KU LEUVEN

FACULTEIT INGENIEURSWETENSCHAPPEN

Interactive Demo on the Indoor Localization, Control and Navigation of Drones

Rian Beck
Mathias Bos

Thesis voorgedragen tot het behalen van de graad van Master of Science in de ingenieurswetenschappen: werktuigkunde

Promotoren:

Prof. dr. ir. Jan Swevers
Prof. dr. ir. Goele Pipeleers

Assessoren:

Prof. dr. ir. D. Vandepitte
Dr. ir. W. Decré

Begeleiders:

Dr. ir. M. Verbandt
Dr. ir. J. Gillis
Ir. A. Astudillo Vigoya

Academiejaar 2018 – 2019

Preface

We would like to thank our promotors and mentors for granting us a lot of confidence and freedom in shaping the demo we created in this master's thesis. We are also grateful for the many interesting discussions with the researchers from the MECO research team that helped us along the way. We would also like to thank the following people for testing our demo in advance: Liese, Mauricio, Michael, Dries, Kamiel, Paul, Bjorn, Rik, Koen, Lieselot, Anke, Axel, Pieter, Benjamin, Bert, Remco, San and Brecht. Special thanks to Billy and Michelle. Our hope is that this demo will be used frequently and that it will amaze many spectators. Moreover we hope future students will build upon this project to explore many more possibilities of autonomous flying drones and have just as much fun as we did. We would also like to thank the jury for reading the text.

Rian Beck
Mathias Bos

Contents

Abstract

Unmanned aerial vehicles (UAV's) are a subject of great interest in the world of research, industry and commerce. This thesis contributes to the practical aspects of automating drone flight and through the development of a live demo setup offers an interactive view on both recent developments in mechatronics such as Model Predictive Control (MPC) for motion planning purposes and more established basic control principles. As such it aims for a broad range of audiences, addressing uninformed enthusiasts up to experienced researchers.

The study goes into the domains of modeling, localization, control and navigation, and develops a structure to combine all aspects in a framework that is both robust and flexible regarding demo execution. It makes four contributions to these domains. Firstly an asynchronous Kalman filter (AKF) is developed and implemented for position and velocity state estimation with accurate timing handling. Secondly the design and implementation of an inversion-based feedforward controller with zero phase filtering for trajectory tracking is established. Thirdly it integrates the OMG-tools motion planning software and provides an experimental validation. Finally a Finite State Machine (FSM) is developed to yield situation specific behavior with integrated monitoring for detection and safe handling of non-nominal events.

The result of the study is an operational demo setup with a set of visually impressive and interactive tasks, that provides the freedom to execute tasks in any arbitrary order. It is available as an open-source software package, which together with a modular design encourages further contributions to the current setup.

Samenvatting

Onbemande luchtvaartuigen (UAV's) of drones genieten een grote interesse in zowel de onderzoekswereld, de industrie als de commerciële wereld. Deze thesis draagt bij aan de praktische aspecten van automatisatie van drones. De ontwikkelde live demo biedt een interactieve blik op zowel recente ontwikkelingen in de mechatronica zoals Model Predictive Control (MPC) voor motion planning als meer ingeburgerde principes uit de regeltechniek. Hierbij wordt gemikt op een breed publiek, gaande van geïnteresseerde leken tot gespecialiseerde onderzoekers.

De studie beschouwt de modellering, lokalisatie, controle en navigatie van drones, en ontwikkelt een structuur om deze aspecten te combineren in een robuust kader dat flexibiliteit toestaat bij de uitvoering van de demo. Ze stelt vier bijdragen in deze domeinen voor. Een asynchroon Kalman filter (AKF) voor de schatting van positie en snelheid met nauwkeurige tijdsregistratie, het ontwerp en de implementatie van een feedforward controller gebaseerd op modelinversie met zero phase filter voor trajectory tracking, de integratie van de motion planning software OMG-tools en de ontwikkeling van een Finite State Machine (FSM) voor situatiespecifiek gedrag met geïntegreerde monitoring voor het detecteren van en veilig omgaan met niet-nominale situaties.

Het resultaat van de studie is een operationele demo met een reeks visueel indrukwekkende en interactieve taken, waarbij de taken in willekeurige volgorde uitgevoerd kunnen worden. De ontwikkelde software is open-source beschikbaar wat samen met het modulaire design uitnodigt tot verdere uitbreiding van de huidige opstelling.

List of Figures and Tables

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

AGV	Automatic Guided Vehicle
AKF	Asynchronous Kalman Filter
DWA	Dynamic Window Approach
FB	Feedback
FF	Feedforward
FFT	Fast Fourier Transform
FRF	Frequency Response Function
FSM	Finite State Machine
IMU	Inertial Measurement Unit
LPF	Low Pass Filter
LTI	Linear Time-Invariant
MPC	Model Predictive Control
OCP	Optimal Control Problem
OMG-tools	Optimal Motion Generation-tools
PID	Proportional-Integrative-Derivative
PM	Phase Margin
RMS	Root Mean Square
ROS	Robot Operating System
RRT	Rapidly-exploring Random Trees
SISO	Single-Input Single-Output
TEB	Timed Elastic Band
UAV	Unmanned Aerial Vehicle
ZOH	Zero Order Hold
ZPETC	Zero Phase Error Tracking Control
ZP LPF	Zero Phase Low Pass Filter

Symbols

α	Roll or pitch angle (rad)
μ	Mean position measurement noise amplitude (m)
$\nu_{t'_{meas}}$	Innovation
θ	Yaw angle (rad)
$\Theta(s)$	Continuous-time Laplace transform of yaw angle
Θ	Vector of unknown coefficients
σ	Standard deviation on position measurement noise amplitude (m)
ϕ	Phase $(^\circ)$
Φ	Velocity data and input data matrix
ω	Angular frequency (rad/s)
ω_c	Crossover frequency (rad/s)
τ	Time constant (s)
a	Linear acceleration (m/s^2)
a_i	i'th denominator coefficient of continuous-time transfer function
A	Continuous-time state matrix
A_d	Discrete-time state matrix
b_0	Numerator coefficient of continuous-time transfer function
B	Continuous-time input matrix
B_d	Discrete-time input matrix
c	Phase correction factor $(^\circ)$
C	Output matrix
D	Feedthrough matrix
$D(s)$	Continuous-time controller transfer function
e_p	Position error (m)
e_v	Velocity error (m/s)
$E_p(s)$	Continuous-time Laplace transform of position error
$E_v(s)$	Continuous-time Laplace transform of velocity error
$H_{XJ}(s)$	Continuous-time transfer function of the LTI system H with input J and output X
$H_{XJ}(z)$	Discrete-time transfer function of the LTI system H with input J and output X
I_3	Identity matrix of dimension 3x3
j	Normalized drone input command $(-)$
$J(s)$	Continuous-time Laplace transform of drone input commands
$J(z)$	Discrete-time Laplace transform of drone input commands
k	Discrete-time variable $(-)$
K	Controller gain (m^{-1})
K_d	Derivative controller gain $(m^{-1}s)$
K_i	Integrative controller gain $((ms)^{-1})$
K_p	Proportional controller gain (m^{-1})
$L_{t'_{meas}}$	Kalman gain vector

p	Position (m)
$\hat{P}$	Estimation error covariance matrix
$P(s)$	Continuous-time Laplace transform of position
Q	Process noise covariance matrix
R	Measurement noise covariance matrix (m)
${}^{r}_{c}R$	Rotation matrix from child frame to reference frame
s	Continuous-time Laplace variable (s^{-1})
$S_{t'_{meas}}$	Innovation covariance matrix
${}^{r}t_{c}$	Translation vector of child frame expressed in reference frame
t_i	i'th controller time instance (s)
t_{meas}	Measurement time instance (s)
${}^{r}_{c}T$	Transformation matrix from child frame to reference frame/Pose matrix of child frame expressed in reference frame
T_d	Derivative time (s)
T_i	Integration time (s)
T_s	Sampling time (s)
$T_{s,ctrl}$	Controller sample time (s)
$T_{s,meas}$	Measurement system sample time (s)
v	Linear velocity (m/s)
V	Velocity data vector
$V(s)$	Continuous-time Laplace transform of velocity
$V(z)$	Discrete-time Laplace transform of velocity
x	State vector
$\hat{x}_{t_{i+1}\|t_i}$	A priori estimate of x
$\hat{x}_{t_i\|t_i}$	A posteriori estimate of x
$\dot{x}$	Time derivative of x
x_i	i'th state in state space representation
y	Output vector
z	Discrete-time Laplace variable $(-)$
$z_{t'_{meas}}$	Position measurement (m)

Chapter 1

Introduction

In recent years unmanned aerial vehicles (UAV's) have gained great interest. Their use in both industrial and commercial applications is the subject of many ongoing research studies. They can be deployed in locations inaccessible to humans, in situations where human intervention is either difficult or impossible and even when the environment is harmful to humans. A first example of their use is in the inspection of industrial plants, where the bird view of a UAV offers possibilities beyond what human inspection can offer. The Smart Tooling project is one of the initiatives directed towards the development of such applications [1, 2]. Another application which has become increasingly important in research is the autonomous delivery of packages. Nowadays more and more clients place online orders leading to a large increase in package deliveries. A possible solution to facilitate this delivery scheme is to replace traditional delivery vans by UAV's. A third possible use of UAV's is to aid in search and rescue parties, in situations where the terrain is inaccessible or for example to scout a burning building. A common denominator in all of these applications is the need for both localization of the vehicle, as well as the navigation through its surrounding environment.

This thesis is commissioned by and made in collaboration with the MECO research team at KU Leuven (Faculty of Engineering Science, Department of Mechanical Engineering). One of MECO's research domains is the optimal control and autonomous navigation of mechatronic systems. To this end they develop the underlying algorithms as well as the higher level software required to steer UAV's autonomously from point A to point B. An important part of applied research is the ability to showcase the practical relevance and potential in real-life situations. Therefore they wish to develop an indoor drone demo setup, to interactively show the current possibilities in localization, control and navigation.

In order to address that desire, this thesis proposes to let the demo illustrate three different levels of autonomous flight. The first one, called setpoint tracking, amounts to either the drone staying in one particular position as accurately as possible, or tracking a variable setpoint. The second level is the tracking of a trajectory that is given in advance. The third and last level is the most advanced one: it requires the

drone to fly autonomously through the room, by tracking an automatically generated trajectory. The demo is available as an open-source software package on GitHub and a video is available on YouTube [3, 4].

This introduction first considers the state of the art in drone localization, control and navigation. Next it presents in detail the design problems to which this thesis proposes solutions and the challenges that are faced when constructing a demo on autonomously flying drones. It concludes with the outline of the text.

1.1 State of the art

1.1.1 Localization

Indoor localization of drones relies on at least one of three following principles: on-board camera vision based, external camera vision based or beacon based, possibly combined with inertial measurement unit (IMU) data. This section introduces existing systems in all three categories from which the applied system in this thesis is selected. The selection itself is elaborated in Chapter 2. Also a hybrid solution can be implemented, which combines measurements of multiple systems. In order to improve the state estimate obtained through the measurements, a state estimator can be used to combine the information in the measurements with a model of the drone.

Measurement systems

On-board camera position tracking of the drone by using computer vision algorithms is a first option to solve the localization problem. Either the images are processed on-board as in [5] or the drone sends out a video feed to process the images off-board as done by [6]. It is difficult to recognize and track arbitrary objects based on camera images. A smart solution to make visual recognition easier is by using simple tags, such as Apriltags [7]. These tags, resembling QR-codes, are rapidly recognized in an image, and based on the projection of the tag on the image plane, the pose of the camera with respect to the tag can be estimated. The APRIL Robotics Laboratory at the University of Michigan developed a nice application using these Apriltags to track the pose of a box at runtime [8, 9], as illustrated in Figure 1.1.

In the category of external camera setups, a similar technique as just described is applicable: tracking Apriltags or other types of tags attached to the drone using external camera's attached to the ceiling. The number of cameras, the spacing between them and the angular coverage of the lens determine the height up to which this system can operate successfully [11]. The detection of tags with external camera's falls under the denominator of passive camera systems.

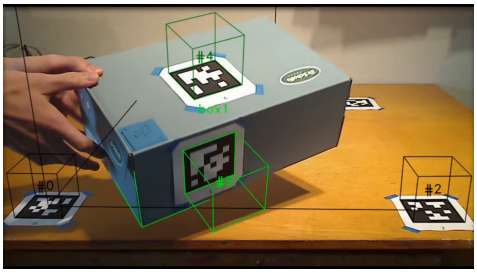


Figure 1.1: Demo of Apriltag localization [9].

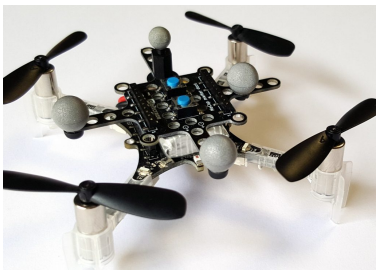


Figure 1.2: Reflective markers used for localization in the Vicon system depicted by the grey balls [10].

The other subcategory of external camera setups is that of the active camera systems. These systems emit light and capture the reflection on reflective markers placed on the tracked object as shown by Figure 1.2. An example of such a system is the Vicon motion capture system, as used by [12].

Next to on- and off-board computer vision based solutions, beacon based localization systems are available. They consist of external modules, one of which is mounted on the drone. One or several other modules are placed around the operating area. The latter serve as points of reference to which the former determines its relative position in case of an intelligent vehicle module. Alternatively the surrounding system determines the position of a passive drone module. A calibration prior to using the system allows to convert the relative to an absolute position measurement. These beacons can use different underlying physical principles to communicate with each other such as ultrasonic or infrared beams.

An example of such a system is the HTC Vive virtual reality gaming system. Tracking of the beacons is done using Valve's lighthouse technology. Two base stations periodically emit infrared light beams that sweep the room. A series of sensors on the tracked object register the time at which the light beam sweeps over them. Through triangulation and combination of this information with measurements of a built in inertial measurement unit (IMU), the position and orientation of the object with respect to these base stations is determined. Either HTC's trackers are used, or it is also possible to build a custom tracker [13].

State estimation

If only raw data measurements would be used to obtain a state estimate of the drone, measurement noise would have a direct influence on the computed control actions. The use of a low pass filter can aid in diminishing this effect by only letting frequencies pass that adhere to the dynamic capability of the system. A Kalman filter is a very commonly used algorithm for such a type of filter that uses a model based approach to measurement filtering. Internally it combines a model of the drone dynamics

with measurements to obtain an accurate state estimate. Moreover the model based approach allows to estimate states for which no measurements are available. When the measurements of the localization system are generated at a different rate than the one at which the controller requests state estimates, an asynchronous version of the Kalman filter (AKF) is required. This is common practice to a much lesser extent than the synchronous version. Therefore this thesis develops and implements a custom AKF algorithm. By considering variable prediction and correction time intervals, a better state estimate can be acquired than would be the case with a standard, synchronous Kalman filter [14, 15].

1.1.2 Control

The stabilization and attitude control of quadcopters is a problem that has been thoroughly investigated for more than a decade. The modeling and control can be seen as a mature technology [16]. Therefore this thesis only considers higher level control, at the level of velocity and position control. The combination of feedback and feedforward control for trajectory tracking is explored, as the addition of feedforward control shows high potential for improving tracking performance [17].

In order to facilitate a set of different tasks in the demo, each task can rely on a dedicated controller, or a combination of controllers, eg. only feedback control or a combination between feedback and feedforward control. This type of state-dependent hybrid control has already been used in similar projects, for example by switching between time-based and event-based controllers during execution of a task [12].

1.1.3 Navigation

Navigating efficiently and collision-free from a starting point to a desired goal in an uncertain environment requires the use of a motion planner. The uncertainty exists in: the modeled dynamics; the accuracy of tracking; obstacle sizes, locations and velocities; environmental disturbances such as wind (very limited in an indoor setting) and turbulence; pose information [18].

A first category of planners solves the motion planning problem by formulating it as an optimal control problem (OCP). To cope with the uncertainties described above, the OCP can be solved with a receding horizon approach, known as Model Predictive Control (MPC). The Optimal Motion Generation tools (OMG-tools) designed by MECO is such a motion planner. This is the motion planner used in this thesis. Its possibilities have been experimentally validated on automated guided vehicles (AGV's) and machine tools in [19, 20], but never on UAV's. Two other OCP-based open source motion planners are discussed next and the possible benefits of using OMG-tools are described.

The first planner uses a method known as the Dynamic Window Approach (DWA) to collision avoidance. It searches for a set of admissible vehicle velocities that can be reached in a short time interval taking into account the vehicle dynamics. A velocity is considered admissible whenever it allows the vehicle to brake before hitting the

nearest obstacle present along the trajectory generated by that velocity. In order to determine this dynamic window of velocities, an adequate dynamic representation of the vehicle is required. The method then picks from this admissible set the velocity that minimizes a given objective function. This objective function makes a trade-off between three terms; one representing the distance to the goal, another the clearance to the nearest obstacle on the trajectory, and a third the magnitude of the velocity. The working principle is illustrated in Figure 1.3. Advantages to using this method are that it allows for fast computations while simultaneously taking into account the kinematics and dynamics of the vehicle. A downside to this method however is the tuning of a set of parameters, which largely depend on the situation [21, 22]. Another downside is that it looks only over a very short time horizon in the future.

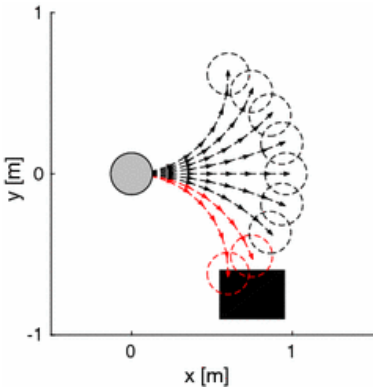


Figure 1.3: Illustration of the Dynamic Window Approach algorithm for obstacle avoidance [23]. Black arrows form the admissible set of inputs, red arrows would cause collision and are not added to the set.

The second planner uses so-called Timed Elastic Bands (TEB) to calculate a time optimal trajectory. It does so by constructing a trajectory of poses where each pose is assigned a timestamp, depending on the kinematic constraints. This trajectory is then optimized by minimizing an objective function penalized with all considered constraints such as velocity and collision avoidance constraints. The optimization problem is recalculated at every time instant, including an updated vehicle state and environment. Even more than the DWA method, this method needs extensive tuning before it can be used successfully. It also does not allow a trajectory to change drastically once an initial guess has been calculated. For example when an obstacle is passed on one side, this cannot be changed, even if it becomes more optimal to pass it along the other side at a certain point in time [24].

The MECO team developed OMG-tools to tackle some of these previously discussed issues. It once again solves an optimization problem, but this time the trajectories are formulated using B-spline basis functions. Manipulation of these splines allows to introduce the desired constraints. These trajectories are then optimized by minimizing either the time required to reach the goal, or the distance to the goal over a certain time horizon. Obstacles are modeled as simple geometric

shapes such as circles and rectangles in 2d or spheres and cuboids in 3d to make the OCP tractable. One of the main benefits to OMG-tools is its ease of use. It requires relatively little tuning in order to work properly in a broad range of scenarios [25, 20]. In contrast to the two previous approaches, OMG-tools does allow the user to provide a predicted motion trajectory of a moving obstacle. This allows the vehicle to handle dynamic situations in a more efficient manner, since it can at any moment predict the best way to pass an obstacle an adapt the path accordingly.

Another important category of motion planners mentioned by [18] is that of the sampling-based randomized motion planning techniques, such as Rapidly-exploring Random Trees (RRT). Figure 1.4 shows an example of a motion planning problem solved with RRT. These techniques work very well in high-dimensional configuration spaces, but need some extensions to cope with environmental uncertainty. RRT is quite widespread and successfully applied in driverless car competitions such as the MIT DARPA Urban Challenge vehicle [26, 27].

Also specifically for drones there are initiatives to stimulate the development of autonomous navigation algorithms, for example the IROS autonomous drone racing challenge [28]. Different strategies are used to complete the obstacle course as quick as possible; some drones solely rely on monocular vision [29] while others combine prior knowledge of the track in combination with vision and deep neural networks to navigate successfully [30].

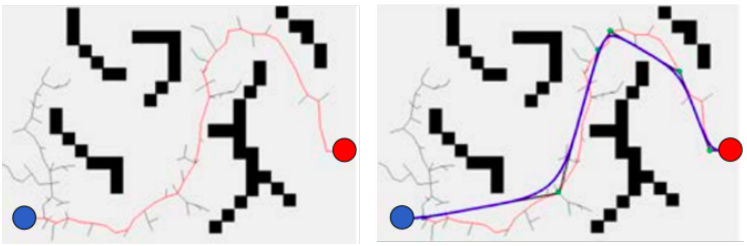


Figure 1.4: Illustration of Rapidly-exploring Random Trees (RRT) algorithm for motion planning [31].

1.2 Problem statement and challenges

It is important that the demo appeals to all types of audience, in the sense that it provides value to both researchers and a regular, non-informed audience. Making the demo visually impressive as well as adding an interactive aspect aids in accomplishing this goal. Aside from these qualitative specifications, the demo is bound to a space restriction. The provided indoor flight area where the demo takes place is a room with dimensions 5.5m x 3.6m x 2.5m (taking into account a safety margin). The UAV can be any quadrotor drone suitable for indoor flight in the given room which is displayed in Figure 1.5.

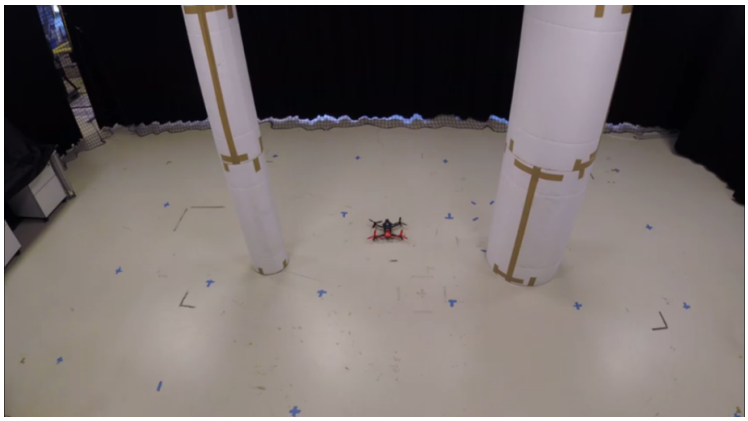


Figure 1.5: The provided flight area in the Robotics lab.

Five different aspects have to be elaborated in this thesis: modeling, localization, control, navigation and the combination of these aspects to make them function together in a demo. A model of the drone dynamics must be derived and identified to provide the basis for model based localization and control. Existing localization technology must be fitted into the demo setup, with the possibility for custom recalibration. A position and velocity state estimator is required to provide the controller with state estimates at arbitrary time instances, and to combine the measurements with model information. Position and velocity controller design is needed in order to track given trajectories, for example those autonomously calculated by the motion planner. Low level attitude control is, as said before, not a topic of interest in this thesis. For navigation the motion planner OMG-tools must be integrated into the practical setup. Control and navigation is only investigated for one single drone, not for cooperative drones or drone swarms. The automatic detection of obstacles, for example computer vision based, is also not considered in this thesis. All the mentioned aspects need to be combined into a demo, taking into account safety and robustness considerations.

The practical implementation of the demo faces a number of challenges. Firstly, unlike AGV's that drive around on solid ground, UAV's move in thin air. This causes issues of drift and turbulence. To deal with these issues, this thesis decides not to try and model nonlinear effects, but rather design the controllers to deal with these disturbances. The current implementation thus works with a constant model, without going into learning control. Feedforward control accounts for desired behavior that falls under the simple model, whereas feedback control copes with deviations from the ideal simple model. The current implementation works with a constant model, without going into learning control.

Next, solving motion planning problems at runtime as an OCP provides quite a computational challenge. Therefore it is of utmost importance that both the obstacles and drone are modelled as simply as possible without losing valuable information. The different obstacle types in OMG-tools are investigated and their

influence on computation time quantified. The obstacles with lowest computation cost are selected for the demo. Also many research studies work towards running computations embedded on the drone hardware itself. This is difficult in practice however since the on-board computational power is limited. Even when performing the computations off-board, most motion planners still require quite some time to solve the optimization problem at each iteration. One way to reduce computation time is to not try and solve the problem to optimality up to a very low tolerance, but to solve until a satisfactory outcome is obtained. In practice it is more useful to receive an updated quasi optimal trajectory with higher tolerance but at a higher rate, than to receive an optimal trajectory at a rate far too low to cope with any disturbances present.

A challenge related to the combination aspect is that even if all modules individually work as desired, it is not guaranteed that the ensemble will function in a satisfactory manner. Especially timing is a complicating factor. Therefore, this thesis spends a lot of effort and attention to timing issues. The design of an asynchronous Kalman filter enables the interaction of the position measurements, model and controller while accurately holding track of timing information.

Another challenge related to the demo aspect itself is that different types of tasks need different controllers (or the same controller with different parameters) and some tasks even require multiple controllers on their own. Also changing from one task to another should happen flawlessly at any given time, without having to interrupt the demo. Implementing the demo as a Finite State Machine (FSM) facilitates adaptivity to the specific situation.

Finally two important features are safety and robustness. Any spectator should be able to interact, without causing misbehavior or without compromising safety. Also when non-nominal situations occur, like the loss of measurements or communication, the demo must detect this deviation from nominal operation and react in a conservative, safe way. Monitoring is an essential function in the detection of non-nominal behavior and is therefore integrated in the FSM implementation.

1.3 Text outline

Chapter 2 first discusses the choice of drone and localization system, followed by an overview of the used software and the implementation structure of the demo. In Chapter 3 the modeling of the drone as a linear holonomic system is explained. Chapter 4 describes both the integration of the measurement system in the code structure to obtain global pose measurements, and the subsequent state estimation using an asynchronous Kalman filter. The part on navigation with the functioning of OMG-tools is discussed in Chapter 5. Chapter 6 then focuses on the implementation of the position and velocity control. The integration of all the previous modules into a functional and entertaining demo is then presented in Chapter 7 by first describing all tasks separately followed by a description of the actual demo.

Chapter 2

Materials & methodology

This chapter first discusses the selection of the hardware used to construct the demo. The hardware items that must be selected are the drone itself and the localization system. Viable alternatives with their specifications are proposed and a value analysis is carried out resulting in the choice of hardware used in this thesis. Mounting of the chosen hardware onto the drone is shortly mentioned next. This is followed by a description of the framework in which the software is written, together with a summary of the used software packages. The chapter concludes with an overview of the implementation approach of the different modules into a functional demo.

2.1 Hardware selection

The requirements for the combination of drone and localization system are as follows. Due to the limited size of the flight area and the need for dynamic flying when avoiding obstacles, a small, agile drone is required. It must either provide sufficient on-board computational power or alternatively be able to communicate with an off-board computer with sufficiently low latency. The localization system ought to be compatible with the drone and with the centralized computer system. This compatibility concerns the software and communication medium as well as the possible weight of an external module placed onto the drone. Furthermore a sufficient update rate of location measurements is required, taking into account that multiple tracked objects might use the same localization system.

Extra advantages for both hardware items are a low cost and easy interface. For the drone a longer flight time is a surplus. The localization system is more attractive when providing 3D position or 6D pose measurements compared to 2D measurements. It is also beneficial if it can function even with an obstructed line of sight between the tracked object and some reference in the environment. The need for line of sight restricts the number and size of physical obstacles that can be placed. The required working volume could be added to this list of requirements, but all systems discussed here satisfy the minimum working volume that was specified in the problem statement of Chapter 1. Therefore it is not a suitable point of comparison.

The demo is designed to run on a standard laptop with no special computational capacity or graphics card. Therefore computer selection is not treated.

2.1.1 Localization system candidates

Table 2.1 shows the localization systems that this thesis considers. Different types of localization are discussed, both on-board as well as off-board systems. In the introduction the Vicon system was described as a possible localization system candidate. Without going into detail on the actual price of the system, this system is ruled out because the price is multiple orders of magnitude larger than that of the other systems discussed in this text. The accuracy of the other systems is good enough for the desired application, such that the large cost for the more expensive system is not justifiable. As mentioned before, systems with high accuracy and a 3D or 6D pose estimate are preferred. A higher update rate also benefits the tracking accuracy of the drone. The main features for each type of localization system are discussed below.

System	**HTC Vive**	**Games On Track**	**Marvelmind**
Price(€)	(-/0) 720/400	(-) 600+	(0) 400
Update rate (Hz)	(+) >60	(-) 20	(0) <45
Accuracy (mm)	(+) 10	(+) 10	(+) 20
Line of sight	(0) yes	(0) yes	(0) yes
Pose estimate	(+) 6D	(0) 3D	(-) 2D
Module weight (g)	(-) 89	(0) 10	(-) 59
Ease of use	(0)	(0)	(+)
System	**Pozyx**	**Apriltag**	**Overhead camera's**
Price(€)	(-) 599	(+) 0	(+) 0
Update rate (Hz)	(+) 80	(0) 10-50	(0) 10-50
Accuracy (mm)	(-) 100	(0) ?	(0) ?
Line of sight	(+) no	(0) yes	(0) yes
Pose estimate	(+) 6D	(+) 6D	(+) 6D
Module weight (g)	(0) 12 + 25	(+) 0	(+) 0
Ease of use	(+)	(0)	(-)

Table 2.1: Candidate localization systems with their specifications and relative scores [32, 33, 34, 35, 36].

The external, beacon based systems are the HTC Vive, Games On Track, Marvelmind and Pozyx. The *HTC Vive* as mentioned in the introduction is a virtual reality gaming system. Its high accuracy, superior update rate (up to 100 Hz even in case of multiple tracked objects) and the ability to supply a 6D pose measurement are the most significant advantages. The cost of €720 mentioned in Table 2.1 is the one for the entire system with base stations, headset, two controllers and an extra tracker. to provide a fair comparison with the other systems that don't provide multiple tracked beacons, only the strictly necessary components (base stations and a single tracker) are considered. This lowers the cost to €400. Nevertheless the Vive controllers are very useful for demo purposes. Compared to most other systems the drawback of the Vive is its relatively large weight, since a drone has to be able to carry the tracker in addition to its own weight. For most drones this will not drastically deteriorate the dynamic behaviour, but the flight time is expected to decrease significantly. This however poses less of an issue when the demo allows the flexibility to interrupt for battery replacement.

Games On Track and *Marvelmind* both work with beacons emitting ultrasound. Their update rates (20 Hz and 45 Hz respectively, to be divided by the number of tracked objects) are significantly lower than that of the Vive, without delivering the advantage of a lower price. Another major disadvantage is that neither can provide a 6D pose measurement, since the Games On Track system is restricted to 3D and the Marvelmind system even to 2D position measurements. Their modules do however weigh less than that of the vive, especially in the case of the Games On Track system.

Pozyx calculates the position and orientation of a tag attached to the drone trough the use of a wireless radio technology called ultra-wideband. The largest advantage to this technology is that it is the only system discussed, which does not require line of sight for the system to function properly. The accuracy however is lower than that of most other systems thus making it less suitable in a small environment. Furthermore it does not provide orientation info unless multiple antennas are used.

The last two table entries *Apriltag* and *Overhead camera's* use computer vision based solutions to provide a pose estimate. The Apriltag solution refers to the situation in which tags are spread out over the environment in which the drone flies. The on-board camera is used to scan the images before either processing them on-board, or sending the footage off-board to an external computer. The research report for an application developed at the University of Michigan states tag detection times of around 22 ms for a 640 x 480 image [36]. However processing these images off-board, if the on-board computational power is lacking, introduces extra latency as an unwanted result. An Overhead camera system uses the opposite approach tracking a single tag attached to the drone using camera's attached to the ceiling. The largest downside in the overhead configuration available at the department is that the available software can only provide 2D measurements. 6D measurements can be obtained but this software is not available and the preference goes to ready-to-use systems.

Systems using tag tracking also do not inherently provide info based on multiple sensors, as is the case in for example the Vive which combines IMU and infrared data. Another downside to such passive systems is the influence of lighting conditions on the performance, which makes them less robust. Robustness is however important since any loss of pose info would result in a fail of the demo.

The selected localization system is the HTC Vive, as will explained further in Section 2.1.3.

2.1.2 Drone candidates

The specifications of the candidate drones are displayed in Table 2.2. Because the demo takes place in a limited space as mentioned before, the drone is preferably small and lightweight. Moreover, lightweight drones generally suffer less damage in case of of a crash. For these reasons the DJI Phantom and drones of similar size and weight are immediately eliminated from the discussion. The Parrot AR drone, commonly used in academic drone projects, e.g. [37], is left out of the discussion, since the Parrot Bebop 2 can be seen as its improved successor. The main features for each drone are discussed below.

Drone	**Parrot Bebop 2**	**Intel Aero**
Price(€)	(+) ~400 / 0	(-) 1000
Frame size (mm) / Weight (g)	(0) 290 / 511	(-) 360 / 865
Camera	(0) 1080p, 30fps	(0) 1080p
Computational power	(-) dual-core Parrot P7	(+) Intel Atom x7
Flight time (min)	(+) 20-25	(+) 20-25
Open source	(+) yes	(+) yes
Drone	**DJI Mavic**	**DJI Spark**
Price(€)	(-) ~1000	(+) ~450
Frame size (mm) / Weight (g)	(-) 335 / 743	(+) 170 / 300
Camera	(0) 4K, 30fps	(0) 1080p, 30fps
Computational power	(0) Myriad 2 MA2155	(-) Myriad 2 MA2450
Flight time (min)	(0) ~20	(-) ~15
Open source	(+) yes	(-) no

Table 2.2: Candidate drones with their specifications and relative scores [38, 39, 40].

The *Parrot Bebop 2* has as a first advantage that there is one available in the department, such that the purchase price drops to zero. Next to that, its ease of communication with a PC makes it very attractive, thanks to the open source interface created by *Autonomy Lab* called *Bebop Autonomy* [41]. Finally it is a visually attractive drone, which is a nice added bonus when used in a demo. A possible disadvantage is the lower video quality than e.g. the DJI Mavic, but since

the goal is not to produce high quality footage this drawback becomes irrelevant. The camera of the Bebop as well as those of the other drones suffice for the localization systems that use an on-board camera.

The *Intel Aero*, a more expensive and visually less attractive drone, is aimed more towards developers than the Bebop. It comes with an open source interface as well. Its superior computational power makes it more suitable for on-board calculations. An MIT project implementing full on-board state estimation and control of an autonomous flying plane proves the potential since the project uses an Intel Atom X7 processor, the same processor as on the Intel Aero [42]. It is however the largest an most heavy of the four drones under comparison.

The *DJI Mavic*'s superior camera quality is not an advantage in this thesis, since the image resolution is unnecessarily high for localization purposes. It does however influence the price in a negative way. The other DJI drone called the *DJI Spark* is the smallest drone making it suited for flying in tight, indoor spaces. But this is advantage is outweighed by the downside of it being more commercial and rather inaccessible compared to the previous three drones.

The Parrot Bebop 2 is chosen to be the most suitable drone for this demo, as will be clarified in the following section.

2.1.3 Value analysis & final selection

The drone choice partially depends on the choice of localization system and vice versa. A smaller drone for example will not have the necessary computational power to perform localization on-board, thus it will have to rely on an off-board type of localization. A localization system with a heavier tracker module will on the other hand require a larger drone to be able to carry the additional weight. First a score is computed for all localization systems and drones to find the most suitable candidates, after which the combination is evaluated to determine compatibility between the two.

Localization system

The indications (+),(0) and (-) in Tables 2.1 and 2.2 are relative scores. This means that a negative score does not necessarily indicate that this alternative is not suited for the application. In order to calculate a relative score for each system, the (+),(0) and (-) scores are treated as 1, 0 and -1. Update rate, accuracy and pose estimate are weighted double, since they are of large interest for the demo. After summation the HTC Vive receives a relative score of 5, Games On Track scores -1, Marvelmind 0, Pozyx 3, Apriltags 4 and the Overhead camera system 3. Based on this comparison, the HTC Vive turns out to be the most suited localization system.

Drone

When choosing a drone, the most important quality is compactness, since the demo will take place in an indoor room with limited dimensions. The larger the drone, the fewer the possible tasks that can be explored in the demo. Therefore this criterion is weighted double. Computing the scores in the same way as for the localization systems then gives following results: the Parrot Bebop 2 receives a score of 2, Intel Aero 0, and the DJI Mavic and Spark a score of -2 and 0 respectively. Therefore the Parrot Bebop is the preferred drone choice.

Final choice

As mentioned before the choice of localization system has an important influence on the choice of drone. The relatively large weight of the Vive Tracker (89 g) for example excludes the DJI Spark (300 g) since it is not able to carry such a large mass. The Parrot Bebop 2 is the second smallest drone, but it is still capable of flying with the added mass of the Vive tracker. The Vive is also a system which makes use of an external computer through which it connects, making it easier to do most of the computations off-board. This renders large on-board computational power unnecessary which makes the combination of Vive and Bebop even more fit since the Bebop does not possess such high computational power. An added benefit of using choosing the Bebop is that it offers an easy to use interface. The above considerations confirm the choice of the Parrot Bebop 2 in combination with the HTC Vive, which is therefore also the hardware used in this thesis. Figure 2.1 shows the combination of the selected drone and tracker from the Vive localization system.

Figure 2.1: The selected drone with tracker from the Vive localization system mounted on top.

2.1.4 Tracker mounting

Before the chosen drone and tracker can be used, the tracker has to be mounted onto the drone. Special care has to be taken when mounting the tracker, since after purchase a problem became apparent: once the drone takes off, all received measurements reduce to zero values. The cause is the following. As mentioned

before, the tracker combines optical measurements with IMU data. The sensitivity of the IMU to vibrations renders the data useless whenever the vibrational amplitude becomes too high. Therefore the main hurdle to overcome when using this system on a drone is to isolate the tracker mechanically from the vibration source.

An ad hoc solution is used here. Two different structures are explored for mounting the tracker to the drone. In the first, the tracker is mounted on top of the drone; in the second one, it is mounted at the rear as shown in Figure 2.2. Both mounting options provide integrated vibration isolation as well as damping through the use of small rubber bellows and polyurethane foam. Mounting the tracker on top results in the least amount of false measurements and moreover the balance of the drone is disturbed less. Hence this is the setup used in this thesis.

Figure 2.2: Two alternatives to tracker mounting: on top or at the rear.

2.2 Software dependencies

As mentioned in the introduction, not all software parts used in this project are self-written. Several software libraries are available to facilitate the development of parts of the demo. All code for execution of the demo is written in the programming language Python. Modeling, controller design and data analysis is performed in Matlab. This section gives a rundown of the most important external software used.

First of all a software framework is needed, which allows different parts of code to communicate and function in parallel. Robot Operating System (ROS) is chosen since it consists of such a modular structure [43]. ROS offers the ability to simultaneously run multiple pieces of code corresponding to multiple functionalities, while providing two-way communication between them. Moreover it allows for easy integration of distributed systems over Wi-Fi and the ROS-community is a vast supplier of plug-and play implementations of interfaces with UAV's and localization systems. Previous experience with similar projects supports this choice since it has proven to work well.

Within this framework three different parts of the demo make use of external software packages. The communication between software and drone hardware relies on a package called *Bebop Autonomy*. It provides an interface between the software running on the computer and the drone itself over a Wi-Fi connection. In addition a range of parameters allows the user to fine-tune the flying behavior as desired [41].

Since this is a demo commissioned by MECO, the navigational part relies on the OMG-toolbox as it is developed in-house. The demo can therefore serve as a tool to show on the one hand the potential of this software while exploring its limitations on the other hand. This is elaborated further in Chapter 5.

The third and last part that requires additional software is the localization. In order to extract pose measurements from the HTC Vive, the open-source package Triad OpenVR [44] is used. It serves as a wrapper for the PyopenVR package which in turn depends on SteamVR, an application developed to use the Vive system for virtual reality gaming [45, 46]. This however only allows to read out the poses of multiple tracked objects. Since two additional controllers are used as well, an extra piece of code provides the ability to read out the controller buttons [47]. This way the demo can to a great extent be carried out without having to sit behind a computer, thereby benefiting its interactive aspect.

2.3 Implementation approach

This section introduces the implementation structure of the demo. It starts from a functional decomposition and gives an overview of the main functionality of all the parts, together with the interfacing between these parts. It concludes with a note on how this structure is implemented in software.

Figure 2.3 schematically represents the different functions: perception, world model, control, monitoring and navigation (motion planner).

The *perception* is the link between the measurement hardware and the control system. It involves reading in raw data and representing it with respect to a known, calibrated world reference frame. The *world model* contains three different parts. The first one is a mathematical model of the drone's dynamics. Next to this it includes all details about the environment, both the placement of obstacles as well as the location of all room edges. And thirdly it contains a state estimator to compute and store a pose and velocity estimate of the drone based on the combination of measurements coming from the perception, the inputs coming from the controller and the drone model stored in the world model itself. The perception and the estimator together form the localization. The main goal of a centralized world model is to make sure that all components in the system use the same information at all times. They all contribute to this single representation instead of each relying on a separate and different version of world information.

The *monitor* is vital to ensure safe and correct execution of the demo. It detects non-nominal situations, as well as nominal events that ask for a change in behavior of the system in order to perform a desired task. In case of each of these events, it

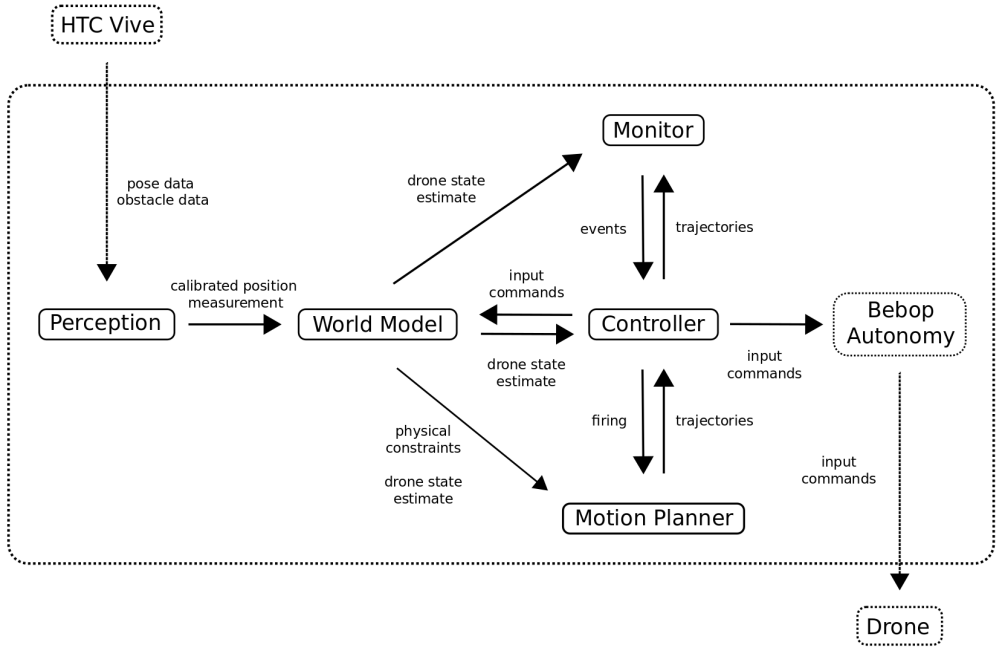


Figure 2.3: Implementation structure based on functional decomposition. The large dotted frame indicates the distinction between software (inside) and hardware (outside). Terms inside full frames are components developed in this thesis. Terms inside dotted frames are adopted components. Arrows indicate information flow.

triggers the rest of the system to adapt its behavior. The *controller* is responsible for all actions that generate input directly to the drone. It computes input commands based on drone state information and desired behavior as defined by the task. The *motion planner* is in charge of the navigation. It generates trajectories from the drone's position to a desired goal given physical limitations, such as drone dynamics, obstacles and flight area size. It is inactive during standby or tasks where only lower level control is required.

Characteristic to the implementation of the demo in this thesis, is the approach to cope with situation specific behavior of the control system. The demo provides five different tasks, each meant to illustrate a different level of autonomous flight. Tasks are divided into states, where every state uniquely defines the behavior of each component in the entire system. The state-specific behavior for each functional component is embodied by a Finite State Machine (FSM). The FSM presents itself implicitly in the structure depicted by Figure 2.3, since the monitor is the component that triggers necessary state transitions in the FSM. Other components adapt their behavior according to the current state. This structure also provides the ability to switch between tasks and states at any given moment which is valuable in emergency situations. Details on the specific tasks will be discussed in Chapter 7.

In the software implementation, the functional decomposition adheres to the nodular structure of ROS. The perception is handled by a node called *Vive Localization*. The world model is integrated together with the main functionality of the FSM in the node *Bebop Core*. This node is the heart of the demo and drives the rest of the system. The controller has its own ROS node *Controller*, in order to allow it to run at its own controller update rate different from the rate of other nodes. The motion planner consists of a separate node *Motion Planner* as well, because the computation of trajectories takes a relatively long time. Running it isolated from the other functionalities assures that it does not block other operations. The monitor is integrated into the controller and core nodes.

Chapter 3

Modeling of drone dynamics

The state estimator and autonomous control from Chapters 4 and 6 rely on linear time-invariant (LTI) continuous-time models of the drone's position and velocity response with respect to the control inputs. This chapter elaborates on how these dynamic models are obtained and how the parameters are estimated. The identified models are stored in the world model cf. Section 2.3 allowing other components of the implementation to access them when necessary.

The first section provides a qualitative derivation of the model based on physical insight, resulting in a set of linear time-invariant transfer functions with unknown parameters. In a second section, these unknown parameters are identified by conducting a least squares discrete-time domain identification. The final section concludes with an explanation on how the derived models are then used by the other system components.

3.1 Mathematical model derivation

The nomenclature and orientation of the axes on the Bebop drone are clarified by Figure 3.1. The drone possesses four degrees of freedom: rotation along the x-, y- and z-axis as well as translation along the z-axis. Each of these has a corresponding normalized input ranging from -1 to 1: for the x- and y-direction the inputs are proportional to the roll and pitch angle, whereas for the z-axis the translational and rotational input are proportional to the respective translational and rotational velocity along this axis. The flight control interface designed by the drone manufacturer Parrot is intended to be intuitive for a human navigator; it decouples the inputs in such a way, that the frame indicated in Figure 3.1 in which the inputs are valid, does not roll or pitch along with the drone body. This frame does however rotate along with the yaw of the body. Yaw is further on designated with the symbol θ and its time derivative with ω.

The dynamical model now describes the response of the closed loop system comprising the internal flight controller and drone dynamics to the different inputs. Since the inputs are decoupled in the manner described above, in theory an input

along one axis only influences the output along that same axis. Therefore the models are set up accordingly as single-input single-output (SISO) systems for each direction.

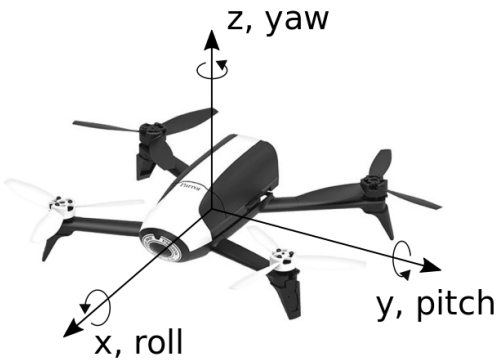


Figure 3.1: The x-, y- and z-axis with the corresponding roll, pitch and yaw angle indicated on a Parrot Bebop 2 drone.

3.1.1 X and Y model

Figure 3.2 schematically shows an idealized 1D model in the Laplace domain (with s as Laplace variable) for both the x- and y-direction. In this model, j is the normalized input command sent to the drone. It is equal to the desired reference angle divided by the maximum attainable angle α_{max}. The applied reference for the pitch and roll angle leads to a linear movement along x and y respectively.

The proposed model originates from the reasoning that the real angle α reaches the desired reference angle after some delay expressed by τ. Next the model assumes that the roll and pitch angles are proportional to the acceleration along the respective axes. Velocity and position then follow through integration.

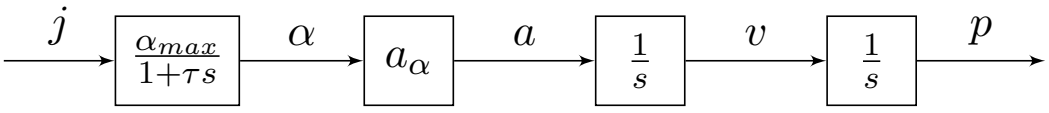


Figure 3.2: Idealized continuous-time model. j represents the normalized input command. α is the pitch or roll angle for an input over x or y respectively. τ is a time delay constant. a, v, p correspond to the acceleration, velocity and position.

Adding damping (which is ignored in the idealized model), the proposed general 1D input-output model from input to velocity becomes:

$$H_{VJ}(s) = \frac{V(s)}{J(s)} = \frac{b_0}{s^2 + a_1 s + a_0} \tag{3.1}$$

The position model follows through integration:

$$H_{PJ}(s) = \frac{P(s)}{J(s)} = \frac{1}{s} H_{VJ}(s) = \frac{b_0}{s^3 + a_1 s^2 + a_0 s} \tag{3.2}$$

3.1.2 Z and θ model

Both for the linear and angular z-direction a transfer function of one order lower than for x and y suffices, since the input for this direction is proportional to velocity instead of acceleration.

The general velocity transfer functions in the z-direction then become:

$$H_{VJ,z}(s) = \frac{V_z(s)}{J_z(s)} = \frac{b_0}{s + a_0} \tag{3.3}$$

$$H_{\Omega J}(s) = \frac{\Omega(s)}{J_\theta(s)} = \frac{b_0}{s + a_0} \tag{3.4}$$

Integrating to obtain the position along the z-axis and yaw angle then gives:

$$H_{PJ,z}(s) = \frac{P_z(s)}{J_z(s)} = \frac{b_0}{s^2 + a_0 s} \tag{3.5}$$

$$H_{\Theta J}(s) = \frac{\Theta(s)}{J_\theta(s)} = \frac{b_0}{s^2 + a_0 s} \tag{3.6}$$

3.2 Least squares discrete-time domain identification

The identification consists in finding the parameters b_0, a_i in equations (3.1) - (3.6) for which the simulated dynamic response best resembles the real drone behavior. The 'best' fit is quantified in terms of the least square error on the difference equation representation of the corresponding transfer function.

In order to perform an identification of the unknown parameters in these transfer functions, the continuous time representations must be discretized first. The 'pole-zero matching' method is selected as discretization scheme, in order to guarantee preservation of the minimum phase property of transfer functions (3.1) - (3.6) [48]. Minimum phase for a linear model means that all continuous-time zeros of the transfer function are located in the left half of the complex plane (real part smaller than zero). For a discrete time transfer function this is equivalent to all zeros lying inside the unit circle. In contrast to a minimum phase SISO model, the step response of a non-minimum phase model first deviates in the opposite direction to the reference. This is illustrated qualitatively in Figure 3.3 for a general minimum phase and a non-minimum phase system. In practice the drone does not show this behavior, such that we want to avoid introducing it into the model artificially. The discrete-time transfer function equivalent to the continuous-time variant in Equation 3.1 is given by

$$H_{VJ}(z) = \frac{b_{0,d}}{z^2 + a_{1,d} z + a_{0,d}} \tag{3.7}$$

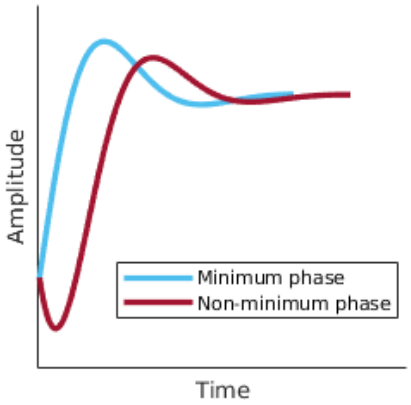


Figure 3.3: Step response of a general minimum phase and a non-minimum phase SISO model. The minimum phase variant sets off in the direction of the reference, while the non-minimum phase variant sets off in the opposite direction.

In a following step, four separate experiments are set up to gather the data required to perform the identification. In each of these experiments, a block pulse is applied to one of the four degrees of freedom x, y, z and θ in order to excite a broad range of frequencies. Figure 3.4 shows the applied block pulse input, the position measurement and the velocity which is obtained as numerical derivative of the position. The length of the pulses is limited by the size of the flight area where the demo is held. The height of the block pulses varies to obtain an average model over a range of inputs. Each experiment is conducted for a duration of ten periods to provide a sufficient amount of data for averaging out unwanted variations.

Both inputs and outputs are low pass filtered with a Butterworth filter before identification in order to eliminate higher frequency noise. The cutoff frequency of the filter is chosen to be five times higher (as a rule of thumb) than the highest frequency contained in the identified system. The crossover frequency (iteratively determined) is taken as a measure for this highest frequency.

Now the discrete time parameters $b_{0,d}$, $a_{i,d}$ can be determined using a least-squares fit based on an ARX model structure [49]. Elaborating this for the discrete-time velocity model in the x-direction given by equation (3.7) gives the following difference equation:

$$\begin{aligned} & b_{0,d}J(z) = z^2V(z) + a_{1,d}z^2V(z) + a_{0,d}V(z) \\ & \Leftrightarrow b_{0,d}j[k] = v[k+2] + a_{1,d}v[k+1] + a_{0,d}v[k] \\ & \Leftrightarrow v[k+2] = -a_{1,d}v[k+1] - a_{0,d}v[k] + b_{0,d}j[k] \\ & \Leftrightarrow v[k] = -a_{1,d}v[k-1] - a_{0,d}v[k-2] + b_{0,d}j[k-2] \end{aligned}$$

where $k = 0, 1, 2, ...$ is the discrete time variable. The last equation can then be rewritten as

$$V = \Phi \cdot \Theta \tag{3.8}$$

with

$$V = \begin{bmatrix} v[2], & v[3], & \dots & v[N] \end{bmatrix}^T$$

$$\Phi = \begin{bmatrix} -v[1] & -v[0] & j[0] \\ -v[2] & -v[1] & j[1] \\ \vdots & \vdots & \vdots \\ -v[N-1] & -v[N-2] & j[N-2] \end{bmatrix}$$

$$\Theta = \begin{bmatrix} a_{1,d} & a_{0,d} & b_{0,d} \end{bmatrix}^T$$

and N the size of the data set. v is the velocity data obtained in the experiment in Figure 3.4.

The unknown parameters are then determined as the least-squares solution for Θ of the over-determined system of equations (3.8). In MATLAB this can easily be done using the backslash command: $\Theta = \Phi \backslash V$. The procedure is fully analogous for the models in the y- and z-directions.

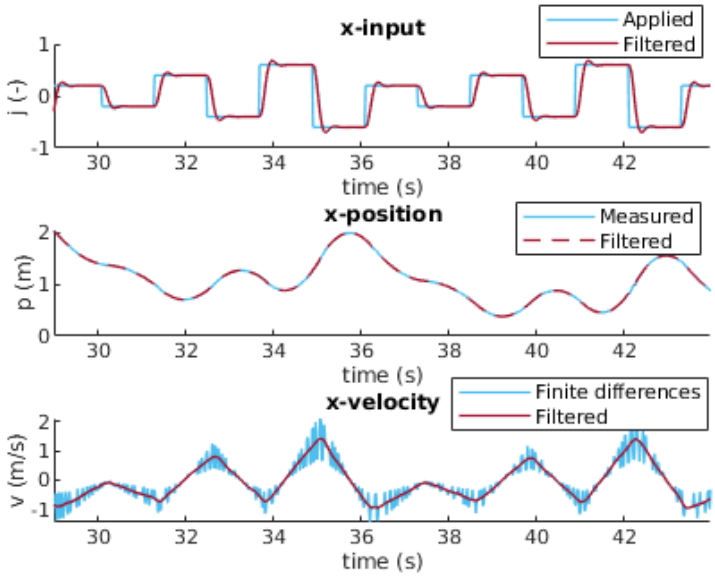
Since the measurement data received from the Vive localization system is sampled at a sampling time interval $T_{s,meas}$, this is also the sampling time used to convert the discretized transfer function into a continuous representation. Numerically the resulting continuous time transfer functions then become (after transforming the discrete-time transfer functions back to the continuous-time domain using the pole-zero matching method):

$$H_{VJ,x}(s) = \frac{22.51}{s^2 + 6.167s + 1.532}\ (m/s) \qquad H_{PJ,x}(s) = \frac{22.51}{s^3 + 6.167s^2 + 1.532s}\ (m)$$

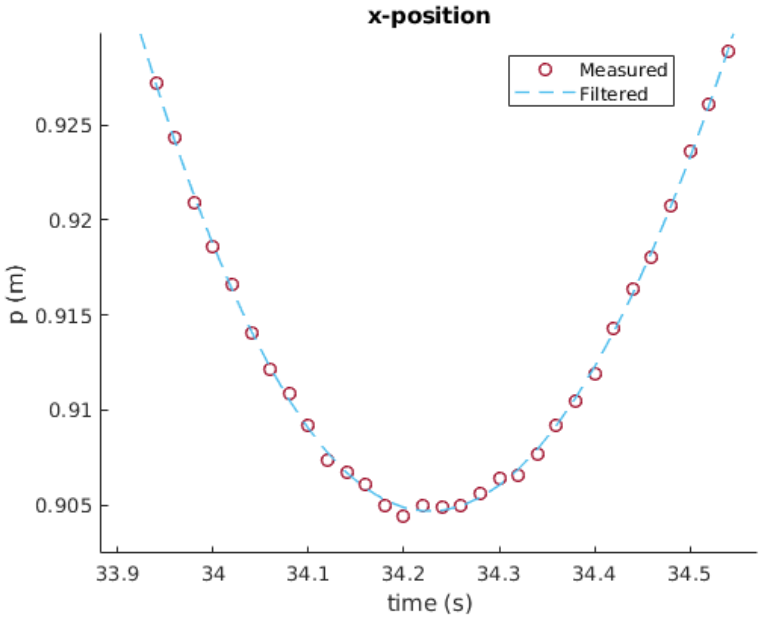
$$H_{VJ,y}(s) = \frac{18.96}{s^2 + 5.147s + 2.116}\ (m/s) \qquad H_{PJ,y}(s) = \frac{18.96}{s^3 + 5.147s^2 + 2.116s}\ (m)$$

$$H_{VJ,z}(s) = \frac{6.066}{s + 6.26}\ (m/s) \qquad H_{PJ,z}(s) = \frac{6.066}{s^2 + 6.26s}\ (m)$$

$$H_{\Omega J}(s) = \frac{5.66}{s + 3.262}\ (rad/s) \qquad H_{\Theta J}(s) = \frac{5.66}{s^2 + 3.262s}\ (rad)$$



(a) Input, position and velocity.



(b) Zoom on the position measurement.

Figure 3.4: Excerpt from the identification experiment for the x-direction. (a) From top to bottom: the input, position response and velocity response (numeric difference) together with their filtered counterpart. (b) Zoom on the position measurement to illustrate the quality of the measurement system.

The quality of the identified models is validated against the measurement data by feeding the model the same input as applied to the drone during the experiment. Figure 3.5 displays this comparison for the x-direction. The velocity model fits the measurement data rather accurately. However, from the position fit it becomes clear that there is a drift effect present which is not accounted for in the linear model. The error on the position grows, since the small velocity error is integrated over time. Nevertheless the model is adequate for the intended use, as it is only extrapolated over very short time intervals (in the order of 10 ms).

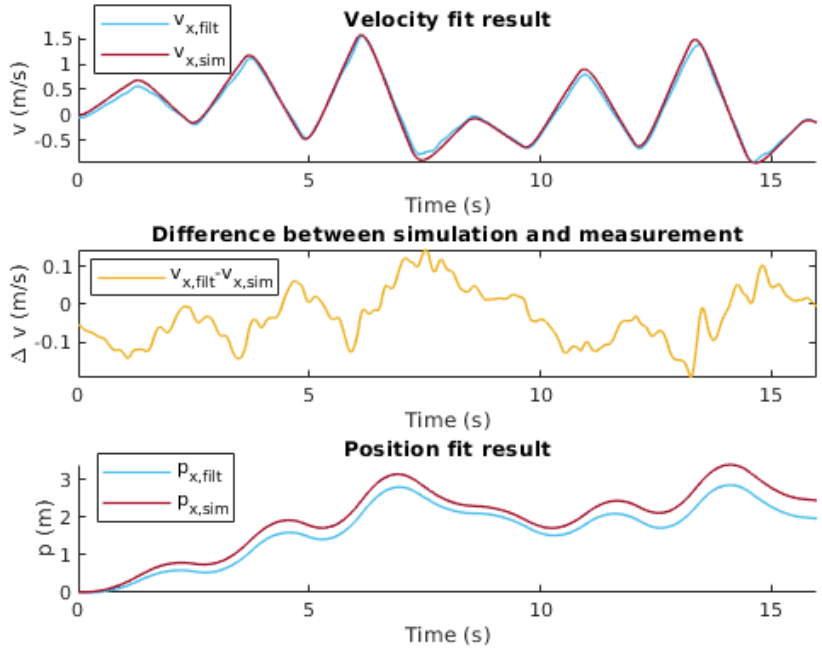


Figure 3.5: Result of the model fit for the x-direction. Comparison between simulation of identified transfer function and experiment measurement data for the same block pulse input signal.

3.3 Further use of the identified models

The state estimator that will be described in the following chapter requires a discretized version of the continuous-time position and velocity models. Before discretization the transfer function representation is converted to a state space representation, according to the controllable canonical form:

$$\begin{aligned}\underbrace{\begin{bmatrix}\dot{x}_1\\ \dot{x}_2\\ \dot{x}_3\end{bmatrix}}_{\dot{x}} &= \underbrace{\begin{bmatrix}0 & 1 & 0\\ 0 & 0 & 1\\ 0 & -a_0 & -a_1\end{bmatrix}}_{A}\underbrace{\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix}}_{x} + \underbrace{\begin{bmatrix}0\\ 0\\ 1\end{bmatrix}}_{B} j\\ \underbrace{\begin{bmatrix}p\\ v\end{bmatrix}}_{y} &= \underbrace{\begin{bmatrix}b_0 & 0 & 0\\ 0 & b_0 & 0\end{bmatrix}}_{C}\begin{bmatrix}x_1\\ x_2\\ x_3\end{bmatrix} + \underbrace{\begin{bmatrix}0\\ 0\end{bmatrix}}_{D} j\end{aligned} \tag{3.9}$$

where p is the position and v the velocity in the x-, y- or z-direction. The discrete-time state space model is then calculated using a Forward Euler discretization scheme:

$$\dot{x}(t) \approx \frac{x(t+T_s) - x(t)}{T_s} \tag{3.10}$$

$$\implies x(t+T_s) \approx \underbrace{(I + T_s A)}_{A_d} x(t) + \underbrace{T_s B}_{B_d} j(t) \tag{3.11}$$

$$C_d = C, \quad D_d = D \tag{3.12}$$

The matrices with subscript 'd' are the state space representation matrices for discrete time. This discretization is however not calculated in advance since the discretization sample time is variable due to the asynchronicity in the control system. It is therefore calculated at every time step, using the variable sample time T_s.

The controller also uses the velocity model to compute the required inputs to reach a desired velocity set point. It therefore has to invert this model, as described in more detail in Chapter 6.

3.4 Conclusion

In this chapter a linear time-invariant continuous-time dynamic model of the drone was derived, together with its discretized counterpart. First a set of general transfer functions with unknown coefficients was defined based on physical insight in the system, and this for each of the four degrees of freedom. Next a linear least squares discrete-time domain identification was conducted to determine these unknown coefficients. Since this is a linear model, nonlinear effects such as drift were not taken into account. The model does however perform considerably well as shown in the following chapters.

Chapter 4

Localization

To control and navigate the drone successfully, the corresponding components require good estimates of the position, orientation and velocity of the drone, i.e. the drone's state. This chapter discusses how these estimates are obtained by combining measurements and drone model information.

First it describes the implementation that obtains position measurements from the HTC Vive. This corresponds to the perception part of the implementation structure proposed in Section 2.3. Coordinate reference frames are defined to ensure that each component refers to a commonly known reference. To initialize the relations between these frames, a calibration procedure is proposed and implemented.

Next an algorithm for a state estimator is proposed which combines these measurements with the model derived in the previous chapter. This algorithm, called an asynchronous Kalman filter (AKF), estimates the states based on the inputs sent by the controller and taking into account the asynchronicity of the system. Even though this project only uses a single localization system, the implementation aims for generality and therefore provides the possibility to use different measurement systems or expand the localization system to combine measurements from multiple measurement systems.

4.1 Perception

As mentioned in Chapter 2, raw position readouts are retrieved using the *Triad OpenVR* Python package, which is built upon the *PyopenVR* library [44, 45]. A minor adaptation with large consequences this thesis makes to *Triad OpenVR* is the use of Python's $atan2()$ function instead of $atan()$ when calculating Euler angles, to enable four quadrant operation.

Five reference frames are defined in order to relate all pose measurements, as shown in Figure 4.2. The *World frame* is the global reference frame to which drone pose estimates, trajectories and obstacle locations are referred. The raw pose readouts are expressed in the *Vive frame*, which is fixed to the Vive's base stations. They represent the pose of the tracker, depicted by the green axes in Figure 4.1, with respect to the Vive frame. The tracker and the drone are assumed to be rigidly

connected, despite minimal deformations of the flexible suspension presented in Chapter 2. Therefore the *Drone frame* represented by the red axes in Figure 4.1 has a fixed rotation and translation with respect to the *Tracker frame*. The addition of the *World yaw frame* is a direct consequence of the model derived in Chapter 3. Recall that drone inputs are valid in a frame that does not pitch or roll along with the body of the drone, but only yaws along with it. The World Yaw frame embodies exactly this. Its origin coincides with that of the World frame.

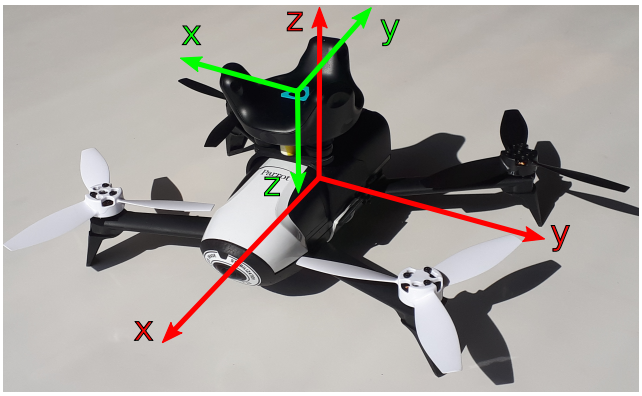


Figure 4.1: Orientation of drone and tracker coordinate frames. The origin of the tracker frame lies in the center of the mounting plane [50]. The origin of the drone frame lies in the geometric center between the four propellers.

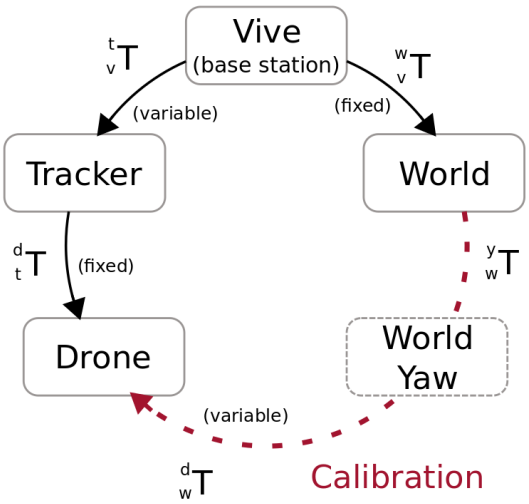


Figure 4.2: Transformation tree with all coordinate reference frames and indication of fixed or variable transforms.

4.1.1 Coordinate transformations in ROS

Practical implementation of coordinate reference frames and the coordinate transformations between them in ROS is handled by the *tf2* library [51]. *tf2* defines coordinate transformations as ROS *geometry_msgs/TransformStamped* messages, meaning they contain the following information:

- time stamp
- child frame and reference frame
- translation (x, y ,z)
- rotation as quaternions (x, y, z, w)

The time stamp of each frame corresponds to the last time at which the new pose of the frame is computed and broadcasted. It ensures that only transforms defined at equal time instances are used. This is the main advantage of using *tf2* over another method such as explicitly storing the transformation matrices and communicating these via messages. For times at which no transform was broadcasted, the module automatically interpolates in time between the known coordinate transformations.

The three last bullets contain information equivalent to the well known homogeneous transformation matrix or pose matrix ${}^{ref}_{child}T$ between a reference frame and a child frame:

$$ {}^{ref}_{child}T = \begin{bmatrix} {}^{ref}_{child}R & {}^{ref}t_{child} \\ 0 & 1 \end{bmatrix} $$

where ${}^{ref}_{child}R$ and ${}^{ref}t_{child}$ are the 3x3 rotation matrix and the translation vector respectively of the child frame expressed in the reference frame. This matrix can be interpreted either as a transformation from the child frame to the reference frame, or alternatively as the pose of the child frame expressed in the reference frame. Its inverse gives the transformation from reference frame to child frame.

4.1.2 Calibration procedure

The fixed transforms indicated in Figure 4.2 are defined as follows. ${}^{d}_{t}T$ follows directly from the tracker mounting on the drone and can be derived from the setup in Figure 4.1.

${}^{w}_{v}T$ is defined by calibration. First the user places the drone at the desired World frame origin location, with the axes of the Drone frame aligned to the desired axes of the World frame. Next the calibration fixes the World frame as the frame that at the instant of calibration coincides with the Drone frame: ${}^{w}_{v}T = {}^{d}_{v}T = {}^{d}_{t}T\ {}^{t}_{v}T$. From that moment on, any pose of the drone with respect to the global reference frame is uniquely defined as:

$$ {}^{d}_{w}T = {}^{d}_{t}T\ {}^{t}_{v}T\ {}^{v}_{w}T $$

4.2 State estimation: asynchronous Kalman filter

The raw measurement data from the HTC Vive is not used directly by the controller to calculate the control inputs. It is first processed by a state estimator to combine the measurements with model information to ensure that a good estimate is available at arbitrary time instances. The state estimator discussed in this section estimates the drone's position and velocity in 3D, as well as the yaw angle. The type of estimator used for this application is the asynchronous Kalman filter. To explain the choice for an asynchronous variant of the Kalman filter, following paragraphs first summarize the more generally used synchronous Kalman filter and then motivate the asynchronous character of the estimator used in this project.

A synchronous Kalman filter consists of two subsequent calculation steps. First there is the *prediction step*. Based on the last state estimate and the current input that is applied to the system, a prediction is made of where the system will be at the next point in time using a model of this system.

Next there is the *correction step*. The estimate predicted in the first step is updated using the measurement of (a linear combination of some of) these states. These measurements can come from a single sensor, or measurements of multiple sensors can be combined. The combination of model and measurements is weighted according to their relative uncertainty. This uncertainty is represented by the measurement and process noise covariance matrices, assuming a Gaussian distribution for the measurement and process noise. The total uncertainty on the estimate is expressed by the estimation error covariance. One way to look at the Kalman filter as an algorithm that minimizes this estimation error covariance [52].

The synchronous approach poses two issues in the context of the modular structure that this thesis implements. Firstly, the estimator must cope with asynchronicity between the controller and the perception part. Measurements are received at a rate different from the rate at which the controller generates inputs. Moreover, the implementation aims for generality in the use of different measurement systems, or in using multiple systems simultaneously. Therefore the rate at which new measurements become available is not known beforehand. Secondly, communication time delay between the ROS nodes must be taken into account when processing the measurement and state data.

To cope with the asynchronicity a synchronous Kalman filter using the latest measurement and input at each iteration data could be used. However this is not such an accurate approach since timing differences between measurement and input are ignored. Also it cannot obtain a state estimate at every arbitrary time instant. The loss of timing information also hinders solving the second issue.

Using an asynchronous version of the Kalman filter allows to solve both these issues. It allows the controller to run at an update rate different from the localization system, by applying prediction steps whenever a new input is available and by applying correction steps whenever a new measurement becomes available. An

additional benefit to this approach is that the current measurement system can be replaced by any other system running at another rate, without affecting its operation. This increases the modularity of the demo and enables the possibility of comparison between measurement systems. This last feature is not further elaborated in this thesis, as the current measurement system suffices for demo purposes. Now a more detailed explanation of the working principle of the asynchronous Kalman filter follows.

4.2.1 General working principles

Since the identified drone model of Chapter 3 is not identical in the x- and y-direction, it is valid only in the World Yaw frame that was introduced in Section 4.1. Therefore this is also the frame in which all asynchronous Kalman filter operations are computed.

The execution of the prediction and correction step do not occur at a synchronous rate, rendering it essential to keep track of the different time instants at which the input and localization data is acquired. Also any delay introduced due to the communication between the different parts of code can be taken into account this way. Therefore each measurement, input and state estimate is assigned a time stamp at the time when it is generated by perception, controller or world model respectively. All time stamps are referred to a time reference which is common to all nodes because of the ROS time primitive.

Figure 4.3 shows a sketch of the interaction between perception, world model and controller along the time axis (oriented downwards). The arrows on the right of the controller timeline denote the times t_i $(i = 1, 2, ...)$ at which the controller sends out input commands to the drone. The arrows coming from perception indicate the times t_{meas} at which a new measurement comes in. The AKF stores the last input that was applied before the latest measurement as well as all inputs from the latest measurement up to the current time.

The two distinct operations that are performed are the same as in the synchronous Kalman filter. One is the *prediction step*, which estimates the state at time t_{i+1} given the input and state at time t_i by using the identified model. The second one is the *correction step*. The latter differs from the synchronous variant in the sense that it involves several prediction steps up to the new measurement before the actual correction step. The reason for this is that in order to execute the correction step, a state estimate must be available at time t_{meas}. In contrast to the synchronous Kalman filter where this is always the case, this is usually not the case in an asynchronous one as the red and blue lines do not coincide.

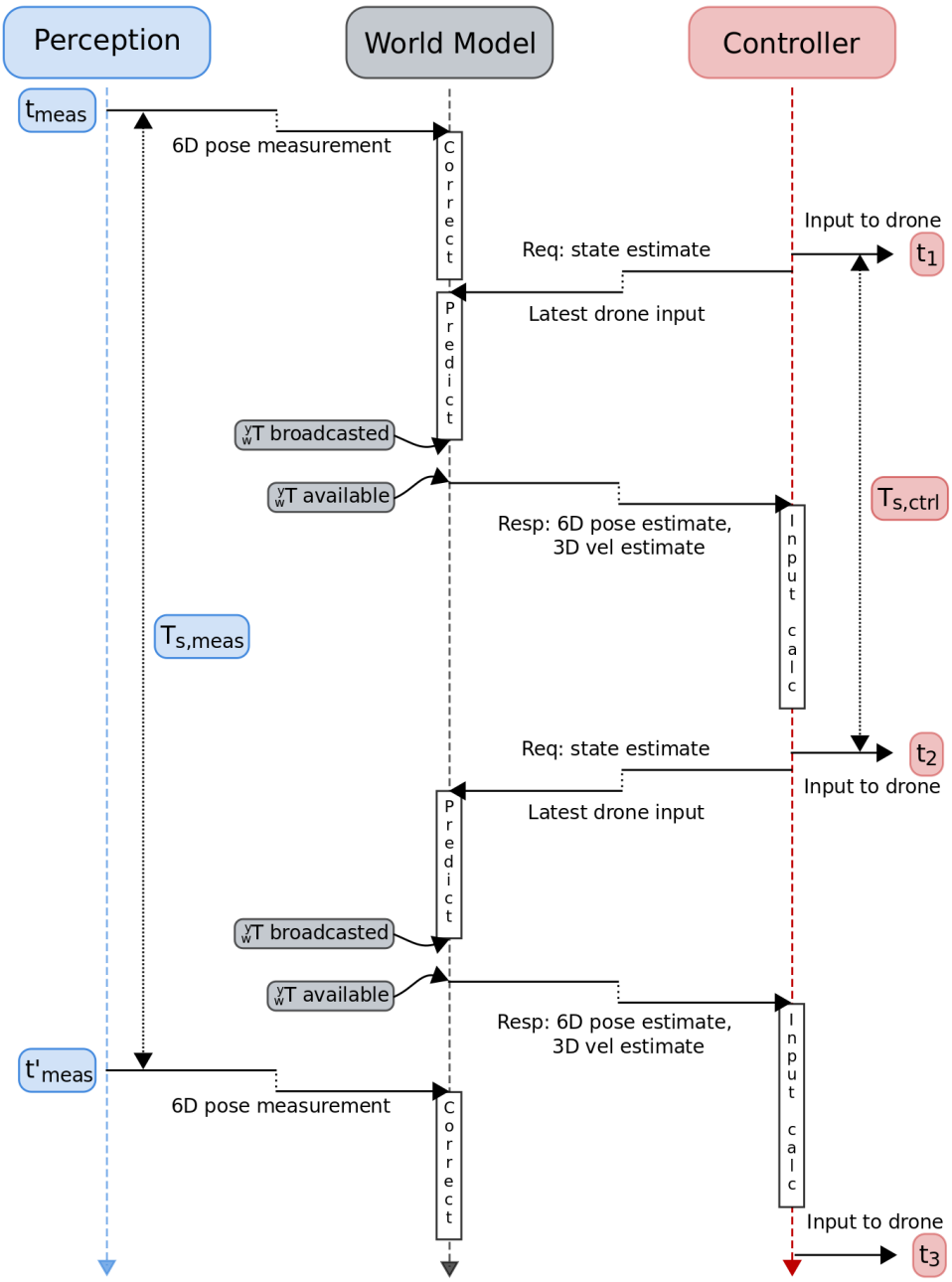


Figure 4.3: Timing diagram of the interaction between perception, world model and controller. Indicated time instances are the times at which measurements (t_{meas}) and drone inputs ($t_i,\ i = 1, 2, ..$) take place, together with their respective sample times ($T_{s,meas},\ T_{s,ctrl}$). All in- and outputs to and from the world model, including communication delay, are represented by the shifted arrows. Calculation times for corrections, predictions and input commands are illustratively shown (not up to scale), as well as transform broadcast delay.

As time goes by, the controller sends out input commands at times t_i, and in return requests a position estimate at time t_{i+1} from the AKF as illustrated in Figure 4.3. Since this time t_{i+1} is at a point in the future, the AKF will perform a prediction step to compute a state estimate for that future time instance. The controller then uses this state estimate to calculate the set of inputs that will be applied at time t_{i+1}. Note that over these predictions, the estimation error covariance grows.

In practice, the time t_{i+1} for which the input command is computed does not exactly coincide with the actual time at which the input command is sent, due to delay and imperfections on the rate at which the code runs. However, at the time of a correction, the actual times t_i at which the input commands were sent *are* known, since they are assigned a time stamp in the controller when applied. Hence at the correction step, the predictions starting from the previous measurement time t_{meas} up to the current measurement time $t_{meas}^{'}$ can be recomputed with more accurate timing information to obtain a better estimate. Note that because of this, the time step over which the prediction is made becomes variable and unique for every prediction step. This motivates the use of a continuous-time model that is discretized at every prediction, as introduced in Section 3.3. After recomputing the predictions, the AKF performs a correction step, thereby shrinking the estimation error covariance again.

After such a cycle of predictions and a correction, the time of the latest measurement $t_{meas}^{'}$ is made the new t_{meas}, and the procedure repeats itself.

The following part first discusses the mathematics of the prediction and correction step. Next it elaborates on deviations from the ideal situation of Figure 4.3. This ideal situation is not always satisfied due to lag in the communication between nodes or because of the loss of a signal. It explains how the implementation proposed in this thesis deals with these cases.

Prediction step

Whenever the controller needs an estimate of the states at the next point in time, it sends out the latest input command along with the request for the estimate. This input is then used to perform a prediction step over the required time step $T_{s,ctrl}$.

Recall from Chapter 3 that the discretized A_d and B_d state space representation matrices are given by the following formulas:

$$\begin{cases} A_d = T_s A + I \\ B_d = T_s B \end{cases} \tag{4.1}$$

with A and B the continuous time state space matrices, I the identity matrix and T_s the time over which the prediction will be performed. The C and D matrices remain unchanged after discretization. Although this is a very simple Euler approximation, the section on the AKF performance further on will prove that the approximation suffices since the linear extrapolation is only made over a short time interval in the order of 0.01 s.

The prediction step is then represented by:

$$\hat{x}_{t_{i+1}|t_i} = A_d \hat{x}_{t_i|t_i} + B_d j_{t_i} \tag{4.2}$$

$$\hat{y}_{t_{i+1}} = C \hat{x}_{t_{i+1}|t_i} + D j_{t_i} \tag{4.3}$$

where $\hat{x}_{t_{i+1}|t_i}$ is the a priori state estimate, j_{t_i} represents the vector of input commands and $\hat{y}_{t_{i+1}}$ is the vector containing the position and velocity estimate.

The a priori update of the error covariance matrix is found as:

$$\hat{P}_{t_{i+1}|t_i} = A_d \hat{P}_{t_i|t_i} A_d^T + Q \tag{4.4}$$

where Q is the process noise covariance matrix, which is assumed to be constant over time.

Correction step

During the correction step, when predictions have been performed up to t'_{meas}, the Kalman filter incorporates the new measurement. The correction is applied using the following formulas:

$$\nu_{t'_{meas}} = z_{t'_{meas}} - C \hat{x}_{t'_{meas}|t_i} \tag{4.5}$$

$$S_{t'_{meas}} = C \hat{P}_{t'_{meas}|t_i} C^T + R \tag{4.6}$$

$$L_{t'_{meas}} = \hat{P}_{t'_{meas}|t_i} C^T S^{-1}_{t'_{meas}} \tag{4.7}$$

$$\hat{x}_{t'_{meas}|t'_{meas}} = \hat{x}_{t'_{meas}|t_i} + L_{t'_{meas}} \nu_{t'_{meas}} \tag{4.8}$$

where $\nu_{t'_{meas}}$ is the innovation, $z_{t'_{meas}}$ is the measurement, $S_{t'_{meas}}$ is the innovation covariance matrix and $L_{t'_{meas}}$ represents the Kalman gain. R is the measurement noise covariance matrix, which is also assumed to be constant over time. $\hat{x}_{t'_{meas}|t'_{meas}}$ finally represents the a posteriori state estimate.

The a posteriori update of the error covariance matrix is then calculated as:

$$\hat{P}_{t'_{meas}|t'_{meas}} = (I - L_{t'_{meas}} C) \hat{P}_{t'_{meas}|t_i} \tag{4.9}$$

The Q and R matrices used in the prediction and correction step can be seen as tuning parameters. The ratio between the elements in the Q matrix on the one hand and the R matrix on the other hand determines the extent to which the state estimate is corrected towards the measurement. This can be seen in formula 4.7 since R has a direct influence on the size of the Kalman gain through the innovation covariance matrix [52].

4.2.2 Timing practicalities

The following part concerns the approach taken to cope with practical timing imperfections of the operating system.

The timeline during nominal operation is depicted in Figure 4.3. Depending on the relative size of the measurement update rate and the controller update rate, more or less inputs are received in between two consecutive measurements. This does however not alter the principle of predictions and corrections.

In practice three deviations from this ideal scheme occur due to communication delay and finite computational time of arithmetic operations. Either a measurement or input command is received 'too late', or a transformation between two coordinate reference frames is already broadcasted but not yet available to be read.

Measurement received too late

Figure 4.4 illustrates the first timing deviation. In real time t'_{meas} falls before t_3, but the communication delay on the measurement at t'_{meas} causes the AKF to receive it later than the input command at t_3. The input command is therefore already stored in the AKF before the processing of the measurement starts. However, this latest input command mustn't be used yet for prediction up to t'_{meas}.

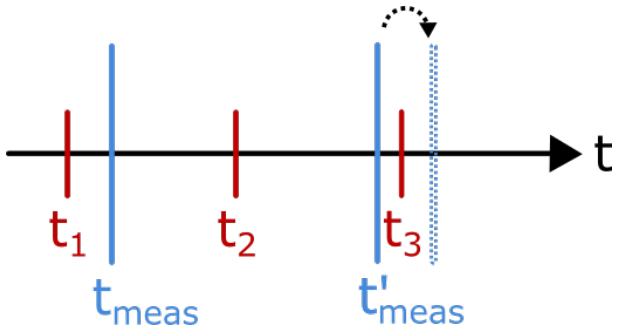


Figure 4.4: Illustration of measurement with large delay causing it to be received after an input command that falls later in time. Solid lines are the real times of measurements and inputs, the dotted line is when the AKF receives the measurement (with delay).

The suggested solution to this problem consists of checking whether the time stamp of the last input in the list of past input commands is situated later in time than t'_{meas}. If this is the case, this input command is removed from the current list of commands, until after the correction step is finished. Then it is re-added to be used in the next correction step.

Input received too late

Figure 4.5 shows the second deviation. In this case the AKF has not yet received the latest input command at t_3 before the new measurement at t'_{meas} arrives. Therefore a single prediction step from t_2 up to t'_{meas} has to be made when recomputing the predicted state at t'_{meas}. This input is then stored however so it can be used in the upcoming correction when a new measurement arrives.

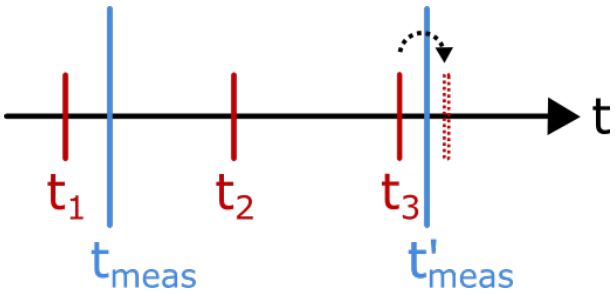


Figure 4.5: Illustration of input with large delay causing it to be received after a measurement that falls later in time. Solid lines are the real times of measurements and inputs, the dotted line is when the AKF receives the input (with delay).

Transform broadcast delayed

The transforms, introduced in Section 4.1, are broadcasted by the ROS nodes for other ROS nodes to be read. Again there is a delay between broadcasting the transform and the availability for reading the transform.

The orientation of the World Yaw reference frame in which the drone model and drone inputs are defined is characterized by the drone yaw estimate $\hat{\theta}$. Each position estimate $\hat{x}_{t_{i+1}|t_i}$ is defined in the rotated frame based on the yaw $\hat{\theta}_{t_{i+1}|t_i}$ at the same time instance t_{i+1}. However, it can happen that when the controller starts to compute the next input based on the state estimate $\hat{x}_{t_{i+1}|t_i}$, that the corresponding World Yaw frame is not yet available due to the delay on the broadcast. This leads to an incorrect transformation of $\hat{x}_{t_{i+1}|t_i}$ which in turn influences the computation of the inputs. Although this is a small error per discrete iteration, its effect is not negligible as the error integrates over time.

The practical solution to this problem exists in the world model delaying the return of the new state estimate until the new transform has become available as depicted in Figure 4.3. Then it is guaranteed that the controller uses adequate information to calculate its input commands. The delay is in the order of a millisecond, such that waiting does not cause time shortage for computations during a controller update cycle.

4.2.3 AKF performance

The performance of the AKF is compared to the most basic way of using the measurement system: always returning the last available measurement as position estimate. This is equivalent to a zero order hold (ZOH) approximation of the drone state. The velocity in that case is retrieved through finite difference of the position measurement.

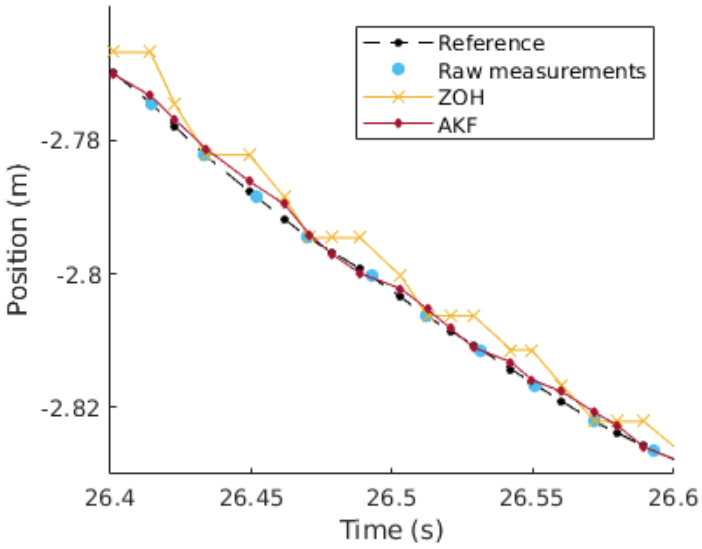
Due to the high quality of the measurements, a good approximation of the exact position state of the drone can be obtained by linear interpolation of the measurements followed by low pass filtering. The filtering improves the approximation since it removes high frequency noise caused by the measurement system. The deviation of the estimate from the approximately perfect position is then taken to be the estimation error in the following analysis.

Figure 4.6 illustrates the performance of the AKF compared to ZOH for two different ranges of acceleration values, indicated by the peak and root mean square (RMS) values of the acceleration over the range. From the figure it is clear that both for lower and higher accelerations the estimation errors for the AKF are significantly below that of the ZOH approximation. The figure also mentions the mean and peak error over the displayed time range for both AKF and ZOH. These results are obtained with following tuning values for the measurement noise covariance and process noise covariance matrices

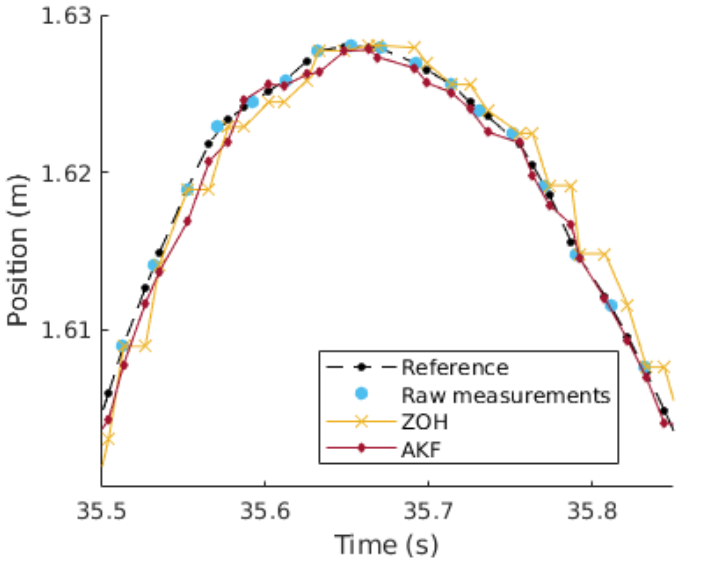
$$R = 1\ (m^2), \qquad Q = \begin{bmatrix} 1\ (s^6) & 0 & 0 \\ 0 & 10\ (s^4) & 0 \\ 0 & 0 & 1\ (s^2) \end{bmatrix}$$

Adding noise to the measurements and lowering the update rate simulates the use of a measurement system of lower quality than the HTC Vive. Figure 4.7 proves that for a system with less than half of the current measurement update rate and significantly more noise, the AKF estimates are still of high quality.

The performance could be improved by more elaborate tuning of the measurement and process noise covariance matrices. However, because the quality of the estimates is currently more than sufficient for this application, the rough tuning given above is maintained.



(a) Lower accelerations



(b) Higher accelerations

Figure 4.6: AKF performance test in x-direction for varying acceleration ranges. Both images display a short subset of the time window of 15 s in which the mentioned range of accelerations is reached. Errors are calculated based on the vertical distance between ZOH or AKF and Reference.
(a) Lower accelerations (peak 0.35 m/s^2, RMS 0.16 m/s^2).
AKF: mean error = 1.2 mm, peak error = 10.9 mm
ZOH: mean error = 3.2 mm, peak error = 15.1 mm
(b) Higher accelerations (peak 1.7 m/s^2, RMS 0.93 m/s^2).
AKF: mean error = 3.2 mm, peak error = 26.5 mm
ZOH: mean error = 8.1 mm, peak error = 37.1 mm

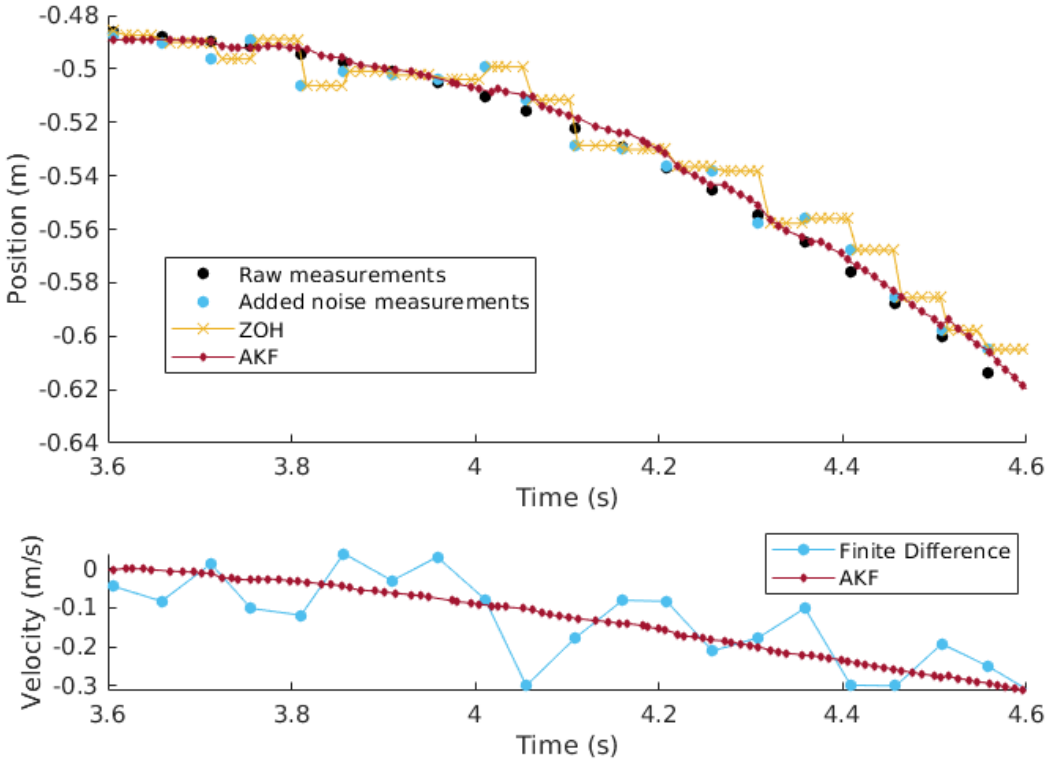


Figure 4.7: Performance illustration of the AKF in the x-direction on a position measurement with added Gaussian noise (mean $\mu = 0\ m$, standard deviation $\sigma = 0.005\ m$) and lowered rate (20 Hz). Kalman filter tuning: $Q = 10^{-5} \cdot I_3$, $R = 1m^2$.

4.3 Conclusion

This chapter discussed the localization, which is split up in perception and state estimation. For the former the software to handle the HTC Vive hardware was introduced, coordinate reference frames were defined and a calibration procedure for a global world reference frame was developed. The latter involved the design of a state estimator that deals with the asynchronous character of the operating system; the asynchronous Kalman filter holds track of timing information and as such copes with communication delay in the system and imperfect controller and measurement system rates. The validation showed that despite the high quality of the Vive localization system, the AKF offers a noticeable improvement with respect to a zero order hold approach where the latest available measurement is directly returned as position estimate. Moreover it proved that for measurements of lower quality, i.e. at a lower rate and suffering from higher noise content, the added value of the AKF is even more significant. The current implementation would therefore produce satisfactory state estimates even with a measurement system of lower quality than the HTC Vive.

Chapter 5

Optimal autonomous navigation

Navigation consists of the autonomous planning of trajectories and the control involved to track them afterwards. This chapter considers the motion planning part of navigation. The control is discussed in the following chapter. Motion planning comprises the generation of trajectories through the world between a starting point and an end goal, given physical constraints such as available space, obstacles and dynamic limitations of the vehicle. A trajectory is defined as a series of positions with a corresponding time at which the position is reached and a corresponding velocity which is attained at that time. It is not to be confused with a path, which is merely a series of positions. In motion planning often the distinction is made between global and local planning. Global planning amounts to finding a rough route from start to end goal, e.g. between all static obstacles. Local planning then comes down to finding the optimal trajectory along parts of that route, avoiding the static as well as potential dynamic obstacles that are encountered. Dynamic obstacle avoidance is briefly covered in this thesis, and the implications of a moving obstacle on the parameters of the optimization solver are discussed. Only a very simple dynamic obstacle avoidance problem is presented, but it already shows a valuable incentive towards autonomous drone navigation through dynamic environments.

The Optimal Motion Generation-tools (OMG-tools) developed by the MECO research team provide a toolbox to solve motion planning problems in a receding horizon fashion. It requires modeling of the environment as simple geometrical shapes. The dynamic model of the drone is also simplified, which means it doesn't take into account the model derived in Chapter 3. The following sections discuss how OMG-tools is used as a motion planner in the control system, how the drone and obstacles are modeled and how the configuration settings of OMG-tools influence the performance. After a note on how the solver handles infeasible problems, A set of experiments is conducted to compare computation times of the different obstacles. The chapter concludes with a note on how to successfully incorporate dynamic obstacles.

5.1 Motion planning using OMG-tools

OMG-tools formulates the motion planning problem as an optimal control problem (OCP) with either the total travel time or the distance to the goal as the objective. The constraints are the initial position, the terminal position, the flight area, the avoidance of obstacles and the drone dynamics represented by a maximum velocity and maximum acceleration. Internally, trajectories are represented through the use of a B-spline formulation. Obstacle avoidance is based on the separating hyperplanes theorem [53]. The mathematical optimization problem is formulated in CasADi and this problem is solved with IPOPT, which uses an interior point algorithm. Details about the implementation of OMG-tools are found in [54].

These trajectories are iteratively computed in an MPC manner: every iteration the optimization problem is solved and a trajectory is returned. Only the first few terms of the solution are actually applied before a new trajectory becomes available which is more up-to-date.

The type of optimization problem solved by OMG-tools in our context is the most basic motion planning problem: a *point-to-point* problem. This amounts to calculating a complete path from starting point to endpoint at once. Extension to more complicated algorithms that divide the flight area into subsets and locally compute an optimal trajectory is not necessary in this demo, due to the restricted flight area and the limited amount of obstacles. As a consequence there is no distinction between the global and local planner when working with this problem type. Only the local planner is active.

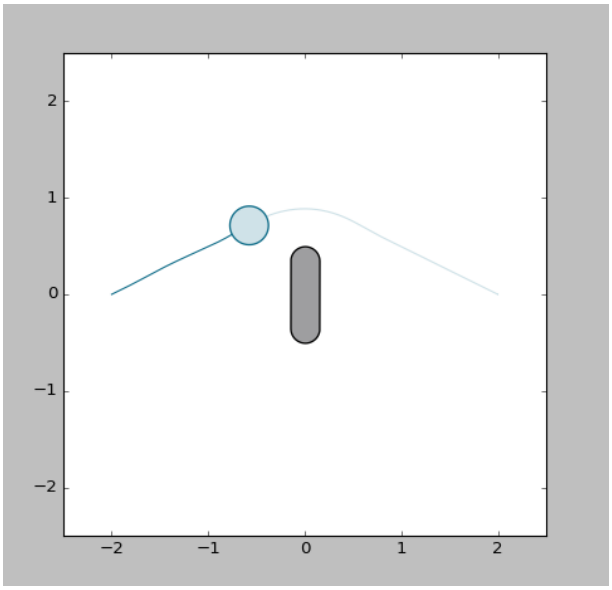


Figure 5.1: Example of the solution to a 2D point-to-point problem solved by OMG-tools. The blue circle is the vehicle avoiding the grey beam-shaped obstacle. It is moving from left (initial position) to right (goal).

An example of a trajectory generated by OMG-tools as a solution to this type of problem is shown in Figure 5.1. The light blue circle represents the drone, the curved line emerging from it is the calculated trajectory that avoids the dark grey obstacle. The figure also illustrates the use of simple shapes for modeling the vehicle and obstacles.

5.1.1 Vehicle and obstacle modeling

vehicles and obstacles are represented by simple geometric shapes to make the mathematical formulation in the optimization problem tractable. The shape chosen for the drone is a sphere. The advantage is that a sphere is computationally efficient to work with. The cost is that at the top and the bottom of the drone, a lot of unoccupied space is attributed to the drone. This limits the height of openings the drone can navigate through. The vehicle dynamics model is taken to be *holonomic 3D*. This model is selected because it works with dynamics that are independent in three orthogonal directions x, y and z, which corresponds well with the dynamic model derived in Chapter 3. A holonomic vehicle model does not consider orientation of the vehicle, meaning the generated trajectories also contain no orientation information.

Obstacles are modeled as cylinders of infinite height, prisms, plates or beams of infinite height. Cylinders of infinite height are circles that are extended infinitely in the z-direction. Mathematically this means the constraint they generate is only expressed in the xy-plane. As will be illustrated experimentally this reduction in dimension is beneficial for the computation time of the optimization. Moreover, circle constraints are, just like spherical ones, computationally more efficient than plates and polyhedrons. This is due to the separating hyperplanes algorithm that is more expensive for obstacles with more vertices. A beam is a rounded rectangular shape, as depicted by the grey obstacle in Figure 5.1. The semicircles at the two ends make it more suited for hyperplane separation than rectangles. Since a beam also consists of a 2D shape which is infinitely extended in the z-direction for 3D navigation purposes, the same computational benefit as described above applies.

5.1.2 OMG-tools configuration settings

The use of OMG-tools entails a number of configuration settings. The ones with largest influence on performance are discussed here and the chosen values are explained.

Fixed or variable time Two different optimization objectives exist for point-to-point problems. The first option is to minimize the distance to the goal at every time step, taking the time horizon as a fixed value. The second option is to take the time itself as the optimization objective. These two formulations lead to different trajectories. In the first formulation, the calculated trajectory results in an optimal solution only for the first part of the trajectory. While the drone flies along the trajectory, the parts further on get updated and become more optimal as well. This

does make for shorter computational times, which is beneficial when using OMG-tools in an MPC manner. The solution to the second formulation immediately gives a more optimal trajectory over the whole time horizon, at the expense of longer computation times. This increase in time is mainly due to an increase in optimization variables as well as the highly non-convex nature of the problem. This second option is preferable in situations where trajectories can be computed offline. For real time demo purposes however, the first option is preferred. The time is therefore set to fixed.

2-norm or ∞-norm constraints The holonomic vehicle dynamics model implies that the drone's maximum velocity and acceleration can be imposed as a constraint in the optimization problem in two ways: either separately for each direction, or coupled. Mathematically this corresponds to limiting the ∞-norm or alternatively the 2-norm respectively of the velocity and acceleration vectors.

Figure 5.2 depicts the difference in obtained trajectories. Intuitively, the trajectory obtained by limiting the 2-norm of the vectors strikes as 'more optimal'. The other solution is in fact optimal as well since the drone will reach the target in the same time window, but while the separate velocity components do not exceed the imposed velocity constraints, the size of the resulting velocity vector does. Therefore a limit on the 2-norm is selected as constraint.

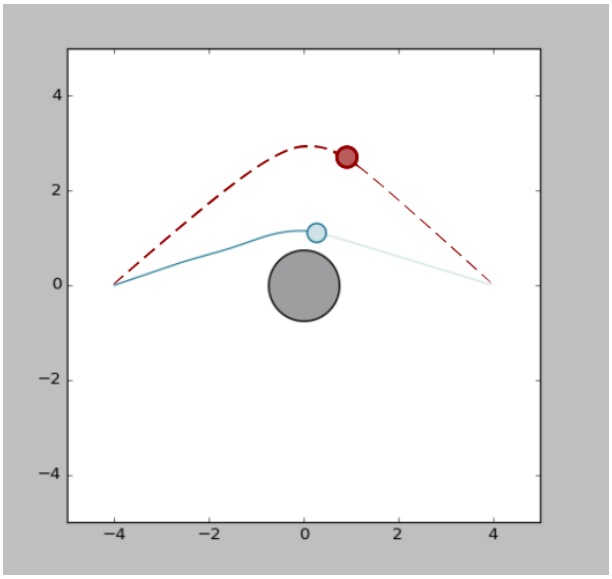


Figure 5.2: Difference in calculated trajectory depending on the type of velocity constraint. The red dotted line represents the use of the infinity norm, the full blue line represents that of the 2-norm.

Hard vs soft terminal constraint Setting a hard terminal constraint requires the goal to be reached at the end of the horizon time. This reduces the region of attraction for the MPC control, but guarantees that the drone does not set off and finds itself stuck at the wrong side of an obstacle halfway before reaching the goal. Given the small flight area and the limited number of obstacles, setting the terminal constraint to a hard constraint and combining this with a sufficiently long horizon time provides success in most practical use cases. 30 s proves to be more than sufficient for demo purposes without increasing the computational load significantly.

The drone finding itself stuck halfway can occur due to the non-convexity of the optimization problem in combination with a soft terminal constraint. Initially, the trajectory does not yet reach up to the final goal, but in time it is shifted further in a receding horizon fashion. In some cases this can lead to the drone ending up at a position where staying in place and not moving towards the end goal corresponds to a local minimum of the optimization cost. This situation cannot occur with a hard terminal constraint, because not reaching the end goal then corresponds to an infinite cost.

When dynamic obstacles are involved however, using a hard terminal constraint might lead to infeasibility of the optimization problem. At a certain point along the trajectory it can happen that an obstacle is temporarily blocking the end goal. This would result in the solver not finding a solution to the problem, thereby failing to compute the rest of the path. In this case it is thus best to only use a soft terminal constraint to avoid infeasibility.

Amount of knots The amount of knots is a design parameter for the spline basis in which trajectories are represented. The higher this number, the more complex the trajectories that can be formulated and thus the more complex the environment that can be navigated through. The downside to using more knots is a significant rise in computation time due to the increase in optimization variables. In practice ten up to twenty knots are used, depending on the complexity of the obstacle configuration.

Safety margin The OMG-tools obstacle avoidance allows for a safety margin, implemented as a soft constraint with a safety weight that trades off extra safety distance to obstacles against the original motion planning objective [54]. Without this safety margin, the planned trajectory passes the active obstacles at exactly the drone radius, meaning that in practice, any deviation from the desired trajectory causes a collision between drone and obstacle. An alternative solution would be to make the drone model radius larger than the real drone, hence imposing a hard constraint. However, in case of deviation from the trajectory towards the obstacle due to tracking errors, the virtual drone and obstacle overlap even though the real drone does not collide with the obstacle. This results in an infeasible situation for the solver. Therefore the soft safety margin is preferred over this second solution.

The size of the safety margin and the safety weight are selected according to the difficulty of the obstacles and are tuned experimentally. In practice the size of the safety margin lies between the drone radius and twice that radius.

5.2 Static obstacle experiments

In order to compare the computational efficiency of different shapes that can be used to model obstacles, eleven experiments are carried out with the actual demo setup. In each experiment, a varying number of obstacles obstructs the path between starting and endpoint, forcing OMG-tools to plan a trajectory around or between them. For each MPC iteration the computation time is stored. Figure 5.3 shows the average and peak computation times for the performed experiments[1]. The first experiment is carried out without obstacles present to provide a reference for comparison.

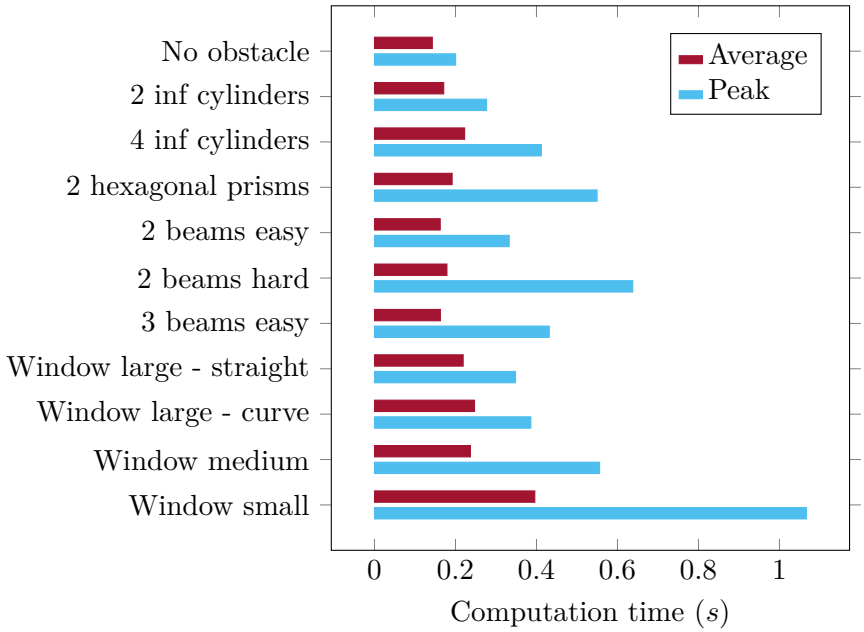


Figure 5.3: Average and peak computation times for different obstacle configurations.

5.2.1 Computation time comparison

The computation times for infinite cylinders are compared to those for hexagonal prisms, since a finite cylinder does not exist in OMG-tools. Figure 5.3 shows that avoiding 2D obstacles which are extended infinitely in the z-direction performs better than 3D prism obstacles, especially when considering peak times.

The beams experiments represent a slalom movement as displayed in Figure 5.4 on the left. In the easy variants, there is almost no overlap between the beams and the curvature of the trajectory is limited. In the hard variant, the plates overlap strongly and the trajectory requires a high curvature to reach the goal without collision. For three beams, the limitations of using OMG-tools in the configuration described above become clear, as the hard variant is termed infeasible by the solver.

[1] All computations are performed on a laptop with Intel Core i5-4210M CPU @ 2.60 GHz x 4 processor and 8 GB of memory.

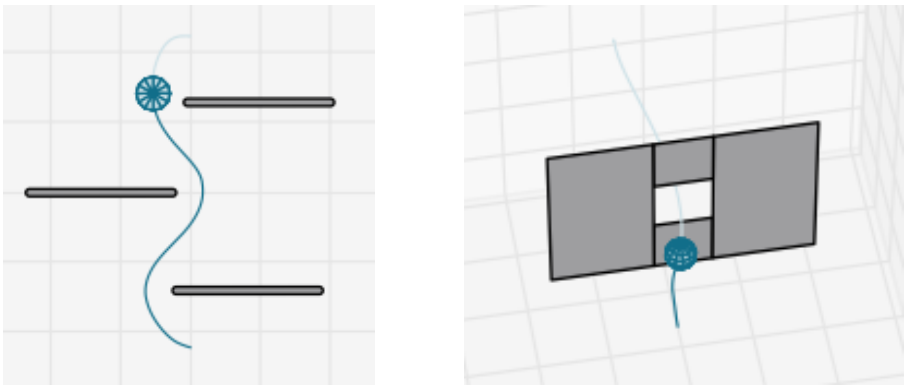


Figure 5.4: Simulation equivalent of the '3 beams easy' experiment (left) and the window experiment with curved trajectory (right).

In the last four experiments the drone has to navigate past four plates as depicted in Figure 5.4. The plates are arranged in such a way that they represent a wall reaching from floor to ceiling and from one side of the flight area to the opposing side, leaving only a hole or window for the drone to fly through. As for the slalom, the vertical plates can again be modeled either as beams or plates. Currently, OMG-tools only supports the extension of 2D shapes in the z-direction, and not in the other two directions. Therefore only the right and left obstacle can be modeled as beam obstacles, whereas the remaining two consist of plate objects. Computation times for this obstacle setup are largely dependent on the size of the window. Shrinking the size of the window results in a large increase in both average as well as peak times. Whenever starting and endpoint lie in such a way that the path has to curve considerably through the window, especially when the window becomes smaller, OMG-tools is no longer able to solve the problem. This once again illustrates the limits of the obstacle avoidance for the current configuration.

Peak times are much higher than average computation times because the optimization solver uses so called *warm starting* of the optimization; it takes the solution from the previous iteration as an initial guess for the current iteration. Since there are no dynamic obstacles present in these experiments, the optimal solution remains nearly the same, except for variations due to tracking errors and refinement of the solution with the receding horizon as mentioned in Subsection 5.1.2. The peak time therefore mostly occurs in the very first iteration, when no good initial guess is available yet.

5.2.2 Handling of infeasible problems

During execution of these experiments, an important pitfall became clear. Whenever a problem is assessed as infeasible, the solver still returns the last guess for the solution of the optimization problem. Since it is infeasible however, that solution

does not satisfy the constraints, e.g. the trajectory runs through obstacles or falls outside the room. Hence a very important monitoring function, cf. Section 2.3, is to detect infeasibility of the solution and to prevent the controller from tracking the provided erroneous trajectory.

5.3 Dynamic obstacle avoidance

When dynamic obstacles are involved, not only the state of the drone has to be updated at each iteration, but also the position and velocity of all dynamic obstacles. Since now not only the drone position changes, but the environment changes as well, the new solution to the problem will differ more from the previous one than in the case when only static obstacles are present. Therefore warm starting will not provide the same benefit as before, which has a negative impact on computation time. It is thus even more essential to use obstacle types that provide a low computational cost such as infinite cylinders or extended beams.

Further on in Chapter 7, a single dynamic obstacle scenario is presented. No static obstacles are present and the dynamic obstacle is modeled as an infinite cylinder. Instead of flying towards an end goal, the drone has to remain at a desired location (both starting and endpoint of the optimization problem). Whenever a dynamic obstacle passes this position, the drone has to move aside to avoid collision and wait until the dynamic obstacle has moved on to return to its original position. In order to accomplish this, the terminal constraint has to be adapted as discussed in Section 5.1.

5.4 Conclusion

In this chapter OMG-tools is proposed as a motionplanner to autonomously compute trajectories from starting to endpoint. It takes into account all constraints to the problem: vehicles dynamics, obstacles and the room dimensions. The computation times and feasibility are optimized by modeling the obstacles and vehicles in an efficient way, and by adapting the parameters which have a large computational influence.

Despite this, navigating in small spaces still leads to a dramatic increase in computation time which was illustrated in the window experiments. Nex to large computation times, there still are several scenario's in which the optimization solver does not succeed in finding a viable trajectory at all. When the curvature along the trajectory becomes too high, especially in the case of computationally heavy obstacles, the problem is declared infeasible. A possible solution would be to use a more complex method, for example the multiframe approach, to solve these types of problems.

Finally an approach to dynamic obstacle avoidance was introduced.

Chapter 6

Position and velocity control

This chapter concerns the controller, cf. the structure in Section 2.3, which decides on the actuation of the mechatronic system. In Chapter 3, the drone's dynamics were analyzed and shaped into an input-output model. Based on this model and state information, the controller computes which input it has to apply to the drone, in order to reach a desired output.

The shape of this output depends on the task and required level of tracking. The most basic level of tracking is a constant or variable setpoint (setpoint tracking), both controlled with proportional-derivative-integrative (PID) feedback. Tracking of a fully predefined trajectory is the next level, which uses a combination of feedback and feedforward control implemented as a zero phase error tracking controller (ZPETC). The final level is the control involved in autonomous navigation, which currently combines feedback control with a Model Predictive Control (MPC) approach. Ideally these autonomously computed trajectories would be tracked by the full combination of MPC, feedback and ZPETC since this would significantly improve tracking performance. This is however not yet attained in the current implementation, but it is touched upon in the text.

First the chapter motivates why a combination of feedback and feedforward control is preferred in trajectory tracking. It then elaborates on the design of each of these two components separately. The feedback controller used in setpoint tracking is (despite small differences in tuning) the same as the one used in trajectory tracking and is therefore not discussed separately. Next a trajectory tracking experiment follows to demonstrate the performance of the combined feedforward-feedback controller. Finally the interaction between the feedback control on one side and MPC on the other is considered, and a possible future extension of MPC combined with ZPETC is discussed.

6.1 Combined feedforward-feedback control in trajectory tracking

The trajectory tracking controller consists of a combination of feedforward and feedback control because they are complementary in their advantages and disadvantages. Pure feedforward control requires a near-perfect model, which is not available, in order not to drift away from the reference trajectory. A feedback controller on the other hand corrects for these modelling imperfections because it reacts to errors on the desired behavior. The downside to this principle is that a pure feedback controller by definition requires this tracking error to function. A feedforward controller does not rely on an error and can thus act more swiftly.

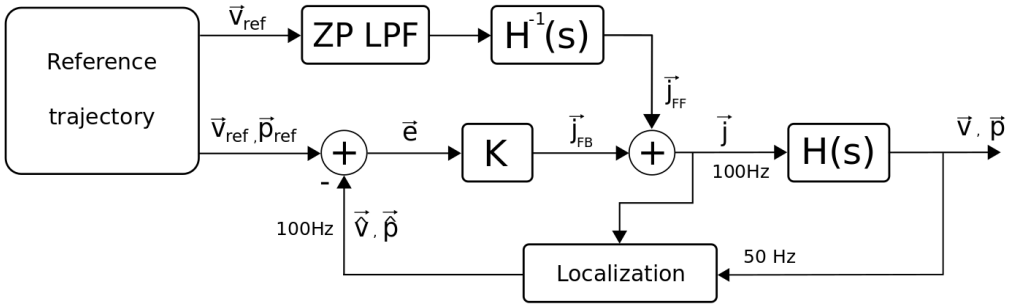


Figure 6.1: Control diagram for combined feedforward and feedback control for tracking a reference trajectory. The drone dynamics $H(s)$ receive as input the combined feedforward ($\vec{j}_{FF}$) and feedback ($\vec{j}_{FB}$) commands. $\vec{v}_{ref}$ and $\vec{p}_{ref}$ together form the reference trajectory. K is the matrix of feedback gains. ZP LPF is a zero-phase low pass filter applied to the data fed to the inverted drone dynamics $H^{-1}(s)$.

Figure 6.1 depicts the control diagram for the combined feedforward and feedback control in trajectory tracking. New input commands are sent to the drone at a rate of 100 Hz. The drone dynamics, represented by $H(s)$ were derived in Chapter 3. The localization block consists of the perception and state estimation from Chapter 4. Relating this scheme to the coordinate reference frames defined in Chapter 4, it is important to note that the positions and velocities in the reference trajectory are expressed in the World frame, whereas the position and velocity error as well as the drone input commands are expressed in the World Yaw frame. This way the input commands are computed in the reference frame in which the derived model is valid.

6.1.1 Feedback controller design

For the third order system from input to position (x- and y-direction), the chosen feedback controller is a proportional-integrative-derivative (PID) controller. As will be described in Chapter 7, pure proportional control results in behavior similar to an undamped spring: the drone oscillates around the goal without reaching

standstill. Pure derivative control results in movement like in a viscous fluid: the controller only reduces the velocity to zero, but does not drive the drone to the desired position. The combination of the two yields a desirable behavior for positioning and trajectory tracking. An integrator is added in order to eliminate steady state errors and to improve tracking performance. The following describes the design of the PID-compensator and evaluates its performance.

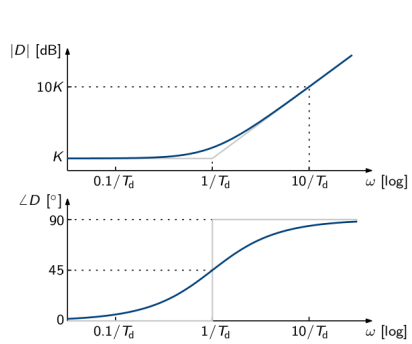
The linear and angular z-directions are treated separately, because of their lower order model. Here a P- or PI-controller suffices.

PID-compensator design

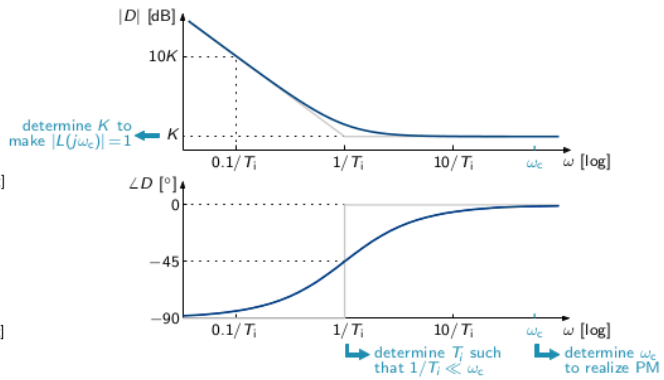
Generally, a PID-compensator has the following continuous time transfer function:

$$\begin{aligned} D(s) &= K(1 + \frac{1}{T_i s})(1 + T_d s) \\ &= \underbrace{K(1 + \frac{T_d}{T_i})}_{P} + \underbrace{\frac{K}{T_i}\frac{1}{s}}_{I} + \underbrace{K T_d s}_{D} \\ &= K_p + \frac{K_i}{s} + K_d s \end{aligned} \tag{6.1}$$

The design comprises the choice of the parameters K_p, K_i and K_d in expression (6.1). Following the approach of [55], the design of a PID-compensator consists of the combined design of a PD- and a PI-compensator. Figure 6.2 qualitatively shows the bode diagrams for PD- and PI-compensators and the design philosophy for the latter.



(a) PD-compensator.



(b) PI-compensator.

Figure 6.2: Qualitative bode diagrams of PD- and PI-compensators. The design philosophy for the PI-compensator is indicated [56].

The approach is a frequency response based procedure, in which the objective is to maximize the closed loop bandwidth, under the constraint of a desired phase margin (PM). The size of the desired phase margin depends on the task at hand. E.g. positioning allows for more aggressive feedback than in the case of trajectory tracking

with or without MPC. Further details on this are discussed later. For the analysis here it suffices to state that the lower the phase margin, the more aggressive the resulting controller, but also the larger the maximum overshoot in the step response of the closed loop system becomes [55].

The design approach consists of the following steps. First choose the new crossover frequency ω_c of the compensated system as the frequency at which the phase of the uncompensated system is equal to $\phi = -180 + PM - lead + lag + c$. PM is the desired phase margin, *lead* is an anticipated lead addition of 90° due to the derivative term, *lag* is an anticipated lag of 10° to 15° added by the integrator, and c is a correction term because the derivative and integrative terms do not add exactly 90° of lead and 10° - 15° of phase lag.

Next take the derivative time T_d equal to $T_d = 10/\omega_c$ to place the high phase lead at the desired frequency (see phase in Figure 6.2a). Select the integration time T_i such that the contribution of the integrative part to the phase at ω_c equals the anticipated lag of 15°. To this end, take the integration time between $T_i = 3.73/\omega_c$ and $T_i = 6/\omega_c$ depending on whether a strong or more moderate integrative action is desired. Same as with the choice of PM, the size of the integrative action depends on the desired behavior during a specific task.

Finally select K such that the crossover frequency of the compensated system is indeed located at ω_c. In other words, K must be such that

$$\begin{aligned} |D(j\omega_c)H(j\omega_c)| &= 1 \\ |K(1 + \frac{1}{T_i j\omega_c})(1 + T_d j\omega_c)H(j\omega_c)| &= 1 \\ \implies K = \frac{1}{|(1 + \frac{1}{T_i j\omega_c})(1 + T_d j\omega_c)H(j\omega_c)|} \end{aligned}$$

During positioning (hovering in place), PM is taken equal to 30° with high integrative action. The obtained design parameters in this case for the x-direction are

$$\begin{aligned} K = 0.9456\ m^{-1},\ T_d = 1.8930\ s,\ T_i = 0.7061\ s \\ \implies K_p = 3.4806\ m^{-1},\ K_i = 1.3391\ (ms)^{-1},\ K_d = 1.7900\ m^{-1}s \end{aligned}$$

Figure 6.3 shows the resulting frequency response of the open loop compensated system for the x-direction. The y-direction is fully analogous. The linear and angular z-directions follow an analogous design procedure, but omit the derivative term.

Because in trajectory tracking the velocity reference is defined together with the position reference, this thesis uses a slightly different implementation than the classic PID implementation. In the traditional expression, the drone input command in the continuous-time Laplace domain $J(s)$ is given by

$$J(s) = D(s)E_p(s) = K_p E_p(s) + \frac{K_i}{s} E_p(s) + K_d s E_p(s)$$

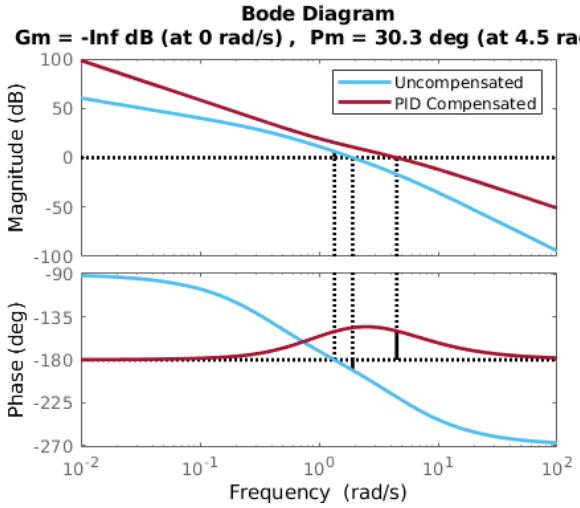


Figure 6.3: Bode diagram of the open loop uncompensated and PID-compensated system, with indication of corresponding gain and phase margin of the closed loop system.

where $E_p(s)$ represents the position tracking error and $D(s)$ is the continuous-time transfer function as in Equation (6.1). In this expression, the velocity error $E_v(s)$ is not used directly but is computed as a function of the position error $E_p(s)$: $E_v(s) = sE_p(s)$. In this thesis however the error on the velocity can be directly computed as the difference between the reference velocity and the estimate of the velocity produced by the state estimator discussed in Chapter 4. This is preferable because the estimator returns a smoothed velocity estimate that is corrected by measurements. Therefore the expression above can be replaced by:

$$J(s) = K_p E_p(s) + \frac{K_i}{s} E_p(s) + K_d E_v(s)$$

The discrete-time domain controller after discretization with a Tustin discretization scheme is:

$$\begin{aligned} j[k+1] = \; & j[k] + \left(K_p + \frac{K_i T_{s,ctrl}}{2}\right) e_p[k+1] \\ & + \left(-K_p + \frac{K_i T_{s,ctrl}}{2}\right) e_p[k] \\ & + K_d(e_v[k+1] - e_v[k]) \end{aligned}$$

where $j[k+1]$ is the input command that will be applied as the next control input and $j[k]$ is the control input that was sent at the beginning of the current controller cycle. K_p, K_i and K_d are the control parameters that were computed above. $T_{s,ctrl}$ is the controller update time, with a value of 0.01 s. $e_p[k]$ and $e_v[k]$ are the position and velocity tracking errors respectively at time k.

Figure 4.3 in Chapter 4 contains detailed timing information on when control inputs are computed and sent out. The input command for the next cycle is computed as soon as the position and velocity estimate come in from the estimator. It is possible to do this in advance since the computed estimates are valid at the next controller time instance in the future. The inputs are then applied at the very beginning of the next controller cycle, to ensure that the rate at which they are sent is as constant as possible. The Kalman estimates benefit from a more constant rate of the controller, as they assume a fixed controller update rate for the prediction estimates.

PID closed loop performance

The position error on which the feedback controller acts usually lies in the range of a few centimetres since the position setpoints lie close to the drone, both for positioning and (MPC) trajectory tracking. Therefore a relevant measure of performance for the PID controller is its response to a small position reference step input of 10 cm.

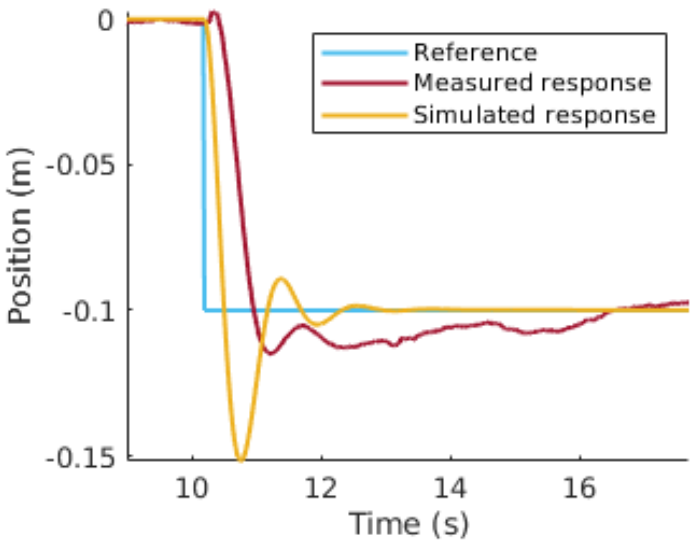


Figure 6.4: Step response of the closed loop PID-compensated system (x-direction) for a step reference input of 10 cm.

Figure 6.4 depicts the result of an experiment to validate this position reference step response of the closed loop PID-compensated system. The figure also shows the simulated closed loop response for comparison. There clearly is a discrepancy between the response of the idealized linear model and the real behavior of the drone; the measured response starts of with a small bump in the opposite direction to the step reference and the shape and amplitude of the remainder of the response also differ from that of the simulated response. The first discrepancy, the bump, is most likely caused by the flexibility of the tracker suspension that was presented in Chapter 2. The sudden shock of the step response shakes the suspension and distorts

the position measurements of the drone, which was assumed to be rigidly fixed to the Vive tracker.

The second discrepancy, the difference in amplitude and shape of the response, is explained by nonlinearity of the real dynamic system which is not captured in the derived linear time invariant model.

Despite the discrepancy between modeled and real behavior, the performance is adequate for the use in positioning and tracking.

6.1.2 Feedforward controller design

The feedforward control converts reference velocities from the reference trajectory to drone input commands.

As a first method to control the drone with feedforward velocities, it was opted to use a software package developed by *Autonomy Lab* which changes the drone interface to a velocity controlled interface [57]. After testing this package turned out not to work properly. The velocity tracking exhibits erratic behavior, and the desired velocity is often not attained. Therefore the package is replaced by a second, self-implemented method based on the inversion of the identified velocity model.

The top line in Figure 6.1 with $ZP\ LPF$ and $H^{-1}(s)$ forms the feedforward control in this implementation. Because the velocity model returns a velocity as output for a given drone input, the inverse model yields a required drone input command for a desired velocity output. Inverting the model for the x-direction gives the frequency response in blue in Figure 6.5.

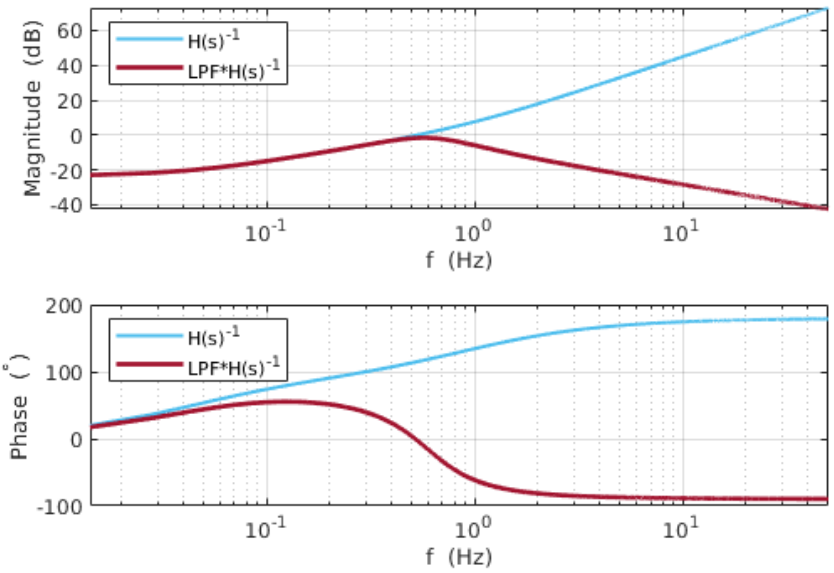


Figure 6.5: Bode plot of filtered vs non-filtered inverse transfer function.

In practice, simply inverting the model is not an adequate way of computing input commands. Since the magnitude of the velocity frequency response descends at high frequencies (recall from Chapter 3 that the velocity model in the x- and y-direction is a second order system with no zeros), the inverted system rises. As an unintended consequence, high frequency components in the input signal are amplified. Furthermore, the model itself is not at all accurate for these higher frequencies, as can be seen in Figure 6.6. There the relative difference between the identified frequency response and the empirical frequency response is plotted in blue. The latter is obtained as the fast fourier transform (FFT) of the output divided by the FFT of the input. The frequency response function (FRF) of the relative difference FRF_{diff} is given by

$$FRF_{diff} = \frac{H_{identified} - H_{empirical}}{H_{identified}}$$

and is a measure for the uncertainty on the model. This is discussed more in depth in [17, 58], where an approach for inversion-based controller design is provided. The design presented below is based on this approach.

The proposed solution is to low pass filter the inverted system with a Butterworth filter, such that high frequency amplification is inhibited. The cutoff frequency of this LPF is chosen such as to weaken the amplification at frequencies with high model uncertainty. As depicted in Figure 6.6 this boils down to making a trade-off; the cutoff frequency has to be placed low enough such that it falls before the point where the FRF_{diff} and the phase difference start rising drastically, but high enough to prevent useful model information from being discarded. For the x-direction this is taken to be at a frequency of 0.6 Hz. The resulting low pass filtered inverted system is displayed in Figure 6.5 in red.

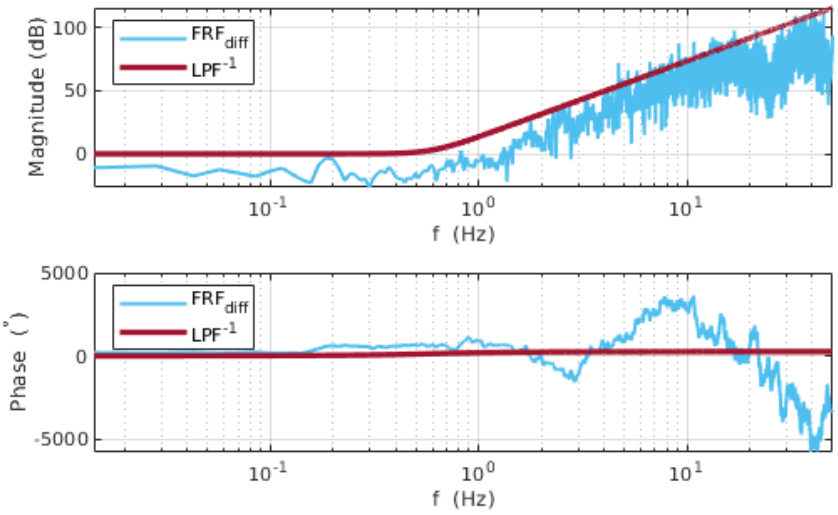


Figure 6.6: Relative difference between identified and empirical transfer function (FRF_{diff}) and low pass filter inverse (LPF^{-1}).

Whenever a given velocity trajectory is sent through this model, the Butterworth filter introduces a frequency dependent phase lag, which adds delay to the generated feedforward inputs. This makes for a mismatch in timing between the position and velocity trajectory reference since the former is not delayed. If these feedforward inputs are applied in combination with the inputs from a feedback controller acting on the position trajectory, this mismatch of inputs will deteriorate the tracking performance.

The proposed solution is to implement the LPF as a zero phase low pass filter (ZP LPF). Figure 6.7 illustrates how to achieve this zero phase shift by sequentially applying two LPF's: first an LPF is applied in reverse direction on the predefined trajectory, in order to create phase lead on the signal [58]. This action treats the entire trajectory at once before the actual flight. Next a second LPF is applied, equal to the previous filter, but this time in the forward direction and at runtime since it is combined with the inverse model. The consecutive phase lead and lag of both filters compensate each other, such that the resulting feedforward input command corresponds with the feedback command at the same time. Note that this approach implies that the zero phase filter is non-causal, and it is only possible to apply it when the trajectory is known in advance, as is the case in the current application of trajectory tracking. This non-causal way of eliminating phase lag in the feedforward control is a solution to the zero phase error tracking control (ZPETC) problem [59].

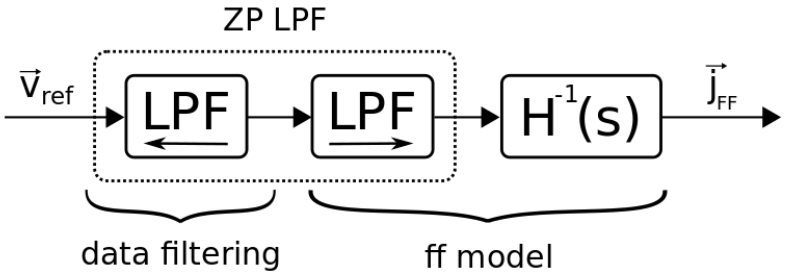


Figure 6.7: Schematic representation of zero phase low pass filter (ZP LPF) required for zero phase error tracking control (ZPETC). The two LPF's consist of the same low pass filter but the first applied in reverse direction on the velocity trajectory ($\leftarrow$) and the second in forward direction ($\rightarrow$). The reverse filter is executed in advance as soon as the reference trajectory becomes available. The forward filter is included in the state space representation of the inverse velocity model.

For practical application, the continous-time LTI transfer function of the inverted velocity model together with the forward LPF is discretized with a Tustin discretization scheme and expressed in a state space representation. The backward LPF is implemented as a digital filter and is applied directly on the data representing the trajectory.

6.1.3 Trajectory tracking experiments

Now a number of experiments is conducted to quantify the performance of the combined feedforward-feedback (FF-FB) controller with respect to the feedback controller on its own. The drone has to track the same reference trajectory for a range of increasing maximum velocities. Each time the entire reference trajectory is scaled in time such that the maximum velocity over the trajectory corresponds with the maximum specified value. Figure 6.8 depicts this reference trajectory and corresponding tracking errors for a maximum velocity of 1.2 m/s and in the case of the combined FF-FB controller. This velocity is also the default maximum velocity for the tracking of arbitrary trajectories in the demo.

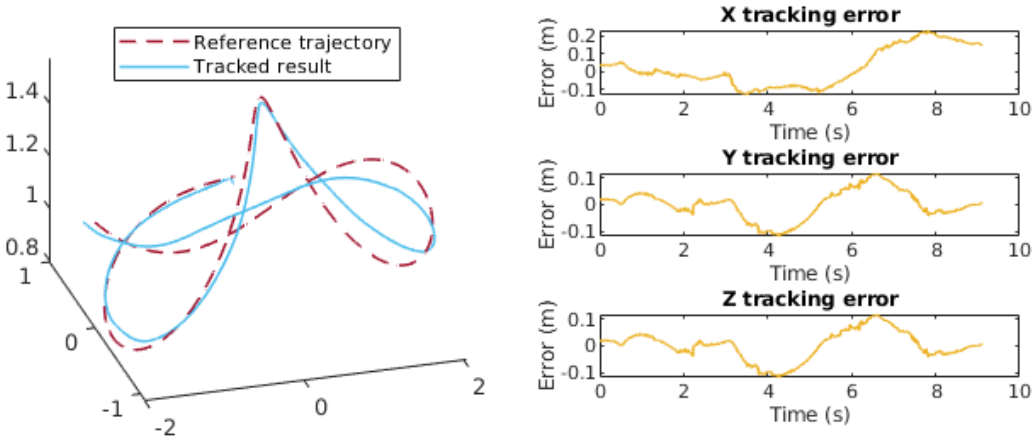


Figure 6.8: Experiment for performance quantification of trajectory tracking with combined feedforward-feedback control. Result for the 'medium' experiment (maximum velocity of 1.2 m/s).

Figure 6.9 shows the results as average and peak position tracking errors for four different maximum velocities ranging from 0.6 m/s up to 2.4 m/s, both for combined feedforward-feedback tracking as well as feedback only tracking. The figure shows that for medium to (very) high velocities (1.2 m/s, 1.8 m/s and 2.4 m/s), the addition of the feedforward controller provides a major benefit. The fastest experiment cannot be safely conducted in the Robotics lab with feedback only, because the peak error becomes excessively large. This high velocity is however completed by the feedforward-feedback controller with a tracking error lower than that of feedback only in the medium velocity experiment.

For the lowest velocity, feedback alone is better than combined feedforward and feedback. This is because of practicalities in the programming of the filtering of trajectories. The current implementation involves meticulous padding to prevent data from being cut off by the filtering operation, as well as shifting of the data to prevent a time shift between the feedback and the feedforward signal. This shifting time is however taken as an average value over all velocities. At low speed, the

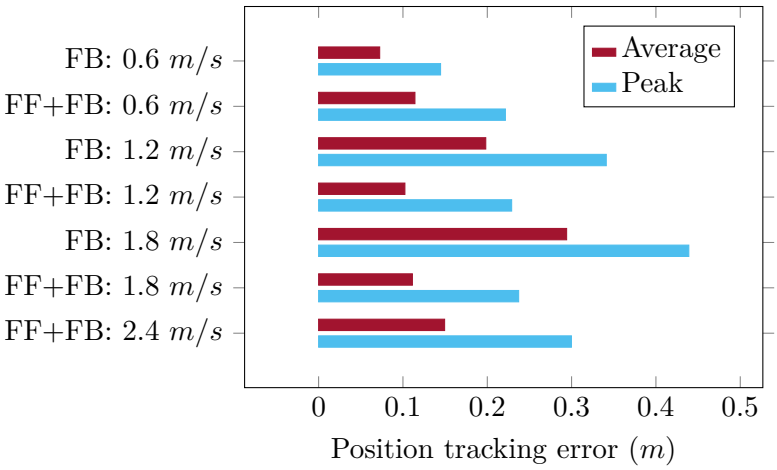


Figure 6.9: Average and peak position tracking errors for varying velocities, both feedforward (FF) and feedback (FB) combined and FB only. The experiments are slow (0.6 m/s), medium (1.2 m/s), fast (1.8 m/s) and very fast (2.4 m/s).

imperfection on the shifting time results in a small mismatch between the feedback and feedforward signal, thus resulting in a larger tracking error. Since the total error at this speed is lower, the effect of this mismatch is more outspoken. At higher speeds, the benefits of feedforward strongly outweigh this small disadvantage of the implementation. Moreover, in practice the tracking error at low velocity turns out to be mostly due to leading of the drone with respect to the trajectory. This means that the error normal to the trajectory is a lot smaller than the data in Figure 6.9 suggests. Visually, the FF+FB combination strongly outperforms the FB only. For the demo this visual effect is most important, and as such the FF+FB tracking is successful, even at low speeds. But in the absolute quantification of trajectory tracking, where the timing of the position is also important besides the staying on the given path, there is room for improvement.

6.2 Model Predictive Control with feedback control for autonomous navigation

Chapter 5 already mentioned that the motion planner OMG-tools is used in an MPC fashion. It periodically provides reference trajectories with a receding horizon that allow the drone to autonomously navigate in an obstructed environment. Figure 6.10 shows the control diagram involved in autonomous navigation. During autonomous flight the velocities are limited to the 'medium' velocity (1.2 m/s) setting. This maximum velocity is a trade-off between having a small tracking error while still flying at a sufficiently high velocity.

Because of the periodic updates of the MPC controller, feedback of the changes in the environment and drone state is inherently present. Ideally, only the first input of the reference trajectory is applied before a new trajectory is calculated which

incorporates the latest changes. However, the MPC updates occur at a relatively low rate of 2-5 Hz, due to the relatively large computation times (in the order of 0.2 s to 0.4 s on average, recall from Section 5.2). In order to navigate safely, the drone must make corrective actions more frequently. Therefore, the actual position and velocity controller that tracks the reference trajectories in between OMG-tools updates runs at a higher rate of 100 Hz. This means that the latest reference trajectory is tracked for a range of inputs, until a new one becomes available.

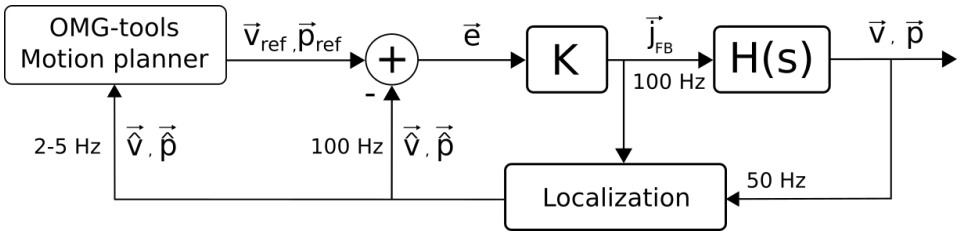


Figure 6.10: Control diagram for feedback control combined with OMG-tools.

Subsection 6.1.3 showed that the feedback-feedforward controller provided a large improvement in tracking accuracy compared to the feedback only controller, especially when moving to higher velocities. However, in the current implementation the entire trajectory must be known in advance in order to execute all filtering and padding that was described in Subsection 6.1.2. When working with OMG-tools which periodically updates the trajectory, this is not the case. The current implementation does not guarantee that the old trajectory smoothly connects to the new trajectory. This would result the drone 'jumping' between subsequent parts of the path. Therefore the trajectory is tracked by solely using the feedback controller.

An improved implementation could overcome this problem by treating the previously calculated trajectory simultaneously with the newly calculated trajectory by connecting them before filtering to assure continuous, smooth flight. This would strongly improve the accuracy of the tracking in autonomous flight, and thus allow for more dynamic flying.

Note that for an environment where all obstacles are static, the MPC approach is not strictly necessary for optimal autonomous navigation. The optimal trajectory could also be computed entirely in advance, and subsequently tracked with high accuracy using the combined feedforward-feedback control described in the previous section. However, this thesis also aims to show an incentive towards dynamic obstacle avoidance, where the feedback on the environment is crucial. Therefore the more general approach with receding horizon is retained.

6.3 Conclusion

In this chapter a feedback and feedforward controller have been developed to generate the desired inputs for steering the drone. The feedback controller consists of a PID controller where the P and I term are based on the position error, whereas the D term is based on the velocity error taking into account the desired velocity. This last addition really boosts the performance in practice and makes the controller more responsive. The feedforward controller comes down to a zero phase error tracking controller which uses forward and reverse filtering combined with an inverse model of the drone to generate lag-free drone inputs. In all possible tasks that the drone can execute, the feedback part is used as the basic controller. Addition of the feedforward part does provide large benefits for tracking accuracy, especially at higher velocities. However because of the current implementation, feedforward can only be used when the full trajectory is known beforehand which is not the case when using OMG-tools in an MPC manner. Therefore an adaptation to this part to incorporate processing of trajectories at runtime would certainly benefit the speed and accuracy at which the drone can fly between obstacles. Another aspect that would improve performance is a higher update rate of the MPC controller since this increases responsiveness towards disturbances and modelling errors. Due to hardware limitations this was not possible in this thesis.

Chapter 7

Demo

This chapter elaborates on how all building blocks presented in the previous chapters are combined into an illustrative demo. It explains how the demo is implemented as a Finite State Machine (FSM) to allow the flexibility to freely pass from one task to the other, depending on the audience's wishes and which tasks are available to choose from. Aside from the possibility to choose the order of tasks, a default scenario is suggested. Necessary safety measures are discussed as well.

7.1 Finite State Machine definition

Since an important part of the demo is the interactive aspect, one of the Vive controllers is reserved for the audience to use, while the other one is used by the operator. The audience Vive controller has limited functionality in the sense that it is only active during part of the demo, when interaction is desired and safe. The operator Vive controller on the other hand grants the operator the freedom to adapt the demo scenario to the wishes of the audience present, while still maintaining full control. At all times, he can select which control principle to illustrate as well as repeat the same illustration whenever the audience desires without re-initializing the software. In order to facilitate this flexibility, the demo is implemented as an FSM. This concept was already touched upon in Chapter 2, but now a more elaborate discussion is in place.

An FSM in this context is a programming structure where the *state* of the system is uniquely defined at all times as one out of a finite number of predefined states. Each state in turn uniquely specifies the behavior of all entities in the system in the form of predefined *actions*. For safety reasons and fluent operation it is crucial that the drone is in the state expected by the operator, such that no undesirable behavior occurs. Therefore, state information is always displayed on the output screen. Transitions between states are triggered by specific *events*, either internally monitored or externally imposed by operator actions. Finally *tasks* are composed as a fixed sequence of states. This is to ensure that during execution of a task, all actions are performed in the correct order without accidentally skipping some actions.

Grouping sequences of states in tasks is an elegant way to use the same actions (e.g. performing position feedback, autonomous navigation, etc.) in varying contexts. For example, flying autonomously to an arbitrary goal can either occur as illustration itself when avoiding a set of obstacles, or alternatively as a tool for positioning the drone in the desired spot before the actual illustration begins. Yet they rely on the exact same building blocks that were described in Chapters 5 and 6. The structure of the FSM will be schematically illustrated for one task.

7.2 Available tasks

The tasks that are available in the current version of the demo are *undamped spring and viscous fluid*, *drag drone*, *track drawn trajectory*, *point-to-point* and *dodge dynamic obstacle*. Following section describes what these tasks do, which principles they illustrate and the modeling that is involved. The main idea is that different tasks illustrate subsequent levels of autonomous flight. Three levels are explored starting from basic constant setpoint tracking ('stay where you are'), to trajectory tracking ('track where you want to be') up to automatic trajectory generation and tracking ('navigate through the world'). They illustrate the control principles of setpoint tracking and disturbance rejection, feedforward and PID-feedback control, trajectory tracking and navigation using MPC. In this regard the use of an FSM has the advantage that the appropriate tuning is selected depending on the task and state at hand. More difficult obstacles for example require more prudent parameters, whereas less challenging tasks allow for more aggressive settings. This adaptivity to each situation is part of what makes the control system advanced.

7.2.1 Undamped spring and viscous fluid

These two very similar tasks illustrate the equivalence of the terms in a PID-controller with their mechanical counterparts using simple feedback control. More specifically, it illustrates the effect of the proportional and derivative components of this feedback controller. In the first of the two tasks, only the proportional component is retained. Looking back at the physical reasoning which lead to the drone model in Chapter 3, it can be seen that the for the x- and y-direction, the input command is proportional to a force. The mechanical equivalent of a component yielding a force proportional to a displacement is a spring. This means that when pulling or pushing the drone away from its setpoint, it will behave similarly to an undamped spring that oscillates around its center point.

In the second task, the derivative component is retained. The output generated by the controller can now be seen as a force which is proportional to the velocity of the drone. The mechanical component corresponding to this behavior is a linear damper or dashpot. Pulling or pushing the drone around, one will notice that the higher the velocity, the fiercer the drone fights the disturbance. It does not return to its original position however once the disturbance disappears.

7.2.2 Drag drone

This task extends the static setpoint tracking to a variable setpoint using a PID-feedback controller. The variable setpoint is adjusted by holding the trigger button on the audience Vive controller and moving the controller around. The audience can intuitively and visually identify closed loop dynamics, and the concept of closed loop bandwidth and resonance is nicely clarified by gradually increasing the frequency at which the controller is displaced. For the sake of interactivity, the spectators can take the controller and play around with the drone while standing in a safe, shielded area. A virtual room boundary prevents anyone from choosing a setpoint that lies too close to the room edge, thereby avoiding a crash of the drone.

The performance of the PID-feedback on a step reference input has been discussed in Section 6.1.

7.2.3 Track drawn trajectory

The next step is to track an arbitrary trajectory. Again to introduce interactivity, the trajectory can be drawn by a spectator. Drawing the trajectory works as follows: while holding the trigger on the audience Vive controller, the location of that controller is sampled at the localization rate and stored in a list. Taking the numerical derivative of that list results in the corresponding velocity trajectory. Together the position and velocity trajectory form the reference trajectory to be tracked. It can happen that a spectator draws part of the trajectory too close to the wall of the room. In order to remedy this, it is mapped onto the feasible set of positions inside the room. After drawing, the drone takes off, flies to the starting point of the trajectory, and executes the tracking of the trajectory. Figure 7.1 shows this sequence represented as part of the FSM.

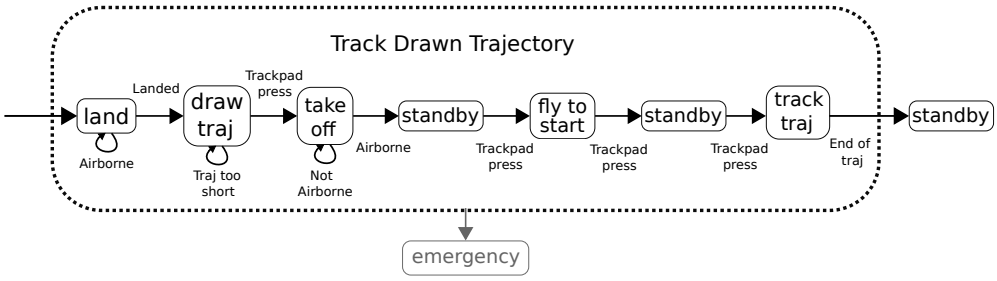


Figure 7.1: FSM representation of the 'Track drawn trajectory' task. Labels inside full frames are states. Other labels are events, triggered by the monitor. At any time a state can be interrupted when the emergency state is evoked.

The drawn trajectory can be intractable because of two reasons: high velocities that safety precautions don't allow, or sharp corners and thus high accelerations that the drone's dynamics don't allow. The first problem is overcome by checking the maximum velocity along the drawn trajectory, and scaling the reference trajectory

such that the highest velocity along the trajectory corresponds to the highest allowable velocity. This scaling is obtained by first linearly interpolating the list of positions according to the desired maximum velocity. Afterwards the new velocities are derived from the position trajectory. The second problem is solved with low pass filtering of the position trajectory. A Butterworth low pass filter with a cutoff frequency equal to the bandwidth of the identified drone model filters out frequencies that the drone dynamics physically cannot attain.

An example of a drawn trajectory is shown as the orange line in the lower left corner of Figure 7.2. The actual flight trajectory of the drone is represented by the green line. The performance of the tracking of an arbitrary drawn trajectory has been discussed at the end of Section 6.1.

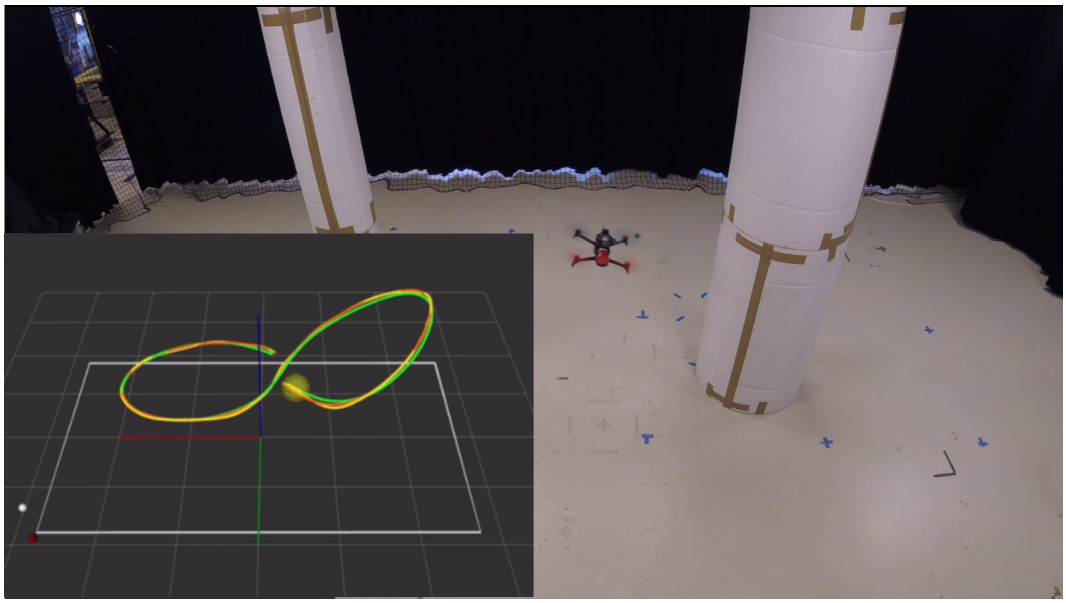


Figure 7.2: Illustration of the track drawn trajectory task. The drone tracks an arbitrary trajectory drawn by one of the spectators. Lower left corner: virtual representation of the task. The yellow ball represents the drone, the orange line is the drawn trajectory, the green line is the actual path flown by the drone.

7.2.4 Point-to-point

This task is the collection of all sub-tasks involving static obstacle avoidance. It comprises the computation of an optimal trajectory through the obstructed flight area by OMG-tools and the tracking of that trajectory as discussed in Chapters 5 and 6. First a set of physical obstacles is placed by the audience. They can position the obstacles at arbitrary locations to avoid suspicion of hard-coded, fine-tuned environments. Subsequently this task is invoked to specify the desired end goal, to which the drone must navigate while avoiding the obstacles present. This end goal is indicated by the operator through the use of the operator Vive controller.

As the automatic detection of obstacle poses is not a subject of investigation in this thesis, the obstacles are converted to a software representation by locating them with the audience Vive controller. Each physical obstacle has a specific representation in OMG-tools which is limited by the set of available shapes discussed in Section 5.1. The different obstacle types and there effect on computation time have been discussed in Section 5.2. Modeling each obstacle as one of these types requires a trade-off between the most efficient representation and the best resemblance of the actual situation. The following paragraphs explain the definition of the different obstacles and the relation between the physical obstacles and the OMG-tools representations.

Cylindrical obstacles

A cylinder reaching from floor to ceiling is best represented in OMG-tools by a 2D circle object that is extended infinitely in the vertical direction. For this type of obstacle the lowest computation times have been recorded. Therefore it allows the largest number of distinct obstacles to be placed while still providing a sufficiently high MPC update time. Figure 7.3 shows the drone navigating between several cylinders that obstruct the path from start to endpoint.

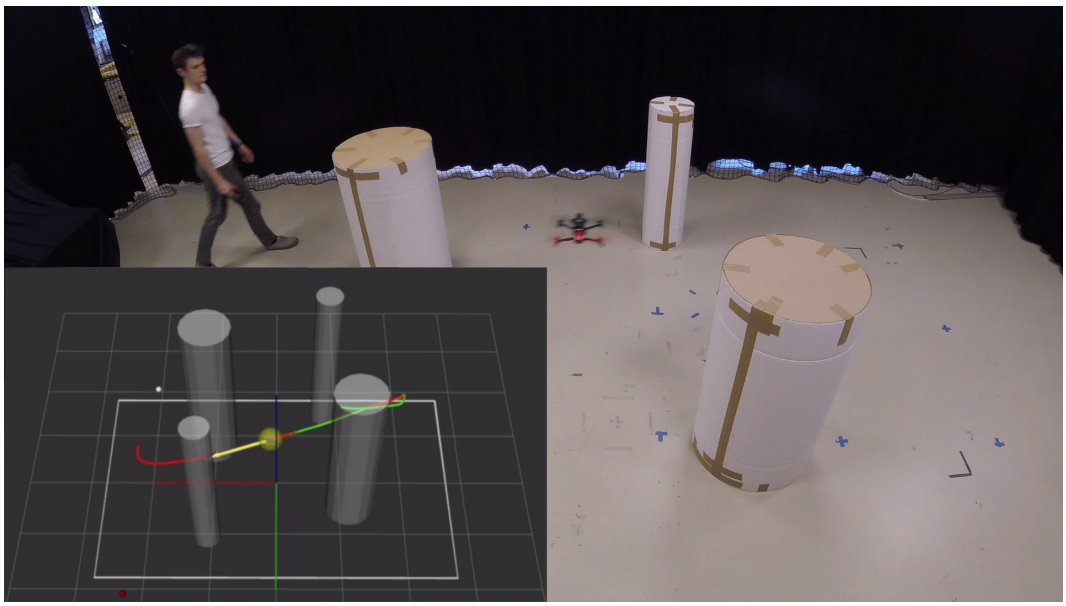


Figure 7.3: Illustration of autonomous navigation between cylindrical obstacles. The drone autonomously navigates between a set of cylinders obstructing the room. The virtual representation now also contains the cylindrical obstacles. The red line is the computed reference trajectory from the motion planner, the yellow arrow is the estimated velocity of the drone.

Slalom obstacles

The drone must slalom between poles placed on a line. The operator indicates which side of the pole may be passed. In OMG-tools, the slalom is represented by the 'beam' obstacle class. The beam extends from the location of the pole up to the edge of the room. An example of a drone slalom between two poles is given by Figure 7.4.

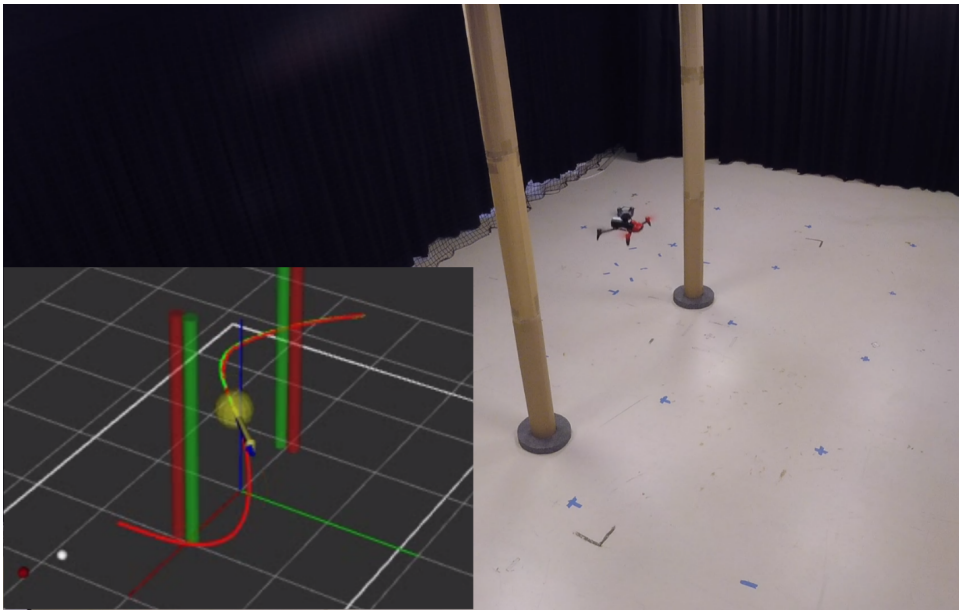


Figure 7.4: Illustration of the autonomous navigation between slalom obstacles. The drone autonomously slaloms between two pillars since it is only allowed to pass on the green side of each pillar.

Hexagonal obstacles

OMG-tools does not provide cylindrical obstacles with finite height. A good approximation that does exist is a prismatic regular hexagon. Recall however from Section 5.2 that hexagonal obstacles are computationally more expensive than (infinite) cylinders.

Window obstacles

The most challenging obstacle to model and navigate through is the window. Section 5.2 presented it as four plates surrounding the hole, or alternatively as a combination of two beams and two plates. Physically this obstacle is visualized by only retaining the edges of the window, thereby keeping the plate approach 'behind the scenes'. This is once again to obtain a visually more attractive demo. During the demo a narrow window is used as illustrated in Figure 7.5.

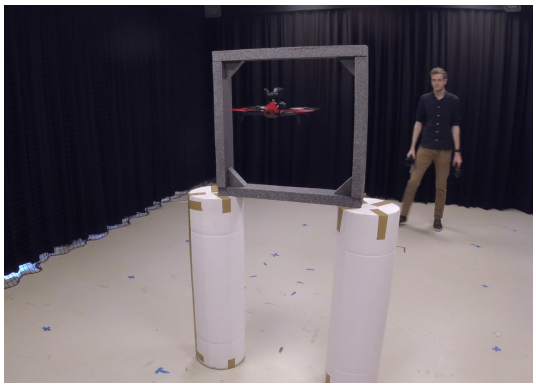


Figure 7.5: Illustration of the autonomous navigation with a window obstacle.

7.2.5 Dodge dynamic obstacle

Finally there is one last task which involves a dynamic instead of a static obstacle. After the drone has taken off and hovers in place, a dynamic obstacle passes through the room. The operator fulfills the role of dynamic obstacle by holding the operator Vive controller and walking towards the drone. From this Vive controller the position and velocity of the obstacle are derived, therefore allowing OMG-tools to predict the expected trajectory. It then computes a trajectory for the drone to evade the moving obstacle, after which it quickly returns to its original position. Figure 7.6 depicts the drone dodging a dynamic obstacle (a walking person).

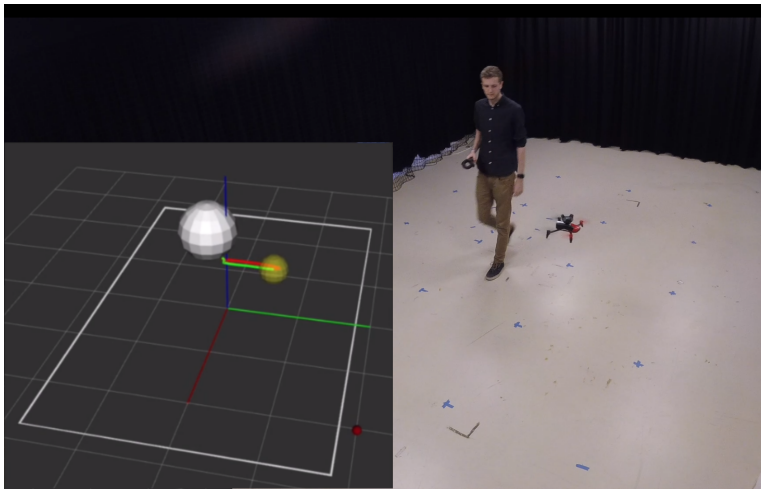


Figure 7.6: Illustration of the dodge dynamic obstacle task. The drone autonomously flies aside to let a moving obstacle (person holding a Vive controller) pass before returning to its original position. The white ball in the virtual representation is the obstacle. The red line, again the reference trajectory computed by the motion planner, plans to return to the initial position at the center of the room when the obstacle has passed.

7.3 Demo execution

An extensive user manual is found in the 'Readme' document accompanying the program code. Here a brief summary is provided. First the default scenario is discussed, followed by a number of safety measures to prevent accidents in the case of non-nominal events.

7.3.1 Default scenario

As mentioned before, the demo intends to show the autonomous capabilities of the drone in a fashion of increasing difficulty. It starts by illustrating the simplest type of feedback control in the undamped spring and viscous fluid tasks. Next it proceeds to the more challenging tasks of drag drone and tracking a drawn trajectory. In a subsequent step the level of autonomy is increased by letting the drone plan its path autonomously, thereby increasing the difficulty from a few cylinders, plates and hexagonal prisms up to more challenging obstacles like the slalom or a tight window. Finally the demo finishes with an example of dynamic obstacle avoidance through the task of dodge dynamic obstacle.

The order of tasks presented here is a default sequence that is certainly not strictly imposed. The FSM implementation grants the flexibility to change the order arbitrarily whenever the operator or audience desires without having to reboot the program.

Each task is separately requested by the operator via the ROS graphical user interface *rqt* after which the desired sequence of states initiates. A virtual version of the drone and its surroundings is displayed in the ROS visualization environment *rviz*. It shows the contours of the flight area, the current location of the drone, the placement of the obstacles and the location of the Vive controllers as well as trajectories generated by the motion planner or by drawing.

7.3.2 Safety measures

Since part of the demo requires an operator to be present inside the flight area, and some tasks involve spectators entering the flight area, strict safety measures must be taken.

Firstly, to ensure safety of the audience, the flight area is shielded by a safety net. While the drone is in the air, no spectator is allowed to enter the flight area. Whenever a spectator wants to place an obstacle or draw a trajectory, the drone automatically lands and stays on the ground until the operator triggers the event that all obstacles are placed, after which the drones takes off again. During the drag drone task, the spectators can manipulate the drone while safely standing outside the flight area.

Secondly, to ensure that no spectator accidentally sends an unintended instruction to the drone when pressing a wrong button on the Vive controller, the controller reserved for the operator has different functionality compared to the one for the audience. The operator's controller can trigger events in the FSM, indicate OMG-tools goals and make the drone take off or land. The audience controller has limited functionality and is only active in FSM tasks and states where its input is explicitly required.

Thirdly, anticipating possible accidents due to improper use of the drag drone or trajectory drawing task, the physical limits of the room are taken into account as mentioned in Section 7.2 by establishing virtual room boundaries inside the physical room. If a setpoint is placed outside these virtual limits, it is automatically projected back onto the edges of the virtual room, thereby preventing the drone from hitting the physical room edges. This means the drone can be dragged freely through the flight area as long as it remains far enough from the edges of the room. The same happens for a drawn trajectory; all points drawn outside the virtual room are projected back onto the edges of this room.

Finally, to increase the operator's safety, the FSM is developed such that in case of emergency, the drone automatically brakes. This is possible thanks to the built-in brake function of the Parrot Bebop 2, which operates based on the internal IMU and optical flow sensors. The emergency state is invoked when an invalid measurement is detected (when the vibration problem discussed in Section 2.1 causes malfunctioning of the localization system), or when an infeasible problem is detected. The problem with the latter has been mentioned in Section 5.2; the optimization solver returns its last guess of the solution, even when the computation of an optimal trajectory was not successful, meaning that the returned solution is nonsense. The reasoning is that it is better to do nothing (brake) than to act based on unreliable or completely wrong information. Performing feedback based on an invalid measurement or tracking a trajectory that is nonsense is very dangerous and is therefore strictly avoided.

7.4 Results and conclusion

This chapter explained how the demo is implemented in an FSM structure to allow for flexibility during the demo, and presented the different tasks that are available to display the concepts of localization, control and navigation. Next follows an evaluation of the functioning of the demo.

The FSM is robust, and allows tasks and states to be interrupted at any instant. Invalid measurements and infeasible problems are correctly detected by the monitor, such that the drone can be guaranteed not to follow infeasible trajectories or perform control actions based on erroneous measurements. The invalid measurements do however still occur sometimes, causing the demo to be interrupted. Moreover the fluent execution of the demo is still very dependent on how well the Vive controllers function; the button presses are sometimes not detected when the Vive controllers have been idle for a couple of seconds. This problem is most probably due to a software issue in PyopenVR or SteamVR.

The interactivity is strongly appreciated by most audiences, especially in the drag drone task and the drawing of arbitrary trajectories. Even under erratic inputs, the drag drone task remains functional, and arbitrary trajectories are tracked with similar quality as observed in the experiment of Section 6.1. Cylinder obstacle placement by the audience often causes challenging navigation problems, which are successfully executed by the drone. Slaloming works robustly and navigation through a narrow window is successful as long as the required curvature of the path through the window is limited. The lack of feedforward control during autonomous navigation restricts the velocities and accelerations reached during point-to-point tasks, due to limited tracking accuracy. The dodge dynamic obstacle task functions well, and excites the audience since failing to evade the human would result in a human-drone collision.

Chapter 8

Conclusion

Previous chapters presented practical designs, implementations and integrations of results of existing research for the modeling, localization, control and navigation of drones. The last chapter concluded on how to unify these features in an appealing and interactive demo, which is made available as an open-source software package. This concluding chapter provides a summary of the contributions made in this text and relates the presented work to the initially set goals. It also provides suggestions for future research related to the thesis.

8.1 Summary of research

The research objective is to produce an interactive demo on the indoor localization, control and navigation of drones. This demo intends to illustrate MECO's activities in the field of optimal control and autonomous navigation of UAV's, with a broad target audience of both researchers and non-informed spectators. Chapter 2 presents the selection of the hardware used to accomplish this goal, and the approach taken for implementation. A functional decomposition consisting of perception, world model, control, monitoring and navigation forms the basis of the software developed in this thesis.

More concretely, the successful construction of such a demo in summary requires following efforts. The model that describes the drone dynamics must be both as simple as possible, yet as complex as necessary. Chapter 3 builds towards a SISO linear time-invariant model in each direction, which is proven to be adequate both for localization as for controller purposes. Chapter 4 explains the integration of the Vive localization system into the demo setup and presents the implementation of an asynchronous Kalman filter that performs position and velocity state estimation. The algorithm meticulously holds track of timing information, in order to appropriately deal with timing issues such as communication delay and imperfect controller and measurement rates. Its performance is validated and its added value also illustrated in a scenario with added noise and lowered update rate. MECO's motion planner OMG-tools and its use in a practical setting is presented in Chapter 5. Adequate

modeling of obstacles as simple shapes and proper choice of configuration parameters of the optimizer for a given task are crucial for successful navigation. The influence of the choice of obstacle type on computation times is analysed. Chapter 6 discusses the design of suitable controllers for the subsequent levels of autonomous flight: setpoint tracking, trajectory tracking and optimal autonomous navigation. The considerable added value of inversion-based feedforward control with zero phase filtering superposed on feedback control is proven in a trajectory tracking experiment. This thesis does not yet reach a full implementation of combined feedforward-feedback control in the context of receding horizon autonomous navigation, but the possible benefit of extending the current implementation is made clear. Finally Chapter 7 explained the highest level of advanced control in the thesis; a Finite State Machine incorporates adaptivity to situation changes, arbitrary task selection at any time and safety measures through monitoring. This flexible structure allows for reconfiguration of control parameters and motion planner settings at runtime, meaning the demo can operate continuously without rebooting between different tasks or emergency situations.

The interactive aspect is met with the use of Vive controllers and specifically designed tasks that allow the audience to interact with the drone in a safe way. The audience can drag the drone around to illustrate variable setpoint tracking, they can creatively draw trajectories to be tracked by the drone and they can arbitrarily place obstacles to make the navigation as challenging as desired.

The resulting demo is fully operational and accessible to audiences of any kind. However some struggles with hardware malfunctioning remain. The Vive tracker still sometimes refuses to function properly due to excessive vibrations. The presented ad hoc solution of flexible suspension with additional damping solves the problem up to an acceptable level, but it is not flawless. Also the Vive controllers that tend to suspend remain a problem as this interrupts fluent workflow during the demo.

In summary, the main contributions of this thesis to the autonomy of mechatronic systems are the development and practical implementation of an asynchronous Kalman filter and a model inversion-based feedforward controller, the integration of the Model Predictive Control-based motion planner OMG-tools, and the design and implementation of a Finite State Machine for high level discrete control. These contributions are useful in the sense that they illustrate how practical applications can be developed starting from the theoretical concepts of modeling, state estimation, feedforward control and Model Predictive Control. The demo shows the research and industrial world an experimental validation of OMG-tools, with its potential, but also limitations as a motion planner. A high number of obstacles or complex obstacle types drastically increase the computation time of the optimizations or worse, renders the problem infeasible for the solver. Next to that, the demo is useful as it intends to inspire future generations of researchers with an impressive, fun-to-watch illustration of what the research domains of mechatronics have to offer.

8.2 Suggestions for future research

The modular structure of the implementation allows for and even encourages future extensions of the currently available demo. Some suggestions are made here.

The first and most obvious extension that should be made to the demo is extending the OMG-tools navigation with a feedforward controller, as this would significantly improve tracking performance, leading to faster and more accurate flying. For even more dynamic flight, reaching higher accelerations and velocities, efforts should go out to modeling (e.g. the introduction of a nonlinear model) and controller design (incorporating the new model, or alternative advanced control techniques entailing for example iterative learning control) as well as improvements in the computation time of optimal trajectories.

The demo could also be extended with more involved navigation tasks, such as finding an exit in a maze, or completing a sort of racetrack with waypoints. More extensive navigation could also include division of the world into smaller areas, in order to split the large navigation problem into smaller, more tractable problems. The distinction between global and local planner that was omitted in this thesis could be added in that regard. OMG-tools also provides techniques for these more involved navigation problems, such as the multiframe approach. Another OMG-tools feature which is not yet elaborated is motion planning for vehicle fleets. Very impressive visual results could be obtained by flying multiple drones in formation through an obstructed area, while simultaneously offering an experimental validation of another set of OMG-tools' capabilities.

In the implementation framework, the motion planner operates independently from the controller (apart from the exchange of necessary information), such that it is rather straightforward to replace the current motion planner with another one. Given the limitations of the OMG-tools motion planner, the integration of a different motion planner would be interesting for comparison. In the same fashion the localization system could easily be replaced by another one without drastically affecting the functioning of the other components. The other localization systems discussed in the introduction of this text or even a combination of some of them could be tested. This could serve as a solution for the loss of measurements from one system, as the other system could take over.

Automatic obstacle detection was not treated in this thesis. The extension of the framework with obstacle detection either via the on-board camera, off-board cameras or tracking devices on the obstacles would increase the autonomy of the current setup.

Lastly the combination of the track drawn trajectory task and navigation with MPC offers interesting possibilities; instead of blindly tracking the drawn trajectory with the presented feedforward-feedback controller, future research could extend this task to tracking via MPC. This way the drone could track the trajectory in an optimal way, while taking into account constraints such as obstacles obstructing the reference path.

Bibliography

[1] Interreg, "Smart tooling." https://www.grensregio.eu/projecten/smart-tooling, 2018.

[2] BASF België | BASF Antwerpen, "Smart tooling." https://www.basf.com/be/nl/who-we-are/Group-Companies/BASF-Antwerpen/Projects/Smart-tooling.html, 2019.

[3] R. Beck and M. Bos, "Interactive autonomous drone demo." https://github.com/RianBeck/DroneDemo, 2019.

[4] R. Beck and M. Bos, "Interactive demo on the indoor localization, control and navigation of drones." https://www.youtube.com/watch?v=4jUyKBdEDlI, 2019.

[5] A. Hussein, A. Al-Kaff, *et al.*, "Autonomous indoor navigation of low-cost quadcopters," in *2015 IEEE International Conference on Service Operations And Logistics, And Informatics (SOLI)*, pp. 133–138, IEEE, 2015.

[6] A. Garcia, E. Mattison, and K. Ghose, "High-speed vision-based autonomous indoor navigation of a quadcopter," in *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, pp. 338–347, IEEE, 2015.

[7] M. Krogius, A. Haggenmiller, and E. Olson, "Flexible layouts for fiducial tags (under review)." Under Review.

[8] J. Wang and E. Olson, "Apriltag 2: Efficient and robust fiducial detection," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4193–4198, 10 2016.

[9] SwatBotics, "Demo of april tag localization system." https://www.youtube.com/watch?v=Y8WEGGbLWlA, 2012.

[10] Bitcraze AB, "Mocap deck." https://www.bitcraze.io/2018/06/mocap-deck/, 2019.

[11] R. Van Parys, M. Verbandt, *et al.*, "Distributed coordination, transportation & localisation in industry 4.0," in *17th International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, IEEE, 2018.

[12] D. W. Mellinger, *Trajectory Generation and Control for Quadrotors*. PhD thesis, University of Pennsylvania, 2012.

[13] G. Espinosa and M. Rubenstein, "Towards mixed reality system with quadrotor: Autonomous drone positioning in real and virtual.," in *Workshop: Robotics in Virtual Reality, 2018 IEEE International Conference on Robotics and Automation (ICRA 18).*, 2018.

[14] J. Kim and B. Kim, "Accurate states estimation using asynchronous kalman filter with encoder edges for tmrs," in *2013 13th International Conference on Control, Automation and Systems (ICCAS 2013)*, pp. 1528–1533, 2013.

[15] G. Xue, Y. Xu, *et al.*, "The fractional kalman filter-based asynchronous multirate sensor information fusion," *Complexity*, vol. 2018, pp. 1–10, 2018.

[16] B. Deepak and P. Singh, "A survey on design and development of an unmanned aerial vehicle (quadcopter)," *International Journal of Intelligent Unmanned Systems*, vol. 4, no. 2, pp. 70–106, 2016.

[17] S. Devasia, "Should model-based inverse inputs be used as feedforward under plant uncertainty?," *IEEE Transactions on Automatic Control*, vol. 47, pp. 1865–1871, Nov 2002.

[18] N. Dadkhah and B. Mettler, "Survey of motion planning literature in the presence of uncertainty: Considerations for uav guidance," *Journal of Intelligent & Robotic Systems*, vol. 65, pp. 233–246, Jan 2012.

[19] T. Mercy, R. Van Parys, and G. Pipeleers, "Spline-based motion planning for autonomous guided vehicles in a dynamic environment," *IEEE Transactions on Control Systems Technology*, vol. 26, pp. 2182–2189, Nov 2018.

[20] R. Van Parys and G. Pipeleers, "Spline-based motion planning in an obstructed 3d environment," in *Proceedings of the 20th IFAC World Congress*, vol. 50, pp. 8668–8673, Elsevier, 2017.

[21] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, 1997.

[22] P. Saranrittichai, N. Niparnan, and A. Sudsang, "Robust local obstacle avoidance for mobile robot based on dynamic window approach," in *2013 10th International Conference on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology*, pp. 1–4, IEEE, 2013.

[23] D. Claes and K. Tuyls, "Multi robot collision avoidance in a shared workspace," *Autonomous Robots*, vol. 42, pp. 1749–1770, Dec 2018.

[24] C. Rosmann, F. Hoffmann, and T. Bertram, "Timed-elastic-bands for time-optimal point-to-point nmiscar model predictive control," in *2015 European Control Conference (ECC)*, pp. 3352–3357, EUCA, 2015.

[25] G. Rossetti, R. Van Parys, *et al.*, "Optimal autonomous quadrotor navigation in an obstructed space," in *Proceedings of the 4th International Workshop on Research, Education and Development on Unmanned Aerial Systems*, pp. 19–24, 2017.

[26] L. Ma, J. Xue, *et al.*, "Efficient sampling-based motion planning for on-road autonomous driving," *IEEE Transactions on Intelligent Transportation Systems*, vol. 16, pp. 1961–1976, Aug 2015.

[27] Y. Kuwata, G. A. Fiore, *et al.*, "Motion planning for urban driving using RRT," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1681–1686, Sep. 2008.

[28] IROS Secretariat, "Iros autonomous drone racing 2018." https://www.iros2018.org/competitions, 2018.

[29] S. Li, M. Ozo, , *et al.*, "Autonomous drone race: A computationally efficient vision-based navigation and control strategy," *CoRR*, vol. abs/1809.05958, 2018.

[30] E. Kaufmann, M. Gehrig, , *et al.*, "Beauty and the beast: Optimal methods meet learning for drone racing," *CoRR*, vol. abs/1810.06224, 2018.

[31] K. Wei and B. Ren, "A method on dynamic path planning for robotic manipulator autonomous obstacle avoidance based on an improved rrt algorithm.," *Sensors (Basel, Switzerland)*, vol. 18, no. 2, 2018.

[32] Okreylos, "Htc vive localization system." http://doc-ok.org/?p=1478, 2016.

[33] Games on Track, "Games on track vive localization system." http://www.gamesontrack.com/satellites.html, 2018.

[34] Marvelmind Robotics, "Marvelmind localization system." https://marvelmind.com/, 2017.

[35] Pozyx NV, "Pozyx localization system." https://www.pozyx.io/store/detail/2, 2019.

[36] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pp. 3400–3407, May 2011.

[37] A. Hernandez, C. Copot, *et al.*, "Identification and path following control of an ar.drone quadrotor," in *2013 17th International Conference on System Theory, Control and Computing (ICSTCC)*, pp. 583–588, IEEE, 2013.

[38] Parrot, "Parrot bebop drone specifications." http://blog.parrot.com/2016/01/12/comparison-bebop-2-vs-bebop-drone/, 2019.

[39] Intel, "Intel aero drone specifications." https://software.intel.com/en-us/aero/drone-kit, 2019.

[40] DJI, "Dji drone specifications." https://www.dji.com, 2019.

[41] M. Monajjemi, "Bebop autonomy." https://bebop-autonomy.readthedocs.io/en/latest/, 2015.

[42] L. Hardesty, "Autonomous robotic plane flies indoors," *Targeted News Service*, 2012.

[43] Open Source Robotics Foundation, "Robot operating system." http://www.ros.org/about-ros/, 2019.

[44] Triad Semiconductor, "Triad openvr." https://github.com/TriadSemi/triad_openvr, 2018.

[45] C. Bruns, "Pyopenvr." https://github.com/cmbruns/pyopenvr, 2019.

[46] Valve corporation, "Steamvr." https://steamcommunity.com/steamvr, 2017.

[47] S. Pfeiffer, "Read out htc vive controller keypresses." https://gist.github.com/awesomebytes/75daab3adb62b331f21ecf3a03b3ab46, 2019.

[48] A. G. Mutambara, "Chapter 8 - digital control systems," in *Design and Analysis of Control Systems - 1st Edition*, p. 652, CRC Press, 1999.

[49] L. Ljung, "Chapter 4 - models of linear time-invariant systems," in *System Identification: Theory for the User*, pp. 69–126, Englewood Cliffs, New Jersey 07632: P T R Prentice Hall, 1987.

[50] HTC Corporation, "Htc vive tracker (2018) developer guidelines ver. 1.0." https://dl.vive.com/Tracker/Guideline/HTC_Vive_Tracker(2018)_Developer+Guidelines_v1.0.pdf, 2018.

[51] T. Foote, E. Marder-Eppstein, and W. Meeussen, "tf2." http://wiki.ros.org/tf2, 2019.

[52] T. Kailath, A. H. Sayed, and B. Hassibi, "Chapter 9 - the kalman filter," in *Linear Estimation*, pp. 310 – 369, Upper Saddle River, New Jersey 07458: Prentice Hall, 2000.

[53] S. Boys and L. Vandenberghe, "Chapter 2 - separating and supporting hyperplanes," in *Convex Optimization*, pp. 46–50, The Edinburgh Building, Cambridge, CB2 8RU, UK: Cambridge University Press, 2004.

[54] R. Van Parys, *Fast and Distributed Model Predictive Control: Tailored solutions for mechatronic systems*. PhD thesis, KU Leuven, Celestijnenlaan 300 bus 2420, B-3001 Leuven (Belgium), October 2018.

[55] G. F. Franklin, J. D. Powell, and A. Emami-Naeini, "Chapter 6 - the frequency-response design method," in *Feedback Control of Dynamic Systems*, pp. 328–452, Edinburgh Gate, Harlow, Essex CM20 2JE, England: Pearson Education Limited, 2015.

[56] G. Pipeleers and J. Swevers, "C9. Control design using frequency response methods," in *Control Theory - Handouts*, CuDi VTK vzw, 2017.

[57] Autonomy Lab, "bebop_vel_ctrl." https://github.com/AutonomyLab/bebop_vel_ctrl, 2016.

[58] H. Elci, R. W. Longman, and R. Ugoletti, "Discrete frequency based learning control for precision motion control," in *Proceedings of IEEE International Conference on Systems, Man and Cybernetics*, vol. 3, pp. 2767–2773, Oct 1994.

[59] M. Tomizuka, "Zero phase error tracking algorithm for digital control," *Journal of Dynamic Systems, Measurement, and Control*, vol. 109, pp. 65–68, Mar 1987.