Master's Project

Czech
Technical
University
in Prague

F3

Faculty of Electrical Engineering
Department of Computer Science

Kalman filter application for localization improvement of multi-copter UAV

Tomáš Trafina

Supervisor: Dipl. Ing. Tomáš Meiser
Field of study: Cybernetics and Robotics
Subfield: Robotics
May 2018

Acknowledgements

I would like to express my gratitude to my supervisor Tomáš Meiser for the useful comments, remarks, and engagement through the learning process of this master thesis. Furthermore, I would like to thank Milan Rollo for giving me the opportunity to participate in such project. I would like to thank my loved ones, who have supported me throughout the entire process, both by keeping me harmonious and helping me putting pieces together.

Declaration

Abstract

The purpose of this work is to develop mathematical apparatus for a coaxial hexacopter, equipped with IMU (inertial measurement unit) and GNSS (global navigation satellite system) sensors, which will improve the accuracy of the aircraft's localization based on its mathematical model involving Kalman filtering approach. The main goal is to improve data from GNSS sensor as the data from IMU already satisfy the required qualities.

Two methods were chosen to verify the expected improvement. The primary one is a mathematical analysis of the aircraft's trajectories constructed using raw data from GNSS sensor for one model and processed data from Kalman filter for the other one. The secondary method involves usage of another sensor in the system, a lidar (light detection and ranging). Raw data of all three sensors is acquired during a flight of the aircraft. Then two point cloud models are constructed using the two trajectories data described above. Visual comparison is used to determine if the point cloud from processed data has better accuracy than the one from raw data.

Primary results of this work are a mathematical model of the real aircraft and filtering apparatus which increases localization data accuracy. Moreover, algorithms for raw data acquisition and its fusion into 3D models were developed.

Keywords: UAV, IMU, GNSS, Kalman filter, loacalization, mathematical model, coaxial hexacopter

Supervisor: Dipl. Ing. Tomáš Meiser
Dept. of Computer Science, FEE,
CTU in Prague
Technická 2
16627 Praha 6

Abstrakt

Cílem této práce je vyvinout matematický aparát pro koaxiální hexakoptéru, vybavenou IMU (intertial measurement unit) a GNSS (global navigation satellite system) senzory, který zvýší přesnost lokalizace tohoto letounu pomocí jeho matematického modelu s využitím přístupu Kalmanova filtrování. Důraz je kladen především na vylepšení dat z GNSS senzoru, neboť přesnost dat z IMU již splňuje požadované nároky.

Pro ověření očekávaného zlepšení byly zvoleny dvě metody. Hlavní z nich je matematická analýza dvou trajektorií zkonstruovaných jednak ze surových dat samotného GNSS senzoru a druhak ze zpracovaných dat, která jsou výstupem z Kalmanova filtru. Pro druhou metodu je do systému zaveden třetí senzor, lidar (light detection and ranging). Při letu UAV jsou zaznamenána data ze všech tří senzorů, následně jsou zkonstruovány dva modely mračen bodů pomocí dvou trajektorií popsaných výše. Pohledovým porovnáním lze určit zda mračno bodů ze zpracovaných dat dosahuje vyšší přesnosti.

Výstupem této práce je především matematický model fyzického letounu a filtrační aparát pro zvýšení přesnosti lokalizačních dat. Dále byly vyvinuty algoritmy pro záznam surových dat jednotlivých senzorů a jejich následné využití pro konstrukci 3D modelu.

Klíčová slova: UAV, IMU, GNSS, Kalmanův filtr, lokalizace, matematický model, koaxiální hexakoptéra

Překlad názvu: Aplikace Kalmanova filtru pro vylepšení lokalizace multi-rotorového UAV

Contents

Figures

Chapter 1

Introduction

Nowadays unmanned aerial vehicles (UAVs) development is an important topic in the field of robotics. More and more applications are taking advantages common to all UAV types to achieve specific goals. The main reason is that no pilot is seated inside a UAV, therefore not threatened by possible dangers of a mission. UAVs are smaller and lighter than manned aerial vehicles. Thanks to that the aircraft drains less energy from its power source, and is also able to access much smaller spaces. Not only they can access spaces tighter than a human can move through but also are capable of automated data acquisition, processing, and even running AI (artificial intelligence) algorithms to navigate itself autonomously for example.

First known applications were military missions in danger zones, mainly surveillance. As times goes, parts and sensors for construction of these vehicles get less on price, and other industries are getting hands-on development for their custom applications. These days, UAVs are used for agriculture (terrain mapping), scientific (collecting statistics data), commercial (package delivery) and also for recreational purposes such as filming or drone racing. Applications similar to the military ones take place in commercial sector too. Private companies often use UAVs to guard their storages, office buildings, and other holdings.

A proper UAV application requires the involvement of various technology fields and brings a lot of challenging issues with itself. Even development of a simple recreational drone requires a good knowledge of physics to deal with the aerial vehicle's behavior, system control theory to be able to command the drone and radio-electronics for communication between an operator and the vehicle. Moreover, more advanced applications may require a certain level of AI implemented, because it may be instructed to follow waypoints in a map by its own for example. With „self-flying“ aircraft comes a problem of localization, because when a UAV needs to go somewhere, firstly, it must know where it is and if it is not already there. Moreover, with swarm missions (more UAVs cooperating on a task), we are in need of team mapping and communication features as we want each UAV to provide information to the others. In the end, there are the most advanced (scientific) applications, where developers must be able to incorporate custom modules with particular and often highly accurate sensing capabilities, for example, chemical sensors.

1.1 Goal description

One of the most important things for self-flied UAVs is localization. It is used for an aircraft's autonomous control and also as a source of information for an application managing the UAV's mission goal (localizing certain object, constructing a map, etc.). Localization starts with the integration of sensors into the system. The sensors can provide measurements of various physical quantities with various accuracies depending on the type of such sensor.

There are two types of problem which may come up during the development of the localization process. In the first place, we might need information about a physical quantity which is not directly measurable by any sensor present in the system. On the other hand, while having a sensor which can directly measure desired quantity, the sensor may provide noisy or low-frequency measurements which cannot be used for precise localization. In some cases, a developer can merely interchange the inappropriate sensor for a different one. However, in most cases, the precision is still not satisfied with the given application. That is the time where software solutions come in. There are post-processing methods which can achieve good estimates of real values of the physical quantities which are previously measured with low accuracy/frequency or are not measured at all. It is possible to acquire the non-measured quantities using a fusion of the data based on known physical principles. Low accuracy/frequency data problem is solved by use of filtration processes. General processes of fusion and filtration of data can be vastly improved when fitted right onto a specific application. Each of them has its pros and cons for the desired usage so their way of implementation should be picked and tuned wisely to achieve the optimal fit considering required specifics. Some of the methods can be used together to deliver even more precise results.

Nowadays many industrial applications call for objects inspections or mapping of an environment related to their business. Among others, the fields of usage may include agriculture, industrial and security. Agriculture field calls for construction of 3D models of structures or natural domains, crop monitoring, wildlife observation and others. Industrial applications make use of power lines inspection, storage containers maintenance, product pipeline monitoring, etc. Security involves holdings guarding, intruder tracking and so on. In all these cases, sense of the surroundings is essential. It can be achieved by use of various sensors as radar, lidar, or a camera. The process of identifying features in the environment and reacting to them correctly is inevitably dependent on self-localization.

The primary goal of the project is to develop a mapping system which will be capable of creating precise 3D maps of agricultural areas. Currently the system utilizes aerial vehicle with a lidar (light detection and ranging) sensor, which is capable of providing relative positions of scanned surfaces, an IMU (inertial measurement unit), which gives high rate information about the vehicle's orientation (three angles deviations relative to a given reference frame), and RTK GNSS (real-time kinematic global navigation satellite system) sensor, which provides the vehicle's position (translation relative to a reference point). The localization is crucial for the maps precision. As found in previous experiments, the accuracy of position and attitude sensors, mainly the GNSS one, is not sufficient enough for our target application. The goal of this work is to develop a solution which improves the accuracy of position and attitude information from data provided by IMU and GNSS sensor to be able to construct more precise 3D maps.

1.2 Knowledge assumption

To correctly understand the presented work, the reader is supposed to have at least basic knowledge in the fields of the differential calculus, kinematics and dynamics of a rigid body, linear algebra, system analysis/control theory (state space description, linearization, discretization) and statistics (probability of a random variable).

Part I

Problem overview

Chapter 2
Robot Localization

Robot localization is a process of determining robot's position and pose (orientation/attitude) respectively to the surrounding environment (or to the initial position and pose in case of odometry). Localization is one of the most fundamental competencies required by an autonomous system as the knowledge of its position and pose is necessary for real-time decisions about future actions and potential mapping process (creating a map of such environment). In this chapter, we will introduce some of the widely used localization approaches.[SG16] We will structure the description of them such that final selection for our application is justified and presented along with other similar methods in the same field. Only brief insight into the problematics will be given focusing mainly on advantages and disadvantages which are crucial for the final selection. Further on we will use abbreviation PP for "position and pose."

2.1 Localization approaches

In this section, we will compare representative existing localization methods to choose an approach which fits our application the most. We can split them into two major categories. The first one called odometry (or sometimes inertial localization) covers methods which utilize sensors onboard the vehicle and does not require any external support. They use motion sensing to estimate a robot's PP relatively to its previously known PP. That involves sensors such as revolution counters, angle sensors (synchro, resolver, rotary encoder), accelerometers, gyroscopes, magnetometers, pressure sensor, radar/lidar, camera from which a robot can acquire its linear and angular velocity vectors and update its PP over time based on these. It may also be able to determine a vehicle's pose based on measurements of known environmental properties (magnetic field, gravity). The second category, which can be called supportive, is any localization method which utilizes developed infrastructure of supportive devices in the environment. It can localize a vehicle in local (user-defined) coordinate frame or in standardized global coordinate system (WGS-84, PZ-90, ...)[ees14]. Such localization is called global localization or geolocation. This category of methods mostly uses the following sensors: GNSS sensors, radios, set of cameras, etc.

Besides the used sensors, we can identify some other major parameters for each approach. Mostly we are interested in the UAV application's properties such as operational area (is it outside/inside, how wide is it, is there weather influence, etc.) and knowledge of the environment. Its structure may be known prior to the localization process in the form of some map representing various features of the surroundings which can be used as a localization medium. Types of the representation vary depending on the corresponding application by important features like dimensionality of the coordinate space, required accuracy, detail and density, data volume limitations, etc. Further description is not within the scope of this work. Examples can be found in [YJC12], [Fai09] and [FZY$^+$14]. The knowledge may also be acquired during the process of localization (SLAM)[KTH15]. Sometimes the environmental knowledge is not utilized at all.

The following sections present four representatives of methods widely used for UAV localization. Note that not all existing approaches are presented as other individual cases may exist. We chose those related to UAV localization problem. Each section is structured such that: the first paragraph introduces the fundamental principle of the method, the second one lists sensors which are (should be) utilized in it, and the third one presents an example of an application in which the method is often used. Finally, advantages and disadvantages of it are covered to point out relevant specification used for the selection decision making.

2.1.1 Dead reckoning

One of the inertial localization method, dead reckoning, is a process of calculating the evolution of PP in time $t + \Delta t$ from known PP at time t and states changes (derivatives) measured during the time interval Δt. A wheeled ground vehicle going straight forward may use a sensor to read its wheel's velocity (angular, calculating peripheral) and therefore can determine its following position from known previous one. However, the wheel can slip on the surface. In every system, there is present such error caused by similar unpredictable and unobservable effects. That is why dead reckoning is subject to cumulative errors and tends to diverge over time from the real states of the system. The accuracy depends mainly on the presence of unobservable events which adds the errors.

This approach utilizes sensors which are either able to record difference of the desired quantity (angle sensors, revolution counters, etc.) or can determine the rate of change of particular physical quantity during the time interval (accelerometers, gyroscopes) and derive the difference using integration process.

Dead reckoning is often used together with a global localization module. Every-day example of its usage is navigation systems in cars. When used

GNSS sensor loses the ability to localize itself (car in a tunnel), the dead reckoning subsystem may temporarily take over to provide estimates of the current location until the global localization ability is restored. More advanced applications utilizing UAVs uses this method to get the information about PP with high rate required by auto-piloting systems. Because GNSS systems give position readings with low frequency (in order of units or tens of Hertz) the time gaps between the readings are filled by computed estimates to provide the information to control feedback of the auto-pilot. Nowadays, this approach is also subject of interest to pedestrians localization for personal usage [SF17] or used as a part of localization system of drive-less cars [AMY$^+$17].

This approach alone needs initial PP for the calculation of the others in time. The benefit of this approach is simpler hardware and software implementation. It is usable indoors and outdoors with sensors mounted onboard a vehicle. It is mainly used for real-time localization and is mostly used without the knowledge of an environment. However, in some cases, it also utilizes a map of the environment.[O'K06] The most crucial property of this approach is the cumulation of error over time.

2.1.2 Visual odometry

Let's introduce a second inertial method. This one is widely used in many variations utilizing various types of sensors and algorithms. Visual odometry is a process of determining PP change of a robot by analyzing captured images over time. In one moment a robot captures an image of the surroundings and compares it to the image captured one timestep ago. By finding so-called features (well distinctive points/contours/areas) in the images, it can determine a PP transformation of the camera which took those images, therefore, deriving the change of the robot's PP relatively to the surroundings. There are plenty of approaches for the features detection which are not within the scope of this work; examples may be found in [HKM13]. Moreover, when the environment is known (we have a geo-referenced model of it) global localization may be performed based on a comparison of captured images with the model.

This principle may utilize common RGB camera or other imaging sensors. We can assume cameras capture images in a different spectrum than the visible light. Usage of IR (infra-red) is the same except there is only one intensity value measured for each pixel (as opposed to colored image where three RGB values are recorded). Moreover, depth sensors or lidars[SZC$^+$17] can be utilized by creating different, 3D images of the environment. Finding features in these is a little bit different, but the principle remains the same. Review of visual odometry approaches can be found in [AMSI16].

A general example of an application using this approach is an application utilizing SLAM algorithm mentioned earlier. Specific example: A UAV uses a camera on-board to capture images of the surroundings and can construct a 3D model of it and localize itself inside it simultaneously. It uses the model to avoid obstacle and plan trajectory to be able to map the whole desired area. Distinguishable points in the area may be geo-referenced to make the localization and mapping processes referenced in a global coordinate system.

This method is usable both indoors and outdoors, used sensors and algorithm implementation should be adjusted to that. It utilizes only onboard sensors without any need for environmental interference (assuming sufficiently heterogeneous properties of the environment for proper feature detection meaning no large areas without distinct features). Another assumption is a sufficient features correlation between each two consecutive images. It is mostly implemented for real-time localization as for off-line trajectory reconstruction it would require storage of a significant amount of data. The knowledge of a map of the surroundings can greatly improve the performance but this method can be, and it is, used mainly for simultaneous localization and mapping (SLAM)[TUI17] which can localize without prior knowledge of the environment while effectively creating a map of it. The significant advantage of this method is that it does not have to be combined with any other and yet it is usable for various applications. The disadvantage is the high-performance requirement for extensive calculations done to retrieve the PP.

2.1.3 Triangulation

In localization, triangulation is a process of localizing a robot by forming triangles to it from known points (beacons) in the environment. It is the main representative for supportive localization category. Various sensors can be used to detect the beacons and measure either distance to them or angles under which they are perceived relative to the robot. With basic rules of trigonometry, the algorithm can determine a location of the robot relative to the beacon grid. There is an assumption that the robot can distinguish the beacons from each other (using image processing to detect signs, receiving identification radio signal from the beacons, etc.). When the global positions of the beacons are known, we can call this method a global localization (the robot indirectly localizes itself in a global coordinate system). Otherwise, it is called a localization in a local coordinate frame.

GNSS localization

GNSS is an acronym for “Global Navigation Satellite System.” A GNSS sensor can determine its position in a global coordinate system. The principle of triangulation is used to determine the position. The technique uses pseudo-random codes received from four or more satellites in Earth’s orbit to determine the sensor’s distance from each of them. Along with information about the satellites positions, the device can determine its position usually within a few meters. The more satellites are involved in this process, the more precise the localization is.[Nov18] A satellite is included in such solution if its signal to noise ratio and position over the horizon are both satisfying defined minimal values.

The sensor used for this is dedicated GNSS sensor. It consists of a board and an antenna. It may have various parameters including number of frequencies on which a signal can be received, what satellite systems it can use, if it records raw data or some processed solution, etc.

With the sensor's necessity of clear view at the sky (the satellites in orbit), this is an outdoor solution for localization. Besides real-time localization, it can be used for post-processing trajectory reconstruction because of the small volume of the data needed to be stored. The post-processing is widely used with the type of sensors which can record raw measurements. It is supposed that the satellite network itself is handling localization of the beacons (satellites). Therefore we assume this method to be the type which does not need any prior knowledge (all information about satellites is retrieved from the incoming radio messages). This method is not able to provide information about the attitude.

Wi-fi positioning system

Wi-fi positioning is a localization method which uses infrastructure of radio transceivers (beacons) statically placed in the operational area. A device on a vehicle communicates with all reachable beacons by radio messages. The mostly used localization approach is taking measurements of the received signals intensity (RSSI) and using the method of "fingerprinting"[YDCPC17]. As a vehicle can only localize itself relatively to the grid of beacons (hotspots), we call this a localization in a local coordinate frame. To perform global localization, the device onboard the vehicle must first acquire the information about the hotspots positions in the global coordinate system. However, in many robotic applications, there is no need for localization in a global coordinate frame, so only defined local frame is used. Some applications as personal localization with the help of mobile devices can use the developed infrastructure of wi-fi hotspots in urban areas to perform global localization. The hotspots are being geo-localized by correlation with GNSS system used by the mobile device. The accuracy of this localization method depends on the number of reachable beacons along with used signal processing method.

"Sensors" used for this method are radio transceivers. Some of them are placed in the environment, and usually, one is mounted on a vehicle which we want to localize. When the mobile device is combined with GNSS sensor, it can assist in geolocation process of the beacons in range. When creating own infrastructure of beacons grid, the beacons my be able to localize themselves relatively to each other and automatically create a 3D arrangement in which they are set.

This approach can be considered as an indoor analogy for GNSS localization (can also be used outdoors near the beacons) as it uses the same principle of triangulation. It is mostly being used as a real-time localization, and if used for global localization, it requires knowledge of the environment (locations of the beacons). When using the developed infrastructure, such knowledge is acquired via the radio communication (similar as with satellites in GNSS localization). The disadvantage is that the analysis of incoming signals often does not provide a very accurate estimate of position. This is caused mainly by the phenomenon of signals reflection in the environment. Moreover, it is not able to determine attitude at all. As it was said, the main advantage is the possibility of utilization of developed infrastructure.

Others

The principle of triangulation can be used with wide variety of sensors. If a robot can discover the beacons (camera) and acquire distances (depth sensor) or relative angles (image processing) it can localize itself relatively to the beacons grid.[YM05] The same goes for cameras or ultrasonic sensors grid established in the environment which can determine the position of the vehicle.

2.1.4 Measurements fusion

All of the previously stated methods were only elementary approaches ensuring derivation of 'position OR pose' or both with accuracy not sufficient enough for most of the advanced applications. Localization of a robot is commonly a subject of implementing more of the elementary methods together into a localization system. For example, a widely used outdoor solution is a fusion of position readings from a GNSS sensor and orientation measurements from accelerometers (measuring the vector of the gravitational pull) or magnetometers (using knowledge about the local magnetic field). There are plenty of combinations which can be done, each method brings in some new features (increasing the precision when correctly implemented) but also some limitations (heavier robot, costly sensors, more power drain, etc.). Mostly used approach for a UAV operating outdoors is dead reckoning based on linear and angular acceleration values continuously being corrected by position measurements from GNSS sensor and records from IMU (inertial measurements unit) detecting the UAV's attitude.

To further improve the process of localization, data processing can be used. Many of the physical quantities being measured by the localization sensors are not independent. Physical relations form a correlation between the measurements. For example, a robot's acceleration measured by accelerometers is highly correlated with velocity measurements which might originate from visual odometry and position readings from GNSS sensor. Software-aided processing (data filtering) of datasets consisting of readings from various sensors may vastly improve the estimate of the real PP.

These techniques may adjust a robot to be suitable for an outdoor and indoor application and may permit real-time as well as post-processing usage (depending on computing power and data storage available). There are applications which might utilize some prior knowledge if it is available and also incorporate sensors mounted in the environment. This approach is highly adjustable to fit the needs of a particular application.

2.2 Common data filtering

As it was stated in the goal description, we want to improve the localization of the UAV without any new hardware added. That is the reason why we have chosen a software solution for the improvement. We had already used primitive measurements fusion based on linear time interpolation of position information from GNSS sensor and attitude readings from IMU. Moreover, we used data filter built-in into the IMU which was providing attitude values based on measurements from multiple sensors. However, we want to develop similar but more precise filtering process by implementing filter fusing the information from both localization sensors on board (more sensors in the future).

There are plenty of filters already known which can somehow get more accurate information from noisy measurements, from systems of sensors, by integration of prior knowledge, etc. Note that we are not speaking about digital filters which apply to an individual signal's values and are analogies to analog filters just in discrete time samples. We talk about data filters which understand and can interpret the input values as physical quantities knowing relations between them. In the following subsections, we will cover three related types of such data filters which are widely used in the field of robotics.

2.2.1 Discrete Bayes filter

Recursive Bayesian estimation also known as Bayes filter is a discrete time filter which utilizes probabilistic approach for estimating an unknown probability density function of a desired phenomenon. It is recursive and uses incoming measurements as well as a mathematical process model for its estimates. In robotics, it is used as an algorithm for calculating the probabilities of multiple beliefs to allow a robot to get partial knowledge of its PP. The filter outputs probabilities of a robot being in various states (PPs) based on previously estimated states probabilities and incoming measurement. It does so by utilizing probability density functions of the sensors (probability that they will report certain value while in a certain state) and the system (robot) itself (probability of getting into the certain state while knowing the previous state).[Rub18] An example utilizing localization in topological domain is stated in [CHL14].

It comes out that the filter requires a lot of prior knowledge to be fully operational. Also, it can operate only over discretized intervals of values and states and uses a significant volume of data storage for all this information. This filter is mostly utilized in small domains of operation which have to be interpreted as discrete (for example grid maps), mainly used for 2D localization indoors in a known environment. Regarding these facts, this filter is not applicable to our application. But it sets a whole family of filters which are derived from this one.

2.2.2 Particle filter

Particle filter is a set of genetic algorithms based on Bayesian filtering. The process consists of estimating the internal states of a system while partial observations are available and random unobservable perturbations are present in the system as well as the sensors measurements. Particle filtering uses a genetic selection sampling approach with a set of particles to represent the posterior distribution of some stochastic process. The process model can be non-linear and initial state along with noise distributions may be arbitrary. This filter is even able to perform estimation without knowledge of a state-space model or state distributions. However, a map of the surroundings must be available. This approach covers the localization problem of unknown position in a known environment. It is based on assigning weights to individual particles in a set according to the incoming measurements and re-sampling them with respect to those weights to acquire an estimate from high particle density areas. More detailed description can be found in [Thr02].

This approach is not suitable for our application as it requires a limited area of operation and prior knowledge of the environment.

2.2.3 Kalman filter

Kalman filter (KF) can be marked as the continuous version of the previously described general Bayes filter and the special case of the particle filtering (sometimes referred in reverse). It has one crucial assumption that all probability functions utilized in the process are Gaussian (normal distribution). It is recursive and operates in discrete time moments as the general Bayes filter. Its difference is the possibility for continuous domain application as well as the simple definition of the probability functions resulting in lower data storage occupied. Because of our application in continuous domain representing a wide and unknown area of operation and possibility to accept the Gaussian assumption, we picked this type of filter to improve the localization of our UAV.

Kalman filter at its **basic form** is an estimator for linear systems. Its implementation is done using linear algebraic expressions. It uses series of noisy measurements along with the virtual evolution of the mathematical model to estimate state variables of the system for each timestep. It is done by using joint probability distribution over the variables. The big advantage of this method is the ability to make estimations of the states at a higher rate than the sampling frequencies of the sensor included. Moreover, it can also handle various sampling frequencies among multiple sensors. As said earlier, the filter estimates the states of the system. That means that it can estimate all states of the system, even those directly unobservable (not measured by any sensor), as long as they are part of the state space model provided. It is also necessary for the system to be fully observable (regarding system theory). In some implementations, also unknown input signals or noise signals entering the system can be estimated. For convenient functionality of the filter, it must be provided with good estimates of variances and covariances of the process noise and measurement noise. Practically the filter is reversing the addition of those noises into the system based on their most accurate description provided. It is an optimal state estimator in terms of minimizing the expected value of mean-squared error (MSE) between real value and the estimate. [LKDM17]

Implementation of this method requires the construction of a mathematical model of the physical system and reasonable estimates of the noise signals characteristics. Moreover, the system must be assumed to operate around a chosen working point (each state of the system should not diverge a lot from its defined working value). A significant advantage is a possible fusion of all measured physical quantities into one model where they influence each other, therefore in most cases bringing redundant information into the system which may improve the precision of the estimates. With the usage of calculated estimation of covariances in each step, the filter can effectively "switch" between sensors when some of them is reporting inaccurate measurements by assigning it lower weight. Kalman filter can smoothen noisy measurements.

There are also more advanced versions of Kalman filter. The first mentionable is **extended Kalman filter** (EKF) which is a nonlinear version of the basic KF and performs linearization about an estimate of the current mean and covariance. It uses a nonlinear model of a system (the functions must be differentiable), including state transition model (along with state/input relations) as well as observation model (relation between states and measurements). The process utilizes the Jacobian (partial derivatives) and evaluates it at each time step with current predicted states. Essentially it linearizes the nonlinear functions around the current state (estimate).[Jr.18a] Examples of usage can be found in [KA06] and [MDA07].

Unlike its linear counterpart, the EKF is generally NOT an optimal estimator. Also, if the initial estimate is wrong or the transition model derived incorrectly, it is much more prone to quick divergence because of the linearization. Another problem is that EKF tends to underestimate the actual covariance matrix and therefore risks becoming inconsistent in the statistical sense without the addition of correction noise. It is also computationally more demanding than the linear KF.

The **unscented Kalman filter** (UKF) is another advanced type of KF. The difference in implementation UKF and EKF is that UKF does not require the computation of Jacobian. That computation is not trivial, and sometimes the Jacobian is very difficult or even impossible to derive analytically. To evade this, the UKF only require the provision of functions that describe the system's transition model and measurement model. The UKF works on a principle of generating randomly distributed points and applying the system's models to them along with unscented transformation. This generation of points may, in some cases, be also computationally demanding. EKF and UKF are hard to compare without a specific case as they each perform very differently in various applications. The readers are referred to [WM00] and [CSH17] for further details.

There are many variations among the specific types of KF, among filters in general or among other data processing approaches. Each possible combination can be discovered to be useful for a specific case of usage, has its advantages and disadvantages. The stated overview should not and cannot give complete insight into all currently developed techniques.

We decided to implement the basic linear Kalman filter to improve the localization ability of our UAV based on the following reasons. It was experimentally proved that our UAV is moving beyond reasonable values of attitude angles (maximal deviations of $\pm 15°$) and we plan to incorporate this algorithm as a real-time solution onto the onboard computer's platform which has not sufficient computing performance for EKF or UKF in our case. Its detailed description and implementation will be stated in chapter 5.

Part II

Data processing

Chapter 3

Data acquisition

This chapter describes software subsystems dedicated to sensors data acquisition and its fusion into a final 3D model. It tightly follows up the work presented in my bachelor thesis [Tra16]. The previous version of the application was vastly rearranged to improve efficiency during development and ability to detect and solve issues without losing valuable data from testing flights. Supporting features were added, and imprecise program algorithms improved. From now on we will use terms "in-field" and "desktop" application/process/etc. In-field means it is done by the aircraft's onboard ARM computer in the operational area where a mapping takes place, and desktop stands for post-processing done using higher performance PC in an office.

Let's briefly sum up the state of the system after the work submitted in [Tra16]. In-field system acquires data from three sensors (lidar, IMU, differential RTK GPS) and uses the onboard computer's time along with estimated hardware line delays to pin timestamps to the incoming packets. It also vastly parses the incoming data and saves them into binary forms defined by our group's standard. Because of that, a significant amount of original information is dropped. IMU settings are statically set beforehand, using third-party desktop application. The error caused by wrong heading reported by IMU must be manually corrected by visual evaluation of a model which is time demanding and requires a skilled observer. Lidar's datagram packets using UDP (user datagram protocol) are assumed to be in correct order (regarding time of creation) as they arrive at the computer's input port, and no check for this is applied. All of these stated facts are considered issues to be resolved to improve the system's correctness and reliability.

Along with core features of the system, there is a parser for lidar's data packets and interpolation modules for lase points timestamp and laser firings azimuth reading. Moreover, developed mathematical algorithm used for the construction of a point cloud model (coordinate systems transformations), a module for correction of the points positions offsets caused by the mounting position of the GPS antenna, and LAS output module are present.

3.1 Used sensors

3.1.1 Lidar

Lidar stands for „Light Detection And Ranging“ or „Laser Imaging, Detection, And Ranging.“ The term was introduced as a portmanteau of words „light“ and „radar“ which was previously treated as an acronym for „Radio Detection And Ranging.“ However, no particular consensus on capitalization of the word „lidar“ exists. Used cases include many variations like „LIDAR“[Geo15], „LiDAR“[JHM18], „Lidar“[LZM17] or „lidar“[Met18]. In this thesis, the variant „lidar“ is used.

The sensor itself is based on a similar principle as a radar except it uses light beams instead of radio signals. The primary function is to measure distance in various directions. General lidar fires laser beams into its field of view (FOV) and once a laser pulse is fired, it travels through space until it hits solid object where it reflects. The corresponding sensor in the emitter/receiver pair detects an energy peak of the pulse reflected by the target object. The unit can determine the object’s distance based on the measured time between firing and detection and the known speed of light in the environment (time-of-flight principle). Additionally, the amount of energy contained in the returned pulse can be measured to determine the reflectivity of the object, hereafter identify some of its surface’s parameters.

A significant advantage of lidars against cameras used along with photogrammetry approach is that lidar can measure more energy peaks returned from one single laser pulse fired. That allows it to detect more surfaces covering up each other as long as all nearer surfaces than the farthest one are at least partially translucent (better be transparent)[Tra16]. Examples of utilizing this feature can be mapping terrain beneath treetops or riverbeds through a water mass. However, a lidar sensor is only capable of providing distance, and intensity measurements with position reported relatively to the sensor’s body frame. For an absolute localization of a scanned point, other sensor’s must be utilized. For this reason, the following two sensors are also utilized.

The lidar we use is VLP-16 from Velodyne company. It has FOV of 360° horizontally (it rotates around) and $\pm 15°$ vertically and fires 300 000 laser beams per second. Rotation frequency can be set between 5 Hz and 20 Hz and the effective range of measurement is from $0,5$ m to 130 m. Moreover, it is capable of retrieving two returns of a laser beam, the strongest one (according to energy) and the last one. The lidar is using UDP through Ethernet network to communicate with the computer. Laser measurements are being sent in data packets using reserved network port. The device has an option to connect external GNSS sensor directly to the lidar’s processor. With the provision of supported NMEA messages and PPS (pulse per second), the lidar also sends position packets on another port. The packet includes original NMEA message and parsed information from it. More importantly, when NMEA message and PPS are valid according to [Vel18] the internal clock of the device is synchronized with the UTC and states microseconds passed after the beginning of an hour.

3.1.2 RTK GNSS sensor

This sensor is responsible for localization in a global coordinate system (principle stated in 2.1.3). The system may either use spherical coordinates with the origin somewhere near the middle of Earth (various standards put it in slightly different places) or Cartesian coordinates with origin at any place (commonly Earth center or some GNSS ground station's position in a local area).

There are four global satellite systems currently operational. GPS (United State's Global Positioning System), GLONASS (from Russian abbreviation, managed by Russia), BDS (BeiDou Navigation Satellite System from China) and Galileo (created by European Union). Each has its own set of satellites. As the GPS is the oldest systems, every GNSS sensor is capable of localizing itself using the GPS. Because the need for the most satellites visible, more and more sensors start to incorporate the other systems too. Also, the satellites broadcast the code messages on more than a single frequency. Some of the sensors are capable of receiving multiple frequencies, therefore, increasing the chance to receive a message.

Moreover, special types of GNSS sensors exist. They were developed for applications requiring centimeter-level precision (land survey, hydrographic survey, UAV navigation, etc.). One such type is RTK GNSS sensor. RTK stands for Real-Time Kinematic and is a technology which uses carrier-based ranging instead of the code-based one. The device determines the number of carrier cycles between a satellite and the sensor and phase of the carrier wave at the signal's reception time. Cycle and phase measurements ensure more accurate estimation of the distance to satellite contrary to the code-based method. The calculated ranges still include errors (satellite clock and ephemerides shift, and errors caused by ionosphere and troposphere). To eliminate these errors, we use another device designated as a base station (BS) with a well-known position. It can provide corrections for the mobile device on a vehicle, also called rover in this case, in real time by radio or in post-processing via correction files.[Nov18] Another significant advantage of using the pair rover station and BS is that positions can be easily referenced in local coordinate system (Cartesian) which is used for more effective calculations in mapping systems.

Description of specific sensor used in the current setup will be covered in 3.3.1 further down in this chapter.

3.1.3 IMU

IMU stands for “Inertial Measurement Unit.” Basic IMU uses accelerometers and gyroscopes which can measure linear and angular accelerations correspondingly. Other physical quantities such as speeds and positions (both linear and angular) are most often determined by integration. The IMU is often capable of acquiring the initial attitude by measuring the direction of gravitational pull (acceleration). Moreover, units can be equipped with magnetometers which are helping to determine the unit’s attitude and an ambient pressure sensor which enhances the altitude measurements. Advanced units often incorporate GNSS sensor to provide direct position measurement (without integration). IMU can use software-aided improvements to provide more accurate information by fusing and filtering data from multiple sensors. Almost all IMUs available do not allow to configure their built-in filters, so a developer is unable to make custom adjustments. Luckily raw data from each sensor individually can be retrieved and own processing software can be applied which is the final goal of this work.

The IMU integrated into our system is 3DM-GX4-45 manufactured by Lord Microstrain company. Its components are enclosed in a durable and compact box equipped with mounting holes along with precision alignment holes. Moreover, its cable connector has a screw terminal for safe operation in a system with frequent vibrations. The whole device consists of three significant subsystems inside. The IMU subsystem itself (equipped with accelerometers, gyroscopes, magnetometers and ambient pressure sensor), GPS subsystem (provides position based on signals received from satellites by externally connected GPS antenna) and Kalman filter. The filter can provide all previously mentioned quantities with higher frequency and also can derive quantities such as velocities, absolute Euler angles and precision estimations (covariance matrices) for each measurement. The IMU subsystem is supported by a complementary filter so it can also provide absolute Euler angles. (Absolute is meant to be relative to the calibration state.)

3.2 System improvements

The following paragraphs cover robustness improvement of the module ensuring the measurements acquisition. The first topic covers correction of imprecise heading records provided by IMU. The second one presents rearrangement in sensors records handling and aligning with respect to time. Finally, we will introduce the implementation of sorting algorithm for lidar packets further improving the records alignment. The first two issues are tightly connected to the localization process involving IMU and GNSS sensor. The third one is related to the point cloud construction process.

3.2.1 Non-uniform heading

One of the most significant problems in the system was the fact that the IMU was reporting incorrect heading readings. The Euler angles taken from the Kalman filter was nearly correct but had a constant offset from the real values. The filter itself integrates the quantities very precisely but is dependent on initial vector given during its initialization. That was previously read from the IMU subsystem which reported an incorrect attitude. As seen in [Tra16], the problem caused the constructed point clouds to be unusable. A manual correction had to be done to at least somehow correct the model. This solution was unacceptable because it required the model to be constructed several times and simple trajectory reconstruction (post-processing localization) was not possible (not usable for Kalman filtering).

One idea was to use GPS readings to estimate the heading, assuming the vector of speed has the same direction as the heading of the aircraft. However, with the usage of a multi-copter, there is a problem with the fact that the aircraft can move itself to all sides independently of current heading direction. A possible solution would be adding acceleration (velocity) readings to get the correlation between those two sensors and estimate the axes misalignment. Another idea was to utilize second GPS antenna. With two measured positions of the antennas and knowledge of their mounting configuration on the aircraft, we would be able to get the heading vector. Those two solutions are demanding either on complex software solution or costly hardware adjustment, so we were looking for a simpler one.

After rigorous research, it was found that the IMU can initiate itself automatically without the need of a user to mediate the transfer of the initial vector. The process is not simple and requires the onboard system to send specific commands in specific order to the IMU. Previously, only simple inline implementation of hexadecimally represented commands was used in the source code to set up the IMU. This new initialization process strictly requires the communication to be reliable (acknowledged), ordered and error-checked. Therefore implementation of service library for the IMU was needed.

Because the whole project had utilized Java programming language for its implementation and such library was not provided in this language, I had to implement it by myself. The implementation was done as much general as it could be to make the future development easier in case of additional changes (addition of commands for example). Present stable version includes automated insertion of magic word bytes, calculation of payload length, fields lengths and checksums and setting correct fields structure based on provided descriptors. Moreover, it can recognize acknowledge messages for corresponding sent packets and therefore can arrange re-sends to ensure specific commands were executed with positive acknowledgments before continuing with a procedure. The programmer is informed during the whole application run about the status of the communication channel.

3.2.2 Sensors time synchronization

In the system, there are three sensors from which the data has to be merged to form final point cloud model (or two sensors in case of localization). For this process, we need to tell which measurements from different sensors belong to each other with respect to time alignment. We cannot use together measurements which arrived at the onboard computer at the same time as each link between a sensor, and the computer has different parameters, and the latency varies. To precisely align the data we want to work with timestamps recorded right in the moment of a measurement acquisition. That has to be done by the sensors devices. In our case the GNSS sensor links UTC with its measurements, IMU uses GPS time (GPST), and the lidar has its independent internal clock which starts from zero at the device's startup and counts microseconds in the format of unsigned long until it overflows and then repeats.

Previously, the computer's time of a packet's arrival was used along with estimated time of data transfer throughout the corresponding link. This approach was inaccurate and had tens of milliseconds errors. To be able to merge the measurements precisely we had to develop a solution such that we get the same time system to each sensor used. Regarding GNSS sensor, and IMU there is not a big problem as they both are capable of requiring GPST along with leap seconds from the satellites, therefore, can use both UTC or GPST as timestamps. For the lidar, the best and easiest solution is to utilize hardware connection with external GNSS sensor to achieve exact time synchronization to UTC.

The newly integrated GNSS sensor (described in 3.3.1) allows us to use its output serial port to provide NMEA messages of the desired type and PPS with defined duty cycle. We have set the configuration of the sensor according to [Vel18] and made a hardware serial connection between the lidar and the sensor. The internal clock of the lidar is synchronized when NMEA message with positive validity flag and valid PPS arrives. However, the system will switch back to the internal clock whenever the PPS lock is lost. For this reason, we implemented a subsystem which can parse the lidar's position packets and keep track of whether the timestamp is currently usable or not. An operator is also notified about the state in real time to know accurately in what time intervals the dataset will be usable for model construction. Details of this problematics are covered in the following section.

3.2.3 UDP packets timing

Because of the specification UDP has, it is not ensured that the packets incoming from the Ethernet link arrive in the order in which they have been captured. This disorder is also supported by usage of Ethernet switch in our rig. For a precise point cloud construction, we need to ensure that we only process valid lidar data with a correct timestamp. Therefore we assume a data packet to be valid only if it lies between two adjacent position packets (according to time) which both has valid NMEA message and PPS lock. Otherwise somewhere between them, the lock was lost, and the whole group of data packets must be considered unusable and dropped. To be able to evaluate each interval between two adjacent position packets we must first sort all received packets (data and position ones) onto one timeline.

Let's assume packets of both types are arriving at an input to the onboard computer. We need to implement a round-buffer which will fill itself with the incoming packets. An incoming packet will be inserted such that the buffer is ordered (according to the timestamp in headers of the UDP packets). We define time interval t_u after which we can be sure that no incoming packets can be inserted before the packet with timestamp $t - t_u$, where t is the current time. Packets with timestamp which satisfies $t_p < t - t_u$ are considered "confirmed". For such packets filtering process can now be applied. If we already have position packets for both ends of a data packets group we can classify the group as stated above. Algorithm illustration can be found in figure 3.1. Green position packets have valid NMEA massage and PPS lock, red ones not. Once a data packet is accepted or discarded it is removed from the buffer to make space for new packets, so the colored packets are displayed only for the sake of understanding.

3.3 New features

3.3.1 Single GNSS device

The previously used GNSS module was a differential RTK GPS system (Piksi RTK). Differential means it operates with a pair of devices and is giving a relative position of each one to another. That had utilized a second device marked as a ground station (GS) placed somewhere in operational area and radio transceivers to make the two devices communicate in real time. This particular devices provided only final processed data and did not allow many configurations.

Currently used GNSS sensor is RTK OEM-6 from NovAtel company. As opposed to the previously used device it is capable of working with all existing satellite systems (GPS, GLONASS, ...). The system is widely configurable and provides an open solution allowing modifications to fit into the localization system. It outputs data with raw measurements which can be processed in various ways depending on specific application. The sensor (along with appropriate antenna) is capable of acquisition of signals on two different frequencies which raises the number of reachable satellites (if a satellite's signal is weak on one frequency it can be sufficiently strong on the other one). It also improves the accuracy of measurement (if signals on both frequencies are strong enough, their distance measurements can be combined). To keep the advantages of RTK device while using just a single device we separately

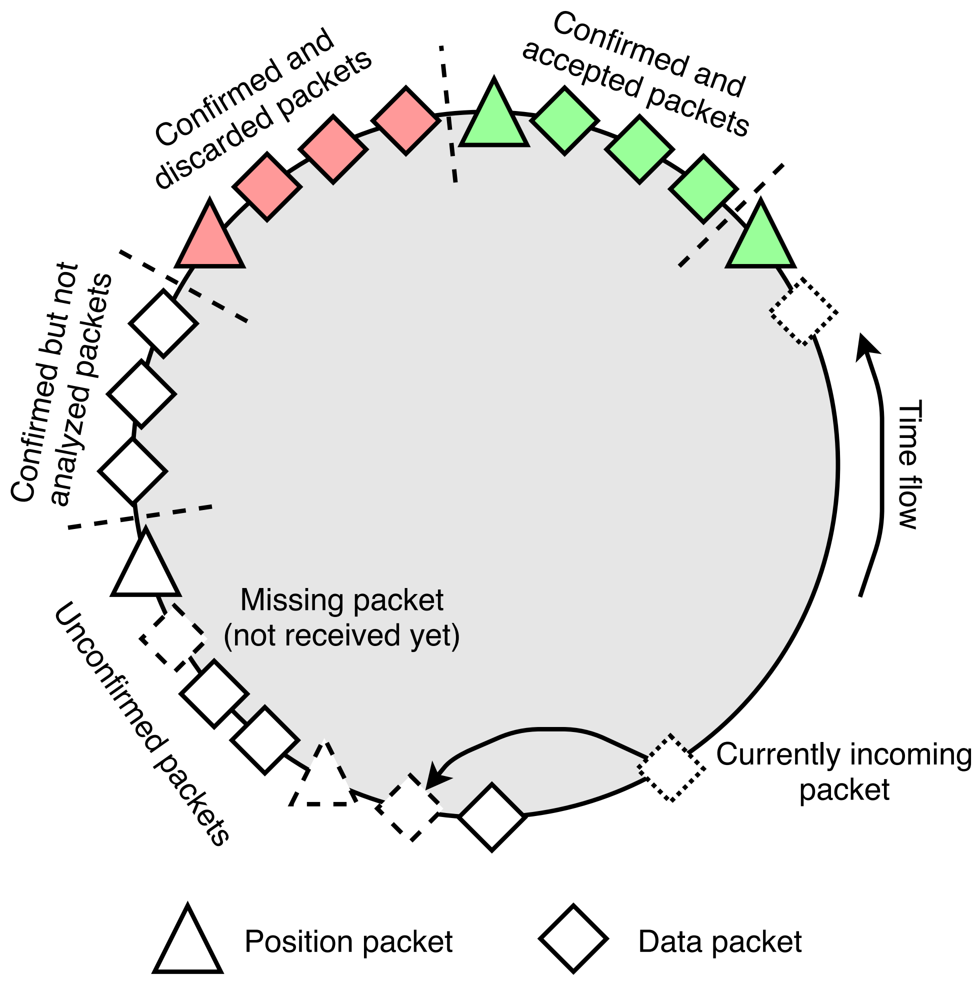

Confirmed and discarded packets
Confirmed and accepted packets
Confirmed but not analyzed packets
Time flow
Missing packet (not received yet)
Unconfirmed packets
Currently incoming packet
Position packet
Data packet

Figure 3.1: Round buffer implementation

acquire measurements from a static ground stations net around the globe. Such ground station has precisely targeted geodetic position and also continuously measures it using its GNSS sensor. From the differences between known and measured position, it can calculate climate and ionosphere errors corrections which are then used to correct the measurements done by the GNSS sensor onboard the mapping vehicle.

The correction process can be done in real time when there is some radio connection (GSM mostly) between the rover's sensor and the GS. This approach is stated to be less precise[Tů16]. On the other hand with some time delay (a couple of hours) a ground station can provide its processed measurements which generates more accurate position estimates when merged with the rover's measurements. Because of this, we acquire only raw data recorded by the rover's sensor and process them later in the desktop application.

3.3.2 Data handling

Mainly for the reason of the **program modularity**, we decided to redesign the system such that it has two main modules. The first module encloses only the very raw data acquisition, while the second part (consisting of smaller modules) handles the data processing. It is now possible to decide which data processing modules are used in-field and which in the desktop application. Usefulness of this fact is an independent recording of the measurements and preserving the recorded information in a raw and unchanged form. That allows executing multiple data processing algorithms over the same data set introducing various parameters and using various information from the data.

Lidar's UDP packets incoming through the Ethernet line are now directly dumped into a file, IMU's and GNSS sensor's packets on the serial lines are saved to files as binary streams. As mentioned earlier, the data also goes to parsers and information processors to provide the operator with **real time status updates** during the acquisition. The leading indicators include counters of packets recorded in past second and validity flags indicators to see if lidar data are being correctly timestamped. The GNSS sensor uses dedicated software library along with a CUI (character user interface) where a lot of status indicators are continuously updated (number of satellites, etc.). Moreover, all of the system's messages are logged into a file for future analysis.

Because most of the data processing modules were moved to the desktop application, **configuration file for the rover** was shortened. It includes IP addresses declarations, true/false switches for some utilities, baud rates, communication ports, etc. All these types of information are mainly used to establish working communication lines and data storage. They are constant for one rig setup. However, there is one set of parameters which affects the data itself. It is the mounting orientation of the IMU. Because of IMU's internal measurement process, this transformation cannot be done in post-processing, and the IMU has to know its changed reference frame before an acquisition. The service library described in 3.2.1 can set the parameters from the file automatically just before the actual mapping. Any other parameters influencing a model's construction were moved to the desktop configuration file.

As pointed out above, the parsing, processing, and calculation **algorithms** (point cloud creation and trajectory reconstruction) are fully configurable and **can be repeatedly used** over the same data without the data being lost. Variable parameters such as mounting position and orientation of the lidar, minimal and maximal range of the laser beams, various validity checks, etc. are included in configuration file dedicated for the desktop application as mentioned earlier. Moreover, the program implementation itself can be changed when a bug is found. The correction is then checked using the same input data again. This major rearrangement had conclusively split the actions of data acquisition and data processing, so they are fully independent as long as the format of files which mediate the exchange of raw data is standardized and preserved. Files conversion chain is illustrated in figure 3.2.

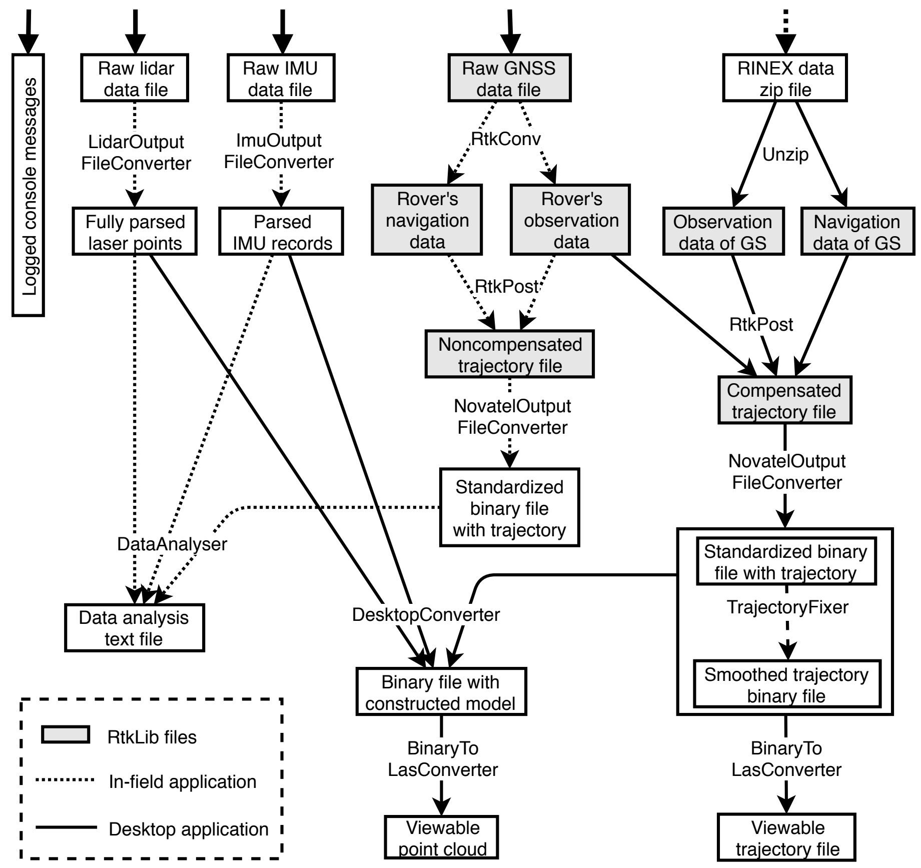


Figure 3.2: Files conversions flow diagram

The process of point cloud's construction was revised, and the desktop application's **code** refactored to be straightforward and **transparent for newcomers** to the project. Moreover, the executable parts of the program were implemented as general as possible to be able to operate over files in arbitrary file tree structure. All types of data had its core file name and file suffix assigned. At this point, the Java implementation consists of multiple executable modules which each is responsible for converting one file type into another by doing required actions over the data inside. This modular approach helps all desktop processes including the filtering to be done effectively and not to involve unnecessary parts of the program.

Above the Java executable parts there is a developed **Python application** which controls the flow of the data from the very beginning to the end of a model's creation. It creates predefined folder structure and constructs model requested by a user. During the model creation, it checks whether some needed files already exist and if they were created using the same configuration as provided. If so, they are not recreated again to save time and the already existing files are used instead. The application is also capable of executing group conversion to create all available models from data of one whole day for example. The user is kept informed during the whole process knowing in which phase it is and approximately how long it would take to finish. The folder structure is shown in 3.3 and the Python application's state machine in 3.4.

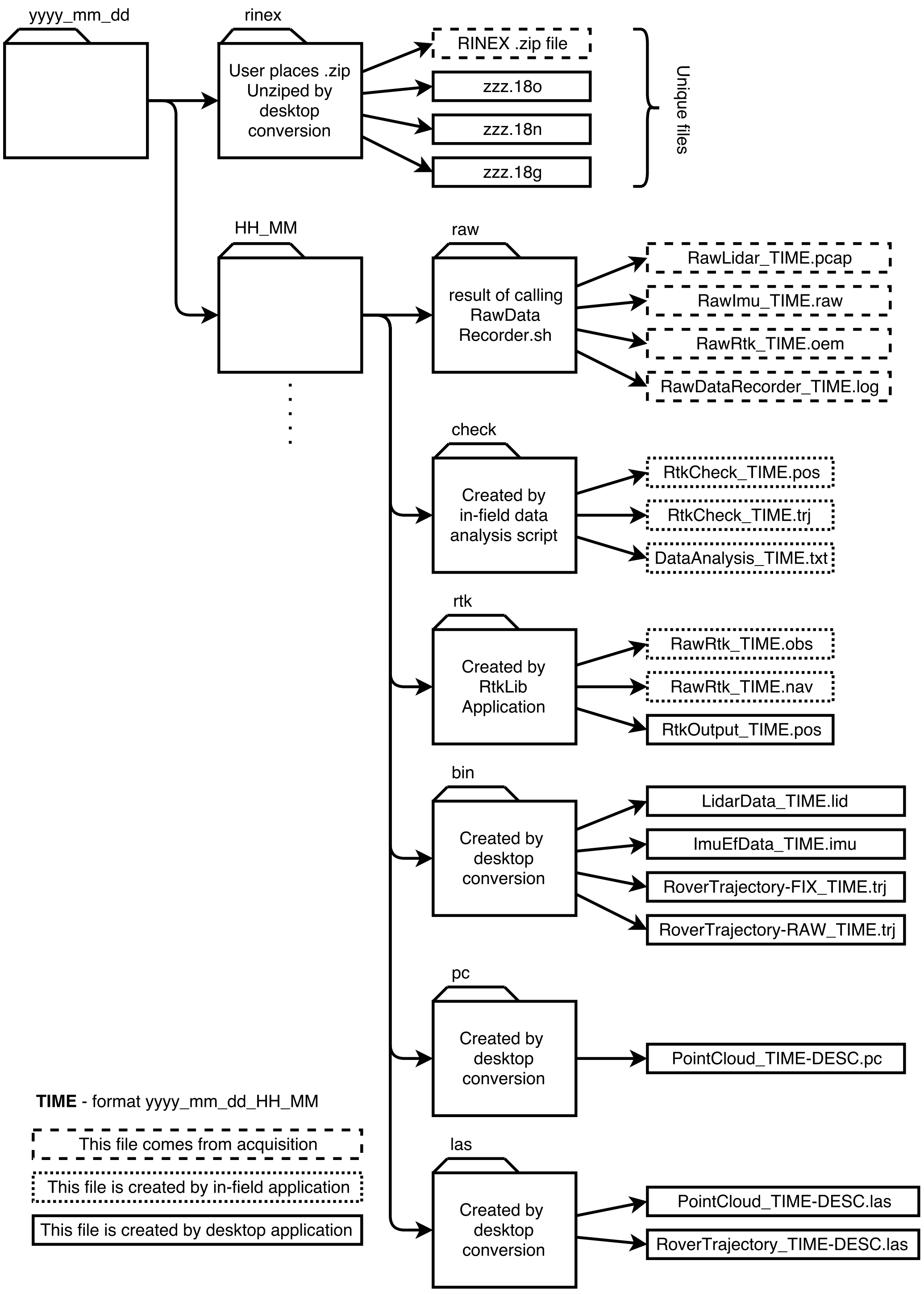
To make the creation of a model available to a common user. A **graphical user interface** (GUI) was developed and implemented into the Python application. A user can now create, store and load configuration files and make conversions without knowing the inner folder structure or whole file names. Matching of files stated in figure 3.4 works on a principle of date and timestamp included in the files names. To support this methodology, also in-field data acquisition and analysis (described further down in 3.3.4) programs follow the same files structure/names standard. The GUI window is shown in 3.5.

3.3.3 In-field GUI

As with the desktop conversions, we also wanted the data acquisition process to be usable by a common user. For this reason, another GUI application was developed (by the author's colleague). It involved automated command sequences executed via buttons, console messages view and the aircraft's position and attitude. Remote console operation was already done previously, and this application only enclosed it in a user-friendly environment. However, until this moment we hadn't had remote access to the aircraft's position/attitude measurements. Because there is no need for high precision and the GNSS sensor on-board is operated by third-party software, we decided to get all of the information just from the IMU. That involved real-time reading of its serial port along with its dumping into a file. The newly developed features include partial parsing of needed data onboard the aircraft and sending them to remote computing machine using server/client communication architecture. Data-buffering and reconnection after connection interruption were implemented too. In short, this feature bridges the serial communication line over a TCP connection and parses the data.

3.3.4 In-field statistics

When we want to create a map of some certain area or reconstruct the trajectory of a flight, we need to be sure we acquired enough usable data to do so. Otherwise, we would need to repeat the flight. Therefore we need an analysis utility to be able to tell that in the field. For this reason, part of the data parsing is done right after the flight, and another part of the application was developed. The new program part goes through each sensor's raw data file separately and counts number of records, number of valid records, minimal, maximal and average value, and so on. The most important feature is valid data intervals matching which can tell during what percentage of the flight time the data from all three sensors (two in case of trajectory reconstruction) was valid. Only in those fully overlapping intervals, a point cloud (trajectory) can be constructed in the post-processing calculations. The raw data remains intact during this process.



yyyy_mm_dd
rinex
User places .zip Unziped by desktop conversion
RINEX .zip file
zzz.18o
zzz.18n
zzz.18g
Unique files
HH_MM
raw
result of calling RawData Recorder.sh
RawLidar_TIME.pcap
RawImu_TIME.raw
RawRtk_TIME.oem
RawDataRecorder_TIME.log
check
Created by in-field data analysis script
RtkCheck_TIME.pos
RtkCheck_TIME.trj
DataAnalysis_TIME.txt
rtk
Created by RtkLib Application
RawRtk_TIME.obs
RawRtk_TIME.nav
RtkOutput_TIME.pos
bin
Created by desktop conversion
LidarData_TIME.lid
ImuEfData_TIME.imu
RoverTrajectory-FIX_TIME.trj
RoverTrajectory-RAW_TIME.trj
pc
Created by desktop conversion
PointCloud_TIME-DESC.pc
las
Created by desktop conversion
PointCloud_TIME-DESC.las
RoverTrajectory_TIME-DESC.las
TIME - format yyyy_mm_dd_HH_MM
This file comes from acquisition
This file is created by in-field application
This file is created by desktop application

Figure 3.3: The application's folder structure

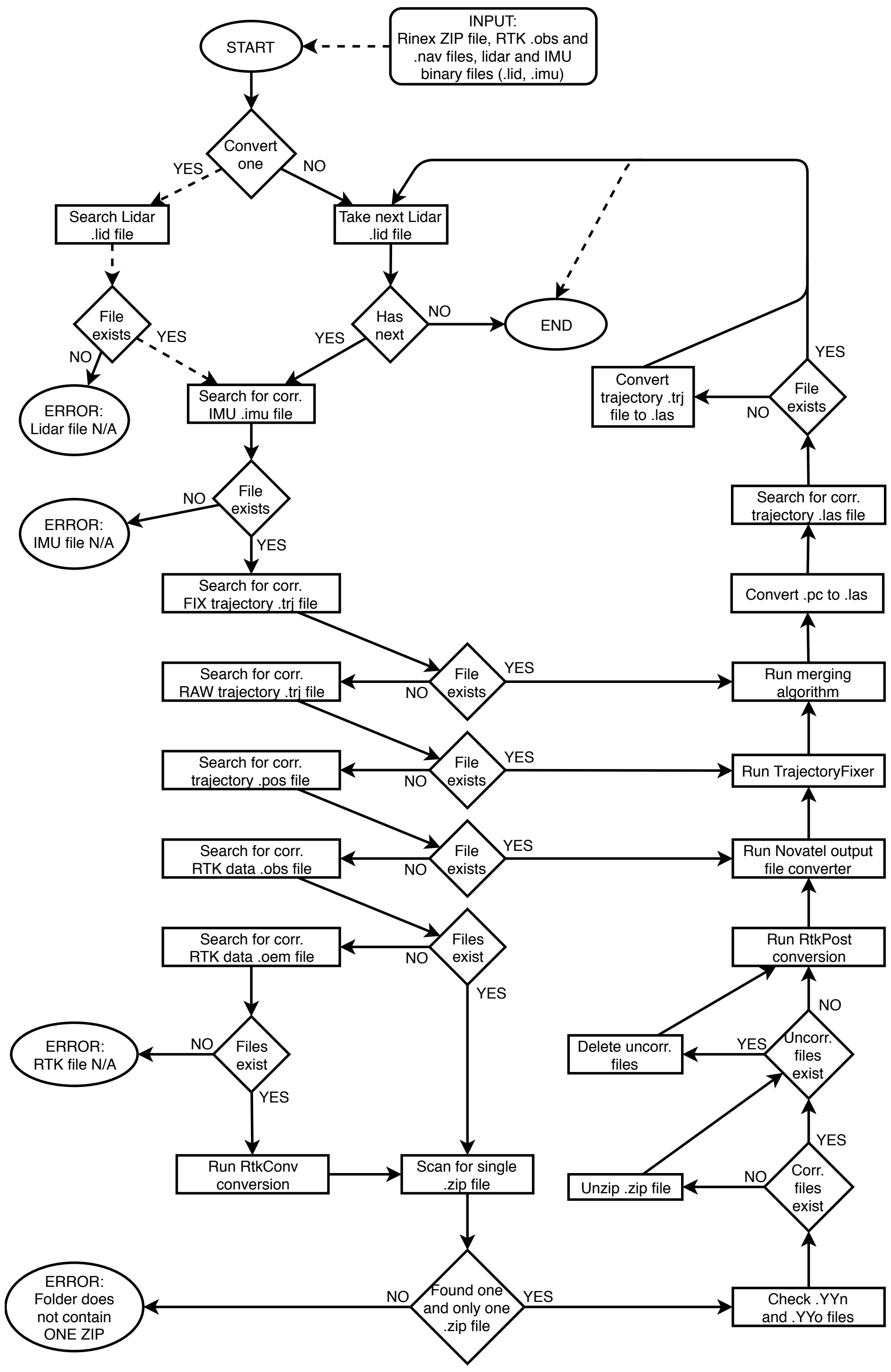
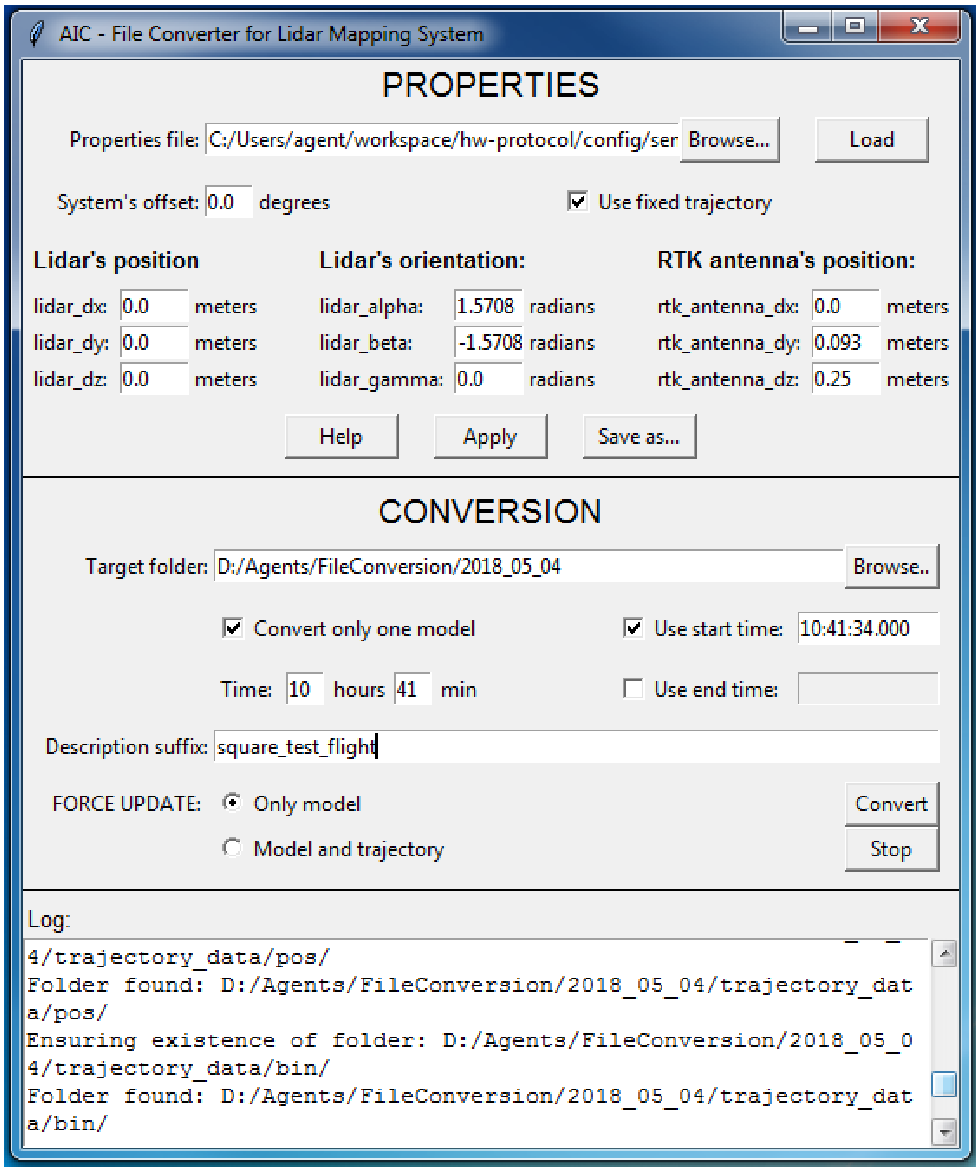

INPUT: Rinex ZIP file, RTK .obs and .nav files, lidar and IMU binary files (.lid, .imu)
START
Convert one
YES
NO
Search Lidar .lid file
Take next Lidar .lid file
File exists
YES
NO
ERROR: Lidar file N/A
Has next
YES
NO
END
Search for corr. IMU .imu file
File exists
NO
YES
ERROR: IMU file N/A
Search for corr. FIX trajectory .trj file
File exists
YES
NO
Search for corr. RAW trajectory .trj file
File exists
YES
NO
Search for corr. trajectory .pos file
File exists
YES
NO
Search for corr. RTK data .obs file
Files exist
YES
NO
Search for corr. RTK data .oem file
Files exist
NO
YES
ERROR: RTK file N/A
Run RtkConv conversion
Scan for single .zip file
Found one and only one .zip file
NO
YES
ERROR: Folder does not contain ONE ZIP
Check .YYn and .YYo files
Corr. files exist
NO
YES
Unzip .zip file
Uncorr. files exist
YES
NO
Delete uncorr. files
Run RtkPost conversion
Run Novatel output file converter
Run TrajectoryFixer
Run merging algorithm
Convert .pc to .las
Search for corr. trajectory .las file
File exists
NO
YES
Convert trajectory .trj file to .las

Figure 3.4: Models creation flow diagram



Properties file: C:/Users/agent/workspace/hw-protocol/config/ser

System's offset: 0.0 degrees

[✓] Use fixed trajectory

Lidar's position

lidar_dx: 0.0 meters
lidar_dy: 0.0 meters
lidar_dz: 0.0 meters

Lidar's orientation:

lidar_alpha: 1.5708 radians
lidar_beta: -1.5708 radians
lidar_gamma: 0.0 radians

RTK antenna's position:

rtk_antenna_dx: 0.0 meters
rtk_antenna_dy: 0.093 meters
rtk_antenna_dz: 0.25 meters

Target folder: D:/Agents/FileConversion/2018_05_04

[✓] Convert only one model

[✓] Use start time: 10:41:34.000

Time: 10 hours 41 min

[ ] Use end time:

Description suffix: square_test_flight

FORCE UPDATE: [✓ Only model] Model and trajectory

Log:

```
4/trajectory_data/pos/
Folder found: D:/Agents/FileConversion/2018_05_04/trajectory_dat
a/pos/
Ensuring existence of folder: D:/Agents/FileConversion/2018_05_0
4/trajectory_data/bin/
Folder found: D:/Agents/FileConversion/2018_05_04/trajectory_dat
a/bin/
```

Figure 3.5: GUI for point cloud construction

Chapter 4
Mathematical model

In this chapter, we introduce detailed derivation of a mathematical model of the hexacopter's dynamics using Newton's and Euler's law. In the first place we propose coordinate systems which are used during the whole filtration process, then structure, physical parameters and basic behavior of the vehicle are described. After that derivation of the vehicle's state space model is deduced and linearized model constructed. The behavior of the model is shown on system's responses during elementary actions performed by the hexacopter.

4.1 Coordinate systems

There are more options when it comes to the choice of coordinate system in which we want to operate. There is no strict standard given but there are some cases which are being widely used and are also implemented in science community libraries for various platforms and programming languages. In general, an arbitrary coordinate system can be chosen as it only affects the inner system's representation and with use of correct transformations the input and output formats are intact.

Firstly, we want to stick to right-handed systems which satisfy orientation of axes such that $X \times Y = Z$. We need two coordinate systems, one bound to the frame of the vehicle (mobile) and the other one bound to an environment (local ground). The fixed coordinate system also called inertial (ICS), is a system where the first Newton's law is considered valid. For ICS we are going to use North-East-Down (NED for short) system with axes marked as P, Q, R correspondingly. We want the PQ plane to be parallel to the local ground. Therefore the origin of ICS is supposed to be as near as possible to a site where the vehicle operates to comply the Earth's surface approximation by a plane. The mobile frame will be called VCS (Vehicle's Coordinate System). To respect the designation and orientation of RPY (roll, pitch, yaw) angles to axes (according to used IMU's documentation) we use X axis pointing to the front of the vehicle, axis Y pointing to the right and axis Z going downwards (FRD for short). The illustration can be seen in figure 4.1.

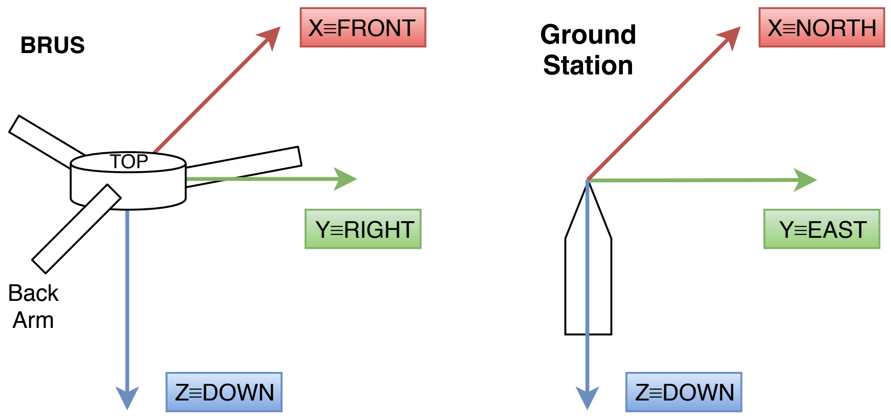


Figure 4.1: Used coordinate systems: VCS and ICS

4.1.1 Gimbal lock

When introducing a coordinate system, we must also consider its properties related to rotational transformations and what calculus are we going to use to apply them correctly. Considering transformation using RPY angles, we know that we have to conduct three separate transformations chained one after another. There are specific cases of an object's orientation which cause trouble, and we call them gimbal lock.

A gimbal is a (virtual) ring that is suspended so it can rotate about an axis. Gimbals are typically nested one within another to accommodate rotation about multiple axes. They appear in gyroscopes and inertial measurement units to allow the inner gimbal's orientation to remain fixed while the outer gimbal suspension assumes any orientation[Str08]. "Some coordinate systems in mathematics behave as if there were real gimbals used to measure the angles, notably Euler angles. For cases of three or fewer nested gimbals, gimbal lock inevitably occurs at some point in the system due to properties of covering spaces (described below)."[Wik18]

A gimbal lock is a loss of a degree of freedom (DOF). Not to understand wrong, there is no physical lock applied onto the object. A gimbal lock happens when two of the virtual gimbals align, and there are fewer than $n-1$ gimbals left, where n is the dimension of the system. There is a gimbal lock configuration in each system which uses only n virtual gimbals for its calculations. A little demonstration of the problem follows. Let's say we have a telescope on a tripod and it uses an azimuth-elevation system. We track flying object which approaches from the east and is moving right above our position. As it is right above our tracking system, it sharply turns towards the south. Because we have achieved maximal elevation and aligned the two gimbals, we are unable to perform smooth tracking, because fast azimuth change is first needed to acquire the target again[Pop98]. Considering an aircraft using RPY angles system, the gimbal lock configurations are when the pitch is either $\pi/2$ or $-\pi/2$. In this case, roll and yaw gimbals are aligned.

That is a big problem for automated systems as the sensors are unable to distinguish the correct angles. When the aircrafts body rolls, is it roll change or yaw change? The same problem applies to regulators which should navigate the aircraft to some given orientation from a gimbal lock configuration. This problem is usually solved by the addition of fourth virtual gimbal by using quaternions during calculations[CPN16]. In our case of multi-rotor UAV, this situation will never happen. The UAV is not performing any acrobatic maneuvers and therefore keeps its roll and pitch angles within low values. Moreover, it is equipped with parachute system which triggers when one of the two angles goes out of range $\pm\pi/4$.

4.2 Used vehicle

4.2.1 Preliminar notions

Before deriving the mathematical model initial assumptions and neglection of some present physical rules must be introduced to simplify the process. We assume the vehicle to be rigid body therefore no deformation of it is taken into consideration. Center of the VCS is placed into the geometrical center of the body where also barycenter (center of mass) is assumed to be located. Axes of VCS are identical with the principal ones, therefore we suppose the inertia matrix to be symmetric and diagonal[Fit11] (the vehicle is also assumed to be inertia-symmetrical around axis Y)

$$\mathbf{I} = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix} \tag{4.1}$$

4.2.2 Behavior

BRUS is a coaxial hexacopter with Y-architecture. One arm points straight backward and two arms point forwards and sideways (front-left, front-right). Angles between each pair of arms are the same, $2\pi/3$. Upper rotors rotate clockwise, the lower ones counter-clockwise (when looking from the top). The blades are always shaped in a way in which they cause negative thrust along axis Z (upwards). Therefore blades on odd and even rotors are diametrically opposite. From now on we will use motors numbering according to figure 4.2.

The architecture preserves the main principle of multi-copters that all propellers are mounted horizontally and being equipped with fixed-pitch blades. It differs from widely known quadcopter mainly in thrust actions needed by individual motors to perform the desired angular change. The reason is the arm located along an axis of the vehicle. Let's assume the UAV is hovering in

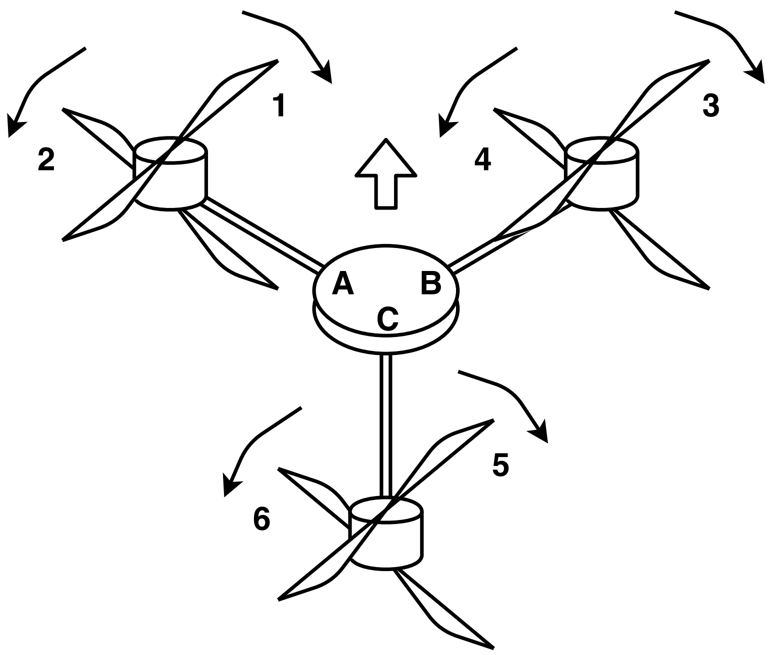


Figure 4.2: BRUS - view from the top, the front of the drone is up

mid-air: each propeller causes the same amount of angular momentum to the aircraft, the UAV is oriented horizontally, and the lift is fully compensating gravitational force. Few examples of elementary attitude changes follow.

To achieve roll right, we want propellers on arm A to accelerate (preserving same speed within the pair) and propellers on arm B to decelerate by the same value. To perform heading (yaw) change clockwise (looking from the top) we make the even propellers to spin faster and the odd ones to go slower, therefore maintaining the same lift. The moment of inertia of the rotors will cause non-zero angular momentum along the UAV's vertical axis. For pitch down, we want to accelerate the rotors pair on arm C and/or decelerate propellers on arms A and B (all four by the same value). The examples are illustrated in figure 4.3.

Another minor difference of this frame type is that each motor does not have the same torque effect on each axis of the aircraft. While changing yaw it only matters the difference of sums of thrusts between even and odd propellers, during roll arm propellers on arm C are not getting involved and the pairs on arms A and B have equal influence. With pitch, however, we must consider the projections of motors distances from the aircraft's center onto the axis Y. A single propeller on arm A or B causes less torque around axis Y than propeller on arm C with the same thrust. The related calculation will be stated further below.

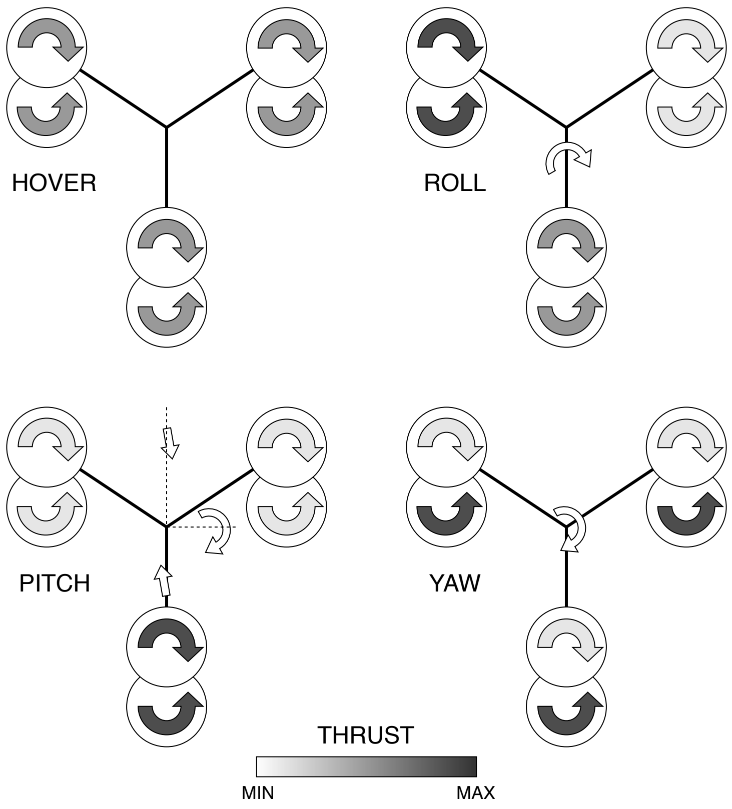


Figure 4.3: BRUS - Elementary attitude changes

4.2.3 Degrees of freedom

This topic needs to be discussed clearly. As we assume rigid body without deformations, the number of degrees of freedom (DOFs) for the physical object is 6. 3 for linear movement and 3 for angular movement. No constraints are given by the surroundings of the aircraft, so there is no loss in DOFs. However, we can also define controllable DOFs. Those are DOFs which can be directly controlled by inputs of the system; the others are derived. Without further proof, let's state that the number of controllable DOFs is 4, because of present holonomic constraints between the propellers. A practical proof was given in 4.2.2.

4.2.4 Physical parameters

The following constants defining the vehicle's physical parameters were acquired from its technical specifications. The operational weight of the drone (including battery) is $M = 7\ kg$. Distance between the center of the drone and an axis of each motor is $R = 0,6\ m$. Moment of inertia is described by vector $I = [I_{xx}\ I_{yy}\ I_{zz}] = [0,65\ 0,65\ 1,33]\ kg \cdot m^2$, where its values are the diagonal elements of the inertia matrix (the rest is zero). The UAV's air resistant areas for corresponding axes are $S = [0,15\ 0,15\ 0,21]$. Coefficient $k = 0,0974$ represents ratio between motors torque and thrust $T_i = k \cdot \tau_i$. We assume the gravitational acceleration to be $g = 9,81\ ms^{-2}$ and air density $\rho_0 = 1,2256\ kg \cdot m^{-3}$ at sea level and temperature of 15 $°C$.

4.3 Non-linear model

The motion of the body can be divided into two parts: linear motion (translation of the barycenter) and angular motion (rotation around the barycenter). As with all other rigid bodies without any kinematic or dynamic constraints, the body has 6 degrees of freedom. There are three possible elementary translations of the barycenter, one along each axis, and three elementary rotations of the body around the barycenter, one about each axis. The control of DOFs is implemented by adjusting rotational speeds of individual motors.

We want to provide a mathematical model of the 3D motion of a rigid body exploiting Newton's and Euler's equations. For the simplest model of a multi-copter, only three states would be enough to describe angular velocities caused by propellers thrusts. Moreover, we want to be able to observe linear velocities in the vehicle's frame. As the forces generated by propellers cause accelerations (angular and linear), each of position/attitude state itself forms a second-order system. Therefore we end up with a 12-dimensional system. Integrated position and attitude cannot be expressed in the mobile reference system. Because IMU and GPS sensors measures quantities in ICS, we introduce the final integrated position/attitude in the ICS. On the other hand, accelerations are caused by force mainly bound to vehicle's frame (thrust, friction force). Therefore we want the velocities (linear and angular) states to be referenced in VCS. Further on we will use superscript G (as ground) to label vectors referenced in ICS.

4.3.1 Euler angles

The Euler angles are used to describe the orientation of a rigid body in three-dimensional Euclidean space. They can also be used to perform transformation of vectors from one reference frame into another. In aviation the angles are typically assigned to roll (ϕ), pitch (θ) and yaw (ψ) which are related to axes X, Y and Z of the mobile frame respectively. To avoid redundancies and preserve unambiguous description of any configuration, the following ranges are used: $\phi \in [\pm\pi]$, $\theta \in [\pm\pi/2]$, $\psi \in [0, 2\pi]$. Of course the gimbal lock problem is still present. During reconstruction of such an orientation or during transformation between reference frames, strict compliance of order is required. Euler angles represent three elemental rotations. We are going to use ZYX (yaw, pitch, roll) order[SSVO09]. This means that rotation from one system to another starts with rotation around Z axis first, then around new axis Y' and finally around X". The elementary rotations are described by the following matrices:

$$\mathbf{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c(\phi) & -s(\phi) \\ 0 & s(\phi) & c(\phi) \end{bmatrix}, \tag{4.2}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} c(\theta) & 0 & s(\theta) \\ 0 & 1 & 0 \\ -s(\theta) & 0 & c(\theta) \end{bmatrix}, \tag{4.3}$$

$$\mathbf{R}_z(\psi) = \begin{bmatrix} c(\psi) & -s(\psi) & 0 \\ s(\psi) & c(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{4.4}$$

where $s(\alpha)$ and $c(\alpha)$ stands for $\sin(\alpha)$ and $\cos(\alpha)$ respectively. Assuming the IMU gives us angles which led to transformation from ICS to VCS, we want to do reverse transformation, which leads us to the final compact rotation matrix

$$\mathbf{R}_{zyx}(\phi,\theta,\psi) = \mathbf{R}_z(\psi) \cdot \mathbf{R}_y(\theta) \cdot \mathbf{R}_x(\phi)$$

$$\mathbf{R}_{zyx}(\phi,\theta,\psi) = \begin{bmatrix} c(\theta)c(\psi) & s(\phi)s(\theta)c(\psi) - c(\phi)s(\psi) & c(\phi)s(\theta)c(\psi) + s(\phi)s(\psi) \\ c(\theta)s(\psi) & s(\phi)s(\theta)s(\psi) + c(\phi)c(\psi) & c(\phi)s(\theta)s(\psi) - s(\phi)c(\psi) \\ -s(\theta) & s(\phi)c(\theta) & c(\phi)c(\theta) \end{bmatrix}. \tag{4.5}$$

4.3.2 Kinematic model

Linear state coordinates introduced in 4.1, can be summarized into vectors

$$\mathbf{v} = [v_x,\ v_y,\ v_z]^T,$$
$$\mathbf{v}_G = [\dot{p},\ \dot{q},\ \dot{r}]^T.$$

The relation between these two vectors is given by

$$\mathbf{v}_G = \mathbf{R}_{zyx}(\phi,\theta,\psi) \cdot \mathbf{v}. \tag{4.6}$$

Moreover we introduce states related to rotation motion. Euler angles are referenced in ICS and angular velocities are measured with respect to axes of VCS:

$$\boldsymbol{\omega} = [\omega_x,\ \omega_y,\ \omega_z]^T,$$
$$\boldsymbol{\omega}_G = [\dot{\phi},\ \dot{\theta},\ \dot{\psi}]^T.$$

According to [LBD+09], the relation between these states is given by the following equation and matrix for angular transformations:

$$\boldsymbol{\omega}_G = \mathbf{T}_{xyz}(\phi,\theta) \cdot \boldsymbol{\omega}, \tag{4.7}$$
$$\mathbf{T}_{xyz}(\phi,\theta) = \begin{bmatrix} 1 & s(\phi)t(\theta) & c(\phi)t(\theta) \\ 0 & c(\phi) & -s(\psi) \\ 0 & \frac{s(\phi)}{c(\theta)} & \frac{c(\phi)}{c(\theta)} \end{bmatrix}, \tag{4.8}$$

where $t(\theta) = \tan(\theta)$. Finally the kinematic model of the hexacopter is described as

$$\begin{aligned}
\dot{p} &= v_x[c(\theta)c(\psi)] + v_y[s(\phi)s(\theta)c(\psi) - c(\phi)s(\psi)] + v_z[c(\phi)s(\theta)c(\psi) + s(\phi)s(\psi)] \\
\dot{q} &= v_x[c(\theta)s(\psi)] + v_y[s(\phi)s(\theta)s(\psi) + c(\phi)c(\psi)] + v_z[c(\phi)s(\theta)s(\psi) - s(\phi)c(\psi)] \\
\dot{r} &= v_x[-s(\theta)] - v_y[s(\phi)c(\theta)] + v_z[c(\phi)c(\theta)] \\
\dot{\phi} &= \omega_x + \omega_y[s(\phi)t(\theta)] + \omega_z[c(\phi)t(\theta)] \\
\dot{\theta} &= \omega_y[c(\phi)] + \omega_z[-s(\phi)] \\
\dot{\psi} &= \omega_y[\frac{s(\phi)}{c(\theta)}] + \omega_z[\frac{c(\phi)}{c(\theta)}].
\end{aligned} \tag{4.9}$$

4.3.3 Dynamic model

Now its time to exploit the two main laws applied to a rigid body. Newton's law for total force acting on a rigid body states

$$m(\boldsymbol{\omega} \times \mathbf{v} + \dot{\mathbf{v}}) = \mathbf{F}, \tag{4.10}$$

where m is the total mass of the body and vector $\mathbf{F} = [F_x\ F_y\ F_z]$ stands for sums of forces in direction of corresponding axis. Symbol "$\times$" represents cross product of two vectors. Euler's equation introduces total torque applied to a rigid body

$$\mathbf{I} \cdot \dot{\boldsymbol{\omega}} + \boldsymbol{\omega} \times (\mathbf{I} \cdot \boldsymbol{\omega}) = \boldsymbol{\tau}, \tag{4.11}$$

where **I** is inertia matrix of the body and vector $\boldsymbol{\tau} = [\tau_x \ \tau_y \ \tau_z]$ stands for sums of torques applied to corresponding axis. Symbol "$\cdot$" represents standard matrix/vector multiplication. Based on assumption 4.1 we will denote

$$I_x = I_{xx}, \quad I_y = I_{yy}, \quad I_z = I_{zz}.$$

According to 4.10 and 4.11 we get motion differential equations for the body

$$\begin{aligned}
F_x &= m(\dot{v}_x + \omega_x v_z - \omega_z v_y) \\
F_y &= m(\dot{v}_y + \omega_z v_x - \omega_x v_y) \\
F_z &= m(\dot{v}_z + \omega_x v_y - \omega_y v_x) \\
\tau_x &= I_x \dot{\omega}_x + I_z \omega_z \omega_y - I_y \omega_y \omega_z \\
\tau_y &= I_y \dot{\omega}_y + I_x \omega_x \omega_z - I_z \omega_z \omega_x \\
\tau_z &= I_z \dot{\omega}_z + I_y \omega_y \omega_x - I_x \omega_x \omega_y
\end{aligned} \tag{4.12}$$

Rewriting 4.12 into state space representation as

$$\begin{aligned}
\dot{v}_x &= \omega_z v_y - \omega_y v_z + \frac{F_x}{m} \\
\dot{v}_y &= \omega_x v_z - \omega_z v_x + \frac{F_y}{m} \\
\dot{v}_z &= \omega_y v_x - \omega_x v_y + \frac{F_z}{m} \\
\dot{\omega}_x &= [\omega_y \omega_z (I_y - I_z)] \frac{1}{I_x} + \frac{\tau_x}{I_x} \\
\dot{\omega}_y &= [\omega_x \omega_z (I_z - I_x)] \frac{1}{I_y} + \frac{\tau_y}{I_y} \\
\dot{\omega}_z &= [\omega_x \omega_y (I_x) - I_y] \frac{1}{I_z} + \frac{\tau_z}{I_z}.
\end{aligned} \tag{4.13}$$

We almost have the dynamics covered but vectors **F** and $\boldsymbol{\tau}$ consist from a lot of external effects to be taken care of. Firstly let's break down the force vector. To start we assume four types of acting forces

$$\mathbf{F} = F_g \cdot \mathbf{R}^T \cdot \hat{\mathbf{e}}_r - F_T \cdot \hat{\mathbf{e}}_z + \mathbf{F}_w, \tag{4.14}$$

where F_g is scalar value equal to gravitational pull, $\hat{\mathbf{e}}_r$ is unit vector along axis R (ICS frame), F_T stands for total thrust caused by the propellers, $\hat{\mathbf{e}}_z$ for unit vector along axis Z (VCS frame). $\mathbf{F}_w$ represents vector of forces caused by air speed of the aircraft (wind push and air friction). Because all six rotors are mounted horizontally, total thrust is an easy calculation

$$F_T = \sum_{i=1}^{6} T_i, \tag{4.15}$$

where T_i stands for thrust of the corresponding propeller. The friction force is given by

$$\mathbf{F}_w = \begin{bmatrix} F_{wx} \\ F_{wy} \\ F_{wz} \end{bmatrix} = \begin{bmatrix} -\rho \cdot v_{xa}^2 \cdot S_x \cdot C_{Dx} \cdot sign(v_{xa}) \\ -\rho \cdot v_{ya}^2 \cdot S_y \cdot C_{Dy} \cdot sign(v_{ya}) \\ -\rho \cdot v_{za}^2 \cdot S_z \cdot C_{Dz} \cdot sign(v_{za}) \end{bmatrix}, \tag{4.16}$$

where ρ is the density of air, S_i is area of the body the air presses on along the corresponding axis. C_D stands for drag coefficient (unitless) for the corresponding axis. Vector $\mathbf{v}_a$ represents air speed of the aircraft (aircraft's speed relative to the air mass) which is calculated from wind speed and aircraft's ground speed as

$$\mathbf{v}_a = \mathbf{v} - \mathbf{v}_w, \tag{4.17}$$

where $\mathbf{v}_w$ is wind speed relative to the ground. Speed vectors must be referenced in the same coordinate frame. We assume variable air density

$$\rho = \rho_0 \cdot e^{(-10^{-4} \cdot h)}, \tag{4.18}$$

where ρ_0 represents air density at sea level with temperature of 15 $°C$. h stands for height above the sea level, therefore should be defined by sum of state $-r$ and a constant value of ICS origin's height above the sea level. Now we can rewrite first part of 4.13 as

$$\begin{aligned}
\dot{v}_x &= \omega_z v_y - \omega_y v_z - g[s(\theta)] + \frac{F_{wx}}{m} \\
\dot{v}_y &= \omega_x v_z - \omega_z v_x + g[s(\phi)c(\theta)] + \frac{F_{wy}}{m} \\
\dot{v}_z &= \omega_y v_x - \omega_x v_y + g[c(\phi)c(\theta)] - \frac{F_T}{m} + \frac{F_{wz}}{m}.
\end{aligned} \tag{4.19}$$

Total torque applied to the body is composed as

$$\boldsymbol{\tau} = \boldsymbol{\tau}_P - \boldsymbol{\tau}_g + \boldsymbol{\tau}_w, \tag{4.20}$$

where $\boldsymbol{\tau}_P$ stands for thrust caused by the propellers, $\boldsymbol{\tau}_g$ is gyroscopic effect and $\boldsymbol{\tau}_w$ represent torques done by wind. Gyroscopic action can be described as

$$\boldsymbol{\tau}_g = \sum_{i=1}^{6} I_{Pi}(\boldsymbol{\omega} \times \hat{\mathbf{e}}_z) \cdot (-1)^i \cdot \omega_{Pi}, \tag{4.21}$$

where i always represents the corresponding propeller, I_{Pi} is moment of inertia of the rotor and ω_i its rotational speed. Small I_P along with the fact that odd and even rotor groups counter themselves out a lot allow us to fully neglect this effect. Torques caused by the thrust actions of individual propellers are directly based on the hexacopter's architecture

$$\boldsymbol{\tau}_P = \begin{bmatrix} \tau_{Px} \\ \tau_{Py} \\ \tau_{Pz} \end{bmatrix} = \begin{bmatrix} (T_1 + T_2 - T_3 - T_4) \cdot R_y \\ (T_1 + T_2 + T_3 + T_4) \cdot R_x - (T_5 + T_6) \cdot R \\ (-T_1 + T_2 - T_3 + T_4 - T_5 + T_6) \cdot k \end{bmatrix}, \tag{4.22}$$

where T_i is thrust $[N]$ of the corresponding propeller,

$$\begin{aligned}
R_x &= R \cdot \cos(\pi/3) \\
R_y &= R \cdot \sin(\pi/3)
\end{aligned}$$

and R is the distance between a motor's center and the hexacopter's body center (same for each motor). k is constant coefficient representing ratio between torque and thrust. All that being set the second part of 4.13 has now the following form

$$\begin{aligned}
\dot{\omega}_x &= \frac{1}{I_x}\{[\omega_y\omega_z(I_y - I_z)] + (T_1 + T_2 - T_3 - T_4)\cdot R_y + \tau_{wx}\}\\
\dot{\omega}_y &= \frac{1}{I_y}\{[\omega_x\omega_z(I_z - I_x)] + (T_1 + T_2 + T_3 + T_4)\cdot R_x - (T_5 + T_6)\cdot R + \tau_{wy}\}\\
\dot{\omega}_z &= \frac{1}{I_z}\{[\omega_x\omega_y(I_x) - I_y] + (-T_1 + T_2 - T_3 + T_4 - T_5 + T_6)\cdot k + \tau_{wz}\}.
\end{aligned} \tag{4.23}$$

The dynamic model represented by states $\hat{\mathbf{x}} = [v_x \;\; v_y \;\; v_z \;\; \omega_x \;\; \omega_y \;\; \omega_z]^T$ is now completed.

4.3.4 Summary

The general form of our non-linear system can be expressed as

$$\dot{\mathbf{x}}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t)), \tag{4.24}$$

where

$$\mathbf{x} = [\omega_x \quad \omega_y \quad \omega_z \quad v_x \quad v_y \quad v_z \quad \phi \quad \theta \quad \psi \quad p \quad q \quad r]^T$$

$$\mathbf{u} = [T_1 \quad T_2 \quad T_3 \quad T_4 \quad T_5 \quad T_6 \quad g]^T.$$

Concrete form is given by subsystems 4.9, 4.19 and 4.23 as

$$
\begin{aligned}
\dot{\omega}_x &= \frac{1}{I_x}\{[\omega_y\omega_z(I_y - I_z)] + (T_1 + T_2 - T_3 - T_4)\cdot R_y + \tau_{wx}\}\\
\dot{\omega}_y &= \frac{1}{I_y}\{[\omega_x\omega_z(I_z - I_x)] + (T_1 + T_2 + T_3 + T_4)\cdot R_x - (T_5 + T_6)\cdot R + \tau_{wy}\}\\
\dot{\omega}_z &= \frac{1}{I_z}\{[\omega_x\omega_y(I_x) - I_y] + (-T_1 + T_2 - T_3 + T_4 - T_5 + T_6)\cdot k + \tau_{wz}\}\\
\dot{v}_x &= \omega_z v_y - \omega_y v_z - g[s(\theta)] + \frac{F_{wx}}{m}\\
\dot{v}_y &= \omega_x v_z - \omega_z v_x + g[s(\phi)c(\theta)] + \frac{F_{wy}}{m}\\
\dot{v}_z &= \omega_y v_x - \omega_x v_y + g[c(\phi)c(\theta)] - \frac{(T_1 + T_2 + T_3 + T_4 + T_5 + T_6)}{m} + \frac{F_{wz}}{m}\\
\dot{\phi} &= \omega_x + \omega_y[s(\phi)t(\theta)] + \omega_z[c(\phi)t(\theta)]\\
\dot{\theta} &= \omega_y[c(\phi)] + \omega_z[-s(\phi)]\\
\dot{\psi} &= \omega_y[\frac{s(\phi)}{c(\theta)}] + \omega_z[\frac{c(\phi)}{c(\theta)}]\\
\dot{p} &= v_x[c(\theta)c(\psi)] + v_y[s(\phi)s(\theta)c(\psi) - c(\phi)s(\psi)] + v_z[c(\phi)s(\theta)c(\psi) + s(\phi)s(\psi)]\\
\dot{q} &= v_x[c(\theta)s(\psi)] + v_y[s(\phi)s(\theta)s(\psi) + c(\phi)c(\psi)] + v_z[c(\phi)s(\theta)s(\psi) - s(\phi)c(\psi)]\\
\dot{r} &= v_x[-s(\theta)] - v_y[s(\phi)c(\theta)] + v_z[c(\phi)c(\theta)]
\end{aligned}
\tag{4.25}
$$

and friction force is given by equations 4.16 and 4.18.

4.4 Linear model

Now we want to construct a linearized algebraic model because we are going to design linear Kalman filter. Also, we would like to test out the filter in simulation first, and therefore we need a regulator for the system to be able to acquire characteristic behavior signals of the system. The easiest way to get it is to provide the linear model to a function in Matlab and get fully designed state feedback. From now on we will use n for the number of states, m for the number of inputs and s for the number of outputs (not to confuse with position states as p, q, r are usually used here).

4.4.1 Linearization

The issue with our transfer function $\mathbf{f}$ is the nonlinearity which brings problems of solution uniqueness and existence. The trigonometric functions are not related in elementary way and also lead to multiple equilibria present in the system. From now on we will assume small oscillations system to be sure it stays in close neighborhood of chosen equilibrium which is

$$\mathbf{x}_e = \begin{bmatrix}0 & \dots & 0 & p_e & q_e & r_e\end{bmatrix}^T \in \mathbb{R}^{12}$$

where vector $\boldsymbol{\varepsilon}_e = [p_e \ q_e \ r_e]^T$ represents NED coordinates of the area of operation. Inputs

$$\mathbf{u} = \begin{bmatrix}T_1 & T_2 & T_3 & T_4 & T_5 & T_6 & F_g\end{bmatrix}^T,$$

are set to equalize the motors thrusts with gravitational pull while causing no torque to the body.

$$\mathbf{u}_e = \begin{bmatrix}\frac{mgR}{2R+1} & \frac{mgR}{2R+1} & \frac{mgR}{2R+1} & \frac{mgR}{2R+1} & \frac{mg}{2R+1} & \frac{mg}{2R+1} & mg\end{bmatrix}^T,$$

To approximate the small deviation non-linear system in this equilibrium we replace sin and tan functions with theirs arguments and function cos with unity.

$$\begin{aligned}
\dot{\omega}_x &= \frac{1}{I_x}\{[\omega_y\omega_z(I_y - I_z)] + (T_1 + T_2 - T_3 - T_4)\cdot R_y + \tau_{wx}\}\\
\dot{\omega}_y &= \frac{1}{I_y}\{[\omega_x\omega_z(I_z - I_x)] + (T_1 + T_2 + T_3 + T_4)\cdot R_x - (T_5 + T_6)\cdot R + \tau_{wy}\}\\
\dot{\omega}_z &= \frac{1}{I_z}\{[\omega_x\omega_y(I_x) - I_y] + (-T_1 + T_2 - T_3 + T_4 - T_5 + T_6)\cdot k + \tau_{wz}\}\\
\dot{v}_x &= \omega_z v_y - \omega_y v_z - g\theta + \frac{F_{wx}}{m}\\
\dot{v}_y &= \omega_x v_z - \omega_z v_x + g\phi + \frac{F_{wy}}{m}\\
\dot{v}_z &= \omega_y v_x - \omega_x v_y + g - \frac{(T_1 + T_2 + T_3 + T_4 + T_5 + T_6)}{m} + \frac{F_{wz}}{m}\\
\dot{\phi} &= \omega_x + \omega_y\phi\theta + \omega_z\theta\\
\dot{\theta} &= \omega_y - \omega_z\phi\\
\dot{\psi} &= \omega_y\phi + \omega_z\\
\dot{p} &= v_x + v_y[\phi\theta - \psi] + v_z[\theta + \phi\psi]\\
\dot{q} &= v_x\psi + v_y[\phi\theta\psi + 1] + v_z[\theta\psi - \phi]\\
\dot{r} &= -v_x\theta - v_y\phi + v_z
\end{aligned} \tag{4.26}$$

Now we realize proper linearization using partial derivations:

$$\mathbf{A} = \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{x}}\bigg|_{\mathbf{x}=\mathbf{x}_e, \mathbf{u}=\mathbf{u}_e} = \begin{bmatrix} \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{A}_1 \\ \mathbf{E} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{E} & \mathbf{O} & \mathbf{O} \end{bmatrix} \in \mathbb{R}^{n \times n}, \tag{4.27}$$

where $\mathbf{O} \in \mathbb{R}^{3x3}$ represents zero matrix, $\mathbf{E} \in \mathbb{R}^{3x3}$ represents identity matrix and $\mathbf{A}_1$ is

$$\mathbf{A}_1 = \begin{bmatrix} 0 & -g & 0 \\ g & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

Linearized input matrix $\mathbf{B} \in \mathbb{R}^{n \times m}$ is

$$\mathbf{B}_c = \frac{\partial \mathbf{f}(\mathbf{x}, \mathbf{u})}{\partial \mathbf{u}}\bigg|_{\mathbf{x}=\mathbf{x}_e, \mathbf{u}=\mathbf{u}_e} = \begin{bmatrix} \frac{R_y}{I_x} & \frac{R_y}{I_x} & -\frac{R_y}{I_x} & -\frac{R_y}{I_x} & 0 & 0 & 0 \\ \frac{R_x}{T_y} & \frac{R_x}{T_y} & \frac{R_x}{T_y} & \frac{R_x}{T_y} & -\frac{R}{T_y} & -\frac{R}{T_y} & 0 \\ -\frac{k}{I_z} & \frac{k}{I_z} & -\frac{k}{I_z} & \frac{k}{I_z} & -\frac{k}{I_z} & \frac{k}{I_z} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ -\frac{1}{m} & -\frac{1}{m} & -\frac{1}{m} & -\frac{1}{m} & -\frac{1}{m} & -\frac{1}{m} & \frac{1}{m} \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ & & & \vdots & & & \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{4.28}$$

The continuous time state space model is described as

$$\dot{\mathbf{x}}(t) = \mathbf{A} \cdot \mathbf{x}(t) + \mathbf{B} \cdot \mathbf{u}(t).$$

4.4.2 Regulator

For testing purposes, we want to experiment with the filter on virtual simulated data, because we can acquire real values of the states in time. For reasonable courses of the state signals, we need to feed the system with correct input values simulating the action of regulator onboard the aircraft which is trying to compensate unknown external effects onto the system and follow the desired reference. The regulator's development is not within the scope of this thesis, so only a brief description is provided further on.

First thing to do is determining if the system is controllable. We inspect it by constructing the well-known matrix of controllability and checking its rank.

$$\mathtt{C} = \begin{bmatrix} \mathbf{B} & \mathbf{AB} & \dots & \mathbf{A}^{n-1}\mathbf{B} \end{bmatrix} \in \mathbb{R}^{n \times nm}$$

$$rank(\mathtt{C}) = n \quad \rightarrow \quad system\ fully\ controllable$$

We have great advantage of knowing all states at any time therefore the matrix C representing mapping of state vector onto output vector is arbitrary. As stated in 4.2.3 we are only able to control 4 states to arbitrary courses, the others are derived. We are not interested in forcing the velocities and roll/pitch angles to follow some reference. We only want to tell the aircraft which position to reach and what heading to maintain which also satisfies the number of directly controllable states. Let us then choose output matrix

$$\mathbf{C} = \begin{bmatrix} 0 & \dots & 0 & 0 & 1 & 0 & 0 \\ 0 & \dots & 0 & 0 & 0 & 1 & 0 \\ 0 & \dots & 0 & 0 & 0 & 0 & 1 \\ 0 & \dots & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \in \mathbb{R}^{n \times s} \tag{4.29}$$

for output definition

$$\mathbf{y}(t) = \mathbf{C} \cdot \mathbf{x}(t).$$

We need to design state feedback with integrated output feedback to be able to control states of the system because we are not able to provide reasonable reference signals for motors which represent direct input to the system. Therefore we follow the scheme in figure 4.4.

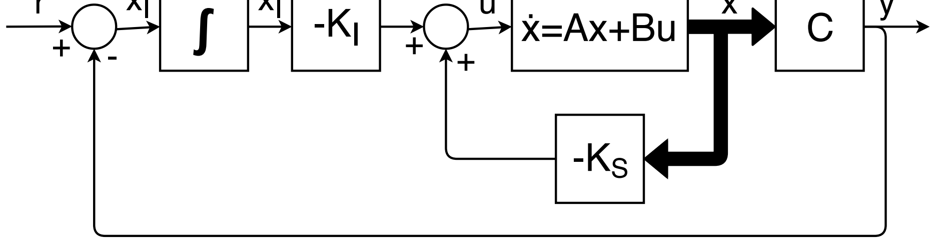


Figure 4.4: System with state feedback regulator

The whole system can be now described by the following equations

$$\begin{bmatrix}\dot{\mathbf{x}}\\ \dot{\mathbf{x}}_I\end{bmatrix} = \begin{bmatrix}\mathbf{A}-\mathbf{B}\mathbf{K}_F & -\mathbf{B}\mathbf{K}_I\\ -\mathbf{C} & \mathbf{O}\end{bmatrix}\cdot\begin{bmatrix}\mathbf{x}\\ \mathbf{x}_I\end{bmatrix} + \begin{bmatrix}\mathbf{B}\\ \mathbf{O}\end{bmatrix}\cdot\mathbf{u} + \begin{bmatrix}\mathbf{O}\\ \mathbf{E}\end{bmatrix}\cdot\mathbf{r}$$

$$\mathbf{y} = \begin{bmatrix}\mathbf{C} & \mathbf{O}\end{bmatrix}\cdot\begin{bmatrix}\mathbf{x}\\ \mathbf{x}_I\end{bmatrix}, \tag{4.30}$$

where all new vectors and matrices have dimensions to fit the previously stated ones. There are four new states in the system therefore 16 in total. We choose 16 positions for the poles of the system. Knowing that we cannot place more than $rank(B)$ poles in the same place, we experimentally picked poles such that the system follows the reference correctly but with non-extreme propellers actions which would not be reachable with the real aircraft. We cannot insert saturation of input u into the system because the regulator would not handle it, therefore the only option how to not overload the propellers is to pick the poles carefully. Finally chosen poles positions are

$$\mathbf{p} = [-1\ -1\ -1\ -1.5\ -1.5\ -1.5\ -2\ -2\ -2\ -2.5\ -2.5\ -2.5\ -3\ -3\ -3]^T$$

Simple call of Matlab's function "$place(\mathbf{A}_f, \mathbf{B}_f, \mathbf{p})$" with matrices

$$\mathbf{A}_f = \begin{bmatrix}\mathbf{A} & \mathbf{O}\\ -\mathbf{C} & \mathbf{O}\end{bmatrix} \in \mathbb{R}^{(n+s)\times(n+s)}; \quad \mathbf{B}_f = \begin{bmatrix}\mathbf{B}\\ \mathbf{O}\end{bmatrix} \in \mathbb{R}^{n\times m}$$

will give use matrix $\mathbf{K}_f$ from which we acquire our feedback matrices

$$\mathbf{K}_f = \begin{bmatrix}\mathbf{K}_s & \mathbf{K}_I\end{bmatrix}.$$

We could implement this model into Simulink by using the overall matrices $\mathbf{A}_f$, $\mathbf{B}_f$, $\mathbf{K}_f$, but that would result in unstructured state vector including the states we are not interested in (the ones used for control). Also this approach is more illustrative and gives an insight into the system and the regulator separately. We have sent reference signals to the input of the system to see its behavior, the state response is illustrated in figure 4.5, motors actions can be seen in figure 4.6.

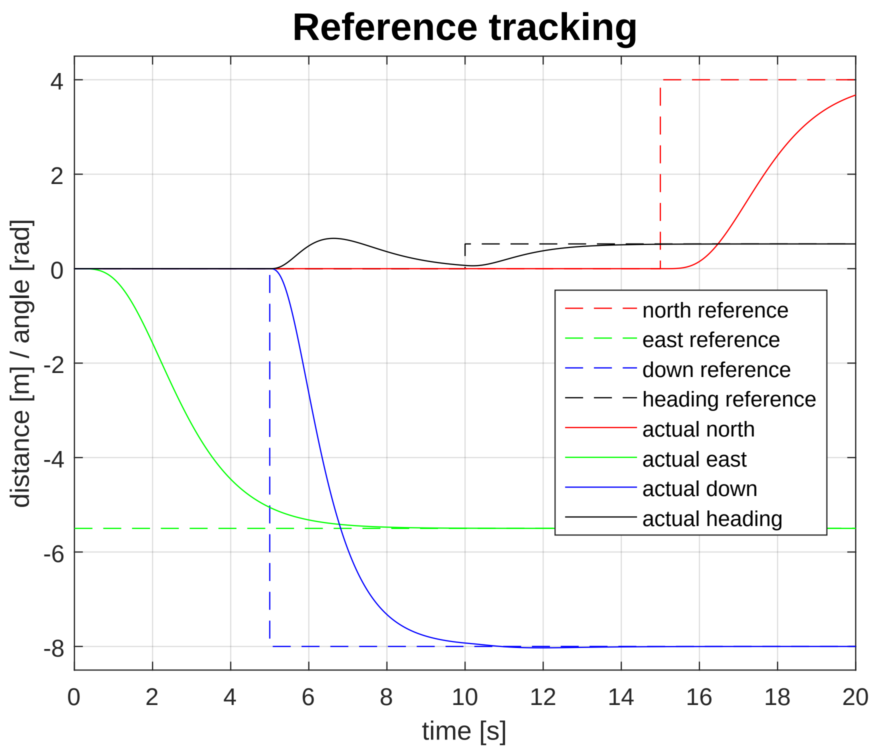


Figure 4.5: Regulated system reference tracking

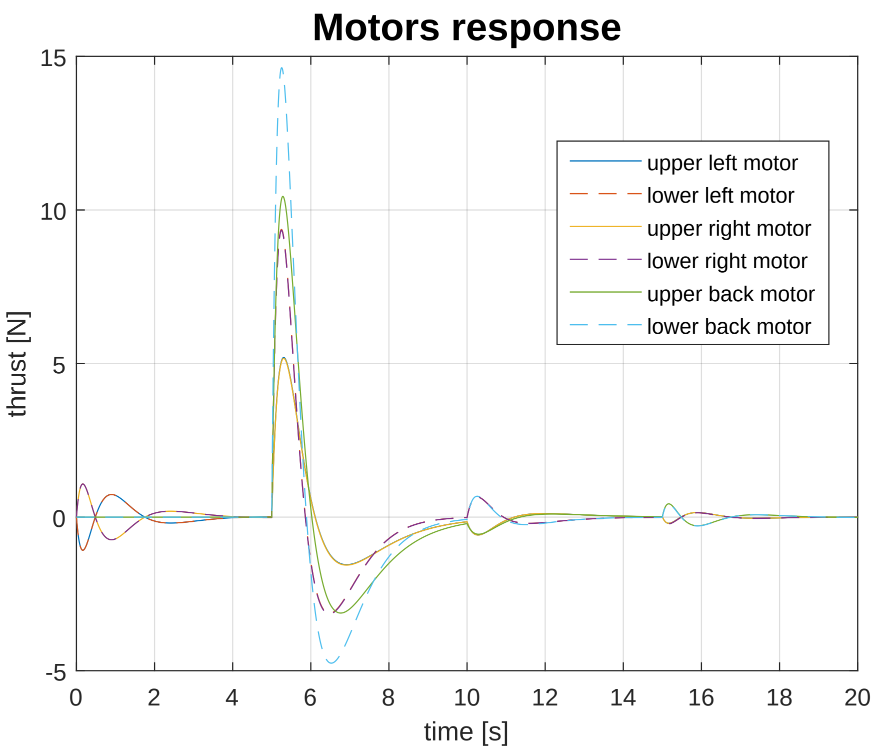


Figure 4.6: Regulated system motors responses

Chapter 5

Kalman filtering

In this chapter, we introduce the methods used for development of specific Kalman filter fitted onto the chosen aircraft. Firstly we introduce basic statistics knowledge needed for understanding the filter; then we describe the basic algorithm along with improvements fitted to our application. In the second section, we describe the construction of a virtual system for simulation of the real one. At last, the filtering algorithms are tuned using knowledge about the physical system and measurements acquired during experimental flights.

5.1 Theoretical introduction

In this section, we will briefly introduce basic statistical terms and relations needed for understanding the Kalman filtering features. More detailed description of the filter's algorithm implementation will follow.

5.1.1 Univariate Gaussian

We desire a unimodal, continuous representation of probabilities which model the real world and is easy to define and calculate. Gaussian probability distribution (GPD) $N(\mu, \sigma^2)$ has these specifications. For one random variable, there are only two parameter values which fully defines the function. Mean (μ), which represents the average value of given dataset, and variance (σ^2), which defines the spread of values of the dataset from the mean value. A more practical term is "standard deviation" (σ) values which can be derived as a square root of the variance value. We can use the 68-95-99,7 rule to understand this parameter better. If we take particular set (large enough) of values of random variable with GPD and acquire its mean, the σ values is derived such that 68 % of dataset values lies within interval $\mu \pm \sigma$, 95 % lies within $\mu \pm 2\sigma$ and $99,7$ % lies within $\mu \pm 3\sigma$. These facts were covered only to ensure a clear understanding of the following text, other properties of the distribution will not be covered as they are subject to an extensive study of the field of statistics itself.[Jr.18b]

The Kalman filter supposes all random events in the system to be subjects of GPD. With its usage, we can acquire computationally optimal mathematical algorithms to implement. However, we must keep in mind that real-world events do not comply with this distribution. Sometimes a random variable can be well approximated by GPD but cannot reach negative values for example (or have limits/saturation on both sides), this fact truncates the "tails" of the distribution and slightly changes some of its properties. Also, in the real world, some variable may generate more values $\mu+$ close to the mean and lower number of values $\mu-$ which are farther, this forces the GPD graph representation to skew to one side of the mean. There are more such phenomena which we are not able to examine precisely and therefore use the GPD approximation.

5.1.2 Multivariate Gaussian

The previous two paragraphs described GPD for a single random variable. While assuming physical quantities/events of the real world, many of them are somehow correlated and dependent on each other. These relations are described by correlated variance, also called covariance values. Multivariate GPD $N(\mu, \sigma^2)$ uses, along vector of values μ, a covariance matrix σ^2. It includes variance values for corresponding random variables on its main diagonal and covariance values between two random variables in the row and the column corresponding to the two variables. Both combinations of the row/column indexes have the same value. Therefore the covariance matrix is symmetrical along the main diagonal. For more detailed description, readers are referred to [Jr.18b] or any statistics book.

5.1.3 Kalman filter algorithm

As we mentioned earlier, the Kalman filter is a state estimator. Its main advantage compared to raw measurements only is the knowledge (estimate) of the physical system's mathematical model derived in the previous chapter. We will introduce the basic version of the filter which assumes knowledge of the inputs to the system (actions of the system's actuators).

Basically the filtering process is composed of two differentiable steps, prediction and update. Let's assume we will work with prediction/update frequency of f_s, therefore timestep is $T_s = 1/f_s$. Because the filter is digital (discrete-time) we need to provide it with discrete version of the system's matrices. These are acquired by standard discretization of continuous state space representation using the chosen step time. All sensory data are sampled with the same period, T_s for this example. Let's call the discretized matrix A a transition matrix F and new output matrix representing measured states as H. The other system's discretized matrices are denoted as B_d, D_d. To further distinguish new output from the one we used for the regulator earlier, we denote the measured output $\mathbf{z}$. A general discrete-time state space system with external effects can be described as:

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{F}\cdot\mathbf{x}_k + \mathbf{B}_d\cdot\mathbf{u}_k + \mathbf{M}\cdot\mathbf{d}_k + \mathbf{w}_k\\ \mathbf{z}_k &= \mathbf{H}\cdot\mathbf{x}_k + \mathbf{D}_d\cdot\mathbf{u}_k + \mathbf{N}\cdot\mathbf{d}_k + \mathbf{v}_k,\end{aligned} \tag{5.1}$$

where $k \in \mathbb{Z}_{0+}$ represents time step number, $\mathbf{d}_k$ stands for unknown input vector and matrices $\mathbf{M}$, $\mathbf{N}$ states influence of it on future state and current output vector. Process noise $\mathbf{w}_k$ and measurement noise $\mathbf{v}_k$ are mutually uncorrelated, unbiased (with zero mean value), white noise (Gaussian) signals with covariance matrices $\mathbf{Q}$ and $\mathbf{R}$ respectively.

As we only want to design simple filter, we do not want to have known and unknown input signals in the system at the same time. Solution for that type of problem can be found in [YZF13]. We assume only $\mathbf{u}_k$ as the system's input and use it either as known or unknown depending on particular situation. We also neglect direct relation between input and output, therefore setting $\mathbf{D}_d = \mathbf{O}$. Further on, we will work with a system

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{F}\cdot\mathbf{x}_k + \mathbf{B}_d\cdot\mathbf{u}_k + \mathbf{w}_k\\ \mathbf{z}_k &= \mathbf{H}\cdot\mathbf{x}_k + \mathbf{v}_k.\end{aligned} \tag{5.2}$$

Before we can run the iterative estimation algorithm of the filter, we need to declare some necessary constants and initial values of used variables. The constants which needs to be set are matrices $\mathbf{Q}$, $\mathbf{R}$, $\mathbf{H}$. In the calculation there will be matrix P, also denoted as estimate covariance, which will be transferred over timesteps and needs to be initialized for the first iteration. The same manner goes with state vector $\mathbf{x}$. For simplicity, let's assume the starting step $k = 0$. After the initialization phase we consider the constants and $\mathbf{x}_{k=0}$, $\mathbf{P}_{k=0}$ to be set.

The first step of estimation is simple prediction of the system's evolution in time calculated from estimated previous state (initial guess) and current input according to the state space model:

$$\begin{aligned}\mathbf{x}_{k|k-1} &= \mathbf{F}\cdot\mathbf{x}_{k-1} + \mathbf{B}_d\cdot\mathbf{u}_k && \textit{Predicted state estimate}\\ \mathbf{P}_{k|k-1} &= \mathbf{F}\cdot\mathbf{P}_{k-1}\cdot\mathbf{F}^T + \mathbf{Q} && \textit{Predicted estimate covariance.}\end{aligned} \tag{5.3}$$

The second step is update of the prediction with usage of measured data at current time step.

$$\begin{aligned}
\mathbf{y}_k &= \mathbf{z}_k - \mathbf{H}\cdot\mathbf{x}_{k|k-1} && \textit{Innovation pre-fit residual}\\
\mathbf{S}_k &= \mathbf{R} + \mathbf{H}\cdot\mathbf{P}_{k|k-1}\cdot\mathbf{H}^T && \textit{Innovation covariance}\\
\mathbf{K}_k &= \mathbf{P}_{k|k-1}\cdot\mathbf{H}^T\cdot\mathbf{S}_k^{-1} && \textit{Optimal Kalman gain}\\
\mathbf{x}_k &= \mathbf{x}_{k|k-1} + \mathbf{K}_k\cdot\mathbf{y}_k && \textit{Updated state estimate}\\
\mathbf{P}_k &= \mathbf{P}_{k|k-1} - \mathbf{K}\cdot\mathbf{H}\cdot\mathbf{P}_{k|k-1} && \textit{Updated estimate covariance}
\end{aligned} \tag{5.4}$$

When provided with a correct mathematical model which behavior is similar to the real system, the filter is always able to converge to the correct estimate (close to the real state). This can be used as an advantage during initiation of the variables $\mathbf{x}$, $\mathbf{P}$ which can be set to almost any value/s. It is most convenient to provide the best initial values estimate for quicker convergence. The filter is tuned by changing the matrices $\mathbf{Q}$ and $\mathbf{R}$ for best approximation of the real noises present in the system.[Kal60] We assume the matrices to be constant over time.

5.1.4 Filling measurements gaps

The previously stated example counted on measurements from all sensors being available at each time step of the estimation. However, to fully utilize all advantages of Kalman filtering, we want the filter to provide estimates with higher frequency (for example ten times faster than the sampling one of the sensory data). That is achieved by running the first prediction step with higher frequency so it can execute multiple times before a measurement is available to make an update step. The following pseudo-code illustrates the implementation:

```
measurements=load_array_of_measurements(source);
set_initial_values(t,x,P);
mIdx=search(measurements(mIdx).time>t);

while(t<=endTime){
   do_prediction_step();

   if(t>=measurements(mIdx).time){
      do_update_step();
      mIdx=mIdx+1;
   }
   t=t+Ts
}
```

5.1.5 Measurements of various frequency

Moreover, we have got two sensors which each provide the measurements with different sampling frequency. GNSS sensor samples much slower (20 Hz with RTK after post-processing) than the IMU (250 Hz). We want the prediction to be corrected by all IMU measurements available, therefore develop a way to use only part of the sensory data at a time. It is important to keep track of the form of matrices **R** and **H** because they are dependent on the relation between particular measurement and the states (what states are included in the measurement). Concrete forms of the matrices can be found in attached materials (kalman_filter.m).

```
while(t<endTime){
  do_prediction_step();

  if(imuMeasAvilable or rtkMeasAvailable) {
    if(imuMeasAvilable and rtkMeasAvailable) {
      set_zHR_for_both_sensors();
    } else if(imuMeasAvilable) {
      set_zHR_for_imu_sensor();
    } else if(rtkMeasAvailable) {
      set_zHR_for_rtk_sensor();
    }
    do_update_step();
  }
  t=t+Ts;
}
```

5.2 Simulation

As we discussed earlier, we want to establish working simulation of the previously described model to be able to design the desired filter. We want the simulated environment to be as much similar as the real one. The state space model was derived as linear only to allow the simple design of the regulator, for all other simulation parts we can introduce nonlinearities into the model. Therefore the external effects will be implemented as non-linear relations of the states. However, we still assume the vehicle's states to stay near the working point to satisfy the linear approximation of the model. Based on this assumption we can put axes $P \equiv X,\ Q \equiv Y,\ R \equiv Z$.

5.2.1 Gravitational pull

To start with the simplest external effect, we introduce the gravitational pull of Earth represented by the gravitational acceleration in the direction of axis R. The acceleration is assumed constant with value of $9,81\ ms^{-2}$. According to 4.26, we use the constant only as a coefficient for states relations in equations for $\dot{v}_x$ and $\dot{v}_y$. As an input it is used for $\dot{v}_z$ calculation by simple addition. Simulink implementation can be seen in 5.1. System's reaction to the presence of the pull is included in figures 5.3 and 5.4.

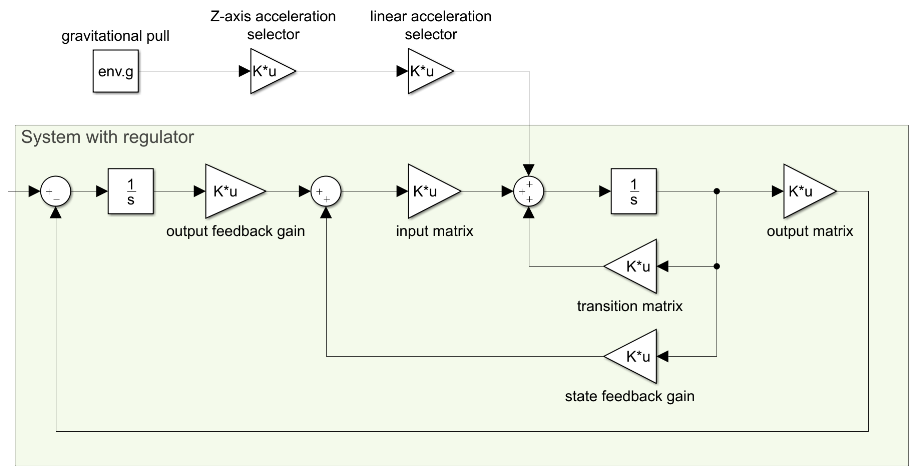


Figure 5.1: Simulink implementation of gravitational external force

5.2.2 Air friction

Because of the fairly big surface of the aircraft and significant density of air along with wind speed and the aircraft's speed being nonzero, we want to utilize the effect of air friction into the model. The force relations are implemented using equations 4.16, 4.17 and 4.18. For the sake of simpler implementation and easier diagnosis, we assume the height value affecting the air's density to be constant (concretely 150 meters above the sea level [ASL]). We can do that because of minimal changes in density proportionately to the height ASL.

The wind torque actions cannot be simply described by mathematical relations based on wind speed only. Because of the symmetrical surface of the aircraft, simple air mass movement would not cause any torque to the body. It can be caused by wind flow around the body parts and rotating propellers. The phenomenon of buoyancy is mainly involved. It can be estimated by methods involving modeling aerodynamics of the aircraft. That, however, is not within the scope of this work. Therefore we will force the final torque quantity into the model directly (without any relation to observable physical quantity), trying to cover all extremes which may happen in a real environment. The Simulink diagram is shown in figure 5.2. It takes the linear velocity of the aircraft in ICS as an input and outputs addition to system's states before the integration the same way as the gravitational pull in 5.1. Airspeed represents the relative speed of the aircraft to the air mass surrounding it.

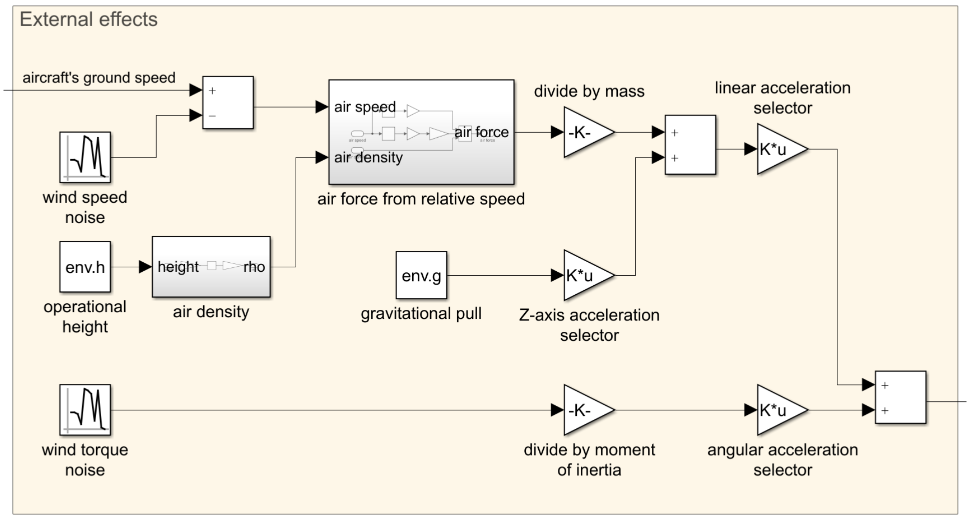


Figure 5.2: Simulink implementation of wind effects

The noise blocks generate white noise signals. Each of the six signals (vector of three for each block) is generated using different seed value to ensure not correlated noises in the system. Kalman filter approach expects the noises in the system to be unbiased meaning the white noises should have the mean value of 0. However, we want to simulate real usage outdoors where the wind might blow the same direction with the same speed for several minutes (and similarly for hours). Therefore we set the wind speed noise to have mean values $[-5;\ 3;\ 0]$ for axes X, Y, Z respectively. Variances are set to $[1;\ 1;\ 1/16]$ assuming that the wind does not blow as much from the top and the bottom. The sample time is set to $0,1\ s$. The wind torque noise has all mean values zero, variances 10^{-4}. System's responses after addition of these effects can be seen in figures 5.3 and 5.4.

5.2.3 Diagnostics

The whole state vector $\mathbf{x}$ can be observed in the simulation. Moreover, we can acquire also vector $\dot{\mathbf{x}}$ just before the integration process. Thanks to this we can acquire time courses of the following vectors:

1. Vehicle's coordinate system
   a. Angular acceleration
   b. Linear acceleration
   c. Angular speed
   d. Linear speed

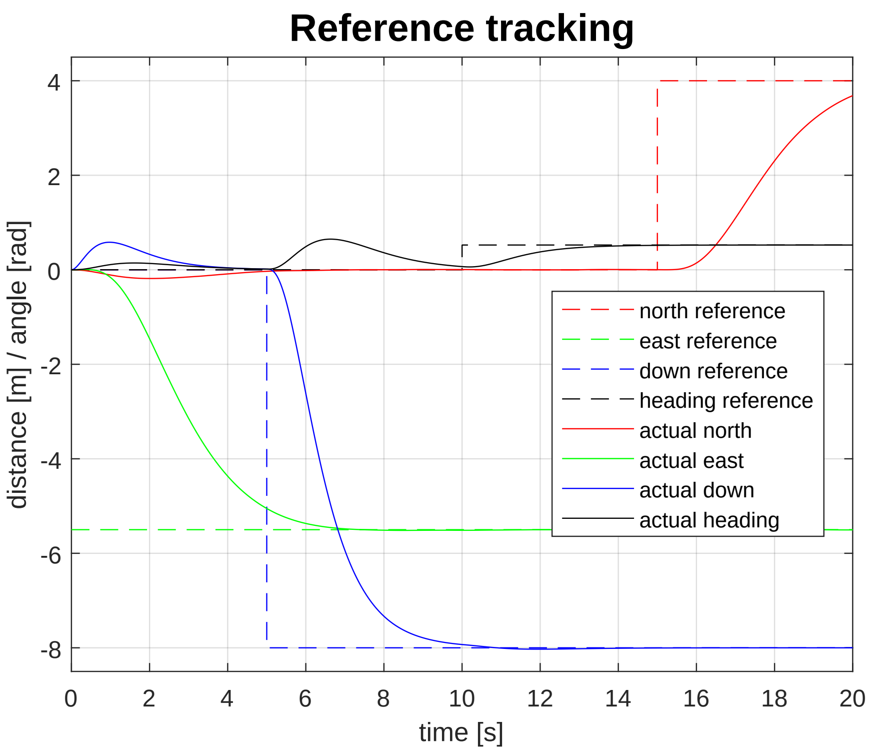


Figure 5.3: Regulated system reference tracking with ext. influence

2. Inertial coordinate system
    a. Angular speed
    b. Linear speed
    c. Attitude (Euler angles)
    d. Position (North, East, Down)

These can be effectively used for evaluation of the final filter during simulation because we will try to get as close as possible to these states.

5.2.4 Virtual sensors

In real mapping process, we are unable to reach the previously described states of the system directly. Sensors must be used to acquire the information. Those bring one more variance to the signals. Each sensor adds up measurement noise to quantities which it measures. We simulated this intervention more accurately than the system's noises because we were able to derive estimates of variance values with the help of technical documentation of the sensors. You can see the implementation in figure 5.5.

Again, the noise generators are all set to different seeds (even from the system's noise signals) to satisfy the condition of not correlated noise signals in the system. All of them have their value set to 0 and variances of 1/22500

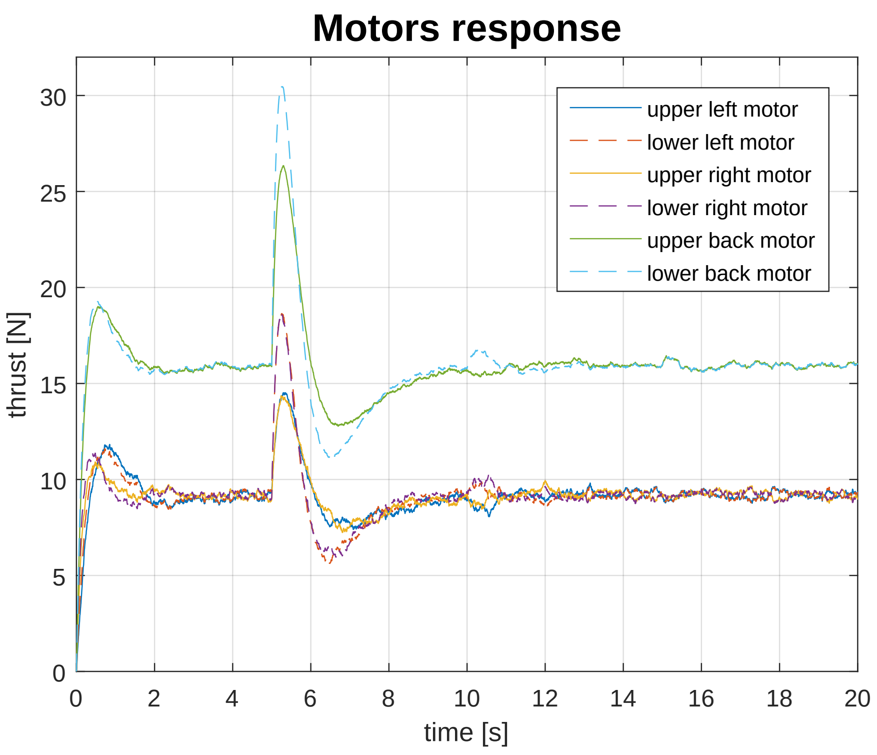


Figure 5.4: Regulated system motors responses with ext. influence

for each acceleration measurement (angular and linear) and 1/14400 for Euler angles measurements. For position values, they were all set to 1/9. Sample times were also set accordingly to our usage of the particular sensors: $0,004\ s$ for IMU and $0,05\ s$ for GNSS sensor. The difference between the actual states and measured values is illustrated on position measurement in 5.6.

5.2.5 Improved mathematical model

Because of the form of the mathematical model derived earlier, we were only able to incorporate position and orientation measurements into the filter. That had lowered the amount of information inserted into the estimator and discarded the information about accelerations. Therefore, we wanted to extend the model such that all available measurements could be used. To do this, we introduce a new set of states which describe angular ($\boldsymbol{\alpha}$) and linear ($\mathbf{a}$) accelerations along all three axes in the VCS. Derivation of acceleration is called jerk and is the third derivative of a position/orientation. Because we assume the aircraft to be a rigid body, there are no external influences which can alter jerk values in time.

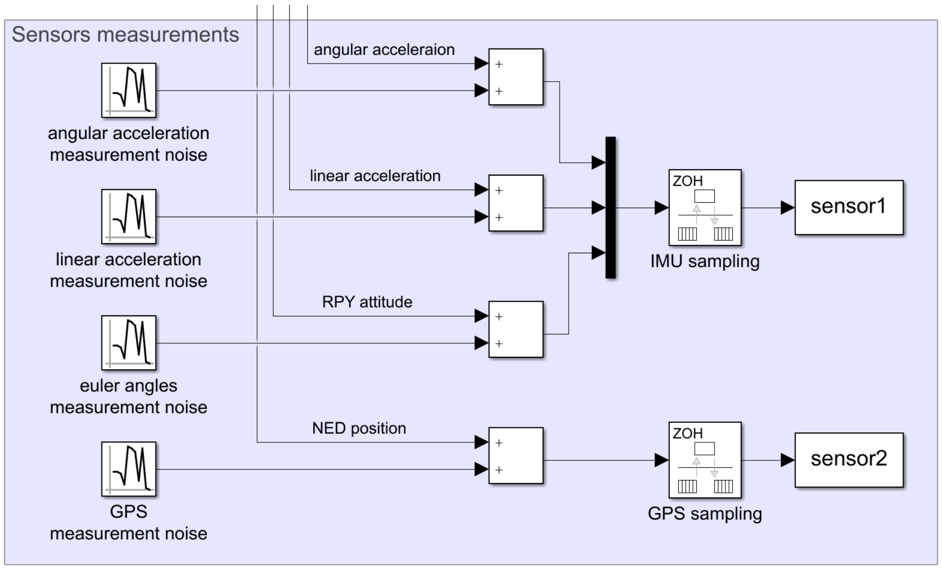


Figure 5.5: Sensors noise simulation

$$\boldsymbol{\alpha} = \begin{bmatrix} \alpha_x \\ \alpha_y \\ \alpha_z \end{bmatrix}, \quad \mathbf{a} = \begin{bmatrix} a_x \\ a_y \\ a_z \end{bmatrix}$$

Current discrete model in use is described by the following transition matrix:

$$\mathbf{x} = \begin{bmatrix} \boldsymbol{\alpha} \\ \mathbf{a} \\ \boldsymbol{\omega} \\ \mathbf{v} \\ \boldsymbol{\varphi} \\ \boldsymbol{\varepsilon} \end{bmatrix}, \quad \mathbf{F} = \begin{bmatrix} \mathbf{E} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{E} & \mathbf{O} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{T}_s & \mathbf{O} & \mathbf{E} & \mathbf{O} & \mathbf{O} & \mathbf{O} \\ \mathbf{O} & \mathbf{T}_s & \mathbf{O} & \mathbf{E} & \mathbf{O} & \mathbf{O} \\ \mathbf{T}_s^2/2 & \mathbf{O} & \mathbf{T}_s & \mathbf{O} & \mathbf{E} & \mathbf{O} \\ \mathbf{O} & \mathbf{T}_s^2/2 & \mathbf{O} & \mathbf{T}_s & \mathbf{O} & \mathbf{E} \end{bmatrix},$$

where each stated element of matrix F is a matrix of dimensions 3×3. **E** stands for unity matrix and other stated values represent unity matrix multiplied by that constant. Matrices **H** corresponding to various measurements are defined such that they satisfy the correct mapping of states onto the measurements (see attached script). We excluded the elements related to gravitational pull, because we expect the motors regulators to compensate its effect during a flight. We do not need to keep track of matrix **B** form, because in our case all inputs are unobservable, therefore the prediction simplifies to

$$\mathbf{x}_{k+1} = \mathbf{F} \cdot \mathbf{x}_k + \mathbf{w}_k. \tag{5.5}$$

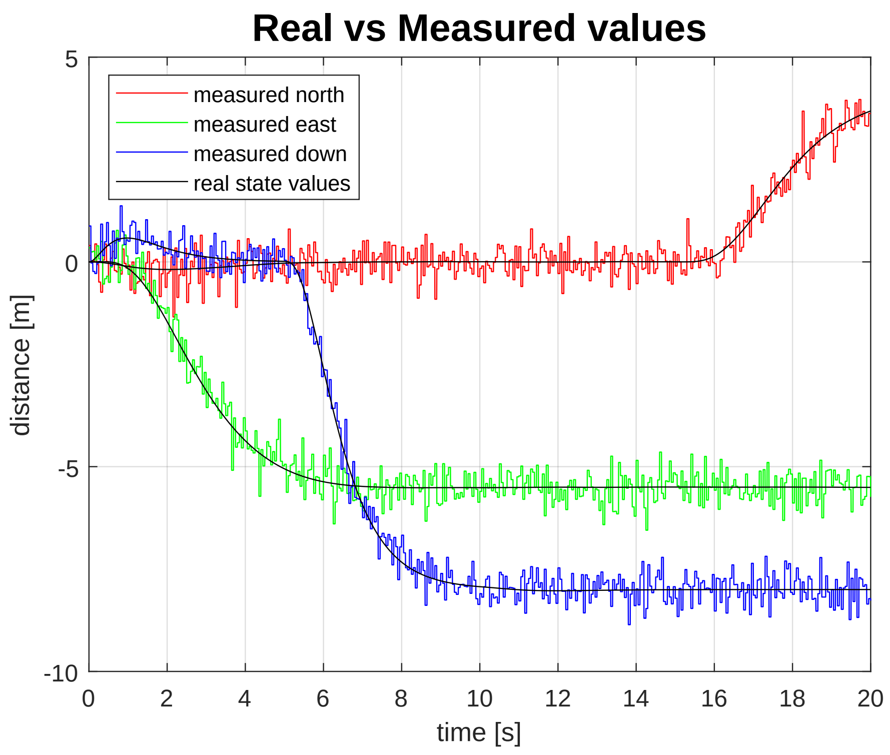


Figure 5.6: Simulated measurement

5.3 Real system application

To fit the filter to our real application with physical UAV, we conducted several test flights with the two sensors onboard the UAV to collect measurements data. The flights were done above wide airfield grass area. From all flights, we have chosen the one which characteristics were the closest to those in future planned mapping missions. You can see the trajectory pattern flown in figure 5.7. The heading was tried to be preserved the same all the time because we would like the lidar to maintain a similar orientation relative to the ground during whole mapping mission in the future. From now on, all stated data is from the chosen dataset and time axis covers the same time interval in each graph.

5.3.1 Differences from simulation

After processing the in-field data, we discovered huge differences between the real and the simulated measurements conducted earlier. This fact had several reasons to happen and was expected. Comparison of simulated and real measurements can be seen in figures 6.1 through 6.4 in chapter 6. The mathematical model serves only as an approximation, it was linearized around a working point and derived using only basic physical laws while neglecting a

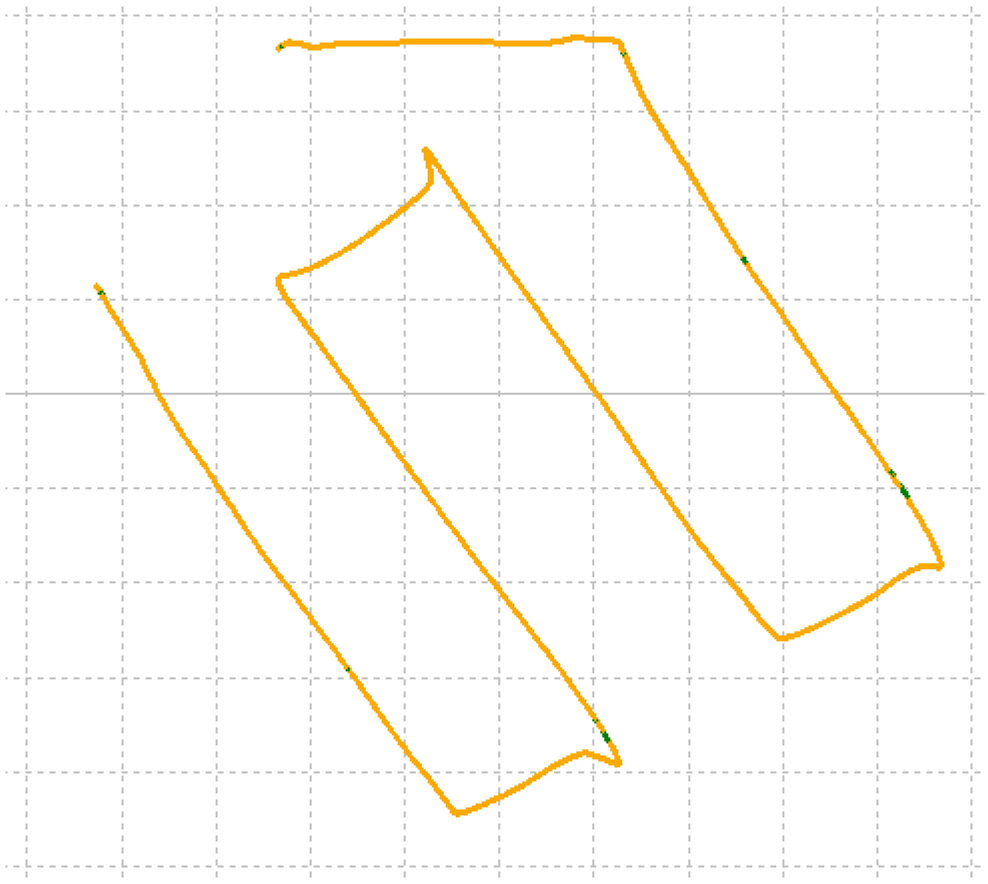


Figure 5.7: Test flight trajectory pattern

lot of others. From the measured vibrations, it is obvious that the assumption of a rigid body which is not subject to any deformation degrades the model's reflection of the real aircraft where the arms bend because the motors cause thrust at the arms ends. Moreover, incorrect process and measurement noises might have been used previously. Also, the noises do not have to satisfy the assumption of a Gaussian probability distribution. The wind actions could be different too. These were unobservable during the flight.

5.3.2 Measurements variance identification

There is a more accurate method to acquire matrix $\mathbf{R}$ for a sensor than by derivation from the sensor's specification. Because such matrix describes the variances of individual measurements (assuming their noise is Gaussian) it can be calculated from a static measurement. If we keep the sensor's state which is being measured the same during the whole measurement, all that is recorded is noise (biased). It is then easy to compute the mean value for each measured quantity, subtract it from the values and identify the variance of them. We have done 15 minutes static measurements for both sensors using configurations intended for usage in the final mapping application. Then used Matlab software to calculate covariance matrix for each sensor.

Because of each sensor measures more physical quantities, we acquired not only variance value for each quantity but also cross-relations between the quantities represented by covariance values. Because the measurements haven't got infinitely long duration and some quantities such as Euler angles and corresponding angular accelerations DO correlate with each other, the calculated covariance values were non-zero. However, even if they are dependent, the device has separate sensors which measure the quantities separately. Therefore, we assume the covariances to be zero. The sensors noise covariance matrix $\mathbf{R}$ is then constructed by placing variances of measured quantities onto its main diagonal while leaving all other elements zero.

5.3.3 Filter tuning

Now that we have fixed the matrices R, we have to determine the matrix Q. It can be derived using the following formula:

$$\mathbf{Q} = \int_0^{\Delta t} \mathbf{F} \cdot \mathbf{Q}_C \cdot \mathbf{F}^T \; dTs, \tag{5.6}$$

where $\mathbf{Q}_C$ is a matrix defining variances of particular states noises. It has non-zero values only on its main diagonal. Because we assume the only process noise to be caused by forces and torques, it affects only states $\boldsymbol{\alpha}$, $\mathbf{a}$ representing the accelerations of the aircraft. There is no other physical quantity present to affect other states directly. We implemented automated algorithm which is able to calculate matrix $\mathbf{Q}$ from six provided variance values (three for each vector of acceleration). The general form of the matrix $\mathbf{Q}$ is

$$\mathbf{Q} = \begin{bmatrix} \mathbf{S}\cdot T_s & \mathbf{S}\cdot T_s^2/2 & \mathbf{S}\cdot T_s^3/6 \\ \mathbf{S}\cdot T_s^2/2 & \mathbf{S}\cdot T_s^3/3 & \mathbf{S}\cdot T_s^4/8 \\ \mathbf{S}\cdot T_s^3/6 & \mathbf{S}\cdot T_s^4/8 & \mathbf{S}\cdot T_s^5/20 \end{bmatrix}, \quad \mathbf{S} = \mathbf{E}\cdot \begin{bmatrix} s_1 \\ \vdots \\ s_6 \end{bmatrix} \in \mathbb{R}^{6\times 6}. \tag{5.7}$$

We iteratively changed the input s_i values, firstly each value in a trinity (s_1, s_2, s_3 and s_4, s_5, s_6) having the same value, then fine tuned some of them. The goal was to acquire as much similar output from the virtual sensors assigned to the simulation as the real data measured in-field. This approach ensured the best approximation which could be acquired with all the knowledge we had. As said before, real process and measurement noises are for sure not exactly Gaussian, therefore we can never get the simulated datasets the same as the measured ones. As an outcome of this fact, the filter using matrices $\mathbf{Q}, \mathbf{R}$ for estimation with Gaussian noises involved will always provide only an **approximation** of the model and **estimation** of the model's states.

Part III

Evaluation

Chapter 6
Experiments

6.1 Trajectory filtering

Several experiments were done to derive the filter's parameters. Some of them were already covered in the previous chapter. To compare the two implementations with 12-state and 18-state mathematical models, we used the same dataset of measured values from the testing flight as an input to each filter. A filter was always initialized with the first available measurement. Therefore the initial state estimate was fairly accurate. The parameters needed for the Kalman filter were tuned for the 18-state filter and applied to both (considering truncated matrices, etc.).

We acquired filtered values from a particular filter and calculated differences between them and the values measured by the sensors. Absolute values of those differences are illustrated in figures 6.5 through 6.8. These values cannot be used directly for a filter's evaluation. For that, we would have to compare the filtered values against real values which we do not know. However, it is obvious that very low values indicate over-fitted tracking of the measured values by the filtered values (which we do not want). Moreover, these diagrams can be used to detect the most critical time intervals, where attention should be focused on.

Figures 6.9 through 6.12 shows two different quantities in two different times filtered by both filters. As we can see in the first example, not only that the 12-state filter did not filter out the noise, it also added up more of it. The filter was also tuned individually (to acquire parameters optimal for this filter) to provide a more convenient output, but it still had worse performance than the 18-state filter, because it could not predict the future orientation well without knowing the angular acceleration (it often overshot). In this configuration, it practically just implements dead reckoning approach as it predicts the velocities from past measurements and once a new measurement is available, it corrects itself. No redundant information is introduced into the system.

On the other side, the 18-state filter can perform better position and attitude prediction as it acquires information about their second derivations (angular and linear acceleration). The filtered curve is smoother than the original one and does not incorporate the system's noise as much as the 12-state filter. The filter seems to overshoot sometimes, but this cannot be proofed without knowledge of the real values (also sensor can measure incorrectly). Another advantage of the 18-state filter is that it can partially eliminate outlying values which were recorded incorrectly and did not reflect the real state. The second example illustrates this. The filter does not let the values to diverge as much when incorrect measurements arrive. However, we can see that because there were five outlying records, the filter was not able to eliminate their negative effect completely. This situation might be a subject for other filtering approach taking care of outlying values.

6.2 Point cloud improvement

In the goal description, we stated that we would use one more method for evaluation of the accuracy of filtered values, comparison of point clouds constructed using the original data from sensors and the filtered data. However, we experienced major dysfunction of the used lidar. Luckily we were able to record some data after successful development of all the algorithms related to the point cloud construction (data acquisition) process, and therefore we can propose at least sample of a model created with our mapping system with the hardware mounted on an experimental ground vehicle, see 6.13. Unfortunately, the lidar malfunctioned before we could perform a flight with the mapping rig mounted to UAV. Therefore we are not able to provide a comparison using the second proposed evaluation method. No spare identical lidar was available in our institution.

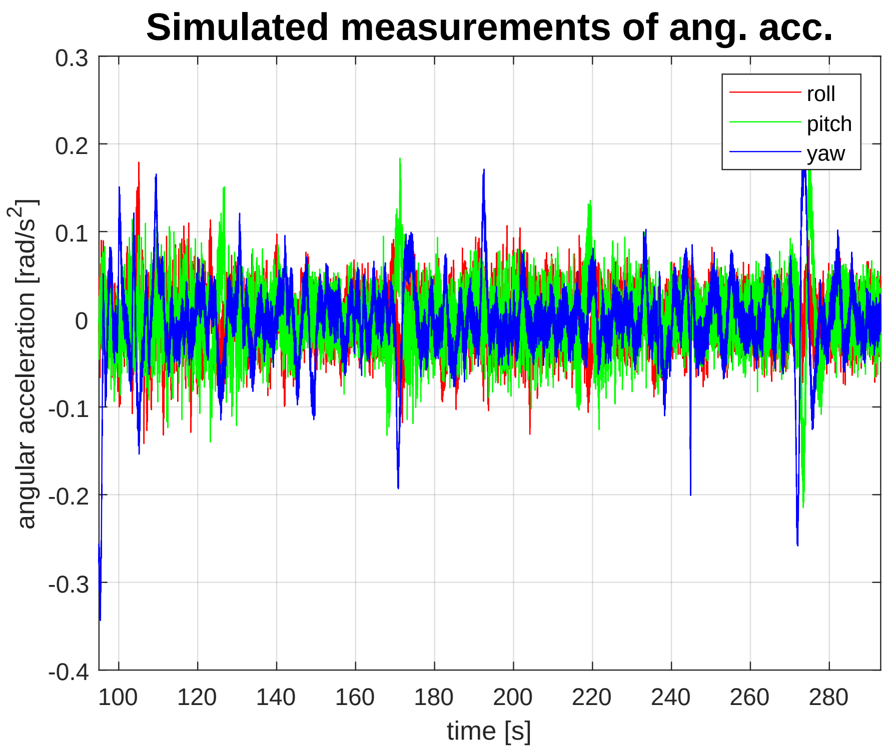


Figure 6.1: Simulated measurement of ang. acc.

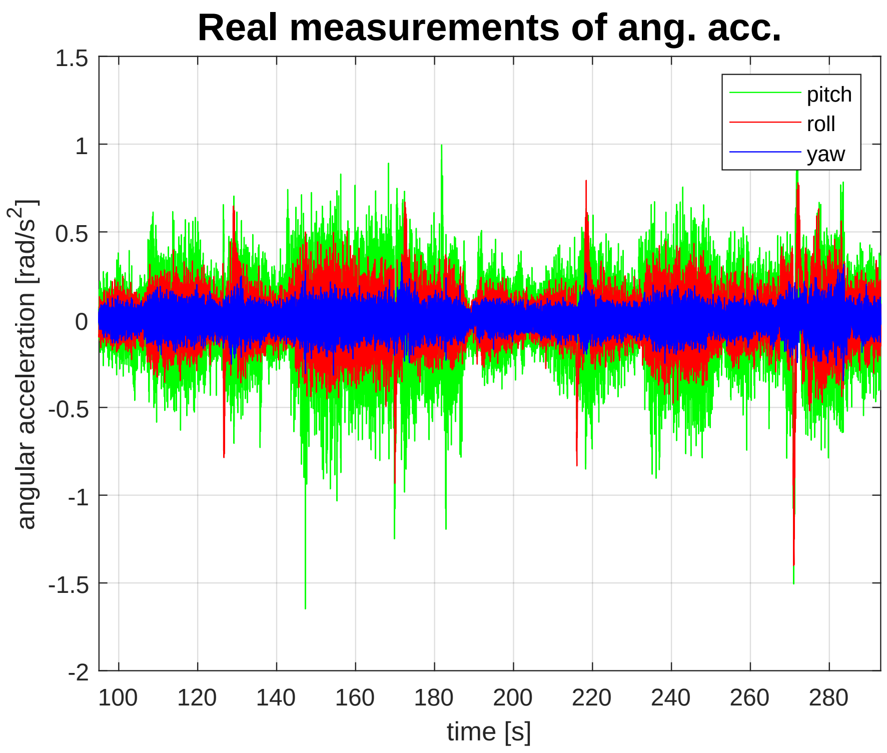


Figure 6.2: Real measurement of ang. acc.

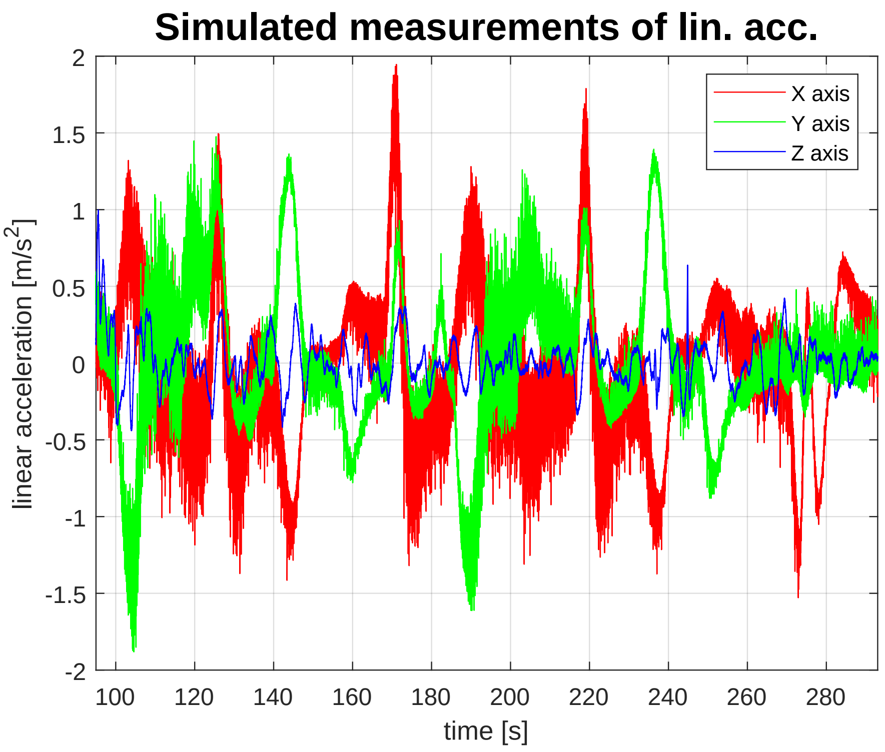


Figure 6.3: Simulated measurement of lin. acc.

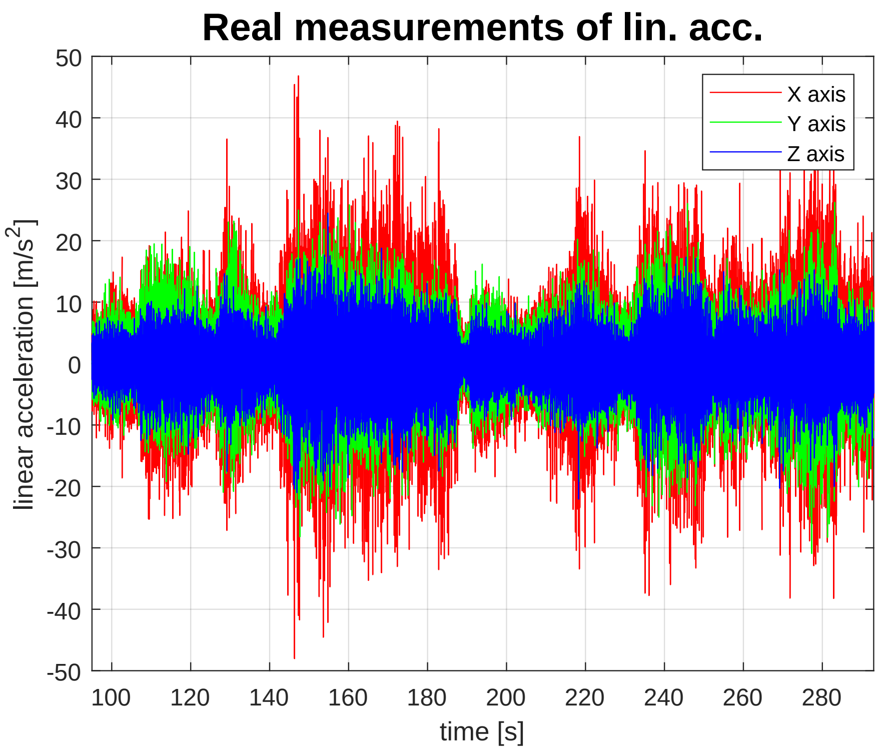


Figure 6.4: Real measurement of lin. acc.

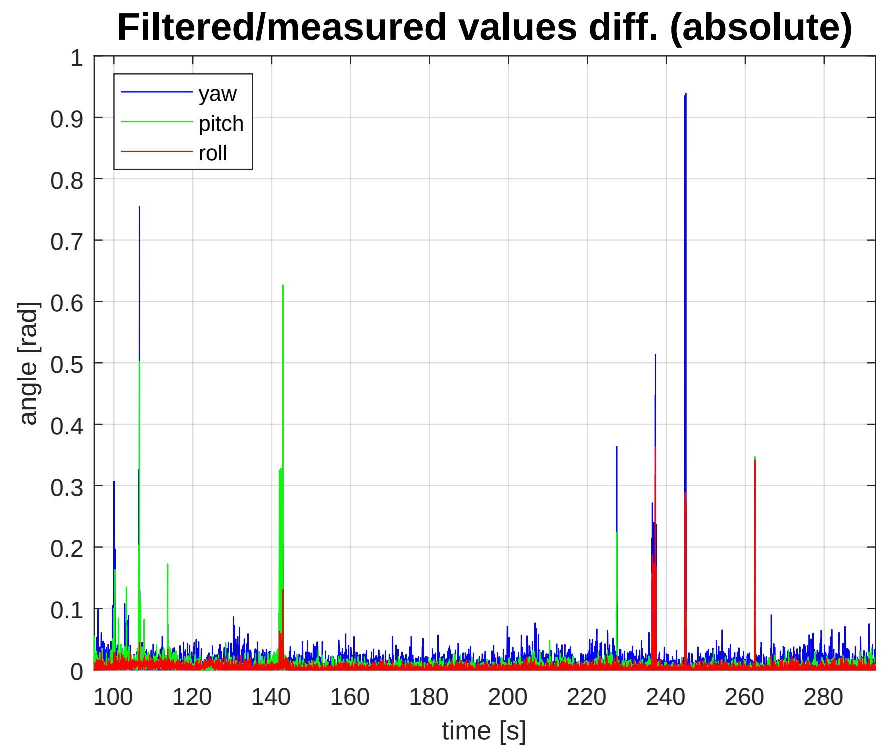


Figure 6.5: RPY values difference of 12-state system

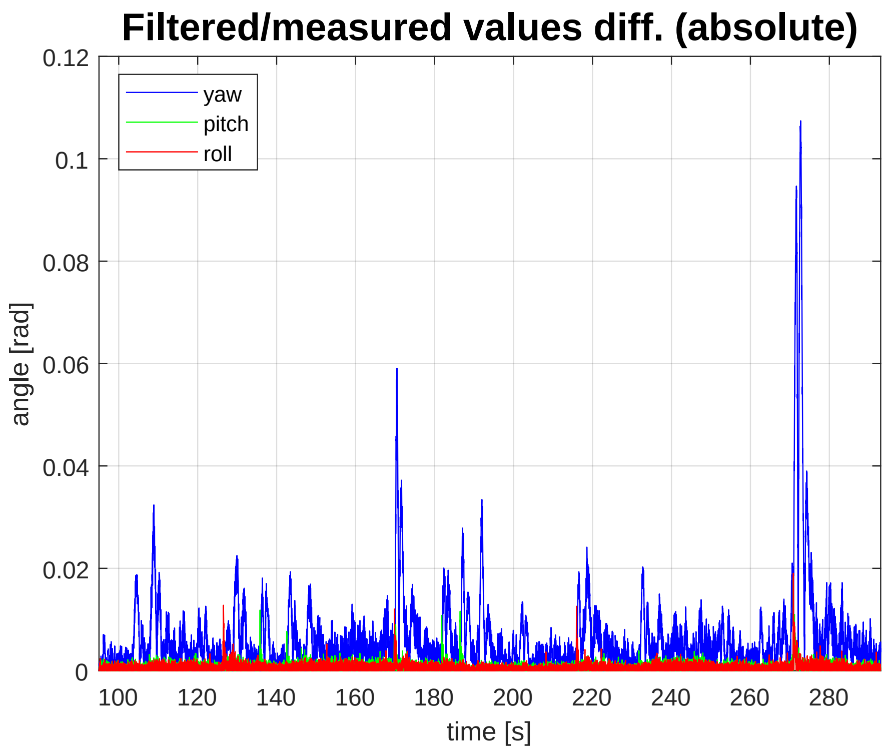


Figure 6.6: RPY values difference of 18-state system

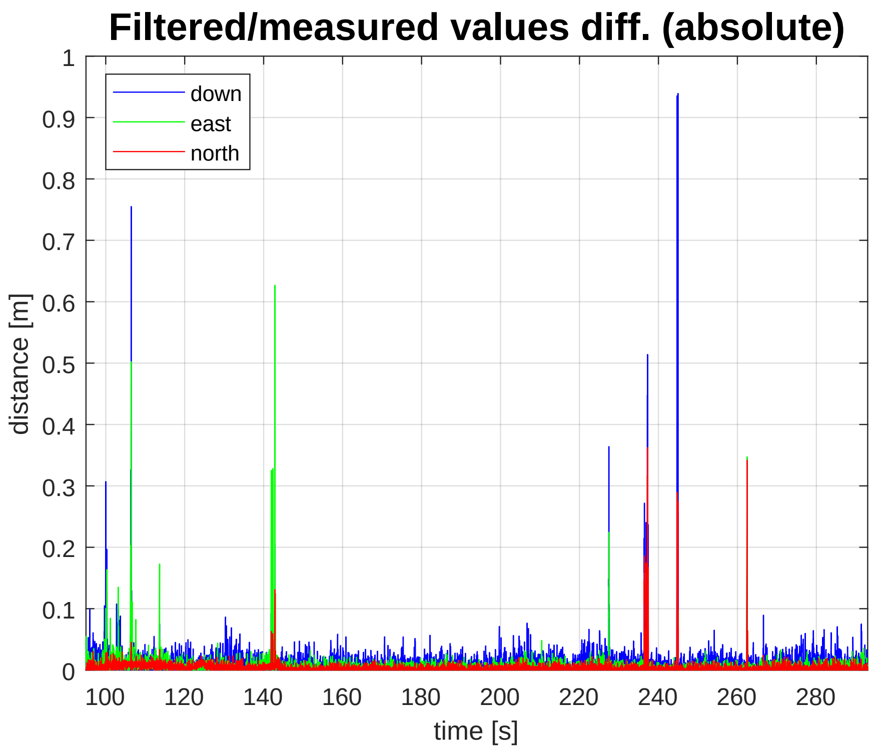


Figure 6.7: NED values difference of 12-state system

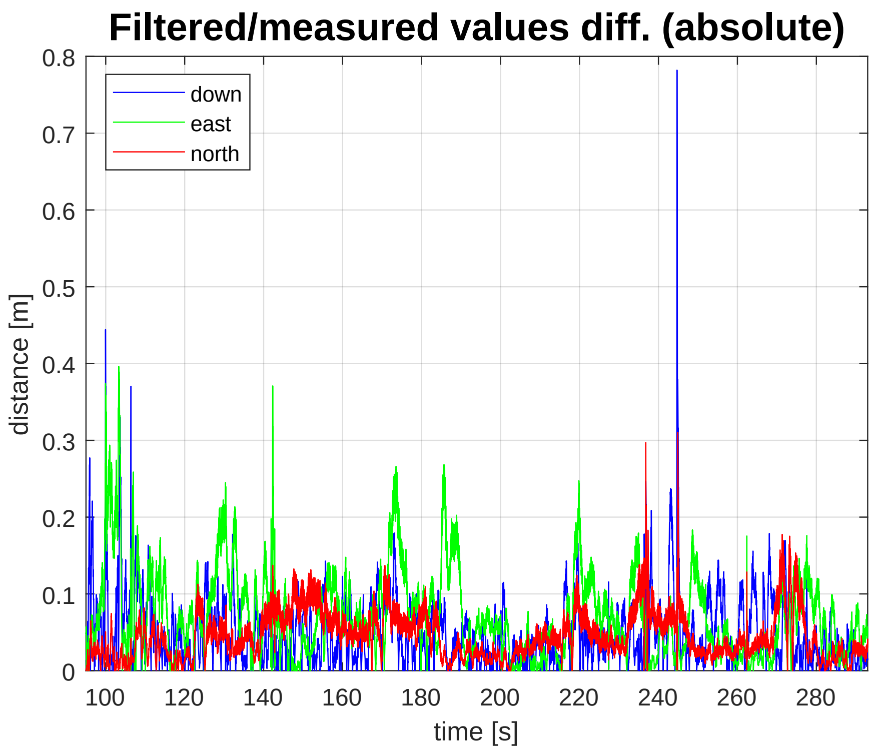


Figure 6.8: NED values difference of 18-state system

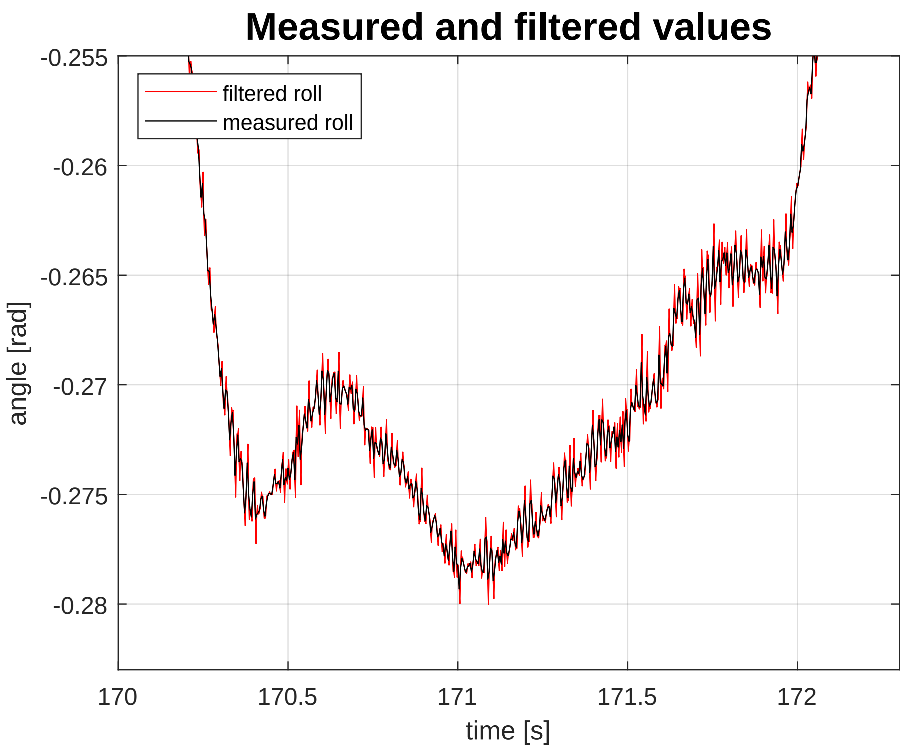


Figure 6.9: Filtered/measured roll values of 12-state system

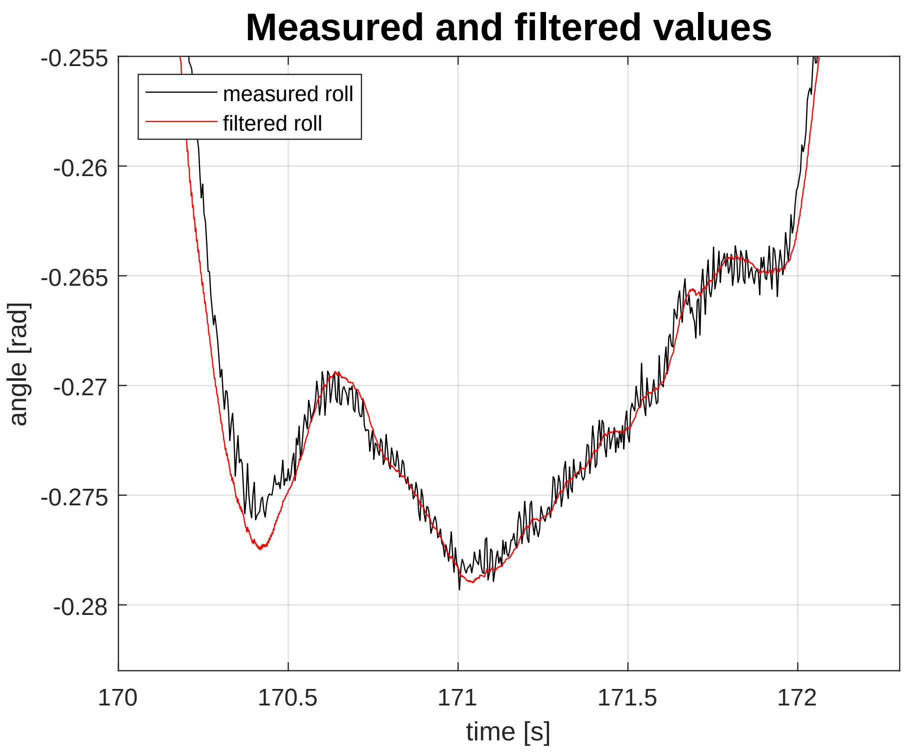


Figure 6.10: Filtered/measured roll values of 18-state system

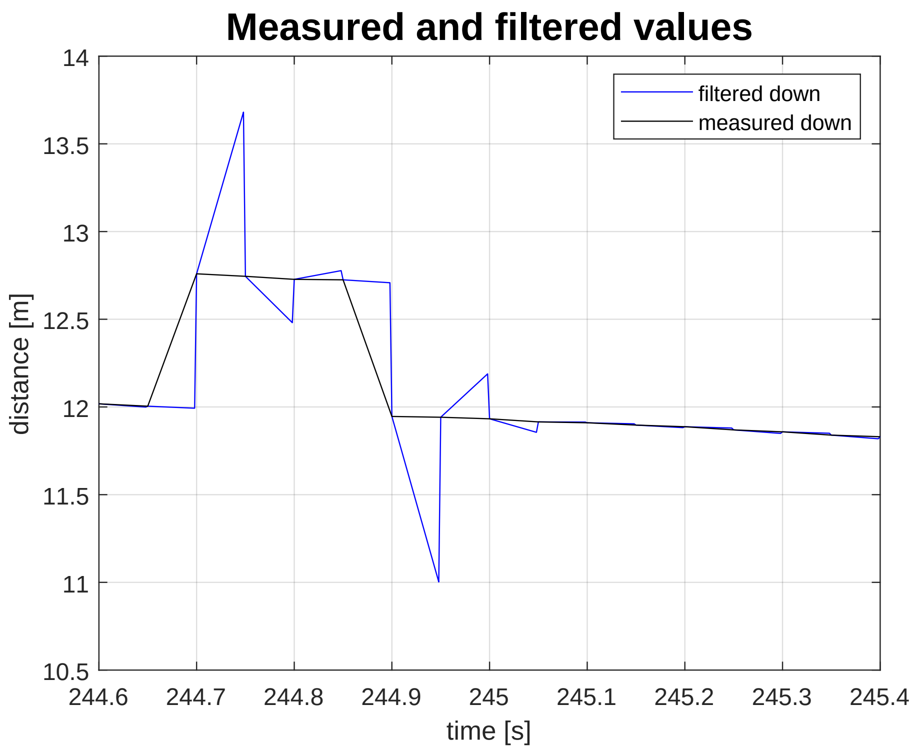


Figure 6.11: Filtered/measured down values of 12-state system

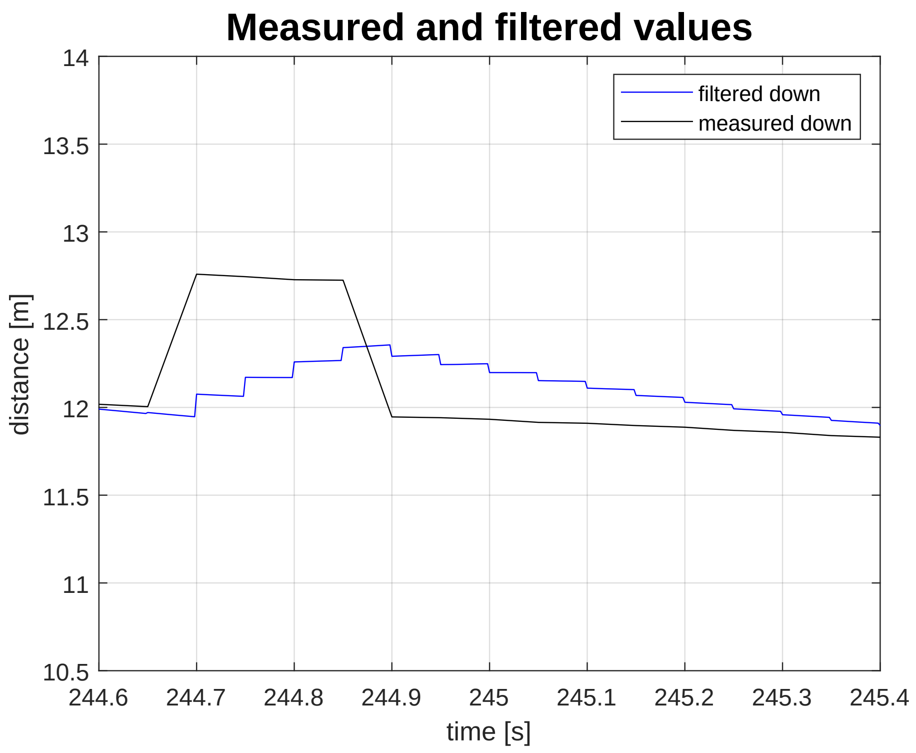


Figure 6.12: Filtered/measured down values of 18-state system

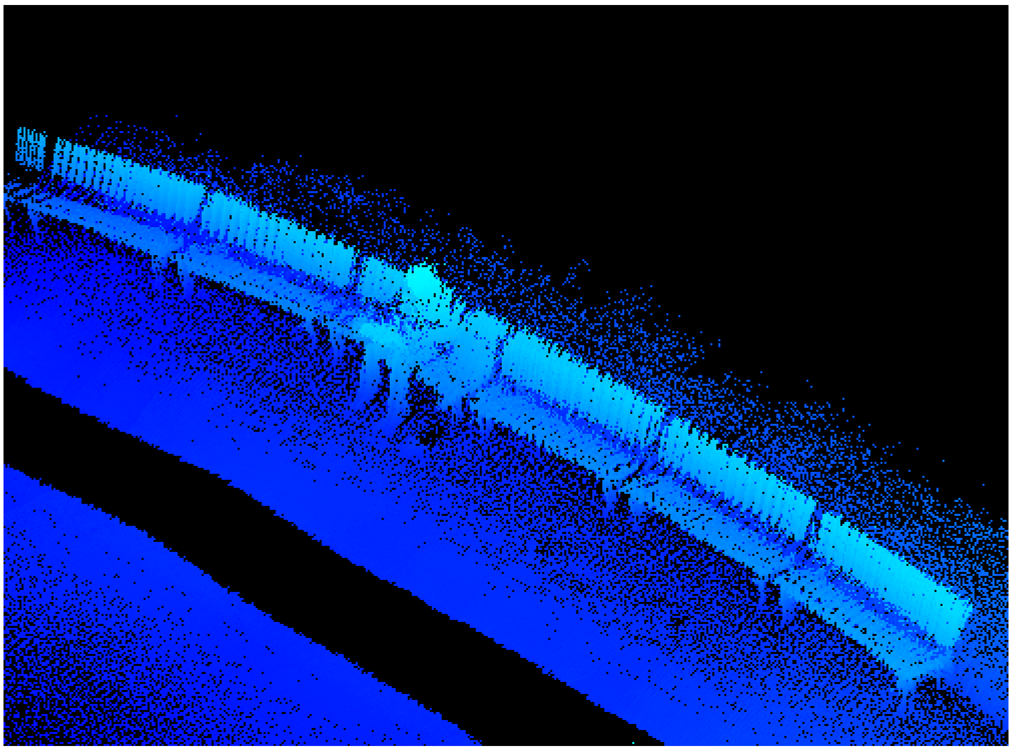


Figure 6.13: Exemplary point cloud - Operator sitting on a bench

Part IV

Conclusions

Chapter 7
Conclusions

7.1 Summary

In this work, we developed a fully operational software interface between the sensors (IMU, GNSS sensor, lidar) and the ARM computer built-in into the mapping rig to be able to record and store raw data measurements from the sensors. Previously used DGPS system involving two devices was swapped for new, more advanced, GNSS sensor which utilizes a single device. The whole application enclosing creation of a point cloud 3D model was broken down into separate modules with various functions which can be applied as necessary, either in-field or in desktop mode. Raw data analysis module was created to check the usability of the data right after its acquisition along with informative indicators provided for the user during each process done above the data. Network communication protocol for the graphical user interface added to support the raw data acquisition process was developed (GUI development itself is not within the scope of this work). GUIs for raw data analysis module and the module for construction of a point cloud were designed and implemented. All sensors are being correctly configured and measurements time synchronized properly to achieve accurate model construction. Python application was created to enclose and manage the modules with their subprocesses.

Mathematical apparatus was developed to improve the precision of the noisy measurements acquired by the sensors. Beginning with the research on localization methods and their possible improvements, the linear Kalman filtering was chosen as the best fit for our application. Matlab and Simulink programs were chosen to implement the algorithm, mainly because of simple analysis and illustration of the events/signals of the system. Another reason was the possibility to run simulations involving numerical solvers for differential equations. Kalman filter was implemented, analyzed, upgraded and finally tuned to its final form. Descriptive outputs and their evaluation were introduced to prove the successful accomplishment of the established goal.

Finally, we proposed illustrative examples to prove the correct functionality of the filtering algorithm. A presentation was done by comparison of raw measured data and the filter data in critical sections of the signals. We also intended to illustrate the localization improvement by comparison of point cloud models created from the raw data and the filtered data. However, a problem with the lidar unit appeared shortly after successful implementation of its raw data acquisition. It was not possible to acquire data from UAV in flight along with the data from lidar, because it was not working and RMA took too long for the lidar to not arrive before completion of this work.

There were some unexpected deviations from the originally planned work. The most significant one, influencing the filtering process, was the derivation and implementation of the extended mathematical model to ensure usage of all available measured data. Another mentionable issue was the time-ordering of packets from the lidar unit.

We have accomplished the desired accuracy improvement of the system's measurements. The results of the work meat their requirements to construct more precise point cloud models than with the usage of unfiltered data. The improvement lies in the removal of the noise from the measurements and making values estimates in time moments when there is no measurement available, effectively interpolating the data and providing the output data with much higher frequency. That is important because records from the lidar (acquisition of 300k points per second) should be each matched with proper position and orientation records assigned to the exact time moment.

7.2 Project applications

The presented solution was designed as a mapping system for outdoor usage. Moreover, it is limited to use with particular UAV (BRUS), because the filtering process relies on knowledge of the physical system. However, the mathematical model is not very hard to modify to fit different UAV. The system is suitable for instructed users as the proposed interfaces for operators does not require a deeper knowledge of the system.

This work was primarily dedicated to a project devoted to the mapping of forestry areas. However, it does not differentiate from other outdoor usages as long as the GPS antenna has an unobstructed view of the sky. The system is not usable in urban areas between tall buildings or indoors. To modify the system for these applications, visual odometry or wi-fi positioning system would have to be utilized to acquire the aircraft's position in such places.

7.3 Open issues

One remaining issue is a problem with post-processed data from the GNSS sensor. When switching between the two types of solution fixes, the trajectory curve experiences discontinuities. With a higher number of records outlying from the original trajectory, the Kalman filter is not able to solve this situation. Solving this problem requires deeper analysis of the post-processing algorithms done by the third party software involved. If required, implementation of another filtering process would help. It should be able to detect such "jumps" as high derivations of the values courses over time and nullify the effect to keep the smooth course of the trajectory function.

7.4 Future work

Possible future progress on this project includes:

1. Derivation of a more sophisticated mathematical model, effectively getting a better approximation of the real physical system. Such process would involve precise measurement of the UAV's physical parameters and utilization of until now neglected physical laws related to the system such as aerodynamics, the propellers dynamics, and deformation of the body.;
2. Incorporating the control signals being sent from the auto-pilot to the propellers to improve the step of prediction based on the system's evolution over time. That requires knowledge of the propellers physical parameters, to properly extend the mathematical model;
3. Utilization of more sensors (measurements) into the localization system and using their redundant information to support the estimation process further. We can utilize more subsystems of the IMU including integrated GNSS sensor, pressure sensor, and magnetometers. Also, measurements from autopilot's sensors can be used;
4. Implementation of variance information provided by some of the sensors to change the measurements variances matrices in the filtering process over time to ensure better estimation;
5. Tuning the filter with use of absolute localization station. With it, we would be able to calculate differences between filtered and real values and evaluate the filter's performance;
6. Development and testing of Extended KF and Unscented KF to see if we can more accurately estimate the states based on a non-linear mathematical model of the system.

Appendices

Appendix A
Abbreviations

All abbreviations used in this thesis in alphabetical order:

ARM - Advanced RISC Machine

ASL - Above the Sea Level

BRUS - Bezpilotní rotorový univerzální systém (Czech abbreviation)
... In English: Unmanned rotor universal system

CUI - Character User Interface

DGPS - Differential Global Positioning System

DOF - Degree Of Freedom

IR - Infra-Red

EKF - Extended Kalman Filter

ENU - East, North, Up

FOV - Field Of View

FRD - Front, Right, Down

GCS - Ground Coordinate System

GPD - Gaussian Probability Distribution

GPS - Global Positioning System

GPST - Global Positioning System's Time

GSM - Global System for Mobile communications

GUI - Graphical User Interface

ICS - Inertial Coordinate System

IMU - Inertial Measurement Unit

KF - Kalman filter

MSE - Mean-Squared Error

NED - North, East, Down

PC - Personal Computer

PP - Position and Pose

PPS - Pulse Per Second

PZ-90 - Parametry Zemli 1990 goda (Russian abbreviation)
... In English: Parameters of Earth in 1990

RGB - Red, Green, Blue

RISC - Reduced Instruction Set Computing

RMA - Return Merchandise Authorization

RPY - Roll, Pitch, Yaw angles

RSSI - Received Signal Strength Indication

RTK - Real Time Kinematic

TCP - Transmission Control Protocol

UDP - User Datagram Protocol

UKF - Unscented Kalman Filter

UTC - Coordinated Universal Time

WSG84 - World Geodetic System 1984

Appendix B
Bibliography

[AMSI16] O. A. Aqel, M, H. Marhaban, M., I. Saripan, M., and B. Ismail, N., *Review of visual odometry: types, approaches, challenges, and applications*, October 2016.

[AMY+17] N. Akai, L. Y. Morales, T. Yamaguchi, E. Takeuchi, Y. Yoshihara, H. Okuda, T. Suzuki, and Y. Ninomiya, *Autonomous driving based on accurate localization using multilayer lidar and dead reckoning*, 2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC), Oct 2017, pp. 1–6.

[CHL14] O. G. Crespillo, O. Heirich, and A. Lehner, *Bayesian gnss/imu tight integration for precise railway navigation on track map*, 2014 IEEE/ION Position, Location and Navigation Symposium - PLANS 2014, May 2014, pp. 999–1007.

[CPN16] Sandhya Rani Chapala, Gangadhara Sai Pirati, and U. R. Nelakuditi, *Determination of coordinate transformations in uavs*, 2016 Second International Conference on Cognitive Computing and Information Processing (CCIP), Aug 2016, pp. 1–5.

[CSH17] J. P. Condomines, C. Seren, and G. Hattenberger, *Invariant unscented kalman filter with application to attitude estimation*, 2017 IEEE 56th Annual Conference on Decision and Control (CDC), Dec 2017, pp. 2783–2788.

[ees14] Navipedia contributors, *Reference frames in gnss*, April 2014, http://www.navipedia.net/index.php?title=Reference_Frames_in_GNSS&oldid=12706.

[Fai09] Nathaniel Fairfield, *Localization, mapping, and planning in 3d environments*, Master's thesis, University of Oxford, January 2009.

[Fit11] Richard Fitzpatrick, *Principal axes of rotation*, ONLINE, March 2011, http://farside.ph.utexas.edu/teaching/336k/Newtonhtml/node67.html.

[FZY$^+$14] Mengyin Fu, Hao Zhu, Yi Yang, Meiling Wang, and Yu Li, *Multiple map representations for vehicle localization and scene reconstruction*, 17th International IEEE Conference on Intelligent Transportation Systems (ITSC), Oct 2014, pp. 2241–2242.

[Geo15] U.S. Deparment of the Interior | U.S. Geological Survey, *Light detection and ranging (lidar)*, January 2015, https://lta.cr.usgs.gov/lidar_digitalelevation.

[HKM13] J. Hartmann, J. H. Klüssendorff, and E. Maehle, *A comparison of feature descriptors for visual slam*, 2013 European Conference on Mobile Robots, Sept 2013, pp. 56–61.

[JHM18] N. Jeong, H. Hwang, and E. T. Matson, *Evaluation of low-cost lidar sensor for application in indoor uav navigation*, 2018 IEEE Sensors Applications Symposium (SAS), March 2018, pp. 1–5.

[Jr.18a] Roger R. Labbe Jr., *Kalman and bayesian filter in python*, vol. 531, ch. The Extended Kalman Filter, pp. 409–434, May 2018.

[Jr.18b] ———, *Kalman and bayesian filter in python*, vol. 531, ch. The Extended Kalman Filter, pp. 87–112, 147–180, May 2018.

[KA06] A. G. Kallapur and S. G. Anavatti, *Uav linear and nonlinear estimation using extended kalman filter*, 2006 International Conference on Computational Inteligence for Modelling Control and Automation and International Conference on Intelligent Agents Web Technologies and International Commerce (CIMCA'06), Nov 2006, pp. 250–250.

[Kal60] Rudolph Emil Kalman, *A new approach to linear filtering and prediction problems*, Transactions of the ASME–Journal of Basic Engineering **82** (1960), no. Series D, 35–45.

[KTH15] A. R. Khairuddin, M. S. Talib, and H. Haron, *Review on simultaneous localization and mapping (slam)*, 2015 IEEE International Conference on Control System, Computing and Engineering (ICCSCE), Nov 2015, pp. 85–90.

[LBD$^+$09] D. Lee, T. C. Burg, D. M. Dawson, D. Shu, B. Xian, and E. Tatlicioglu, *Robust tracking control of an underactuated quadrotor aerial-robot based on a parametric uncertain model*, 2009 IEEE International Conference on Systems, Man and Cybernetics, Oct 2009, pp. 3187–3192.

[LKDM17] A. Lazarov, C. Kabakchiev, A. Dimitrov, and D. Minchev, *Kalman tracking filter in 3-d space*, 2017 18th International Radar Symposium (IRS), June 2017, pp. 1–10.

[LZM17] T. Li, D. Zhu, and M. Q. H. Meng, *A hybrid 3dof pose estimation method based on camera and lidar data*, 2017 IEEE International Conference on Robotics and Biomimetics (ROBIO), Dec 2017, pp. 361–366.

[MDA07] G. Mao, S. Drake, and B. D. O. Anderson, *Design of an extended kalman filter for uav localization*, 2007 Information, Decision and Control, Feb 2007, pp. 224–229.

[Met18] C. Metz, *How driveless cars see the world around them*, The New York Times (2018).

[Nov18] NovAtel Inc., *An introduction to gnss*, April 2018, https://www.novatel.com/an-introduction-to-gnss/chapter-5-resolving-errors/real-time-kinematic-rtk/.

[O'K06] J. M. O'Kane, *Global localization using odometry*, Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006., May 2006, pp. 37–42.

[Pop98] Adrian Popa, *What is meant by the term gimbal lock?*, ONLINE, June 1998, http://www.madsci.org/posts/archives/aug98/896993617.Eg.r.html.

[Rub18] Michael Rubinstein, *Introduction to recursive bayesian filtering*, ONLINE, May 2018, https://people.csail.mit.edu/mrub/talks/filtering.pdf.

[SF17] Y. Sakuma and M. Fujii, *A study on direction estimation of movement by multiple sensors for pedestrian dead-reckoning*, 2017 Fifth International Symposium on Computing and Networking (CANDAR), Nov 2017, pp. 603–605.

[SG16] Huang Shoudong and Dissanayake Gamini, *Robot localization: An introduction*, pp. 1–10, American Cancer Society, 2016.

[SSVO09] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics*, 1 ed., vol. 632, Advanced Textbooks in Control and Signal Processing, no. 978-1-84628-642-1, Springer-Verlag London, 2009, pages 48-51.

[Str08] Jonathan Strickland, *What is a gimbal – and what does it have to do with nasa?*, ONLINE, May 2008, https://science.howstuffworks.com/gimbal.htm.

[SZC$^+$17] Z. Su, X. Zhou, T. Cheng, H. Zhang, B. Xu, and W. Chen, *Global localization of a mobile robot using lidar and visual features*, 2017 IEEE International Conference on Robotics and Biomimetics (ROBIO), Dec 2017, pp. 2377–2383.

[Thr02] Sebastian Thrun, *Particle filters in robotics*, Uncertainty in AI (UAI), 2002.

[Tra16] Tomáš Trafina, *Construction of 3d point clouds using lidar technology*, Bachelors thesis, 2016, Czech Technical University in Prague.

[TUI17] Takafumi Taketomi, Hideaki Uchiyama, and Sei Ikeda, *Visual slam algorithms: a survey from 2010 to 2016*, IPSJ Transactions on Computer Vision and Applications **9** (2017), no. 1, 16.

[Tů16] Zuzana Tůmová, *Přesná lokalizace malých bezpilotních prostředků s využitím gnss*, Bachelors thesis, 2016, Czech Technical University in Prague.

[Vel18] Velodyne Acoustics Inc., *Vlp-16 user manual and programming guide*, 63-9243 rev. d ed., 2018.

[Wik18] Wikimedia Foundation Inc., *Gimbal lock*, January 2018, https://en.wikipedia.org/wiki/Gimbal lock.

[WM00] E. A. Wan and R. Van Der Merwe, *The unscented kalman filter for nonlinear estimation*, Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373), 2000, pp. 153–158.

[YDCPC17] Simon Yiu, Marzieh Dashti, Holger Claussen, and Fernando Perez-Cruz, *Wireless rssi fingerprinting localization*, Signal Processing **131** (2017), 235 – 244.

[YJC12] Chuho Yi, S Jeong, and J Cho, *Map representation for robots*, 18–27.

[YM05] D. C. K. Yuen and B. A. MacDonald, *Vision-based localization algorithm based on landmark matching, triangulation, reconstruction, and comparison*, IEEE Transactions on Robotics **21** (2005), no. 2, 217–226.

[YZF13] S. Z. Yong, M. Zhu, and E. Frazzoli, *Simultaneous input and state estimation for linear discrete-time stochastic systems with direct feedthrough*, 52nd IEEE Conference on Decision and Control, Dec 2013, pp. 7034–7039.