ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Ultra-Wideband Localization Error Correction using Gaussian Process for UAV Flight

Master Thesis

Johann Diep

Fall 2019

Advisor: Dr. Tobias Nägeli, Tinamu Labs

Prof. Dr. Otmar Hilliges, Advanced Interactive Technologies, ETH Zürich

Abstract

In recent years, unmanned aerial vehicles (UAV) have gained more interests in the research community. Due to their improved videography capabilities, they have the potential to bridge the gap between precise and flexible filming. However, their current localization methods are not reliable enough for tasks such as the following of a predefined trajectory.

Motivated by this gap, the work in the context of this thesis is to establish an accurate localization system for UAVs based on commercially available ultra-wideband (UWB) technology. It describes the implementation of a system based on filtered multilateration using distance measurements between the drone-mounted tag and several environment-fixed anchors. Here, methods are presented to estimate the anchor placements as well as to correct for the systematic biases of the range readings. The former is solved by means of Gauss-Newton optimization using inter-anchor distance estimations. For the latter, in order to correct for the systematic distance measurement errors, a learning-based framework is proposed, where Gaussian Process models are derived from training data given by a motion capture system and incorporated in the corresponding equation of the state estimation algorithm.

For validating the performance of the developed localization algorithm, various experiments were conducted. As a baseline, the developed method was compared to filtered multilateration without error modeling. It is shown, that the presented approach improves the accuracy of predefined trajectory following. It is able to capture the accuracy of an advanced localization system and therefore provides a long-term replacement solution for environment-independent learning-based flight applications.

Keywords: Ultra-Wideband, Localization, Unmanned Aerial Vehicle, Gaussian Process, Extended Kalman Filtering, Gauss-Newton Optimization, Guidance, Navigation, Control.

Acknowledgements

This thesis was conducted at the *Advanced Interactive Technologies* Lab at *ETH Zurich* and in collaboration with its spin-off company *Tinamu Labs*. First, I would like to thank my supervisors Dr. Tobias Nägeli and Martin Rutschmann for their continuous guidance, support and patience. Also thanks to Anton Ledergerber from the *Institute of Dynamic Systems and Control* for taking his time for my questions and remarks. I also would like to express my gratitude to Prof. Dr. Otmar Hilliges for providing me with the opportunity as well as the infrastructure to conduct this final thesis of my academic education. My gratitude also goes to all my friends and family for supporting me during this time. Lastly, a special thanks goes to the safety net in the flight area for safely catching *Bebop* whenever it gained self-awareness and tried to attack me or other students nearby.

By signing this statement, I confirm that I read the information notice on plagiarism, independently produced this thesis and met the general practice of source citation in this subject area.

Zurich, 20. December 2019

Johann Diep

Contents

List of Figures

Chapter 1

Introduction

1.1 Tinamu Labs

This master thesis project was conducted in collaboration with *Tinamu Labs*[1]. This *ETH Zurich* spin-off focuses on the commercialization of drone localization and control technology for image and data gathering in new markets such as movie production, sports broadcasting, indoor inspection and warehouse management.

1.2 Motivation

In order to create stable tracking shots, the directors from the media and entertainment industry still rely heavily on large equipment such as dollies or cranes (see figure 1.1). In the first case, the camera is mounted on a wheeled platform rolling along a predefined track. With the latter, the camera is placed at the top of an extended remote-controlled arm. In both methods, the idea is to separate the translational from the rotational camera motion. The former can be automated such that the operator is only responsible for capturing the perfect angle. Moreover, filming often requires more than one take of the same scenery. Therefore, having the camera path fixed enables precise repetitive shootings. However, these methods are very expensive and inflexible. Due to their magnitude in size, transportation and setup are cost- and time-intensive. In contrast, hand-held camera stabilizer mounts such as *Steadicam*[2] offer an alternative more flexible solution at the expense of having more degrees of freedom to control while filming, diminishing exact re-shootings of an entire scene.

[1] Source: http://www.tinamu-labs.com/wp/ 18. October 2019

[2] *Steadicam* is a brand of camera stabilizer mounts for motion picture cameras (Source: https://www.tiffen.com/pages/steadicam/ 3. October 2019).

Figure 1.1: Dolly and crane setup[3]in modern film productions or sports broadcasting for smooth camera movements.

Figure 1.2: The *DJI Inspire 2* drone[4]is comprised with a panoramic rotating camera gimbal for professional filmmaking.

In recent years, UAVs have gained more attentions in the research community due to their high maneuverability and applicability. Because of their improved videography capabilities (see figure 1.2), they offer the potential of bridging the gap between precise and flexible filming. However, their current positioning systems are not reliable enough for tasks such as prespecified trajectory following. As described by Siegwart and Nourbakhsh [26], a multitude of possible localization methods have been researched with the goal to fill this gap. Here, the work in the context of this thesis is to establish an accurate localization system for UAVs based on commercially available UWB technology.

[3]Images taken from: https://www.macleans.ca and https://www.nofilmschool.com 18. October 2019

[4]Image taken from: https://www.sleeklens.com 12. November 2019

1.3 Outline

This thesis describes the implementation of an accurate localization system for UAVs based on UWB range measurements with commercially available chip- and antenna modules. A moving drone-mounted UWB module, hereafter called tag, can be located by ranging to environment-fixed UWB modules, hereafter called anchors, whose positions are surveyed in advance. Knowing the distance measurements between tag and anchors, the position can then be determined by means of multilateration[5]. Geometrically, the tag is located at the intersection of circles defined by the anchor positions and the measured ranges. However, manual measurement of the anchor positions is a time-consuming process. Additionally, distance measurement biases are, for instance, correlated with the poses of the ranging modules (see figure 1.3).

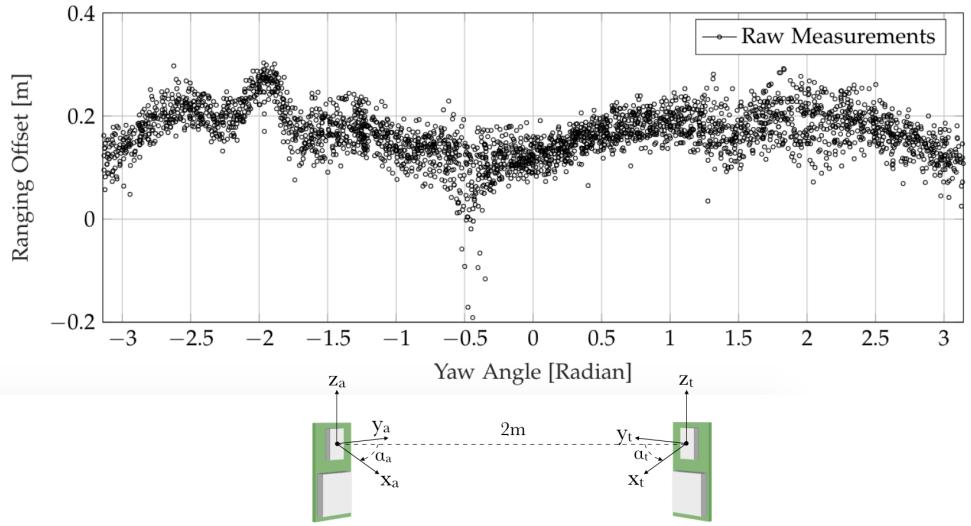

Figure 1.3: Ranging experiment with two UWB modules (anchor on the left and tag on the right) fixed at 2 meters distance from each other. As illustrated, the orientation of the anchor was set to $\alpha_a = 60°$. Measurement data were then acquired while the tag unit was rotated along its z-axis. The offset, which indicates the deviation from the expectation, is visibly correlated with the corresponding orientation of the tag.

Therefore, the core of this thesis focuses on a method for anchor setup self-calibration as well as the stochastic modeling of the ranging offsets given labeled training data from an accurate motion capture system. The final product captures the accuracy of an advanced localization system and provides a long-term replacement solution for environment-independent flight applications.

[5]In contrast to angulation, where position is determined by a set of angle measurements.

1.3.1 Topics

The remainder of this thesis is organized as following:

- The subsequent sections within this chapter gives an overview of the mathematical notations, the interacting modules of the developed localization framework, the related works, the used hardware and the applied coordinate system conventions.
- Chapter 2 introduces the preliminaries and mathematical foundations.
- Chapter 3 describes the core methods that are building up on each other. In order to determine the anchor positions within the flying space, an anchor self-calibration procedure is proposed. Furthermore, an explanation of the baseline guidance, navigation and control framework is given. Lastly, the incorporation of the stochastic models which denote the individual ranging errors is described.
- The performance of the anchor self-calibration procedure and the learning-based approach in localization and trajectory following accuracy respectively are evaluated in chapter 4 with numerous experiments.
- Lastly, a summary of the achievements with a short outlook is given in chapter 5.

1.4 Mathematical Notations

This section provides few small notes on the used mathematical notations in this report in order to avoid confusions. In general, whenever a nomenclature symbol is duplicated, their respective meaning should be clear from the context. Furthermore, if not specified otherwise, coordinates as well as distances are given in meters, covariances in square meters and time specifications in seconds. Additionally, the following notation rules are made:

- throughout this report, variables, which can be substituted with numerical values or expressions,
- as well as the denomination for coordinate frames are italicized,
- vectors are denoted with bold lowercase characters,
- matrices with bold uppercase characters,
- discrete-time dependency is described with a subscript k,
- and continuous-time dependency with a subscript t.

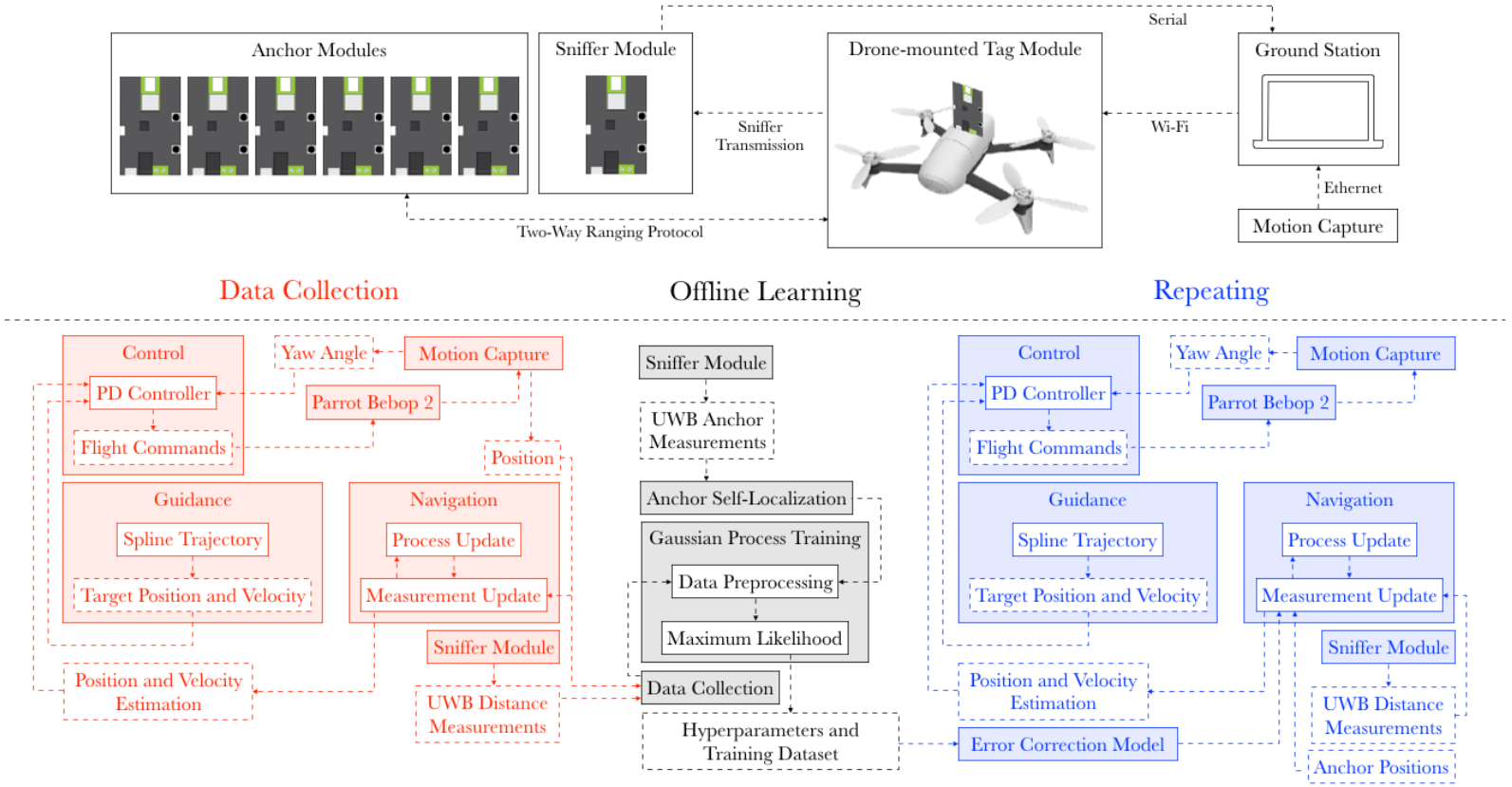

Figure 1.4: Flowchart of the used hardware and their corresponding connections as well as the three interacting phases for the learning-based flight application. Thereby, the data collection, learning and repeating phases are indicated in red, black and blue respectively.

1.5 Workflow

This section gives a brief summary of the workflow for the learning-based flight application. A graphical illustration is depicted in figure 1.4 as a flowchart which helps understanding the explained concepts. Detailed explanation of the implementation keypoints can be found in chapters 3. The overall pipeline consists of three interacting modules running in sequential order:

- Data collection phase: This routine is responsible for collecting the necessary data for the learning phase. Thereby, the UAV is commanded to autonomously follow a predefined trajectory using precise pose feedback from a motion capture system while distance measurement data are collected from the interrogations with the anchors.
- Learning phase: Given the anchor positions via the self-localization procedure, the collected ground-truth positions of the UAV as well as the corresponding UWB data, labeled training data can be generated and used to learn the hyperparameters of the stochastic ranging error models. This process is completed offline between the data collection and repeating flight.
- Repeating phase: During this phase, the previous defined trajectory is followed while only UWB measurements are used for position estimations. Thereby, the learned error models from the previous phase provide correction terms for the ranging measurement equations in the estimation algorithm.

1.6 Related Work

There are many publications on beacon-based localization using UWB technology for various mobile robotics applications [1]. Thereby, the achieved positioning accuracy depends on the quality of the measured distances. A variety of past works have been dealing with correcting the process of the distance estimation per se. This ranges from empirical determined lookup tables for the signal strength-dependent correction terms [7], sophisticated leading edge detection algorithms [22] to the identification and mitigation of non-line-of-sight effects [2].

In contrast, not much research have been conducted in mathematically modeling the systematic error of the range estimations (see subsection 2.1.5) noticeable in the usage of commercial UWB modules such as the *Loco Positioning Node* (see subsection 1.7.1). In this context, Ledergerber and D'Andrea [17] propose applying Gaussian Process method to model the biases based on gathered training data. Specifically, their training data are collected by

commanding an UAV equipped with a tag module to follow random trajectories within the flight space while constantly logging range data with all nearby anchors. In doing so, an accurate motion capture system is used to capture the positions and attitudes of the drone over time. Thereby, it is ensured to cover as much of the flight domain as possible. With the collected training data, models can be derived which describe the probability distributions of the biases for a specific position and attitude, such that free movements within the flight domain is possible. By augmenting the state estimator with the derived Gaussian Process models as described by Ko and Fox [15], it is shown that the method enhances the localization accuracy compared to conventional uncorrected multilateration approaches.

Hence, this thesis explores the applicability of their proposed procedure specifically for pre-determined path flights. More precisely, it is researched whether error models derived from gathered training data only along the desired path can allow for accurate repetition while solely relying on raw UWB measurements for pose estimations.

1.7 Hardware and Platforms

This section gives a brief overview of the used hardware and platforms. A graphical illustration of the their connections can be found in figure 1.4. It is to mention that the core of the developed learning-based framework is independent of the specific hardware setup. In principle, any mobile robot and localization method can be used in combination with an accurate motion capture system.

1.7.1 Loco Positioning Node

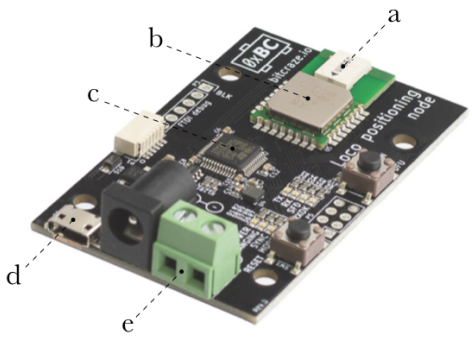

Figure 1.5: *Loco Positioning Node*[6] with its integrated wireless transceiver module: (a) antenna, (b) *DWM1000* module, (c) microcontroller, (d) micro-USB and (e) screw terminal connector.

[6] Image taken from: https://www.bitcraze.io 18. October 2019

The *Loco Positioning Node* is a low-cost multifunctional UWB positioning device created by *Bitcraze* (see figure 1.5). It is equipped with a *Decawave DWM1000* wireless transceiver module [9] and can either act in anchor, tag or sniffer mode (the latter was modified as described in appendix A.2). With multiple powering options (micro-USB, DC-jack or screw terminal connector) and an on-board microcontroller, distances are directly calculated from time-of-flight (ToF) measurements. By means of an open-source serial communication tool[7], these data can be retrieved directly at the interrogating tag by means of a micro-USB connection to the computer. However, for the main part of this project, they are collected over radio by connecting a single module in modified sniffer mode to the ground station. The anchors are distributed on wooden poles (see figure 3.1) and powered via their terminal connectors with portable chargers. Lastly, one tag module is mounted on the drone as described in subsection 1.7.3. For this project, unless specified otherwise, the UWB radio settings (bitrate, preamble and transmit power) are kept at their default configurations.

1.7.2 Parrot Bebop 2

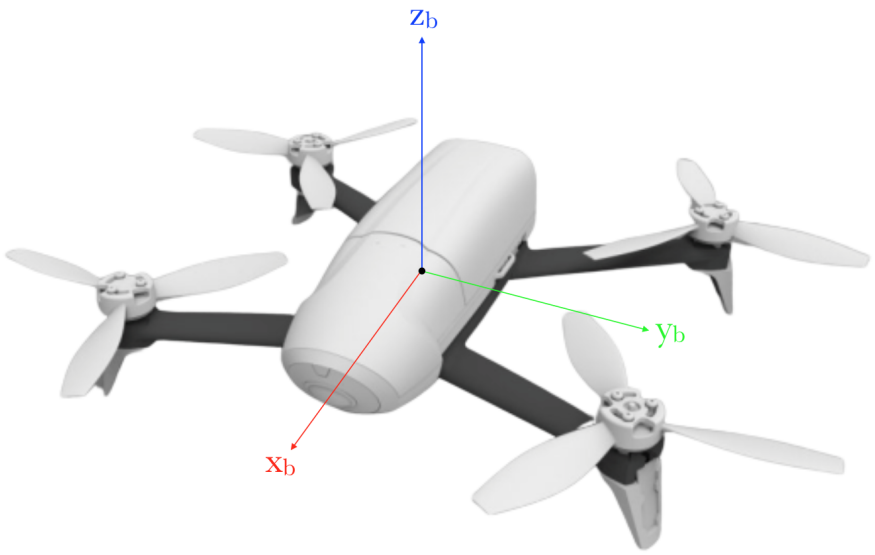

Figure 1.6: A model[8] of the *Parrot Bebop 2* drone and its corresponding body-fixed frame defined by the *Parrot's* SDK. It is equipped with a camera pointing in the positive x-direction.

The experiments were carried out on the *Bebop 2* drone (see figure 1.6). It is a small radio-controlled quadcopter made by *Parrot* which uses four body-

[7] `picocom` was used for serial communications
(Source: https://linux.die.net/man/8/picocom 7. October 2019).

[8] Image taken from: https://www.3printr.com 31. October 2019

fixed rotor blades to create lift for maneuver, vertical take-off and landing. Due to its small size and low overall weight, it can achieve up to 25 minutes of active flight time[9].

Utilizing the `bebop_autonomy`[10] *ROS*[11] driver, movement commands can be transmitted over Wi-Fi by sending a specific message corresponding to the following physical inputs from the ground station:

- the reference pitch angle θ,
- roll angle φ,
- vertical velocity v_z
- and yaw rate ω_ψ.

Specifically, the message contains four numbers within the interval $[-1, 1]$ corresponding to the signed percentage of the maximal physical parameters[12] θ_{max}, φ_{max}, $v_{z,max}$ and $\omega_{\psi,max}$ respectively.

The SDK defines a body-fixed frame $\mathcal{B}$ as shown in figure 1.6, where a conventional right-handed coordinate system with origin lying at the center of gravity of the aircraft is used. The main direction is along the longitudinal axis, whereas the front-facing camera is pointing in the positive x-direction. The z-axis is chosen to point upwards. A positive pitch angle θ input results in a positive rotation around the y-axis according to the right-hand-rule, which moves the drone forward in positive x-axis. The same applies for a positive yaw rate ω_ψ input which results in positive rotation around the z-axis. However, a positive roll angle φ input leads to a negative rotation around the x-axis, which moves the drone in positive y-axis. It can be assumed, that the on-board controller keeps the height constant during a pitch, roll and yaw command. Lastly, a positive vertical velocity v_z input moves the drone towards positive z-direction.

1.7.3 Complete Assembly

As illustrated in figure 1.7, the UWB tag is attached vertically on top of the drone with a custom 3D-printed mount in order to provide line-of-sight conditions with most anchors. By stepping down the power from the drone

[9] According to official documentation
(Source: https://www.parrot.com/us/drones/parrot-bebop-2 6. October 2019).

[10] Open-source *ROS* driver for the *Parrot Bebop 2* drones based on the official SDK
(Source: https://bebop-autonomy.readthedocs.io/en/latest/index.html
6. October 2019).

[11] Throughout this project, *ROS Kinetic* for *Ubuntu 16.04* was used
(Source: http://wiki.ros.org/kinetic/Installation/Ubuntu 8. October 2019).

[12] According to official documentation of the `bebop_autonomy` driver, those can be reconfigured to some degree.

[13] Image taken from: https://www.3dconnexion.com 2. November 2019

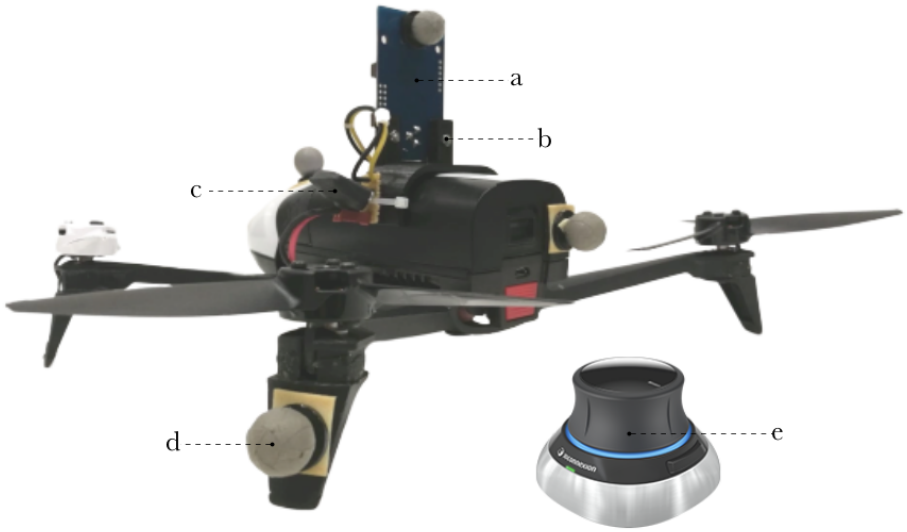

Figure 1.7: The complete assembled drone setup: (a) *Loco Positioning Node*, (b) 3D-printed mount, (c) switching voltage regulator, (d) *Vicon* reflective markers and (e) *Spacemouse*[13].

battery to 5V using a switching voltage regulator[14], the module is powered on-board without adding additional batteries so that the added weight is limited to a minimum. An additional *Spacemouse*[15] joystick with its corresponding *ROS* driver[16] is used to quickly launch and land the drone on command.

1.7.4 Vicon Motion Capture System

Vicon is an accurate and precise 3D motion capture system. Several high-speed infrared cameras are permanently installed over the flying space in the research lab and used to track the motions of reflective markers placed on a rigid object such as the quadcopter (see figure 1.7). Thereby, the positions and orientations of the drone during a flight can be determined. Specifically, the orientation of the body-fixed with respect to the inertial frame (see section 1.8) is given in quaternion form. By connecting the ground station to the *Vicon* machine over a router via Ethernet and using the open-source `vicon_bridge`[17] *ROS* driver, these measurements are published on a *ROS* message topic.

[14]Source: https://www.dimensionengineering.com/products/de-sw050 18. October 2019
[15]Source: https://www.3dconnexion.com/spacemouse_wireless/en/ 18. October 2019
[16]Source: http://wiki.ros.org/spacenav_node 8. October 2019
[17]Source: https://www.github.com/ethz-asl/vicon_bridge 6. October 2019

Ground-truth data are sampled at high-frequency (up to 200Hz) from *Vicon* for three purposes:

- to control the yaw angle of the drone during a flight maneuver, avoiding erroneous estimations from inaccurate on-board sensors,
- to generate labeled training data for constructing the error models
- and to evaluate the accuracy of the estimated positions by means of filtered UWB multilateration.

1.8 Coordinate Systems

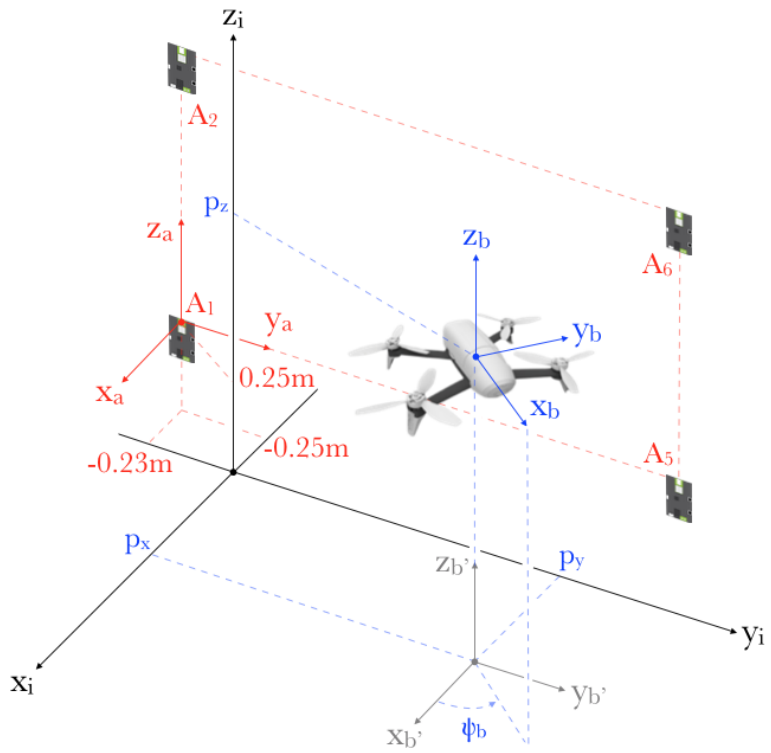

Figure 1.8: The coordinate systems for this work illustrated: (i) *Vicon* inertial frame (black), (a) anchor frame (red) and (b') initial object-fixed frame (gray) and (b) body-fixed frame (blue).

This section explains the coordinate conventions used for the methods and the experiments described in chapters 3 and 4 respectively. Figure 1.8 illustrates all the coordinate systems which are explained in the following.

A total of 6 anchor modules are used for this project. The anchor frame $\mathcal{A}$ is constructed based on the placement of the UWB anchor poles, where two anchors are mounted at the bottom and at the top of each pole (refer

to subsection 3.1.1 for further details). For simplicity it is assumed that the antennas A_1, A_2, A_5 and A_6 are located on a vertical plane, such that there is no horizontal displacement between the lower and the corresponding upper anchor. Thereby, A_1 and A_5 are placed at the bottom while A_2 and A_6 at the top with the same heights respectively. Based on those assumptions, the origin of the anchor frame is set at the antenna center of anchor A_1. The y-axis is defined by the A_1-A_5 antenna line-of-sight, the z-axis by the A_1-A_2 antenna line-of-sight and the x-axis according to the right-hand-rule.

With the *Vicon* calibration wand, the origin of the right-handed inertial frame $\mathcal{I}$ is placed in front of the anchor frame as denoted in the image such that there is no rotation between the frames. Unless specified otherwise, all position and velocity estimations from here on are described in this inertial frame.

An initial object-fixed coordinate frame $\mathcal{B}'$ is defined by the motion capture system at object creation by grouping all the reflective markers on the rigid body in the *Vicon* GUI. At start, this frame is a displaced coordinate system sharing the same orientation as the inertial frame. Thereby, the origin is defined as the geometric center of the grouped markers. In order to align this object-fixed with the body-fixed frame of the *Bebop* (compare with subsection 1.7.2), the drone is placed such that its front-facing camera is pointing in the positive direction of the x-axis of the inertial system before initialization. Moreover, the markers are placed around the drone such that the geometric center of the grouped markers approximately corresponds to the center of gravity of the UAV. The *Vicon* system then describes the spatial change of the body-frame over time as its translation and rotation with respect to the inertial frame.

Chapter 2

Preliminaries

This chapter includes all the preliminaries and mathematical foundations to understand the work in this thesis. The information are kept short and there is no intended order or dependencies between the different sections. Whenever a theory is implemented or expanded in later chapters, a reference is made to the specific passage here. Hence, the reader can also choose to skip this chapter completely and use it only as a reference later on if required.

2.1 Ultra-Wideband Technology

The following section attempts to provide a short overview on the theory and the shortcomings of modern UWB technology. For more information, the interested reader is referred to the literature by Sachs [14].

2.1.1 Bandwidth-Timing Tradeoff

Heisenberg's uncertainty principle correlates the frequency with the timing of a signal as follows:

$$\Delta f \Delta t \geq \frac{h}{4\pi} \tag{2.1}$$

Thereby, h is the Plank constant and the range of frequencies Δf used for a signal is called bandwidth. In order to achieve narrow pulses Δt, the bandwidth need to be above a certain threshold, such that accurate timing is possible. UWB is a communication method in wireless networking which achieves high bandwidth connections while keeping the transmit power very low. Therefore, the resulting timing resolution is fine enough to distinguish the direct path signal from other reflected signals in the channel impulse response.

2.1.2 Time-of-Flight Principle

In this thesis, commercially available UWB modules (see subsection 1.7.1) are used for the localization of a quadcopter in a local coordinate system by means of multilateration. Thereby, the distances between the drone-mounted tag and each fixed anchor node in the environment are required. The ToF principle is a method for measuring the distance between a transmitter and a receiver based on the time a signal takes to travel between them. A simple method to determine ToF t_{flight} is to subtract the time of signal transmission $t_{\text{transmission}}$ at the transmitter from the time of signal reception $t_{\text{reception}}$ at the receiver as follows:

$$t_{\text{flight}} = t_{\text{reception}} - t_{\text{transmission}} \tag{2.2}$$

Because time in this context is related to propagation of radio waves, the distance $d_{\text{measurement}}$ between tag and anchor can be estimated using the speed of light in vacuum c:

$$d_{\text{measurement}} = t_{\text{flight}} \cdot c \tag{2.3}$$

Since the signals travel approximate at the speed of light, exact and synchronized time stamping is central for proper results[1]. However, this requires high precision clocks as well as cross-device synchronization, which increase the complexity of the overall system. Specifically, the clocks of the interrogating modules need to be run at the same rate. In contrast, by following a two-way ranging approach (see subsection 2.1.3), it is shown that the extent of clock differences can be mitigated.

2.1.3 Two-Way Ranging Protocol

This subsection introduces the two-way ranging protocol [8] implemented in the firmware of the *Loco Positioning Node* for a more robust ToF estimation. As shown in figure 2.1, the tag initiates the protocol by transmitting a poll packet at timestamp t_1. The default protocol contains four consecutive messages between tag and anchor, whereby the last message contains a report of all the timestamps from the anchor. The flight time t_{flight} is then calculated using the following formula:

$$t_{\text{flight}} = \frac{(t_4 - t_1)(t_6 - t_3) - (t_3 - t_2)(t_5 - t_4)}{(t_5 - t_1) + (t_6 - t_2)} \tag{2.4}$$

[1]Note that a timing error of 1 nanosecond already lead to a distance error of around 0.3 meters.

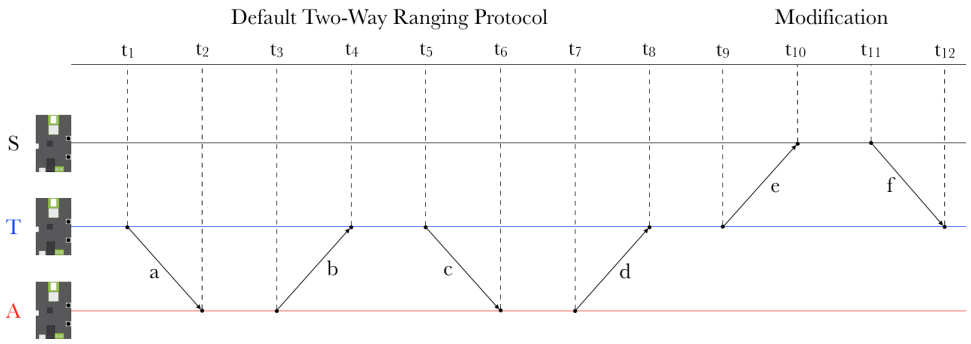

Figure 2.1: The two-way ranging protocol used for ToF estimation between a tag (blue) and an anchor (red) module. The default protocol consists of the (a) poll, (b) answer, (c) final and (d) report message. The extended protocol with a sniffer (black) includes further messages, such as the (e) result and (f) switchback message, which is explained in appendix A.2.

Here, t_i with $i = 1 : 6$ are the timestamps of signal transmission and reception at the tag and anchor respectively. After a completed two-way interaction with one anchor, the tag calculates the ToF and starts initiating the protocol with the next anchor.

In general, a transmitted message is received by all the modules within the UWB network. The firmware of the *Loco Positioning Node* enables a targeted information exchange between two modules by requiring a message to contain the unique address of the destination module, hereafter called device index. These device indices must be set by the user once at system setup. Thus, a module can receive messages by filtering for its own specific index. Furthermore, the message also contains the individual nametag (poll, answer, final and report) which triggers a specific execution at the destination node.

2.1.4 Documented Sources of Estimation Error

First practical tests demonstrate a deviation of the measurements from the actual distances. For this project, this error needs to be considered in order to achieve acceptable accuracy for flight applications. According to official documentations of the *DWM1000* wireless transceiver module, three main error sources are identified which mitigate the distance estimation accuracy.

One contribution stems from the antenna delay [10], which describes the time it takes for a signal to propagate through the antenna at the transmission or reception module respectively. Neglecting this effect results in an inaccurate estimate of the actual ToF between the antennas, since it integrates a time lag to the actual timestamps (see figure 2.2). This delay factor is different for each module due to distinctions in manufacturing. The firmware of the *Loco Positioning Node* contains a parameter that allow for individual

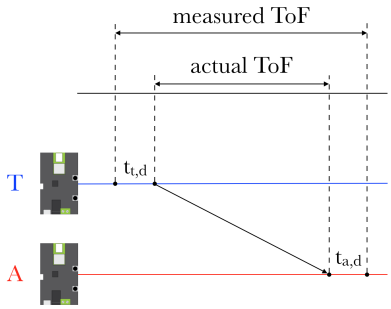

Figure 2.2: The influence of the antenna delays $t_{t,d}$ and $t_{a,d}$ of the transceiver and the receiver module respectively on the measured ToF. As can be seen, the signal is not immediately transmitted at timestamping. On the downside, the signal is not recorded immediately at reception.

tuning of the delay value. Furthermore, *Decawave* suggests a sophisticated calibration procedure based on mathematical optimization to determine the specific value for each device. In principle, by placing the UWB modules at known distances from each other, the individual delay values are systematically varied until the deviations between the actual distances and the corresponding means of the measurements are minimized.

As indicated in subsection 2.1.2, clock differences can also cause erroneous estimations. Furthermore, a frequency offset of an oscillator from its reference value gives rise to a clock drift [7] and therefore directly affects the accuracy of the timestamping process. However, it can be shown that the effect can be mitigated by following the pre-implemented two-way ranging protocol (see subsection 2.1.3), since the distance estimation error is dominated by the difference between the reply times of the two interrogating modules:

$$\delta_{\text{error}} \propto (t_5 - t_4) - (t_3 - t_2) \tag{2.5}$$

Hence, as long as this difference is kept small in the design of the embedded system, the error introduced by the clock drift is negligible.

Lastly, further documentations also describes a relationship between the reported timestamp of a received signal and the corresponding signal level [7]. It is demonstrated, that the bias in distance estimation can vary from around -11 to 8 centimeters mainly due to this correlation. *Decawave* suggests an empirical lookup table where the range bias corresponding to a specific received signal level can be determined.

2.1.5 Problematic Observations from First Experiments

Further testing with the modules in the domain which are relevant for mobile robot navigation reveals some significant shortcomings in the perfor-

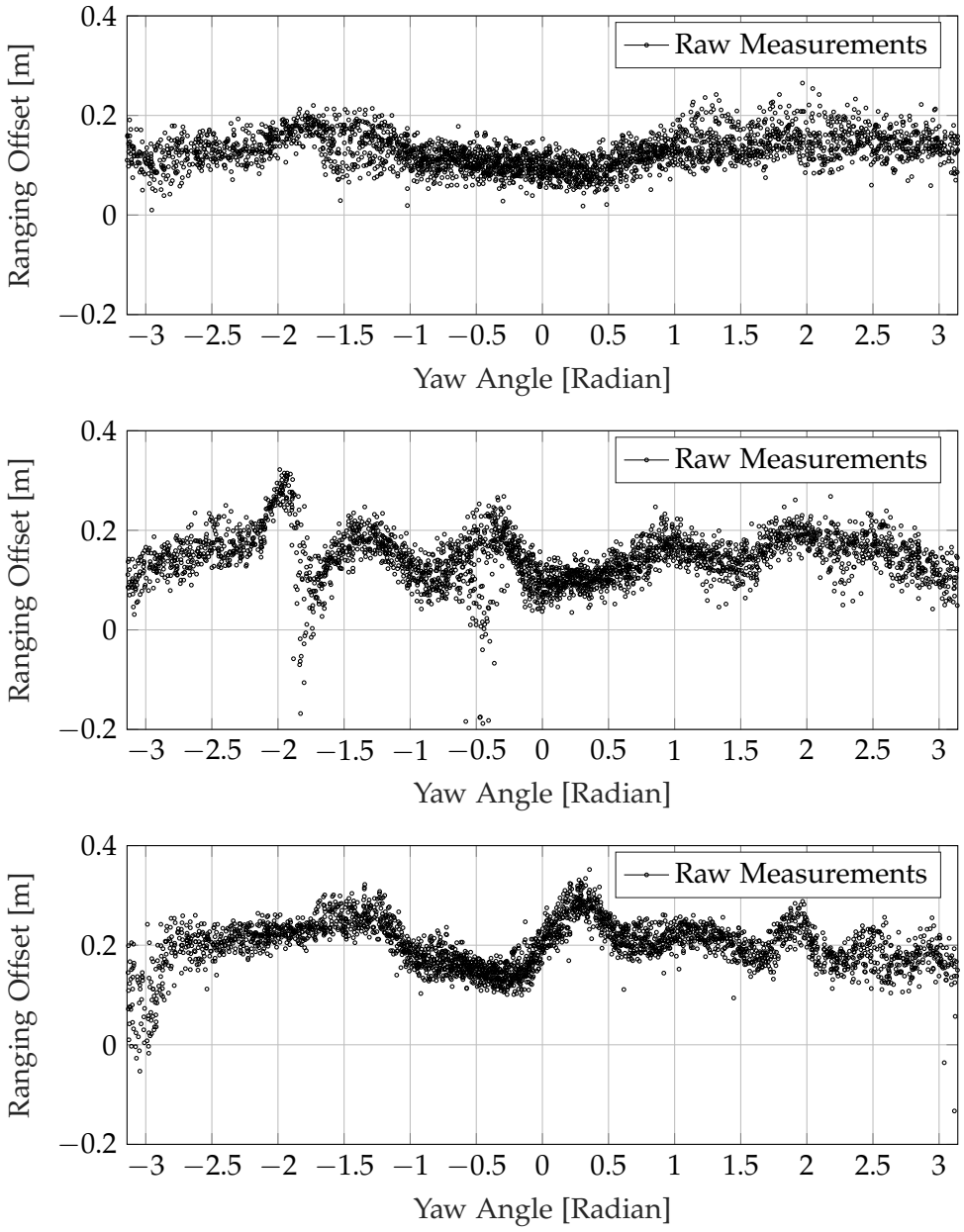

Figure 2.3: Ranging measurement data acquired from similar experiments as illustrated in figure 1.3. For each plot the orientation of the anchor is set differently (from above: $\alpha_a = 0°$, $\alpha_a = 40°$ and $\alpha_a = -40°$). Range data are then gathered by rotating the tag along its z-axis. As can be seen, the offset characteristics are different depending on the specific poses of the modules. Note that the results for the last two mentioned experimental setups are not exactly mirror-congruent despite the orientations for the anchors only differ in the sign.

mance of the *Loco Positioning Node*. The data were collected by connecting the tag to a computer by means of a micro-USB connection. For this project, the estimation error δ_{error} is calculated as the difference between the actual and the measured distance:

$$\delta_{\text{error}} = d_{\text{actual}} - d_{\text{measurement}} \tag{2.6}$$

The following behaviours can be noted:

- The highest possible data measurement frequency for 6 distance measurements, one from the interrogation with each anchor, is averaged at around 20Hz. It was maximized by reducing the processing time upon signal reception in the firmware for the tag as much as possible without breaking its state machine.
- At this rate, zero distance measurements can be retrieved occasionally.
- In the first testing setup, two modules were facing each other ($\alpha_a = \alpha_t = 0^\circ$ in the convention as defined in figure 1.3) and samples were collected while moving the tag away from the anchor along the connecting line. At distances within 1.5 to 5 meters, one can observe that the average offset at each measurement position is approximate constant 10 centimeters.
- However, for distances between 0.5 and 1.5 meters, the average offsets increase up to 20 centimeters for certain measurement positions.
- For even lower distances, measurements can not be retrieved.
- Furthermore, as previously showcased in figure 1.3, a correlation between the distance estimation error and the specific rotation angle of the tag module can be observed.
- Similar experiments where the anchors were differently oriented present non-congruent offset characteristics (see figure 2.3).
- Thereby, the immediate surroundings of the modules are not to be neglected. The above experiments were conducted with the tag mounted on top of the drone. The assembly was then placed on a rotating platform while maintaining line-of-sight condition with the anchor. However, by repeating the measurements with the same setup where the tag was mounted above a cardboard box instead, an increase of 10 centimeters for the offset means within 1.5 to 5 meters can be noticed.
- Most of the observed offsets exceed the range of the expected error due to received signal level as described in subsection 2.1.4.
- To establish a basic knowledge on the maximum range of the modules, a widescale test was conducted. Thereby, the tag was moved away

from the fixed anchor along their connecting line while facing each other ($\alpha_a = \alpha_t = 0°$). The experiment was performed outdoor on a racetrack such that the actual distance can be read from the ground markings. At maximal adjustable transmit power (33.5 dB), the furthest distance, where data are still received at the tag, is observed to be approximate 25 meters.

- Furthermore, it is noted that the data rate for a measurement with a single tag-anchor pair at the maximal distance is reduced down to around 2Hz.

The above observations give an idea about the limitation of these commercially available UWB technology. Although not explicitly measured for other types of orientations in an experimental setting, it is not far-fetched to assume that the offset correlates with the full pose of the interrogating tag as well as the responding anchor. Hence, as a summary, the specific mean offset from distance measurements between two modules seems to be dependent on the following conditions:

- the positions and orientations of the tag,
- and the anchor
- as well as the immediate surrounding area of both modules.

2.1.6 Potential Sources

This section attempts to explain the causes of the observations. It is hypothesized that the error is accumulated by the sources stated in 2.1.4. Moreover, the specific radiation pattern of the antennas [9], which defines the directional variation of the transmitted power, can influence the received signal strength at the other end. Ledergerber et al. [19] describe these effects with the following formulation:

$$h_{\mathrm{CIR},t} = \sum_{n=1}^{m} h_{\mathrm{tx},t}(\alpha_{\mathrm{tx},n}) * h_{\mathrm{env},t,n} * h_{\mathrm{rx},t}(\alpha_{\mathrm{rx}.n}) \tag{2.7}$$

Here, m states the number of multipath propagations and α_n the angle of departure or arrival of the corresponding signal. Hence, the channel impulse response is a convolution of the specific impulse responses from the transmitter, receiver and the environment respectively. Timestamping of an arriving signal is determined by the leading edge in the resulting function. Changing the orientation of one module directly influences the respective impulse response and consequently the position of the leading edge in the channel impulse response. This results in an angular dependency of the ranging offset. Furthermore, nearby objects with radio reflective surfaces,

such as the drone itself, can cause multi-path propagation, which give rise to the appearance of secondary signal paths close to the direct path. As a consequence, the leading-edge pulse can not be clearly detected in the channel impulse response, which would lead to inaccurate timestamping. The drop of data rate at the maximum distance can be explained by an increased loss of data packages in widescale applications. In contrast, no measurements can be retrieved for close distances because the state machines might have difficulty dealing with the corresponding required data processing speed.

2.1.7 Correction Approach

To ensure a high data rate for distance measurements with all 6 anchors, the usage of the UWB modules is limited to a small indoor space (see figure 3.1). Further, neglecting the ranging offsets would lead to inaccurate position estimations by means of multilateration. Hence, the offsets need to be captured beforehand. Despite all its dependencies, their occurrences appear to be systematic. Specifically, the means of the offsets from the interrogation with each anchor stay roughly equal under the same or similar conditions. Therefore, a large part of the work in the context of this thesis focuses on the modeling of the described ranging errors. Since they are influenced by a large number of factors, the derivations of empirical measurement models from first principles are complex and therefore excluded. In contrast, machine learning techniques such as Gaussian Process (see section 2.2), which are able to learn the variation of the offsets based on training data, are highly suitable for solving this task.

2.2 Gaussian Process

The state of a dynamic system can be recursively estimated by a Bayes filter. Probabilistic prediction and measurement model are key components of each filter describing the change of the process and the sensor measurements over time. However, deriving parametric models to describe highly nonlinear physical effects based on first principles can be complicated. In contrast, these models can be directly learned from training data using non-parametric machine learning techniques such as Gaussian Process. A complete coverage of this topic is beyond the scope of this report. The following subsections only provide the necessary equations for further discussions in the subsequent chapters. The interested reader is referred to the literature by Rasmussen and Williams [24] as well as the publication by Ko and Fox [15] for more information.

2.2.1 Recursive Bayesian Estimation

A Bayes filter keeps track of the probability distribution of the system state $\boldsymbol{x}_k$ conditioned on all past sensor observations $\boldsymbol{z}_{1:k}$. It consists of a prior update step based on the process model (equation 2.8) and a posterior update step based on the measurement model (equation 2.9):

$$p(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k-1}) = \int \underbrace{p(\boldsymbol{x}_k \mid \boldsymbol{x}_{k-1})}_{\text{process model}} \underbrace{p(\boldsymbol{x}_{k-1} \mid \boldsymbol{z}_{1:k-1})}_{\text{previous posterior}} d\boldsymbol{x}_{k-1} \tag{2.8}$$

$$p(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k}) \propto \underbrace{p(\boldsymbol{z}_k \mid \boldsymbol{x}_k)}_{\text{measurement model}} \underbrace{p(\boldsymbol{x}_k \mid \boldsymbol{z}_{1:k-1})}_{\text{prior}} \tag{2.9}$$

Here, $\boldsymbol{z}_{1:k-1}$ and $\boldsymbol{z}_{1:k}$ describe the sensor measurements obtained up to timestep $k-1$ and k respectively. The process model is a probabilistic model of the system dynamics. The measurement model describes the likelihood of making an observation z_k at the state x_k. Both models are usually characterized using a finite set of parameters (hence called parametric models). In contrast, Gaussian Process derives these models from training data which cannot be defined by such a finite set of parameters (hence called non-parametric models).

2.2.2 Regression

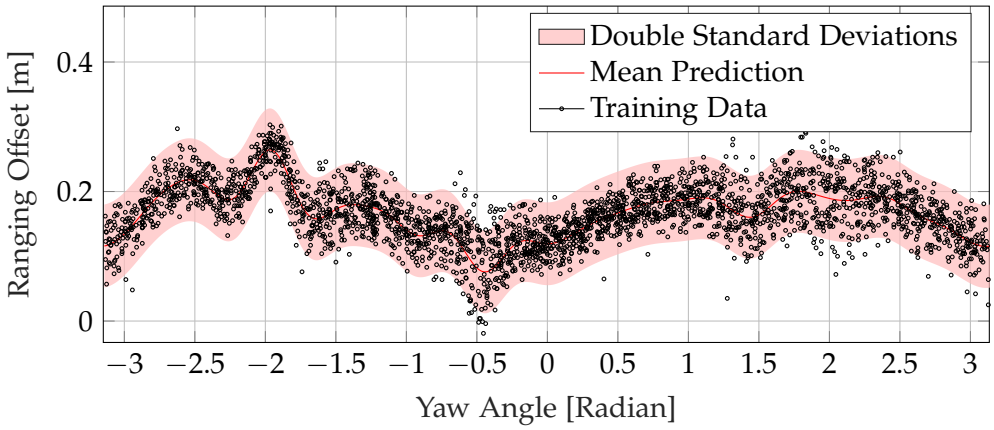

Figure 2.4: The correlation illustrated in figure 1.3 was captured by estimating the Gaussian Process mean (red) and variance (bright red) at each prediction angle. Thereby, the standard periodic kernel was used, which allows the function to repeat itself at both ends.

Gaussian Process is a method for learning the posterior distributions over functions from a finite set of training data (see figure 2.4). These data are assumed to be drawn from the following disturbed process:

$$y_i = f(\boldsymbol{x}_i) + \varepsilon \tag{2.10}$$

ε is an additive Gaussian noise with zero mean and variance σ_{n}^2. The training data is written as $\boldsymbol{D} = (\boldsymbol{X}, \boldsymbol{y})$, where $\boldsymbol{X} = \left[\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n\right]$ is a $d \times n$ matrix containing n d-dimensional inputs $\boldsymbol{x}_i$ and $\boldsymbol{y} = \left[y_1, y_2, ..., y_n\right]^\top$ a $n \times 1$ vector consisting of n scalar outputs y_i. Thereby, given a test input $\boldsymbol{x}_*$ and the training data, a Gaussian Process defines a normal distribution over the output y_* with the following mean (equation 2.11) and variance (equation 2.12):

$$\mu_{\mathrm{GP}}(\boldsymbol{x}_*, \boldsymbol{D}) = \mathbb{E}(f_* \mid \boldsymbol{x}_*, \boldsymbol{D}) = \boldsymbol{k}_*^\top \left[\boldsymbol{K} + \sigma_{\mathrm{n}}^2 \boldsymbol{I}\right]^{-1} \boldsymbol{y} \tag{2.11}$$

$$\sigma_{\mathrm{GP}}^2(\boldsymbol{x}_*, \boldsymbol{D}) = \mathrm{Var}(f_* \mid \boldsymbol{x}_*, \boldsymbol{D}) = k(\boldsymbol{x}_*, \boldsymbol{x}_*) - \boldsymbol{k}_*^\top \left[\boldsymbol{K} + \sigma_{\mathrm{n}}^2 \boldsymbol{I}\right]^{-1} \boldsymbol{k}_* + \sigma_{\mathrm{n}}^2 \tag{2.12}$$

Here, k is the kernel function which defines the degree of similarity between two input points. It is described by a set of parameters $\boldsymbol{\sigma}_{\mathrm{k}}$. $\boldsymbol{k}_*$ is a $n \times 1$ vector containing the kernel values between $\boldsymbol{x}_*$ and each input from the training set $\boldsymbol{X}$ and $\boldsymbol{K}$ is a $n \times n$ matrix containing the kernel values between the training inputs. More specifically, $k(\boldsymbol{x}_*, \boldsymbol{x}_i)$ defines the entries of $\boldsymbol{k}_*$, wheras $k(\boldsymbol{x}_i, \boldsymbol{x}_j)$ defines the value at the i-th row and j-th column of $\boldsymbol{K}$. As can be noted from the above expressions, the complete training set is explicitly required at each prediction step in order to estimate the marginal distribution, which explains its non-parametric characteristic.

2.2.3 Marginal Likelihood

The process noise and the kernel function are described by a set of hyperparameters $\boldsymbol{\sigma}_{\mathrm{h}} = \left[\sigma_{\mathrm{n}}, \boldsymbol{\sigma}_{\mathrm{k}}\right]^\top$. The optimal set can be determined by maximising the following log marginal likelihood:

$$\log p(\boldsymbol{y} \mid \boldsymbol{X}) = -\frac{1}{2} \boldsymbol{y}^\top \left[\boldsymbol{K} + \sigma_{\mathrm{n}}^2 \boldsymbol{I}\right]^{-1} \boldsymbol{y} - \frac{1}{2} \log \left|\boldsymbol{K} + \sigma_{\mathrm{n}}^2 \boldsymbol{I}\right| - \frac{n}{2} \log(2\pi) \tag{2.13}$$

The optimal noise and kernel smoothness parameters, which can be learned with gradient-based optimization methods, provide a prior notion of the underlying process function (equation 2.10).

2.3 Extended Kalman Filtering

The Kalman filter (KF) provides the exact solution to the Bayesian state estimation problem for a linear time-invariant system with Gaussian distributions for the process and measurement noise. In contrast, the extended Kalman filter (EKF) is a suboptimal extension of the KF to nonlinear systems [6]. It is derived by linearizing the nonlinear system equations about the latest state estimate. The following lists the recursive formulation of the discrete-time EKF with the assumption of additive process and measurement noise.

By initializing $\hat{\boldsymbol{s}}_0^+ = \boldsymbol{s}_0$, $\boldsymbol{P}_0^+ = \boldsymbol{P}_0$, the prior update is given as:

$$\hat{\boldsymbol{s}}_k = \boldsymbol{q}_{k-1}(\hat{\boldsymbol{s}}_{k-1}^+, \boldsymbol{u}_{k-1}, \boldsymbol{w}_{k-1} = \boldsymbol{0}) \tag{2.14}$$

$$\boldsymbol{P}_k = \boldsymbol{A}_{k-1}\boldsymbol{P}_{k-1}^+\boldsymbol{A}_{k-1}^\top + \boldsymbol{Q}_{k-1} \tag{2.15}$$

Furthermore, the equations for the posterior update is written as following, where $\hat{\boldsymbol{z}}_{\boldsymbol{k}} = \boldsymbol{h}_k(\hat{\boldsymbol{s}}_k, \boldsymbol{n}_k = \boldsymbol{0})$:

$$\boldsymbol{K}_k = \boldsymbol{P}_k\boldsymbol{H}_k^\top\left[\boldsymbol{H}_k\boldsymbol{P}_k\boldsymbol{H}_k^\top + \boldsymbol{R}_k\right]^{-1} \tag{2.16}$$

$$\hat{\boldsymbol{s}}_k^+ = \hat{\boldsymbol{s}}_k + \boldsymbol{K}_k\left[\tilde{\boldsymbol{z}}_{\boldsymbol{k}} - \hat{\boldsymbol{z}}_{\boldsymbol{k}}\right] \tag{2.17}$$

$$\boldsymbol{P}_k^+ = \left[\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H}_k\right]\boldsymbol{P}_k \tag{2.18}$$

$\hat{\boldsymbol{s}}_k$ and $\hat{\boldsymbol{s}}_k^+$ define the prior and posterior state estimates with $\boldsymbol{P}_k$ and $\boldsymbol{P}_k^+$ as the corresponding covariance matrices. $\boldsymbol{q}_{k-1}$ and $\boldsymbol{h}_k$ define the process and measurement equations respectively, $\boldsymbol{u}_{k-1}$ the system input and $\tilde{\boldsymbol{z}}_{\boldsymbol{k}}$ the measurement. The process $\boldsymbol{w}_{k-1}$ and measurement noise $\boldsymbol{n}_k$ are assumed to be Gaussian distributed with zero means and covariances $\boldsymbol{Q}_{k-1}$ and $\boldsymbol{R}_k$ respectively. The Jacobian matrices $\boldsymbol{A}_{k-1}$ and $\boldsymbol{H}_k$ of the model equations with respect to the state evaluated at $\hat{\boldsymbol{s}}_{k-1}^+$ and $\hat{\boldsymbol{s}}_k$ respectively linearize the system equations around the current state estimates.

Intuitively, the mean state estimate is predicted forward and the deviation between the actual and the predicted measurement is corrected using the undisturbed non-linear formulations. In contrast, the corresponding variances are calculated using the linearized forms.

2.4 Random Acceleration Model

The EKF explained in previous section 2.3 requires a suitable process model. There is a wide variety of possible kinematic models for the purpose of target tracking. For this project, a simple second-order random acceleration model [25] is chosen and the kinematic of the target is approximated by the motion of a point corresponding to its center of gravity. For the 1D-case, the velocity is constant except for an additional Gaussian noise term w_t:

$$\frac{\partial^2 r_t}{\partial t^2} = w_t \tag{2.19}$$

Where r_t denotes the object position, the first derivative its velocity v_t and the second derivative its acceleration a_t. Specifically, the noise term is assumed to be drawn from a zero-mean normal distribution with variance σ_w^2. With that, the first-order state equation is formulated as follows, where the dot notion for time differentiation is used:

$$\boldsymbol{s}_t = \begin{bmatrix} r_t \\ \dot{r}_t \end{bmatrix} = \begin{bmatrix} r_t \\ v_t \end{bmatrix} \quad \dot{\boldsymbol{s}}_t = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \boldsymbol{s}_t + \begin{bmatrix} 0 \\ 1 \end{bmatrix} w_t \tag{2.20}$$

A discretization of the continuous state equation 2.20 is required for further numerical computation. Thereby, a zero-order hold during sampling time[2] ΔT for the noise vector w_t is assumed. Specifically, $w_t = w_{k\Delta T}$ for $t \in \big[k\Delta T, k\Delta T + \Delta T\big)$. In the subsequent steps, the full 6-dimensional discrete-time state $\boldsymbol{s}_k = \big[\boldsymbol{r}_k, \boldsymbol{v}_k\big]^\top$ is introduced, where $\boldsymbol{r}_k$ denotes the 3-dimensional coordinate and $\boldsymbol{v}_k$ the corresponding velocity vector. Following an exact discretization approach [4], the discretized linear equation for the process $\boldsymbol{s}_k = \boldsymbol{q}_{k-1}(\boldsymbol{s}_{k-1}, \boldsymbol{w}_{k-1})$ is described as following:

$$\boldsymbol{s}_k = \underbrace{\begin{bmatrix} 1 & 0 & 0 & \Delta T & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta T & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta T \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\boldsymbol{A}_{k-1}} \boldsymbol{s}_{k-1} + \underbrace{\begin{bmatrix} \frac{1}{2}\Delta T^2 & 0 & 0 \\ 0 & \frac{1}{2}\Delta T^2 & 0 \\ 0 & 0 & \frac{1}{2}\Delta T^2 \\ \Delta T & 0 & 0 \\ 0 & \Delta T & 0 \\ 0 & 0 & \Delta T \end{bmatrix} \boldsymbol{w}_{k-1}}_{\breve{\boldsymbol{w}}_{k-1}} \tag{2.21}$$

The additional term is grouped in the adjusted noise vector $\breve{\boldsymbol{w}}_{k-1}$. Consequently, the new resulting noise covariance $\breve{\boldsymbol{Q}}_{k-1}$ is scaled by the sampling time:

[2] The time interval between successive samples of a continuous function.

$$\check{\mathbf{Q}}_{k-1} = \begin{bmatrix} \frac{1}{4}\Delta T^4 & 0 & 0 & \frac{1}{2}\Delta T^3 & 0 & 0 \\ 0 & \frac{1}{4}\Delta T^4 & 0 & 0 & \frac{1}{2}\Delta T^3 & 0 \\ 0 & 0 & \frac{1}{4}\Delta T^4 & 0 & 0 & \frac{1}{2}\Delta T^3 \\ \frac{1}{2}\Delta T^3 & 0 & 0 & \Delta T^2 & 0 & 0 \\ 0 & \frac{1}{2}\Delta T^3 & 0 & 0 & \Delta T^2 & 0 \\ 0 & 0 & \frac{1}{2}\Delta T^3 & 0 & 0 & \Delta T^2 \end{bmatrix} \sigma_{\mathrm{w}}^2 \tag{2.22}$$

Where it is assumed, that the noise vector variables $w_{i,k-1}$ are independent and identically distributed. Therefore, the random acceleration term is linked with the sampling time. Intuitively, this means that a small time window between samples limits the extent of possible state change. Within the context of recursive estimation, the sampling time corresponds to the time interval between successive EKF iterations[3].

[3]The elapsed time is measured by the stopwatch timer function in *Matlab*.
(Source: https://www.mathworks.com/help/matlab/ref/tic.html 30. November 2019)

Chapter 3

Methods

3.1 Anchor Self-Calibration Procedure

Standard UWB localization methods require the knowledge of the anchor positions in advance. The current position of an object can then be determined by a set of range measurements between the body-fixed mobile tag and each room-fixed anchor by means of multilateration. However, manual measurements of the anchor configuration is time-consuming and prone to inaccuracy, which therefore prevent large scale deployment of UWB-based positioning systems. Hence, similar to the method described by Pelka et al. [23], this section describes a simple optimization-based self-calibration algorithm which estimates the anchor locations within the flight setup. The proposed method was evaluated with ground-truth data from the *Vicon* system (see subsection 4.1) and is incorporated in further methods described in sections 3.2 and 3.3.

3.1.1 Anchor Setup and Unknown Coordinates

The 6 anchors are placed in a triangle formation such that line-of-sight of a tag with each anchor as well as line-of-sight between the anchors is largely given within the flying space. The anchor frame $\mathcal{A}$ is then constructed according to the conventions described in section 1.8. In order to lower the computational effort for the optimization problem, the number of unknown coordinate parameters are narrowed down by the following assumptions (compare with figure 3.1):

- the 6 anchor nodes are identically distributed on 3 wooden poles. Two anchors are mounted at the top and bottom of each pole, where the distance between their antennas is fixed and measured to be $d_{\mathrm{h}} = 2.2\mathrm{m}$,

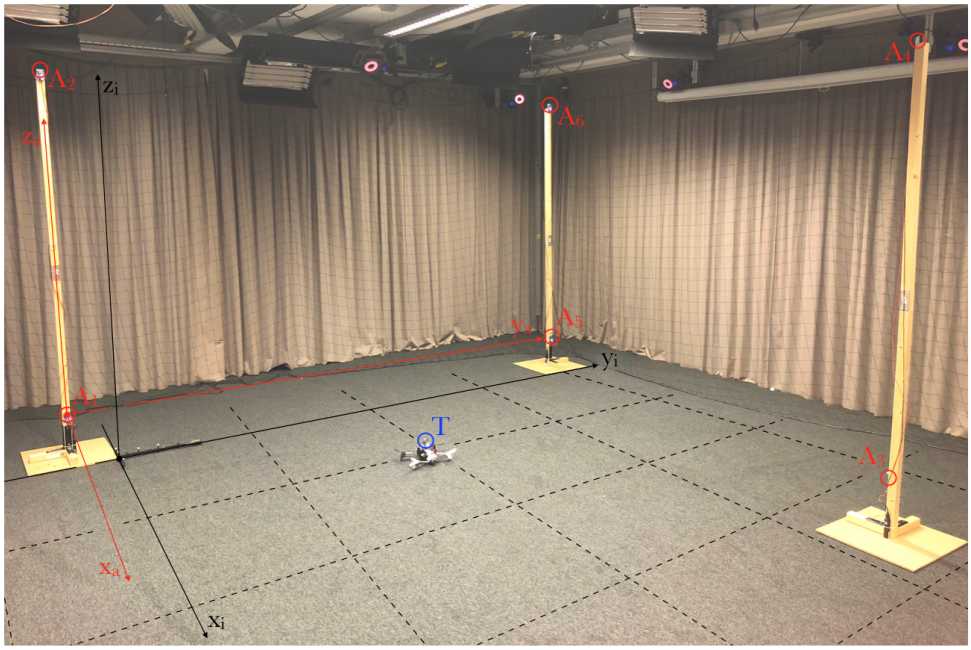

Figure 3.1: The flying space with its 6 placed UWB anchor nodes (A_1 to A_6, circled red), the drone-mounted tag can be seen in the middle (circled blue). As mentioned in section 1.8, the origin of the anchor frame $\mathcal{A}$ (marked red) is set at the antenna center of A_1. Using the *Vicon* wand, the origin of the inertial frame $\mathcal{I}$ (marked black) is placed in front of the anchor frame $\mathcal{A}$ with no relative rotation. The relative translation of the two frames is denoted in figure 1.8.

- antenna A_1 is set to be the origin ${}_{\mathcal{A}}\boldsymbol{a}_1 = {}_{\mathcal{A}}[0,0,0]^\top$ of the anchor coordinate system,
- without loss of generality, the line-of-sight of antennas A_1 and A_5 determines the y-axis, where the distance between the antennas is unknown. Hence, antenna A_5 has coordinates ${}_{\mathcal{A}}\boldsymbol{a}_5 = {}_{\mathcal{A}}[0, y_5, 0]^\top$,
- antenna A_3 is located in positive x- and y-direction and at the same height as antennas A_1 and A_5, such that ${}_{\mathcal{A}}\boldsymbol{a}_3 = {}_{\mathcal{A}}[x_3, y_3, 0]^\top$
- and since it is assumed that the poles are standing perfectly vertical, there is no horizontal displacement between the lower and the corresponding upper anchor. Therefore, all the top antennas share the same x- and y-coordinates as the bottom ones: ${}_{\mathcal{A}}\boldsymbol{a}_2 = {}_{\mathcal{A}}[0,0,d_\mathrm{h}]^\top$, ${}_{\mathcal{A}}\boldsymbol{a}_4 = {}_{\mathcal{A}}[x_3, y_3, d_\mathrm{h}]^\top$ and ${}_{\mathcal{A}}\boldsymbol{a}_6 = {}_{\mathcal{A}}[0, y_5, d_\mathrm{h}]^\top$.

Thus, only 3 unknown anchor frame coordinates $\boldsymbol{\sigma}_\mathrm{c} = [x_3, y_3, y_5]^\top$ need to be determined.

3.1.2 Data Post-Processing

By modifying the UWB firmware according to appendix A.2, inter-anchor distance measurements can be collected over radio. As explained in subsection 2.1.5, zero measurements can be retrieved occasionally. These and other outliers are removed beforehand. In this context, an outlier is classified as a value with more than three scaled median absolute deviations[1] away from the median [20]:

$$\begin{aligned} \sigma_{\mathrm{m}} &= \mathrm{median}(\left|\boldsymbol{d}_{uv} - \breve{d}_{uv}\right|) \\ \breve{d}_{uv} - 3\,\sigma_{\mathrm{m}} &\leq d_{i,uv} \leq \breve{d}_{uv} + 3\,\sigma_{\mathrm{m}} \end{aligned} \tag{3.1}$$

Here, $\breve{d}_{uv}$ is the median of the dataset $\boldsymbol{d}_{uv}$ consisting of the n distance measurements $d_{i,uv}$ collected between anchors u and v. This is implemented in the *Matlab* built-in function called `rmoutliers()`[2].

As illustrated in figure 2.3, a variance in the data can be observed at fixed poses. Hence, in order to get an estimate $\hat{d}_{uv}$ of the underlying distance, multiple range data are collected and simply averaged per anchor pair in order to reduce the effect of noise. Here, $\hat{d}_{uv}$ represents the distance estimation between anchors u and v, where anchor u is switched in tag mode for the interrogation. Furthermore, In order to guarantee for symmetry, the mean between $\hat{d}_{uv}$ and $\hat{d}_{vu}$ is taken for further processing.

3.1.3 Gauss-Newton Formulation

The theory behind Gauss-Newton optimization is explained in appendix A.3. Given the processed intermediate anchor distances from section 3.1.2, the multilateration problem can be solved by reformulating the anchor self-localization task as a non-linear least squares problem with the following residuals:

$$r_{uv} = \|{}_{\mathcal{A}}\boldsymbol{a}_u - {}_{\mathcal{A}}\boldsymbol{a}_v\|_2 - \hat{d}_{uv} \tag{3.2}$$

Here, ${}_{\mathcal{A}}\boldsymbol{a}_i$ represents the position of the i-th anchor in the anchor frame and $\hat{d}_{uv}$ the mean distance estimation between anchor u and v. By following the iterative formulation, a solution to the unknown anchor setup can be found which minimizes the sum of squared residuals. Specifically, the algorithm

[1] Compared with the standard deviation, median absolute deviation is more robust against outliers. With standard deviation, the distances from the mean are squared so that outliers are weighted more heavily. In contrast with median absolute deviation, the disparities of a small group of outliers are insignificant.

[2] Source: https://mathworks.com/help/matlab/ref/rmoutliers.html
19. December 2019

determines $\hat{\sigma}_c$ in such a way that the differences between the observed and the computed distances for all anchor pairs are minimized. Note that not all residuals is a function of the unknown coordinates. Nonetheless, for the sake of implementation simplicity, all possible anchor pairs are included in the overall sum. The irrelevant residuals do not influence the result of the optimization as they only provide added constants to the objective function. Furthermore, the residuals are counted twice due to their symmetry characteristics. Specifically, r_{uv} is the same as r_{vu} after data processing. Since this only provides a constant scale, it does not affect the solution neither.

Lastly, the homogeneous transformation (see equation A.3) is applied in order to transform the anchor positions from the anchor to the inertial frame.

3.2 Baseline Framework

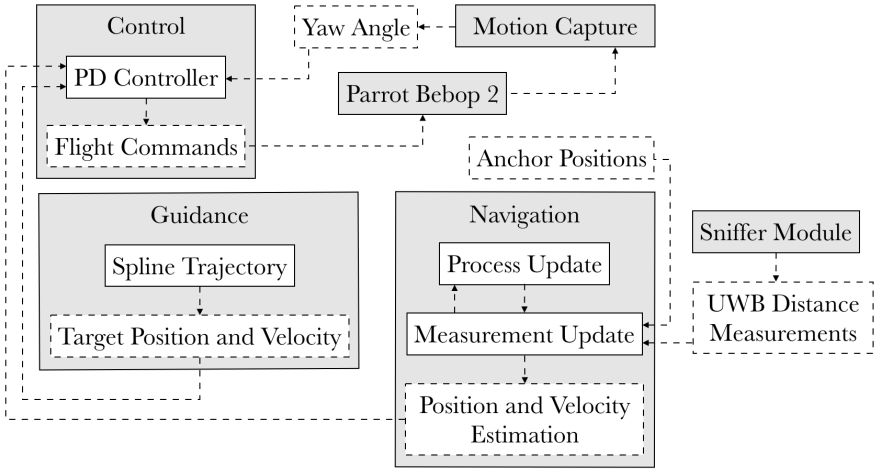

Figure 3.2: Flowchart of the interacting modules for the baseline guidance, navigation and control framework. Here, the model for the range measurements is uncorrected.

Following the approach by Guo et al. [11], this section presents the general guidance, navigation and control framework for this project as a baseline method. A graphical illustration of the workflow can be found in figure 3.2 which helps understanding the explained concepts. While the guidance and navigation parts are calculating the corresponding reference and current pose respectively, the control unit uses these to produce the appropriate flight commands for the UAV. Thereby, the navigation step is based on filtered multilateration and makes use of the anchor positions determined by the self-calibration procedure (see section 3.1). Here, the distance data from the UWB modules are directly used for position estimations without correcting for the biases in the range measurement model. Section 3.3 presents the

modifications to this framework to accommodate for range correction. The functionality of the drone as well as the motion capture system are previously described in subsections 1.7.2 and 1.7.4. The presented method was demonstrated with UAV test flights and evaluated with ground-truth data from the *Vicon* system (see section 4.2).

3.2.1 Navigation

To obtain a state from the measured ranges, the data are filtered and processed through an EKF. Compared with simple multilateration in terms of Gauss-Newton optimization as described in subsection 3.1.3, the EKF leverages the estimation accuracy by including the prior knowledge of the target kinematics as described in section 2.4. The recursive estimation is started after the take-off command is given by the user and the drone reaches its hovering state. The following subsections describe the measurement model for the posterior update.

Measurement Update

The discrete measurement model describes the expected sensor measurements. In this context, the measurements are related to the range readings between the drone-fixed tag and the environment-fixed anchors. Therefore, the noise-augmented euclidean norm models the distance reading. The entries of the vector $\boldsymbol{z}_k = \boldsymbol{h}_k(\boldsymbol{s}_k, \boldsymbol{n}_k)$ are written as:

$$z_{i,k} = \|\boldsymbol{r}_k - \boldsymbol{a}_i\|_2 + n_{i,k} \tag{3.3}$$

Where i specify the corresponding anchor index as stated in figure 3.1. Thus, all 6 measurements are used for the posterior update step. The anchor positions are estimated by the self-localization method, whereby $\boldsymbol{a}_i$ denotes the position of the i-th anchor. Furthermore, the fluctuations in the readings are modeled by an additional Gaussian noise term $\boldsymbol{n}_k$. Assuming the variables of the noise vector to be independent and identical distributed, the complete covariance is written as $\boldsymbol{R}_k = \sigma_{\mathrm{n}}^2 \boldsymbol{I}_{6\times 6}$, where σ_{n}^2 describes the variance of the measurements per anchor.

Since the measurement function is non-linear, a linearization around the current state is required. The rows of the Jacobian matrix $\boldsymbol{H}_k = \frac{\partial \boldsymbol{h}_k}{\partial \boldsymbol{s}_k}|_{\hat{\boldsymbol{s}}_k}$ evaluated at the prior state estimate are written as following:

$$\boldsymbol{H}_{i,k} = \begin{bmatrix} \frac{\hat{r}_{\mathrm{x},k} - a_{\mathrm{x},i}}{\|\hat{\boldsymbol{r}}_k - \boldsymbol{a}_i\|_2} & \frac{\hat{r}_{\mathrm{y},k} - a_{\mathrm{y},i}}{\|\hat{\boldsymbol{r}}_k - \boldsymbol{a}_i\|_2} & \frac{\hat{r}_{z,k} - a_{z,i}}{\|\hat{\boldsymbol{r}}_k - \boldsymbol{a}_i\|_2} & \boldsymbol{0}_{1\times 3} \end{bmatrix} \tag{3.4}$$

Measurement Outliers Rejection

The EKF is not robust against outliers in the measurements, which would lead to erroneous estimations. To avoid injecting distance readings which largely deviate form their expectations, these outliers are removed before commencing the full measurement update step. By using the Mahalanobis distance [5], outliers from the anchor measurements can be detected and removed by identifying their variance-weighted errors:

$$m_{i,k}^2 = \frac{\left(\tilde{z}_{i,k} - \hat{z}_{i,k}\right)^2}{\boldsymbol{H}_{i,k}\boldsymbol{P}_k\boldsymbol{H}_{i,k}^\top + \sigma_\mathrm{n}^2} \tag{3.5}$$

The larger the generalized distance $m_{i,k}$, the more likely the measurement $\tilde{z}_{i,k}$ is an outlier, where an empirical threshold is set for the classification. If the measurement with anchor i is detected as an outlier, the corresponding i-th row of $\boldsymbol{H}_k$, i-th row and column of $\boldsymbol{R}_k$ as well as the i-th element of $\tilde{\boldsymbol{z}}_k$ and $\hat{\boldsymbol{z}}_k$ are removed for the current posterior update step.

3.2.2 Guidance

This module is responsible for generating the reference trajectory for the flight. With the corresponding position controller described in section 3.2.3, trajectory control can be achieved by alternating the current reference pose at a specific frequency such that the drone is following a time-variant goal position. In order to compute the control command for the drone, the position controller requires a reference position $\check{\boldsymbol{r}}_t$ and velocity $\check{\boldsymbol{v}}_t$ as well as a reference yaw angle $\check{\psi}_t$, where t denotes the current continuous time[3]. For the experiments, two trajectory patterns are implemented which are described in the following.

Circular Trajectory

A circular trajectory at constant height h_z is generated for the first flight pattern. The time-variant positions are defined as:

$$\check{\boldsymbol{r}}_t = \begin{bmatrix} m_\mathrm{x} + r\sin(2\pi f t) \\ m_\mathrm{y} - r\cos(2\pi f t) \\ h_\mathrm{z} \end{bmatrix} \tag{3.6}$$

Where $\boldsymbol{m} = \left[m_\mathrm{x}, m_\mathrm{y}\right]^\top$ and r specify the center coordinate on the xy-plane and the radius of the circle respectively. f defines the frequency at which

[3]Similar to section 2.4, the current time is calculated using the stopwatch timer function in the implementation.

the reference state is changed. The corresponding velocities are tangential to the circle with the following directions, where the absolute velocity is linked with the frequency:

$$\check{\boldsymbol{v}}_t = \begin{bmatrix} 2r\pi f \cos(2\pi f t) \\ 2r\pi f \sin(2\pi f t) \\ 0 \end{bmatrix} \tag{3.7}$$

The reference yaw angle $\bar{\psi}_t$ is defined as following in order to keep it within the interval $\left[0, 2\pi\right)$:

$$\check{\psi}_t = 2\pi f t - 2\pi \lfloor f t \rfloor \tag{3.8}$$

Hence, the drone is moving along a circular path with a specified velocity where its heading is pointing towards the flying direction. For the variant, where the front-camera is pointing towards the center of the circular trajectory, the reference yaw angle is shifted as following:

$$\check{\psi}_t = 2\pi f t - 2\pi \lfloor f t \rfloor + \frac{1}{2}\pi \tag{3.9}$$

Spline Trajectory

More complicated trajectories can be defined with cubic spline polynomials. As presented in section A.6, they are constructed by piecewise third-order polynomials which pass through a finite number of checkpoints in the flying space in a specified order, hence resulting in a smooth overall trajectory. For simplicity, the spline trajectory is kept at a constant height h_z and therefore the z-axis coordinates of the waypoints are set to be equal.

However, for the method to be implemented as described, at least one set of axis-coordinates of the waypoints needs to be monotonically increasing in the defined order. Hence, for generalization, n_d monotonically increasing auxiliary coordinates $\boldsymbol{u}_d$ are introduced such that two separate splines can be interpolated in the new ux- and uy-space respectively. This is implemented using the *Matlab* built-in function `csape()`[4]. By combining the x- and y-values of the corresponding query points at the locations defined by $\boldsymbol{u}_q$, the coordinates of the desired spline trajectory can be obtained. With n_q query point coordinates $\boldsymbol{x}_q$ and $\boldsymbol{y}_q$, the time-variant positions are defined as:

$$\check{\boldsymbol{r}}_t = \check{\boldsymbol{r}}_i = \begin{bmatrix} x_{i,q} & y_{i,q} & h_z \end{bmatrix}^\top \tag{3.10}$$

[4]Source: https://www.mathworks.com/help/curvefit/csape.html 10. December 2019

Here, $i = \left\lceil \frac{n_q t}{t_f} \right\rceil$ defines the specific query point. Furthermore, t_f specify the overall time for the drone to complete the full spline trajectory. In case of $i > n_q$, the drone has reached the last query point and is commanded to land. The velocities are approximate tangential to the spline and their absolute values are assumed to be constant[5] at $v_a = \frac{l_s}{t_f}$, where l_s is an estimate of the complete spline length. It is approximated by the sum of the euclidean distances between consecutive query points. Thus, the spatial-dependent velocity vectors are calculated as following:

$$\check{\boldsymbol{v}}_t = \check{\boldsymbol{v}}_i = \begin{bmatrix} \frac{\Delta x_{i,q}}{\|\check{\boldsymbol{r}}_{i+1,q} - \check{\boldsymbol{r}}_{i,q}\|_2} v_a & \frac{\Delta y_{i,q}}{\|\check{\boldsymbol{r}}_{i+1,q} - \check{\boldsymbol{r}}_{i,q}\|_2} v_a & 0 \end{bmatrix}^\top \tag{3.11}$$

Whereby the numerators are defined as:

$$\begin{aligned} \Delta x_{i,q} &= x_{i+1,q} - x_{i,q} \\ \Delta y_{i,q} &= y_{i+1,q} - y_{i,q} \end{aligned} \tag{3.12}$$

Lastly, the reference yaw angle, which is normalized to $[0, 2\pi)$, is described with the atan2-function as follows:

$$\check{\psi}_t = \check{\psi}_i = \begin{cases} \text{atan2}(\Delta y_{i,q}, \Delta x_{i,q}) & \text{for atan2}(\Delta y_{i,q}, \Delta x_{i,q}) \geq 0 \\ 2\pi + \text{atan2}(\Delta y_{i,q}, \Delta x_{i,q}) & \text{for atan2}(\Delta y_{i,q}, \Delta x_{i,q}) < 0 \end{cases} \tag{3.13}$$

3.2.3 Control

A simple position and an attitude controller is implemented for autonomous mobility in the lab space. The posterior state estimate from the EKF provides a feedback for the position controller to assist the actuation of the quadcopter towards a desired reference position given by the guidance module. As mentioned in section 1.7.4, the *Vicon* system provides an accurate measure of the attitude, which is given in quaternion form. Those measurements are directly used to control the heading in order to avoid implementing an attitude estimator from the on-board IMU. This simplification is a reasonable choice since attitude estimation is beyond the focus of this project.

A PD-controller is used for regulating the position. The proportional part contributes an actuation scaled by the position error in the direction of the reference position. In contrast, the derivative part generally counteracts large overshoots by considering the derivative of the error. However, the implementation in this project differs insofar as the velocity difference is

[5] It is noted that this assumption is bad in case of sudden directional changes such as for narrow curves.

used instead of the derivative of the position error. The control message for translation is therefore written as:

$$\boldsymbol{u}_{\mathrm{t}} = k_{\mathrm{p}}\left[{}_{\mathcal{B}}\check{\boldsymbol{r}}_t - {}_{\mathcal{B}}\hat{\boldsymbol{r}}_k^+\right] + k_{\mathrm{d}}\left[{}_{\mathcal{B}}\check{\boldsymbol{v}}_t - {}_{\mathcal{B}}\hat{\boldsymbol{v}}_k^+\right] \tag{3.14}$$

Here, the 3-dimensional translational input for the drone is expressed in the body-fixed frame in order to match the action of the physical inputs as defined in subsection 1.7.2. The positions and velocities are given by the navigation and guidance modules. Using equations A.3, A.4 and A.5, the vectors are transformed to the body-fixed frame. k_{p} and k_{d} represent the gains of the controller.

Additionally, the heading is regulated by a P-controller. Therefore, the *Vicon* quaternion measurement of the rigid body is transformed to Euler angles by means of equation A.6. The corresponding yaw angle is then normalized to $[0, 2\pi)$ by applying the following mapping for negative angles $\hat{\psi}_k^{(-)}$:

$$\hat{\psi}_k = \hat{\psi}_k^{(-)} + 2\pi \tag{3.15}$$

Note that the resulting yaw angle measures the counter clockwise rotation around the body-fixed z-axis as illustrated in figure 1.8. With that, the control message for the heading is defined as:

$$u_{\mathrm{r}} = k_{\psi,\mathrm{p}}\left(\check{\psi}_t - \hat{\psi}_k\right) \tag{3.16}$$

This result in clockwise rotation for $u_{\mathrm{r}} < 0$ and counter clockwise rotation for $u_{\mathrm{r}} > 0$. However, for the case where the absolute value of the resulting angle $\left|u_{\mathrm{r}}\right|$ is greater than π, the drone does not rotate according to the shortest angular direction[6]. Hence, for this case, the following adaption is made in order to choose the correct direction:

$$u_{\mathrm{r}} = -k_{\psi,\mathrm{p}}\,\mathrm{sgn}(\check{\psi}_t - \hat{\psi}_k)\left(2\pi - \left|\check{\psi}_t - \hat{\psi}_k\right|\right) \tag{3.17}$$

3.3 Learning-based Framework

This chapter explains how Gaussian Process models describing the systematic ranging offsets of the individual distance estimations between the modules can be integrated into the EKF in order to improve the localization accuracy. Thereby, the baseline guidance, navigation and control framework

[6]This happens, for example, in the case where $\check{\psi}_t$ is immediately greater than zero while $\hat{\psi}_k$ is immediately smaller than 2π, hence resulting in an unfavorable clockwise rotation.

depicted in section 3.2 is augmented towards a learning-based structure, where the idea is to capture the accuracy of the *Vicon* system. It is to mention that the presented framework is independent of the aforementioned motion capture system. In principle, any accurate localization system can be used instead for generating the required training data.

As previously mentioned in section 1.5, the complete flowchart, illustrated in figure 1.4, consists of the data collection, offline learning and repeating phases. During the data collection phase, the drone is commanded to follow a predefined flight trajectory while using precise *Vicon* position measurements as feedback for the controller. Along this path, the range measurements with each anchor are continuously logged together with the corresponding *Vicon* ground-truth positions of the UAV. Given the collected data after this flight as well as the anchor positions determined by the self-localization method described in section 3.1, models describing the deviations of the range measurements from the expected distances given the current position of the drone can be derived during the learning phase. Specifically, stochastic ranging error models are developed, which describe the probability distributions of the spatial-dependent distance errors. The stated models are incorporated in the EKF measurement update of the navigation module to improve position estimation accuracy during the repeating phase flight, which only uses UWB ranges to estimate the controller feedback. The following sections elaborate the specific implementation keypoints in detail.

3.3.1 Data Collection Phase

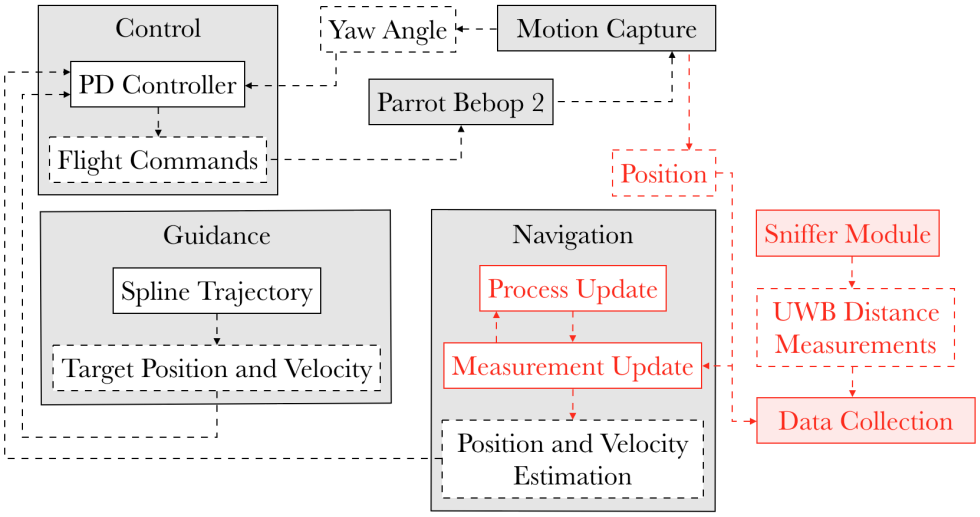

Figure 3.3: Flowchart of the interacting modules during the data collection phase.

Figure 3.3 shows the isolated flowchart for the data collection phase. A majority resembles the baseline framework presented in section 3.2, the differences are highlighted in red. The following subsections are explaining the major changes.

Vicon Measurement Model

The *Vicon* system directly reports the current position of the drone. Nonetheless, in order to get an estimate of the velocity for the controller, the position measurements are passed to the EKF pose estimator. The appertaining measurement model is written as:

$$\boldsymbol{z}_k = \boldsymbol{r}_k = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}}_{\boldsymbol{H}_k} \boldsymbol{s}_k \tag{3.18}$$

Note that the measurement model is linear in the state. Since the simplified process model for the target kinematics introduced in section 2.4 is a linear system as well, the EKF formulation is identical to its linear counterpart.

Data Collection

Matrices $\boldsymbol{V}$ and $\boldsymbol{X}$ are allocated for the collection of the distance measurements as well as the position data respectively. In each iteration, before estimating the prior state, all 6 anchors are interrogated and a *Vicon* measurement of the current position is taken. The corresponding distance measurements as well as the position data are saved columnwise in the aforementioned matrices respectively. Therefore, after n iterations, $\boldsymbol{X} = \begin{bmatrix} \boldsymbol{r}_1, ..., \boldsymbol{r}_n \end{bmatrix}$ is a $3 \times n$ and $\boldsymbol{V} = \begin{bmatrix} \boldsymbol{d}_1, ..., \boldsymbol{d}_6 \end{bmatrix}^\top$ a $6 \times n$ matrix.

3.3.2 Offline Learning Phase

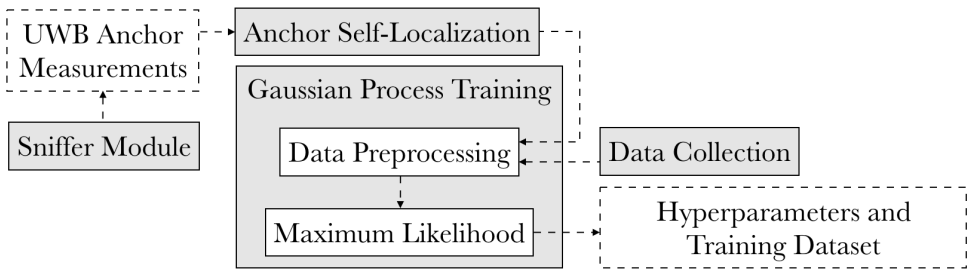

Figure 3.4: Flowchart of the interacting modules during the offline learning phase.

As the name suggests, the offline learning phase illustrated in figure 3.4 is executed between flights. With the collected ranging and position data from the previous phase as well as the given anchor position estimates from the self-localization approach, Gaussian Process models, as described in section 2.2, can be derived to predict the spatial-dependent correction terms for the individual distance measurements. The following subsections are explaining the synthesis of the required training data, the chosen kernel function as well as the optimization for the hyperparameters.

Data Preprocessing

Training data are required for Gaussian Process regression. The goal is to derive functions which maps the current position of the drone to the corresponding ranging offset distributions of the measurements with each anchor. Therefore, six datasets of the form $\boldsymbol{D}_i = \langle \boldsymbol{X}, \boldsymbol{y}_i \rangle$ are generated, where the matrix $\boldsymbol{X}$ corresponds to the gathered position matrix from subsection 3.3.1 and $\boldsymbol{y}_i$ to the deviations $\boldsymbol{\delta}_i$ of the range interrogations with anchor i from the expectation, which its k-th element is calculated as following:

$$\delta_{k,i} = \|\boldsymbol{r}_k - \boldsymbol{a}_i\|_2 - d_{k,i} \tag{3.19}$$

Here, $\boldsymbol{r}_k$ and $d_{k,i}$ denote the position of the drone and the range measurement with anchor i at discrete-time instance k respectively. $\boldsymbol{a}_i$ describes the position of the anchor. Note that the exact antenna position of the tag is not embedded in the data preparation. Instead the measured position of the drone's center of gravity is used. A further inaccuracy can also stem from the anchor position estimates. Therefore, it remains to be tested, if those added errors can be calibrated away with the presented learning-based approach.

Choice of Kernel Function

According to [17], the error in the range measurements can be modeled by the following kernel function:

$$k({}_{\mathcal{D}}\boldsymbol{u}, {}_{\mathcal{D}}\boldsymbol{u}_+) = \sigma_0 \exp\Bigg(\underbrace{-\frac{1 - \frac{{}_{\mathcal{D}}\boldsymbol{u}^\top {}_{\mathcal{D}}\boldsymbol{u}_+}{\|{}_{\mathcal{D}}\boldsymbol{u}\| \|{}_{\mathcal{D}}\boldsymbol{u}_+\|}}{\sigma_1}}_{\text{angle term}} - \underbrace{\frac{\left(\|{}_{\mathcal{D}}\boldsymbol{u}\| - \|{}_{\mathcal{D}}\boldsymbol{u}_+\|\right)^2}{\sigma_2}}_{\text{distance term}} \Bigg) \tag{3.20}$$

Here, the ${}_{\mathcal{B}}\boldsymbol{u}$ denotes the vector from the tag to the anchor antenna expressed in a specified antenna-fixed frame $\mathcal{D}$ of the tag. The first term within the exponent weighs the angle between two vectors, whereas the second term

correlates the difference in their euclidean lengths. Hence, two positions in the flying space are modeled to have similar ranging offsets if the corresponding tag antennas are equally apart from the anchor antenna and have comparable orientations. Assuming that the anchors are placed with random attitudes and their total number approaches infinity, it is shown that the aforementioned kernel is a good description for modeling the systematic offsets.

However, the above detailed formulation of the kernel requires a tracking of the current orientation of the body-fixed tag, which is not given in the developed framework of this project. Due to the learning-based approach in this project, where the desired flight trajectory is known beforehand, a simplification can be made. In the absence of cross sections[7] in the generated trajectories as well as a dynamic environment, the mapping from the positions to their corresponding range offsets can be assumed to be bijective. Therefore, a standard radial basis kernel function [3] is used in this context:

$$k(\boldsymbol{r}, \boldsymbol{r}_+) = \sigma_0 \exp\left(-\frac{1}{2} \frac{\|\boldsymbol{r} - \boldsymbol{r}_+\|^2}{\sigma_1} \right) \tag{3.21}$$

Therewith, two positions are correlated solely based on their euclidean distance. Hence, points which are close to each other share similar offsets.

Hyperparameter Optimization

Algorithm 1 Marginal likelihood using Cholesky factorization.
Code: `/matlab/gaussian_process/lib/functions/gp/getLogLikelihood.c`

1: **function** GETLOGLIKELIHOOD($\boldsymbol{X}, \boldsymbol{y}_i, \sigma_{i,0}, \sigma_{i,1}, \sigma_{i,\mathrm{n}}$)
2: $\quad \boldsymbol{K} \leftarrow$ CALCULATEKERNELMATRIX($\boldsymbol{X}, \sigma_{i,0}, \sigma_{i,1}$)
Calculating the Cholesky decomposition:
3: $\quad \boldsymbol{L} \leftarrow$ CALCULATECHOLESKY($\boldsymbol{K} + \sigma_{i,\mathrm{n}}^2 \boldsymbol{I}_{n \times n}$)
4: $\quad \boldsymbol{\alpha} \leftarrow \boldsymbol{L}^\top \backslash (\boldsymbol{L} \backslash \boldsymbol{y}_i)$ ▷ backslash operator in *Matlab*
Formulating the log marginal likelihood:
5: $\quad \log(\boldsymbol{y}_i \mid \boldsymbol{X}) \leftarrow -\frac{1}{2} \boldsymbol{y}_i^\top \boldsymbol{\alpha} - \sum_i \log(L_{ii}) - \frac{n}{2} \log(2\pi)$
6: $\quad$ **return** $\log(\boldsymbol{y}_i \mid \boldsymbol{X})$
7: **end function**

The hyperparameters for each model need to be learned. These include both kernel parameters $\sigma_{i,0}$ and $\sigma_{i,1}$ as well as the noise estimate[8] $\sigma_{i,\mathrm{n}}$ for the corresponding dataset. Given the kernel function as well as the dataset, the

[7] Specifically, the absence of reference states which differs only in their orientations.
[8] Despite the same nomeclature, do not confuse this with the variance in the measurement model as mentioned in subsection 3.2.1.

marginal log likelihood objective function as defined by equation 2.13 can be formulated. The practical implementation is shown in algorithm 1. According to [24], using the Cholesky decoposition instead of directly applying the matrix inversion results in better performance in terms of computational time and numerical stability. Cholesky decomposition is pre-implemented in *Matlab* under the function `chol()`[9].

The negative of the subsequent objective function can be minimized with a gradient descent optimization algorithm. In this thesis, the *Matlab* built-in function `fmincon()`[10] is used, which is an optimizer for constrained nonlinear multivariable functions. However, for larger dataset, the latter optimizer has its limitation. Instead, a *Python* implementation using the open-source `GPy`[11] Gaussian Process library which utilizes GPU acceleration can lower the computational time.

3.3.3 Repeating Phase

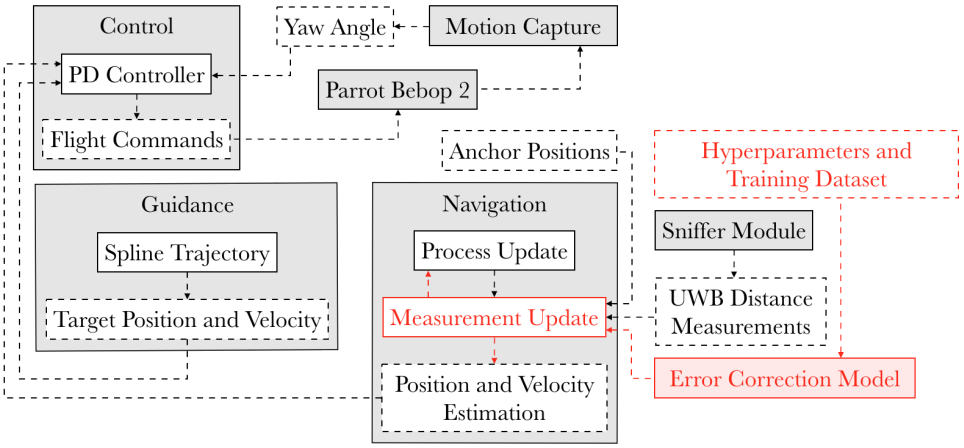

Figure 3.5: Flowchart of the interacting modules during the repeating phase.

The following subsections are explaining the implementation keypoints for the repeating phase, which are highlighted in red in figure 3.5.

Corrected Measurement Model

The Gaussian Process model provides an additional term to the basic EKF measurement equation as formulated in equation 3.3. Following an ap-

[9] Source: https://www.mathworks.com/help/matlab/ref/chol.html 27. November 2019

[10] Source: https://www.mathworks.com/help/optim/ug/fmincon.html 26. November 2019

[11] Source: https://www.github.com/SheffieldML/GPy 10. November 2019

proach described in [15], the modified measurement model is written as:

$$z_{i,k} = \|\boldsymbol{r}_k - \boldsymbol{a}_i\|_2 \underbrace{-\mu_{i,\mathrm{GP}}(\boldsymbol{r}_k, \boldsymbol{D}_i)}_{\text{correction term}} + n_{i,k} \tag{3.22}$$

Here, $z_{i,k}$ defines the model for the range measurements with anchor i. The correction term $\mu_{i,\mathrm{GP}}(\boldsymbol{r}_k, \boldsymbol{D}_i)$ defines the mean estimate of the corresponding Gaussian Process offset distribution at input position $\boldsymbol{r}_k$. Hence, the complete set describes the entries of the measurement vector $\boldsymbol{z}_k = \boldsymbol{h}_k(\boldsymbol{s}_k, \boldsymbol{n}_k)$. Furthermore, the noise $n_{i,k}$ is described by the variance of the aforementioned distribution. Assuming the individual readings to be independent of each other, the complete covariance can be written as a diagonal matrix with the entries $\sigma^2_{i,\mathrm{GP}}(\boldsymbol{r}_k, \boldsymbol{D}_i)$.

Modified EKF Linearization

The augmentation of the correction term in the measurement equation requires a new linearization as described in [15]. The Jacobian matrix $\boldsymbol{H}_k$ follows directly from the definition of equation 2.11. Specifically, the derivative of the Gaussian Process correction term with respect to the state and evaluated at the current position estimate for a single output dimension is written as following:

$$\frac{\partial \mu_{i,\mathrm{GP}}(\boldsymbol{r}_k, \boldsymbol{D}_i)}{\partial \boldsymbol{s}_k}|_{\hat{\boldsymbol{s}}_k} = \frac{\partial \boldsymbol{k}_*^\top}{\partial \boldsymbol{s}_k}|_{\hat{\boldsymbol{s}}_k} \left[\boldsymbol{K} + \sigma^2_{i,\mathrm{n}} \boldsymbol{I}\right]^{-1} \boldsymbol{y} \tag{3.23}$$

As can be noted, the vector $\boldsymbol{k}_*$, which defines the kernel values between the input position and the positions from the training data, is the only term dependent on the state. Furthermore, $\frac{\partial \boldsymbol{k}_*}{\partial \boldsymbol{s}_k}$ is the Jacobian matrix of the kernel vector with respect to the state. The j-th row of this matrix corresponding to the derivative of the kernel function with the j-th training input is written as:

$$\frac{\partial k(\boldsymbol{r}_k, \boldsymbol{r}_j)}{\partial \boldsymbol{s}_k} = \begin{bmatrix} \frac{\partial k(\boldsymbol{r}_k, \boldsymbol{r}_j)}{\partial r_{\mathrm{x},k}} & \frac{\partial k(\boldsymbol{r}_k, \boldsymbol{r}_j)}{\partial r_{\mathrm{y},k}} & \frac{\partial k(\boldsymbol{r}_k, \boldsymbol{r}_j)}{\partial r_{\mathrm{z},k}} & \boldsymbol{0}_{1\times 3} \end{bmatrix} \tag{3.24}$$

Since the kernel function does not depend on the velocity, the last 3 columns are zero. The derivative of the radial basis kernel function with respect to the x-coordinate is defined as following:

$$\frac{\partial k(\boldsymbol{r}_k, \boldsymbol{r}_j)}{\partial r_{\mathrm{x},k}} = -\sigma_{i,0} \exp\left(-\frac{1}{2} \frac{\|\boldsymbol{r}_k - \boldsymbol{r}_j\|^2}{\sigma_{i,1}} \right) \frac{r_{x,k} - r_{x,j}}{\sigma_{i,1}} \tag{3.25}$$

The derivatives with respect to the y- and z-coordinate respectively can be derived analogously. Therewith, equation 3.23 defines the 6-dimensional Jacobian vector for the interrogation with one anchor. Hence, the complete Jacobian matrix $\check{\boldsymbol{H}}_k$ of the correction vector is determined by concatenating all 6 Jacobian vectors horizontally. Lastly, the final Jacobian matrix $\boldsymbol{H}_k$ is calculated by subtracting $\check{\boldsymbol{H}}_k$ from the matrix defined by equation 3.4:

$$\boldsymbol{H}_{i,k} = \begin{bmatrix} \frac{\hat{r}_{\mathrm{x},k} - a_{\mathrm{x},i}}{\|\hat{\boldsymbol{r}}_k - \boldsymbol{a}_i\|_2} & \frac{\hat{r}_{\mathrm{y},k} - a_{\mathrm{y},i}}{\|\hat{\boldsymbol{r}}_k - \boldsymbol{a}_i\|_2} & \frac{\hat{r}_{\mathrm{z},k} - a_{\mathrm{z},i}}{\|\hat{\boldsymbol{r}}_k - \boldsymbol{a}_i\|_2} & \boldsymbol{0}_{1\times 3} \end{bmatrix} - \check{\boldsymbol{H}}_{i,k} \tag{3.26}$$

Chapter 4

Results and Discussion

Anchor self-calibration as well as multiple autonomous quadcopter flight experiments were conducted to evaluate the performance of the developed framework. All the results and their corresponding interpretations are presented in this chapter. *Matlab* was used for the implementation of the methods described in chapter 3. The tests were executed with a 2.5GHz *Intel i5-7300HQ* ground station computer. The required installations, source code and instructions are referred in appendix A.1. Furthermore, the firmware of the UWB modules was modified as described in appendix A.2 in order to retrieve range measurements between anchors as well as between a tag and multiple anchors over radio.

4.1 Anchor Self-Calibration Performance

The validation comprises of two parts. First, in order to grade the correctness of the algorithm, exact inter-anchor distances were simulated with pseudo-data. In a second step, the algorithm was tested with real measurements obtained from the setup. To evaluate the performance of the self-localization procedure, reflective markers were attached on each anchor for ground-truth position data. Specifically, they were attached next to each antenna with known offsets in order to not interfere with their radio propagation. Hence, a comparison can be made between the calculated and the ground-truth anchor positions in order to provide an understanding on the deviations. Flip ambiguities refer to multiple solutions for the anchor configuration given a set of inter-anchor distance measurements. In terms of optimization, the same minimum value of the objective function is occurring multiple times at different locations. The risk here is that gradient descent might converge to the incorrect solution. These wrong ambiguities were avoided by choosing the initialization of the unknown coordinates to be within the enclosed space of the anchors. Hence, it was set to $\sigma_{c,0} = \begin{bmatrix} 1, 1, 1 \end{bmatrix}^\top$ empirically.

4.1.1 Correctness Validation

For the simulation, the chosen anchor placements are listed in table 4.1. Therewith, the resulting 6×6 distance matrix $\hat{\boldsymbol{D}}$ was calculated, where $\hat{d}_{uv}$ is the value at u-th row and v-th column. With the given simulated distance data, multilateration was performed using Gauss-Newton optimization as described in subsection 3.1.3.

Anchor Index	1	2	3	4	5	6
	0	0	5	5	0	0
x-, y- and z-Coordinate	0	0	4	4	6	6
	0	2.2	0	2.2	0	2.2

Table 4.1: The arbitrary defined anchor frame coordinates of all anchors.

The results are illustrated in figure 4.1. The red circles indicate the ground-truth, whereas the crosses the calculated anchor positions. As expected, since the simulated data are not prone to systematic ranging biases, both symbols are congruent and there are no positioning errors. Hence, this demonstrates the correctness of the algorithm.

4.1.2 Validation with Real Range Measurements

Lastly, the same experiment was conducted with real ranging data from a similar anchor setup as shown in figure 3.1. As for data collection, 100 range measurements $\boldsymbol{d}_{u,v}$ were gathered for each anchor pair u and v. In contrast to the simulation, the calculated anchor coordinates were transformed to the inertial frame for comparisons with the *Vicon* measurements.

The new result are presented in figure 4.2. As previously described in subsection 2.1.5, depending on the poses of the interrogating anchors, different ranging biases can be observed. While the approach successfully determines the true anchor positions with exact distances under simulated conditions, which is a prove for its correct functionality, the performance in real environments is correlated with the quality of the distance estimations. As expected, the self-positioning errors rise with the increase of inaccuracy in the distance data. Nonetheless, given the real data, the algorithm is still able to determine the approximate and correctly signed anchor coordinate values. Note that a small fraction of the deviations from the ground-truth stems from the improper placement of the *Vicon* calibration wand. Specifically, it is difficult to exactly place the inertial frame according to the specifications as described in section 1.8. Hence, the calculated anchor points are mapped inaccurately. Furthermore, another error source stems from the measurement errors of the offsets between the individual *Vicon* markers and their corresponding

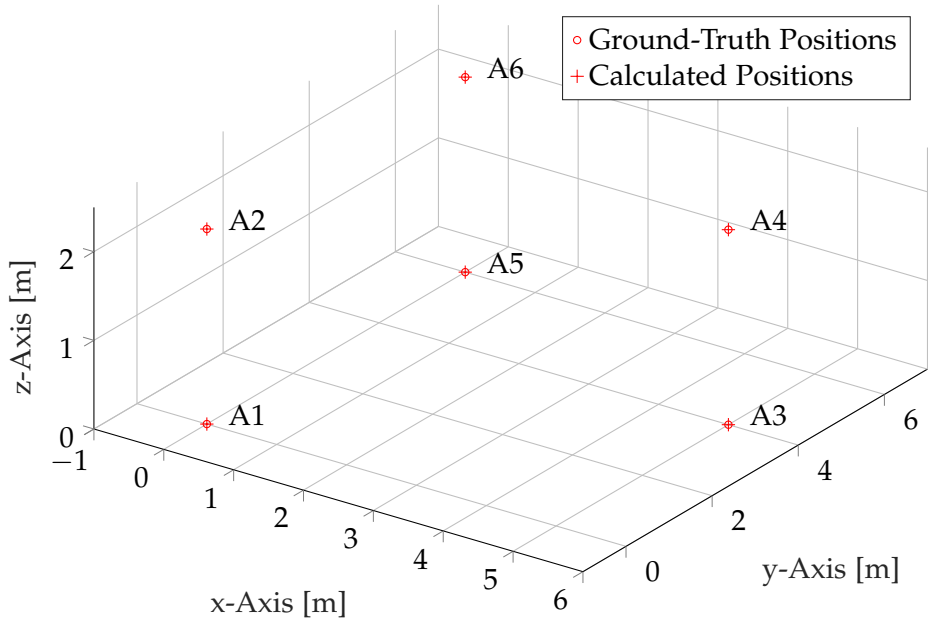

Figure 4.1: Simulation results of the Gauss-Newton multilateration algorithm plotted in the anchor frame.

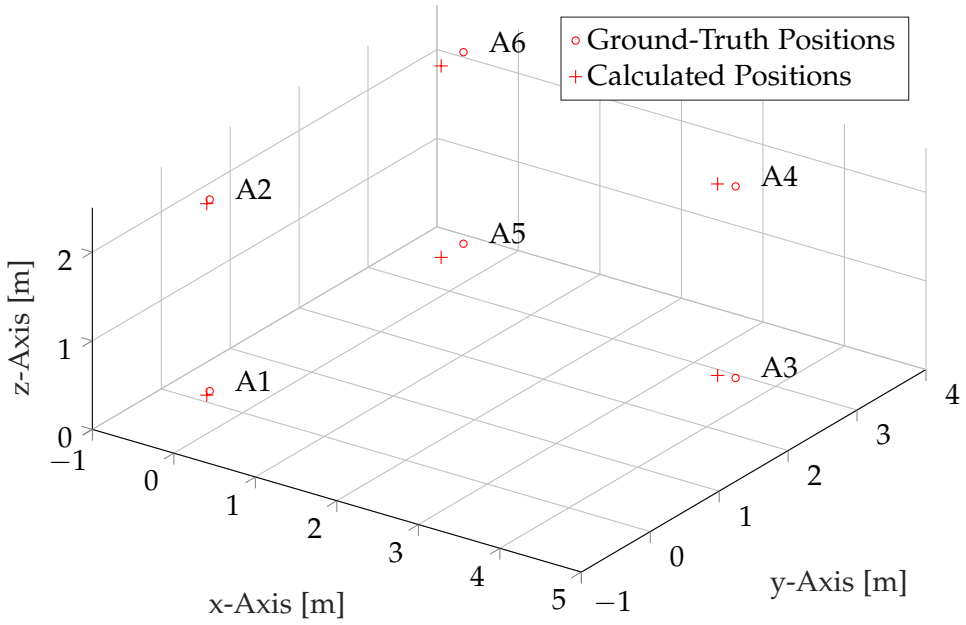

Figure 4.2: Results of the Gauss-Newton multilateration algorithm applied on real data plotted in the inertial frame.

anchor antennas. This is the reason why, for instance, the symbols for the origin A_1 are not coinciding.

The anchor self-calibration approach was embedded in the subsequent experiments in this chapter. It is clear, that the errors in the anchor positions will add another source of inaccuracy for filtered multilateration without error correction models (see section 4.2). The extent, to which this source can be calibrated away with the learning-based approach remains to be investigated (see section 4.3).

4.2 Performance with Uncorrected Measurement Model

The complete drone assembly described in subsection 1.7.3 as well as a similar anchor setup as the one presented in subsection 4.1.2 were used to evaluate the performance of the baseline framework from section 3.2. Empirically, the value for the covariance in the process model (see section 2.4) was set to $\sigma_w^2 = 2$. For the circular flights, the drone was commanded to fly a circular trajectory of radius $r = 1.25$, at height $h_z = 1$ and with midpoint $\boldsymbol{m} = \left[2, 1.5\right]^\top$. The circumnavigation frequency was stated to $f = \frac{1}{25}$. Thereby, the heading is pointing towards the flying direction in one case and towards the center of the circular trajectory in the other. As for the spline flight, by following the approach described in subsection 3.2.2, a S-shaped trajectory was constructed for the second flight pattern. The x-, y- and u-coordinates for the checkpoints were assessed according to table 4.2. $\boldsymbol{u}_q$ contains $n_q = 100$ query points uniformly distributed in the interval $\left[0, 5\right]$. The height was set to $h_z = 1$ and the complete path should be completed in $t_f = 40$. Therewith, the two-fold validation process is described in the following subsections.

Waypoint	1	2	3	4	5	6
	1.5	1.5	2.5	2.5	3.5	3.5
x-, y- and u-Coordinate	3	1	1	3	3	1
	0	1	2	3	4	5

Table 4.2: The checkpoints ($\boldsymbol{x}_d$ and $\boldsymbol{y}_d$) and their corresponding auxiliary coordinates $\boldsymbol{u}_d$ for the S-shaped spline trajectory.

4.2.1 Correctness Validation

As previously described in the preliminaries, the UWB distance measurements are prone to systematic errors. Therefore, in order to prove the correctness of the framework by itself, the first step was to replace the distance model with the *Vicon* model described in subsection 3.3.1. In principle, ac-

curate *Vicon* position measurements were directly used to calculate the state feedback in the navigation module. Empirically, the gains for the controller were set to the following values: $k_\mathrm{p} = 0.5$, $k_\mathrm{d} = 0.3$ and $k_{\psi,\mathrm{p}} = 2$. With an appropriate initialization for the state $\hat{\boldsymbol{s}}_0^+$ and its covariance $\boldsymbol{P}_0^+$, the process update provides a prior estimate for the position and velocity of the drone at the beginning of the recursive estimation. A study on the positioning performance of the *Vicon* system reports sub-millimeter mean absolute error on tracking static objects [21]. Due to the linearity of the prior as well as the posterior step and the high-accuracy of the motion capture system, the prior update initialization does not require much consideration. Hence, the following values were set for the initial state and its corresponding covariance: $\hat{\boldsymbol{s}}_0^+ = \begin{bmatrix}0,0,0,0,0,0\end{bmatrix}^\top$ and $\boldsymbol{P}_0^+ = \sigma_i^2 \boldsymbol{I}_{6\times 6}$ with $\sigma_i^2 = 10$. Furthermore, an empirical determined measurement noise covariance of $\boldsymbol{R}_k = \sigma_n^2 \boldsymbol{I}_{3\times 3}$ with $\sigma_n^2 = 0.001$ leaves a safe margin for possible fluctuations.

Figures 4.3 and 4.4 showcase the motion capture measurements of the drone and its orientations along the flying trajectory. Specifically, the motion of the drone's center of gravity is tracked. As can be seen, after small oscillations at the beginnings due to initial position offsets, the drone is following all the specified trajectories with accuracy, which demonstrates the correctness of the overall framework. Therefore, any deviation from this accuracy can be solely attributed to the error of the UWB range measurements.

4.2.2 Baseline Framework Validation

The same flight experiments were conducted with the complete framework described in section 3.2. The developed anchor self-calibration method from section 3.1 was used at the beginning in order to determine the anchor configurations. Since the update rate of the UWB measurements is much lower compared to the readings of the motion capture system, the command rate for the drone is consequently lower as well. In order to accommodate for this, all the gains for position and attitude control as stated in subsection 4.2.1 were halved. Since the measurement model is non-linear (see subsection 3.2.1), a wrong initialization for the state and its covariance can lead to instability of the filter owing to its linearization. Therefore, an educated guess based on Gauss-Newton multilateration with all anchors similar to subsection 3.1.3 was used for the initial position estimate. Because there are no significant horizontal motions during the hovering phase immediately after take-off, the initial velocity vector was set to zero. Furthermore, the initial covariance was empirically stated to $\boldsymbol{P}_0 = \sigma_\mathrm{i}^2 \boldsymbol{I}_{6\times 6}$ with $\sigma_\mathrm{i}^2 = 0.1$. Lastly, the noise covariance for the measurements with each anchor was set to $\sigma_\mathrm{n}^2 = 0.05$ experimentally.

The extent of using an uncorrected measurement model for position estimation is observable in figures 4.5 and 4.6. In comparison with the results

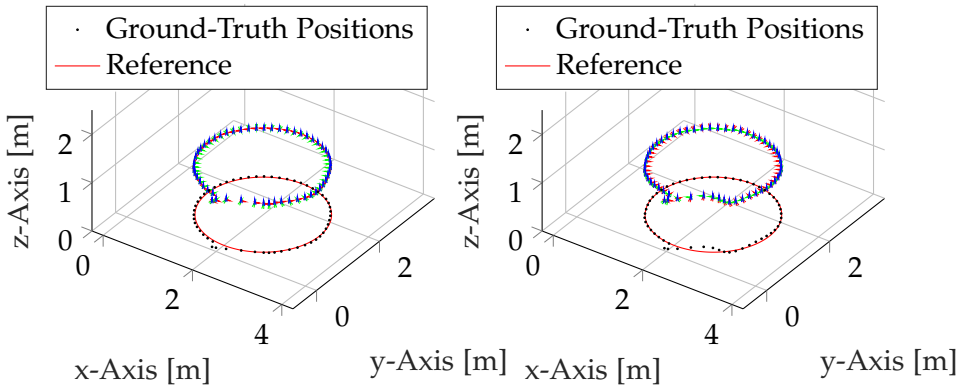

Figure 4.3: Ground-truth measurements of the circular trajectory flights using accurate *Vicon*-feedback control. The body-fixed frame over time is illustrated. The first plot on the left showcase the flight, where the heading is pointing towards the flying direction. For the second plot on the right, the heading is pointing towards the center of the circular trajectory. The ground-truth measurements are projected on the xy-plane (black) and compared with the projected reference trajectory (red).

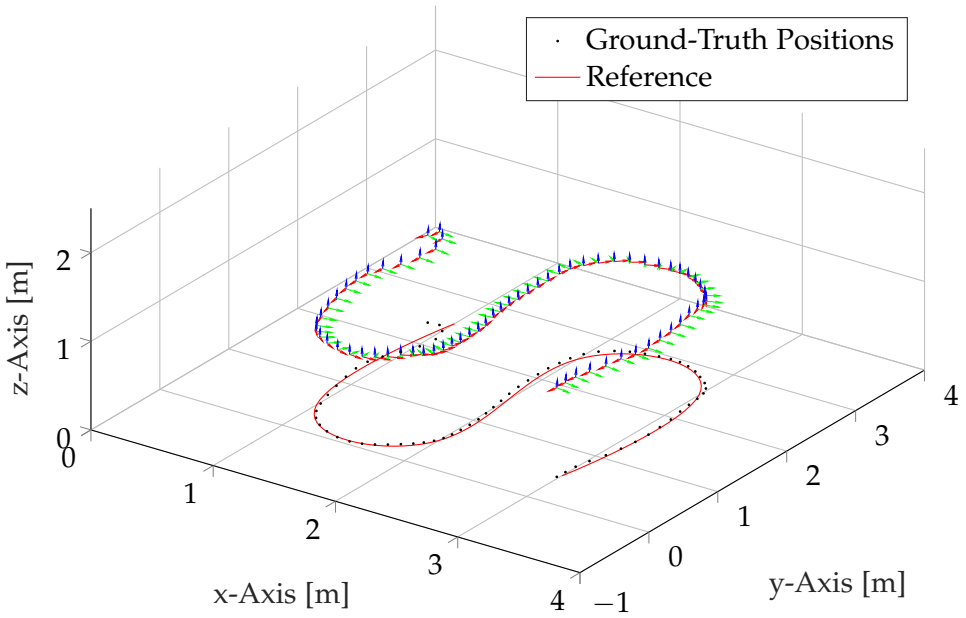

Figure 4.4: Ground-truth measurements of the spline trajectory flight using accurate *Vicon*-feedback control. The body-fixed frame over time is illustrated. Here, the heading is pointing towards the flying direction. The ground-truth measurements are projected on the xy-plane (black) and compared with the projected reference trajectory (red).

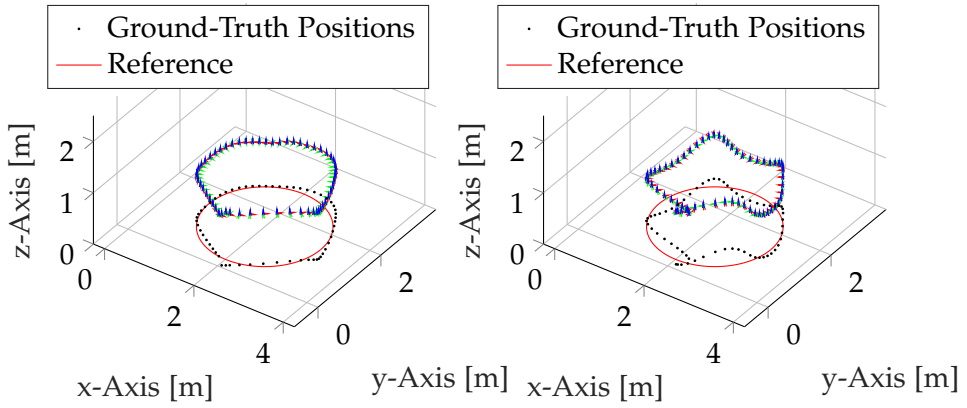

Figure 4.5: Ground-truth measurements of the circular trajectory flights where the corresponding control-feedback were calculated with erroneous UWB distance data and the uncorrected measurement model. As in figure 4.3, the body-fixed frame over time for both heading types as well as the corresponding projected ground-truth measurements (black) and reference trajectory (red) are illustrated.

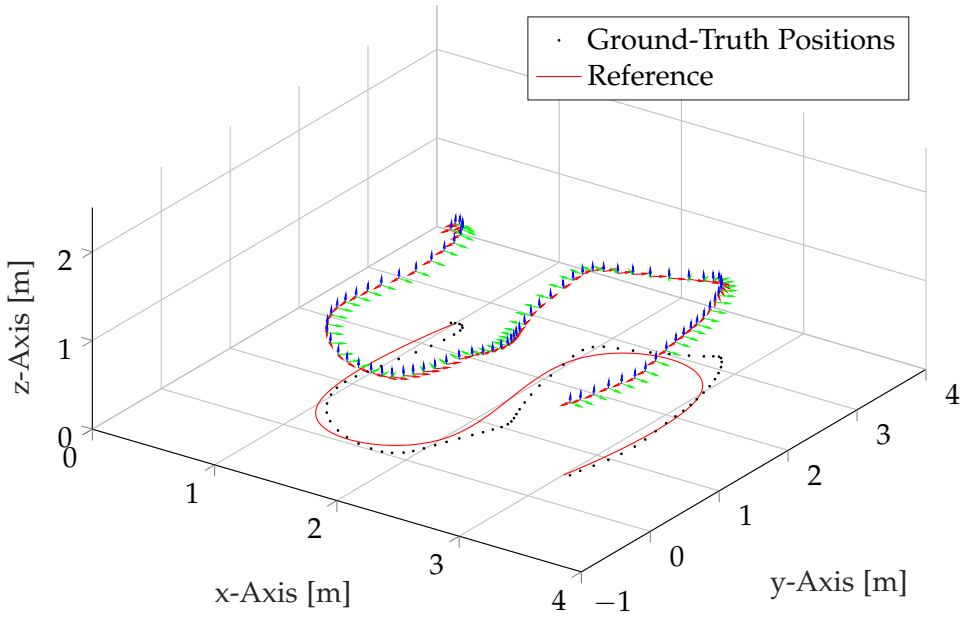

Figure 4.6: Ground-truth measurements of the spline trajectory flight where the control-feedback were calculated with erroneous UWB distance data and the uncorrected measurement model. As in figure 4.4, the body-fixed frame over time as well as the corresponding projected ground-truth measurements (black) and reference trajectory (red) are illustrated.

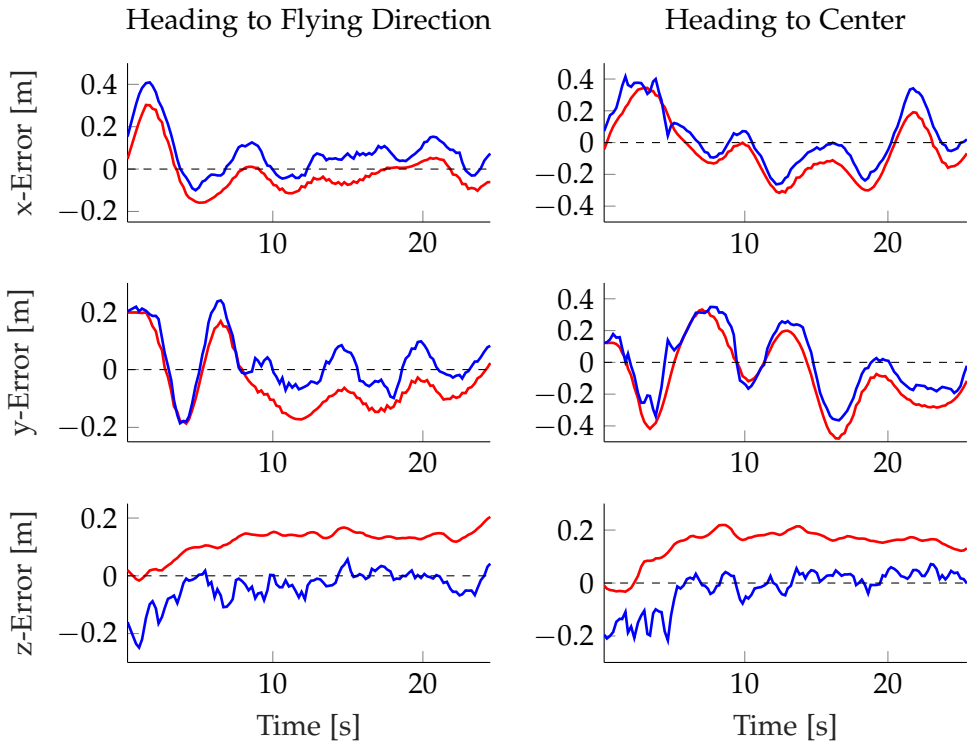

Figure 4.7: Error-time graphs for the circular trajectory flights. The chronological sequences of the offsets in each dimension are illustrated. Thereby, they were calculated as the differences from the reference. Here, the red lines describe the deviations of the ground-truth measurements of the drone's center of gravity, while the blue lines indicate the errors of the position estimations.

from subsection 4.2.1, the accuracy of the trajectory following decreases. Furthermore, from the error graphs for the circular flights (see figure 4.7), non-constant disparities between the ground-truth measurements from the motion capture system and the EKF position estimates can be noticed. It can also be observed that the drone generally flies at a lower altitude. A decrease in localization accuracy is expected and stems from the systematic errors of the UWB measurements as described in subsection 2.1.5 as well as from the lack of correction in the associated filter equations, on the one hand, and the already inaccurate anchor position estimates, on the other. Note that the baseline filtered multilateration algorithm attempts to track the position of the tag instead of the drone's center of gravity, which explains a part of the disparity from the ground-truth measurements. Also, since the tag is mounted above the quadcopter, the controller only tries bring the corresponding antenna up to the desired height, which reasons the flight of the drone at a lower level compared to subsection 4.2.1.

4.3 Performance with Corrected Measurement Model

The same flight experiments as described in subsection 4.2.2 were conducted with the developed learning-based framework. The parameters for the trajectories and as well as for the state estimation and controller were left the same. The only exception is for the data collection phase, where the drone was commanded to fly slower in order to guarantee for capturing enough range data as well as their local fluctuations. In that context, the circumnavigation frequency was set to $f = \frac{1}{60}$ during the data collection phase. As for the spline trajectory, the defined path should be completed in $t_\mathrm{f} = 100$.

Figures 4.8 and 4.9 showcase the improvement in trajectory following accuracy during the repeating phase compared to the experiments with the baseline framework. Furthermore, the error graphs in figure 4.10 presents less disparity between the motion capture data and the position estimates. Hence, one can conclude that the Gaussian Process models, which are derived from the collected training data and embedded in the posterior update step of the EKF, are able to correct the localization errors which stem from the systematic biases of the UWB measurements as well as the erroneous anchor estimates. Due to the congruency in the error graphs, it can be inferred that the range measurements are modeled such that the resulting position estimates directly describe the drone's center of gravity. In the following sections of this chapter, the robustness of the proposed solution is further examined.

4.4 Analysis of Gaussian Process Regression

Figure 4.11 represents the evaluation of the predicted mean $\mu_{i,\mathrm{GP}}(\boldsymbol{r}_k, \boldsymbol{D}_i)$ and standard deviation $\sigma_{i,\mathrm{GP}}(\boldsymbol{r}_k, \boldsymbol{D}_i)$ over the flight domain. Thereby, the ranging bias distribution of the measurements with a single anchor for one of the circular flight experiment is illustrated. Here, the z-coordinates of the testing inputs are fixed at the defined flying height for illustrative purposes.

As expected, the confidence along the specified trajectory is at its highest, whereas the uncertainty grows to a certain value for positions far away from the path. This follows directly from the chosen radial basis kernel function as described in subsection 3.3.2. Specifically, the constant covariance for points $\boldsymbol{r}_{k,\infty}$ far away from the training data can be calculated as following:

$$\sigma^2_{i,\mathrm{GP}}(\boldsymbol{r}_{k,\infty}, \boldsymbol{D}_i) = \sigma_{i,0} + \sigma^2_{\mathrm{i,n}} \tag{4.1}$$

Compared to equation 2.12, the middle term disappears due to its convergence to zero. The plot for the predicted means showcases the kernel regression plane which fits the training data. Similar as for the covariance,

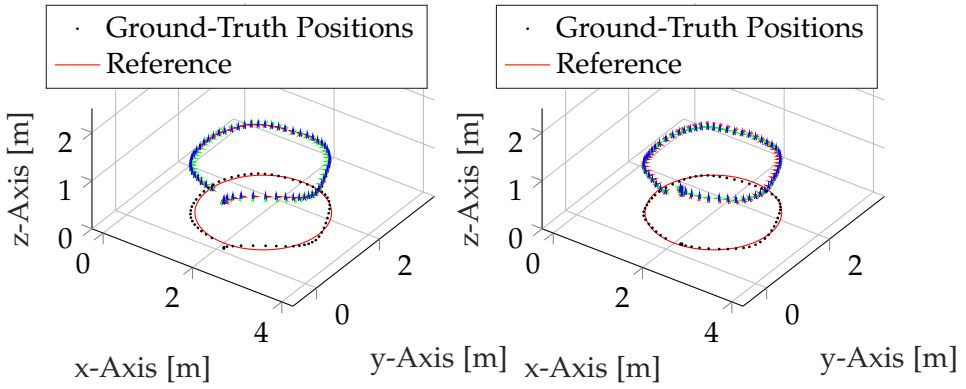

Figure 4.8: Ground-truth measurements of the circular trajectory flights where the corrected measurement model for state estimation with erroneous UWB distance data was used. As in figure 4.3, the body-fixed frame over time for both heading types as well as the corresponding projected ground-truth measurements (black) and reference trajectory (red) are illustrated.

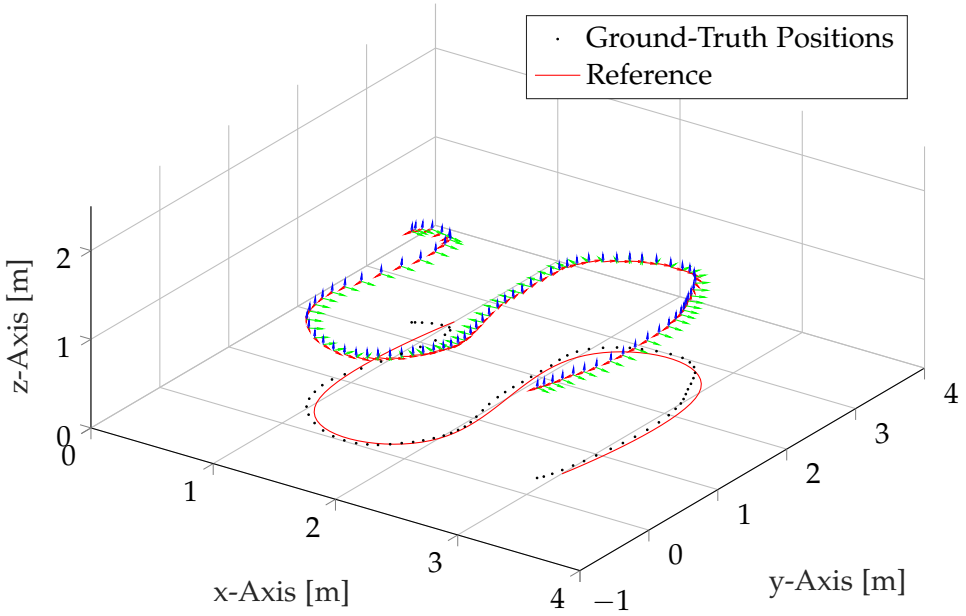

Figure 4.9: Ground-truth measurements of the spline trajectory flight where the corrected measurement model for state estimation with erroneous UWB distance data was used. As in figure 4.4, the body-fixed frame over time as well as the corresponding projected ground-truth measurements (black) and reference trajectory (red) are illustrated.

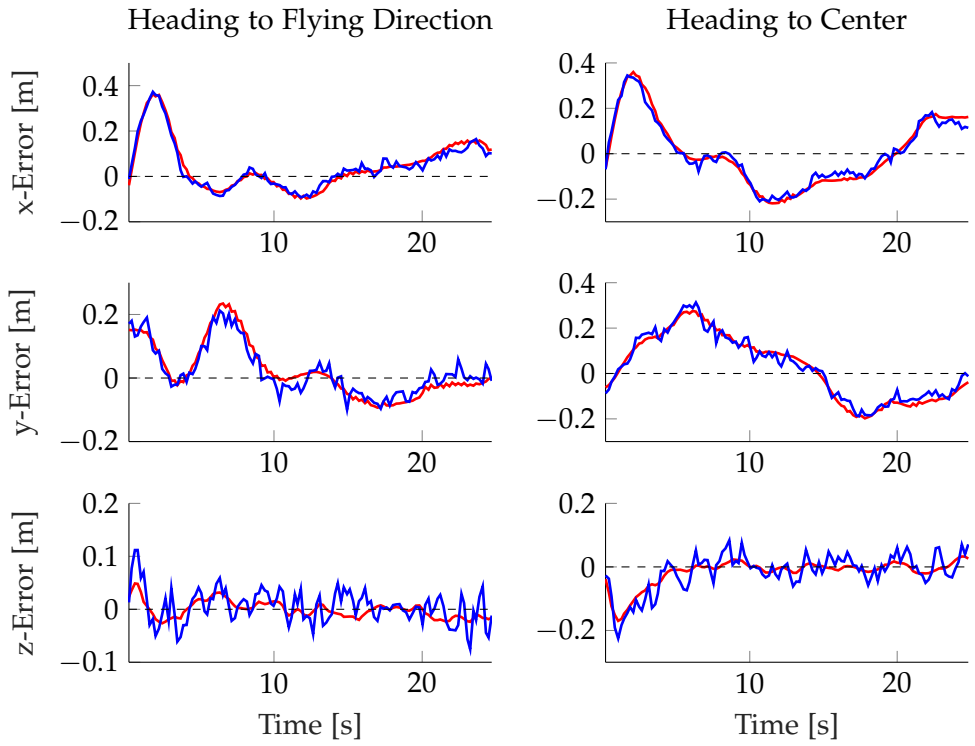

Figure 4.10: Error-time graphs of the circular trajectory flights where the corrected measurement model for state estimation with erroneous UWB distance data was used. Analogously to figure 4.7, the chronological sequences of the errors in each dimension are illustrated, where the red and blue lines denote the errors of the ground-truth measurements and the position estimations respectively.

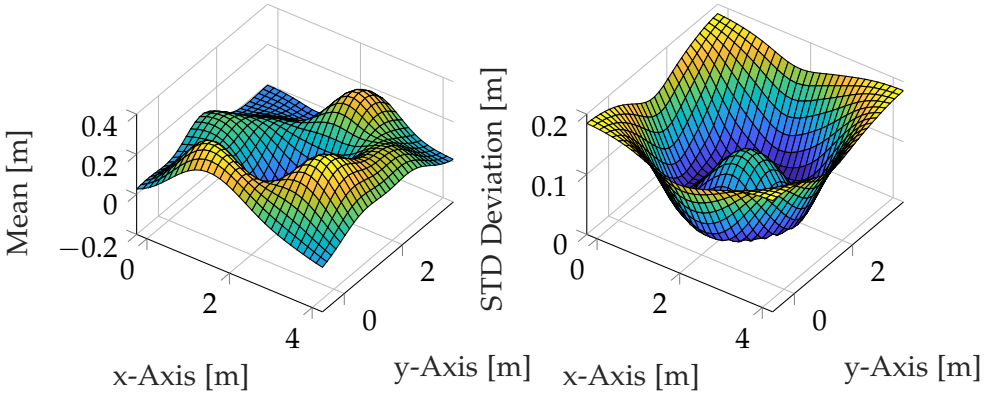

Figure 4.11: Mean (left) and standard deviation (right) predictions for the ranging offsets with a single anchor over the flying space.

it can be shown that the predictive mean for the offsets converges to zero for input points far away from the specified trajectory. Hence, for positions outside the known area, the individual measurement models decompose to their vanilla formulations (see equation 3.3).

4.5 Correlation between Data Size and Accuracy

This section quantitatively evaluates the accuracy of the trajectory following task by the root-mean-square deviation between the *Vicon* position measurements and their corresponding reference. A closer look is taken at the circular flight tests where the heading is pointing in the flying direction. As a benchmark value, the one using accurate *Vicon*-feedback control (see subsection 4.2.1) achieves a RMSE of 0.06 meters. In contrast, the baseline experiment from subsection 4.2.2, where the corresponding control-feedback is calculated with erroneous UWB distance data and the uncorrected measurement model, has a RMSE of 0.2 meters. The training set of the corrected variant as described in section 4.3 consists of $n = 284$ data pairs. These are gathered by lower the circumnavigation frequency down to $f = \frac{1}{60}$, which is relatively slow compared to the usual flight speed. This particular experiment showcases a RMSE value of 0.15 meters. Furthermore, it is shown that more training data collected during a even slower flight does not lead to greater improvements. In contrast, it is interesting to test the trend of the RMSE for lower training sets, which gives an idea on the required execution speed of the data collection flight. Therefore, this section examines the correlation between the size of the training dataset and the achieved flight accuracy. For comparison purposes, downscaled variants of the same dataset were used for hyperparameter learning and the subsequent repeating flights. Specifically, for a downscale factor s, the reduced dataset was created by choosing every s-th entry from the original set. For every flight, it was ensured that no changes to the immediate surroundings of the flying space was made in order to not influence the radio propagation. Table 4.3 lists the results of the experiments:

Downscale Factor	1	2	3	4
RMSE [m]	0.15	0.17	0.18	0.2

Table 4.3: The root-mean-square errors corresponding to specific downscale factors.

As expected, with the decrease of training data, an increase of the error can be observed. Starting at a downscale factor of 4, no accuracy improvement in terms of RMSE can be observed between the learning-based and the baseline method. This can be explained by the fact, that with less training data, spatial changes and fluctuations of the biases remain undetected, which lead

to vague modeling of the offset distributions.

4.6 Impact of Angular Divergences

In order to simulate the dolly application described in section 1.2 with UAVs, one need to be able to change the view direction of the body-fixed camera while the drone is following a predefined trajectory. This does not present a problem for drones which are equipped with panoramic rotating gimbals as displayed in figure 1.2. In contrast, the *Bebop* used in this project is only assembled with one static camera facing in the front-direction (see subsection 1.7.2). For such a performance, besides following the track, the quadcopter additionally requires to adapt its attitude according to the given user command. Providing accurate pose measurements from a motion capture system, this task is reduced down to a control issue. However, in the context of the learning-based approach described in this thesis, the main problematic stems from the change of the ranging biases upon turning the body-fixed tag. In order to test the extent of angular divergences from the data collection flight, models trained on the data gathered during the circular flight where the heading is pointing towards the flying direction were applied to guidance data for the flight where the heading is pointing towards the center of the circular trajectory.

As can be noticed from figure 4.12, the accuracy of the trajectory following decreases compared to the experiment from section 4.3. This is expected, since the bias models are derived from data gathered during a flight with different orientations. Consequently, wrong spatial-dependent offset corrections terms are provided for the repeating flight. Note that in this experiment, the influencing factors as described in subsection 2.1.5 were kept constant for the most part except for the orientation of the tag, which is already enough to lower the localization accuracy.

4.7 Impact of Large Pose Disturbances

Given the analysis of the predictions in section 4.4, it remains to be investigated on how robust the flight is against large pose errors from the reference. All the flight experiments above were conducted by placing the drone close to the initial reference position and with the correct orientation before take-off. In contrast, for the following test, the drone was settled far away from the starting position where no data were collected during the data collection phase and with its heading pointing in an arbitrary direction.

As can be seen in figure 4.13, there are large deviations at the beginning between the measurements from the motion capture system and the position estimates, which can be comparable to the performance of the uncorrected

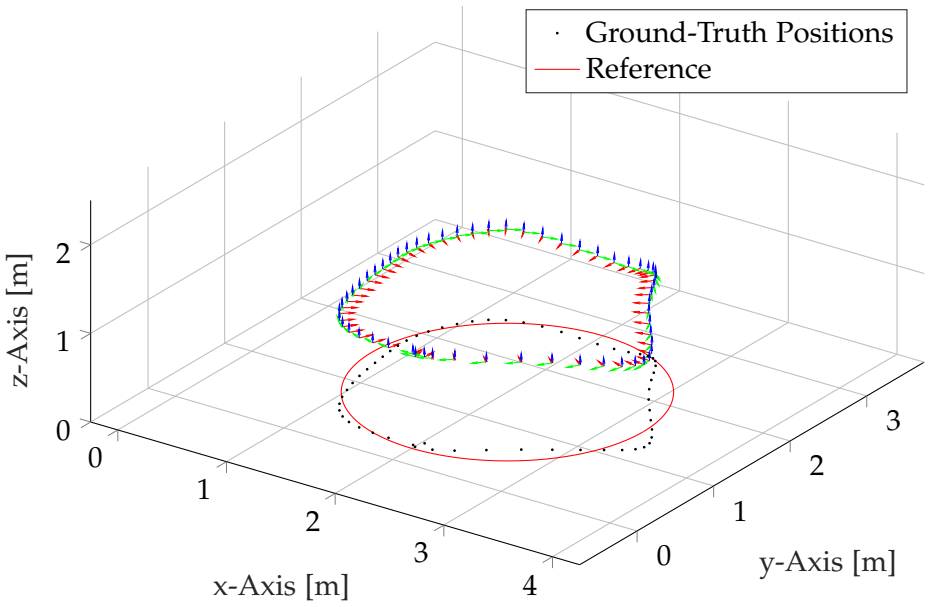

Figure 4.12: Ground-truth measurements of the circular trajectory flight where the corrected measurement model for state estimation with erroneous UWB distance data was used. Here, the drone was commanded to face towards the center. However, the offset models were trained on data gathered during the flight where the heading was pointing in flying direction. As in figure 4.3, the body-fixed frame over time as well as the corresponding projected ground-truth measurements (black) and reference trajectory (red) are illustrated.

measurement model as illustrated in figure 4.7. However, after the reduction of this disparity and some initial oscillations, the positional error converges to zero and the trajectory following reaches the accuracy as described section 4.3. This is beneficial insofar as the vanilla measurement model (see section 4.4) fullfills a robust fail-safe functionality in the case of possible disturbances.

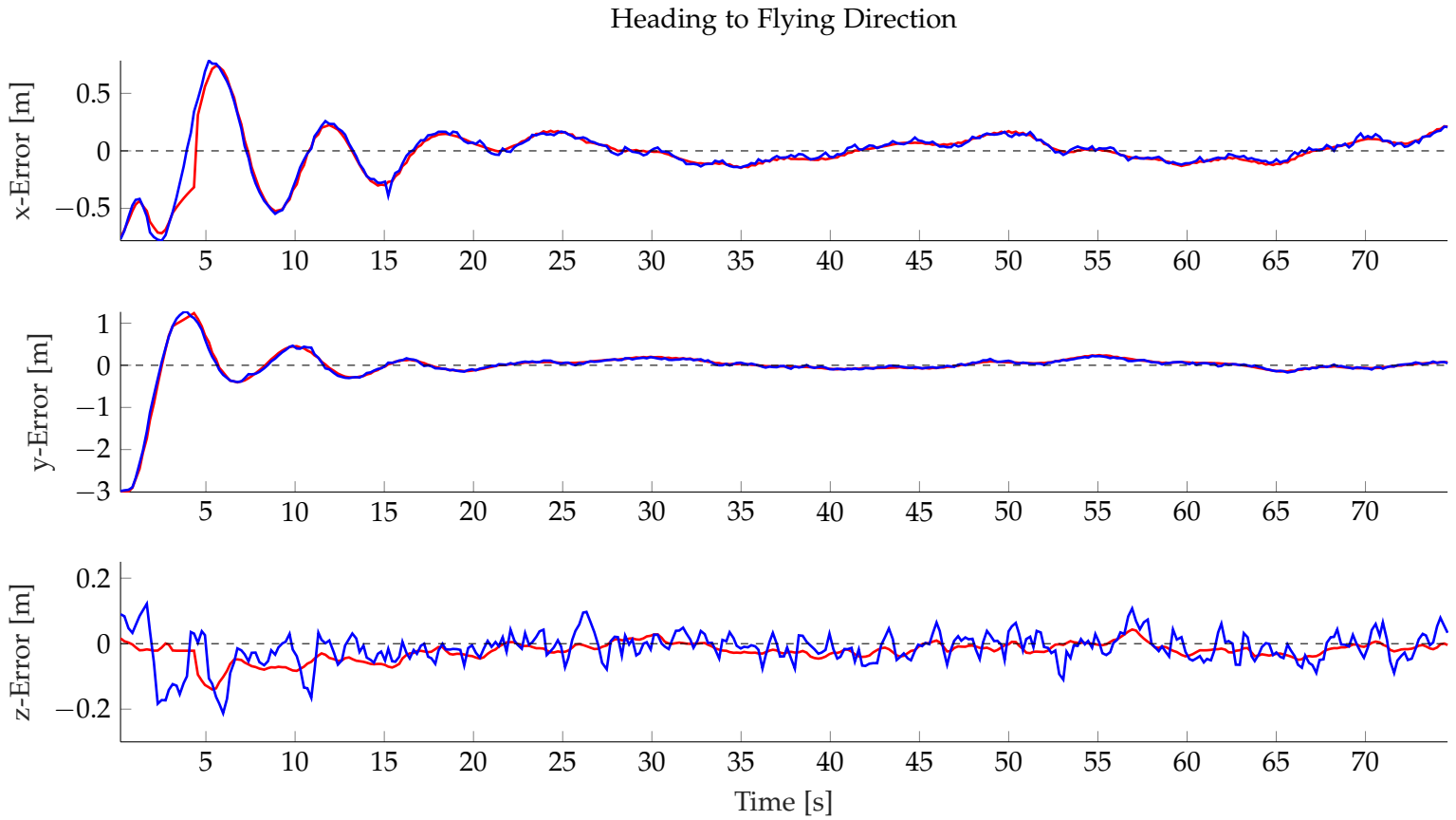

Figure 4.13: Error-time graphs in the case of large initial pose errors. Analogously to figure 4.7, the chronological sequences of the errors in each dimension are illustrated, where the red and blue lines denote the errors of the ground-truth measurements and the position estimations respectively. After extensive fluctuations in the x- and y-dimensions, the drone returns to its assigned trajectory.

Chapter 5

Conclusion

5.1 Achievements

The goal of this master thesis is to develop a localization method for learning-based UAV flights using commercially available UWB technology. The localization is based on filtered multilateration using distance measurements between the drone-mounted tag and several environment-fixed anchors. In this context, methods are proposed to estimate the anchor positions as well as to correct the systematic errors of the range measurements. The former is solved by means of a Gauss-Newton optimization using the collected distance data from individual anchor pairs. For the latter, Gaussian Process models are derived from training data which describe the offset distributions along the flight trajectory for the measurements with each anchor. The training data are collected during the data collection phase, where the drone is commanded to follow a specified trajectory while using accurate pose feedback from a motion capture system. With the derived models during the learning phase, the same trajectory flight can be repeated using raw UWB measurements to estimate the state feedback. Thereby, the Gaussian Process models provide correction terms for the measurement equations in the estimation algorithm.

The developed framework was tested in an indoor laboratory space and showcases improved performance in terms of trajectory following accuracy compared to the baseline framework, which does not correct for the distance estimations. Furthermore, it is robust against pose disturbances in situations, where the drone is not located along the path with known offsets. Here, the degenerated baseline method with its uncorrected measurement model provides a fail-safe functionality in order to bring the drone back on its desired path. However, the experiments also demonstrate some drawbacks. The data collection flight has to be executed slowly in order to guarantee for significant training data. Furthermore, the presented learning-based approach

can not directly be applied for the simulation of dollies, since changing the orientation of the drone during the repeating flight leads to inaccurate trajectory following.

5.2 Future Work

This section describes possible further works, which amongst others contains interesting extensions to the developed framework. Besides the obvious improvements, such as incorporating a full quadrotor kinematic model or a model predictive controller to improve trajectory following accuracy, the following subsections specifically lists subsequent works in the context of further experiments as well as modeling the systematic ranging errors. Furthermore, a possible application with the technology from *Tinamu Labs* is described.

5.2.1 Impact of Obstacles

The influence of immediate obstacles on the accuracy of the flight was tested. Thereby, an obstacle course was placed using cardboards likewise to the one as described in [12]. Using the learning-based approach in this new environment, similar accuracy results can be achieved as described in section 4.3. However, the generalization of these findings remains questionable. It can be presumed, that the accuracy would decrease in environments with the presence of dense structures[1] or radio reflective surfaces due to the absence of signal reception or large fluctuations in the range readings. Furthermore, it has also to be tested to what extent moving objects within the flying space would influence the resulting flight during the repeating phase.

5.2.2 Including Orientation Term in the Kernel Function

The chosen radial basis kernel for this project has the limitiation, that the constructed trajectory can not have cross sections with different tag orientations[2]. By including an orientation term to the kernel function similar to equation 3.20 as well as estimating the current orientations, an additional degree to describe similarity is available, which allows for more complex flight maneuvers. Furthermore, greater accuracy can be achieved for the dolly simulation as described in section 4.6. Here, the modified data collection flight consists of following the predefined trajectory while simultaneously performing yaw rotations. Thus, not only the spatial changes of the range measurements are captured, but also the variations due to angular shifts.

[1] For instance in case of concrete walls.

[2] Different tag orientations at a cross section can occur, for instance, in case of a figure-eight spline, where the heading of the drone is pointing in flying direction.

5.2.3 Adaption for Large Dataset

As can be noted from equation 2.11 and 2.12, for the interrogation with one anchor, the inverse of the specific matrix $\left[\boldsymbol{K} + \sigma_{i,\mathrm{n}}^2 \boldsymbol{I}\right]^{-1}$ is required for the prediction step, which utilizes $\mathcal{O}(n^3)$ running time. Once this inversion is calculated, prediction is $\mathcal{O}(n)$ for the conditional expectation and $\mathcal{O}(n^2)$ for the variance. In case of longer spline trajectories, where the corresponding training dataset is large, this can problematical in terms of computational time for each prediction step. In contrast, by following the approach proposed by Snelson and Ghahramani [27], a model can be described by m so-called pseudo-input positions instead. It is shown, that with their approach, prediction time per test case is reduced to $\mathcal{O}(m)$ for the conditional expectation and $\mathcal{O}(m^2)$ for the variance, where $m \ll n$. Ledergerber and D'Andrea [18] already demonstrate prosperity with this approach for datasets up to 25,000 training points.

5.2.4 Model Local Uncertainties

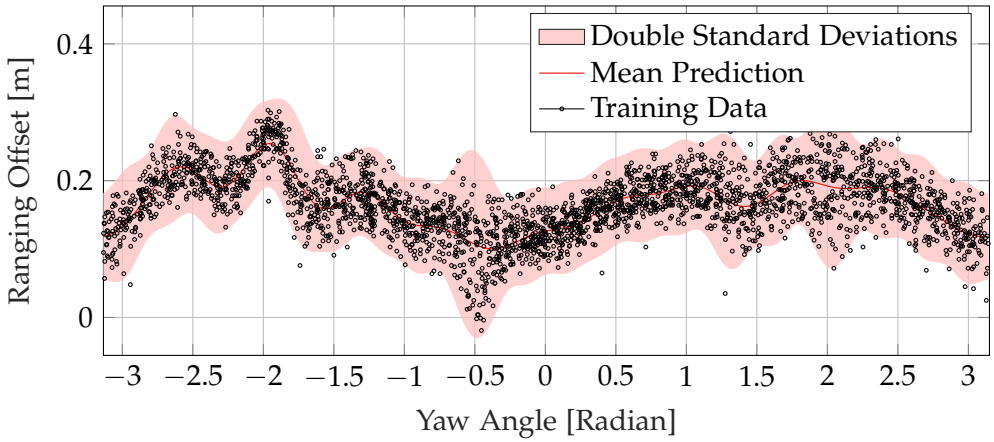

Figure 5.1: Compared to figure 2.4, the correlation illustrated in figure 1.3 was captured by the Sparse Gaussian Process method. As can be seen, the sparse approach is able to increase the standard deviations locally at regions with larger bias fluctuations.

As indicated in subsection 2.1.5, the mean biases are also governed by objects in the immediate vicinity of the antennas due to multipath propagation. As illustrated in figure 5.1, a fluctuation of the offsets can be observed if the line-of-sight is blocked by a radio reflective object, which stems from the distortion of the signal image in the channel impulse response. For such non-line-of-sight conditions, the standard Gaussian Process method does not capture the local uncertainties very well. In contrast, the aforementioned method (see subsection 5.2.3), so-called Sparse Gaussian Process, is much more suitable for capturing these local fluctuations [18] and offers more

robustness against them.

5.2.5 Increasing the Update Rate

The current framework requires the availability of all 6 distance measurements for the posterior update, which results in an update rate of approximate 5 Hz. In contrast, by evaluating the update step upon the reception of a single range estimation, the processing rate can be increased.

5.2.6 Possible Application for Tinamu Labs

Figure 5.2: An user controlling the orientation of the drone with a handheld device[3].

In this project, *Vicon* is used as a reference system in order to build a proof-of-concept prototype. Nonetheless, the framework can be adapted to employ the *VirtualRails* technology from *Tinamu Labs* instead. Analog to the usage of dollies in movie productions, the desired flight path is constructed by lining a custom-made LED-stripe on the ground as indicated in figure 5.3. UAVs, which are equipped with the required hardware, can visually detect these manually set landmarks using computer vision algorithms and follow the specified flight trajectory. The method reports an accurate estimate of the current position along the path, which can be used as training data for the proposed framework in this thesis. This can be useful for developing a

[3]Image from *Tinamu Labs*.

Figure 5.3: Application of the *VirtualRails* technology[4]from *Tinamu Labs* in industrial indoor-environments.

long-term autonomous inspection solution, where it is not suitable to have the LED-stripe permanently laid-out due to space constraints. Here, the *VirtualRails* can be used once to train the drone for a certain flight routine. Afterwards, the routine can be autonomously repeated on command using UWB modules, which are evenly distributed along the walls. Furthermore, given the inclusion of the orientation in the kernel function as described in subsection 5.2.2, the user can have full control over the camera view along the path by using a handheld device as illustrated in figure 5.2.

[4]Image from *Tinamu Labs*.

Appendix A

Appendix

A.1 Repository

All links listed in this section were last visited on 30. November 2019. The following *Gitlab* repository stores all the source code as well as the results from the experiments for this project:

- https://www.gitlab.com/jdiep/master-thesis

A.1.1 Installation

This installation was tested for *Ubuntu 16.04 LTS*. For other operating systems, changes or additional packages might be required. The following software and packages were used:

- *Matlab 9.6*: https://www.mathworks.com/products/matlab.html
- *ROS Kinetic*: http://wiki.ros.org/kinetic/Installation/Ubuntu
- `picocom` serial communication program:
  https://www.github.com/npat-efault/picocom
- `bebop_autonomy` *ROS* driver for the *Bebop* drone:
  https://www.bebop-autonomy.readthedocs.io/en/latest/
- `vicon_bridge` *ROS* driver for the *Vicon* motion capture system:
  http://wiki.ros.org/vicon_bridge
- `spacenav_node` *ROS* driver for the *Spacemouse*:
  http://wiki.ros.org/spacenav_node
- `GPy` Gaussian Process *Python* framework (optional):
  https://www.github.com/SheffieldML/GPy

A.1.2 Matlab Code Structure

All the source code for the flight logic are stored in the `matlab` folder and commented extensively. The following lists the most important classes and functions:

- `BebopControl.m`: Interface to the `bebop_autonomy` library.
- `getGroundTruth.m`: Interface for the *Vicon* measurements.
- `Controller.m`: Implementation of the position and attitude controller.
- `TrajectoryGenerator.m`: Construction of the specific trajectory.
- `ConstantVelocityEKF.m`: EKF implementation using *Vicon* data.
- `ConstantVelocityUWB.m`: EKF implementation using UWB data.
- `ConstantVelocityGP.m`: Gaussian Process EKF implementation using UWB data.
- `DataPrep.m`: Data preprocessing for Gaussian Process.
- `RBFKernel.m`: Calculation of the radial basis kernel matrix.
- `getLogLikelihood.m`: Returning the marginal log likelihood.
- `GaussianModel.m`: Returning all the terms for Gaussian Process regression.
- `GaussianPrediction.m`: Returning the distribution for a specific input.

The following lists the corresponding executable scripts:

- `AnchorCalibMain.m`: Self-calibration of the anchor setup.
- `CircOffsetDataMain.m`: Collection of data along the circular trajectory.
- `SplineOffsetDataMain.m`: Collection of data along the spline trajectory.
- `CovEvalMain.m`: Implementation of the Gaussian Process hyperparameter learning.
- `CircleControl.m`: Circular flight using *Vicon* data.
- `UWBCircleControl.m`: Circular flight using UWB data and uncorrected measurement model.
- `GPSCircleControl.m`: Circular flight using UWB data and corrected measurement model.
- `SplineControl.m`: Spline flight using *Vicon* data.

- `UWBSplineControl.m`: Spline flight using UWB data and uncorrected measurement model.
- `GPSplineControl.m`: Spline flight using UWB data and corrected measurement model.

A.1.3 UWB Firmware

The modification of the *Loco Positioning Node* firmware, which is described in section A.2, is located in the `lps-node-firmware` folder. The vanilla version and instructions on building and flashing the software as well as on how to configure the modes, device indices and the radio settings can be found under:

- https://www.github.com/bitcraze/lps-node-firmware
- https://wiki.bitcraze.io/projects:lps:node
- https://wiki.bitcraze.io/doc:lps:anchor-low-level-config

A.1.4 Starting Instructions

This subsection provide the basic instructions in order to run the executables listed in subsection A.1.2 after installation of the required software (see subsection A.1.1). The following commands should be executed in individual terminal tabs and in order.

Enabling Serial Communication with Matlab

Use this command after connecting the modified sniffer to the ground station computer. Since the exact USB port enumeration can differ, it is recommended to apply command-line completion here.

```
$ sudo chmod 666 /dev/ttyACM0
```

Spacemouse ROS Driver

```
$ roslaunch spacenav_node classic.launch
```

Vicon ROS Driver

```
$ cd vicon_ws
$ source devel/setup.bash
$ cd src/vicon_bridge/launch
$ roslaunch vicon.launch
```

Bebop ROS Driver

```
$ cd bebop_ws
$ source devel/setup.bash
$ cd src/bebop_autonomy/bebop_driver/launch
$ roslaunch bebop_node.launch
```

Bebop Forced Landing

For yet inexplicable reasons, the drone occasionally does not react to the landing command given using the *Spacemouse*. In those cases, the drone can be forced to land by using the following command:

```
$ rostopic pub /bebop/land std_msgs/Empty "{}"
```

A.2 UWB Firmware Modification

The firmware of the *Loco Positioning Node* is written in the *C* language and is designed with supporting the *Crazyflie* quadcopter[1] and its respective platforms in mind. In order to use these off-the-shelf UWB modules as standalone tools for the experiments (see chapter 4), the firmware need to be modified. Specifically, the following requirements need to be fulfilled:

- collecting the ranging measurements between anchors
- as well as between a mobile tag and multiple anchors per radio
- by using an additional sniffer module.

Explaining the complete functionality of the firmware is beyond the scope of this report. Thus, this chapter only covers the customization made in the firmware for this project in note form. References to the specific source files are stated for the interested reader. As for the device indices (see subsection 2.1.3), the following numeration is used:

- the sniffer i_s is set to 0,
- the tag i_t to 7
- and indices for each anchor are numerated from 1 to 6 according to figure 3.1, where i_a is referring to the current anchor which is being interrogated.

According to official documentations[2], the maximal anchor number for the two-way ranging protocol is set to 8. Nonetheless, it is to mention that with

[1] Source: https://www.bitcraze.io/crazyflie-2-1/ 22. October 2019
[2] Source: https://wiki.bitcraze.io/doc:lps:index 27. October 2019

some modifications to the default firmware, this system should be able to scale up to arbitrary number of anchors.

A.2.1 Transmitting Results to the Sniffer

As mentioned in section 1.7.1, the UWB module can be configured in three modes: tag, anchor and sniffer mode. The vanilla version of the latter can be used to retrieve the radio messages exchanged between the tag and each anchor. By default, the tag module does not transmit the calculated distance d to the sniffer. Therefore, the specific source code in the tag firmware is modified such that after each report message from an anchor, the final distance is forwarded to the sniffer module in an additional result message (see figure 2.1). In particular, the following functionalities are added (compare with algorithm 2):

Algorithm 2 The required modifications in the tag firmware.
Code: `/lps-node-firmware/src/uwb_twr_tag.c` (line 178-254)

1: **function** RxCallback()
...
After receiving the report message from an anchor:
2: $t_f \leftarrow$ CalculateTimeOfFlight$(t_{1:6})$
3: $d \leftarrow$ CalculateDistance(t_f)
Transmitting the result message:
4: $p_r \leftarrow$ ResultPackage(d, i_a, i_t, i_s)
5: TransmitMessage(p_r)
6: WaitForResponse()
7: **end function**

- The result nametag is added to the four default options.
- ResultPackage(d, i_a, i_t, i_s): Constructing the result message content p_r consisting of the result nametag, the calculated distance, the device indices of the ranging modules and the device index of the sniffer.
- TransmitMessage(p_r): Transmitting the message to the UWB network using the `dwSetData()`[3] and `dwStartTransmit()`[3] functions. The sniffer module can then receive this message by filtering for its own index after using the `dwStartReceive()`[3] function.
- WaitForResponse(): With the function `dwWaitForResponse()`[3], a waiting time can be set before starting interrogating with the next anchor. This can be convenient if a response message is expected.

[3] Sourcecode: `/lps-node-firmware/vendor/libdw1000/src/libdw1000.c` line: 642-657, 424-438, 469-472 and 411-415

A.2.2 Inter-Anchor Interrogation

Section 3.1 describes an optimization approach to determine the unknown anchor coordinates through a set of inter-anchor distance measurements. However, the default firmware does not enable anchors to interact with each other. Manually setting anchors individually into tag mode over a physical connection is time-consuming and inconvenient. Therefore, the idea is to use an additional sniffer module connected to the ground-station in order to switch the mode of each anchor to tag via radio. In particular, the modified sniffer should fulfill the following criteria:

- switching a specific anchor with an explicit device index into a tag at command, allowing it to range with the other anchors according to the two-way ranging protocol,
- retrieving the distance measurements from the temporary tag (modified according to algorithm 2), these data should then be available at the ground-station via serial port communication
- and switching the tag back into anchor mode after a certain number of measurements at command.

The following list details the implementation of these functionalities (compare with algorithm 3):

- Two additional nametags are added: switch and switchback.
- SwitchCommand() and SwitchBackCommand(): these are functions which return boolean variables for switching an anchor into a tag and vice versa. They listen to the respective user command given over serial port communication.
- SwitchPackage(i_a): constructing the switch message content p_s consisting of the switch nametag and the device index of the targeted anchor.
- TransmitMessage(p_s): Transmitting the message p_s to the UWB network as described in previous subsection A.2.1. The targeted anchor can then switch to tag mode by using the `cfgWriteU8()`[4] function followed by a system reset with `NVIC_SystemReset()`[5].
- For simplicity, the current anchor index is automatically set to 1 at system booting of the sniffer module and gets incremented after each command, resulting in each anchor being switched in order. Hence, the user does not need to specify the targeted anchor.

[4]Sourcecode: `/lps-node-firmware/src/cfg.c` line: 192-210
[5]Sourcecode: `/lps-node-firmware/inc/core_cm0.h` line: 676-684

Algorithm 3 The modified sniffer firmware.
Code: `/lps-node-firmware/src/uwb_sniffer.c` (line 57-134)

```
1:  function TwrSnifferOnEvent()
2:      if SwitchCommand() then
    Transmitting a switch command to the anchor:
3:          p_s ← SwitchPackage(i_a)
4:          TransmitMessage(p_s)
    Incrementing the device index after each command:
5:          if i_a = 6 then
6:              i_a = 1          ▷ return back to the first anchor 1 after a cycle
7:          else
8:              i_a = i_a + 1
    Receiving a result message:
9:      if PackageReceived() then
10:         if nametag = result then
    Possible transmission of a switchback message to the tag:
11:             if SwitchBackCommand() then
12:                 p_sb ← SwitchBackPackage(i_a)
13:                 TransmitMessage(p_sb)
14:             DisplayResultToSerial(p_r)
15: end function
```

- A similar approach is used to switch the tag back to its previous anchor mode. Notice that mode changing does not affect the device index, thus $i_t = i_a$.
- Since the module in tag mode is mostly occupied with two-way ranging, there is the possibility for a switchback message at arbitrary time instance getting blocked due to network traffic. Therefore, if commanded, it is send immediately after receiving the result message within the waiting time of the tag (see figure 2.1).
- DisplayResultToSerial(): Displaying the distance measurements such that they are available for serial communication at the ground-station.

A.2.3 Serial Communication

Distance measurement collection from the setup was done in *Matlab* by using its serial port communication interface[6]. Specifically, by using the `fwrite()` and `fgetl()` functions, the switch and switchback commands can

[6] https://www.mathworks.com/help/instrument/serial-port-interface.html
24. October 2019

be transmitted and the range data from each anchor pair can be collected from a script.

A.3 Gauss-Newton Algorithm

The Gauss-Newton algorithm can be used to solve non-linear least squares problems [13]. Given a vector $\boldsymbol{r}$ consisting of n residual functions with m parameters σ, where $n > m$, the algorithm finds the optimal values $\hat{\sigma}$ which minimizes the following sum of the squares:

$$\hat{\sigma} = \arg\min_{\sigma} \sum_{i=1}^{n} \left| r_i(\sigma) \right|^2 \tag{A.1}$$

Due to its non-linear characteristic of the sum with respect to the parameters, there is no closed-form solution to this problem. Instead, with an appropriate initialization σ_0 for the parameter vector, a solution can be found in an iterative approach based on gradient descent:

$$\sigma_{k+1} = \sigma_k - \left[\boldsymbol{J}_{\mathrm{r}}^{\top} \boldsymbol{J}_{\mathrm{r}} \right]^{-1} \boldsymbol{J}_{\mathrm{r}}^{\top} \boldsymbol{r}(\sigma_k) \tag{A.2}$$

Here, $\boldsymbol{J}_{\mathrm{r}}$ defines the Jacobian matrix of the residual vector with respect to the grouped parameter vector evaluated at the current estimate σ_k. The algorithm terminates when the change of the parameters is below a certain threshold.

A.4 Homogeneous Transformation

Generally, two coordinate frames $\mathcal{A}$ and $\mathcal{B}$ have a position offset and a relative rotation. In this case, the homogeneous transformation matrix $\boldsymbol{T}_{\mathcal{AB}}$, which combines translation and rotation, can be used to transform a point ${}_{\mathcal{B}}\boldsymbol{r}_{\mathrm{BP}}$ from frame $\mathcal{B}$ to frame $\mathcal{A}$ [13]:

$$\begin{bmatrix} {}_{\mathcal{A}}\boldsymbol{r}_{\mathrm{AP}} \\ 1 \end{bmatrix} = \underbrace{\begin{bmatrix} \boldsymbol{R}_{\mathcal{AB}} & {}_{\mathcal{A}}\boldsymbol{r}_{\mathrm{AB}} \\ \boldsymbol{0}_{1\times 3} & 1 \end{bmatrix}}_{\boldsymbol{T}_{\mathcal{AB}}} \begin{bmatrix} {}_{\mathcal{B}}\boldsymbol{r}_{\mathrm{BP}} \\ 1 \end{bmatrix} \tag{A.3}$$

The inverse of the homogeneous transformation matrix can be calculated as following:

$$\boldsymbol{T}_{\mathcal{AB}}^{-1} = \begin{bmatrix} \boldsymbol{R}_{\mathcal{AB}}^{\top} & -\boldsymbol{R}_{\mathcal{AB}}^{\top} {}_{\mathcal{A}}\boldsymbol{r}_{\mathrm{AB}} \\ \boldsymbol{0}_{1\times 3} & 1 \end{bmatrix} \tag{A.4}$$

A.5 Quaternion Conversion

A unit quaternion of the form $\hat{\boldsymbol{u}} = \begin{bmatrix} u_\mathrm{w}, \boldsymbol{u} \end{bmatrix}^\top = \begin{bmatrix} u_\mathrm{w}, u_\mathrm{x}, u_\mathrm{y}, u_\mathrm{z} \end{bmatrix}^\top$ can be converted to rotation matrix form by the following equation [13], whereas $\begin{bmatrix} \boldsymbol{u} \end{bmatrix}_\times$ denotes the skew symmetric matrix of $\boldsymbol{u}$:

$$\begin{gathered} \boldsymbol{R}_{\mathcal{AB}} = \boldsymbol{I}_{3\times 3} + 2u_\mathrm{w}\begin{bmatrix} \boldsymbol{u} \end{bmatrix}_\times + 2\begin{bmatrix} \boldsymbol{u} \end{bmatrix}_\times^2 = \\ \begin{bmatrix} 1-2\left(u_\mathrm{y}^2+u_\mathrm{z}^2\right) & 2\left(u_\mathrm{x}u_\mathrm{y}-u_\mathrm{w}u_\mathrm{z}\right) & 2\left(u_\mathrm{x}u_\mathrm{z}+u_\mathrm{w}u_\mathrm{y}\right) \\ 2\left(u_\mathrm{x}u_\mathrm{y}+u_\mathrm{w}u_\mathrm{z}\right) & 1-2\left(u_\mathrm{x}^2+u_\mathrm{z}^2\right) & 2\left(u_\mathrm{y}u_\mathrm{z}-u_\mathrm{w}u_\mathrm{x}\right) \\ 2\left(u_\mathrm{x}u_\mathrm{z}-u_\mathrm{w}u_\mathrm{y}\right) & 2\left(u_\mathrm{y}u_\mathrm{z}+u_\mathrm{w}u_\mathrm{x}\right) & 1-2\left(u_\mathrm{x}^2+u_\mathrm{y}^2\right) \end{bmatrix} \end{gathered} \tag{A.5}$$

Furthermore, quaternions can also be converted to Euler angles using the following equation:

$$\boldsymbol{s}(\hat{\boldsymbol{u}}) = \begin{bmatrix} \mathrm{atan2}\Big(2\left(u_\mathrm{y}u_\mathrm{z}+u_\mathrm{w}u_\mathrm{x}\right), 1-2\left(u_\mathrm{x}^2+u_\mathrm{y}^2\right)\Big) \\ \mathrm{asin}\Big(2\left(u_\mathrm{w}u_\mathrm{y}+u_\mathrm{z}u_\mathrm{x}\right)\Big) \\ \mathrm{atan2}\Big(2\left(u_\mathrm{x}u_\mathrm{y}+u_\mathrm{w}u_\mathrm{z}\right), 1-2\left(u_\mathrm{y}^2+u_\mathrm{z}^2\right)\Big) \end{bmatrix} \tag{A.6}$$

Here, the order of rotation angles correspond to the 3-2-1 sequence.

A.6 Natural Cubic Splines

This section gives a brief introduction to the theory of natural cubic splines [16]. They are constructed by n piecewise third-order polynomials passing through $n+1$ checkpoints given at $\boldsymbol{x}_\mathrm{d} = \begin{bmatrix} x_0, x_1, ..., x_n \end{bmatrix}$ with values $\boldsymbol{y}_\mathrm{d} = \begin{bmatrix} y_0, y_1, ..., y_n \end{bmatrix}$. Note that the datapoints in $\boldsymbol{x}_\mathrm{d}$ has to be monotonically increasing. The general equation for the spline is written as:

$$s(x) = \begin{cases} s_0(x) = \alpha_0 x^3 + \beta_0 x^2 + \gamma_0 x + \delta_0, & x_0 \le x \le x_1 \\ \quad \vdots & \\ s_{n-1}(x) = \alpha_{n-1} x^3 + \beta_{n-1} x^2 + \gamma_{n-1} x + \delta_{n-1}, & x_{n-1} \le x \le x_n \end{cases} \tag{A.7}$$

The corresponding coefficients of each polynomial are chosen such that the function values as well as the first and second derivatives at each checkpoint between adjacent polynomials are matching. Furthermore, the second derivatives at the endpoints of the spline are set to zero. With that, the problem can be reformulated as solving the following tridiagonal system:

$$\begin{bmatrix} v_1 & h_1 & & & & \\ h_1 & v_2 & h_2 & & & \\ & h_2 & v_3 & h_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & h_{n-2} \\ & & & & h_{n-2} & v_{n-1} \end{bmatrix} \begin{bmatrix} z_1 \\ z_2 \\ z_3 \\ \vdots \\ z_{n-2} \\ z_{n-1} \end{bmatrix} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ \vdots \\ u_{n-2} \\ u_{n-1} \end{bmatrix} \tag{A.8}$$

Whereby the variables are calculated as:

$$\begin{aligned} h_i &= x_{i+1} - x_i \\ b_i &= \frac{1}{h_i}\big(y_{i+1} - y_i\big) \\ v_i &= 2\big(h_{i-1} + h_i\big) \\ u_i &= 6\big(b_i - b_{i-1}\big) \end{aligned} \tag{A.9}$$

Hence, given $z_0 = z_n = 0$, the i-th spline can be written with the above expressions as following:

$$\begin{aligned} s_i(x) = \frac{z_{i+1}}{6h_i}\big(x - x_i\big)^3 + \frac{z_i}{6h_i}\big(x_{i+1} - x\big)^3 + \\ \big(\frac{y_{i+1}}{h_i} - \frac{z_{i+1}}{6}h_i\big)\big(x - x_i\big) + \big(\frac{y_i}{h_i} - \frac{h_i}{6}z_i\big)\big(x_{i+1} - x\big) \end{aligned} \tag{A.10}$$

Bibliography

[1] Abdulrahman Alarifi, AbdulMalik Al-Salman, Mansour Alsaleh, Ahmad Alnafessah, Suheer Alhadhrami, Mai Al-Ammar, and Hend Al-Khalifa. Ultra wideband indoor positioning technologies: Analysis and recent advances. *Sensors*, 16:1–36, 05 2016.

[2] Valentín Barral, Carlos Escudero, José García-Naya, and Roberto Maneiro-Catoira. Nlos identification and mitigation using low-cost uwb devices. *Sensors*, 19:3464, 08 2019.

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[4] Andrea Carron. Lecture Notes in Signals and Systems. https://ethz.ch/content/dam/ethz/special-interest/mavt/dynamic-systems-n-control/idsc-dam/Lectures/Signals-and-Systems/2019/notes/lecture1.pdf, 2019. Online; accessed 18. December 2019.

[5] Guobin Chang. Robust kalman filtering based on mahalanobis distance as outlier judging criterion. *Journal of Geodesy*, 88(4):391–401, Apr 2014.

[6] Rafaello D'Andrea. Lecture Notes in Recursive Estimation. https://www.ethz.ch/content/dam/ethz/special-interest/mavt/dynamic-systems-n-control/idsc-dam/Lectures/Recursive-Estimation/Lecture%20Notes/Lecture09_new.pdf, 2019. Online; accessed 3. December 2019.

[7] DecaWave. Sources of Error in DW1000 based Two-Way Ranging (TWR) Schemes. https://www.decawave.com/sites/default/files/resources/aps011_sources_of_error_in_twr.pdf, 2014. Online; accessed 30. November 2019.

[8] DecaWave. The Implementation of Two-Way Ranging with the DW1000. https://www.decawave.com/wp-content/uploads/2018/10/APS013_The-Implementation-of-Two-Way-Ranging-with-the-DW1000_v2.3.pdf, 2015. Online; accessed 30. November 2019.

[9] DecaWave. DWM1000 IEEE 802.15.4-2011 UWB Transceiver Module. https://www.decawave.com/sites/default/files/resources/DWM1000-Datasheet-V1.6.pdf, 2016. Online; accessed 3. December 2019.

[10] DecaWave. Antenna Delay Calibration of DW1000-based Products and Systems. https://www.decawave.com/wp-content/uploads/2018/10/APS014_Antennna-Delay-Calibration_V1.2.pdf, 2018. Online; accessed 30. November 2019.

[11] Kexin Guo, Zhirong Qiu, Cunxiao Miao, Abdul Zaini, Chun-Lin Chen, Wei Meng, and Lihua Xie. Ultra-wideband-based localization for quadcopter navigation. *Unmanned Systems*, pages 1–12, 02 2016.

[12] Naegeli Tobias Hepp, Benjamin and Otmar Hilliges. Omni-directional person tracking on a flying robot using occlusion-robust ultra-wideband signals. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*. IEEE, 2016.

[13] Marco Hutter, Roland Siegwart, and Thomas Stastny. Lecture Notes in Robot Dynamics . https://rsl.ethz.ch/education-students/lectures/robotdynamics.html, 2009. Online; accessed 9. December 2019.

[14] Sachs Jurgen. *Handbook of Ultra-Wideband Short-Range Sensing: Theory, Sensors, Applications*. Wiley-VCH, 2012.

[15] Jonathan Ko and Dieter Fox. Gp-bayesfilters: Bayesian filtering using gaussian process prediction and observation models. *Autonomous Robots*, 27(1):75–90, Jul 2009.

[16] Arne Morten Kvarving. Lecture Notes in Numerical Mathematics. https://www.math.ntnu.no/emner/TMA4215/2008h/cubicsplines.pdf, 2008. Online; accessed 3. December 2019.

[17] A. Ledergerber and R. D'Andrea. Ultra-wideband range measurement model with gaussian processes. In *2017 IEEE Conference on Control Technology and Applications (CCTA)*, pages 1929–1934, Aug 2017.

[18] A. Ledergerber and R. D'Andrea. Calibrating away inaccuracies in ultra wideband range measurements: A maximum likelihood approach. *IEEE Access*, 6:78719–78730, 2018.

[19] Anton Ledergerber and Raffaello D'Andrea. Ultra-wideband angle of arrival estimation based on angle-dependent antenna transfer function. *Sensors*, 19(20):4466, 2019.

[20] Christophe Leys, Christophe Ley, Olivier Klein, Philippe Bernard, and Laurent Licata. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology*, 49(4):764 – 766, 2013.

[21] Pierre Merriaux, Yohan Dupuis, Rémi Boutteau, Pascal Vasseur, and Xavier Savatier. A study of vicon system positioning performance. *Sensors*, 17:1591, 07 2017.

[22] Sreenivasulu Pala, Sarada Jayan, and Dhanesh G. Kurup. An accurate uwb based localization system using modified leading edge detection algorithm. *Ad Hoc Networks*, 97:102017, 2020.

[23] Mathias Pelka, Grigori Goronzy, and Horst Hellbrück. Iterative approach for anchor configuration of positioning systems. *ICT Express*, 2(1):1 – 4, 2016. Special Issue on Positioning Techniques and Applications.

[24] CE. Rasmussen and CKI. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, Cambridge, MA, USA, January 2006.

[25] Nahum Shimkin. Lecture Notes in Estimation and Identification in Dynamical Systems. http://www.webee.technion.ac.il/people/shimkin/Estimation09/ch8_target.pdf, 2009. Online; accessed 30. November 2019.

[26] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Bradford Company, Scituate, MA, USA, 2004.

[27] Edward Snelson and Zoubin Ghahramani. Sparse gaussian processes using pseudo-inputs. In Y. Weiss, B. Schölkopf, and J. C. Platt, editors, *Advances in Neural Information Processing Systems 18*, pages 1257–1264. MIT Press, 2006.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.