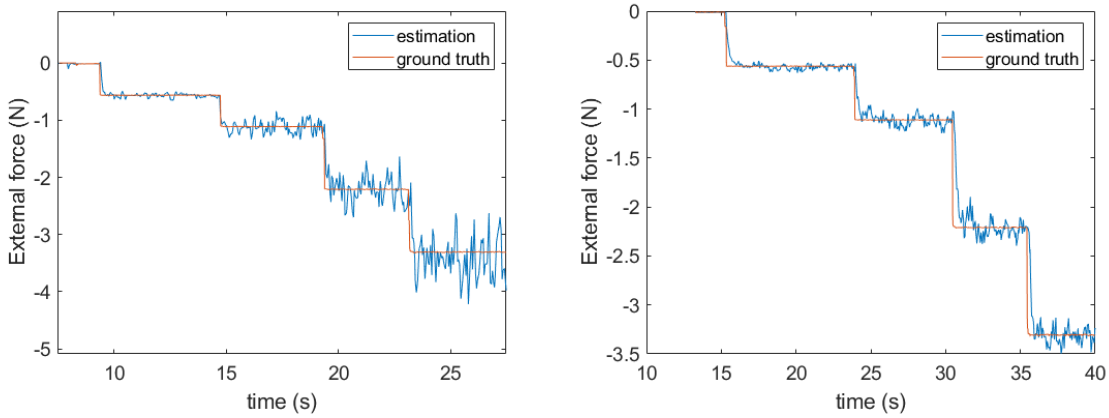


Figure 6.10 Two plots indicating the difference for a relatively low n_a, namely $n_a = 1m/s^2$ and the relatively high $n_a = 50m/s^2$



(a) $n_a = 1m/s^2$ **(b)** $n_a = 50m/s^2$

6.2.4 Discussion of the results

From the plot in figure 6.8 it becomes clear the $\eta_{f_{ext}}$ indeed affects how quickly the estimator think the external force changes. When the external force is following the ground truth slowly a relatively high mean error was observed. For $\eta_{f_{ext}}$ values of 0.5, 5 and $50N/s$ the mean error

is relatively low.
However, these higher values also contain more noise than lower values. This is because for higher values the external force is expected to change more quickly. This cause the estimation to fit on noise. For the lower values, the estimation is less noisy, but too low values causes the external force estimation to change too slowly.
A remark is that the optimal value for $\eta_{f_{ext}}$ is dependent on the trajectory of the desired interaction force. If the actual interaction force changes more slowly, $\eta_{f_{ext}}$ can be lowered. However, for this simulation, the interaction force changed drastically (almost like a step). This means that the value for $\eta_{f_{ext}}$ will also need to be relatively large, causing it to also capture some noise.

It shows from the bar graph in figure 6.9 that a higher value for n_a result into a more precise estimation. Where the best value are observed for $n_a = 5m/s^2, 10, m/s^2 and 50m/s^2$.
Nevertheless higher n_a results into less accuracy as the mean error increases. This can particularly be observed in the plot in figure 6.10. The plot with the highest $n_a = 50m/s^2$ shows that the force estimation is behind relative to the ground truth. As can be seen in equation 6.7 the accelerometer correct the estimation. By putting to less trust in the accelerometer the external force estimation cannot react to rapid changes.
The values for n_a favored for the geometrical pose wrench estimator are $n_a = 5m/s^2 and 10m/s^2$.

Chapter 7

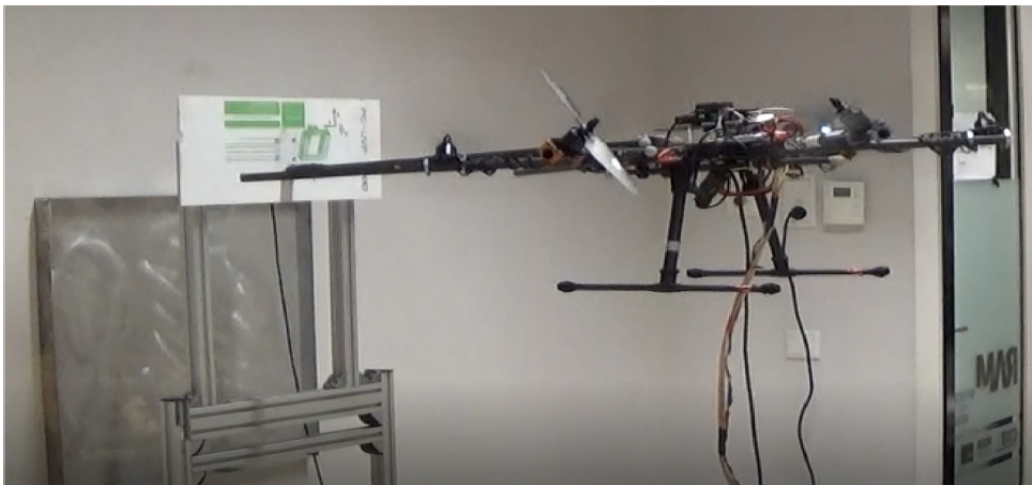
Experimentation: Validation on manipulated real-world data

7.1 Experiment setup

In this chapter, the state estimator will be used to manipulated real-world data. Note that this thesis is conducted during the restrictions of the COVID-19 pandemic. Therefore, a dataset of an earlier conducted experiment has been used and manipulated to also include GPS measurement data.

The conducted experiment is described in [Rashad et al., 2019a]. During this experiment, a fully-actuated hexarotor UAV applies a contact force onto a vertical surface rigidly connected to an ATI mini40 force/torque sensor (ATI Industrial Automation), as seen in figure 7.1. This experiment aims to exert a contact force up to the UAV's rotor limits. The UAV flight is done in an indoor lab environment equipped with a motion-capture system (MoCap). The dataset of this experiment contains all IMU data, information about the involved forces and torques, and pose ground truth obtained from a motion-capture system. As will become evident in the remainder of this section, the data from the MoCap will be manipulated to resemble a GPS signal.

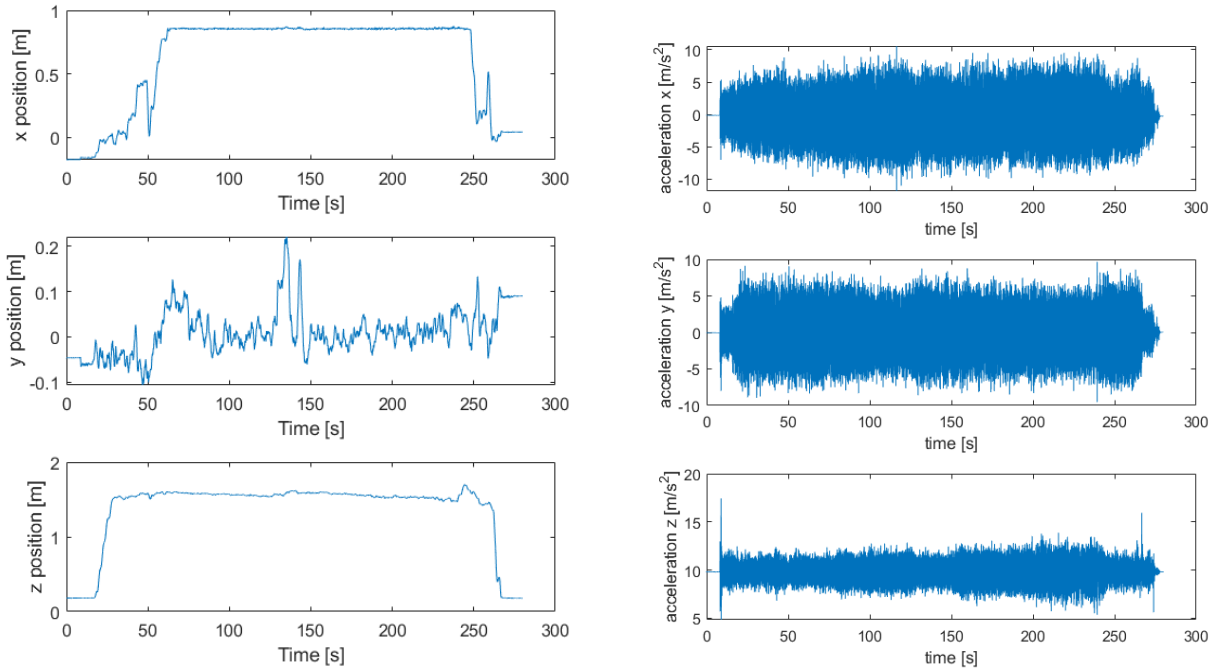
Figure 7.1 [Rashad et al., 2019a]



7.1.1 Dataset analysis

Figure 7.2a in 7.2 shows the measurements obtained by the MoCap and the IMU. First, it can be observed that the robot ascends. Next, the robot moves in the x-direction and around $t \approx 60s$ the robot reaches the vertical surface. At approximately $t \approx 250s$ the robot moves away from the surface and descend back to the initial position.
The plots on the right-hand side in figure 7.2 shows the acceleration per axis. Note that it is difficult to observe the exact acceleration and noise. Likely, the observed behavior of the acceleration is not due to a high noise but due to the UAV's oscillations. Up to $t \approx 8s$ accelerometer measurements without motion can be observed. The variance observed during the non-moving measurements is $\sigma_a^2 \approx 1.0003e-04 m/s^2$.

Figure 7.2 Data on the robot's motion. Plots to the left shows the position obtained from the MoCap. Plot to the right shows accelerometer measurements obtained from the IMU.



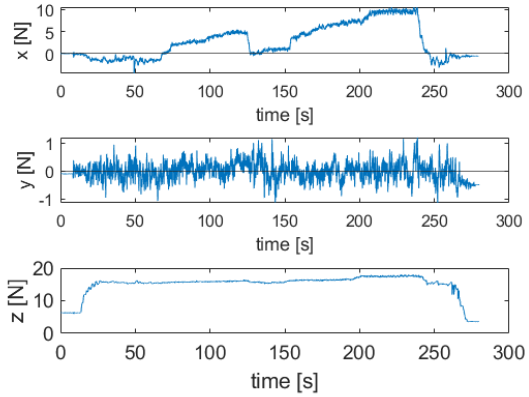
(a) Ground truth position ξ_b^i measured my MoCap **(b)** Linear acceleration $\bar{a}_b^{b,i}$ measured by IMU

Besides motion, the dataset also provides estimated propeller wrench $\bar{W}_{pr}^b$ and an estimated external wrench $\hat{W}_{ext}^b$. The force components can be seen in figure 7.3a and figure 7.3b. $\bar{f}_{pr}^b$ can be used as measurement input for the geometrical pose wrench estimator.
Besides, z component seen in figure 7.3a can be used to calculate the mass $m = \mu_{f_z}(t_1 : t_2) = 1.7kg$ and all components can be used to analyse the noise variance: $\sigma_{f_{pr}} = 0.1N$.
Next to this, the interaction wrench W_{int}^b measured by the force/torque sensor is available as well. The force component of this signal is seen in figure 7.3c, and this signal can be used as ground truth. Though the signal is noisy, it has no components due to aerodynamics or modelling errors. For instance, it can be observed in figure 7.3b that between $t \approx 15s$ and $t \approx 60s$, a positive force is estimated. since the UAV has not reached the vertical surface at this point and no interaction force is measured, this is contribution is likely a aerodynamic effect.

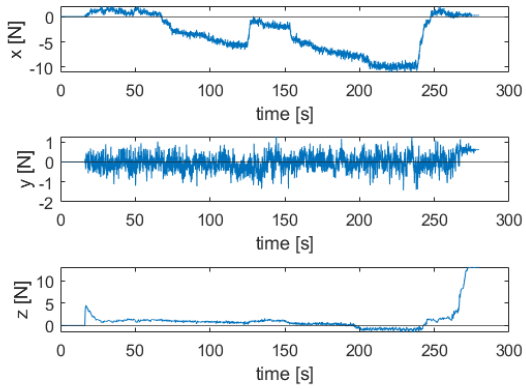
Lastly, the orientation of the UAV during this experiment is more or less equal to zero. There-

fore, the focus of the estimation performance will mostly be the position. //

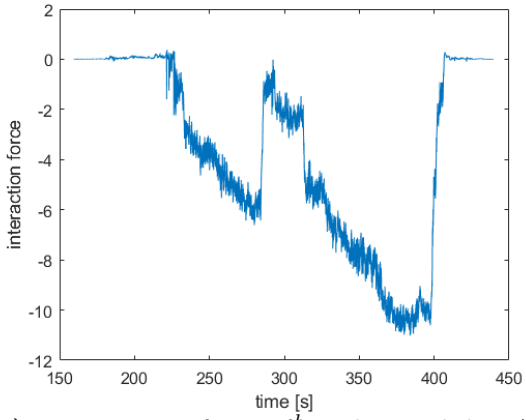
Figure 7.3 Forces acting on the UAV's body. Control force and external force are estimated by the observer in [Rashad et al., 2019a]. The interaction force is measured by a force/torque sensor attached to the vertical surface the UAV interactions with.



(a) Control force $\bar{f}_{pr}$.



(b) External force $\hat{f}_{ext}$.



(c) Interaction force f^b_{int} obtained by ATI mini40 force/torque sensor

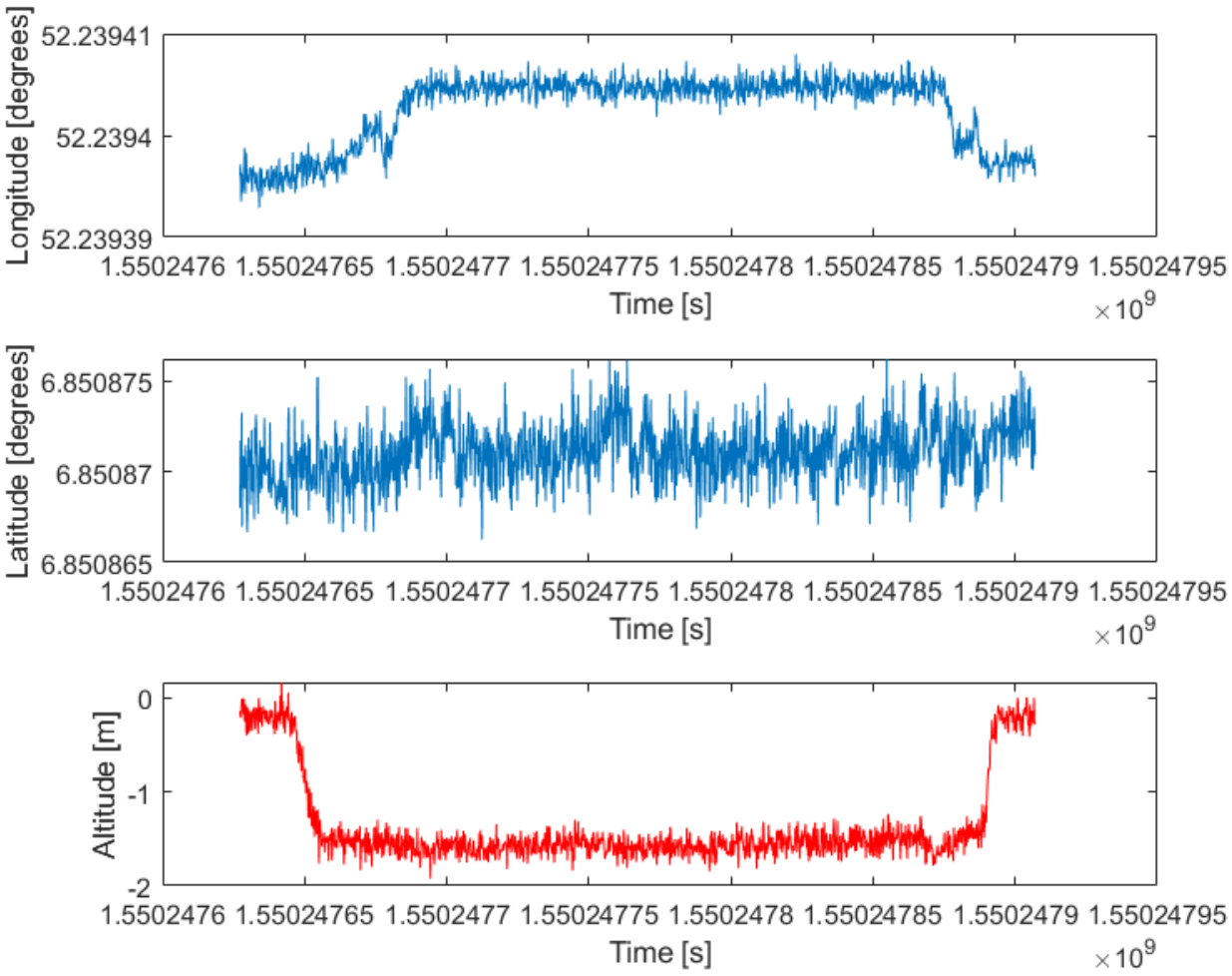
7.1.2 Global Positioning System (GPS) signal simulation

To obtain GPS measurements, the ground truth from the MoCap has been manipulated using the Matlab object *gpsSensor*. This object returns a simulated longitude, latitude and altitude from a GPS, using a positional signal. The object allows to set a desired update rate, reference location, and both horizontal and vertical accuracy.

As will become evident in 7.3, the update rate of standard GPS (approximately, 1 hz), will not suffice for the state estimator using this dataset. Therefore, a GPS real-time kinematics (GPS-RTK) will be simulated. GPS-RTK uses a second base station will operating to increase its accuracy. GPS-RTK can generally measure up to centimeter accuracy. Thus, the desired accuracy will be set to $\sigma^2_{gps,p} = 0.01m$. To show the update rate limitations, two versions will be simulated. In one version the update rate is set to $1hz$ and in the other to $5hz$. Furthermore, the GPS signal will only contain white noise and no biases. This provides more simplistic situations as there is no need to estimate the GPS bias. Note, that for real-world situations, GPS might suffer from biases.

The simulated signal is shown in figure 7.4. The longitude and latitude show a similar trajectory as the horizontal position from the Mocap (figure 7.2a). Note that the altitude is negative conversely to the MoCap data. This is no miscalculation, as the WGS-84 expresses the altitude as a negative value for heights above ground level.

Figure 7.4 Simulated GPS measurements based on the Mocap measurement in figure 7.2a, using the *gpsSensor* Matlab object.



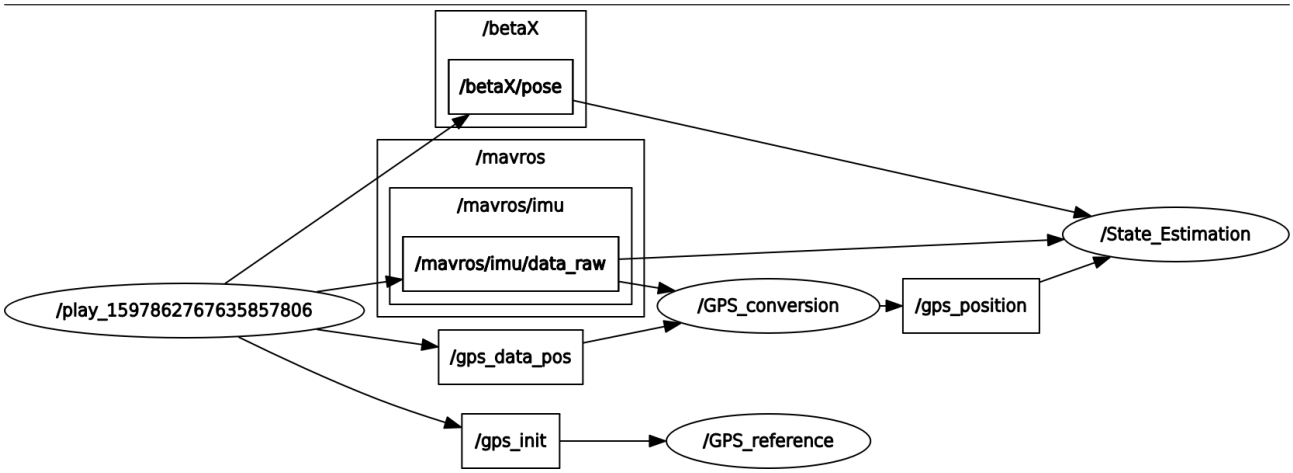
7.2 Experiment 1: Geometrical pose estimator on manipulated dataset

In this section, the geometrical pose estimator will use the signals from the dataset. The aim is to use the result for comparison with the geometrical pose estimator and to find limitations on estimating the position based on real-world IMU measurements and simulated GPs measurements.

As the performance of the estimation is dependent on the tuning parameters, the tuning parameter will be altered to find the best performance. Initially, the observation noise for the GPS $n_{gps,p}$ will be set to $0.01m$, as this is the true variance. As seen in section 7.1, it is indicated that the accelerometer noise η_a is at least $\sigma_a^2 \approx 1.0003e-04m/s^2$. However, this is not certain as it is difficult to observe the true variance in the data.

Figure 7.5 shows involved ROS nodes in this setup. The rosbag of the dataset contains all messages to be used for this experiment. The data does not contain magnetometer measurements or GPS velocity measurements appropriate to correct for the rotational drift. Therefore, the orientation is corrected using the MoCap ground truth from the message \betaX\pose. Similar to the simulation, the node \GPS_conversion converts the GPS data to the ENU-frame, using the simulated GPS position textbackslash gps_data_pos and the IMU measurements \mavros \imu \data_raw. The GPS reference is set using the node \GPS_reference and the message \gps_init. The \State_Estimation node denotes the geometretrical pose estimator. The node subscribes on \mavros \imu \data_raw for the prediction and both \gps_position and betaX \pose are used for the correction.

Figure 7.5 ROS nodes (ellipsoids) and messages (squares) used for experimentation with the geometrical pose estimator



7.2.1 Description of the result

The result description is divided in three parts. As it is not precisely known what the variance in the noise of the accelerometer is, the accelerometer will be tuned first. This will be done for a GPS update rate of 1 hz and 5 hz. Lastly, the observation noise $n_{gps,p}$ of the GPS position will be changed to check if the setup up is optimal.
The position obtained from the MoCap is used as ground truth for the error calculations. This chapter will show plots and bar graphs containing the mean error and variance in the error. The exact values can be found in appendix .

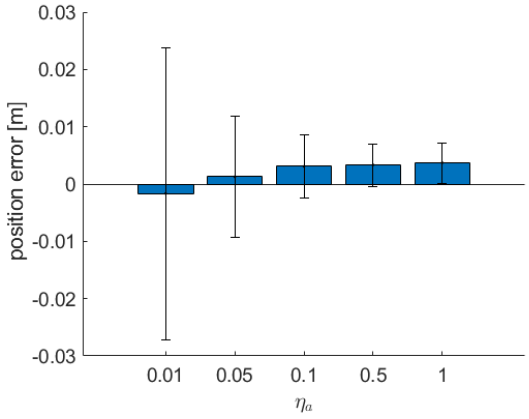
Case 1: Tuning accelerometer with GPS update rate of 1 hz

Figure 7.6 shows the result of tuning the accelerometer, where GPS signal corrects the position at a rate of 1 hz. For the x-axis, shown in figure 7.6a, the mean error is relatively small compared to the variance of the error. The variance decreases as η_a increases. In figure 7.6b, it can be seen that the mean error in the y-axis is relatively high compared to the variance and compared to the x-axis. Both the variance and mean error for the y-axis decrease for higher values of η_a. The z-axis, shown in 7.6c, doesn't show a clear preference for η_a and the errors are relatively small.

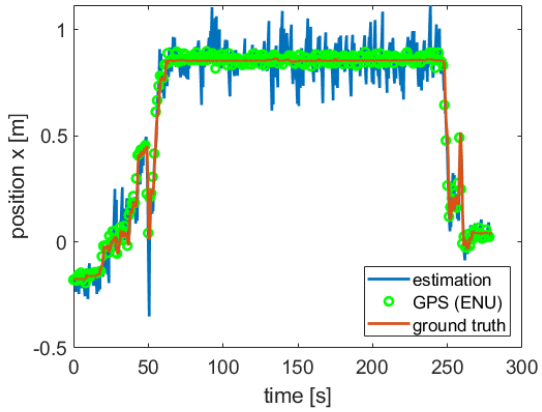
The observed preference for a higher η_a suggests that the true variance in the noise of the accelerometer is approximately 0.5 to 1. However, when looking at the plots of the estimation, GPS measurements, and ground truth, it is revealed that the estimation error is higher than

the error in the position measured by the GPS. This is very dominantly observed for the x-axis (figure 7.6d) and the y-axis (figure 7.6e). The effect is also observed in figure 7.6f, but the blue peaks are relatively small for this axis.

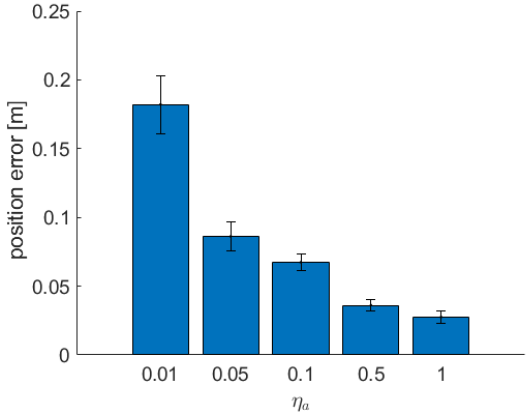
Figure 7.6 Position correction $n_{gps,p}$ fixed to 0.01 m, where measurements come in at 1 Hz. The left-hand side shows error bar graph. The blue bar denotes the mean error and the line denotes the variance. The right-hand side shows three plots, with the estimated position, GPS position and the ground truth, for the simulation with an acceleration variance $\eta_a = 0.5m/s^2$.



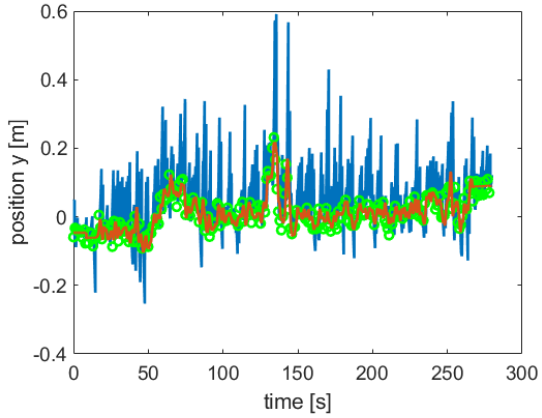
(a) Error bar graph for x position.



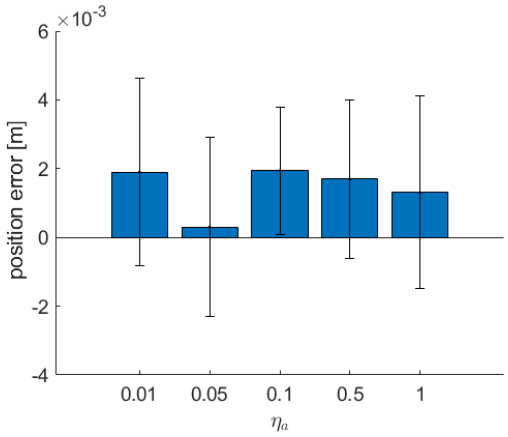
(b) Plots of x position.



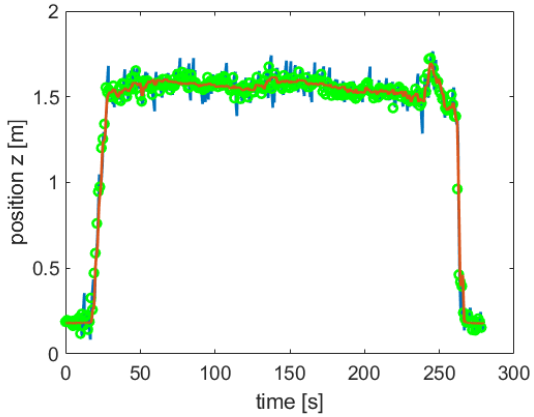
(c) Error bar graph for y position.



(d) Plots of y position.



(e) Error bar graph for z position.

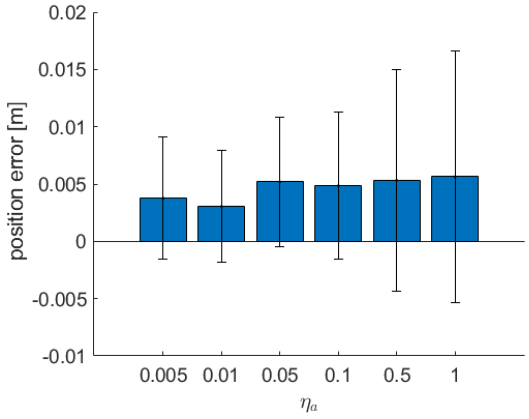


(f) Plots of z position

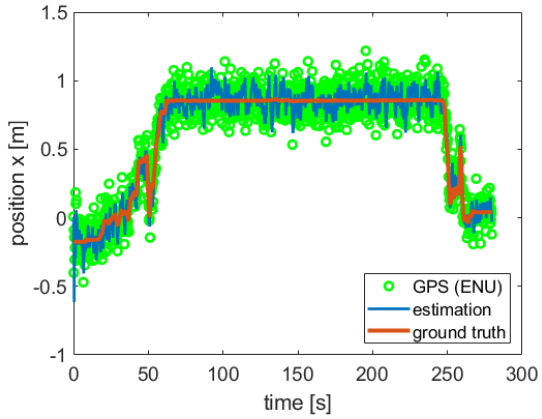
Case 2: Tuning accelerometer with GPS update rate of 5 hz

Figure 7.7 shows the result of tuning the accelerometer while position measurements from the GPS come in at 5 hz.
For this situation, figure 7.7a and figure 7.7e show that both the lowest mean error and variance are found for $\eta_a = 0.01m/s^2$ for the x-axis and z-axis. Similar to case 1, figure 7.7c shows a decreasing mean error for higher values of η_a. However, the variance is lowest for $\eta_a = 0.01m/s^2$. Especially plot 7.7b and plot 7.7f show a significant improvement of the estimated position relative to the position as measured by the GPS. This is also the case for the y-axis in figure 7.7d, but that plot shows the estimation has a slight static error. This can be observed as most blue line are slightly above the red lines.

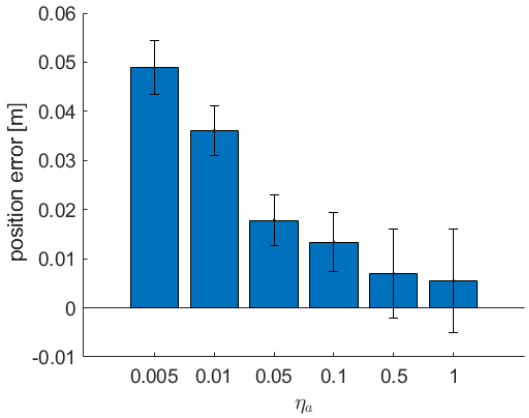
Figure 7.7 Position correction $n_{gps,p}$ fixed to 0.01 m, where measurements come in at 5 hz. The left-hand side shows error bar graph. The blue bar denotes the mean error and the line denotes the variance. The right-hand side shows three plots, with the estimated position, GPS position and the ground truth, for the simulation with an acceleration variance $\eta_a = 0.01 m/s^2$.



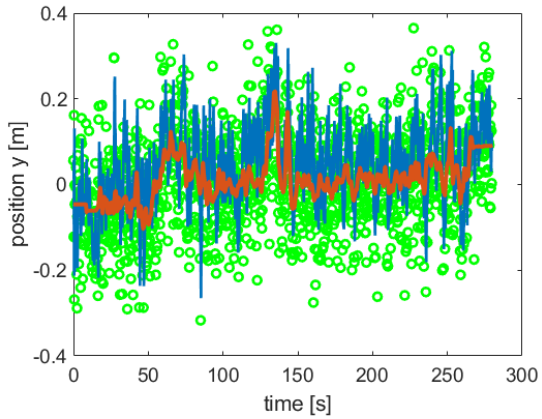
(a) bar x



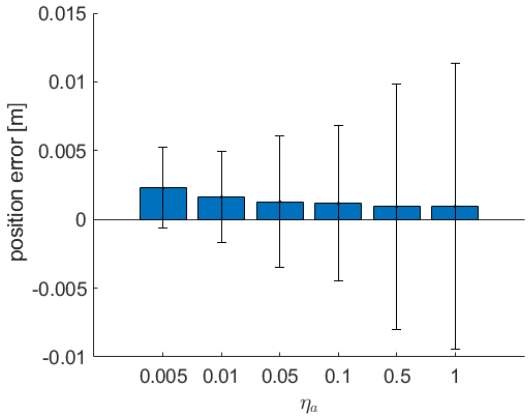
(b) $\eta_a = 0.01$



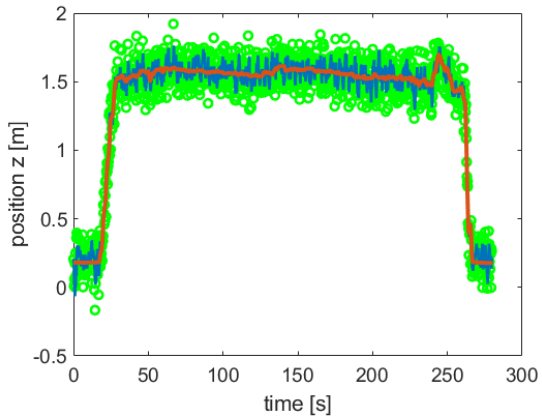
(c) bar y



(d) $\eta_a = 0.01$



(e) bar z



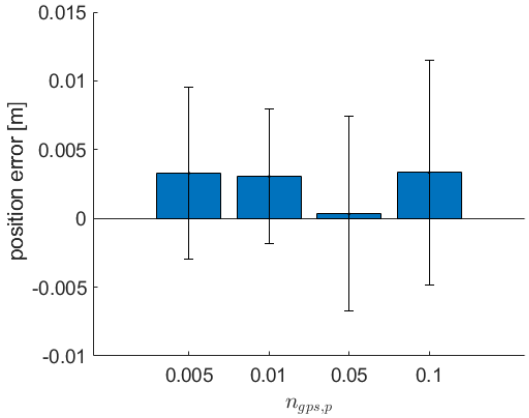
(f) $\eta_a = 0.01$

Case 3: Tuning GPS

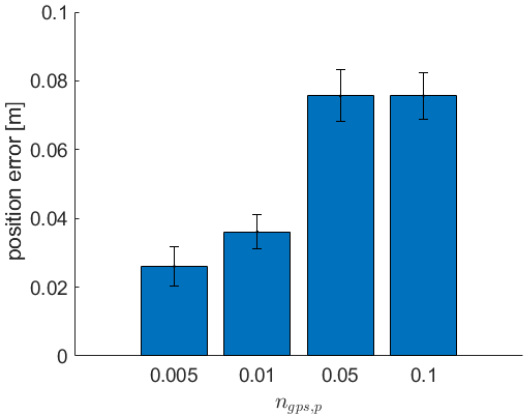
Lastly, it will be checked if the observation noise $n_{gps,p}$ is set to the correct value. As previously shown, the measurement noise η_a shows the best performance for $0.01m/s^2$. Therefore, η_a will be fixed to that value.

Figure 7.8a shows that the lowest variance is found for $n_{gps,p} = 0.01$ in the x-axis. The mean error is the same for each situation, except when $n_{gps,p} = 0.05$, where the mean error is significantly lower. Nevertheless, the variance is significantly higher for this situation.
Figure 7.8b and figure 7.8c show that the best variance is found for $n_{gps,p} = 0.01$. The mean error increase for higher values of $n_{gps,p}$.

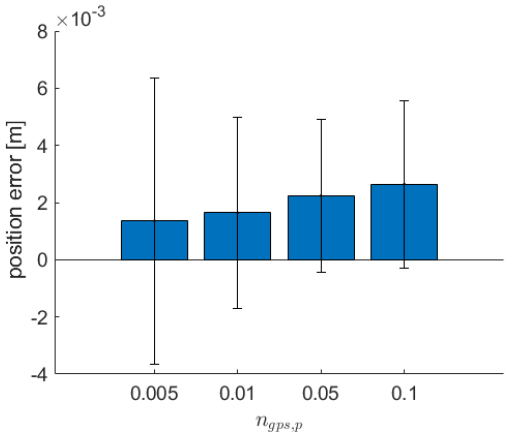
Figure 7.8 η_a fixed to 0.01 m/s^2 and position measurement come in from a GPS at 5 hz.



(a) error bar graph of x position



(b) error bar graph of y position



(c) error bar graph of z position

7.2.2 Discussion on the result

In this section, the geometrical pose estimator used signals from the dataset to estimate the UAV's position. The aim was to find limitations on the position estimation using real-world IMU measurements and simulated GPs measurements.

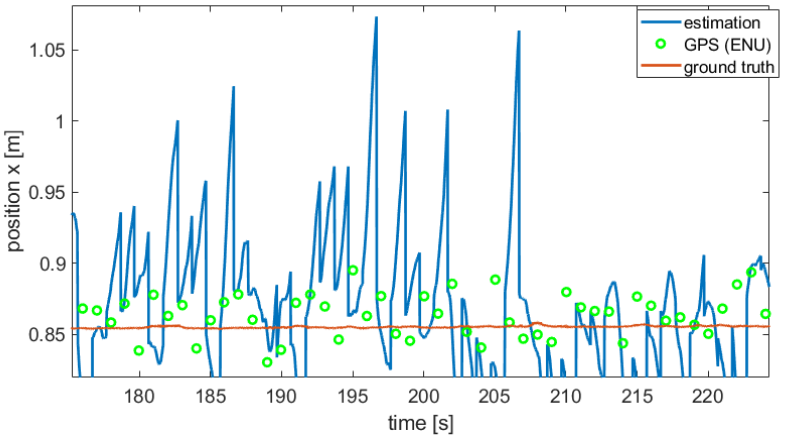
In the first case, the accelerometer was altered for a situation where GPS measurement came in at 1 Hz. The variance seemed to decrease for the x-axis and y-axis when η_a was set higher. Especially for the y-axis, the mean error decreased for higher η_a values. This shows that less trust in the accelerometer leads to an increase in performance.

Nevertheless, the estimation performance showed to be very poor, as lots of large spikes in the estimated position were observed. Moreover, the estimated position showed to be further from the ground truth than the position measured by the GPS.
During the second case, GPS measurement came in at 5 Hz. The variance in the x-axis and y-axis seemed to be lowest for $\eta_a = 0.01m/s^2$. Considering both the mean error and the variance, $\eta_a = 0.01m/s^2$ leads to the lowest error in the z-axis. Besides, during this case the plots do show that the estimation lies closer to the ground truth than the GPS measurements.
It will be concluded that an update rate of 1 Hz for the GPS is too low to correct for integration drift in this situation. Figure 7.9 shows that most of the spikes appear in between GPS measurements (be mindful of the delay while observing). Moreover, figure 7.6 shows that higher η_a values, thus less trust in the IMU, leads to less errors. This indicates that the spikes are caused by integration drift. Besides, the increasing only the GPS update rate significantly improved the performance, making the estimation closer to the ground truth than the GPS measurements. This shows a limitation of the GPS/IMU sensor suite, as the GPS update rate needs to be high enough to correct for the integration drift.

Next, it is was tested if the combination of $\eta_a = 0.01m/s^2$ and $n_{gps,p} = 0.01m$ indeed leads to the best estimation. This is done by fixing η_a to $0.01m/s^2$, and changing $n_{gps,p}$ in case 3. The results are shown in figure 7.8c. It was observed that the lowest variance in the x-axis and y-axis is found for $\eta_a = 0.01m/s^2$. In the z-axis the variance is slightly better for $\eta_a = 0.05m/s^2$. However, $\eta_a = 0.01m/s^2$ shows lower mean error and more importantly, the error in z is smaller than the error in the x and y direction.
It is concluded that the best values are found for $\eta_a = 0.01m/s^2$ and $n_{gps,p} = 0.01m$. The accuracy indicated by variance of the error is $4.9, 5.0 and 3.3mm$ for x, y and z, for a situation where the accelerometer accuracy is probably $\sigma_a^2 = 0.01m/s^2$ and the GPS position accuracy is $0.01m$.

A peculiarity left is that there still seems to be a high mean error in the y-axis. The mean error decreases by less trust in η_a. This error is probably not caused by the GPS, since figure 7.8b shows that less trust in the GPS leads to more errors, and figure 7.7c shows that more trust in the IMU leads to more errors.

Figure 7.9 Zoomed in plot of x position for $\eta_a = 0.01$, where GPS signals come in at 1 hz.



7.3 Experiment 2: Geometrical pose wrench estimator on manipulated dataset

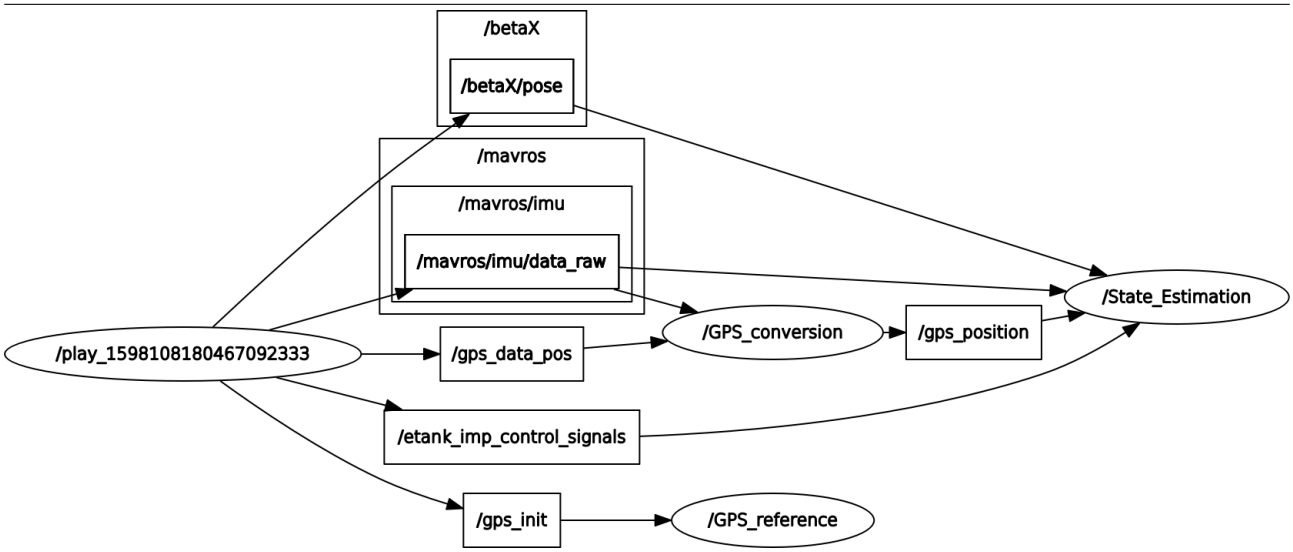
This experiment estimates the position and external force of the dataset using the geometrical pose wrench estimation. It is aimed to find limitations of the force estimation, tightly-coupling and observe differences with the geometrical pose estimator. The exact values encountered in the bar graph are presented in appendix F.4.

Figure 7.10 shows the ROS nodes for experimation with the geometrical pose wrench estimators. The setup up is very similar to figure 7.5, however the geometrical pose wrench estimator also subscribes on \`etank_imp_control_signals`. As explained before, this message comes from the observer described in [Rashad et al., 2019a], and it contains the control force $\bar{f}_{pr}$ to be used for the dynamics-based prediction.

The interaction force obtained from the force/torque sensor will be used as ground truth. As seen in figure 7.3c, this measurement is polluted by a considerably noise. The measurements are still meaningful for calculating the mean error, but since the measurement noise is close the noise in the estimation (as will become evident in the proceeding of this section), the signal is less useful for calculating the variance in the error.

Lastly, the geometrical pose wrench estimator estimator is set using the equations (5.31) and (5.36). That means that if one changes $\eta_{f_{pr}}$, the tuning parameter must be altered as using: $g_{\dot{v}} = \tilde{\eta}^b_\omega - m^{-1}\eta_{f_{pr}}$ and $y_a(\eta) = m^{-1}\eta^b_a + \eta_{f_{pr}}$

Figure 7.10 ROS nodes (ellipsoids) and messages (squares) used for experimentation with the geometrical pose wrench estimator



7.3.1 Description of the result

As the pose estimation for the geometrical pose wrench estimation is dependent on the external force, first the external force will be estimated in case 1. Next, in case 2 the pose will be estimated.

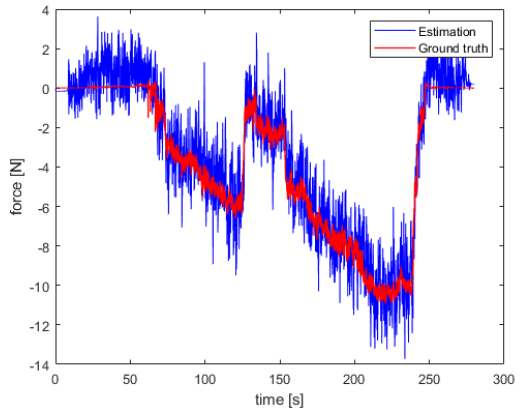
Case 1: Estimating external force

Figure 7.11 shows the results of the estimated external force, for different values of $\eta_{f_{ext}}$. The parameter $\eta_{f_{ext}}$ is used to denote how quickly the external force, modelled as a random walk, is varying.

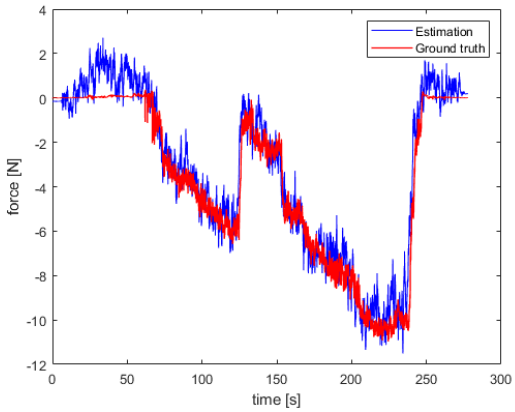
Figure 7.11a,7.11b and Figure 7.11c show the result for simulations with relative low values for $\eta_{f_{ext}}$. It can be seen that the noise is relatively large compared to the ground truth. The estimation in figure 7.11d shows comparable noise with the ground truth and figure 7.11e shows less noise than observed in the ground truth. Nevertheless, as can already be observed by the plot, the situation where $\eta_{f_{ext}} = 0.0001N/S$ shows that the estimation doesn't change quickly enough to follow the actual interaction force.
The bar graph in 7.11f compares the mean error for each situations. It can be seen that the lowest mean error is found for $\eta_{f_{ext}} = 0.0005N/S$, indicating that the estimated external force is changing quickly enough to follow the actual interaction, while also showing the least amount of noise.

Lastly, note that each plot estimates and positive external force up to approximately $t \approx 60s$. This estimation error can also be observed for the mean errors, as all situations show a considerable mean error in the bar graph. As concluded earlier, this positive external force is probably disturbance from for instance aerodynamics.

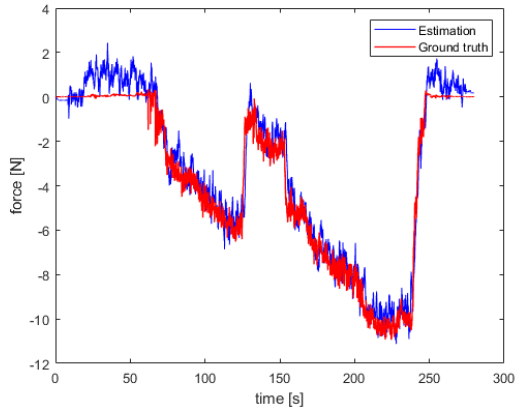
Figure 7.11 Force estimation compared for five different values of $\eta_{f_{ext}}$



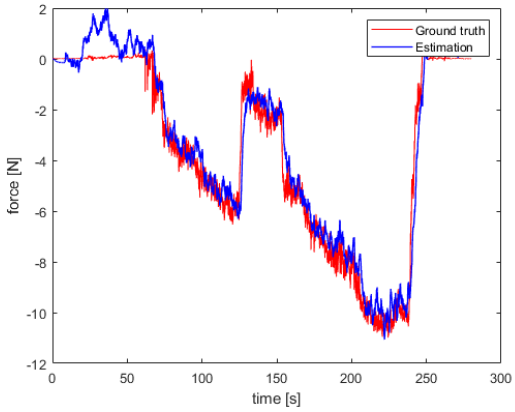
(a) Estimated external force for $\eta_{f_{ext}} = 0.1N/S$



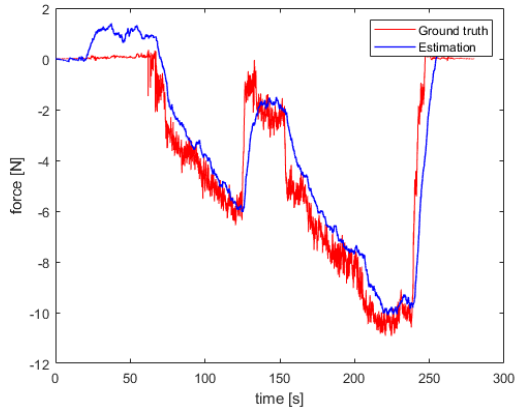
(b) Estimated external force for $\eta_{f_{ext}} = 0.01N/S$



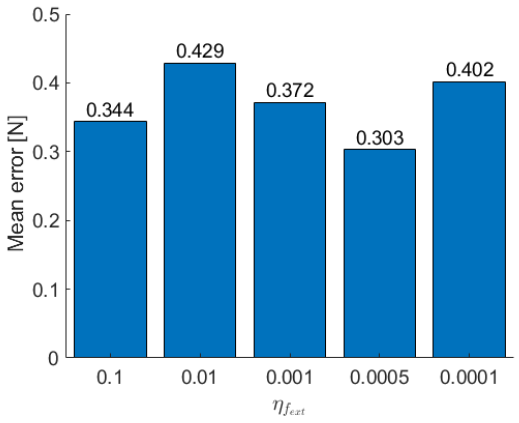
(c) Estimated external force for $\eta_{f_{ext}} = 0.001N/S$



(d) Estimated external force for $\eta_{f_{ext}} = 0.0005N/S$



(e) Estimated external force for $\eta_{f_{ext}} = 0.0001N/S$



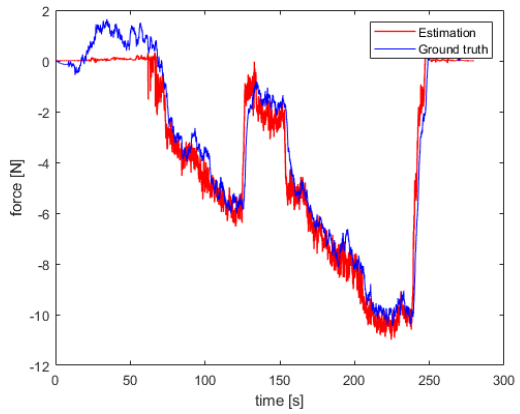
(f) Bar graph denoting mean errors

Case 2: tightly-coupled force and position estimation

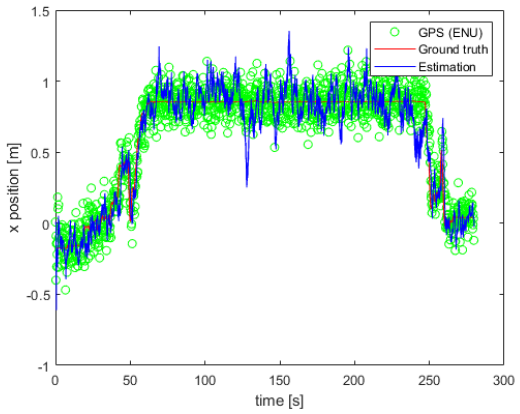
Figure 7.12 shows three scenarios for different values of $n_{f_{pr}}$. As described in 7.1, the value $\eta_{f_{pr}}$ is used to change tuning parameters for the $g_{\dot{v}}$ and y_a.
Figure 7.12a shows the estimated external force and figure 7.12b shows the estimated position for $\eta_{f_{pr}} = 0.01N/s$. It can be seen that the estimated force is more filtered than seen in 7.11d. The positional errors observed are relatively high. Especially, at approximately $t \approx 130s$ and

$t \approx 160s$, two large positional error peaks can be seen. Notice that the first peak happens simultaneously while the interaction force rises. When looking very closely, the force deviates from the ground truth around $t \approx 130s$, due to the delay in the estimation. Also, at $t \approx 160s$ the force is estimated relatively poorly, which immediately causes the positional errors.
The plot 7.12c shows the external force estimation for $\eta_{f_{pr}} = 0.001N/s$. Note that the position is presented later in 7.15. For this value, the external force is estimated with as much noise as the measured interaction force.
Figure 7.12d shows the estimation force and figure 7.12e shows the estimated position for $\eta_{f_{pr}} = 1N/s$. For this value, the external force is estimated similar to 7.12a, but relatively a higher amount of noise is observed in the estimated position.
The bar graph 7.12d compares the estimated position position of the three experiments. Note that the value $\eta_{f_{pr}} = 0.1N/s$ leads to be lowest variance. The mean error seems to increase for high values. What can be observed in the plots is that $\eta_{f_{pr}}$ indeed changes how much of the forces is taking into account for the estimation.

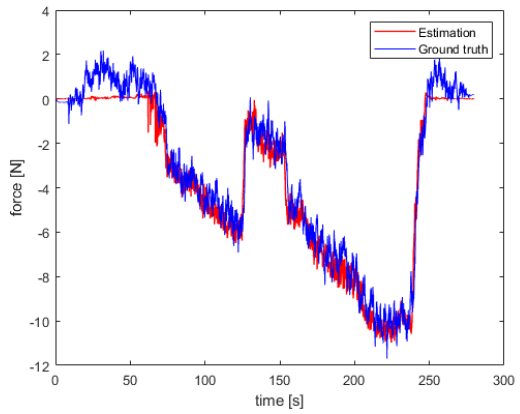
Figure 7.12 Force and position estimation plots, and bar graph comparison (plot of best position estimation will be shown next)



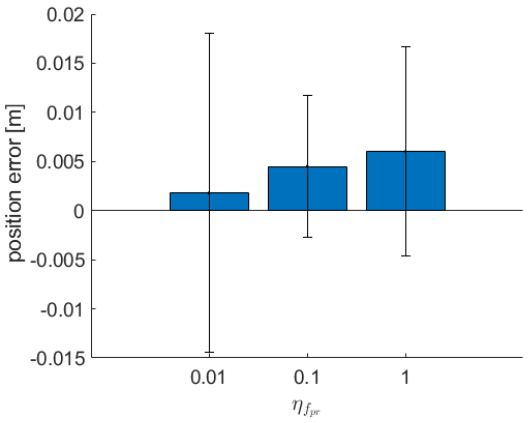
(a) Estimated external force for $\eta_{f_{pr}} = 0.01N/S$



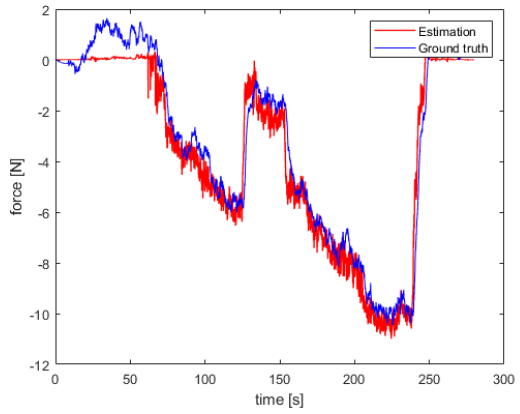
(b) Estimated x position for $\eta_{f_{ext}} = 0.01N/S$



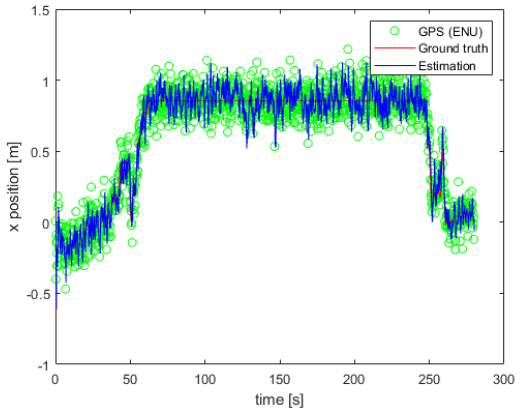
(c) Estimated external force for $\eta_{f_{pr}} = 0.1N/S$



(d) Error bar graph of estimated position/



(e) Estimated external force for $\eta_{f_{ext}} = 1N/S$

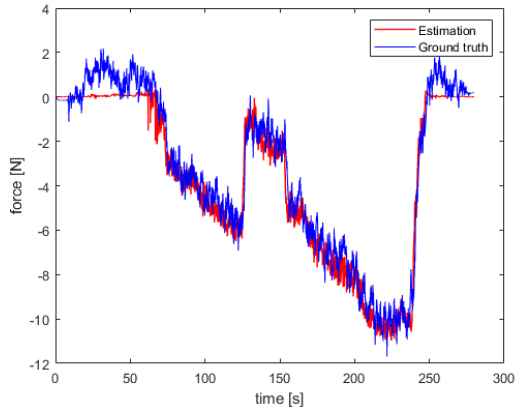


(f) Estimated x position for $\eta_{f_{ext}} = 1N/S$

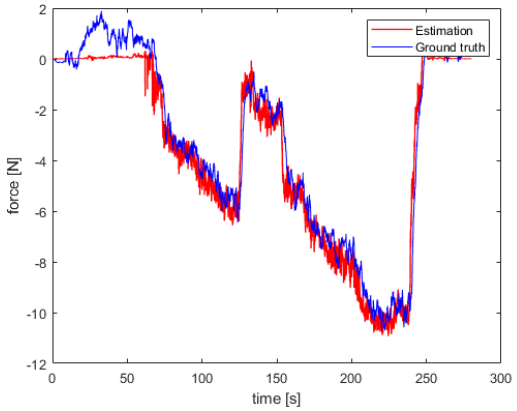
At last, it is checked whether the ratio between the prediction and correction is sufficient. In order to check that, two additional simulations will be performed. Simulation 1, observed in figure 7.12c, will be the same. In simulation 2, $n_{f_{pr}}$ will be set too $n_{f_{pr}} = 0.01$ for $g_{\dot{v}}$, but it will be kept equal for y_a. In simulation 3, $n_{f_{pr}}$ will be set too $n_{f_{pr}} = 0.01$ for y_a, but it will be kept equal for $g_{\dot{v}}$.

Figure 7.14 shows the result of the ratio comparison. Figure 7.13a shows the already en-

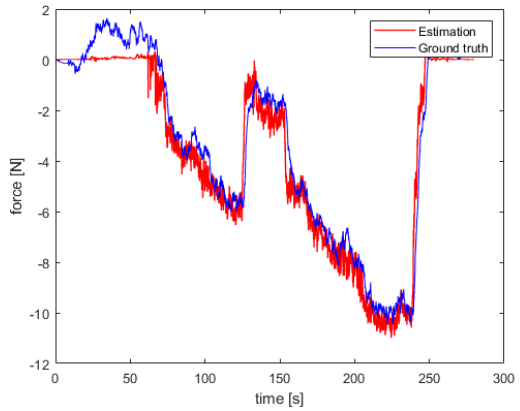
countered situation. For the external force in both figure 7.13b and figure7.13c, less noise is encountered (with also the error caused by the overfiltering). The error bar graph in 7.13d shows that situation 1 (the already encountered situation), leads to the best estimation performance.



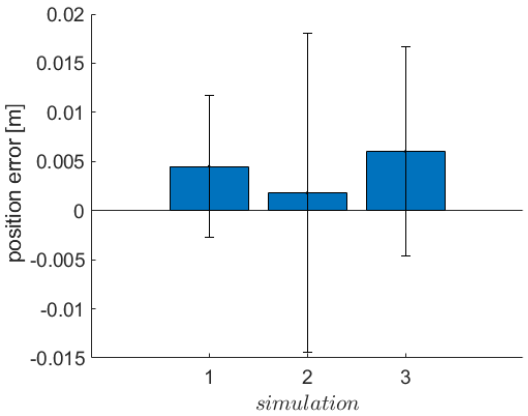
(a) Simulation 1: Estimated external force for $\eta_{f_{ext}} = 0.1N$



(b) Simulation 2: Estimated external force, where $n_{f_{pr}} = 0.01N/s$ for prediction and $n_{f_{pr}} = 0.01N/s$ for correction



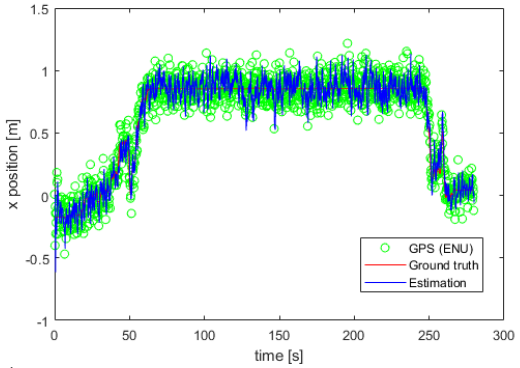
(c) Simulation 3: Estimated external force, where $n_{f_{pr}} = 0.1N$ for prediction and $n_{f_{pr}} = 1N/s$ for correction



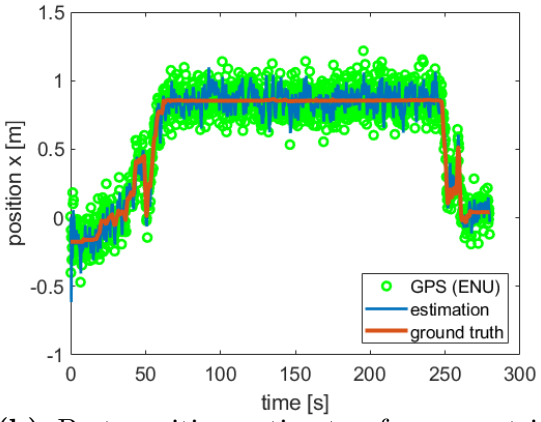
(d) Error bar graph comparing the position error for among the situations.

The best position estimates for both the geometrical pose wrench estimator and the geometrical pose estimator are plotted in figure 7.15.
Figure 7.14a shows that the best position estimation for the tightly-coupled estimator is polluted with more noise than the geometrical pose estimation in 7.14b.

Figure 7.15 Best position estimator for geometrical pose estimator.



(a) Best x position estimation for geometrical pose wrench estimator. Mean error is $4.5mm$, variance in the error is $7.2mm$



(b) Best position estimator for geometrical pose estimator. Mean error is $3.1mm$, variance in the error is $4.9mm$

7.3.2 Discussion on the results

During this experiment the geometrical pose wrench estimator was used on manipulated real-world data. It was aimed to find limitations of the external force estimation, find limitations of the tightly-coupled pose estimation and to compare with the geometrical pose estimator.

First of all, it was showed that lowering $\eta_{f_{ext}}$ resulted into external force estimations less polluted by noise. Up to $\eta_{f_{ext}} = 0.0005N/s$, estimations without significant errors where found. Naturally, the value of $\eta_{f_{ext}}$ is dependent on the slope of desired interaction force. For instance, note that in 7.3c, the highest slope is observed between $t \approx 397s$ and $t \approx 408s$. The slope here is approximately $\Delta f \approx 0.05N/s$ (Note that the best estimation is found for $\eta_{f_{ext}} = 0.05N/s$. This is probably since the slope is much lower during most of the interaction). In cases where a different slope is desired, the value for $\eta_{f_{ext}}$ will likely be different for the best estimations. The estimations in figure 7.11 also showed a drawback of indirectly obtained the interaction force by estimating external forces. Up to $t \approx 60s$, the UAV didn't engage in interaction, but forces where observed. Using the external force as interaction force will inevitable lead to undesired force, or the need for workarounds, as disturbances will be fed into the force controller.

In figure 7.12b, both the external force estimation and pose estimation was considered. It was found that indeed best performance of the pose estimation was found for the identified variance of $n_{f_{pr}} = 0.1N$. This value leads to a mean error in the estimated x position of $4.5mm$ and a

variance of $7.2mm$, for a situation where the accelerometer noise was $\eta_a = 0.01m/s^2$ and the GPS noise $n_{gps,p} = 0.01m$. It was showed that geometrical pose wrench estimator can indeed increase the accuracy of the position measured by the GPS.
However, it was also shown that the geometrical pose estimator was able to estimate the position with a mean error of $3.1mm$ and an accuracy of $4.9mm$. This is a better improvement compared to the tightly-coupled geometrical pose wrench estimator.
It is expected that the lower mean error of the geometrical pose wrench estimator is likely caused by an additional delay. In order to make estimations, the geometrical pose wrench estimators estimates one additional state compared to the geometrical pose estimator. It is expected that the decrease is accuracy is caused by the propeller force. For the geometrical pose estimator the noise is injected into the velocity using the function $g_{\dot{v}} = \tilde{\eta}_\omega v_b^{b,i} - \eta_a$ and for the geometrical pose wrench estimator the function is $g_{\dot{v}} = \tilde{\eta}_\omega v_b^{b,i} - m^1 \eta_{f_{pr}}$. For this scenario, $-\eta_a = 0.01$ and $-m^{-1}\eta_{f_{pr}} = -1.7^{-1}0.1 = 0.0588$. This shows the more noise is inserted by the geometrical pose wrench estimator than by the geometrical pose estimator.

Chapter 8

Conclusion

8.1 Conclusions

In this thesis, a state estimator tightly coupling force and pose estimation has been proposed. The research questions are as follows:

1. How can force estimation be tightly-coupled with pose estimation for a geometric state estimator?
2. What are the limitations of external force estimation for the tightly-coupled state estimator for physical contact interaction using a UAV?
3. How does a state estimator tightly-coupling force estimates into the pose estimation compare against a state estimator exclusively estimating the pose for physical contact interaction using a UAV?

The formulated estimator uses components of a control wrench injected via the propellers as measurement input. The external force is estimated using a model of a random walk. After the external force has been predicted, it can be corrected by using measurements obtained from an accelerometer.
To accomplish a geometric state estimator, the plus operator $\oplus$ and minus operator $\ominus$ have been introduced. These operators allow considering increments and differences while not breaking the constraints imposed by the Lie group a state belongs to.
In total two state estimators have been formulated. The geometrical pose estimator is based on rigid body kinematics. It uses accelerometer and gyroscopic measurements to predict its states. Using a GPS, the prediction can be corrected to contain drift caused by integration. The geometrical pose wrench estimator is based on the geometrical pose estimator, but it adds rigid body dynamics to the prediction.
Both state estimators have been implemented in C++ using software from the Manifold Toolkit [Hertzberg et al., 2011]. The state estimators are validated using a realistic Gazebo simulation. During the experimentation, it was shown that when exploiting a GPS, the update rate should be high enough to correct the drift. For a situation where the update rate was 1 Hz, significant deviations where observed. This caused the position estimations to show a larger error than the raw GPS measurements did. For the same GPS accuracy with an update rate of 5 Hz, the estimations were better than the raw GPS measurements.

The limitations of the external force estimation are tested on a manipulated real-world data. The used dataset is from an experiment where a fully-actuated UAV interactions with a vertical surface in an indoor lab environment. This experiment was intended for the work [Rashad et al., 2019a]. The dataset contains position measurements from a MoCap, measurements from an IMU, an estimated propeller wrench, and the interaction force measured by a force/torque sensors. The data from the Mocap is used to simulate GPS data, by changing the

reference frame to ECEF and introducing additional noise with a variance $\sigma^2_{gps,p} = 0.01m$.
The geometrical pose wrench estimator can estimate external forces up to a similar accuracy as the ground truth from the force/torque sensor. However, because both the ground truth and estimation contain noise in the same range, it is difficult to present the exact accuracy.
Furthermore, it was noted that the estimation accuracy of the external force is dependent on the desired interaction force. If a more drastically changing interaction force is desired, the estimated external force should also change more quickly. Consequently, this will cause the force estimation to contain more noise.
Next to this, the state estimator estimated a positive external force before the UAV came into contact with the vertical surface. Naturally, this is not the interaction force but it is a disturbance. This shows a well-known downside of estimating the interaction force using external force estimation. The external force will be a product of the interaction force, parameter errors and unmodelled effects. Significant deviations from the true interaction force will decrease the performance of the force control, as disturbances and errors will be sent to the controller.

The best estimated position of the geometrical pose wrench estimator (the state estimator tightly-coupling pose and force estimation), showed a mean error of $4.5mm$ and a variance in the error of $7.2mm$. For the geometrical pose estimator, the mean error of the position is $3.1mm$ and the variance is $4.9mm$. This performance was achieved by exploitation of measurements with the following noise variance: Acceleration noise $0.01m/s^2$, position noise $0.01m$, and propeller/control force $0.1N$. Both estimators improved the position accuracy relative to the GPS measurements, as the estimations are more accurate than the raw position measurements from the GPS.
However, the geometrical pose wrench estimator showed to have less accuracy than the geometrical pose estimator, while exploiting more information. The expectation is that the degrade in accuracy is caused by the propeller wrench. It was shown that the measurement noise injected into the velocity by the geometrical pose estimator is $g_{\dot{v}} = \tilde{\eta}_\omega v_b^{b,i} - \eta_a$ and by the geometrical pose wrench estimator $g_{\dot{v}} = \tilde{\eta}_\omega v_b^{b,i} - m^1 \eta_{f_{pr}}$. The differences for this situation is $-\eta_a = 0.01$ and $-m^{-1}\eta_{f_{pr}} = -1.7^{-1}0.1 = -0.0588$. This shows that a larger amount of noise is injected into the velocity prediction for the geometrical pose wrench estimator than for the geometrical pose estimator. Based on this explanation, it is concluded that the pose estimation accuracy of the proposed state estimator tightly-coupling pose and force estimates is limited by the accuracy of the propeller/control force.

8.2 Recommendations on Future work and limitations

During this thesis, limitations regarding external force estimation and accuracy have been found for a state estimator tightly-coupling dynamics with pose estimation.
It was found that the position estimation performance was lower for the geometrical pose wrench estimator than for the pose estimator. This is due to the fact that the propeller force contains more noise than the measured acceleration. The most obvious approach to improve the position estimation for the geometrical pose wrench estimator is to improve propeller force accuracy. Nevertheless, for most situations, this might not be possible.
Another possibility is to reformulate the state estimator such that the acceleration measurements influence the position estimation more directly. In the current design, the measured acceleration is used to estimate an external force $\bar{a}_b^{b,i} = h(f_{ext}^b, ..)$. Subsequently, the external force is used in conjunction with the propeller force to predict the velocity $\dot{v}_b^{b.i} = f(\bar{f}_{pr}^b, f_{ext}^b) + ...$ And next, the velocity is used to predict the position $\dot{\xi}_b^i = f(v_b^{b.i}, ..)$. If one formulates the filter

such the acceleration is not out-weighted by the propeller force, then the position estimation will be estimated with additional information. This might improve the geometrical pose wrench estimator, such that it can estimate with more accuracy than the geometrical pose estimator.

A drawback of estimating the interaction force indirectly, by using the external force, is that the external can be polluted by disturbances and modeling errors, such as wind, friction, or parameter deviations. As shown in chapter 7, the external force from the experiment was polluted with a disturbance before the UAV engaged in interaction. Using such a signal as feedback to a force controller will lead to undesired actuation. Therefore, either the external force should be separated or the feedback should be enabled whenever a contact is detected.
The work [Tomić and Haddadin, 2015] proposes a collision detection based on the frequency characteristics of the external force. If one were to use an indirect interaction force estimation method, such a contact detection procedure would improve the robustness of the control scheme, as disturbances won't affect the control before contact. Whenever the UAV is in contact with an object or surface, aerodynamic effects up to the magnitude of friction will be diminished. Thus, when the contact is detected, the UAV can freely use the interaction scheme.
Another option is to equip the UAV with a force/torque sensor and use interaction force measurements in conjunction with an external force estimator. Of course, this comes at the expense of size, weight and equipment costs, but it can help to improve the performance of the interaction control while being able to attenuate disturbances. For example, such an approach would be helpful for interactions in outdoor environments with lots of disturbances from for instance wind.

A limitation of the proposed state estimator is that it is prone to parameter uncertainties. Naturally, the mass should be known, but to obtain the propeller force, the parameter of the propellers should be known as well. However, there can always be deviation in these parameters. For instance, the amount of trust per propeller might be different due to deformations.
Another limitation of this research is that the state estimators have not been tested for complete real-world scenarios. The dataset did contain real-world data, but the GPS signal was obtained by altering the Mocap signal. Therefore, effects such as GPS biases have not been considered during the experimentation.

Appendices

Appendix A

Derivation of Linear Kalman Filter

In this appendix the linear Kalman Filter will be derived mathematically. The derivation is based on the calculations encountered in [N.A.Thacker, 1998].
To get started the observation model (2.2) will be considered in the estimator (2.4) by substitution:

$$\begin{aligned}\hat{x}_k &= \breve{x}_k + K_k(Hx_k + n_k - H\breve{x}_k) \\ &= \breve{x}_k + K_kHx_k - K_kH\breve{x}_k + Kn_k \\ &= \breve{x}_k + (K_kH)(x_k - \breve{x}_k) + Kn_k \\ &= \breve{x}_k + (K_kH)(x_k - \breve{x}_k) + Kn_k\end{aligned} \tag{A.1}$$

This equation can be substituted in the definition of the process error:

$$\begin{aligned}e_k &= x_k - \hat{x}_k \\ &= x_k - \breve{x}_k - (K_kH)(x_k - \breve{x}_k) - K_kn_k \\ &= (I - K_kH)(x_k - \breve{x}_k) - K_kn_k\end{aligned} \tag{A.2}$$

The process error can be substituted in (2.5) to obtain the process covariance matrix:

$$\begin{aligned}P_k = E[e_ke_k^T] = E[((I - K_kH)(x_k - \breve{x}_k) - K_kn_k) \\ ((I - K_kH)(x_k - \breve{x}_k) - K_kn_k)^T]\end{aligned} \tag{A.3}$$

The term $(x_k - \breve{x}_k)$ can be considered as a process error obtained during the prediction step. Also n_k is the error from the observation sensors. Clearly these two errors are uncorrelated. Therefore we can factorize the two (thus not considering cross-correlation between the two):

$$\begin{aligned}P &= (I - K_kH)E[(x_k - \breve{x}_k)](x_k - \breve{x}_k))^T)(I - K_kH)^T + K_kE[n_kn_k^T]K_k^T \\ &= (I - K_kH)\breve{P}(I - K_kH)^T + K_kRK_k^T\end{aligned} \tag{A.4}$$

In the last step $\breve{P}$ is introduced as covariance matrix of the process error for the predicted states $\breve{P} = E[(x_k - \breve{x}_k)]$. Furthermore the $E[n_kn_k^T]$ is recognised from (2.3). Now the term can be expanded:

$$\begin{aligned}P &= \breve{P} - KH\breve{P} - \breve{P}H^TK^T + KH\breve{P}K^TH^T + KRK^T \\ &= \breve{P} - KH\breve{P} - \breve{P}H^TK^T + K(H\breve{P}H^T + R)K^T\end{aligned} \tag{A.5}$$

This term can be used to update the process covariance matrix after the Kalman gain has been calculated. Nevertheless this version is lengthy and will be simplified later.
Using process covariance update equation, the Kalman gain 'K' can be obtained. The Kalman gain should be an optimal gain for the filter. This can be achieved by minimizing the process error vector e. In order to do this, first the proces covariance matrix update function will be expanded:

$$\begin{aligned} P &= \breve{P} - KH\breve{P} - \breve{P}H^TK^T + KHPH^TK^T + KRK^T \\ &= \breve{P} - KH\breve{P} - \breve{P}H^TK^T + K(H\breve{P}H^T + R)K^T \end{aligned} \tag{A.6}$$

As seen in (2.5) The process covariance matrix describes Expectation between the error samples e_{k-1}, e_k and e_{k+1}. For the next steps the Kalman filter is most concerned about the auto-correlation, the variance, of e_k. Therefore, the trace $Tr(.)$ can be taken:

$$\begin{aligned} Tr(P) &= Tr[\breve{P}] - Tr[K_kH\breve{P}] - Tr[\breve{P}H^TK_k^T] + Tr[K_k(H\breve{P}H^T + R)K_k^T] \\ &= Tr[\breve{P}] - 2Tr[K_kH\breve{P}] + Tr[K_k(H\breve{P}H^T + R)K_k^T] \end{aligned} \tag{A.7}$$

The last simplification can be made since the trace is equal to the trace of its transpose.
The Kalman gain is meant to be an optimal gain which minimizes the (process) error of the filter. The previous equation is a straightforward equality for a term which posses as process error. Therofore, one can obtain the Kalman gain by minimizing the the previous term with respect to the Kalman Gain:

$$\begin{aligned} \frac{dTr(P)}{dK_k} &= \frac{d}{dK_k}Tr[\breve{P}] - 2\frac{d}{dK_k}Tr[K_kH\breve{P}] + \frac{d}{dK_k}Tr[K_k(H\breve{P}H^T + R)K_k^T] \\ &= 0 - 2(H\breve{P})^T + 2K_k(H\breve{P}H^T + R) = 0 \end{aligned} \tag{A.8}$$

And now to solve for K_k:

$$\begin{aligned} 0 &= -2(H\breve{P})^T + 2K_k(H\breve{P}H^T + R) \\ 2K_k(H\breve{P}H^T + R) &= 2(H\breve{P})^T \\ K_k &= \breve{P}H^T(H\breve{P}H^T + R)^{-1} \end{aligned} \tag{A.9}$$

Now the Kalman gain (A.9) can be substituted in (A.6). This leads into cancelling the term to:

$$\begin{aligned} P &= \breve{P} - \breve{P}H^T(H\breve{P}H^T + R)^{-1}H\breve{P} \\ &= \breve{P} - K_kH\breve{P} \\ &= (I - K_kH)\breve{P} \end{aligned} \tag{A.10}$$

This leads to the process covariance matrix update function, which is able to correct $\breve{P}$.
The last equation to derive is the process covariance prediction $\breve{P}$. The term for the error during the prediction step has been defined before $\breve{e}_k = x_k - \breve{x}_k$. The next error can be calculated as follows:

$$\begin{aligned} \breve{e}_{k+1} &= x_{k+1} - \breve{x}_{k+1} \\ &= (Ax_k + Bu_k + \eta_k) - (A\hat{x}_k + Bu_k) \\ &= A(x_k - \hat{x}) + \eta_k \end{aligned} \tag{A.11}$$

And now as covariance matrix:

$$\breve{P}_{k+1} = E[\breve{e}_{k+1}\breve{e}_{k+1}^T] = E[(A(x_k - \hat{x}) + \eta_k)(A(x_k - \hat{x}) + \eta_k)^T] \tag{A.12}$$

The process noise e_k is not correlated with η_k which is the noise from the sensors used as measurement input u. Therefore the term can be factorized:

$$\begin{aligned} \breve{P}_{k+1} &= E[(A(x_k - \hat{x}) + \eta_k)(A(x_k - \hat{x}) + \eta_k)^T] \\ &= E[(A(x_k - \hat{x}))(A(x_k - \hat{x}))^T] + E[\eta_k \eta_k^T] \\ &= AP_kA^T + Q \end{aligned} \tag{A.13}$$

Appendix B

Derivation exponential map identities

The paper [Joan Solá, 2019] shows how to derive the well known "Rodrigues Formula" using a Taylor series. The Rodrigues Formula looks as follows:

$$R = exp(\tilde{\omega}\theta) = I + \tilde{\omega}sin(\theta) + \tilde{\omega}^2(1 - cos(\theta)) \tag{B.1}$$

The series can also be used to derive edge cases where, for instance, a variable on the lie algebra is small in magnitude:

$$\begin{aligned} exp(\tilde{\varepsilon}) &= I + \tilde{\varepsilon} + \frac{1}{2}\tilde{\varepsilon}^2 + \frac{1}{3!}\tilde{\varepsilon}^3... \\ &\overset{\varepsilon \to 0}{=} I + \varepsilon \end{aligned} \tag{B.2}$$

This edge case can also be derived for the same case but negative:

$$\begin{aligned} exp(-\tilde{\varepsilon}) &= I + (\tilde{-\varepsilon}) + \frac{1}{2}(\tilde{-\varepsilon})^2 + \frac{1}{3!}(\tilde{-\varepsilon})^3... \\ &= I - \tilde{\varepsilon} - \frac{1}{2}\tilde{\varepsilon}^2 - \frac{1}{3!}\tilde{\varepsilon}^3... \\ &\overset{\varepsilon \to 0}{=} I - \tilde{\varepsilon} \end{aligned} \tag{B.3}$$

A last Identity that will be derived is for the case $exp(\tilde{\tau}\tilde{\omega}\tilde{\tau}^{-1})$ where $\tilde{\tau} \in so(3)$. Note that this derivation is also true for $R \in SO(3)$:

$$\begin{aligned} exp(\tilde{\tau}\tilde{\omega}\tilde{\tau}^{-1}) &= \tilde{\tau}\tilde{\tau}^{-1} + \tilde{\tau}\tilde{\omega}\tilde{\tau}^{-1} + \frac{1}{2}(\tilde{\tau}\tilde{\omega}\tilde{\tau}^{-1})^2 + \frac{1}{3!}(\tilde{\tau}\tilde{\omega}\tilde{\tau}^{-1})^3... \\ &= \tilde{\tau}\tilde{\tau}^{-1} + \tilde{\tau}\tilde{\omega}\tilde{\tau}^{-1} + \tilde{\tau}\frac{1}{2}(\tilde{\omega})^2\tilde{\tau}^{-1} + \tilde{\tau}\frac{1}{3!}(\tilde{\omega})^3\tilde{\tau}^{-1}... \\ &= \tilde{\tau}(I + \tilde{\omega} + \frac{1}{2}\tilde{\omega}^2 + \frac{1}{3!}\tilde{\omega}^3 + ...)\tilde{\tau}^{-1} \\ &= \tilde{\tau}exp(\tilde{\omega})\tilde{\tau}^{-1} \end{aligned} \tag{B.4}$$

These identities will prove to be useful in derivations that make of an exponential map of a small value.

Appendix C

Derivation of Lie group partial derivatives

$$\begin{aligned}\frac{\partial Rx}{\partial x} &= \lim_{\varepsilon \to 0} \frac{R(x+\varepsilon) - Rx}{\varepsilon} \\ &= \lim_{\varepsilon \to 0} \frac{Rx + R\varepsilon - Rx}{\varepsilon} \\ &= \lim_{\varepsilon \to 0} \frac{R\varepsilon}{\varepsilon} = R \\ \frac{\partial Rx}{\partial x} &= R\end{aligned} \tag{C.1}$$

$$\begin{aligned}\frac{\partial Rx}{\partial R} &= \lim_{\varepsilon \to 0} \frac{(R \oplus \varepsilon)x - Rx}{\varepsilon} \\ &= \lim_{\varepsilon \to 0} \frac{exp(\tilde{\varepsilon})Rx - Rx}{\varepsilon} \\ &= \lim_{\varepsilon \to 0} \frac{(I + \tilde{\varepsilon})Rx - Rx}{\varepsilon} \\ &= \lim_{\varepsilon \to 0} \frac{\tilde{\varepsilon}(Rx)}{\varepsilon} \\ &= \lim_{\varepsilon \to 0} \frac{-(\tilde{Rx})\varepsilon}{\varepsilon} = -(\tilde{Rx}) \\ \frac{\partial Rx}{\partial R} &= -(\tilde{Rx})\end{aligned} \tag{C.2}$$

$$\begin{aligned}\frac{\partial R^{-1}x}{\partial R} &= \lim_{\varepsilon \to 0} \frac{(R \oplus \varepsilon)^{-1}x - R^{-1}x}{\varepsilon} \\ &= \lim_{\varepsilon \to 0} \frac{(exp(\tilde{\varepsilon})R)^{-1}x - R^{-1}x}{\varepsilon} \\ &= \lim_{\varepsilon \to 0} \frac{(R^{-1}exp(-\tilde{\varepsilon}))x - R^{-1}x}{\varepsilon} \\ &= \lim_{\varepsilon \to 0} \frac{(R^{-1}(I - \tilde{\varepsilon})x - R^{-1}x}{\varepsilon} \\ &= \lim_{\varepsilon \to 0} \frac{-R^{-1}\tilde{\varepsilon}x}{\varepsilon} \\ &= \lim_{\varepsilon \to 0} \frac{R^{-1}\tilde{x}\varepsilon}{\varepsilon} = R^T\tilde{x} \\ \frac{\partial R^{-1}x}{\partial R} &= R^T\tilde{x}\end{aligned} \tag{C.3}$$

$$
\begin{aligned}
\frac{\partial(R\tilde{\omega})}{\partial R} &= \lim_{\varepsilon \to 0} \frac{(R \oplus \varepsilon)\tilde{\omega} \ominus R\tilde{\omega}}{\varepsilon} \\
&= \lim_{\varepsilon \to 0} \frac{exp(\tilde{\varepsilon})R\tilde{\omega} \ominus R\tilde{\omega}}{\varepsilon} \\
&= \lim_{\varepsilon \to 0} \frac{log(exp(\tilde{\varepsilon})R\tilde{\omega}(R\tilde{\omega})^{-1})^V}{\varepsilon} \\
&= \lim_{\varepsilon \to 0} \frac{log(exp(\tilde{\varepsilon}))^V}{\varepsilon} = I_3 \\
\frac{\partial(R\tilde{\omega})}{\partial R} &= I_3
\end{aligned}
\tag{C.4}
$$

$$
\begin{aligned}
\frac{\partial\tilde{\omega}R}{\partial R} &= \lim_{\varepsilon \to 0} \frac{\tilde{\omega}(R \oplus \epsilon) \ominus \tilde{\omega}R}{\varepsilon} \\
&= \lim_{\varepsilon \to 0} \frac{\tilde{\omega}exp(\tilde{\varepsilon})R \ominus \tilde{\omega}R}{\varepsilon} \\
&= \lim_{\varepsilon \to 0} \frac{log(\tilde{\omega}exp(\tilde{\varepsilon})RR^{-1}\tilde{\omega}^{-1})^V}{\varepsilon} \\
&= \lim_{\varepsilon \to 0} \frac{log(\tilde{\omega}exp(\tilde{\varepsilon})\tilde{\omega}^{-1})^V}{\varepsilon} \\
&= \lim_{\varepsilon \to 0} \frac{log(exp(\tilde{\omega}\varepsilon))^V}{\varepsilon} = \tilde{\omega} \\
\frac{\partial\tilde{\omega}R}{\partial R} &= \tilde{\omega}
\end{aligned}
\tag{C.5}
$$

Appendix D

Software implementation of state estimators

This appendix shows how the Manifold Toolkit (MTK) [Hertzberg et al., 2011] has been used to implements the state estimators. Some important code of the geometrical pose wrench estimator will be shown. The geometrical pose wrench estimator is showed because it is more elaborate than the geometrical pose estimator. But the implementation is very similar to each other. In order to keep the code clear, irrelevant programming details have been replaced by dots.

D.1 Define a state

Before defining a state, the MTK should know which state belongs to which group. This allows the state estimator to use correct operators for each state. In order to accomplish this, the MTK uses wrappers. To simplify the code, these wrappers are implemented using type definitions:

```
typedef MTK::vect<3, double> vec3;
typedef MTK::SO3<double> SO3;
```

The wrapper defines the operators which correspond to the group a state belongs to. For instance, `vec3` denotes $\mathcal{R}^n$. The wrapper defines plus and minus as the regular '+' and '-'. `SO3` denotes a state $\in SO(3)$. This means that the wrapper defines plus and minus as $\oplus$ and $\ominus$, as seen in chapter 4.

Now that the types have been declared, the state vector can be declared:

```
MTK_BUILD_MANIFOLD ( State ,
(( vec3 ,pos ))
(( vec3, vel ))
(( SO3, orient ))
(( vec3, b_a ))
(( vec3, b_w ))
(( vec3, f_ext ))
);
```

This defines the state vector for the geometrical pose wrench estimator 5.27. Note that `MTK_BUILD_MANIFOLD` defines the type `State` as an object with all individual states.
The state vector for the geometrical pose estimator looks similar but without the state f_{ext}.

D.2 Prediction functions

The prediction model can be programmed as a function. The function will take the latest estimated state $\hat{x}_{k-1}$ with measurement input u and returns the predicted state $\breve{x}_k$.

The geometrical pose wrench estimator is formulated in equation 5.31. The software implementation is as follows:

```
State DynamicsEstimator::process_model(
    const State& s,  const vec3& F_prop, const vec3& w) {
    ...
    State s2; // Predicted state

    // Apply Rotation
    vec3 scaled_axis = (w - s.b_w) * dt;
    SO3 rot = SO3::exp(scaled_axis);


    // State functions
    s2.pos = s.pos + s.orient*s.vel * dt;
    s2.vel = s.vel + dt * (
        s.vel.cross(w) + m_inv*(F_prop + s.f_ext)
        - s.orient.inverse()*g);
    s2.orient = s.orient * rot;
    s2.b_a = s.b_a;
    s2.b_w = s.b_a;
    s2.f_ext = s.f_ext;

    ...

    return s2;
    }
```

The return type of the function `process_model` is of type `State`. The measurement input taken by the function are f_{pr} coded as `F_prop` and $\omega_b^{b,i}$ coded as `w`.
Before the prediction model is implemented the rotation must be applied separately. The function $SO3 :: exp$ effectively accomplishes $\tilde{\omega}_b^{b,i}$. After that the state can be predicted in accordance with the equations from 5.31.
Note that the integration has been implemented using Euler integration. Thus the prediction models are programmed as $\breve{x}_k = \hat{x}_{k-1} + \Delta t(f(x,u))$. The time step Δt is programmed as `dt` and should be known beforehand. In the case of the geometrical pose wrench estimator the time step is equal to the time step at which f_{pr}^b comes in. For the geometrical pose estimator the time step is equal to the time step of the IMU measurements.

Now the process covariance matrix $\breve{P}_k$ will be programmed. The only knowledge needed for an UKF implementation of $\breve{P}_k$ is the measurement covariance matrix Q. As seen in section 2.1.2.

The measurement noise vector for the geometrical pose wrench estimator is as follows: $\eta = \begin{bmatrix}\eta_{f_{pr}}^b & \eta_{\omega}^b & \eta_{b_a}^b & \eta_{b_\omega}^b & \eta_{f_{ext}}^b\end{bmatrix}$.
Matrix $\breve{P}_k$ will also be programmed as a function. The return type is of `ukfom::ukf<UkfState>::cov`. This type is a template which defines a square matrix P with the dimension equal to the size of the state vector. the function for $\breve{P}_k$ takes no arguments:

```
ukfom::ukf<UkfState>::cov DynamicsEstimator::process_noise_cov() {
```

```
        ...
        // Declare covariance matrix
        ukfom::ukf<UkfState>::cov cov = ukfom::ukf<UkfState>::cov::Zero();

        // Set covariance matrix element for each state
        setDiagonal(cov, &State::pos, 0);

        setDiagonal(cov, &State::vel,
                        n_f_pr * dt);

        setDiagonal(cov, &State::orient,
                        n_w * dt);

        setDiagonal(cov, &State::b_a,
                        n_ba * dt);

        setDiagonal(cov, &State::b_w,
                        n_bw * dt);

        setDiagonal(cov, &State::f_ext,
                n_f_ext * dt);

        ...

        return cov;

    }
```

In the code, first an empty covariance matrix is declared `cov` with the same return type of the function. Next `cov` is set to zero. After the declaration of `cov` the values of Q will be set for each state. This is done using the function `setDiagonal`.
The value of each element of Q are programmed without considering $g(\eta)$ from equation 5.31. This is done so the user can set the values in accordance with $g(\eta)$. This avoids the need to reprogram in case of a modelling error.

D.3 Correction functions

The correction will be demonstrated only for $a_b^{b,i}$ in order to keep it short. The implementation it the same for other states but with different equations of course.

Similar to the prediction, the correction $h(x, u)$ is also programmed as a function. The function takes the x and u and returns y in the corresponding type, which is `vec3` for $a_b^{b,i}$:

```
    vec3 DynamicsEstimator::acceleration_observation_model(
            const State& s, const vec3& F_prop){

            return m_inv*(s.f_ext + F_prop ) + s.b_a;
    }
```

Furthermore an implementation for $h(x, u)$ needs knowledge about R. this is needed to deal with the correction noise n:

```
    Eigen::Matrix<double, 3, 3>
        DynamicsEstimator::acceleration_measurement_noise_cov() {

        return n_a * Eigen::Matrix<double, 3, 3>::Identity();
    }
```

D.4 Execute Prediction and Correction

Now that all functions for the prediction and correction are defined, it will be shown how to call them. The functions for the prediction and correction are called in a sensor callback. The prediction function for the geometrical pose wrench estimator is called in the callback of the message received for f_{pr}^{b}. This callback is called `predictionCallback`, and it is programmed as follows:

```
void DynamicsEstimator::predictionCallback(const spc_uav_comm::ControlSignals& msg) {
// Store propeller wrench
    F_prop <<
                    msg.control_wrench.force.x,
                    msg.control_wrench.force.y,
                    msg.control_wrench.force.z;
...
// Call prediction model
 kf_.predict(
    boost::bind(&DynamicsEstimator::process_model, this, _1, F_prop, gyro),
    process_noise_cov());
...

 }
```

First the message for f_{pr}^{b} will be read and the content will be stored within `F_prop`. This variable is passed to the function `predict`. The function `predict` is a member function of a kalman filter object. The instantiation of this object is `kf_`. The arguments passed to the predict function are the function `bind` and the `process_noise_cov()`. `bind` handles the arguments needed for `process_model()`.

The correction is also called in the callback of the corresponding observation. For $a_{b}^{b,i}$ the callback sensor is the IMU.

```
void DynamicsEstimator::imuCallback(const sensor_msgs::Imu& msg){

acc << msg.linear_acceleration.x,
    msg.linear_acceleration.y,
    msg.linear_acceleration.z;
...

    kf_.update(
        acc,
        boost::bind(
                &DynamicsEstimator::acceleration_measurement_model,
                this,
                _1,
                F_prop
        ),
        acceleration_measurement_noise_cov());
    ...
}
```

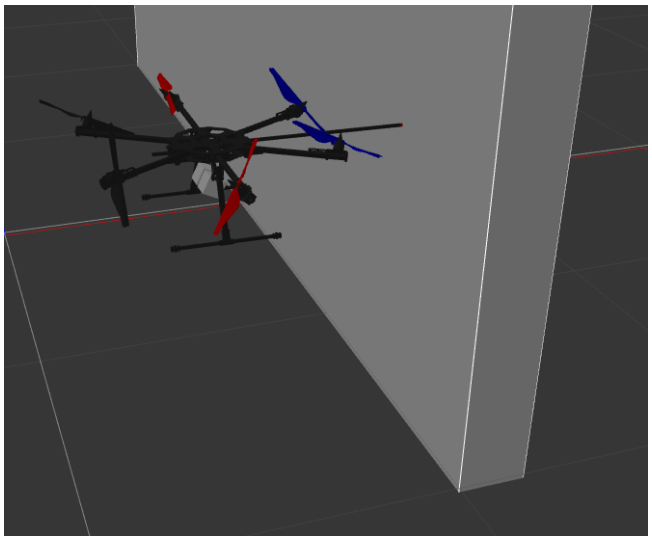
Whenver a new message comes in, the contect is read and stored to `acc`. The update is called similar to the prediction, but now for `acceleration_measurement_model()` and `acceleration_measurement_noise_cov()`

Appendix E

Simulation Seup

The simulation environment used is Gazebo with the RotorS simulator[Furrer et al., 2016]. In Gazebo, the fully-actuated hexarotor called BetaX is simulated as seen in figure E.1. To interface all needed software, the middleware Robot Operating System (ROS) has been used. ROS interfaces the simulation with the: state estimation software, simulated sensors, and the controller. The state estimators have been implemented in C++ using the Manifold Toolkit (MTK) [Hertzberg et al., 2011]. The software implementation of the state estimators is being described in D. Both the geometrical pose estimator and the geometrical pose wrench estimator have their node, which can be launched for each simulation. Moreover, both ROS nodes contain two source files. One source file implements the algorithmic part of the software, while the other file interfaces the state estimator to ROS.
Lastly, the parameters used for the simulation are denoted in appendix E

Figure E.1 BetaX next to a wall in the Gazebo simulation environment



In the remainder of this section the controller, sensors and the used ROS nodes will be described.

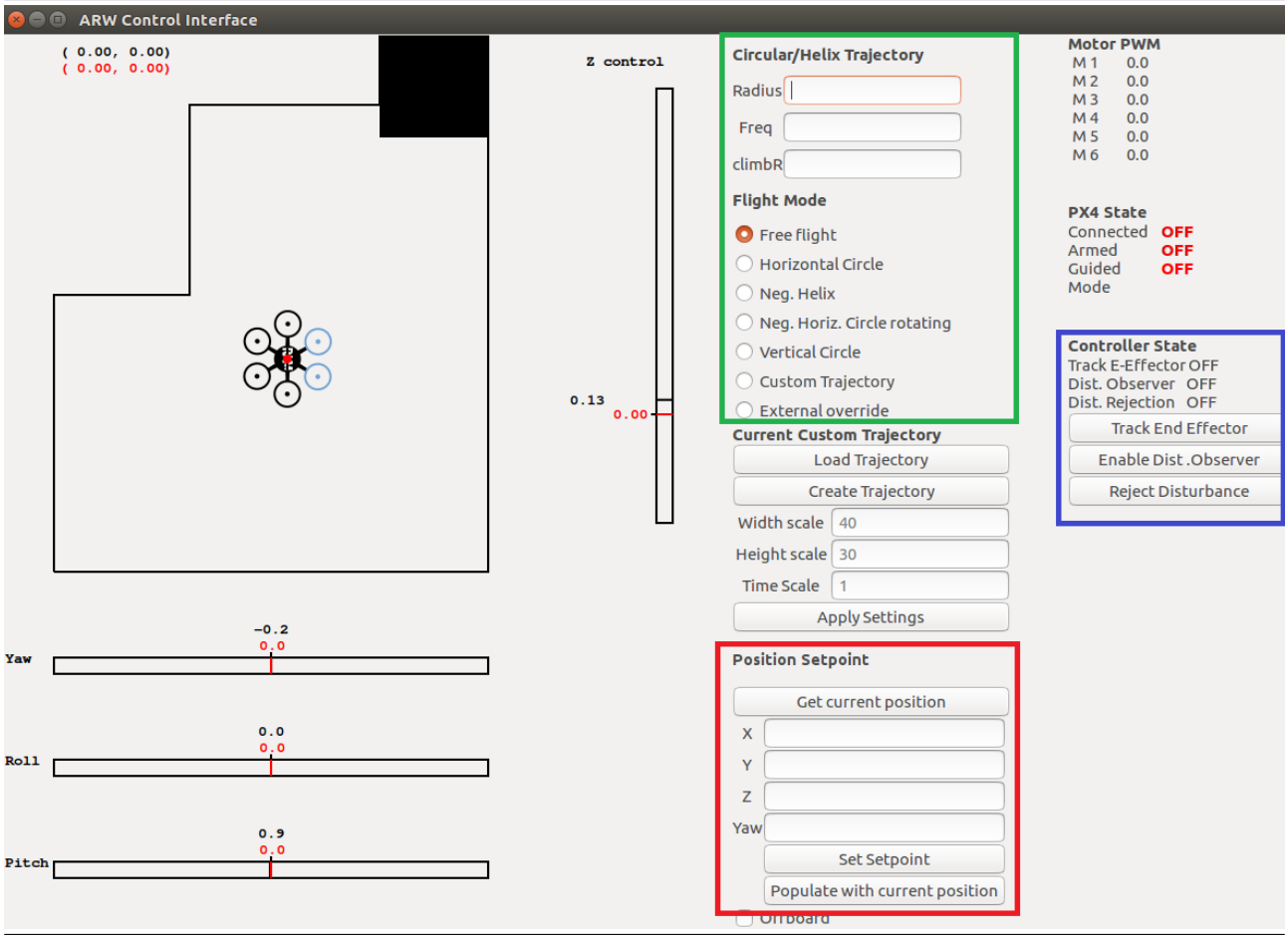
E.1 Controlling the UAV

In order to control the UAV an energy tank-based impedance/wrench controller is used [Rashad et al., 2019b]. The impedance controller extended with energy tanks guarantees that the UAV remains stable while interacting with the wall.
Next to this the offboard controller provides the message `etank_imp_control_signals`. This message returns the propeller wrench W_{pr} and an estimated external wrench $\hat{W}_{ext}$ obtained

from a loosely coupled estimator using ground truth information. The propeller wrench will be used as a measurement input for the geometric pose wrench estimator.

The setpoints to the controller are given by the Graphical User Interface (GUI) shown in figure E.2. The interfaces in the colored boxes can be used to control the UAV. The position and yaw can be manually controlled by using the interface in the red box. The green box can be used for automated flight. One can use the automated flight by setting the windows radius, freq, and climbR, and then tick the box of the desired pattern. Lastly, the blue box can be used for some additional abilities of the controller. For instance, the loosely-coupled external wrench estimation from the controller can be enabled by pushing 'enable Dist. Observer

Figure E.2 GUI to control UAV. The green box highlights the automated trajectory interface, the red box highlights the interface to manually command position setpoints and the blue box shows buttons to use additional functionalities of the off-board controller.



E.2 Simulated sensors

To simulate sensors, the plugin "Hector Gazebo" has been used. The documentation of this plugin can be found on their *wiki*. The sensors implemented are "GazeboRosImu" and "GazeboRosGps". Both sensors use the ground truth and introduce noise and biases in accordance with the models in chapter 3.

Besides the IMU and GPS, a force/torque sensor has been implemented. The sensor can be used as ground truth for the force estimation. The force/torque plugin is called "libgazebo_ros_bumper" which can be found in the Gazebo library. The force sensor gives measurements when in contact

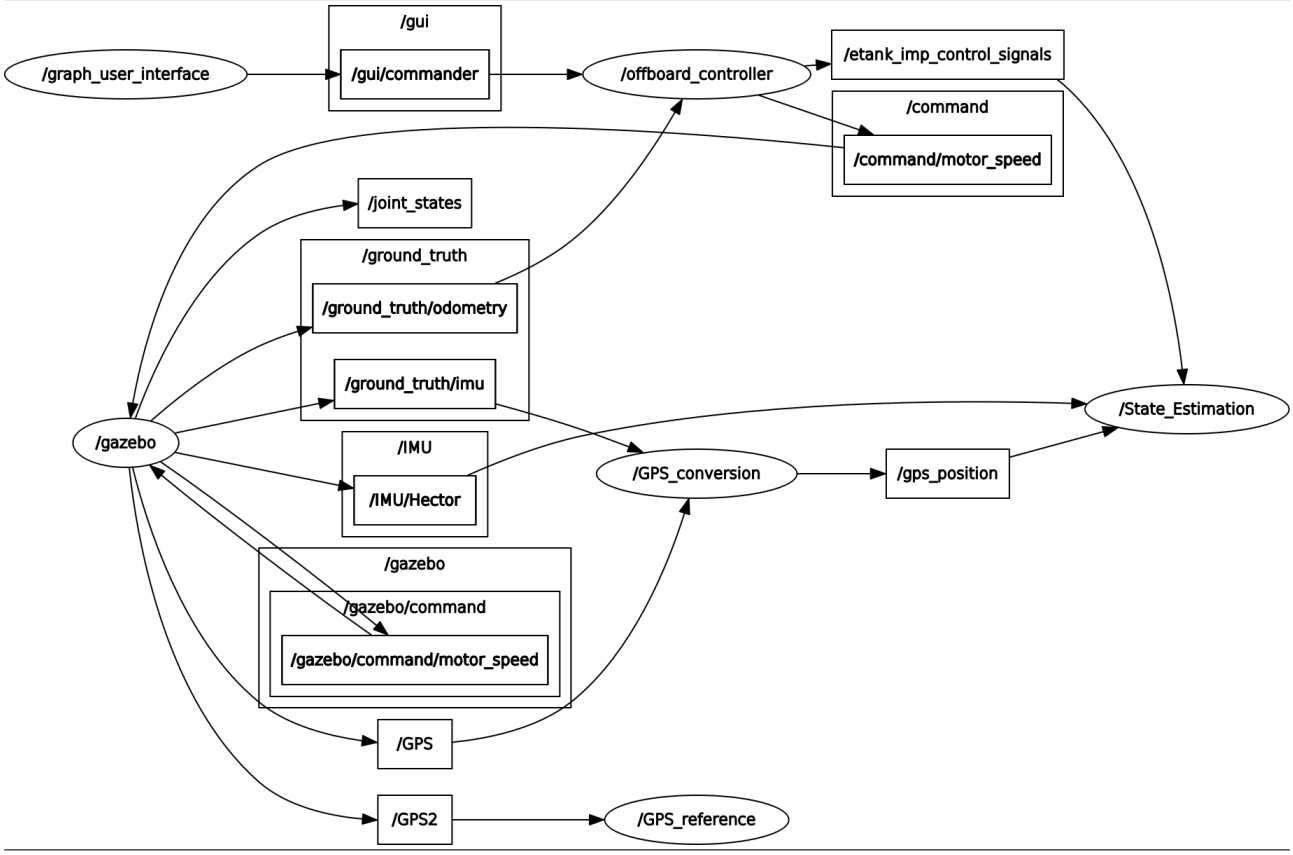
with an object. In the simulation shown in figure E.1, the sensor has been attached to the wall. A small remark is that the raw force/torque measurements are not very intuitive as they jump between zero and the measured force/torque value. Also, the measurement value is twice as large as the actual force or torque. Therefore, some processing is required before the sensor can be used.

The GPS plugin provides GPS messages expressed in the ECEF-frame, as explained in 3.2.1. Because this is unsuitable for local applications, a GPS conversion package is used. This package is called *geodetic utils* and it allows to convert GPS messages to the desired frame. In the case of the interactive UAV, the most convenient frame is the ENU frame, as the ENU frame aligns with the notation in chapter 5.

E.3 ROS architecture

Now that the used nodes have been introduced the architecture is shown in E.3.

Figure E.3 Visualization of ROS architecture. The oval shapes are nodes and the squared shapes are messages. The graph is obtained using the command \rqt_graph.



The node \gazebo is seen in the left of the graph. Gazebo uses this node to publish information, such as ground truths, to other nodes. In the upper left the node \graph_user_interface is shown. This is the control GUI which publishes the message the \offboard_controller uses to control the UAV.
In the below center part of the graph, the \GPS_reference node is shown. This node sets the initial GPS location, which is sent by the message \GPS2. For real-world experimentation, the initial GPS coordinate should to be found precisely, as deviations in the initial GPS coordinate will also lead to deviations in \gps_position. After the initial GPS coordinate is received, the

converted GPS message is published from \GPS_conversion.
The node \State_estimation is the implementation of the state estimators seen in chapter 5. The name is the same for both estimators. The \State_Estimation node subscribes on the following messages: \IMU \Hector, \gps_position and \etank_imp_controls_signals, which are the measurement inputs and observations.

E.4 Simulation parameters

IMU		
parameter	*value*	*unit*
Accelerometer noise (σ_a^2)	0.05	$\frac{m}{s^2}$
Gyroscope noise (σ_ω^2)	0.007	$\frac{rad}{s}$
Accelerometer bias ($\sigma_{b_a}^2$)	0	$\frac{m}{hzs^4}$
Gyroscope bias ($\sigma_{b_\omega}^2$)	0	$\frac{rad}{hzs^2}$
Magnetometer noise (σ_{mag}^2)	0.007	rad
Magnetometer bias ($\sigma_{b_{mag}}^2$)	0	$\frac{rad}{hzs^2}$
Update rate	200	Hz
GPS		
Latitude	52.2393971	°
Longitude	6.8508709	°
Altitude	0	m
Position noise ($\sigma_{gps,p}$)	0.2	m
rate	1 and 10	Hz
UAV		
Mass(m)	1.85	kg
Moment of Inertia - x (J_x)	0.05	$kg.m^2$
Moment of Inertia - y (J_y)	0.05	$kg.m^2$
Moment of Inertia - z (J_z)	0.094	$kg.m^2$
gravity constant (g)	9.81	$\frac{m}{s^2}$

Appendix F

Data from simulation & experiments

F.1 Simulation 1

F.1.1 Case 1

Table F.1: Raw data of tuning orientation by changing η_ω for simulated environment. Errors are calculated on R_b^i in radians. $n_{mag} = 0.05$

η_ω	axis	μ_R	σ_R	RMS_R
0.5	x	-377.5354e-006	12.5662e-009	393.8216e-006
	y	-11.1040e-006	443.5541e-012	23.8060e-006
	z	23.9892e-003	44.7973e-006	24.9051e-003
0.05	x	-408.6279e-006	27.5098e-009	440.9974e-006
	y	-39.4537e-006	3.3755e-009	70.2217e-006
	z	23.6447e-003	44.7016e-006	24.5715e-003
0.005	x	-359.0322e-006	34.9201e-009	404.7393e-006
	y	-66.0110e-006	5.7945e-009	100.7484e-006
	z	23.3782e-003	32.2658e-006	24.0582e-003
0.007	x	-342.5967e-006	35.5174e-009	390.9983e-006
	y	49.8436e-006	2.9823e-009	73.9312e-006
	z	24.3845e-003	104.5037e-006	26.4401e-003
0.0005	x	-355.2900e-006	26.5983e-009	390.9236e-006
	y	79.8152e-006	6.7115e-009	114.3677e-006
	z	27.9147e-003	58.1944e-006	28.9380e-003

Table F.2: Raw data of tuning orientation by changing n_{mag} for simulated environment. Errors are calculated on R_b^i in radians. $\eta_a = 0.005$

n_{mag}	axis	μ_R	σ_R	RMS_R
0.5	x	-347.5803e-006	11.3189e-009	363.4940e-006
	y	-1.9672e-006	121.8556e-012	11.2112e-006
	z	20.5482e-003	37.9309e-006	21.4511e-003
0.05	x	-347.5803e-006	11.3189e-009	363.4940e-006
	y	-1.9672e-006	121.8556e-012	11.2112e-006
	z	20.5482e-003	37.9309e-006	21.4511e-003
0.005	x	-356.0877e-006	12.1396e-009	372.7390e-006
	y	7.0023e-006	3.5766e-009	60.2044e-006
	z	24.2749e-003	26.0047e-006	24.8045e-003
0.007	x	-252.2787e-006	5.1074e-006	2.2737e-003
	y	-352.2569e-006	4.1537e-006	2.0680e-003
	z	31.6302e-003	80.1846e-006	32.8730e-003
0.0005	x	-330.0505e-006	13.0448e-009	349.2482e-006
	y	-59.1876e-006	2.5623e-009	77.8764e-006
	z	23.9707e-003	32.3136e-006	24.6353e-003

Other tuning parameters for case 1:

- $\eta_a = 0.05m/s^2$
- $\eta_{b_a} = 0.000005m/s^3$
- $n_{gps,p} = 0.05m$

F.1.2 Case 2

Table F.3: Raw data tuning simulation position. Errors are calculated on ξ_b^i in meters. $\eta_\omega = 0.005 rad/s$ $n_{mag} = 0.0005 rad$

n_{gps}	axis	μ_p	σ_p	RMS_p
0.005	x	-0.0923	0.1106	0.3451
	y	0.0918	0.0596	0.2607
	z	-0.0485	0.1246	0.3562
0.05	x	-0.0726	0.0529	0.2411
	y	0.0097	0.0208	0.1446
	z	-0.0466	0.0567	0.2427
0.2	x	0.0400	0.0093	0.1043
	y	0.0286	0.0093	0.1006
	z	0.0448	0.0039	0.0768
0.5	x	0.0185	0.0292	0.1718
	y	-0.0669	0.1010	0.3248
	z	-0.0210	0.0283	0.1696
1	x	0.0542	0.0690	0.2682
	y	0.1009	0.0529	0.2511
	z	0.0105	0.0270	0.1645
5	x	-0.0201	0.1067	0.3273
	y	-0.0471	0.0541	0.2372
	z	0.0659	0.0054	0.0988
10	x	-0.0338	0.0590	0.2452
	y	0.1208	0.0185	0.1820
	z	0.1077	0.0066	0.1347

F.1.3 Case 3

Other tuning parameters for case 3:

- $\eta_\omega = 0.005 rad/s$
- $\eta_{b_a} = 0.000005 m/s^3$
- $\eta_{b_\omega} = 0.000005 rad/s^2$
- $n_{gps,p} = 0.2m$
- $n_{mag} = 0.0005 rad$

Table F.5: Raw data tuning simulation position. Errors are calculated on ξ_b^i in meters.

η_a	axis	μ_p	σ_p	RMS_p
0.0005	x	0.0441	0.0904	0.3039
	y	-0.0475	0.1007	0.3208
	z	0.0131	0.0064	0.0812
0.005	x	0.0111	0.0331	0.1823
	y	-0.0500	0.0260	0.1687
	z	-0.0293	0.0288	0.1721
0.05	x	0.0161	0.0063	0.0807
	y	0.0264	0.0096	0.1015
	z	-0.0318	0.0029	0.0626
0.5	x	0.0053	0.0868	0.2947
	y	-0.0355	0.0384	0.1992
	z	0.1437	0.0334	0.2324
1	x	0.0201	0.0374	0.1945
	y	-0.0078	0.0756	0.2750
	z	-0.0665	0.0638	0.2611
5	x	0.0220	0.0666	0.2589
	y	-0.0333	0.0443	0.2132
	z	0.0423	0.0508	0.2292

Table F.4: Raw data tuning simulation velocity. Errors are calculated on $v_b^{b,i}$ in meters per second.

η_a	axis	μ_v	σ_v	RMS_v
0.0005	x	0.0093	0.0007	0.0289
	y	-0.0557	0.0006	0.0611
	z	0.0277	0.0003	0.0324
0.005	x	-0.0116	0.0002	0.0184
	y	-0.0316	0.0005	0.0387
	z	0.0062	0.0015	0.0393
0.05	x	-0.0437	0.0016	0.0594
	y	0.0338	0.0015	0.0514
	z	0.0124	0.0007	0.0291
0.5	x	-0.1518	0.0129	0.1895
	y	0.0673	0.0087	0.1151
	z	0.0084	0.0079	0.0895
1	x	-0.0636	0.0252	0.1709
	y	0.0469	0.0120	0.1192
	z	0.0012	0.0218	0.1475
5	x	-0.1117	0.0337	0.2149
	y	0.0647	0.0430	0.2171
	z	0.0251	0.0352	0.1894

F.2 Simulation 2

Other tuning parameters during simulation 2:

- $\eta_a = 0.05 m/s^2$
- $\eta_{b_a} = 0.000005 m/s^3$
- $n_{gps,p} = 0.05m$
- $\eta_\omega = 0.005 rad/s$
- $n_{mag} = 0.0005 rad$

F.2.1 Case 1

$\eta_{f_{ext}}$	$e_{f_{ext},\mu}$	e_{f_{ext},σ^2}	$e_{f_{ext},rms}$
0.0005	0.2636	0.0628	0.3635
0.005	0.1094	0.0412	0.2303
0.05	0.0453	0.0273	0.1712
0.5	0.0143	0.0495	0.2226
5	0.0097	0.0499	0.2234
50	-0.0102	0.0843	0.2903

F.2.2 Case 2

n_a	$e_{f_{ext},\mu}$	e_{f_{ext},σ^2}	$e_{f_{ext},rms}$
0.005	0.0008	0.1236	0.3511
0.05	0.0143	0.0495	0.2226
0.5	-0.0018	0.0723	0.2686
1	-0.0222	0.0527	0.2304
5	0.0155	0.0304	0.1748
10	0.0175	0.0226	0.1511
50	0.0651	0.0275	0.1779

F.3 Experiment 1

Parameter during this experiment:

$\eta_\omega = 0.01 rad/s$
$\eta_{b_a} = 0.000005 m/s^2$
$\eta_{b_\omega} = 0.000005 m/s^2$
$n_{mag} = 0.001 rad$

F.3.1 Case 1

$n_{gps,p}$	axis	$e_{n_{gps,p},\mu}$	$e_{n_{gps,p},\sigma^2}$	$e_{n_{gps,p},rms}$
0.01	x	-0.0016	0.0255	0.1596
	y	0.1822	0.0211	0.2330
	z	0.0019	0.0027	0.0523
0.05	x	0.0013	0.0105	0.1026
	y	0.0863	0.0106	0.1343
	z	0.0003	0.0026	0.0509
0.1	x	0.0031	0.0055	0.0745
	y	0.0674	0.0063	0.1039
	z	0.0019	0.0019	0.0431
0.5	x	0.0033	0.0037	0.0610
	y	0.0361	0.0043	0.0751
	z	0.0017	0.0023	0.0479
1	x	0.0037	0.0035	0.0595
	y	0.0276	0.0042	0.0708
	z	0.0013	0.0028	0.0529

F.3.2 Case 2

$n_{gps,p}$	axis	$e_{n_{gps,p},\mu}$	$e_{n_{gps,p},\sigma^2}$	$e_{n_{gps,p},rms}$
0.005	x	-0.0038	0.0053	0.0732
	y	-0.0489	0.0055	0.0887
	z	-0.0023	0.0029	0.0541
0.01	x	-0.0031	0.0049	0.0700
	y	-0.0361	0.0050	0.0795
	z	-0.0016	0.0033	0.0578
0.05	x	-0.0052	0.0056	0.0753
	y	-0.0178	0.0052	0.0744
	z	-0.0013	0.0048	0.0691
0.1	x	-0.0049	0.0064	0.0802
	y	-0.0134	0.0060	0.0788
	z	-0.0012	0.0056	0.0750
0.5	x	-0.0053	0.0097	0.0985
	y	-0.0070	0.0090	0.0952
	z	-0.0009	0.0089	0.0943
1	x	-0.0056	0.0110	0.1048
	y	-0.0055	0.0106	0.1030
	z	-0.0010	0.0104	0.1020

F.3.3 Case 3

$n_{gps,p}$	axis	$e_{n_{gps,p},\mu}$	$e_{n_{gps,p},\sigma^2}$	$e_{n_{gps,p},rms}$
0.005	x	-0.0033	0.0062	0.0791
	y	-0.0261	0.0059	0.0809
	z	-0.0014	0.0050	0.0708
0.01	x	-0.0031	0.0049	0.0700
	y	-0.0361	0.0050	0.0795
	z	-0.0016	0.0033	0.0578
0.05	x	0.0003	0.0071	0.0841
	y	-0.0756	0.0075	0.1151
	z	-0.0022	0.0027	0.0518
0.1	x	-0.0033	0.0082	0.0904
	y	-0.0757	0.0067	0.1117
	z	-0.0026	0.0029	0.0542

F.4 Experiment 2

F.4.1 Case 1

Mean errors are already given in figure 7.11
Parameters:
$\eta_a = 0.05 m/s^2$
$\eta_\omega = 0.01 rad/s$
$\eta_{b_a} = 0.000005 m/s^2$
$\eta_{b_\omega} = 0.000005 m/s^2$
$n_{mag} = 0.001 rad$
$n_{gps,p} = 0.01 m$

F.4.2 Case 2

case	axis	μ_p	σ_p	RMS_p
$\eta_{f_{pr}} = 0.1$	x	-0.0045	0.0072	0.0851
	y	0.0409	0.0177	0.1392
	z	0.0044	0.0062	0.0791
prediction lower, correction equal	x	0.0018	0.0162	0.1274
	y	0.0480	0.0177	0.1416
	z	0.0158	0.0191	0.1392
prediction equal, correction lower	x	-0.0060	0.0107	0.1034
	y	0.0275	0.0207	0.1465
	z	0.0052	0.0110	0.1051

Bibliography

[Bloesch et al., 2015] Bloesch, M., S. Omari, M. Hutter and R. Siegwart (2015), Robust visual inertial odometry using a direct EKF-based approach, in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 298–304.

[Brossard et al., 2018] Brossard, M., S. Bonnabel and A. Barrau (2018), Unscented Kalman Filter on Lie Groups for Visual Inertial Odometry, in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 649–655.

[Brossard et al., 2017] Brossard, M., S. Bonnabel and J. Condomines (2017), Unscented Kalman filtering on Lie groups, in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 2485–2491.

[Chadaporn Keatmanee, 2015] Chadaporn Keatmanee, Junaid Baber, M. B. (2015), Simple Example of Applying Extended Kalman Filter.

[Chagas and Waldmann, 2015] Chagas, R. and J. Waldmann (2015), *Observability Analysis for the INS Error Model with GPS/Uncalibrated Magnetometer Aiding*, pp. 235–257, doi: 10.1007/978-3-662-44785-7_13.

[David et al., 2017] David, W., D. Kominiak, E. Fresk and G. Nikolakopoulos (2017), A Geometric Pulling Force Controller for Aerial Robotic Workers * *This work has received funding from the European Unions Horizon 2020 Research and Innovation Program under the Grant Agreement No 644128 AEROWORKS, *IFAC-PapersOnLine*, **vol. 50**, pp. 10287–10292, doi:10.1016/j.ifacol.2017.08.1487.

[Delmerico and Scaramuzza, 2018] Delmerico, J. and D. Scaramuzza (2018), A Benchmark Comparison of Monocular Visual-Inertial Odometry Algorithms for Flying Robots, in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2502–2509.

[Duflos et al., 2006] Duflos, E., D. Pomorski and P. Vanheeghe (2006), GPS/IMU data fusion using multisensor Kalman filtering: Introduction of contextual aspects, *Information Fusion*, **vol. 7**, pp. 221–230, doi:10.1016/j.inffus.2004.07.002.

[Forster et al., 2016] Forster, C., L. Carlone, F. Dellaert and D. Scaramuzza (2016), On-Manifold Preintegration for Real-Time Visual-Inertial Odometry, *IEEE Transactions on Robotics*, **vol. PP**, doi:10.1109/TRO.2016.2597321.

[Furrer et al., 2016] Furrer, F., M. Burri, M. Achtelik and R. Siegwart (2016), *Robot Operating System (ROS): The Complete Reference (Volume 1)*, Springer International Publishing, Cham, chapter RotorS—A Modular Gazebo MAV Simulator Framework, pp. 595–625, ISBN 978-3-319-26054-9, doi:10.1007/978-3-319-26054-9_23. http://dx.doi.org/10.1007/978-3-319-26054-9_23

[Gioioso et al., 2014] Gioioso, G., M. Ryll, D. Prattichizzo, H. H. Bülthoff and A. Franchi (2014), Turning a near-hovering controlled quadrotor into a 3D force effector, in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 6278–6284.

[Grip et al., 2012] Grip, H. F., T. I. Fossero, T. A. Johansent and A. Saberi (2012), A nonlinear observer for integration of GNSS and IMU measurements with gyro bias estimation, in *2012 American Control Conference (ACC)*, pp. 4607–4612.

[Gui et al., 2015] Gui, J., D. Gu, S. Wang and H. Hu (2015), A review of visual inertial odometry from filtering and optimisation perspectives, **vol. 29**, no.20, pp. 1289–1301,

doi:10.1080/01691864.2015.1057616.
https://doi.org/10.1080/01691864.2015.1057616

[Hertzberg et al., 2011] Hertzberg, C., R. Wagner, U. Frese and L. Schröder (2011), Integrating Generic Sensor Fusion Algorithms with Sound State Representations through Encapsulation of Manifolds.

[Joan Solá, 2019] Joan Solá, Jeremie Deray, D. A. (2019), A micro Lie theory for state estimation in robotics.

[Kok et al., 2017] Kok, M., J. D. Hol and T. B. Schön (2017), *Using Inertial Sensors for Position and Orientation Estimation*.

[Li and Mourikis, 2012] Li, M. and A. I. Mourikis (2012), Improving the accuracy of EKF-based visual-inertial odometry, in *2012 IEEE International Conference on Robotics and Automation*, pp. 828–835.

[Lim et al., 2012] Lim, H., J. Park, D. Lee and H. J. Kim (2012), Build Your Own Quadrotor: Open-Source Projects on Unmanned Aerial Vehicles, **vol. 19**, no.3, pp. 33–45.

[Lynen et al., 2013] Lynen, S., M. W. Achtelik, S. Weiss, M. Chli and R. Siegwart (2013), A robust and modular multi-sensor fusion approach applied to MAV navigation, pp. 3923–3929.

[McKinnon and Schoellig, 2016] McKinnon, C. D. and A. P. Schoellig (2016), Unscented External Force and Torque Estimation for Quadrotors, *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) Daejeon Convention Center October 9-14, 2016, Daejeon, Korea*.

[Mourikis and Roumeliotis, 2007] Mourikis, A. I. and S. I. Roumeliotis (2007), A Multi-State Constraint Kalman Filter for Vision-aided Inertial Navigation, in *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pp. 3565–3572.

[N.A.Thacker, 1998] N.A.Thacker, A. (1998), Tutorial: The Kalman Filter.

[Nava et al., 2020] Nava, G., Q. Sablé, M. Tognon, D. Pucci and A. Franchi (2020), Direct Force Feedback Control and Online Multi-Task Optimization for Aerial Manipulators, **vol. 5**, no.2, pp. 331–338.

[Nisar et al., 2019] Nisar, B., P. Foehn, D. Falanga and D. Scaramuzza (2019), VIMO: Simultaneous Visual Inertial Model-Based Odometry and Force Estimation, **vol. 4**, no.3, pp. 2785–2792.

[Nøkland, 2011] Nøkland, H. (2011), *Nonlinear Observer Design for GNSS and IMU Integration*, Master's thesis, Norwegian University of Science and Technology Department of Engineering Cybernetics.

[Qasem and Reindl, 2007] Qasem, H. and L. Reindl (2007), Unscented and Extended Kalman Estimators for non Linear Indoor Tracking Using Distance Measurements, in *2007 4th Workshop on Positioning, Navigation and Communication*, pp. 177–181.

[Qin et al., 2018] Qin, T., P. Li and S. Shen (2018), VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator, **vol. 34**, no.4, pp. 1004–1020.

[Rajappa et al., 2017] Rajappa, S., H. Bülthoff and P. Stegagno (2017), Design and implementation of a novel architecture for physical human-UAV interaction, *The International Journal of Robotics Research*, **vol. 36**, p. 027836491770803, doi:10.1177/0278364917708038.

[Rashad et al., 2019a] Rashad, R., F. Califano and S. Stramigioli (2019a), Port-Hamiltonian Passivity-Based Control on SE(3) of a Fully Actuated UAV for Aerial Physical Interaction Near-Hovering, **vol. 4**, no.4, pp. 4378–4385.

[Rashad et al., 2019b] Rashad, R., J. B. C. Engelen and S. Stramigioli (2019b), Energy Tank-Based Wrench/Impedance Control of a Fully-Actuated Hexarotor: A Geometric Port-Hamiltonian Approach, pp. 6418–6424.

[Ryll et al., 2017] Ryll, M., G. Muscio, F. Pierri, E. Cataldi, G. Antonelli, F. Caccavale and

A. Franchi (2017), 6D physical interaction with a fully actuated aerial robot, in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5190–5195.

[S. Stramigioli, 2001] S. Stramigioli, H. B. (2001), Geometry and screw theory for robotics, *ICRA 2001*.

[Scaramuzza and Fraundorfer, 2011] Scaramuzza, D. and F. Fraundorfer (2011), Visual Odometry [Tutorial], **vol. 18**, no.4, pp. 80–92.

[Stachniss, 2013a] Stachniss, C. (2013a), SLAM03 - EKF. http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/slam05-ekf-slam.pdf

[Stachniss, 2013b] Stachniss, C. (2013b), SLAM06 - UKF. http://ais.informatik.uni-freiburg.de/teaching/ws13/mapping/pdf/slam06-ukf.pdf

[Stramigioli, 2018] Stramigioli, S. (2018), Modern Robotics.

[Sun et al., 2018] Sun, K., K. Mohta, B. Pfrommer, M. Watterson, S. Liu, Y. Mulgaonkar, C. J. Taylor and V. Kumar (2018), Robust Stereo Visual Inertial Odometry for Fast Autonomous Flight, **vol. 3**, no.2, pp. 965–972.

[Tomić and Haddadin, 2014] Tomić, T. and S. Haddadin (2014), A unified framework for external wrench estimation, interaction control and collision reflexes for flying robots, in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 4197–4204.

[Tomić and Haddadin, 2015] Tomić, T. and S. Haddadin (2015), Simultaneous estimation of aerodynamic and contact forces in flying robots: Applications to metric wind estimation and collision detection, in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 5290–5296.

[Wang et al., 2018] Wang, Y., T. Zhang, Y. Wang, J. Ma, Y. Li and J. Han (2018), Compass Aided Visual-Inertial Odometry, *Journal of Visual Communication and Image Representation*, **vol. 60**, doi:10.1016/j.jvcir.2018.12.029.

[Wopereis et al., 2017] Wopereis, H. W., J. J. Hoekstra, T. H. Post, G. A. Folkertsma, S. Stramigioli and M. Fumagalli (2017), Application of substantial and sustained force to vertical surfaces using a quadrotor, in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 2704–2709.

[Yüksel et al., 2014] Yüksel, B., C. Secchi, H. H. Bülthoff and A. Franchi (2014), A nonlinear force observer for quadrotors and application to physical interactive tasks, pp. 433–440.