NTNU – Trondheim
Norwegian University of
Science and Technology

Autonomous Localization and Tracking for UAVs using Kalman Filtering

Johan Jansen

Master of Science in Informatics
Submission date: July 2014
Supervisor: Helge Langseth, IDI

Norwegian University of Science and Technology
Department of Computer and Information Science

NTNU

Abstract

Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Information Science

Master of Science

Autonomous Localization and Tracking for UAVs using Kalman Filtering

by Johan Jansen

Quadcopters and other drones have become more popular as they become more affordable and easier to use. As the accessibility increases, people discover new fields of application and one such potential use is for filming aerial footage. However, piloting a quadcopter requires a lot of training and also requires the pilot to have a visual of where the drone is. This thesis explores on how the drone can be completely automated without the need of a pilot.

Using a small device called a tracker, the user simply presses a button which makes the drone autonomously take off and begin following the person carrying the tracker. The drone will keep a safe distance from the user, while always pointing its camera towards the tracker device with the help of gimbal servos to control camera angle. The quadcopter can adapt to different altitude topologies, for example if the user is climbing or skiing downhill. At any point, the user can land the drone with the click of a button.

This thesis focuses on how cheap drone sensors which suffer from noise and inaccuracy, can be fused together with the help of a Kalman Filter algorithm to generate more accurate localization data for better autonomous navigation. The Kalman Filter algorithm has previously been too complex to run on quadcopter hardware, but new hardware advancements has opened the possibility to explore and research its applicability on a drone. Better sensors exist, but these are very expensive and can cost more than the drone itself. The thesis begins with describing the abilities and limitations of each sensor, then showing how they can be fused together for more accurate navigation data than what the sensors can individually produce by themselves.

Acknowledgements

I wish to thank my supervisor professor Helge Langseth and assisting supervisor Boye Annfelt Høverstad for their advice, aid and guidance, bringing this thesis to completion. Their expertise has been invaluable for my research and reaching my goals.

Besides my supervisors, I also wish to thank the CPTR team: Anders B. Eie, Ole Jørgen Seeland, Sigurd Gran-Jansen and Karen Juul Skarbø for offering me the opportunity to work with drones and access to any technological equipment I needed for my research.

Finally, I would also like to thank my family for their helpfulness and support throughout the year, with special thanks to Arne Kristian Jansen for his valuable feedback and aid in writing this thesis.

Contents

List of Figures

List of Tables

Abbreviations

AI	**A**rtificial **I**ntelligence
AHRS	**A**ttitude and **H**eading **R**eference **S**ystem
CPU	**C**entral **P**rocessing **U**nit
DCM	**D**irecton **C**osine **M**atrix
EKF	**E**xtended **K**alman **F**ilter
GNSS	**G**lobal **N**avigation **S**atellite **S**ystem
GDOP	**G**eometric **D**ilution of **P**recision
GPS	**G**lobal **P**ositioning **S**ystem
FPU	**F**loating **P**oint **U**nit
HDOP	**H**orizontal **D**ilution **O**f **P**recision
INS	**I**nertial **N**avigation **S**ystem
IMU	**I**nertial **M**easuring **U**nit
KF	**K**alman **F**ilter (also known as LQE)
LQE	**L**inear **Q**uadratic **E**stimation (same as Kalman Filter)
MCU	**M**icro **C**ontroller **U**nit
NED	**N**orth **E**ast **D**own (coordinate system)
SDRE	**S**tate **D**ependent **R**icatti **E**quation
SLAM	**S**imultaneous **L**ocation **A**nd **M**apping
UAV	**U**nmanned **A**erial **V**ehicle
UKF	**U**nscented **K**alman **F**ilter
VDOP	**V**ertical **D**ilution **O**f **P**recision

Chapter 1

Introduction

1.1 Background

Interest in autonomous flying drones has grown within the field of AI robotics as micro aerial vehicles have become more affordable and practical to use[1]. Flying drones introduce different capabilities and challenges compared to ground vehicles. In addition to enabling more movement freedom in the form of flight, aerial drones also introduce complications such as limited on-board sensors, computing power and high velocity movements compared to ground robotics.

The goal of this thesis is to get a drone flying autonomously (without a pilot controlling the quadcopter) by following a person using only data from sensors like Global Navigation Satellite System (abbreviated GNSS[1]), accelerometers, magnetometers, barometer and gyroscopes. Complications we have to resolve include time delays in communication, actuator delays, processing delays on the micro-controller and sensor inaccuracies. Because of these delays and inaccurate data readings, we will need some method to filter and remove noise from data to generate better predictions and resolutions over time.

[1]The term GNSS describes a device that receive its location (Longitude, Latitude and Altitude) through satellite navigation. As of April 2013, only the United States NAVSTAR Global Positioning System (GPS) and the Russian GLONASS are operational GNSSs.

1.2 The Vision

The main goal is to have the drone follow a person autonomously, keeping a safe distance while always facing the camera towards the person (shown in Figure 1.1). This is accomplished by using sensors attached to the drone and sensors attached to the person. The sensor module attached to the person will from this point on be referred to as the *tracker*. Wireless technology[2] is used for communication between the drone and the tracker. See Appendix C for a more detailed overview of the Tracker.

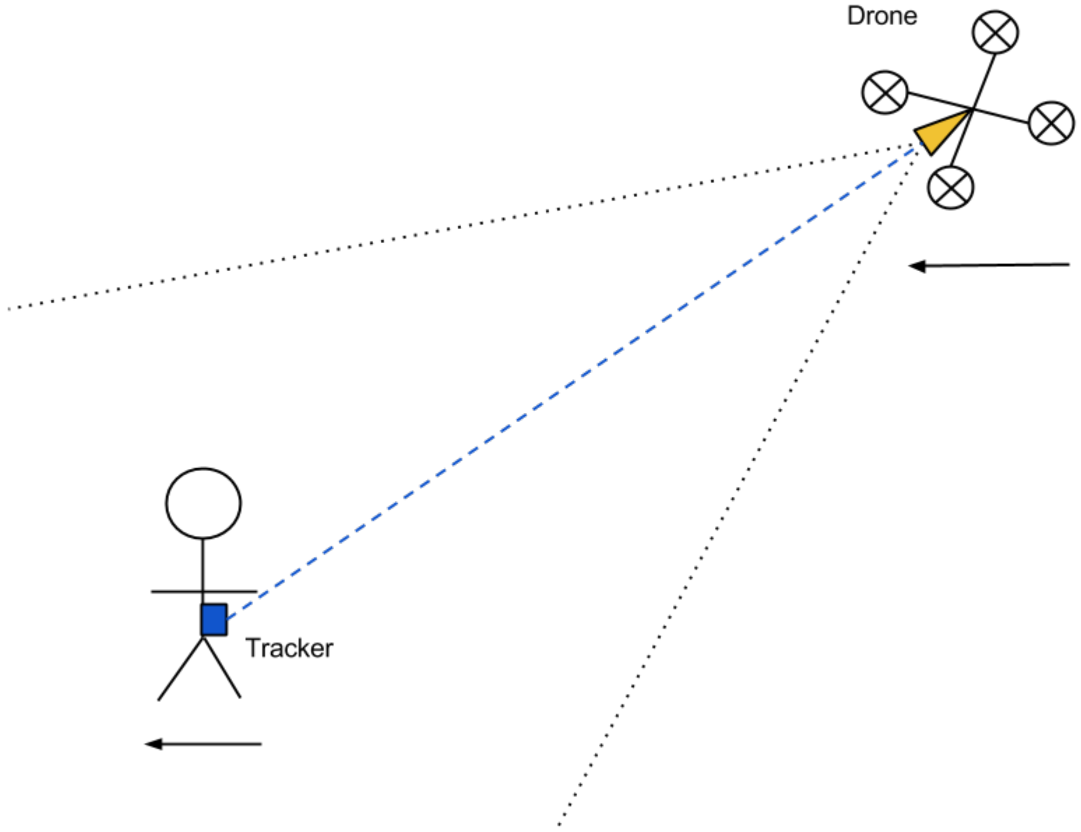


FIGURE 1.1: A concept figure showing the drone autonomously following and filming a person carrying the tracker module.

In addition, the drone has a camera for filming whatever the tracker is attached to, but the camera is only used for filming and does not use any sort of image processing or computer vision. For the drone to be able to film the tracker, it needs to know the

[2]XBee wireless modules which are based on the IEEE 802.15.4-2003 standard radio technology was used for implementing and testing this system. These small, low power-consumption devices provide up to 1.6 km communication distance.

exact position of itself relative to the position of the tracker. The drone can then use this data to control its engines to fly accordingly, so that the camera of the drone points towards the tracker. The camera is equipped with a gimbal controller, which stabilizes the video footage and provides servo control to adjust camera angles. This allows the camera always point towards the tracker, even when the drone is flying straight above the tracker and when the quadcopter is pitching forward or backwards or rolling sideways.

1.3 Short introduction into Quadrocopters

This thesis focuses on the research of autonomous small aerial vehicles (also called Unmanned Aerial Vehicles or UAVs), more specifically quadcopter drones[3]. In this section I will briefly introduce the terminology and concepts of a quadrocopter drone which will be useful in understanding the rest of the thesis. Throughout the thesis, the terms *drone*, *UAV* and *quadcopter* used interchangeably to describe the same flying vehicle used in the experiments.

A quadcopter drone has 6 degrees of freedom, movement in 3-dimensional space (X, Y and Z) and rotation along all its axises (roll, yaw and pitch). See Figure 1.2. Changes in pitch controls acceleration forwards and backwards in the X direction, while changes in roll affects left and right acceleration in the Y direction. To move up and down (Z direction) the quadcopter changes the speed of its engines. Yaw controls which direction the drone is facing in rotation around the Z axis.

External interferences such as wind, air flow and instability in the propeller engines cause some uncertainties in movement that the drone has to compensate for. This is especially important in autonomous drones who have to rely completely on uncertain sensor data as well. A proportional-integral-derivative controller is often used to regulate movement, reduce error and adjust for irregular movement[3].

Another complicating factor is that quadcopters are always in movement except when they have landed. Even when hovering in one place, the drone still drifts and moves because all the engines have to be active at all times to keep the drone airborne. By combining GNSS with an Inertial Navigation System (INS), one can significantly improve

[3]A model aircraft as defined in the Academy of Model Aeronautics National Model Aircraft Safety Code by Fédération Aéronautique Internationale[2]

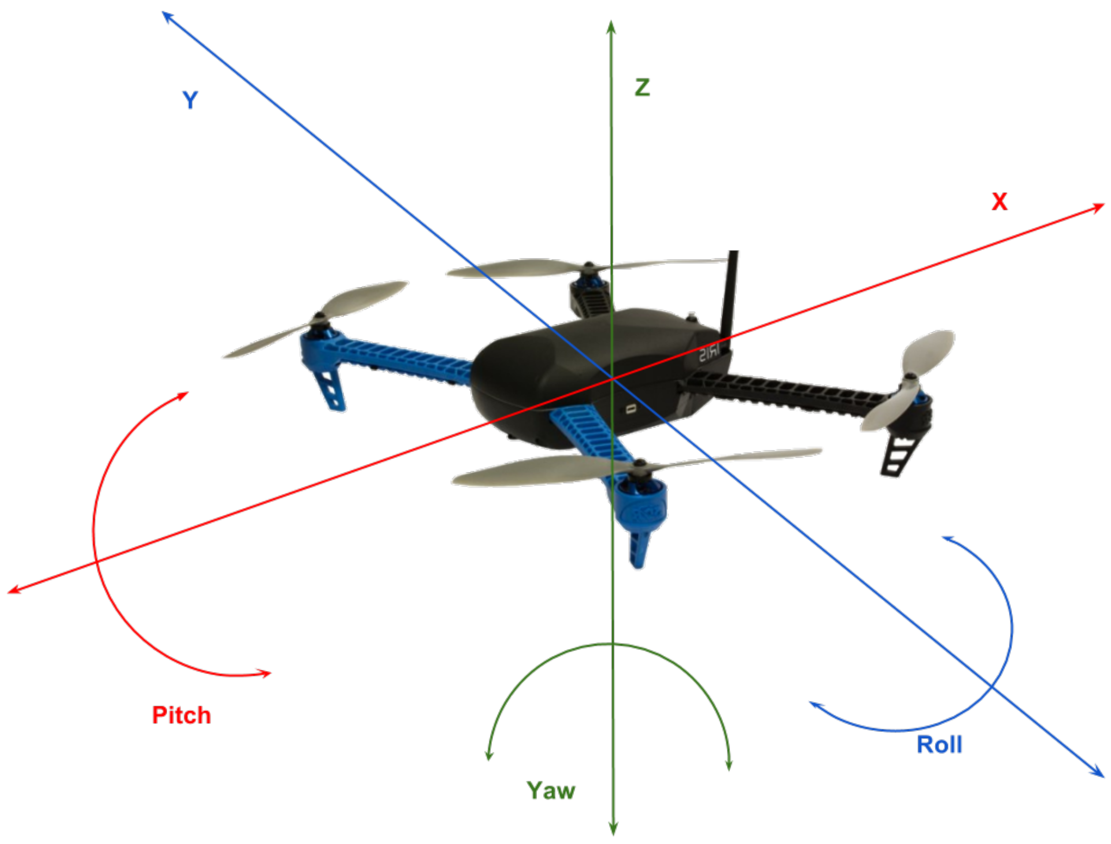


FIGURE 1.2: A quadcopter showing the 6 degrees of freedom associated with UAV's.

the hover performance of the drone[4]. The process of combining the data of two sensors to produce more accurate data is called *data fusion*.

1.3.1 Drone Sensors

Using sensors in UAV's introduces some new challenges compared to ground vehicles. For example sensory data such as sonar which work reliable on the ground, work less reliable on drones because of noise generated by the drone itself. Aerial drones also consume a lot of power, meaning their flight time is limited to relative short intervals of usually about 10 minutes. Another complication is that the Quadcopter has to be in constant movement to maintain altitude and position. Even when hovering in place, the quadcopter has to compensate for drift, wind and any unbalance in one of the four separate engines.

Sensor capabilities vary from drone to drone, but usually include the following:

1. Accelerometer, which gives us 3 degree acceleration sensors

2. Gyroscopes, that tell us about the drone's roll, pitch and yaw orientation (rotation sensors)
3. Magnetometer, that gives us a directional sense
4. A camera, which provides a visual feedback
5. Rangefinder sensors, that can detect distances to surfaces or objects around the drone
6. A GNSS module that gives us an estimation on the drone's absolute position in the world
7. Barometer measuring air pressure giving us estimations for drone altitude

1.3.2 IMU sensors expanded into an INS

The gyroscope, magnetometer and accelerometer is usually described together as an Inertial Measuring Unit. The combined IMU gives us the accelerations and rotations of the drone, which can be observed over time to give an estimated position, velocity and orientation resulting in an Inertial Navigation System (INS). An INS can estimate the location of the drone without the need of external references (such as GNSS). However, the error in the estimation grows over time and becomes very inaccurate after 10 seconds or more[5].

Sensor fusion between INS and GNSS can greatly reduce the error and improve overall localization. To be able to fuse these sensory data together over a period of time and create an accurate view of the drone state, a filter mechanism is usually required (as mentioned in Chapter 3). Combining GNSS and INS data usually gives us a stable estimation with minimal error. The INS is also useful as a short-term fallback when the GNSS stops working (for example losing satellite reception) as described by [1, 5].

1.3.3 Fixed-Wing vs Multi-Rotor

Note there is a large distinction between fixed-wing UAV (used in [6]) and multi-rotor (e.g a quadcopter). A multi-rotor UAVs has the added benefit of being able to hover in one place, while the fixed-wing UAVs has to be in high velocity movement at all time to keep it in the air. In addition the multi-rotor UAVs have usually a much faster turn rate, being able to rotate in place around it's Z axis (yaw). This means different flightpath generation algorithms have to be used depending on the type of UAV. In this thesis we only focus on multi-rotor UAVs which is used in the testing.

1.4 Filters in Signal Processing

The sensor data obtained by the drone can over time be observed as an stream of incoming signal data that also contains noise, errors and even gaps of data. A signal *filter* can process the incoming data from the stream so that it is more reliable and accurate, making it very useful for sensor fusion algorithms. Different filtering algorithms exists and have different properties such as accuracy, computation cost and complexity. Why this is needed in our system is explained in Section 3.2.

1.5 Thesis Objectives

This section describes a set of objectives which will be attempted to solve throughout this thesis. These objectives give a guideline on what the author aims to accomplish at the end and the scope of the thesis.

1. Examine existing data fusion, tracking and localization technologies to determine what is applicable for this project (reviewed in Chapter 2).
2. Evaluate the accuracy and performance of the data fusion methods in both real and simulated environments.
3. Determine which solution is applicable to work on the hardware platform available (limited by memory usage and processing complexity as described in section 3.2).
4. Implement a tracking device that allows the drone to follow and track a person autonomously.
5. Provide automatic takeoff and landing using the on-board Pixhawk autopilot (The Pixhawk autopilot is described in Section 3.7).
6. Implement data fusion algorithm to improve sensor accuracy and reliability.

Chapter 2

State of the Art

2.1 Literature Review

As mentioned in Section 3.2, the sensors available to us have inaccuracies in sensor readings. In addition to noisy data, we also have communication delay between tracker and the drone. There can even occur gaps og missing data since GNSS coverage and wireless communication is not reliable. Even with advantageous conditions, the GNSS location data received is inaccurate at best, usually on the order of a couple of meters. Combined with movement drift caused by wind and engines, the drone will need a method to keep it from diverging too much from it's planned flightpath. One solution is to use sensor fusion between an INS and GNSS receiver to improve reliability and accuracy in the localization data. Quigley et al. [6] has implemented a fixed-wing UAV tracking a target through GNSS localization with the help of extrapolation to generate smooth flight-paths. The flightpath for a fixed-wing is significantly different from a quad-rotor however, and a gimbal[1] is used to make the camera always face towards the target it is tracking while flying in circles around it.

Sensor fusion is usually accomplished through a filtering method, such as the Kalman Filter[1, 4, 7]. The Kalman Filter was introduced by Rudolf Kalman in 1960 for solving linear filtering and prediction problems[7]. Since the paper was published, many variations and improvements have been proposed and discussed in different papers and articles. The alternative methods and their papers will be discussed later in this section.

[1]A gimbal is a device that stabilizes and rotates the camera so that it always faces a fixed position with the help of servo engines, accelerometer and a gyroscope.

Higgins [8] compares the Complimentary Filter and the Kalman Filter as two different sensor fusion algorithms. The Complimentary Filter is the current sensor fusion solution for the drone because of the implementation simplicity and low computational cost of the algorithm compared to other alternatives such as Kalman Filtering or EKF. With recent improvements in MCU hardware however, more complex alternatives such as Kalman Filters have become a viable option for sensor fusion. Higgins [8] concludes that while the Complimentary Filter is indeed much simpler in both implementation and computational cost, it's accuracy compared to a mathematically proven solution such as Kalman Filtering remains uncertain.

Wendel et al. [1], Yoo and Ahn [4] show that accuracy in localization is significantly improved both in simulations and under real flight tests with help of a Kalman Filter. The UAVs in these solutions use GNSS and INS for localization. The articles describe the sensor components as cost constrained and therefore also more prone from suffering in accuracy and reliability. Periods of GNSS loss is compensated by navigating on only the INS, where the GNSS loss is handled as a reading error. The INS does accumulate errors however and it is shown that without the periodically correction of a GNSS, the INS drifts away to unusable levels in a short period of time. Yoo and Ahn [4] shows that accuracy of the GNSS is tightly coupled with the amount of satellites connected to the receiver. When the coverage becomes too poor, the system has a bias towards the now more reliable INS which improves the overall accuracy of the navigation data. The UAV is shown to be able to localize itself even after 20 seconds of GNSS loss.

A similar solution was implemented by Kumar [9], N.Va et al. [10] uses the Kalman Filter solution to fuse data of GNSS and INS both in a simulated environment and on actual hardware. The resulting Kalman filter has nine different state estimates, including position, velocity and attitude[2]. It proved that the aircraft could continue navigating after losing GNSS coverage for up to 8 seconds, albeit with reduced accuracy. Note that this solution does not include data from an altimeter to estimate altitude positions. The positions estimated by this solution is calculated by estimating the error in the INS position estimates. The error is estimated by the Kalman Filter by observing the difference in position between the GNSS and INS which is then used to correct the position estimate of the INS. The solution proposed by N.Va et al. [10] shows that

[2]The *attitude* of a UAV is the combined roll, pitch and yaw relative to the earth surface.

position estimates are improved by sensor fusion, improving the accuracy by about 1-2 meters.

Jonghyuk [11] adds an additional sensor in addition to the GNSS and INS, the Altimeter which improves altitude position estimates in autonomous UAV flight systems. This solution is used for aiding in navigating and mapping an unknown environment, an algorithm called Simultaneous Localization and Mapping (SLAM). It shows that low cost sensors can be used in real-time to improve the position estimates of an autonomous aircraft. Without the aid of a Barometer, the altitude estimates are very inaccurate and unreliable due to the inaccuracy of the GNSS altitude position fixes and the unbounded drift of the INS which requires periodically corrections to limit the drift[5].

The Kalman Filter is designed to solve problems that are linear. Depending on what sensor observations and which estimate predictions we want to make, the underlying system can be either linear or nonlinear. Because the Kalman Filter only works on linear systems, different methods have been developed for nonlinear problems. Take for example the position of an UAV in 3-dimensional space that tracks a different target in the same 3D space. The system dynamics is expressed as the targets cartesian location (position, velocity and acceleration), which gives us a nonlinear relationship between the measurements and the system state. The normal Kalman Filter can give poor performance here, while a nonlinear state-estimation algorithm such as the Unscented Kalman Filter can significantly improve accuracy[12]. An alternate solution is to not observe acceleration, which makes the underlying system linear if we assume that the acceleration is constant in a given timestep.

An alternative filtering method called the Extended Kalman Filter is described by George and Sukkarieh [13], Mao et al. [14] which addresses several problems of the Kalman Filter and is described according to Julier and Uhlmann [15] as the de-facto standard in the theory of nonlinear state estimation, navigation systems and GPS. For cases where systems are nonlinear, the EKF can be used instead of the Kalman Filter which requires linear equations for the measurement and state-transition models. In EKF these requirements are relaxed and models need only to be differentiable instead of linear. EKF works by transforming nonlinear models at each timestep into linearized systems and these linearized systems can be used in the standard Kalman Filter, which is accomplished by applying Taylor Series expansions from calculus. Gray and Maybeck

[16] show how the EKF algorithm can be applied to integrate GPS, Barometer, INS and Radar Altimeter together to generate high precision navigation for landing. The paper shows that these more expensive, reliable and accurate sensors are used for manned aircraft, while the UAVs used in this thesis use cheap and noisy sensors that are prone to errors. The same sensor fusion theory can however be applied to UAVs assuming the hardware on the drone is able to run the EKF fast enough to provide real-time data.

Julier and Uhlmann [15], Wan and van der Menve [17] describe Unscented Kalman Filtering is an alternative nonlinear filter to the EKF. It uses the unscented transform method to pick a minimal set of sample points (called sigma points) around the mean. These points are then used to calculate the mean and covariance of the estimate which removes the requirement to calculate the Jacobian which EKF requires. The nonlinear UKF implemented by St-Pierre and Ing. [12], Julier and Uhlmann [15] are used for improving precision in navigation systems and are compared to the (extended) Kalman filter. St-Pierre and Ing. [12] concludes that the computation time of UKF is significantly higher (2250% increase) with only a slight precision improvement when GPS is available and worse estimation when it is unavailable (only IMU). This contradicts with the results by Wan and van der Menve [17] which conclude that UKF is superior to the EKF, all agree however that the UKF is easier to implement in comparison to EKF.

Mao et al. [14] uses EKF to compensate for GPS loss by measuring distance to two other UAVs. For this system to work, an UAV has to fly within communication range of at least two other UAVs to compensate for GPS loss. The distance metrics used in these articles is in the magnitude of tenfolds of meters, if not kilometers. The problem this thesis focuses on (as described in Section 1.2), requires accuracy of a couple of meters and using multiple UAVs is not very cost effective compared to the alternatives. Furthermore the multiple UAVs is also not applicable to the tracking problem this thesis focuses on.

An entirely different method to solve UAV localization implemented by Kim and Sukkarieh [18] uses a video camera for computer vision to continue generating a flightpath in GNSS denied environments. However, computer vision is computationally expensive and requires more powerful hardware than what is usually available on-board on a drone (often requiring a wirelessly connected ground station computer). This method does introduce the possibility of on-board collision avoidance using computer vision, something which only GPS and INS sensors cannot provide. Furthermore, computer vision is very susceptible to lighting, shadows and surfaces that are transparent or reflective. Achtelik et al. [19] use a stereo camera system combined with an INS to stabilize a quadcopter. Computer vision is used for visual tracking and fused with data from the INS data to improve stabilization. In their conclusion Achtelik et al. [19] mention that a Kalman Filter could further improve the accuracy of the sensor fusion algorithm. A laptop was used in their tests to run computer vision algorithms for visual tracking.

2.1.1 Litterature Review Conclusion

Several solutions have been presented and for a tracking problem, using GPS combined with a sensor fusion method is the most cost-effective and reliable solution given the alternatives explored in this chapter. This solution also fits with our hardware constraints introduced in Section 3.2 (e.g computer vision is not applicable with the current processing power available). If the underlying system is linear, then a normal Kalman Filter should be sufficient for good estimates. Papers describing the different sensor fusion methods contradict in which solution is the most accurate and not all of them include computation cost or memory complexity in their testing, which means this will have to be examined closer through the thesis.

If we assume that the acceleration is constant in a given timestep, we can model the system as linear and use the normal Kalman Filter for state estimates. The solution will first run in a simulated environment on a computer using real flight test data. If the implementation and model is verified, then it will be implemented on the drone hardware platform and evaluated real-time in flight using its autonomous flight system.

Chapter 3

Hardware Platform

3.1 The Quadcopter

This section describes the framework of the quadcopter, which includes the body, arms, engines, battery and propellers of the drone. The propeller engines attached to each arm are controlled by the Pixhawk module described in Section 3.7. Ultimately a new design will be developed for the quadcopter, which at the time of writing is still under development. A concept sketch of the CPTR[1] quadcopter is shown in Figure 3.1. It is designed for easy portability (small size and foldable arms), longer battery lifetime, gimbal-stabilized camera and provides an array of different navigation sensors which will be described in this section.

Until this design is complete however, a 3DR Iris quadcopter was used for testing purposes and development of the tracking system, shown in Figure 3.2. This prototype quadcopter contains the same hardware as the CPTR quadcopter to ensure that it will behave the same when it is finished. At the time of writing the 3DR-Iris quadcopter has a maximum speed of 47 kilometers per hour which indicates how fast the drone must react and how accurate the navigation data needs to be for it to safely operate under these velocities.

[1]CPTR is the name of the new drone design and is an disemvowelement of the word *copter*

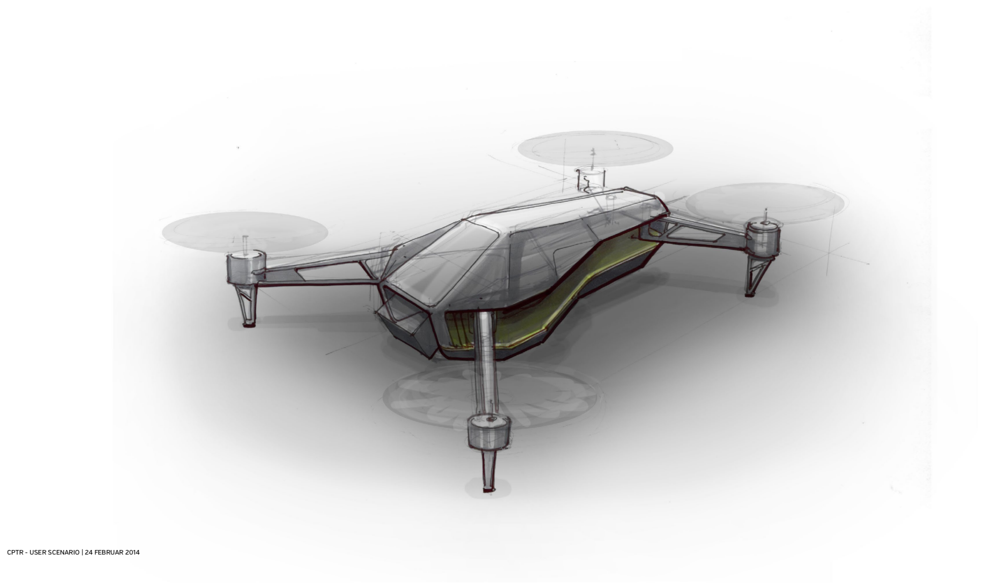


FIGURE 3.1: Concept sketch of the CPTR quadcopter. Used with premission from CPTR.

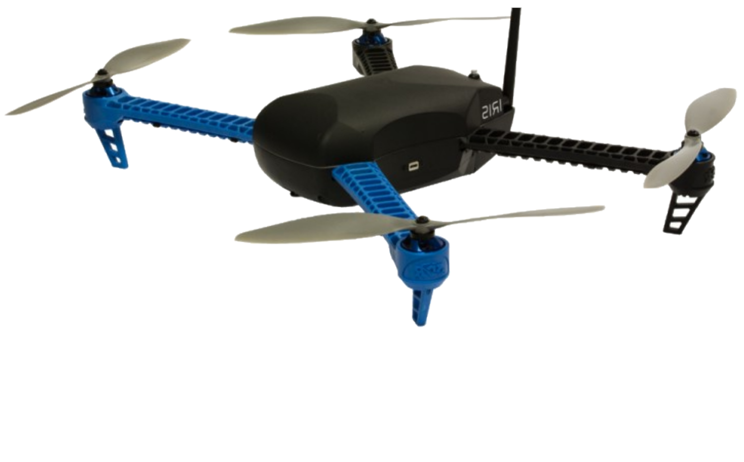


FIGURE 3.2: The 3DR Iris which was used for prototype testing.

3.2 Sensor Limitations

The entire system is built from different hardware components which introduce some limitations. Sensor readings are inaccurate and unreliable and on top of that we introduce wireless communication delay which can cause further complications. From these limitations arises the necessity of a good filtering method for sensor fusion. The filtering algorithm must be able to run on a microprocessor which is very limited in terms of processing power and available memory compared to a traditional desktop computer.

Figure 3.3 gives an overview of all the sensors we fuse (indicated by green boxes) through the Micro-Controller Unit (MCU) on the drone (yellow box) which tells the *Pixhawk* flight controller (red box) how to fly. The following list summarizes the sensor suit available on both the drone and the tracker device:

1. IMU: 9 Degrees of Freedom - MPU-9150 Inertial Measurement Unit
2. GNSS: 3DR GPS Module uBlox LEA-6 with Compass
3. Barometer: MS5611 Barometric Pressure Sensor

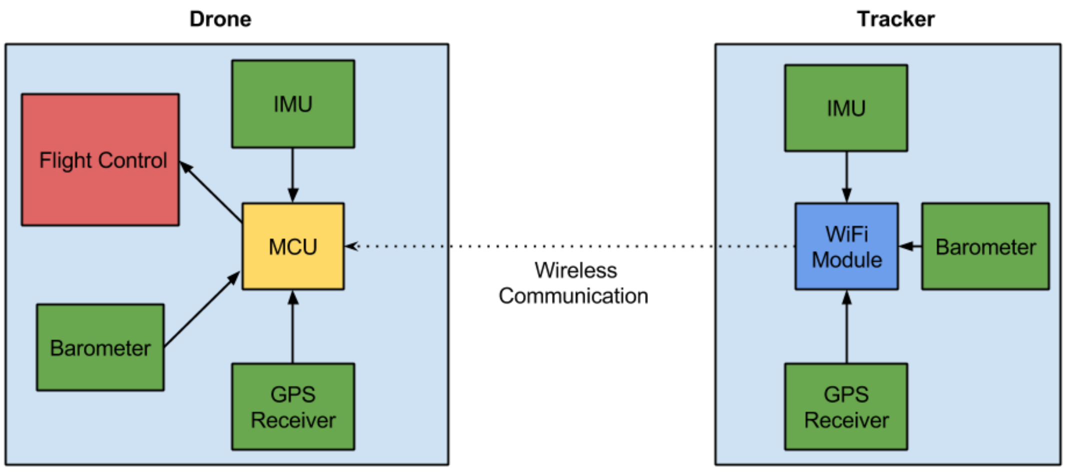


Figure 3.3: An overview of data flow between the components of the entire system.

The rest of this chapter will go a little more into depth for each of the main components of the hardware system and how they will affect the objectives of this thesis.

3.3 IMU/INS

The Inertial Measurement Unit (IMU) operates at a 100Hz sampling rate, which gives us a new reading every 10 milliseconds. It provides sensor readings for acceleration, gyroscopic readings and magnetic values, providing the drone with 9-degrees of freedom sensory awareness. The IMU provides very fast and very accurate readings, but contains a small error that accumulates over time and is unbounded. Furthermore it has constant availability, though strong magnetic fields can disrupt the readings for the magnetometer.

The major limitation, however, is an unbounded accuracy drift that increases over time. If we try to completely rely on the IMU for localization without corrections of an absolute position like GNSS, the error will drift so much that the IMU readings become useless for localizing the drone. Figure 3.4 shows how the sensor readings drift over time:

1. Blue blocky line is the requested rates by the angle controller
2. Red is the current rate
3. The rate error is in green
4. The motor output is purple

The data was sampled while the drone was hovering still in the air which produced a $0.4\,^\circ$ to $0.8\,^\circ$ noisy readings for yaw rotation. This is most likely caused by vibrations, something which the gyroscopes are very sensitive to. Fusing the gyroscope data with magnetometer data could reduce this error.

An IMU can be turned into an Inertial Navigation System (INS) by transforming gyroscopic, acceleration and magnetic data into velocity, orientation and direction estimates. We can rely on dead-reckoning and the INS to provide us with an accurate estimation of our position compared to a position earlier in time. Because of the unbounded error inherent in the IMU, the INS is dependent on external periodic corrections such as a GNSS to keep the error from growing indefinitely. A more in depth look at the design of the INS used in the system is described in Section 4.7.

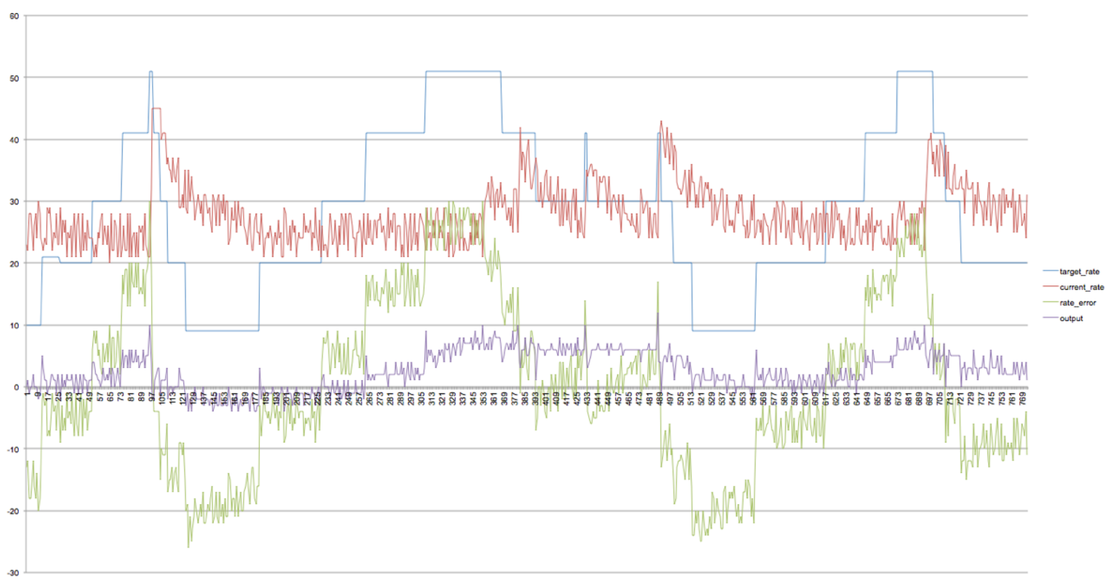


FIGURE 3.4: Yaw rate sampling from the drone using 100Hz sampling rate while the drone is hovering perfectly still.

3.4 GNSS

The LEA-6H GPS module used to track the global position has a 5hz sampling rate. This means we get an update every 5 seconds on our position. It is advertised with an estimated accuracy of 2,5 meter radius, but the actual accuracy of the GNSS module depends on the number of satellites which are currently connected. The radius of inaccuracy is furthermore augmented by the fact that we have to use two separate GNSS receivers with both a radius of uncertainty, shown in Figure 3.5. If drone GNSS accuracy is denoted by d and tracker accuracy by t, the total radius of inaccuracy (σ) we have to work with is:

$$V\delta(d) = \sigma_d^2$$

$$V\delta(t) = \sigma_t^2$$

This means the drone always has too keep a distance of at least σ to ensure no collisions between the tracker and drone can occur. If we want the drone to follow the tracker with a distance of δ, then the total safe distance will be:

$$V\delta(d-t) = \sigma_d^2 + \sigma_t^2$$

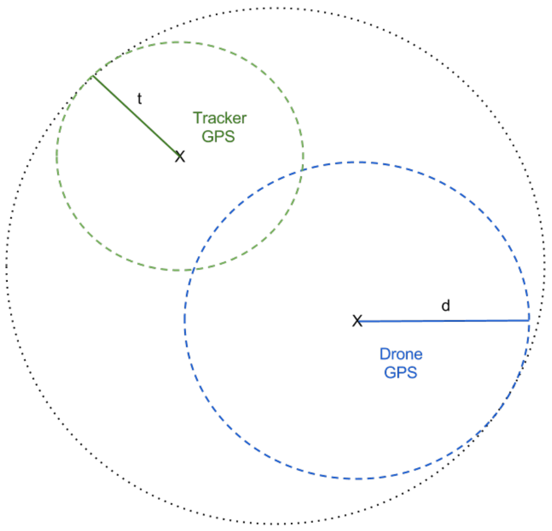


FIGURE 3.5: Shows how the area of uncertainty increases relative to the accuracy of the tracker GNSS and the drone GNSS.

The LEA-6H GPS receiver boasts of a *"High precision 2.5m accuracy"* [20]. To confirm it's accuracy, a small test was performed where the GNSS receiver was placed on the ground and gathered position fixes for a period of five minutes. Figure 3.6 shows the results of this test and tells us that even when the GNSS receiver is placed on the ground and is stationary under a clear sky, it will drift around with variable accuracy. The worst position fixes distinguishes themselves by 4526 centimeters from each other. This deviation will grow when the receiver is moving and with less optimal environmental conditions such as weather and satellite coverage.

3.5 Altimeter/Barometer

The MS5611 Barometric Pressure Sensor boasts of a 10cm precision for reading atmospheric pressure which can be used to calculate the altitude of the drone relative to the ground. The accuracy of the barometer depends on several external factors, air temperature, humidity and sudden changes in air pressure such as wind. The barometer offers accurate altitude readings at 50 Hz, but can be prone to the noisy factors mentioned

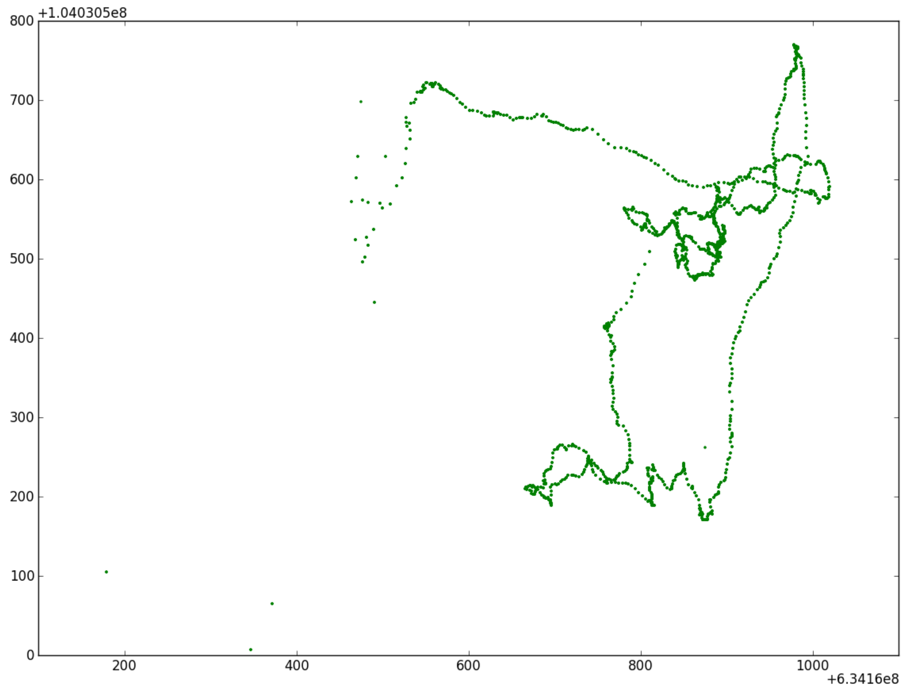


FIGURE 3.6: Latitude and longitude position fixes from a stationary GNSS receiver over a period of five minutes. Each green dot indicates a single position fix.

above. Figure 3.7 shows an example of how the estimated altitude of the altimeter can drift over time. The device was placed stationary on the ground and collected data over a period of 20 minutes. During the elapsed time, the position has drifted by almost 2 meters above the ground, even though the altimeter itself has not moved.

Using the difference between atmospheric pressure and temperature on the ground and in an arbitrary point in the air, we can calculate an altitude estimation above the ground. Observing changes in altitude over time allows us to estimate the altitude velocity as well. A device that estimates the altitude above the ground and estimates the velocity it changes it's altitude is called an Altimeter (altitude meter). This is done by calculating differences in pressure and temperature compared to when the barometer was calibrated on the ground.

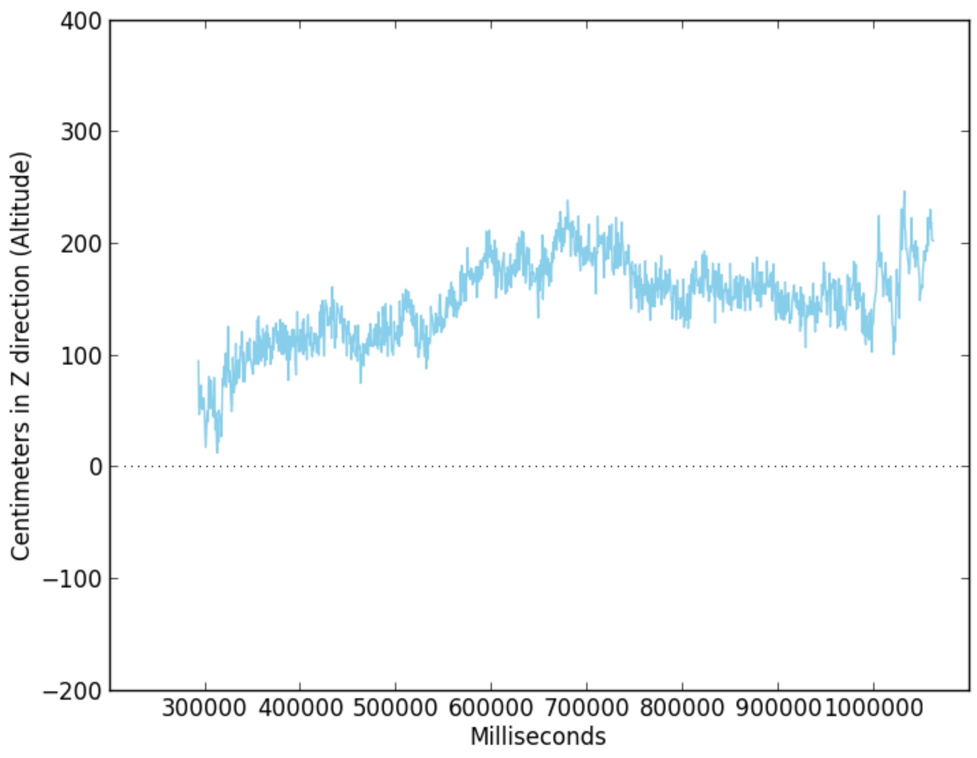


Figure 3.7: Graph showing the altitude position drift of the altimeter when the altimeter is placed stationary on the ground for 20 minutes.

3.6 Pixcuckoo

The Pixcuckoo is a Micro Controller Unit (MCU) is a central control unit that gathers data from all the sensors, runs any filtering algorithms, applies sensor fusion and finally tells the drone how to behave accordingly. The Pixcuckoo uses the Teensy 3.1 microcontroller (Figure 3.8), which uses the Cortex M4 microprocessor. This 32-bit microprocessor chip runs at 72 MHz and has 64 KB available memory, which means the filtering and sensor fusion method must have relatively low processing and memory complexity compared to what is possible to run on a desktop computer. Detailed technical specifications of the MCU can be found in Appendix A. The Pixcuckoo acts as a bridge between the tracker and the Pixhawk (see Section 3.7). It is also worth noting that the Cortex M4 processor does not have a floating point unit (FPU). This means it is very slow when calculating floating point numbers which need to be simulated with 32-bit fixed point integers.

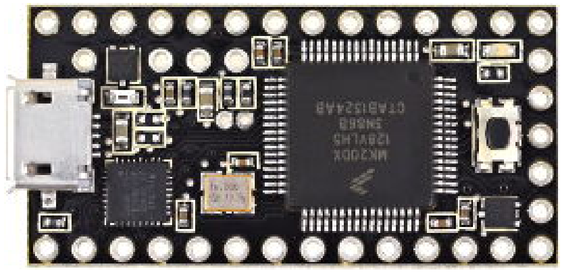


FIGURE 3.8: The Teensy 3.1 Microcontroller Unit used for both the Pixcuckoo and the Tracker. Read Appendix A for more detailed information about the Teensy 3.1 MCU and its Cortex M4 Microprocessor.

3.7 Pixhawk

The Pixhawk Autopilot Module (Figure 3.9) is a flight-controller that supports autonomous flight through instructions by an external source (in our case the Pixcuckoo). This device controls the pitch, roll and yaw along with the propellor spin speed automatically so that the drone stays stabilized in the air. The Pixhawk receives waypoint commands using North-East-Down (NED) coordinates in centimeters relative to where the drone did its takeoff. This provides a high-level control over the drone where the Pixcuckoo is only required to send waypoints on where we want to be relative to the position of the tracker. Appendix B contains detailed hardware description of the Pixhawk board.

Knowledge of our position and the position of the tracker is uncertain however and at the time when this thesis is written, there exist no Kalman Filter to improve the localization estimates. At the time of writing, the Pixhawk uses a *Complimentary* data fusion filter which uses weighted averages between GNSS and IMU to determine its North/East position and combines IMU with Barometer to determine altitude (see Section 4.8). Complimentary data fusion filter is susceptible to errors in the sensors such as variable GNSS coverage or glitches in the hardware since it uses static weights to how much it trusts each sensor. The Kalman filter solution will ideally compensate for this and use the error covariance to determine how reliable a specific sensor is at a given time.

Figure 3.9: The Pixhawk Autopilot Module controls low-level actuators to keep the drone in a specific NED position relative to home (point of takeoff) in addition to controlling the facing angles of the aircraft.

3.8 Tracker

The tracking device is the module attached to the person which the drone will follow. The tracker uses the same Teensy MCU (see Figure 3.8) hardware sensors as the drone (same GNSS, IMU and Barometer) for consistency and simplicity in implementing the Kalman Filter. Using the same hardware sensors means we only need to model the Kalman Filter and the noise covariance of each of the sensors only once and apply the solution to both the drone and the tracker.

The tracker will then transmit its filtered position wirelessly to the Pixcuckoo, which decides what to do with the data. An added benefit is off-loading some computations from the Pixcuckoo to the tracker. It also reduces the amount of data transmitted to the Pixcuckoo, since we only need to send the Kalman estimated position of the tracker instead of all the raw sensor data. Figure 3.10 shows a concept sketch of what the finished Tracker device might look like.

Appendix C contains a more detailed description and images of the Tracker Prototype used in the Phase 3 testing in Section 5.4.

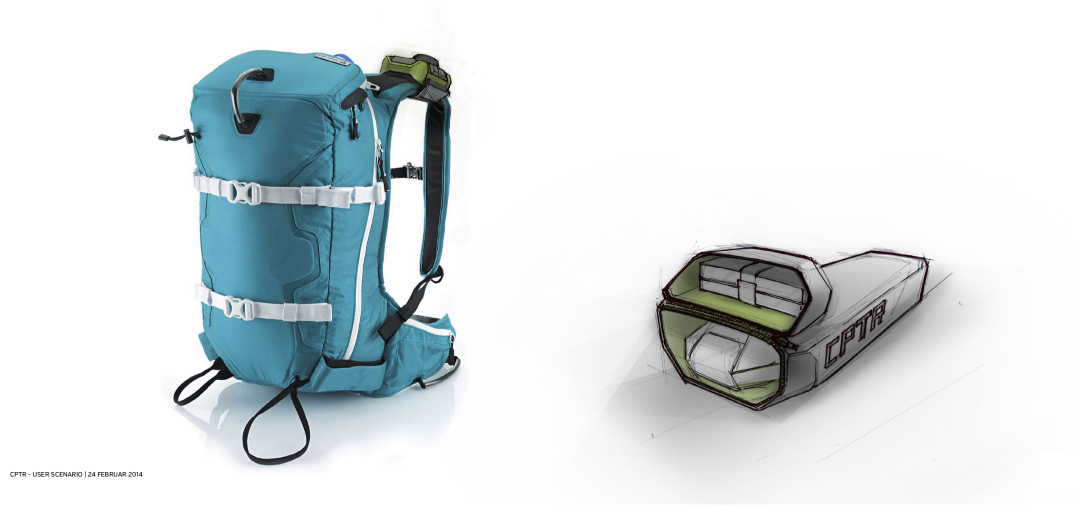


Figure 3.10: A concept sketch of a finished Tracker device (right) and how it might be attached to a piece of clothing such as a backpack (left). Used with permission from CPTR.

Chapter 4

Methodology

4.1 The Problem

In order to control the drone autonomously through software, the drone needs to know its position, velocity and alignment relative to its environment. This is achieved through various sensors that collect data over time. However, physical sensors tend to be inaccurate and riddled with noise, so using only the latest sensor data often leads to problems as described in section Section 3.2. One way of mitigating this problem is to use a wider range of sensory data and collectively use them to more accurately predict the current state. By themselves the sensors give an incomplete view of the drone state and the environment, therefore we are going to look into data fusion and filtering as a method to more accurately estimate the current state based on previous observed data.

4.2 Solution Hypothesis

Figure 4.1 describes how GPS, IMU and a Barometer is fused together with a fusion method to estimate a more accurate and reliable positioning of the drone. Note that we have a 3-dimensional localization, i.e we include altitude as well as position into the localization estimation.

One of the objectives in this thesis is to determine if the Kalman Filter is able to run on a micro-controller with constrained memory and computational power. Additionally the solution is compared to the existing sensor fusion solution, the Complimentary Filter.

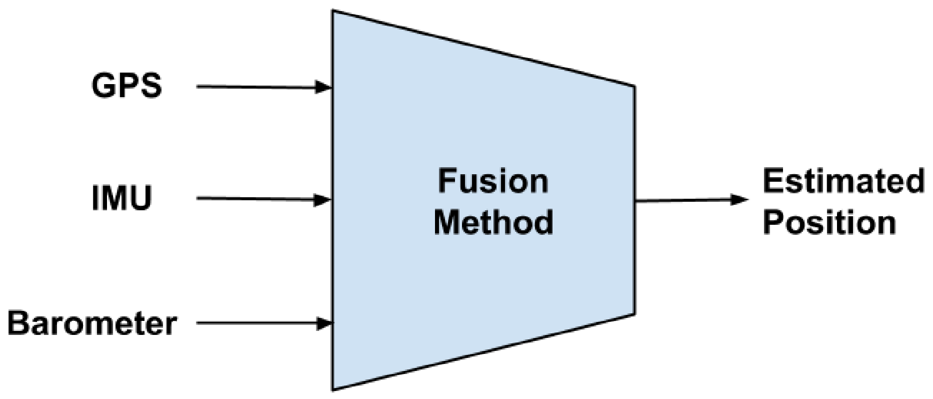


FIGURE 4.1: Figure showing how a more accurate localization position is estimated with the help of sensor fusion.

The goal is to determine which filter is the most effective for sensor fusion on an UAV. In addition to accuracy, we will also compare memory usage and computational complexity to determine which solution is best suited for data fusion in an UAV navigation system. The algorithm must improve sensor accuracy while having a low enough computation time to be able to run live in a micro-controller on board a drone using limited memory as described in Section 3.2, Appendix A and Appendix B.

4.3 Testing Strategy

Before testing the algorithms on-board the quadcopter drone, the algorithms are implemented on a computer without the strict memory and processing constraints (in a simulated environment). To make the tests deterministic and robust, the same recorded flight data will be used for all tests and compared to the existing Complimentary Filter solution described in Section 4.8. The tests also include timing and memory usage profiling to determine if a given algorithm is possible to run on-board on the drone in flight-time. If the simulations show that the linear Kalman Filter is not working properly, one of the more complex nonlinear filtering methods such as Extended Kalman Filtering or Unscented Kalman Filtering will need to be considered as alternative solutions.

4.3.1 Input and Output

This section gives an overview of what data the input and output represents in the filtering algorithms. We have two moving objects, the *tracker* and the *drone*. The tracker can have arbitrary movement while the drone follows the tracker and points its camera towards it at all times. The data available are the GNSS coordinates, barometer measurements and IMU data of both the tracker and the drone. This means in regular time intervals the drone receives Kalman Filtered updates on the *estimated* position and velocity of the person it is supposed to track. The drone has Kalman Filtered position and velocity estimates of itself through on-board sensors. This means input values are three different types of sensor data and the output is an more exact estimation of the position and velocity of both the tracker and the drone.

4.3.2 Data Sampling

The final data sampling rate is not just limited by the individual sensor update rate, but also the processing speed of the microprocessor and the data logging system. Transmitting data to the SD card is done through serial communication with a baudrate of 115200, giving us a theoretical maximum transmission rate of 14 Kb/s (excluding all overhead). Under flight tests, all the sensor readings are stored to a SD card at the given sampling rates shown in Table 4.1. These sampling rates were determined by the limits inherent by the sensors (e.g the LEA-6H GPS receives GPS fixes no faster than 5 times per second) and by testing stable sampling rates. If the sampling rate is too high, the hardware can be prone to data loss or communication errors. Too low sampling rate results in aliased data (as according to the Nyquist theorem), which introduces additional noise and can adversely affects sensor fusion algorithms.

Table 4.1: Sensor Sampling Rates

Component	**Sampling Rate**
IMU	100 Hz
Barometer	50 Hz
GNSS	5 Hz

4.4 Kalman Filter

The Kalman Filter, also known as Linear Quadratic Estimation (LQE), is described as a method for prediction of random signals, separating random signals from random noise and detection of signals in a known form (pulses, sinusoids) in the presence of random noise[7]. The Kalman Filter is an algorithm used for reducing the error generated by noise or gaps in data through estimations based on measuring data over time.

Figure 4.2 shows how the Kalman Filter can be used for fusing INS and GNSS. Using only the INS would result in very inaccurate localization, while using only GNSS would contain noisy coordinates and gaps of data. The gaps in GNSS coordinates are caused by the much lower frequency a GNSS receiver operates compared to how fast the robot moves. Combined together with the help of Kalman Filtering, the figure shows how the algorithm improves the robot localization and filters away noisy GPS coordinates.

A huge advantage of the Kalman Filter is that it is not a growing-memory filter. This means it does not require more memory as it runs over time, which is important because of the limited memory given to us by the on-board hardware in the drone.

4.4.1 Kalman Assumptions

For the Kalman Filter to work correctly, the algorithm has to make some assumptions about the state dynamics and the observational model:

1. That the underlying system is linear (i.e the output is directly proportional to the input),
2. The covariance of the noise can be estimated,
3. That the noise is gaussian,

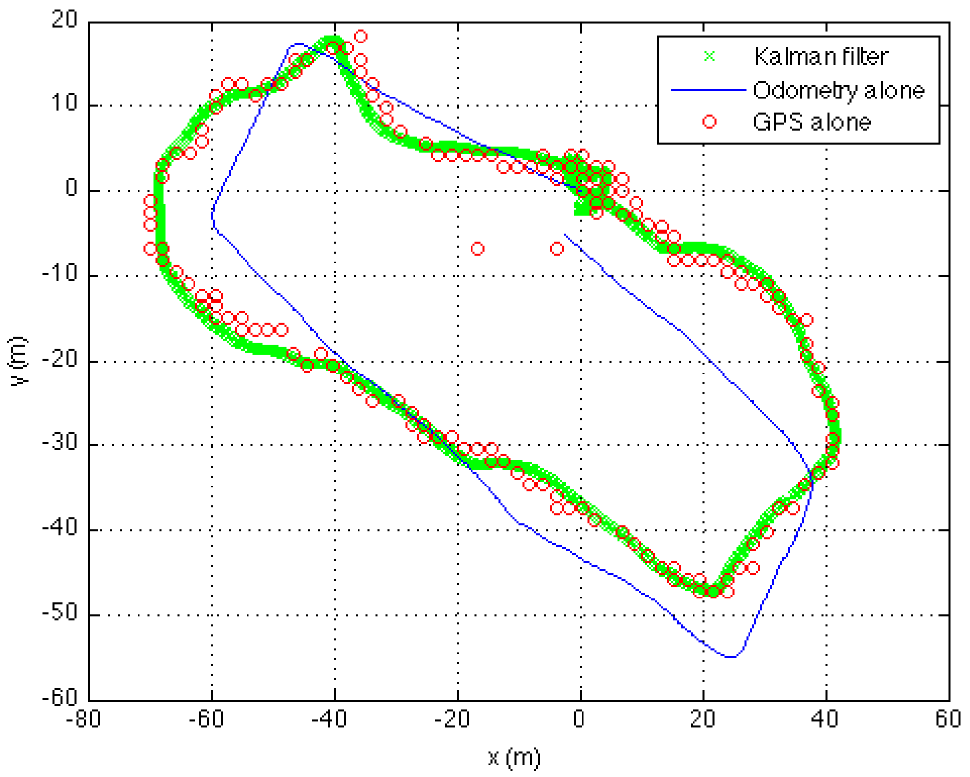


FIGURE 4.2: A simulation using real robot sensor data from an open loop test run. The Kalman filter fuses the GPS and odometry data. Image source: http://www.cheapscience.com/2010/04/kalman-filtering.html

4. The error of an estimation can be observed as a random variable which has a mean value and a variance.

Given these assumptions, the Kalman Filter produces a statistically optimal estimate of the underlying state by recursively operating on a stream of noisy input data. This means it gives us the estimate which minimizes this variance of the estimation error. The algorithm generates the best estimations given our assumptions, but does not necessarily always estimate perfect ground truth.

4.4.2 Kalman Filter Algorithm

The Kalman Filter equations for modeling a problem are as follows, where $\boldsymbol{x}_{t+1}$ is our signal value and $\boldsymbol{z}_{t+1}$ is the measurement noise at timestep $t+1$. A summary of each step in the algorithm and a descriptive list of all Kalman variables can be found in Appendix D. The Kalman Filter is based on linear dynamic systems which are modeled on a Markov Chain infused with Gaussian noise. At each time interval a linear operation is applied to the current state to generate a new state (with noise mixed in).

$$\boldsymbol{x}_{t+1} = F\boldsymbol{x}_t + B\boldsymbol{u}_{t+1} + \boldsymbol{Q}_t \tag{4.1}$$

$$\boldsymbol{z}_{t+1} = \boldsymbol{H}\boldsymbol{x}_{t+1} + \boldsymbol{R}_{t+1} \tag{4.2}$$

Here $\boldsymbol{x}_{t+1}$ denotes our state estimates, $F_{\boldsymbol{x}_t}$ is the state transition model, $\boldsymbol{u}_{t+1}$ is the control signal and $\boldsymbol{Q}_t$ is the process noise. Subscript t is the current discrete timestep (e.g t=1ms, t+1=2ms, etc.). Equation 4.1 evaluates each $\boldsymbol{x}_{t+1}$ at each timestep. Equation 4.2 calculates the measurement of the true state $\boldsymbol{z}_{t+1}$ which is a linear combination of the signal value and the observation noise $\boldsymbol{R}_t$. Q_t and R_{t+1} are both considered to be Gaussian. $\boldsymbol{H}$ is the observation model which defines how each observation affects a given state estimate and B is the control signal model which defines how the control signal affects the state estimates.

The process noise $\boldsymbol{Q}_t$ and observation noise $\boldsymbol{R}_{t+1}$ are assumed statistically independent. The standard deviation of these two noise functions have to be estimated, but in real world problems no signal is pure Gaussian. We can however assume it with some approximation. The Kalman Filter Algorithm will after a period of time approach a correct estimation as long as the model is completely consistent with what's actually happening, even if the Gaussian noise parameters are poorly estimated[21]. However, better estimations of the noise parameters will cause the Kalman Filter to converge faster.

The algorithm works in two steps, prediction and update which are applied at each timestep:

1. Prediction: In the prediction step, the Kalman filter produces estimates of the current state variables, along with their uncertainties.
2. Measurement Update: Once the outcome of the next measurement is observed (including some amount of noise), the estimates in the prediction step are updated using a weighted average (estimates with higher certainty have higher weight).

These two steps are accomplished through two sets of equations. At the first timestep ($t = 0$), we provide the algorithm (prediction step) some initial estimates, known as the *initial conditions*. In future timesteps the algorithm uses the results from the Measurement Update step for estimates.

Prediction

1. Project the state ahead

$$\boldsymbol{x'}_{t+1|t} = F\boldsymbol{x'}_{t|t} + \boldsymbol{Q} + B\boldsymbol{u}_t \tag{4.3}$$

Calculate the rough estimate of $\boldsymbol{x'}$ at timestep $\boldsymbol{t+1}$. This value is updated in the Measurement Update step of the algorithm.

2. Project the error covariance ahead

$$\boldsymbol{P'}_{t+1|t} = F\boldsymbol{P}_{t|t}\boldsymbol{F}^T + \boldsymbol{R} \tag{4.4}$$

Calculate a measure of the estimated accuracy of the state estimate which is used for computing the Kalman Gain in Equation 4.5. P' contains the estimated error covariance. This estimate is refined and improved in step 3 of the Measurement Update after the observation is known. $\boldsymbol{Q}$ denotes the noise in the environment (process noise) and $\boldsymbol{R}$ denotes the noise in the observations.

Measurement Update

1. Compute Optimal Kalman Gain

$$\boldsymbol{K}_t = \boldsymbol{P'}_{t|t-1}\boldsymbol{H}^T(\boldsymbol{H}\boldsymbol{P'}_{t|t-1}\boldsymbol{H}^T + \boldsymbol{R})^{-1} \tag{4.5}$$

The Kalman gain is a function of the relative certainty of the measurements and the current state estimate, so that it works as a bias towards either state estimation or measurements. With a high gain the filter places more weight on measurements while a low gain means the filter follows the model predictions more closely, smoothing out noise but decreasing responsiveness. The Kalman Gain is used in the next equation of the Measurement Update step.

2. Update the estimate via $\boldsymbol{z}_t$

$$\boldsymbol{x}_{t|t} = \boldsymbol{x'}_{t|t-1} + \boldsymbol{K}_t(\boldsymbol{z}_t - \boldsymbol{H}\boldsymbol{x}_{t|t-1}) \tag{4.6}$$

We can now calculate the estimation we are trying to find which is $\boldsymbol{x}_t$. This value is the estimation of $\boldsymbol{x}$ at timestep $\boldsymbol{t}$ (this is the improved estimation value we are going to use and is the output of the algorithm at this timestep).

3. Update the error covariance

$$\boldsymbol{P}_{t|t} = (\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{H})\boldsymbol{P'}_{t|t-1} \tag{4.7}$$

We now recalculate the error covariance $\boldsymbol{P}_t$ which is used in the next timestep $(t+1)$ in the Prediction step 2 (Equation 4.4). $\boldsymbol{I}$ is the Identity matrix.

4.5 Modeling the Kalman Filter

The first step when developing a Kalman Filter for sensor fusion is modeling the problem to fit in the Kalman Model (see Section 4.4.2). We want to know our position in NED centimeter coordinates relative to our takeoff position (X for East, Y for North and Z for Down).

We observe two different positions X, Y and their velocities V_x, V_y from the GNSS receiver and INS. For the altitude Z and altitude velocity V_z we observe three different positions and velocity estimates from the GNSS receiver, INS and the Altimeter.

This means each X, Y and Z we have two different state estimates for their position and their velocities V_x, V_y and V_z. Each of these coordinate axes are independent of each other (e.g the altitude velocity V_z does not affect the position of X) which means that we can run a Kalman Filter for each axis independent from each other in parallel. Figure 4.3 gives an overview of how the model is structured. Three different sources of sensors giving noisy Position and Velocity estimates of the aircraft at different rates and accuracy.

Ideally we could do the calculations concurrently, but this is impossible since the MCU hardware lacks support for multi-threading or context switching. It is also a single-core CPU which means the performance benefits would not be large.

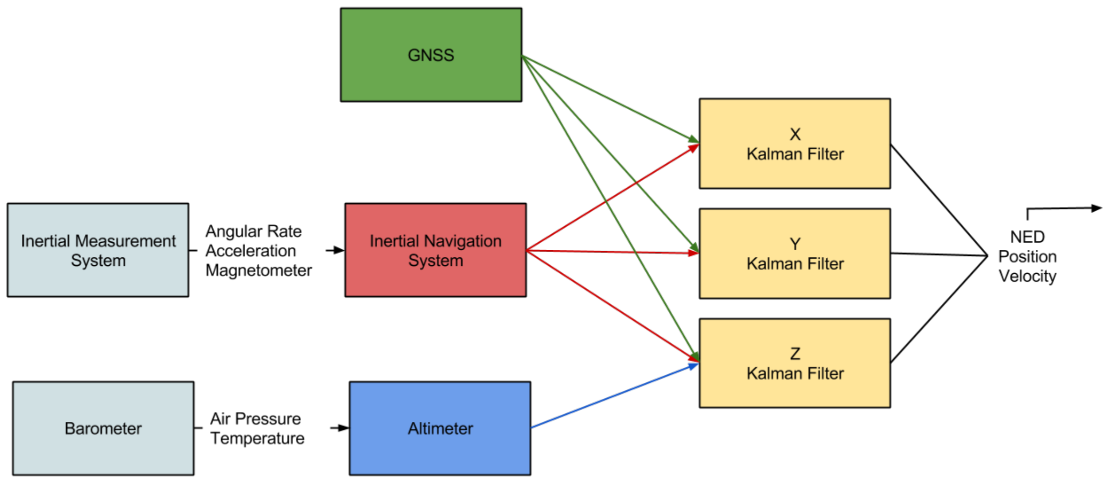


FIGURE 4.3: A overview of how three different Kalman filters run independently to improve the NED position estimates of the aircraft.

4.5.1 State Transition Model

Since we only need to observe velocity and position the transition model F_t is fairly simple as described in Equation 4.8 where Δt is the number of seconds passed since last Kalman iteration.

$$F = \begin{bmatrix} 1 & \Delta t \\ 0 & 1 \end{bmatrix} \tag{4.8}$$

We do not observe acceleration or attitude (these are handled internally by the INS and AHRS respectively, see Section 4.7), the modeled system is linear which is one of the requirements for the Kalman theory to work. The simplifying assumption that acceleration is constant within a given timestep, allows us to safely remove the control signal input U_{t-1} and control signal model B from Equation 4.1 which simplifies it to Equation 4.9:

$$\boldsymbol{x}_{t+1} = F\boldsymbol{x}_t + \boldsymbol{Q}_t \tag{4.9}$$

4.5.2 Observation Model

We have two different observation models, one for the horizontal X and Y positions and velocities shown in Equation 4.10 and one for the vertical Z position and velocity shown in Equation 4.11. The left-hand column for each of the matrices denotes what kind of observation each of the rows represent, while the top row describes whether the given column represents position or velocity.

$$H_X = H_Y = \begin{array}{l} \textbf{X and Y Observation Model} \\ \begin{array}{lc} \text{GNSS Position} & \\ \text{INS Position} & \\ \text{GNSS Velocity} & \\ \text{INS Velocity} & \end{array} \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \end{array} \tag{4.10}$$

$$H_Z = \begin{array}{l} \textbf{Z Observation Model} \\ \begin{array}{l} \text{GNSS Position} \\ \text{INS Position} \\ \text{Altimeter Position} \\ \text{GNSS Velocity} \\ \text{INS Velocity} \\ \text{Altimeter Velocity} \end{array} \begin{pmatrix} 1 & 0 \\ 1 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 1 \\ 0 & 1 \end{pmatrix} \end{array} \tag{4.11}$$

4.5.3 Observation Noise

The observation noise covariance matrix $\boldsymbol{R_t}$ is non-constant as the accuracy and noise of the sensors varies over time. This is because the GNSS covariance is determined using Geometric Dilution of Precision, which is a satellite navigation term that describes the error in the positional measurement precision. More precisely the value we use for the X and Y Kalman Filters is the HDOP (Horizontal Dilution of Precision) and for the Z Kalman Filter the VDOP is used instead (Vertical Dilution of Precision). Kong et al. [22] have shown that using HDOP and VDOP in estimating the observation covariance values is a simple and accurate method of determining the covariance for the GNSS observations, i.e σ for HDOP and λ for VDOP. Note that when the GNSS receiver does not have a 2D position fix, then σ will be so large that the GNSS observation data will be ignored by the Kalman Filter. The same happens for λ if the GNSS receiver does not have a 3D position fix. This distinction allows the Kalman Filter to still use horizontal positioning data even though vertical is not available at the moment.

Because the HDOP and VDOP can change over time, we have to update the observation covariance matrix at each timestep with the values specified in Equation 4.12 where σ denotes the GNSS HDOP, λ is the GNSS VDOP, κ is the covariance of the position estimate in the INS, τ denotes the covariance between the estimated GNSS velocity and INS velocity. Equation 4.12 is the observation noise covariance matrix for both X and Y.

$$R_x = R_y = \begin{bmatrix} \sigma & 0 & 0 & 0 \\ 0 & \kappa & 0 & 0 \\ 0 & 0 & \sigma & 0 \\ 0 & 0 & 0 & \tau \end{bmatrix} \tag{4.12}$$

The exact observation noise covariances are calculated in Section 5.3.4. For now, values are assigned using empirical testing and Kalman Filter simulations on the PC which gave the observation noise matrix described in Equation 4.13.

$$R_x = R_y = \begin{bmatrix} \sigma & 0 & 0 & 0 \\ 0 & 125.0 & 0 & 0 \\ 0 & 0 & \sigma & 0 \\ 0 & 0 & 0 & 150.0 \end{bmatrix} \tag{4.13}$$

The observation covariance for Z includes the Altimeter and it's observation covariance matrix is shown in Equation 4.14.

$$R_z = \begin{bmatrix} \lambda & 0 & 0 & 0 & 0 & 0 \\ 0 & 400 & 0 & 0 & 0 & 0 \\ 0 & 0 & 500 & 0 & 0 & 0 \\ 0 & 0 & 0 & \lambda & 0 & 0 \\ 0 & 0 & 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0 & 0 & 300 \end{bmatrix} \tag{4.14}$$

When the GNSS receiver does not have a 3D fix (i.e can estimate altitude and altitude velocity), then σ will be so large that GNSS observations are ignored in the rest of the calculations.

It is important to note that these covariance matrices are incorrect in the assumption that they are truly diagonal. There exists a correlation between the INS and GNSS for horizontal position estimates, because of the periodic GNSS position fix correction the INS has to prevent it from diverging. This simplification allows us to make some calculation simplifications later on, but will also reduce the accuracy of the Kalman Filter because we are feeding it wrong information and telling it that there is no correlation between GNSS and the INS. The INS altitude estimation is corrected by Altimeter readings which means there is also a correlation between INS and Altimeter altitude accuracy.

4.5.4 Process Noise

The process noise covariance matrix $\boldsymbol{Q}$ represents any environmental disturbances that affect the system dynamics, such engine drift, wind, magnetic disturbances or voltage drop by poor batteries, etc. This means that process noise depends on environmental factors, which makes it difficult to determine a universal and specific value. Higher process noise leads to the Kalman Filter making faster but possibly more noisy estimations, while low process noise makes estimations that are smoothed and converge more slowly to the correct value.

The method used to determine Process Noise in this thesis is by making the drone hover airborne in one place. Using Equation 4.16 the covariance noise is calculated. Then the observaton noise measured from Section 5.3.4 is subtracted from this value and the resulting value should reperesent the process noise. For best results, the drone should collect data in varying weather to best model the noise introduced by different weather conditions.

The tracker operates in a different environment and platform than the drone, which means we cannot model its process noise in the same way. Fot the *tracker*, any environment noise sources are suitably modeled in the Observation Noise Covariance matrices described in Section 4.5.3. Therefore the Process Noise modeling is ignored on the tracker to reduce unnecessary computations. This allows us to simplify Equation 4.4 to Equation 4.15 by removing the Q term:

$$\boldsymbol{P'}_{t+1|t} = F\boldsymbol{P}_{t|t}\boldsymbol{F}^T \tag{4.15}$$

This simplification is only made on the tracker. For the drone, the process noise is still included in the equation.

4.5.5 Calculating Covariance

For calculating the other covariances, we need some sample data to calculate covariance values from, which is achieved by observing the correlation between two sets of data. The two-pass covariance algorithm is described in Equation 4.16, which is used to calculate to covariance between two sets of data ω and β.

$$\text{Cov}(\omega, \beta) = \frac{\sum_{i=1}^{n} \omega_i \beta_i - (\sum_{i=1}^{n} \omega_i)(\sum_{i=1}^{n} \beta_i)/n}{n}. \tag{4.16}$$

This method requires sample data to calculate the variance from. In the first testing phase empirical estimates will be used instead, and more exact covariance values will be calculated and used in testing Phase 2 using Equation 4.16. This will show if and how the more correctly modelled covariances affect the Kalman Filter behaviour when it runs on the same observation data. The empirical covariance values in testing Phase 1 were determined by assigning some random values and observing its effect on the behaviour of the Kalman Filter, using the GNSS covariance as a reference value for the other sensors.

4.6 Driver and Firmware Development

The entire system contains many different sensors and components. This is especially true for the Tracker device which is an entire new piece of technology that needs to be put together. Among the new things that are required to be developed, are driver interfaces, the firmware creating a communication network between each component, the main CPU that runs the Kalman Filter and a wireless communication with an entirely different system. Different scripts and tools for debugging and translating a raw data stream with flight test data also needed implementation.

The Mavlink[1] communication protocol was used for intercomponent communication; for example between Tracker and Pixcuckoo and between Pixhawk and Pixcuckoo. This decision was made mainly because the Pixhawk autopilot already uses Mavlink protocol for communication, and also because the Mavlink standard provides checksums for data integrity and efficient cross-platform data packing so that data works seamlessly between different processors. This allows a 8-bit big-endian processor to communicate with a 32-bit small-endian processor without needing to worry about the low level translation of bits.

4.7 Designing the Inertial Navigation System

The IMU sensor gives us accelerations, gyroscopic (angular velocity) and magnetic readings in three different directions. Accelerations can be integrated to get velocites while gyroscopes and magnetic readings can be combined to give us the angles of the aircraft. Through the laws of physics we can integrate accelerations and angular velocities to determine velocity of the aircraft. By observing the acceleration changes over time we can determine the position of the aircraft relative to a position earlier in time by using the IMU. This system is called an Inertial Navigation System (INS).

Note the difference between the terms *speed* and *velocity*. *Speed* is a scalar quantity that tells us how fast we are moving, while *velocity* is a vector quantity and tells us the rate in which we are changing our position. This means that velocity is directional relative to our current position and is more useful when predicting movement over time.

[1]The specifications for the MAVLink Micro Air Vehicle Communication Protocol can be found here: http://qgroundcontrol.org/mavlink/start

Because of the complexity inherent in the three dimensions, we will need to do some additional calculations if we want to know the velocities in each of the three directions (V_n, V_e, V_d) relative to the ground rather than the aircraft. This North East Down (NED) coordinate system is commonly used in aviation to describe a 3-dimensional position relative to the aircrafts current position (in our case origo is the position of takeoff). Calculating the velocities is difficult because the drone has rotational angles in three degrees of freedom: Roll, Pitch and Yaw (ϕ, θ and ψ). These three combined is called the *attitude* of the drone. Depending on the current attitude of the drone, the values given to us by the accelerometers will mean different things. This is because the accelerometers give us measurements of the gravitational pull of the earth relative to the aircraft minus the acceleration of the aircraft itself. For example when we are flying sideways (e.g with 45 degrees roll), the accelerometers will tell us we are flying sideways in the X direction and the up and down acceleration (Z) will be reduced.

4.7.1 DCM Theory

To solve this problem we introduce the direction-cosine-matrix (DCM) as defined by Luukkonen [3], Premerlani and Bizard [23]. The DCM transformation gives us a velocity relative to the surface of the earth instead of those relative to the aircraft, which can then be fused with GNSS and Altimeter measurements which already are relative to the earth's surface. The DCM matrix is defined in Equation 4.17.

$$\begin{bmatrix} \cos(\theta)\cos(\psi) & \cos(\theta)\sin(\psi) & -\sin(\theta) \\ \sin(\theta)\sin(\phi)\cos(\psi)-\sin(\psi)\cos(\theta) & \sin(\psi)\sin(\theta)\sin(\phi)+\cos(\psi)\cos(\phi) & \sin(\phi)\cos(\theta) \\ \sin(\theta)\cos(\phi)\cos(\psi)+\sin(\psi)\sin(\phi) & \sin(\phi)\sin(\theta)\cos(\phi)-\cos(\psi)\sin(\theta) & \cos(\phi)\cos(\theta) \end{bmatrix} \tag{4.17}$$

Given the three accelerations A_x, A_y, A_z given to us by the IMU, we can transform these with the DCM matrix which gives us the acceleration on the earth frame along North (A_n), East (A_e) and Downward velocity (A_d) (shown in Equation 4.18).

$$\begin{bmatrix} A_n \\ A_e \\ A_d \end{bmatrix} = DCM^T \begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix} \tag{4.18}$$

4.7.2 Calculating Velocity and Position

We can now use the Equation of Motion to calculate our position X, Y and Z relative to the earth frame (NED) in discrete timesteps each time measurements from the IMU is received. This means integrating the accelerations A_x, A_y, A_z to give us an estimation for our velocity V_x, V_y, V_z. Since the acceleration is assumed to be constant in a given timestep t, we integrate the velocites V_n, V_e, V_d analytically rather than numerically as shown in Equation 4.19. The estimated velocities also have to be updated each timestep as shown in Equation 4.20. P denotes position, V velocity and A acceleration in timestep t.

$$P_t = P_{t-1} + V_t + \frac{A_t \Delta t^2}{2} \tag{4.19}$$

$$V_{t+1} = V_t + A_t \Delta t \tag{4.20}$$

For the altitude position calculation Z, we also have to compensate for the earth's gravity by subtracting the constant G and inverting it to get the upward velocity. The resulting Equation 4.21 shows how we can calculate our position in the timestep given by t, where X is our north-south position, Y is the west-east position and Z is the up-down (altitude).

$$\begin{bmatrix} X_t \\ Y_t \\ Z_t \end{bmatrix} = \begin{bmatrix} X_{t-1} + V_n + \frac{A_n \Delta t^2}{2} \\ Y_{t-1} + V_e + \frac{A_e \Delta t^2}{2} \\ Z_{t-1} + V_d - \frac{(A_d - G)\Delta t^2}{2} \end{bmatrix} \tag{4.21}$$

4.7.3 Calculating Dead Reckoning

The process of calculating our position through an INS relative to a position fix earlier in time is called Dead Reckoning. This method allows us to determine our position without the need of external navigation references and gives us a reliable navigation capability under virtually any conditions. The localization provided by the INS is prone to several sources of errors though:

1. Errors in the measurements provided by the IMU,
2. Errors in the discrete integration process performed by a computer,
3. Errors caused by floating point rounding,
4. The inherent and unbounded error growth over time caused by yaw drift

Though there are numerous different sources of error and inaccuracy, only the 4th point in the enumeration above is unbounded and will grow over time. This can be improved by using a periodic update from a GNSS fix to correct the accumulated errors caused by the INS.

4.7.4 DCM Algorithm and the Tracker

The DCM algorithm works very well as long as the IMU is in a fixed position relative to the earth frame. This is very difficult requirement to fulfill for the tracker, where the device can be attached to different locations and in different orientations. Furthermore, vibrations and shaking is much more likely when the IMU is attached to a person on the ground rather than a hovering quadcopter. This makes the sensor much more noisy and difficult to gain any useful data of. Therefore the DCM algorithm is not applicable to the IMU in the tracker.

Designing a new algorithm or method for a new INS system on the tracker is out of the scope of this thesis and is complex enough to become an entire masters thesis by itself. Therefore, the IMU data is not used on the tracker side in the tests. The Kalman models remain the same, except that there are no INS observations. The Kalman Filter therefore only fuses Barometer and GNSS data on the Tracker side for vertical position estimates. For horizontal position estimates from the GNSS, the Kalman Filter acts as a smoothing agent and reduces the noise of "jumping" GNSS position fixes. It also combines the velocity estimates from both historical data and from GNSS data to improve the horizontal position estimates.

4.8 Complimentary Filter

The Complimentary Filter is an alternative and much simpler sensor fusion algorithm compared to the Kalman Filter[8]. It's simpler both in terms of implementation and computation complexity. The filter uses a weighted average solution to fuse the output from different sensor sources into a single output. Each sensor output is weighted with a number between 0 (completely unreliable) and 1 (only reliable output) where the sum of the weight of all sensors is 1. Equation 4.22 gives the fused sensor value z where α indicates the weight of the different sensors F and G at timestep t.

$$z_t = \alpha \times F_t + (1 - \alpha) \times G_t \tag{4.22}$$

Equation 4.22 shows how to fuse two sensors, but the filter can be expanded to fuse any amount of sensors as shown in Equation 4.23. Equation 4.23 fuses n different sensors where the sum of all the sensor weights given by α_i is 1.

$$z_t = \sum_{i=1}^{n} \alpha_i \times F_{i_t} \tag{4.23}$$

The weights in the algorithm can be either static (pre-determined and unchanging) or dynamic (adapts and changes while running). The original sensor fusion solution on the drone uses Complimentary Filtering to fuse GNSS, INS and Altimeter data for localization with dynamic weights where the weights change depending on how reliable the GNSS fixes are. For example if no satellites are visible, the GNSS is weighted zero and therefore more weight is given to the INS and Altimeter readings.

Advantages of the Complimentary Filter is it's ease of implementation and low computational cost. The latter is one of the main reasons it is popular for using with MCUs which have historically limited computational performance. Recent developments in

MCU technology has made better processing chips available at equally low cost, allowing making complex algorithms such as Kalman Filtering a viable option. The Complimentary Filter does have certain drawbacks however. One of the major drawbacks is that it has no theory of optimality to back it up. Tweaking, testing and figuring out the weights is entirely up to the user and even then does not guarantee that the filter will work good in all situations. For example when one of the sensors stops working or begins to give only noisy data, the naive filter will continue to fuse this insensible data with the other sensors that still work correctly.

In conclusion the Complimentary Filter uses the incorrect assumption that there exists a constant α that works good for all the different sensors, in all environments and is unchanging in any point of time. This is not true because GNSS accuracy varies with weather, satellite reception and the fact that sensors can fail due to hardware errors. The Kalman Filter solves this by calculating these α values dynamically at run time based on the historical observation of the different observation data.

Chapter 5

Results

This chapter describes all the test results from a simulated environment on a PC (Section 5.2), a second iteration (Section 5.3) where data from the first Phase 1 is used to tweak, improve and optimize the Kalman Filter solution and finally a third iteration (Section 5.4) where the improved filter will be tested in-flight which the Tracker will use to transmit its location to the drone.

5.1 Experimental Procedure

All graphs in this chapter use the same color encoding as described here (unless noted otherwise):

Red INS

Green GNSS

Cyan Barometer

Purple Fused data by Complimentary Filter

Black Fused data by Kalman Filter

Data collection is done by storing flight data on a SD card while the drone is being steered by a pilot. The drone has to be controlled manually by a pilot since it's autonomous waypoint system is dependent on localization, something which we are trying to accomplish through the Kalman Filter in the first place! All tests were performed in the same area (an open football field) which means that the drone does takeoff and landing on the same altitude (which should be altitude zero). Note that it is possible for the drone to estimate negative altitude, which means that it is in a lower position than it was at takeoff. This is possible when the drone begins on the top of a slope for example.

There are five different sets of test data to evaluate the performance of the Kalman Filter implementation:

A - **Straight line (Figure 5.1a)**
In this test the drone flies in a straight line back and forth with the same altitude. It will land on the same point of takeoff.

B - **Circle (Figure 5.1b)**
The pilot flies the drone in a circular pattern in the same altitude. The drone will land and takeoff at the same point.

C - **Random (Figure 5.1c)**
The flight path in this test is random flight patterns and altitude gains at the pilots discretion.

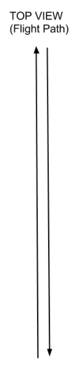
D - **Hover (Figure 5.1d)**
In this test the drone hovers in a single location above takeoff and after a while lands on the same location.

E - **Altitude (Figure 5.1e)**
The drone flies in a straight line while gaining altitude. When the drone is half-way to the end it will begin to decend instead.

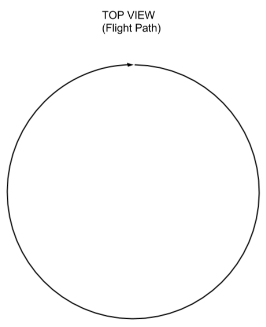
The implementation on the PC uses the same source code that will run on the MCU, though with a different compiler. Both compilers use the optimize for size compilation flag (Os).

5.2 Phase 1 Testing: Off-Line (pre-optimization)

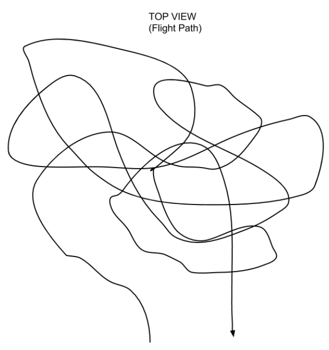
The first tests phase uses real flight data collected under flight and is run off-line on a PC.



(A) Straight Line



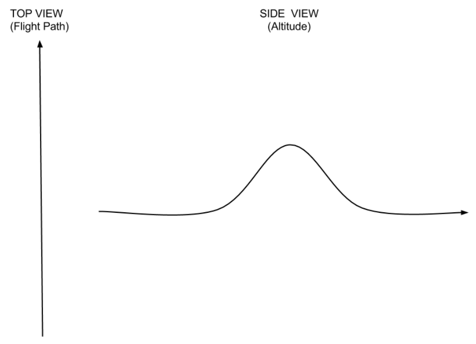
(B) Circle



(C) Random

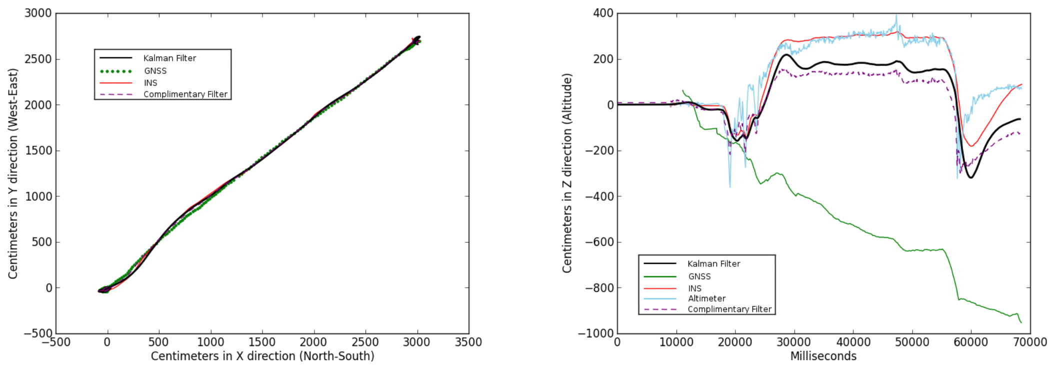
TOP VIEW
(Flight Path)

(D) Hover



(E) Altitude

FIGURE 5.1: The different flight paths used for collecting data for the simulation tests



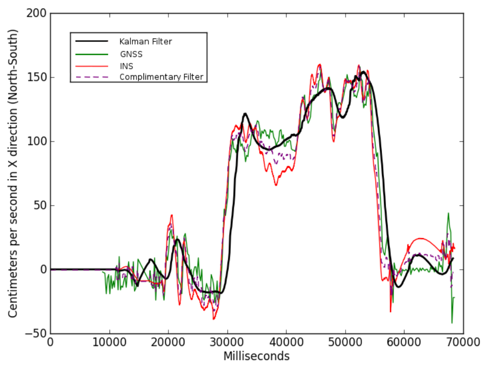
(A) X and Y position relative to takeoff (0, 0) (B) Altitude above takeoff observed over time

FIGURE 5.2: Position estimates for STRAIGHT LINE test

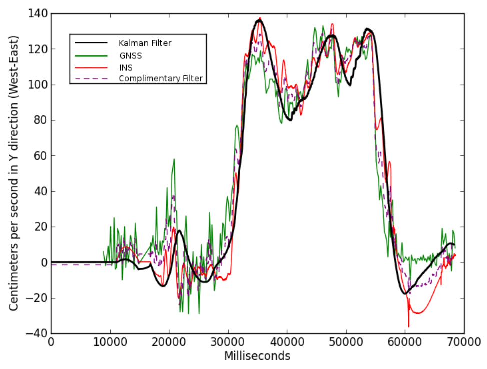
5.2.1 Straight Line

The velocity estimates for the Straight Line flight test is shown in Figure 5.3. The Kalman Filtered estimate (black line) is significantly smoother and less jagged than the other velocity estimates of the sensors or the Complimentary filtered solution. A benefit of less noisy velocity estimate means that it is more reliable and therefore can also contribute to a more accurate position estimate. This is especially true for GNSS position fixes which can glitch or completely lose satellite connection. If this happens, we can instead estimate our current location using the velocity estimates instead. The Complimentary filter position estimates for altitude is negatively affected by the noisy altimeter velocity estimates as shown in Figure 5.3c. Near takeoff and landing the altimeter have some major spikes reaching velocities almost 1000 cm/s which contradicts sensor readings from the INS and GNSS. While the Kalman filtered estimate is not affected by the spike, the Complimentary estimates also spikes whenever the altimeter spikes.

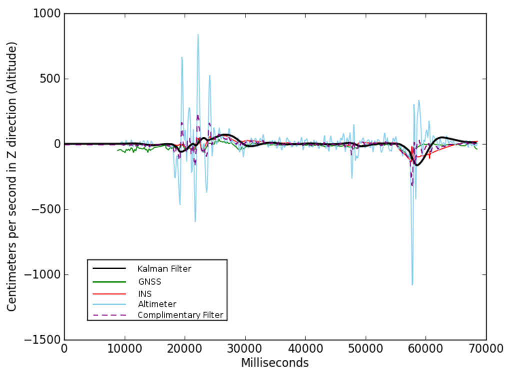
If we observe the altitude above ground in the Straight Line test (Figure 5.2b), we can see that the GNSS altitude position fix drifts away completely and contains little or no useful information and in fact only introduces noise to the sensor fusion process. The cause of this drift can be traced to the calibration process of the GNSS relative NED altitude estimation. On takeoff, the current altitude estimation of the GNSS is stored and use as an negative offset to give us our position relative to takeoff. This means we now have NED coordinates same as the other sensors. The drifting problem arises as the GNSS receives additional satellites and improves it accuracy, it will correct



(A) Straight Line: X-Velocity



(B) Straight Line: Y-Velocity



(C) Straight Line: Z-Velocity

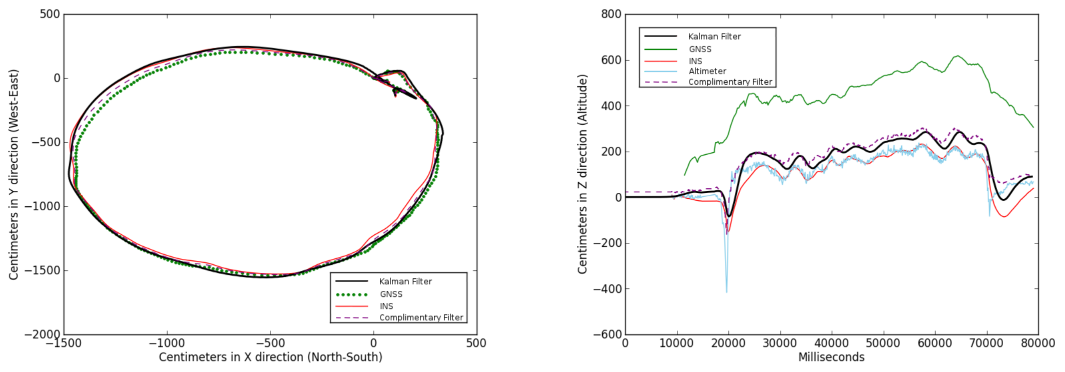
FIGURE 5.3: Velocity estimates for STRAIGHT LINE test observed over time

itself and improve it's altitude estimation from what it initially estimated on takeoff. An attempt to solve this problem is described in the next section (Section 5.3) where in-flight calibration is used to reduce the error from GNSS altitude fixes.

5.2.2 Circle

The circle test shows that the X and Y positions produced by the sensors are very stable and similar (Figure 5.4a). Low noise and discrepancy show that the estimated position is generally correct and that the Kalman Filtered position coincide with the other sensors and the Complimentary solution.

Figure 5.4b describes the Z-position (altitude above takeoff) and shows more discrepancy between the sensors. It shows a good example where the complimentary filter fails to work correctly and both over and underestimates the altitude. According to the complimentary filtered position, the drone is back on the ground (0 cm altitude) in



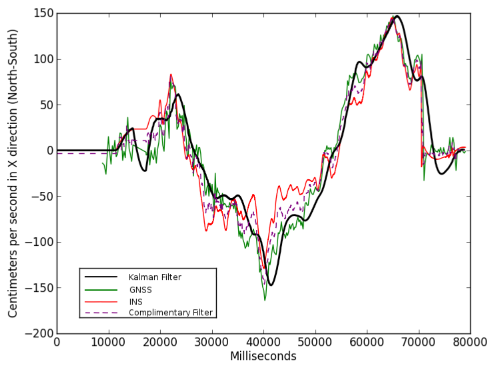
(A) X and Y position relative to takeoff (0, 0) (B) Altitude above takeoff observed over time

Figure 5.4: Position estimates for CIRCLE test

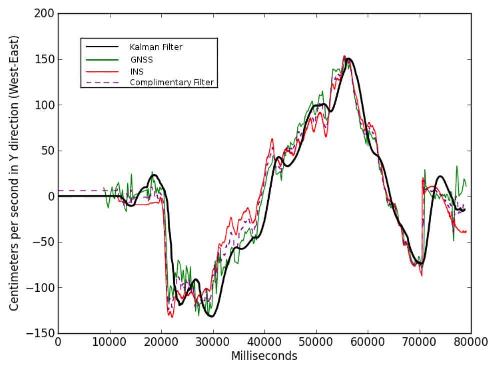
the middle of the flight test and is almost 1 meter above the ground when the drone actually lands. The Kalman Filtered position however keeps a stable 1-2 meter altitude and correctly estimates the position to be near 0 cm altitude after landing. Furthermore the Kalman Filtered position is least affected by the noisy spike caused by takeoff (the INS, Barometer and Complimentary filtered position each undershoot and estimate the position to be at least 2 meters under the ground level while Kalman only has an error of 1 meter below ground level). The reason of the Complimentary estimation overshooting is shown in the altitude velocity Figure 5.5c. The Altimeter both over and underestimates the velocity changes. Note that below 0 altitude positions is possible because the drone can follow the tracker downhill and the drone calibrates ground level (i.e altitude zero) at the point of takeoff, which can be on the top of a hill.

5.2.3 Random

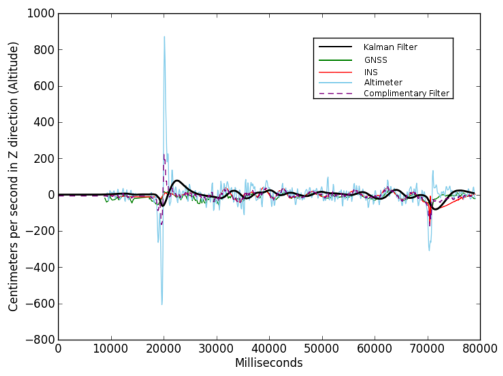
The flight pattern in this test is erratic and random at the pilots discretion, both in altitude and in XY-position. Results from this test is interesting in that it differentiates itself from the other tests because it does not follow a predetermined pattern. Position estimates (Figure 5.6a) by the Kalman Filter is almost 3 meters different from the nearest other estimated position by a sensor. The Complimentary filter behaviour is consistent and always gives and average between the GNSS and INS, while the Kalman Filtered position distinguishes itself by providing a clearly more different estimate at times. This is most obvious when the drone turns in higher velocities (e.g near coordinates -750, -1250 in Figure 5.6a).



(A) X-Velocity



(B) Y-Velocity

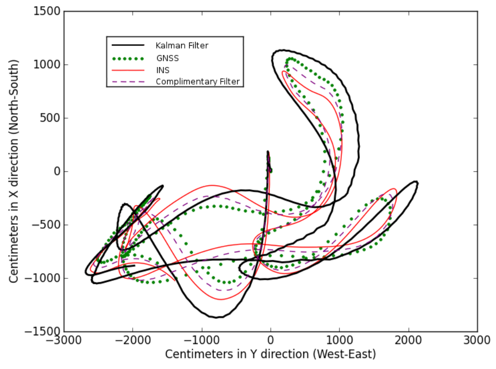


(C) Z-Velocity

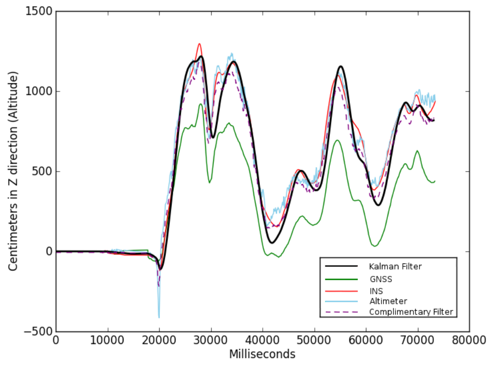
FIGURE 5.5: Velocity estimates for CIRCLE test observed over time

Looking at the altitude position estimation (Figure 5.6b we can see a problem with all sensors. The sensors should have indicated that the drone landed back on altitude zero (on the ground), but instead stops at 10 meters above the ground level. The GNSS position fix is somewhat better, but is still off by about 5 meters. This seems to be caused by a glitch in the drone firmware, since the data is incomplete and stops mid-flight (this can be seen in Figure 5.7a and Figure 5.7b where the data suddenly stops in mid-velocity instead of on zero as it should).

We can still glean some useful information from this test however, especially in the Z-velocity estimation in Figure 5.7c. Here we can see very noisy Altimeter sensor readings with many spikes. At 30 seconds all three sensor readings jump between -500 cm/s to about 500 cm/s while the Kalman Filter takes merely a dip to -250 cm/s before stabilizing at zero velocity again. The effects of this can be observed in Figure 5.6b at 30 seconds where an extra altitude spike in the Altimeter and INS is filtered away in favour for a more smooth position curve. This could be advantageous for the autopilot

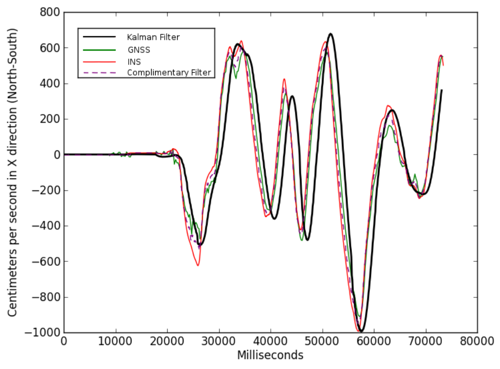


(A) X and Y position relative to takeoff (0, 0)

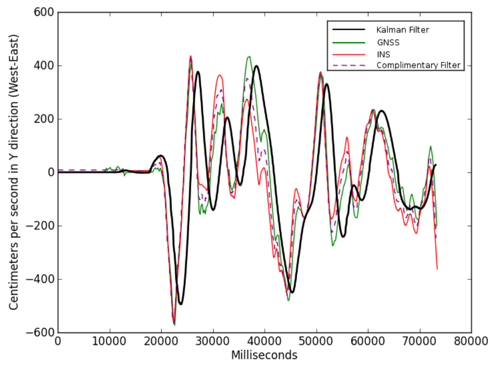


(B) Altitude above takeoff observed over time

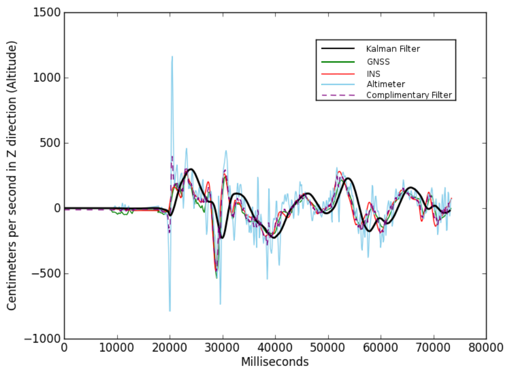
FIGURE 5.6: Position estimates for RANDOM test



(A) X-Velocity



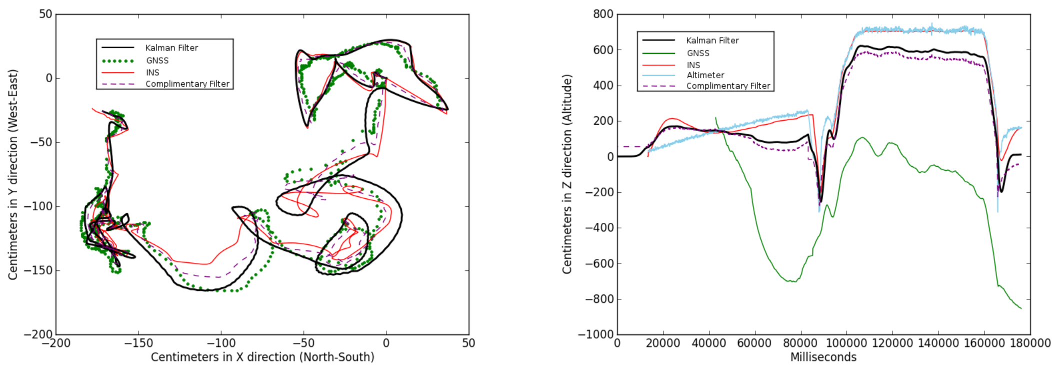
(B) Y-Velocity



(C) Z-Velocity

FIGURE 5.7: Velocity estimates for RANDOM test observed over time

because it should be able to navigate better with stable position data, rather than constantly trying to compensate with minor adjustments. If this really has any effect on in-flight navigation, remains to be seen in the Phase 3 test (Section 5.4).



(A) X and Y position relative to takeoff (0, 0) (B) Altitude above takeoff observed over time

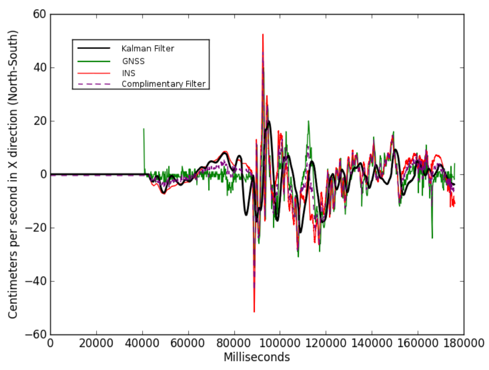
FIGURE 5.8: Position estimates for HOVER test

5.2.4 Hover

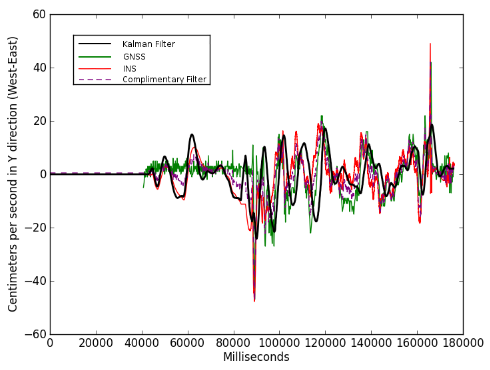
The drone was programmed to hover in a single position in loiter mode. Over a duration of 3 minutes the drone drifted almost 2 meters from it's original position (Figure 5.8a).

Figure 5.8b shows the altitude of the drone throughout the Hover test. Note that the GNSS does not get a proper 3D position fix until about 60 seconds (the green line). Furthermore the INS and Barometer seem to drift and gains altitude with a linear growth even though the drone is on the ground (takeoff does not happen until about 90 seconds into the test). The graph shows that the Kalman Filtered position (black) is more correct and does not drift as much. GNSS altitude seems to be very off-target and noisy here - its altitude estimation is off with about 8 meters. The INS and Barometer show a great dip in altitude at the point of takeoff (about 90 seconds). This is caused by the sudden voltage drop from the batteries when the engines start, but both sensors recover and correct their position after a few seconds. Note that the GNSS position, although inherently 8 meters incorrect, does not have any noise introduced by takeoff. This means that the GNSS altitude fixes contain useful information where the other sensors fail. Finally when the drone lands again, only the Kalman Filter correctly estimates that we are back on 0 cm altitude, while the INS and Barometer are about 2 meters incorrect while the Complimentary filter solution indicates we are almost 2 meters below ground.

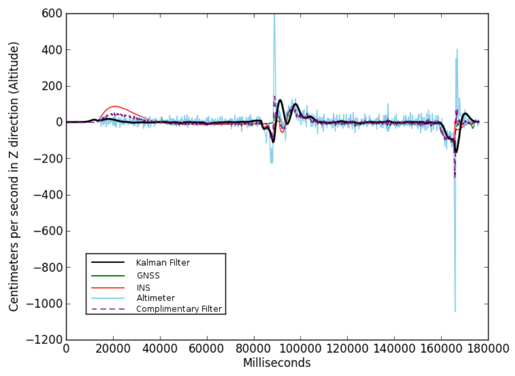
Figure 5.9c describes how the altitude velocity changes over time while the drone is hovering. Note the major spikes in velocities estimated by the barometer near takeoff and landing. The Kalman Filtered velocity estimation is smoothed and is not affected



(A) X-Velocity



(B) Y-Velocity



(C) Z-Velocity

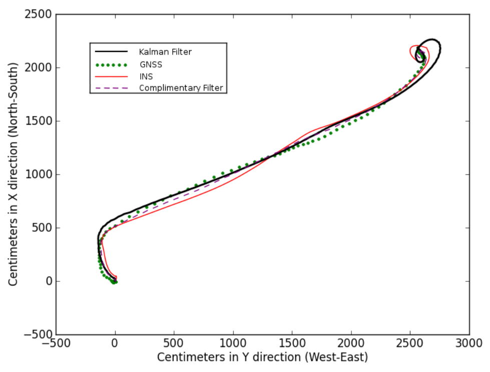
FIGURE 5.9: Velocity estimates for HOVER test observed over time

by the spikes and general noise of the barometer. If we look at the Complimentary filtered velocity (purple) however, we can see that it is more affected by the spikes and the noisy shift of the INS near the beginning (the noisy data appears about 20, 90 and 170 seconds into the test).

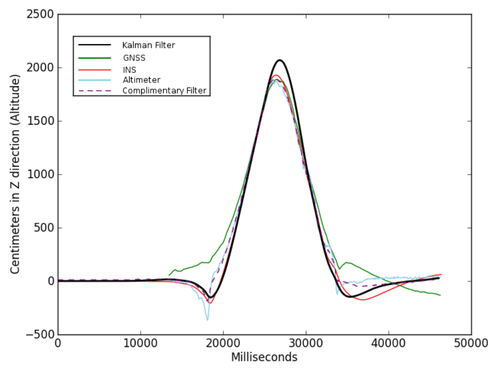
5.2.5 Altitude

This test is similar to the Straight Line test, except that the main purpose was to collect altitude data. In the XY-Position graph seen in Figure 5.10a, we can see that there are no major discrepancy between the position estimates and that the estimate differentiates by 1-2 meters at most.

A more distinguished result can be observed for the altitude position estimation in Figure 5.10b. As usual we have the noisy spikes near takeoff and landing (except for the GNSS) and a nice Gaussian curve generated by the Kalman Filter. If we compare the



(A) X and Y position relative to takeoff (0, 0)



(B) Altitude above takeoff observed over time

FIGURE 5.10: Position estimates for ALTITUDE test

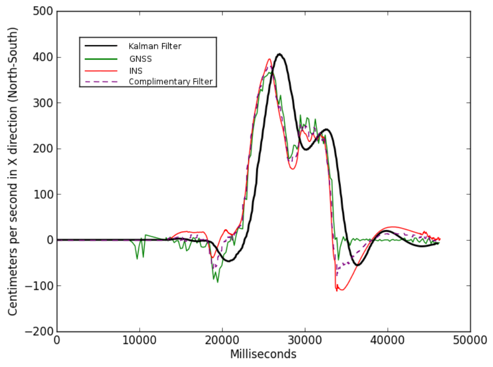
Complimentary filter with the Kalman Filter, we can see that they have similar performance and results. Two notable exceptions are on the top of the curve, where Kalman estimates about 1 meter more altitude and when landing where the Complimentary filter better estimates the ground level position, while Kalman is affected by the noise and estimates a below-ground position.

The velocity estimates show a lot of discrepancy in this test (Figure 5.11). This is especially true in the Y-velocity graph (Figure 5.11b) where the GNSS and INS estimate very different velocities at some points. When using the current noise covariances however, this does not seem to make any major impact on the position estimates which we earlier observed differentiated by 1-2 meters at most.

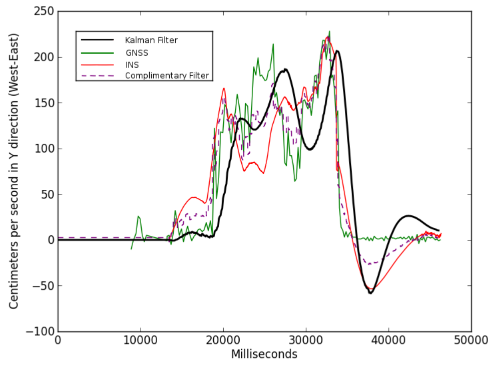
5.2.6 Memory and processing performance

Since runtime is independent from the data (i.e the data itself, not the amount of data), we can test the code on an MCU without the drone to test only the runtime and memory usage. This information can be used in the next phase (Phase 2) when optimizing the Kalman Filter.

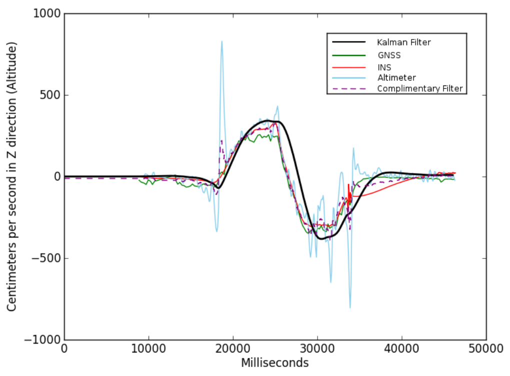
The total CPU load is calculated by multiplying the average load of one iteration with the frequency rate of the filter (100 Hz for the Drone and 10 Hz on the Tracker). This percentage indicates how much time the processor spends on the sensor fusion algorithm. If this percentage is too high, it will interfere with other parts of the system such as reading sensors, autopilot routines or transmitting data. Disturbing the processing loop



(A) X-Velocity



(B) Y-Velocity



(C) Z-Velocity

FIGURE 5.11: Velocity estimates for ALTITUDE test observed over time

Metric	**Platform**	**Kalman Filter**	**Complimentary Filter**
Microseconds	PC	16.43	0.0062
	Tracker	2872.00	< 1
	Drone	1666.22	42.44
Filtering Rate	PC	100 Hz	100 Hz
	Tracker	10 Hz	10 Hz
	Drone	100 Hz	100 Hz
Total CPU Load	PC	0.16%	< 0.01%
	Tracker	2.87%	< 0.01%
	Drone	16.66%	0.04%
Bytes of Memory	PC	1600	24
	Tracker	1248	48
	Drone	1536	264

TABLE 5.1: Comparison of memory and processing performance between the unoptimized Kalman Filter and Complimentary Filter

in this way can make the quadcopter crash. The Drone features task scheduling and a co-processor to avoid a crash, but the autopilot will go into fail-safe mode and make the drone land.

No such scheduling and co-processor exist on the Tracker, so this number must be closely monitored to avoid any resource starvation on the Tracker. It seems that with 10 Hz Kalman filtering rate, the 2.87% CPU load is within safe limits. If the Tracker were to run on 100 Hz, it would consume 28.7% which could cause problems. The reason the drone has to run on 100 Hz, is that its navigation system requires new state estimates every 10 milliseconds for the stabilization controllers to work correctly.

It is clear that the runtime and memory complexity of the Complimentary Filter is neglible on any platform when compared to the Kalman Filter. On the drone the Kalman Filter is over 416 times slower than Complimentary Filtering. The memory usage of Kalman Filter is almost 6 times larger, though still well below 2KB which should be more than acceptable. As shown in Appendix A there is 64KB RAM available on the Tracker and Appendix B shows we have 256KB RAM on the Pixhawk.

If we take a look at the CPU load of the Kalman Filter however, we can see that it consumes 16.66% of the total CPU processing power available on the Pixhawk. If the combined CPU load of all sub-systems and modules is near or above 100%, the combined processing tasks will overload the CPU and cause resource starvation. 16.66% for a single task is too high and indicates that the filter can interfere with the performance of other important systems. Therefore some optimizations and improvements are needed to reduce CPU load, which will be covered in Section 5.3.

The Complimentary Filter on the drone is more complex, as it runs some additional security checks and value verifications to ensure no erratic behaviour from the drone. These checks are also included in the Kalman Filter on the drone side, but not in the tracker. This does not affect the performance analysis because we are interested in the Kalman Filter versus Complimentary Filter on a given platform and not between different platforms.

5.3 Phase 2 Testing: Off-Line (post-optimization)

5.3.1 Code Improvements

By restructuring how data is stored and accessed, a significant improvement can be gained in both reduced memory usage and increased computation performance. The matrices are stored in a 2-dimensional structure, meaning a list of columns that contain rows. This generates unnecessary overhead for memory access and requires additional storage space for memory pointers.

Instead we can store the matrices as a single large structure and access data in a singular fashion. This eliminates the need of additional pointers and also utilizes the CPU cache better, reducing time required to access the data. This should speed up all matrix computations such as multiplication, addition, subtraction and inversion (results shown in Section 5.3.7).

Additional code optimizations were made after profiling and determining where the computation bottle-necks were located, this includes instruction reordering, using unsigned data types for matrix index iteration and other programming language semantics. For the 4x4 matrix inversion in the X and Y Kalman Filters, we can compute the inverse matrix by using the Cayley-Hamilton method instead of Gauss-Jordan elimination which reduces the computation time significantly[1]. Many of our matrices are diagonal and contain many zero values which have no effect on multiplication. By only multiplying non-zero terms in a given matrix, we can significantly reduce the amount of computations performed (often by a order of two or so). Examples of these diagonal matrices include the Observation Noise Covariance Matrix Q, Observation Matrix R_t and State Transition Matrix F_t which have many constant zero values in their matrices.

5.3.2 Model Optimizations

The previous tests have verified that the simplifications in Equation 4.9 (i.e removing the control input signal U) in the Kalman mathematical model still gives sensible and useful output when compared to the Complimentary Filter and the individual sensors without data fusion. This means we can completely eliminate them from the Kalman equations and further reduce the amount of computations required. Furthermore we can make some more assumptions of our matrix computations that allows us to reduce the amount of calculations performed each iteration.

[1]The specific code implementation of this algorithm was borrowed from the MESA implementation of the GLU library found at http://www.mesa3d.org/.

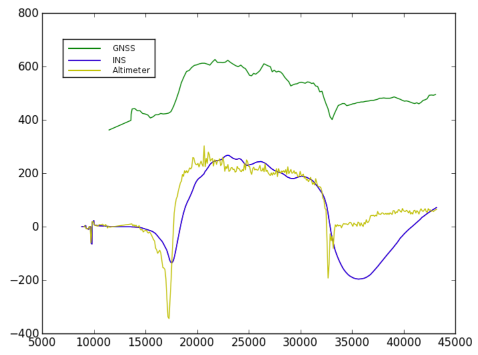
5.3.3 GNSS Altitude Recalibration

As we saw on the previous test results as shown in Figure 5.2b and Figure 5.8b, the GNSS can drift if the initial calibration is incorrect and the altitude position estimation will diverge from the altitude estimates of all the other sensors. To solve this in-flight recalibration was added where a correction offset is calculated and added to the GNSS, allowing us to still obtain some useful information even though the GNSS is heavily off-target. The calibration offset is calculated using Equation 5.1 where Δ is the altitude correction offset in a given time-step t, $\overline{Z_t}$ is the mean altitude given by all other sensors and Z_{GPS_t} is the altitude estimated by the GNSS receiver at the given time-step. Correction needs only to be applied if the difference between the GNSS altitude estimation and the mean altitude of the other sensors is higher than the VDOP, which is a strong indicator that the GNSS altitude estimation is diverging.

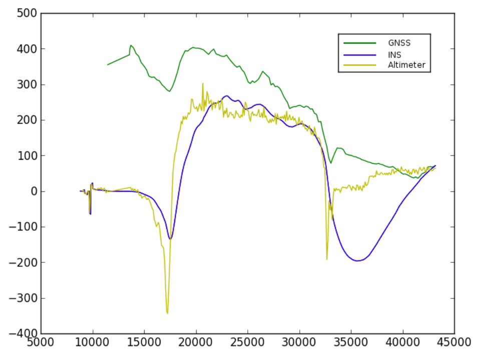
$$\Delta_t = \Delta_{t-1} * 0.98 + (\overline{Z_t} - Z_{GPS_t}) * 0.02 \tag{5.1}$$

Equation 5.1 uses the Leaky Integrator technique to make the recalibration converge to the correction offset which will generate smoother curves and more consistent output. The results of the in-flight calibration is shown in Figure 5.12. The Leaky Integrator technique is similar to how the Complimentary Filter is implemented and is based on the same theory. In a way, this makes the GNSS recalibration method a preliminary sensor fusion step to make the data more reliable, preventing it from diverging from the other sensors. It is important to note that the GNSS altitude recalibration method does break some assumptions in the Kalman theory, namely that our observation covariances are diagonal. By calibrating the GNSS using other sensors, we are creating a correlation between observations of the GNSS and the other sensors. This correlation is ignored because tests show that it does not have a major impact on the position accuracy and still allows the calculation simplifications discussed in Section 5.3.1. Without this method, the diverged GNSS data contains no useful data and only introduces noise to the Kalman equations.

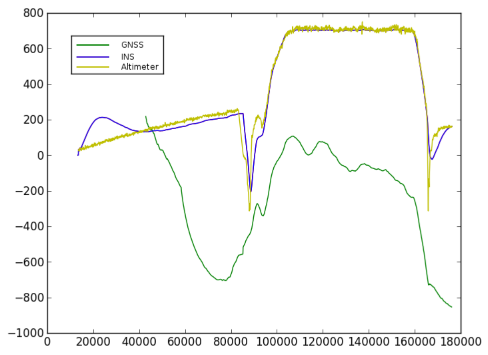
The calibration process makes data received from the GNSS sensor more useful, even when it's initial calibration is incorrect or when the sensor data begins to diverge from the other sensors. Figure 5.12a inherently starts with incorrect calibration and indicates



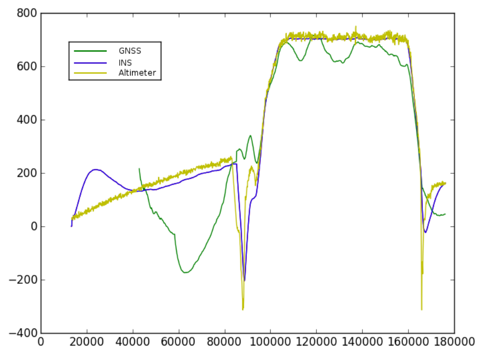
(A) No Calibration



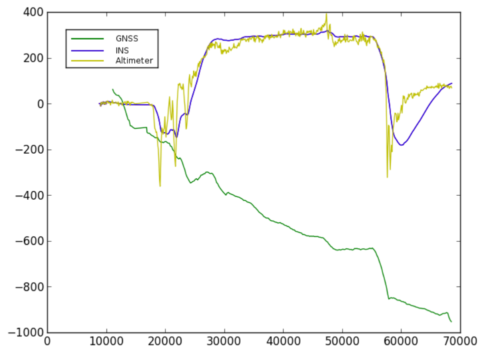
(B) Calibrated



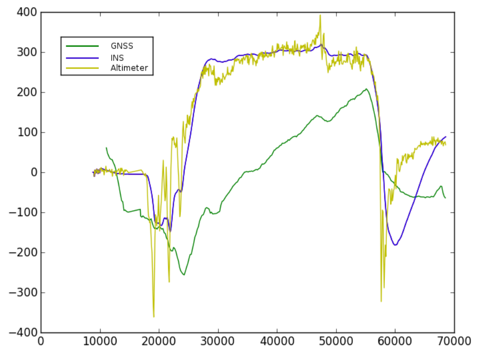
(C) No Calibration



(D) Calibrated



(E) No Calibration



(F) Calibrated

Figure 5.12: Altitude position estimates for the GNSS (green line) before calibration (left column) and after calibration (right colum). Yellow line indicates Altimeter and Blue lines show INS altitude.

that the drone is 4 meters in the air at both takeoff and landing. After calibration (Figure 5.12b) the initial estimate is unchanged, but over time calibrates itself and more correctly estimates that it has landed on the ground.

Figure 5.12c starts with a more correct calibration but begins to diverge as the GNSS gains a better altitude fix. After calibrating as shown in Figure 5.12d, the GNSS no longer diverges and is more consistent with the other data. Here we can even see that the GNSS contains better data than the Altimeter or INS as it does not contain the spike at takeoff or landing. Additionally the GNSS now correctly estimates we are back on the ground, while the INS and Altimeter still thinks we are 2 meters in the air. This shows that after calibration, the GNSS contains useful data that can be even more correct than the other two sensors.

In the final example in Figure 5.12e we have the worst case scenario where the GNSS completely diverges from the other data, containing little or no useful information and may in fact only introduce noise and no useful data to the equation. With the calibration process as shown in Figure 5.12f, the GNSS altitude estimations make more sense and gradually climb towards a more sensible altitude estimation. It now also shows that it has landed back on the ground, instead of 10 meters under the point of takeoff.

5.3.4 Tweaking the Observation Covariance Matrix

The final change before Phase 2 testing, is tweaking the covariance values in the Observation Noise matrix $\boldsymbol{R}_t$ in the Kalman Filter. The covariance values in Table 5.2 and Table 5.3 are calculated using the Equation 4.16 on data collected while the drone is stationary on the ground. The drone was placed on open ground with clear sky for good GNSS coverage and minimal external disturbances such as buildings, vibrations and magnetic influences. Three different sets of data was collected for a duration of 10 minutes each, which is the average flight time of the drone. The right column in Table 5.2 and Table 5.3 is the average covariance value from each of the data sets, which will be used in the Kalman Filter observation covariance matrix. Covariance values in the tables represents centimeters.

Note that the GNSS covariance is calculated dynamically in-flight using HDOP and VDOP as described in Section 4.5 and therefore does not have its own pre-calculated

Direction	**Data Set 1**	**Data Set 2**	**Data Set 3**	**Average**
X-Position	171.52	231.72	159.74	187.66
Y-Position	78.15	91.84	96.54	88.84
Z-Position	260.69	179.55	208.91	216.38
X-Velocity	129.00	346.46	96.69	190.71
Y-Velocity	22.00	274.08	86.08	127.72
Z-Velocity	108.00	164.92	140.31	137.74

TABLE 5.2: INS Covariances

Direction	**Data Set 1**	**Data Set 2**	**Data Set 3**	**Average**
Z-Position	261.86	202.44	208.72	224.34
Z-Velocity	234.71	358.54	375.73	322.99

TABLE 5.3: Altimeter Covariances

covariance numbers. This results in the observation covariance matrices R_x, R_y and R_z described in Equation 5.2, Equation 5.3 and Equation 5.4, respectively. σ is the HDOP and λ is the VDOP for the GNSS receiver.

$$R_x = \begin{array}{l} \textbf{X Observation Noise} \\ \text{GNSS Position} \\ \text{INS Position} \\ \text{GNSS Velocity} \\ \text{INS Velocity} \end{array} \begin{pmatrix} \sigma & 0 & 0 & 0 \\ 0 & 187.66 & 0 & 0 \\ 0 & 0 & \sigma & 0 \\ 0 & 0 & 0 & 190.71 \end{pmatrix} \tag{5.2}$$

$$R_y = \begin{array}{l} \textbf{Y Observation Noise} \\ \text{GNSS Position} \\ \text{INS Position} \\ \text{GNSS Velocity} \\ \text{INS Velocity} \end{array} \begin{pmatrix} \sigma & 0 & 0 & 0 \\ 0 & 88.84 & 0 & 0 \\ 0 & 0 & \sigma & 0 \\ 0 & 0 & 0 & 127.72 \end{pmatrix} \tag{5.3}$$

$$R_z = \begin{array}{l} \textbf{Z Observation Noise} \\ \begin{array}{l|cccccc} \text{GNSS Position} & \lambda & 0 & 0 & 0 & 0 & 0 \\ \text{INS Position} & 0 & 216.38 & 0 & 0 & 0 & 0 \\ \text{Altimeter Position} & 0 & 0 & 224.34 & 0 & 0 & 0 \\ \text{GNSS Velocity} & 0 & 0 & 0 & \lambda & 0 & 0 \\ \text{INS Velocity} & 0 & 0 & 0 & 0 & 137.74 & 0 \\ \text{Altimeter Velocity} & 0 & 0 & 0 & 0 & 0 & 322.99 \end{array} \end{array} \tag{5.4}$$

These new remodeled Observation Noise matrices replace the previous matrices described in Equation 4.12 and Equation 4.14. Changing the covariances will probably not have a profound effect on the result, but can affect which data the Kalman Filter weighs as most reliable in a given situation.

5.3.5 Determining the Process Noise

The process noise describes noise introduce from environmental factors such as wind or engine drift rather than the sensors themselves. By making the drone hold its position while hovering, we can calculate the covariance noise using Equation 4.16. Then the observation noise calculated in Section 5.3.4 is subtracted to find the process noise. Equation 5.5 shows how the process noise for the X filter is calculated, where W_x represents the covariance of position and velocity estimates in the X direction while hovering in one place.

$$Q_x = W_x - R_x = \begin{array}{l} \textbf{X Process Noise} \\ \begin{array}{l|cc} \text{Position} & 133.92 & 0 \\ \text{Velocity} & 0 & 148.71 \end{array} \end{array} \tag{5.5}$$

In the same way we can calculate the process noise for the Y and Z filter, shown in Equation 5.6 and Equation 5.7 respectively.

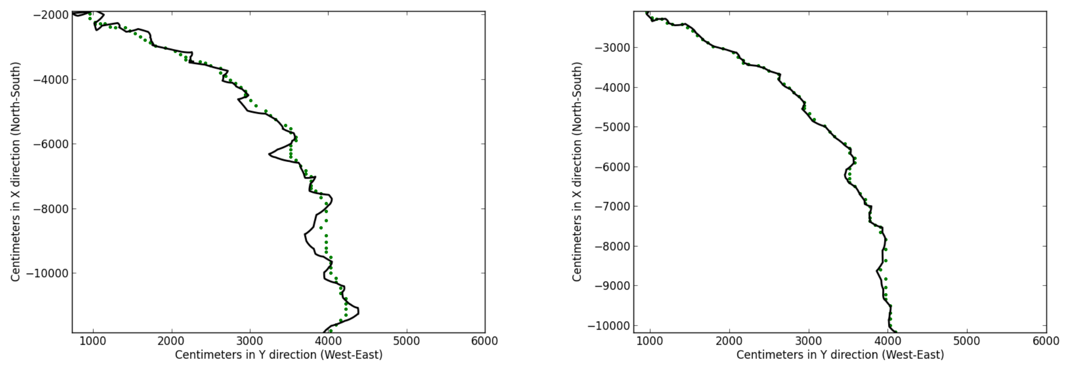
$$\textbf{Y Process Noise}$$

$$Q_y = W_y - R_y = \begin{array}{l} \text{Position} \\ \text{Velocity} \end{array} \begin{pmatrix} 22.84 & 0 \\ 0 & 99.72 \end{pmatrix} \tag{5.6}$$

$$\textbf{Z Process Noise}$$

$$Q_z = W_z - R_z = \begin{array}{l} \text{Position} \\ \text{Velocity} \end{array} \begin{pmatrix} 44.24 & 0 \\ 0 & 517.26 \end{pmatrix} \tag{5.7}$$

The impact of properly modeling process noise into the Kalman equation can be shown in Figure 5.13. While testing, the person holding the tracker moved in an exact circular pattern following drawn lines on the ground. Figure 5.13b shows that including process noise clearly reduces noise while smoothing the position estimates since it estimates the circle pattern more correctly.



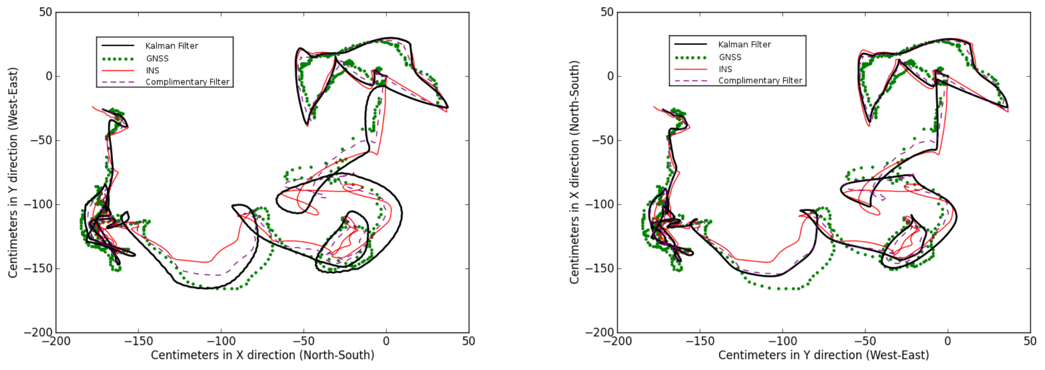
(A) Kalman Filtering without Process Noise (B) Kalman Filtering with Process Noise

FIGURE 5.13: The effects of including process noise in Kalman Filter position estimates

5.3.6 Phase 2 Results

In this section we take a look at some data from the previous and how the new changes to the Kalman Filter described earlier in this section affects the performance of the Kalman Filter. The calibrated GNSS altitude estimation is used by the Kalman Filter, but for reference the original uncalibrated altitude is shown in the graph. The Complimentary filter still uses the uncalibrated GNSS altitude estimations. It should be trivial to adapt the calibration method to this filter as well, but since the aim is to improve the Kalman Filter solution and not the Complimentary filter, no Complimentary filter improvements are realized in this thesis. The same test data as in Phase 1 is used for consistency, but only a subset of the data is compared since the goal of this section is to verify that the Kalman Filter still works correctly after the optimizations and tweaks made in this phase and not review all previous tests with slightly different parameters.

Figure 5.14 shows that the covariance tweaks slightly affects the position estimates of the Kalman Filter. The new Kalman estimates are less averaged and have a more distinct shape, meaning it deviates less from the GNSS and INS sensors.



(A) X and Y position relative to takeoff (0, 0) before changes in Phase 2

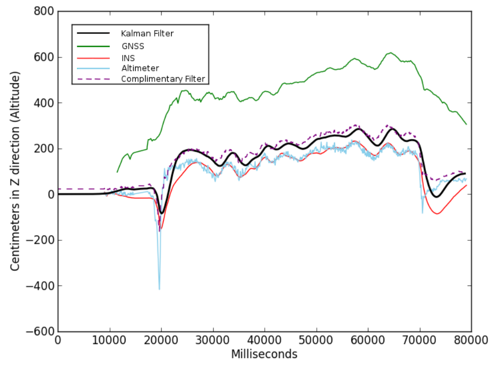
(B) X and Y position relative to takeoff (0, 0) after changes in Phase 2

FIGURE 5.14: XY-Position estimates for HOVER test before and after changes made in phase 2

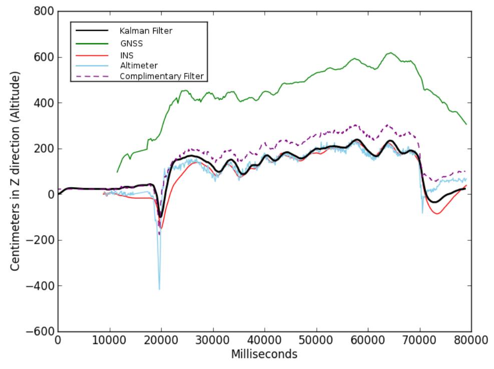
Figure 5.15 shows that the altitude position estimates are improved after the changes. By reducing the noise introduced by the GNSS through the calibration method and with the new covariance values, the Kalman Filter accurately estimates that we land back on the ground while still reducing the error on takeoff that all the other sensors and Complimentary filter suffer from (GNSS excluded). The complimentary filter estimates about 1 meter too high altitude because of the GNSS difference, while the Kalman Filter is not affected negatively by the GNSS.

Figure 5.16 shows again that the new horizontal position estimates are closer to the other sensors and also avoids the squiggly position estimates that Figure 5.16a has, thanks to the new covariance values. This squiggly behavior originates when the Kalman filter oscillates between two positions.

Figure 5.17 shows that the Kalman Filtered altitude estimation is less affected by the GNSS divergence than before. Now the Kalman Filtered positions is the estimate that

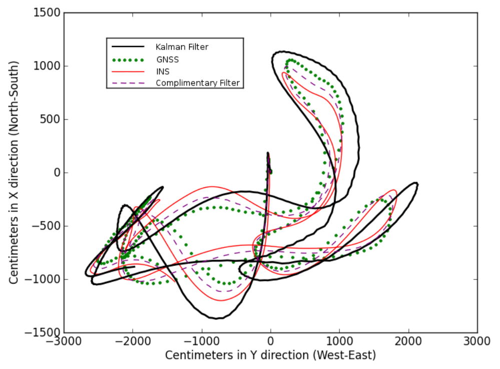


(A) Altitude above takeoff before changes in Phase 2

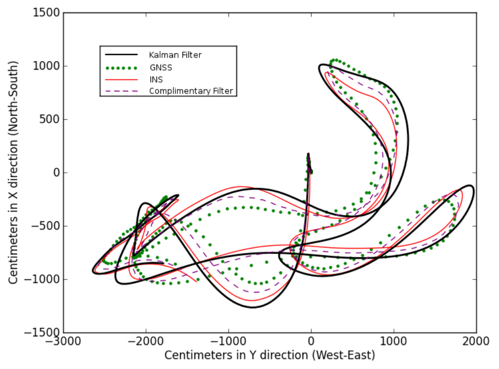


(B) Altitude above takeoff after changes in Phase 2

FIGURE 5.15: Altitude estimates for CIRCLE test before and after changes made in phase 2



(A) X and Y position relative to takeoff (0, 0) before changes in Phase 2



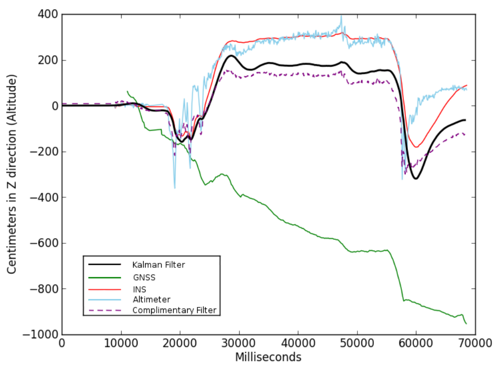
(B) X and Y position relative to takeoff (0, 0) after changes in Phase 2

FIGURE 5.16: XY-Position estimates for RANDOM test before and after changes made in phase 2

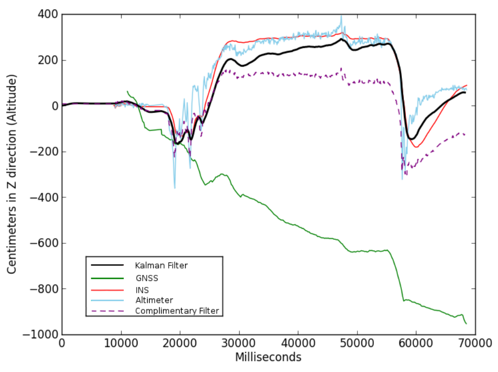
is least affected by the noise caused by the landing which all the other sensors suffer from.

5.3.7 Memory and processing performance

Table 5.4 compares the time and memory usage of the Kalman Filter before and after the optimizations mentioned in this section. The column labeled improvement shows the factor of how much improvement the given metric has benefited from the optimizations mentioned in this section. For example the first row shows that with an improvement factor of 2.31, the Kalman Filter runs more than twice as fast on the PC after the optimizations.



(A) Altitude above takeoff before changes in phase 2.



(B) Altitude above takeoff after changes in phase 2.

FIGURE 5.17: Altitude estimates for STRAIGHT LINE test before and after changes made in phase 2

The unoptimized performance are the same ones first described in Section 5.2.6 which is compared to Complimentary Filtering. Complimentary filter results remain unchanged and are therefore omitted from this section. The time usage represents the average microseconds it takes to complete one Kalman iteration. Filtering rates remain unchanged (100 Hz on PC and drone while 10 Hz on the Tracker).

Metric	**Platform**	**Unoptimized**	**Optimized**	**Improvement**
Microseconds	PC	16.43	7.11	2.31
	Tracker	2872.00	2541.00	1.13
	Drone	1666.22	287.01	5.80
Total CPU Load	PC	0.164%	0.07%	2.34
	Tracker	2.87%	2.54%	1.13
	Drone	16.66%	2.87%	5.80
Bytes of Memory	PC	1600	1392	1.14
	Tracker	1248	1104	1.13
	Drone	1536	1448	1.06

TABLE 5.4: Comparison of Kalman Filter processing performance and memory usage before and after optimizations

Profiling shows that the Kalman Filter on the drone benefited amazingly well from the optimizations, running 5.8 times faster. This reduces the CPU load of the Kalman Filtering from 16.66% to only 2.87% which is much better and acceptable. The large difference in runtime between the Tracker and Drone is because the Tracker does not have a floating-point arithmetic unit. The Kalman Filter algorithm uses decimal numbers for calculations and state estimates, therefore when the CPU does not have a FPU, these type of calculations are extra demanding. Luckily the tracker MCU is fairly powerful for a embedded CPU and can calculate decimals using 32-bit fixed-point arithmetics at a reasonable speed. It is clear that the tracker did not benefit from the optimizations as much as the drone, only improving by a factor of 1.13 because of its lack of a FPU.

As for memory usage, the improvement is small but still noticable. It is clear that the drone benefited the least from the memory optimizations, most likely because the Pixhawk supports native floating point types and therefore the compiler also optimizes differently. Also note that the drone uses slightly more memory than the tracker because it has some additional security checks and pointer references to various submodules which are not needed on the tracker. Even though the memory usage for a Kalman Filter is significant larger, the memory complexity is constant and never increases. Once the memory space has been allocated, we know for sure that the system does not suffer from too low memory.

5.4 Phase 3 Testing: On-Line (Autonomous Flight)

In this section the navigation data is no longer downloaded from the quadcopter and run on a PC, but actually runs on the MCU in the drone and the tracker. This means the drone will actually navigate using the filtered data from the Kalman Filter implemented in this thesis. The tests in this chapter are completely autonomous from takeoff to landing. For safety reasons, a pilot is nearby with a remote controller ready in case something goes wrong. The tracker will move in similar patterns as the previous pilot-controlled tests shown in Figure 5.1 to evaluate how well the tracking system performs and how the drone flies according to the Kalman Filtered tracker position estimates.

5.4.1 Autonomous Flight Tests

The first autonomous tests showed some problems when using the Kalman Filter on the drone. Even though all the values produces by the Kalman Filter were correct or similar to the old solution, the drone would be very twitchy and oscillate when trying to hold its position. Upon closer examination, the Pixhawk turned out to require new position updates at least 100 Hz for its stabilization algorithm to work correctly and the Kalman Filter was running at 10 Hz on the drone at this time.

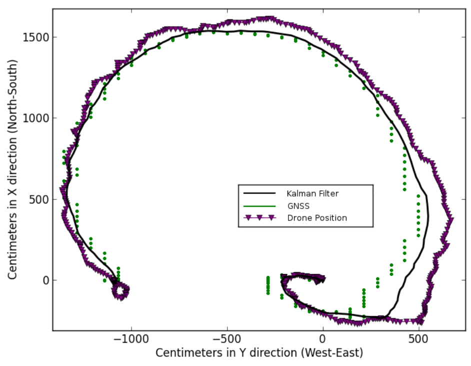
The drone was able to hold its position and fly using the Kalman Filter, but its behavior was still very twitchy. The exact reason of this is unknown, but the problem lies within the integration of the Kalman Filter into the Pixhawk autopilot system rather than the Kalman Filter. The reasoning behind this is that both simulations and Kalman Filtering on the tracker work correctly. Data logs produced show that the state estimates from the Kalman Filter running on the drone are also correct. The possibility of a timing issue as the source of the problem was eliminated by running the Kalman Filter in the background, while still using the state estimates from the old algorithm.

Since the Kalman Filter runs correctly on the Tracker and the drone already has a data fusion algorithm that is tested and works correctly, the Kalman Filtering was not used on the drone side for the live autonomous tests. Fixing this problem means diving into the Pixhawk autopilot firmware which is both out of scope of this thesis and not possible due to time constraints.

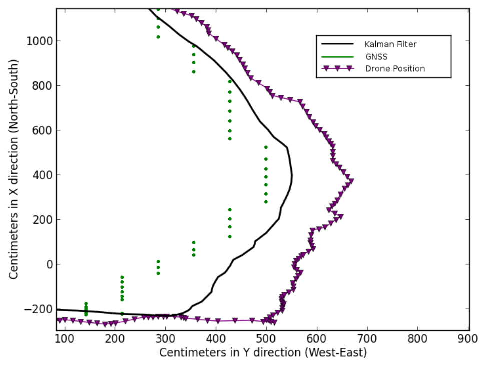
In these tests the drone will try to stay 1 meter away from the Kalman estimated tracker position and 8 meters above the altitude of the tracker as well. This distance makes both a good safety range for inaccurate sensors and a good video angle of the person carrying the tracker. The tracker transmits Kalman Filtered position and velocity estimates while the drone uses Complimentary Filter for navigation because of the problems mentioned above.

Figure 5.18a shows autonomous flight while the person holding the Tracker moves around in a circle. The purple line are positions of the drone following the Tracker. In Figure 5.18b we take a closer look of the right side of the circle in Figure 5.18a. Here it is clear that the GNSS receiver is having problem getting accurate position fixes, while the Kalman Filter handles this problem very well. The drone also has some inaccuracy in following the Tracker, but because the drone films using a wide angle from above, this inaccuracy of about 1 meter is not a problem as long as the drone correctly faces the right direction and correctly angles its camera. When looking at the video footage, it verifies that the camera always films the person holding the tracker and keeps the person centered in the image.

Figure 5.19a shows another autonomous flight test while the person holding the tracker is moving in a straight line. When taking a closer look in Figure 5.19b, we can see that the drone never diverges more than a meter away from the path of the tracker. Again we



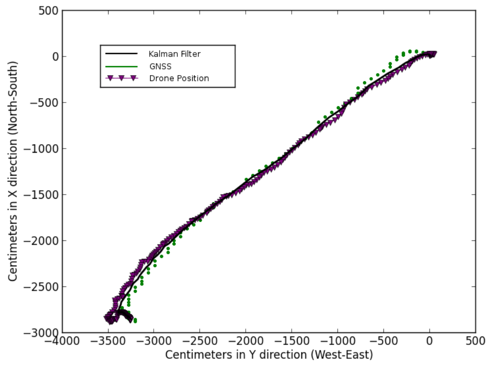
(A) Overview of the circle tracking



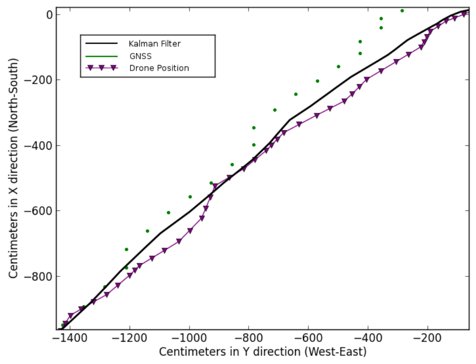
(B) Closeup view of the right-hand side of Figure 5.18a

FIGURE 5.18: XY-Position estimates for autonomous CIRCLE flight using the Tracker

see noisy GNSS position fixes which are smoothed out with the help of Kalman Filtering, providing a more reliable position estimate for the drone to follow.



(A) Overview of the straight line tracking



(B) Closeup view of Figure 5.19a

FIGURE 5.19: XY-Position estimates for autonomous STRAIGHT LINE flight using the Tracker

Finally, Figure 5.20 shows how the Kalman Filter smooths the velocity estimates on the Tracker. The GNSS only updates every 200 milliseconds, but with the help of the Kalman Filter, we also have velocity estimates between these updates. The filtered velocity estimates are also significantly less noisy and converges to a more reliable estimate.

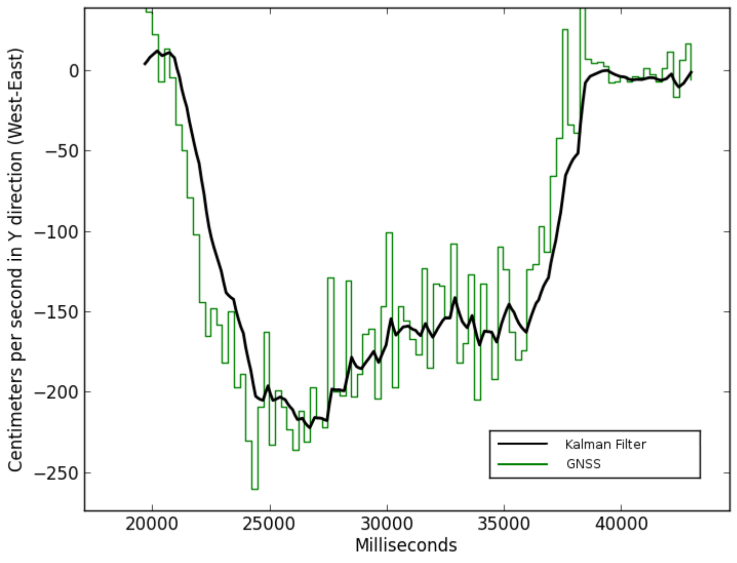


Figure 5.20: Kalman Filtered Y-velocity estimate on the Tracker.

Chapter 6

Conclusion and Discussion

6.1 Discussion

Results from Section 5.4 show that a Kalman Filter can run real-time on the limited hardware of a microcontroller and produce state estimates fast and accurate enough so that they can be used for localization and autonomous navigation. Evaluating the accuracy of the algorithm is difficult, but we have verified how well it works in autonomous flight compared to the existing complimentary filter solution. When both algorithms ran on the same data set, their results were not dramatically different. That is, the Kalman Filter produces estimates that are very close to the complimentary filter solution, seldom diverging by more than 1 meter. This verifies that the Kalman Filter is correct and if modeled correctly, also more accurate than the Complimentary Filter.

The Kalman filter could produce better results if the underlying system model described in Section 4.5 was improved. The INS data is currently dependent on corrections from INS and Altimeter to prevent it from drifting too much. The tweaks and improvements on the Kalman Filter in Section 5.3.3 added yet another correlation between GNSS and all other sensors (i.e when the GNSS altitude estimation diverges from the mean of all other altitude sensor readings by more than the VDOP, then a small correction is applied to prevent it from diverging). These different correlations are not correctly modeled in the observation covariance matrices shown in Section 5.3.4.

The correlations were ignored so that the observation covariance matrices would be a diagonal, allowing some calculation optimizations and simplifications. Test results show that while the computation cost of the Kalman Filter is a lot more expensive than the Complimentary Filter solution, the profiling results show that the Kalman Filter only uses 7% of the total CPU processing power available. Therefore, more interesting and distinct results could be produced by the Kalman Filter either by removing these optimizations and correctly modeling the covariance correlations between different sensors or by removing the correlations completely and let the Kalman Filter algorithm handle the diverging sensors and fuse the data as it should. The Kalman Filter is designed to handle the latter, but that requires a very robust model that accurately describes the underlying system model and its noise. For this to become reliable and dependable, extensive testing, tweaking and verification is required which requires a lot of time.

The sensor fusion model in this thesis also introduced two new observations, namely altitude from the GNSS and climb speed estimate from the Altimeter. These two observations were not included in the Complimentary Filter solution. Both observations are very noisy when compared to the other sensors, but the GNSS altitude estimate is the only absolute altitude estimate that does not drift by external influences except satellite reception. Both the Altimeter and INS are prone to constant drift in altitude and after only 5 minutes of flight time, it has shown that the altitude estimates from the Altimeter and INS can drift up to 5 meters when landing back on the same location as takeoff. The absolute GNSS altitude estimates contains data that could correct this drift assuming it has a stable satellite reception. By adding the GNSS altitude divergence correction in Section 5.3.3, the GNSS altitude estimates become influenced by the inherent drift from the INS and Altimeter which can reduce its overall accuracy.

Correlations between the sensors cause all the sensors to slightly converge and produce similar results. While this works well in practice and the drone navigates fine, it does not always accurately produce good results. The results are good enough for what the system was designed for, namely following and tracking. It does however require a safety distance of about 8 meters on the ground and 8 meters in the air to compensate for inaccurate sensors. The Kalman Filter currently behaves in most cases the same as the Complimentary Filter solution, but the Kalman Filter should perform better in bad situations such as GNSS glitches, magnetic disturbances or vibrations. This is because of Kalmans dynamic sensor weight adjustment as opposed to the static weights of the Complimentary Filter, which makes Kalman Filter more adaptable to different situations.

Results from Section 5.4.1 show that there were problems integrating the Kalman Filter into the complex Pixhawk system, which had problems navigating using Kalman Filtered data for reasons unknown. Collected data from tests show that unless the data was exactly the same as the previous Complimentary Filter, the drone would get erratic behavior the more its results diverged from that of the Complimentary Filter. The Pixhawk system is large and complex consisting of many modules. It was revealed that simply exchanging the old navigation system with the Kalman Filtered navigation system did not work flawlessly. The entire system is most likely fine tuned and tweaked to work with the old navigation system and replacing it with the Kalman Filter broke some assumptions or dependencies in other system modules. The decision to keep the old navigation system on the Pixhawk was made because redesigning the Pixhawk autopilot firmware to fit the Kalman Filter is out of the scope of this thesis, as well as time limitations.

The Kalman Filter ran correctly on the Tracking device however, which has even more limited hardware than the Pixhawk. One advantage of implementing the Kalman Filter on the Tracking device is that it is a new system designed and developed from scratch, with no other modules or systems that could interfere. Tests from Section 5.4.1 show that the drone running on the old navigation system was able to fully autonomously track and follow a user holding the Tracker device which used the Kalman Filter for determining its position. Autonomus takeoff and landing was adding using the Pixhawk autopilot which made the drone completely autonomous from beginning to end of the tests. With the help of the Kalman Filter, the tracker has sub-meter accurate position estimates. Proof of thise was when the quadcopter was able to adjust its yaw and pitch while a person waving the tracker from left to right or up and down while standing still.

6.2 Conclusion

This thesis began with describing how a Quadcopter worked and how it can navigate using an array of different sensors, each with their own abilities and limitations. It focuses on how these sensors could be combined using a method called sensor fusion to produce more accurate and better results than what the sensors are able to do individually. One such algorithm called the Kalman Filter was designed and implemented on both the drone navigation system and the tracking device. Although simulations and test results were promising, the Kalman Filter had problems integrating with the drone navigation system. Luckily the existing navigation system using a sensor fusion method called Complimentary Filter was reliable and robust enough to work along with the Kalman Filtered tracker device.

The Kalman Filter implementation did run fine on the Tracker however, and was able to fuse sensors to produce more reliable and accurate position estimates. These filtered position estimates were transimitted to the drone which used this data to adjust its attitude and position accordingly. This proves that the Kalman Filter design is sound and works in practice as well as in the simulations. It is likely that given more time investment, debugging and research the Kalman Filter should also work as a replacement of the old Pixhawk navigation system which uses complimentary filtering.

Tests show that the Kalman Filter is more reliable and accurate in certain scenarios when compared to the Complimentary Filter. Implementing a Kalman Filter requires a lot more time, testing and tweaking for it to perform correctly. It also requires good insight into the Kalman theory and linear algebra mathematics, while the complimentary filter is fairly simple and intuitive to tweak or fix. The question remains if this investment is worthwhile. If there are sensors that are prone to serious errors or glitches, then a Kalman Filter should definitely be considered as a viable option. The Kalman Filter also performs better if the sensors vary in quality, such as the GNSS which accuracy is dependent on the environment and usually improves over the first few minutes of running. Other than the more complex implementation, the Kalman Filter also requires significantly more processing power. Profiling tests indicate that the Kalman Filter requires up to 400 times more processing power than Complimentary Filtering and 6 times more memory usage. Unless there is ample hardware power available and noisy sensors that fit the Kalman requirements, a more simpler filter should be used. In the tracking system implemented in this thesis however, the hardware could handle a Kalman Filter and produced more accurate estimates than the complimentary filtering, which makes Kalman the right choice.

Through live flight tests, the system was verified to work completely autonomous from beginning to end. A person without any experience or training with drones is able to use the drone for producing aerial footage. This process required no input or control from the user except for indicating when it should takeoff or land by pushing a button on the tracker.

6.3 Further Work

6.3.1 Improving the Kalman Filter

While modeling the Kalman Filter in Section 4.5 an assumption was made that acceleration is constant during a given time-step so that the model would become linear and fit with the requirements of a Kalman Filter. This is however an incorrect model and can lead to inaccurate estimations. By including nonlinear acceleration state estimates using for example the Extended Kalman Filter or Unscented Kalman Filter algorithms, the accuracy of the state estimates could improve even more.

The Kalman Filter could also be used to estimate other nonlinear states such as the attitude of the drone, estimating pitch, yaw and roll by fusing the gyroscopic, acceleration and magnetometer data. Combining acceleration with attitude, the NED velocity can be calculated which was previously done by the DCM algorithm and constant acceleration estimates from the INS.

Another use of the Kalman Filter would be adding wind speed estimates, which would be very useful in estimating how the drone should behave to counter the effects of weather and filter out the wind noise from the speed estimates. An additional wind speed sensor could be fused into the other data to improve the velocity estimates and especially the external wind when the drone is hovering still. This could be difficult because quadcopters generate a lot of wind noise because of the constant spinning propellers which hold the vehicle airborne.

Adding new state estimates could radically increase the model complexity and therefore make it computationally more expensive. However, the test results in Section 5.3.7 show that the current Kalman Filter which has 6 state estimates runs easily within the computational constraints of the micro-controller, which indicates that more complex algorithms should be able to run on the same hardware.

The accuracy of the filter could be better estimated if it also included a software simulation where we know the ground truth and can calculate exactly how much the estimates of a given sensor fusion algorithm diverges from the truth. Data collected from flight test should be used when designing the simulator so that it models real world application as closely as possible. A pitfall of using simulator when designing a Kalman algorithm is that the Kalman model will be designed, tweaked and optimized for the simulator which does not necessarily describe how it will perform in a real world application.

6.3.2 Replacing the Complimentary Filter with Kalman Filter

Both simulation results and the live Tracker results show that the Kalman Filter design is correct. The fact that it does not fit into the Pixhawk navigation system is therefore most likely due to an incompatibility in the Pixhawk system rather than an error in the Kalman Filter. The next step would be to find out why the Pixhawk works fine with data from the complimentary filter, but not the Kalman Filter. Test running the Kalman Filter, but still used the results from complimentary filtering ran also fine without any problems, eliminating any timing problem caused by the Kalman Filter as a possible cause. It is worth noting that even through its behavior was twitchy and erratic, the quadcopter was able to hold its position which means the Kalman Filter did work to some degree. Only by removing the cause of the erratic twitches, the true accuracy and performance of the Kalman Filter compared to the Complimentary Filter can be determined. Simulation results indicate however that the Kalman Filter is more reliable and accurate than the old solution, but if this is something that makes a difference in practice is not confirmed in this thesis.

6.3.3 Collision Avoidance

Collision avoidance is another big problem for the tracking system. Even though it can follow the tracker, it has no inherent method of avoiding collisions, such as trees or power cables. Currently the system works by assuming that the tracker is moving through a obstacle free environment and that the user keeps an eye out for possible collisions. If collision avoidance was completely automated, this would allow the user to ignore the drone tracking and focus on whatever he or she is doing, trusting the drone to handle itself in any situation. The collision avoidance problem is complex and hugely scoped that it could be an entire masters thesis by itself. Therefore we will only take a cursory look at it using experiences gained from working with the drone so far.

One way of implementing collision avoidance is by adding range-finders, sensors that measure the distance between obstacles and the drone. There are two types of range-finders; sonars and laser technology each with their own advantage and disadvantages.

Sonars can detect obstacles in a large cone, but have short range (usually not more than 6 meters). If a drone is flying full speed towards a obstacle at 60 km/h and does not detect the obstacle until it is 6 meters near it, then it has only 0.36 seconds to react and avoid crashing into the obstacle. Furthermore sonars are susceptible to the noise generated by the propellers on the drone making them unsuitable for high velocity obstacle avoidance required when tracking.

Laser range-finders have usually much longer range, but suffer from a narrow beam. Laser range-finders are also more expensive than sonars and because of their narrow beam, more than one laser range-finder is usually required to get a complete obstacle detection. A moving laser range-finder or multiple range-finders can be combined to create a Lidar system. Because Lidars combines the advantages of both sonars and lasers while having none of their disadvantages, they show a lot of potential for collision avoidance. However, Lidars are normally very expensive, a non-millitary grade Lidar costing more than all other components combined, including the quadcopter itself!

Another collision avoidance solution would be to preemptively upload navigation data containing a map an locations of all obstacles. This solution has a couple of drawbacks however, as terrain topology data requires an up-to-date database of obstacles, which can be problematic in remote areas such as mountains, forests or other non-urban areas. In addition to storage problems, topology data only contains static obstacles such as terrain or foliage and does not account for moving objects such as cars or people.

6.3.4 Using Computer Vision to aid Localization and Navigation

Instead of using range-finders, obstacles could be detected using computer vision algorithms to analyze images from the the video camera. However, this requires more powerful hardware to be able to process image data real-time and possibly two separate cameras for stereoscopic vision. Using techniques such as SLAM[18] one could provide both collision avoidance and tracking in GNSS denied environment. The hardware problem could be avoided by adding a separate micro-controller system dedicated for computer vision and only warns the main Pixhawk board of nearby obstacles. Optical flow algorithms could also be potentiality used for velocity estimates, which is a relatively cheap and simple algorithm compared to other computer vision algorithms.

6.3.5 Flight Path Prediction

Another potential area of research and development for the tracking system would be intelligent path generation based on movement data from the tracker. The tracker has an IMU, Barometer and GNSS which means it can predict its movements and actions which the drone can use to improve its tracking performance. For example if the tracker detects a sudden turn or that the tracker is moving in a pattern that usually indicates a jump (e.g a ski jump on a downhill piste), then the drone can use this knowledge to earlier react to these situations and improve its tracking performance accordingly.

Velocity estimates are already readily available as a state estimate produced by the Kalman filter. By calculating the communication delay between Tracker and drone and figuring out the actuator delay of the drone, a new more accurate position estimate can be calculated. Instead of the drone reacting to positions where the tracker has been, the drone will respond to positions where the tracker currently is and possibly where it will be in the future. This will lead to better responsiveness of the drone and because of that, also better tracking.

Appendix A

Tracker Microprocessor

The following table contains the technical specification of the Micro-Controller Unit (nicknamed "Teensy") that is used in the Tracker. The data was obtained from the MCU producers website and can be found here: https://www.pjrc.com/teensy/teensy31.html. Note that the this processor does not feature a Floating Point arithmetic unit, which means it has to simulate using fixed-point decimals which is very slow in comparison.

Processor		MK20DX128VLH5	
	Core	32-bit Cortex-M4	
	Rated Speed	72	MHz
	Overclockable	96	MHz
Flash Memory		256	Bytes
	Bandwidth	192	MBytes/sec
	Cache	256	Bytes
RAM		64	kbytes
EEPROM		2	kbytes
Digital I/O		34	Pins
	Voltage Output	3.3V	Volts
	Voltage Input	5V Tolerant	Volts
Communication			
	USB	1	
	Serial	3	
	SPI	1	
	I2C	1	
	CAN Bus	1	
	I2S Audio	8	

Appendix B

Pixhawk

The Pixhawk uses the STM32F427 microproessor which is designed for medical, industrial and consumer applications where the high level of integration and performance, embedded memories and peripherals inside packages as small as 10 x 10 mm are required. It features a floating-point processing unit which means it is much more effective when doing arithmetics that include decimal numbers.

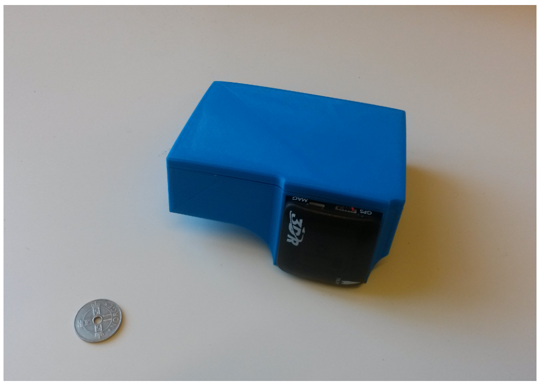
Processor		STM32F427	
	Core	32-bit Cortex-M4	
	Rated Speed	168	MHz
	Fail-safe Co-Processor	32-bit STM32F103	
Flash Memory		2048	Bytes
RAM		256	kbytes
Sensors			
	STMicro L3GD20 3-axis gyro		
	STMicro LSM303D 3-axis accel/mag		
	Invensense MPU6000 3-axis accel/gyro		
	MEAS MS5611 barometer		
Communication			
	USB	1	
	Serial	5	
	SPI	1	
	I2C	1	
	CAN Bus	2	

Appendix C

Tracker Prototype

This appendix describes the development of the third hardware revision of the Tracker (shown in Figure C.1). The outer box itself was created by a 3D plastics printer while the electronic design along with drivers and Teensy firmware was developed by the author and Anders B. Eie (another master degree student working on the CPTR team). The electronic schematic, soldering and selection of components and figuring out how they work was all part of getting the Tracker to work as we wanted.

Note the Ublox GNSS Receiver and Compass attached externally on the side of the tracker in Figure C.1 for better exposure to satellites and less magnetic inteference from other components.


(A) Tracker prototype with lid in place, protecting to the sensitive electronics inside.

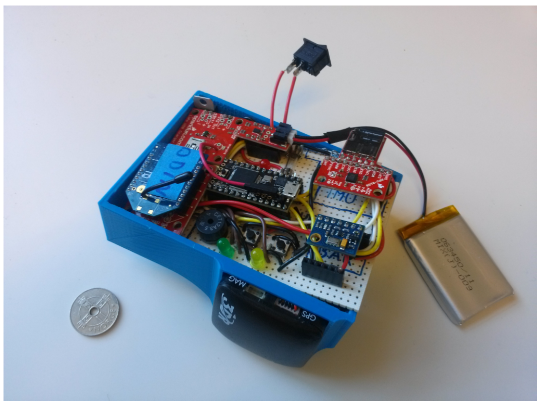

(B) Tracker prototype with lid taken off, exposing its inner electronic components.

Figure C.1: The third revision prototype of the Tracker Device with a "Norwegian krone" coin for scale.

The tracker that you see in Figure C.1 is the third revision of the Tracker, which means we had two earlier designs containing similar or different components for testing. In Figure C.1b the electronic components and sensors are exposed and the third revision Tracker contains the following components:

1. Teensy MCU
2. GNSS and Compass
3. IMU
4. Barometer
5. Data Logger (SD)
6. Control switches and LEDs
7. Speaker (for audio feedback)
8. Wireless transmitter
9. Electronic battery and charging circuit

This is far from the final revision of the Tracker. Further work is required to make it a lot smaller and attach additional buttons for the user to signal the Pixcuckoo that they wish the drone to takeoff, land or to follow the user. It also needs to be fully dust and water proof if it is to be used for activities such as skiing, biking or for watersports.

Appendix D

Kalman Cheat Sheet

This appendix provides a brief overview of the Kalman Filter theory and the different Kalman related variables used throughout the thesis.

Prediction

1. Project the state ahead

$$\boldsymbol{x'}_{t+1|t} = F\boldsymbol{x'}_{t|t} + \boldsymbol{Q} + B\boldsymbol{u}_t \tag{D.1}$$

Calculate the rough estimate of $\boldsymbol{x'}$ at timestep $\boldsymbol{t+1}$. This value is updated in the Measurement Update step of the algorithm.

2. Project the error covariance ahead

$$\boldsymbol{P'}_{t+1|t} = F\boldsymbol{P}_{t|t}\boldsymbol{F}^T + \boldsymbol{R} \tag{D.2}$$

Calculate a measure of the estimated accuracy of the state estimate which is used for computing the Kalman Gain in Equation D.3. This estimate scorrected in step 3 of the Measurement Update after the observation is known (Equation D.5).

Measurement Update

1. Compute Optimal Kalman Gain

$$\boldsymbol{K}_t = \boldsymbol{P'}_{t|t-1}\boldsymbol{H}^T(\boldsymbol{H}\boldsymbol{P'}_{t|t-1}\boldsymbol{H}^T + \boldsymbol{R})^{-1} \tag{D.3}$$

The Kalman gain is a function of the relative certainty of the measurements and the current state estimate, so that it works as a bias towards either state estimation or measurements.

2. Update the estimate via $\boldsymbol{z}_t$

$$\boldsymbol{x}_{t|t} = \boldsymbol{x'}_{t|t-1} + \boldsymbol{K}_t(\boldsymbol{z}_t - \boldsymbol{H}\boldsymbol{x}_{t|t-1}) \tag{D.4}$$

This value is the estimation of $\boldsymbol{x}$ at timestep t (this is the improved estimation value we are going to use and is the output of the algorithm at this timestep).

3. Update the error covariance

$$\boldsymbol{P}_{t|t} = (\boldsymbol{I} - \boldsymbol{K}_t\boldsymbol{H})\boldsymbol{P'}_{t|t-1} \tag{D.5}$$

We now recalculate the error covariance $\boldsymbol{P}_t$ which is used in the next timestep $(t + 1)$ in the Prediction step 2 (Equation D.2).

F_t **Transition Model** A matrix that describes how the state estimates affect each other between iterations. For example, velocity would affect the position estimation for the next iteration. This model is constant and normally does not change over time.

Q **Process Noise** A matrix containing the environmental noise influence. This models any noise that is not in the observations.

R_t **Observation Noise in timestep** t This represents how much noise there is in each sensor and if they affect each other. A high noise indicates that the sensor is more unreliable and will be less weighted when making state estimates.

O_t **Observations made in timestep** t This vector contains all the observations made in timestep t. These observations contain noise as indicated by R_t.

H_t **Observation Model** The Observation Model describes how each observation affects a state estimate. For example, a GPS receiver would affect the state estimates of both positions and heading while a compass would only affect heading.

X_t **State estimate in timestep** t This is the state estimate produced by the Kalman Filter in a given timestep. This value is the data fused estimate which is more accurate than the noisy observations given by O_t.

Z_t **Noisy State Estimate** Z_t denotes the uncorrected state estimate. The error covariance P_t is used to calculate the corrected state estimate X_t.

K_t **Kalman Gain** The Kalman Gain matrix is calculated each timestep and contains how much weight is but on each observation. This essentially fuses each observation together to produce a new and more accurate estimation.

P_t **Error Covariance** Each iteration the Kalman algorithm calculates the error covariance in state estimates which are used to correct future estimates.

B **Signal Input Model** Signal input model describes how the signal input affects the state estimates each iteration.

u_t **Signal Input** The signal input represents external input that is modeled by the state transition model F. This could for example be control input from the pilot controlling position or gravity pulling the position towards the earth.

Bibliography

[1] Jan Wendel, Oliver Meister, Christian Schlaile, and Gert F. Trommer. An integrated gps/mems-imu navigation system for an autonomous helicopter. *Aerospace Science and Technology*, 10(6):527–533, September 2006. URL http://www.sciencedirect.com/science/article/pii/S1270963806000484.

[2] Fédération Aéronautique Internationale. Academy of model aeronautics national model aircraft safety code, January 2014. URL http://www.modelaircraft.org/files/105.pdf.

[3] Teppo Luukkonen. Modelling and control of quadcopter. *Aalto University*, April 2011.

[4] Changsun Yoo and Lee Ki Ahn. Low cost gps/ins sensor fusion system for uav navigation. *IEEE*, pages 8.A.1–1 to 8.A.1–9, 2003. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1245891.

[5] Alison K. Brown. Test results of a gps/inertial navigation system using a low cost mems imu. *Proceedings of 11th Annual Saint Petersburg International Conference on Integrated Navigation System*, May 2004.

[6] Morgan Quigley, Michael A. Goodrich, Stephen Grifths, Andrew Eldredge, and Randal W. Beard. Target acquisition, localization, and surveillance using a fixed-wing mini-uav and gimbaled camera. *IEEE*, pages 2600–2605, April 2005. URL http://faculty.cs.byu.edu/~mike/mikeg/papers/QuigleyGoodrichBeardICRA2005.pdf.

[7] Rudolph Emil Kalman. A new approach to linear filtering and prediction problems. *Transactions of the ASME–Journal of Basic Engineering*, 82(Series D):35–45, 1960.

[8] Walter T Higgins. A comparison of complementary and kalman filtering. *IEEE Transactions on Aerospace and Electronic Systems*, (3):321–325, May 1975.

[9] Vikas Kumar. Integration of inertial navigation system and global positioning system using kalman filtering. Master's thesis, Department of Aerospace Engineering. Indian Institute of Technology, Bombay, July 2004.

[10] Thang N.Va, Tang P. Vb, Ninh V.Vc, Trinh C.Dc, and Tan T.Dc. Application of extended and linear kalman filters for an integrated navigation system. *ICCE*, January 2012.

[11] Kim Jonghyuk. *Autonomous Navigation for Airborne Applications*. PhD thesis, Department of Aerospace, Mechanical and Mechatronic Engineering. University of Sydney, Sydney, May 2004.

[12] Mathieu St-Pierre and Denis Gingras Dr. Ing. Comparison between the unscented kalman filter and the extended kalman filter for the position estimation module of an integrated navigation information system. *IEEE Intelligent Vehicles Symposium*, pages 831–835, June 2004. URL `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1336492`.

[13] Michael George and Salah Sukkarieh. Tightly coupled ins/gps with bias estimation for uav applications. URL `http://prism2.mem.drexel.edu/~vefa/research/HeliExtLoadStabil/relevantPapers/imu/george.pdf`.

[14] Guoqiang Mao, Sam Drake, and Brian D. O. Anderson. Design of an extended kalman filter for uav localization. *IEEE, Information, Decision and Control*, pages 224–229, 2007. URL `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4252506`.

[15] Simon J. Julier and Jeffery K. Uhlmann. Unscented filtering and nonlinear estimation. *Proceeedings of the IEE*, 92(3):401–421, March 2004. URL `http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1271397`.

[16] Robert A. Gray and Peter S. Maybeck. An integrated gps/ins/baro and radar altimeter system for aircraft precision approach landings. *IEEE*, 1:161–168, May 1995.

[17] Eric A. Wan and Rudolph van der Menve. The unscented kalman filter for non-linear estimation. *Proceedings of Symposium 2000 on Adaptive Systems for Signal Processing, Communication and Control (AS-SPCC)*, pages 153–158, 2000. URL http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=882463.

[18] Jonghyuk Kim and Salah Sukkarieh. Slam aided gps/ins navigation in gps denied and unknown environments. *The International Symposium on GNSS/GPS*, December 2004. URL http://users.cecs.anu.edu.au/~Jonghyuk.Kim/pdf/GNSS2004-SLAMaidedGPSINS-No37-Kim.pdf.

[19] Markus Achtelik, Tianguang Zhang, Kolja Kiihnlenz, and Martin Buss. Visual tracking and control of a quadcopter using a stereo camera system and inertial sensors. *IEEE International Conference on Mechatronics and Automation*, pages 161–168, August 2009.

[20] u-blox AG. u-blox 6 gps modules, 2010. URL http://www.terraelectronica.ru/pdf/UBLOX/LEA-6H.pdf.

[21] Greg Welch and Gary Bishop. An introduction to the kalman filter. September 1997. URL http://clubs.ens-cachan.fr/krobot/old/data/positionnement/kalman.pdf.

[22] Qingsheng Kong, Shenglei Xu, and Sang sun Lee. Using pdop to estimate kalman filter's measurement noise covariance for gps positioning. *ICTTE International Conference on Traffic and Transportation Engineering*, 26, 2012. URL http://www.ipcsit.com/vol26/7-ICTTE2012-T012.pdf.

[23] William Premerlani and Paul Bizard. Direction cosine matrix imu: Theory. *Draft*, 5 2009.