Agile quadrotor maneuvering using tensor-decomposition-based globally optimal control and onboard visual-inertial estimation

by

Fabian Riether

B.S., Engineering Cybernetics (2013)
University of Stuttgart

Submitted to the Department of Mechanical Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Mechanical Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2016

Author .....
Department of Mechanical Engineering
August 01, 2016

Certified by .....
Sertac Karaman
Associate Professor of Aeronautics and Astronautics
Thesis Supervisor

Certified by .....
John Leonard
Professor of Mechanical and Ocean Engineering
Mechanical Engineering Faculty Thesis Reader

Accepted by .....
Rohan Abeyaratne
Quentin Berg Professor of Mechanics
Chairman, Department Committee on Graduate Theses

Agile quadrotor maneuvering using tensor-decomposition-based globally optimal control and onboard visual-inertial estimation

by

Fabian Riether

Submitted to the Department of Mechanical Engineering
on August 01, 2016, in partial fulfillment of the
requirements for the degree of
Master of Science in Mechanical Engineering

Abstract

Over the last few years, quadrotors have become increasingly popular amongst researchers and hobbyist. Although tremendous progress has been made towards making drones autonomous, advanced capabilities, such as aggressive maneuvering and visual perception, are still confined to either laboratory environments with motion capture systems or drone platforms with large size, weight, and power requirements.

We identify two recent developments that may help address these shortcomings. On the one hand, new embedded high-performance computers equipped with powerful Graphics Processor Units (GPUs). These computers enable real-time onboard processing of vision data. On the other hand, recently introduced compressed continuous computation techniques for stochastic optimal control allow designing feedback control systems for agile maneuvering.

In this thesis, we design, implement and demonstrate a micro unmanned aerial vehicle capable of executing certain agile maneuvers using only a forward-facing camera and an inertial measurement unit. Specifically, we develop a hardware platform equipped with an Nvidia Jetson embedded super-computer for vision processing. We develop a low-latency software suite, including onboard visual marker detection, visual-inertial estimation and control algorithms. The fullstate estimation is set up with respect to a visual target, such as a window. A nonlinear globally-optimal controller is designed to execute the desired flight maneuver. The resulting optimization problem is solved using tensor-train-decomposition-based compressed continuous computation techniques. The platform's capabilities and the potential of these types of controllers are demonstrated in both simulation studies and in experiments.

Thesis Supervisor: Sertac Karaman
Title: Associate Professor of Aeronautics and Astronautics

Mechanical Engineering Faculty Thesis Reader: John Leonard
Title: Professor of Mechanical and Ocean Engineering

Acknowledgments

First and foremost, I would like to thank my advisor Sertac Karaman. His knowledge and unending trust in my abilities to realize those ideas that we generated during scheduled meetings, random but extended discussions in the lab and late-night email conversations greatly pushed what I achieved during my Master's degree. I would also like to thank my lab mates Abhishek Agarwal, John Alora and Thomas Sayre-McCord. The crucial, and countless, coding- and segfault-sessions in "the basement lab" would have never been as productive and fun without you. A big thanks goes to all undergraduates and exchange students who were part of this project, especially Roberto Brusnicki whose passion and experience for designing and building all sorts of cool robots greatly helped to kick-start the project.

I would also like to thank Luca Carlone for sharing his wisdom and experience on visual-inertial estimation and principled execution in engineering projects. Working with Alex Gorodetsky's cutting-edge contributions in the field of compressed computation pushed my time at MIT from mere engineering to research. I am proud and grateful for this opportunity. I also have to thank John Leonard for being the Mechanical engineering faculty thesis reader.

Lastly on this side of the pond, I would like to thank my roommates and friends who became wonderful companions on this two-year journey of research, classes, sports, personal development and life in general.

Finally, I would like to express my profound gratitude to my amazing parents and brother for their limitless support and understanding for a son and brother who is doing well, but is constantly away from home.

This thesis was partially funded by AFOSR through grant FA9550-14-1-0399 and by NSF through grant CNS-1544413. The German Academic Exchange Service and the Fulbright Commission supported my studies abroad at MIT. Their support is gratefully acknowledged. Senator Fulbright once noted that *"The Fulbright Program aims to bring a little more knowledge, a little more reason, and a little more compas-*

sion into world affairs and thereby increase the chance that nations will learn at last to live in peace and friendship." I am proud to say that it definitely brought knowledge, reason and compassion into my life, and I will aspire to contribute to build a world living in peace and friendship.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Over the last few years quadrotors have become increasingly popular in industry, consumer entertainment, research and amongst hobbyists. Many open source projects offer solutions ranging from simple flight controllers up to the full suite of components that are required for versatile deployment of unmanned aerial vehicles (UAV) including control, guidance and mission planning [40]; engineering magazines featured tutorial-style introductions to modeling, planning and control of quadrotors [42]; undergraduate controls classes have been taught using toy drones [36].

For researchers, the quadrotors' high maneuverability, complex underactuated dynamics and affordable price point make them an attractive platform choice to demonstrate novel approaches to robotic autonomy.

However, limited onboard computing power oftentimes creates severe bottlenecks. These bottlenecks lead to two types of quadrotors: a) completely autonomous but slow-moving vehicles with larger size, weight and power requirements as in [49]; b) fast, agile platforms that require a priorly known environment and offboard support through motion capture systems (as in [44]) or planning algorithms.

This separation impeded transitioning new solutions from the lab environment to real-world field deployment. Bridging this gap requires taking solution for estimation

and scene understanding used in slow, fully autonomous platforms to smaller platforms and speed them up under the constraint of limited onboard computing power. Control algorithms need to account for nonlinear effects that cannot be neglected during more agile maneuvers any longer [52]. Figure 1-1 illustrates this problem setup conceptually. On the one hand, a heavily researched solution approach falls under

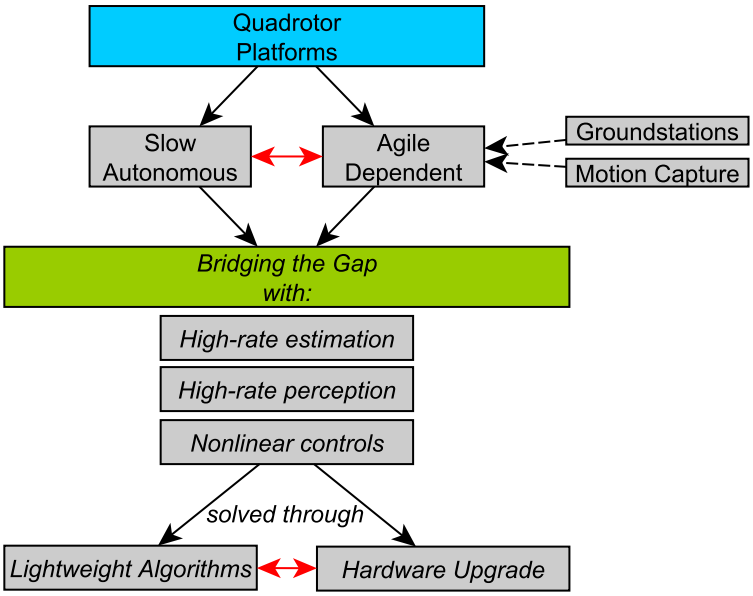


Figure 1-1: Bridging the Gap between Agile and Autonomous Quadrotors: Via lightweight algorithms, or upgraded hardware.

the notion of 'lightweight algorithms' where the low computational complexity shall enable them to be used *onboard* of small platforms as in [3] or [57]. On the other, recent developments in the realm of embedded computing promise to bridge the gap by significantly upgrading the computational hardware.

The main contribution of this thesis is the design and implementation of a small quadrotor platform equipped with a high-performance embedded computing unit that enables the use of high-rate, high-resolution computer vision for visual-inertial estimation, and the implementation of optimal control policies designed using compressed-continuous computation methods based on tensor-train-decomposition.

NVIDIA's computing platform Jetson TK1 offers desktop-like multicore-computing

power with an additional integrated GPU unit. The quadrotor platform built in this thesis is developed from the ground up to accommodate this computing unit without additional overhead in mechanical design. The quadrotor's sensor set allows complete vision-based autonomy. This work also comprises the development of a comprehensive set of software components including software in the loop simulation, safety handling and estimation and control algorithms. To maximize the usability of the platform for further research purposes, a motion capture system is fully integrated into the system. Testing validates the platform's usability.

A use case scenario that guides this thesis compares most adequately to disaster relief scenarios where quadrotors are used to explore environments fast and autonomously, e.g. partly destroyed buildings. In this setup, quickly navigating through openings like doors, windows and holes is an important task.
Figure 1-2 gives an abstract visualization of this task: A *scene target* within a scene is identified and a *target region* for the quadrotor to fly to is derived from the location of this scene target. In this thesis, the scene target is a marker setup that resembles a window.

Plenty of research has been focused on relatively slow maneuvers such as hover and landing as e.g. presented in [61]. On the contrary, flying aggressive trajectories has required the use of fast motion capture systems ([44] or [8]). These systems guarantee high estimation accuracy. A controller's robustness margin therefore needs to cover model inaccuracies only, but less so estimation inaccuracies. Aggressive maneuvers have therefore mostly been confined to lab spaces. We hypothesize that a potential solution with promising outlook is to provide a stochastic optimal controller solved to global optimality for initial conditions in the entire state space. With this controller, not being able to track an "optimal trajectory" does not imply leaving the trajectory's region of attraction. In fact, there is no one precomputed trajectory but an optimal feedback control policy for every point in state space. However, solving for these optimal control policies under nonlinear systems generally suffers from the curse of dimensionality: The complexity scales exponentially with the number of dimensions

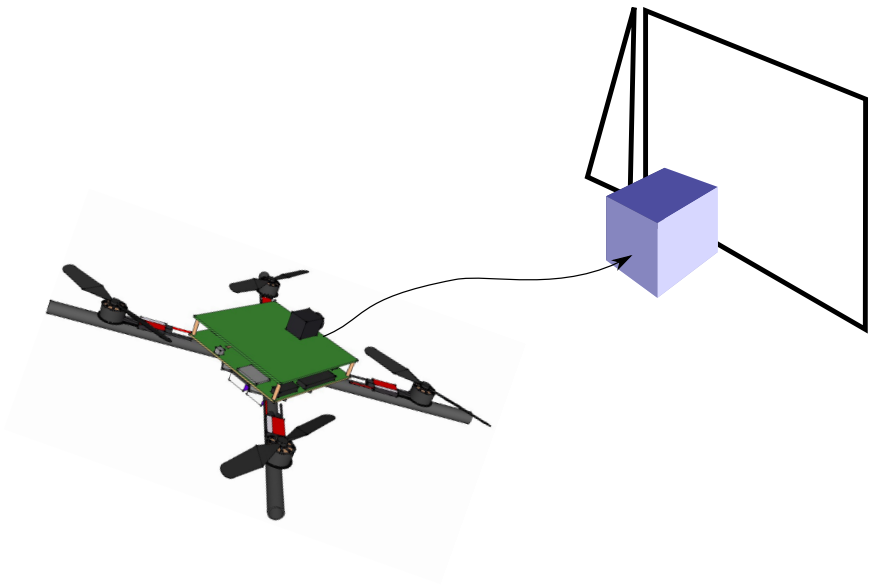


Figure 1-2: Scene Environment with quadrotor, marked window as scene target and target region to fly through (purple).

in the worst case [11]. Novel approaches in applying tensor-train-decomposition-based optimization to dynamic programming of stochastic optimal control problems are able to offer a solution. An optimal control policy for a 7-dimensional stochastic system was presented in [24]. This thesis demonstrates, as a proof-of-concept, the use of these new kind of controllers for quadrotor control under visual-inertial fullstate estimation.

1.2 Related Work

Over the last years research work *on* quadrotors and research *utilizing* quadrotor platforms have been increasing steadily. They are used to demonstrate vision-based estimation algorithms, trajectory planning, mapping and novel control algorithms. This section introduces work related to this thesis.

Hardware Multiple related open source projects exist. Some of them started academically and are now spun off into companies: Gurdan et al. present a small, energy-efficient quadrotor with high-rate sensor data acquisition well suited for re-

search [26]. Now, companies like *Ascending Technologies* offer comparable platforms [4]. The Pixhawk project [43] designed an open-source low-latency, high-performance hardware/controller-suite that builds on top of the LCM-communication framework (LCM is also used in this thesis and discussed in detail in Section 3.2). The *Ardupilot*- and *Dronecode*-projects have grown to a large, industry-supported open source project to foster the deployment of "cheaper, better and more reliable unmanned aerial vehicles" [41]

Estimation Early work on quadrotors as well as experiments involving highly dynamic flight maneuvers relied on external support for both computation and estimation. Expensive motion capture systems provided accurate and fast position estimates ([44] or [8]). With increasing computing power on smaller and smaller devices, researchers started to shift computational load away from these external, offboard resources. Achtelik et al. present a lower cost alternative to motion capture systems using illuminated colored balls mounted on the quadrotor that could be tracked from external cameras [2]. With the nascence of smaller and lighter cameras, novel platforms now oftentimes include vision-based estimation and navigation. The diverse vision-assisted estimation and control concepts can be divided into groups of increasing capabilities to solve high-level tasks:

Image-based visual servoing addresses the task to make a quadrotor hover over or in front of given visual cues and operates on image-level measurements. Chaumette [13] and Hutchinson [33] give introductions to this topic. Approaches for image-based trajectory tracking, taking into account the underactuated dynamics of quadrotors are discussed by several authors [28],[27],[37]. Grabe et al. [25] use onboard computed optical flow to stabilize a quadrotor's position.

Position-based visual servoing, in turn, addresses a comparable task but usually estimates the quadrotor's full state, as e.g. presented in [61] for vision-based take-off, hovering and landing. This relates closely to full visual-inertial estimation that fuses vision-based measurements with accelerometer and gyroscopic data to generate fullstate estimates. Taking away priorly known visual cues from the environment

moves this estimation task into the realm of simultaneous localization and mapping (SLAM), where not only the robot's pose is estimated, but also the position of features in the environment. An introduction and comprehensive survey is given by [10]. Many researchers exploit different variations of SLAM: some approaches do not include running a full SLAM to reduce computational load to enable onboard computation ([52] or [1]) or to enable heuristic real-time obstacle avoidance [3]. This kind of onboard visual-inertial estimation proved to enable accurate figure flying [34].

3D-reconstruction of the environment and mapping applications like [20], [35] or [39] build on top of that. Barry et al. exploit the nonholonomic dynamics of UAVs and present a novel, computationally inexpensive algorithm for stereo matching [7]. These approaches enable steps towards improved scene understanding. More complex platforms include, *e.g.*, laser scanners for better scene understanding during outdoor-indoor transition [49].

Planning and Control Much research has been conducted on designing fast and reliable control and trajectory planning algorithms. Since quadrotors are highly dynamic systems, aggressive trajectories can be realized, but the planned trajectories need to comply with the platform's dynamic constraints. Shen at el present a trajectory planning algorithm for quadrotors that takes into account the trajectory's effect on vision-based estimates [53]. Costante et al. propose a perception-aware path planning approach to find trajectories that best support the vision-based state estimation [15]. An approach for real-time trajectory planning through given waypoints in flat output space under dynamic constraints is discussed in [45]. Quadrotors that are capable of juggling poles are presented by [8]. To increase the agility, a quadrotor with variable pitch has been designed by [16]; they are able to fly multiple flips.

A popular control approach, also summarized in the comprehensive tutorial [42], utilizes a cascaded control scheme where an outer-loop-controller generates a reference *orientation* from translational state errors. This reference orientation is then tracked by an orientation-inner-loop-controller. This approach is also detailed in 6.2 in this thesis. Nonlinear controllers have been designed to work directly on the nonlinear

dynamics and achieve convergence for flight states far away from hover conditions [42] [53].

Very recently, techniques for compressed computation have been applied to stochastic optimal control problems. Horowitz et al. solve an optimal control problem in high dimensions by exploiting linear HJB equations and give a simulated quadrotor example [30]. Gorodetsky et al. introduce a general framework to solve stochastic optimal control problems that does not require linear HJB equations and can handle input constraints [24]. This thesis utilizes this control framework to compute an optimal feedback controller that complies with the nonlinearity, actuator constraints and stochasticity of the quadrotor dynamics.

1.3 Contributions

The main contributions of this thesis can be divided into two aspects: (a) hardware implementation and software architecture design, including estimation and control solutions, and (b) a proof-of-concept demonstration in simulation and experiment of using stochastic optimal controllers, computed through tensor-train-decomposition-based compression techniques.
Specifically,

1. The designed quadrotor platform is equipped with a high-performance embedded computing unit with a GPU that enables the use of fast, high-resolution computer vision, estimation and control *on board*.

2. The provided baseline estimation is set up as visual-inertial estimation and runs fully on board. The developed platform offers sufficient computing reserves to add scene understanding, feature tracking or obstacle detection using standard approaches from e.g. the openCV libraries that can exploit the onboard GPU unit.

3. The modular software design allows to easily switch in and out new estimators, controllers, feature trackers, etc. Due to the use of the LCM-message-handling-

framework low data exchange latency is achieved and signals from high-level tasks down to motor-level commands can be handled by the same infrastructure.

4. A tensor-train-decomposition-based approach is used to synthesize a globally optimal controller that complies with the nonlinear, stochastic quadrotor dynamics and actuator constraints.

5. Simulation studies and experiments present a proof-of-concept for the usability of these recent developments in utilizing compressed computation techniques for controller synthesis.

1.4 Organization

This thesis is organized as follows:
Chapter 2 details the hardware and electronics design of the quadrotor platform. Design considerations and selection of components are discussed. Chapter 3 describes the software architecture and introduces the roles of various system components like estimators, controllers and vision-related software pieces. The setup reveals the ease of substituting in new or additional software parts. It also presents the communication architecture between quadrotor, motion capture system and ground station.

The mathematical models for the quadrotor's dynamics are given in Chapter 4, including actuator and battery dynamics. A simulator features all these phenomena.

Chapter 5 presents the visual detection of a scene target and two estimation algorithms to generate fullstate estimates. Both use priorly known visual features in the environment and onboard accelerometer and gyroscopic measurements. The estimators' performance is evaluated with experimental data.

Chapter 6 describes a control structure that cascades the underactuated dynamics into a position- and an orientation subsystem. Two controllers are discussed for the position subsystem: first, a commonly used PD controller that acts on linearized

dynamics. Second, a globally optimal, stochastic optimal controller that takes into account the nonlinearity of the quadrotor dynamics and actuator constraints. In this chapter, the controller performance is evaluated in simulation.

An experimental performance evaluation of the overall integrated systems is presented in Chapter 7. It shows the platform's hover capabilities and demonstrates that tensor-decomposition-based nonlinear stochastic optimal controllers can control the quadrotor to approach a target region.

Finally, Chapter 8 summarizes this thesis, discusses limitations and suggests starting points for future work.

Chapter 2

Quadrotor Hardware Implementation

This chapter describes the quadrotor's hardware design and the electronic architecture. The platform was built entirely from the ground up using a variety of off-the-shelf components from both hobbyist RC-stores and specialized manufacturers like *PointGreyResearch*. This approach allowed to build an optimized platform to accommodate the *NVIDIA TK1* computing unit and the required sensor setup for full autonomy without additional mechanical overhead. The following sections cover the overall architecture, the computing unit, the sensor setup, the actuation system and the motion capture system which was used to acquire ground truth data.
In addition, for subsystems like the camera with static, geometric parameters, the identified parameters are discussed in this chapter.

2.1 System Architecture

This section introduces the quadrotor's overall design and architecture.
The quadrotor carbon-fiber frame *Turnigy Talon V2* is an off-the-shelf quadrotor frame and was chosen for its capability to house the *TK1* computing unit in its center. Additional electronic components were installed on a PCB board above the *TK1*. Figure 2-1 shows the current version of the quadrotor platform. In this flight-ready configuration, with all components mounted including batteries, a safety cage and motion capture markers, the platform weighs 1.28kg. Its inertia values, determined

by a bifilar pendulum experiment, and propeller geometry data can be found in Table 2.1.

Figure 2-1: Quadrotor Build

Table 2.1: Quadrotor Hardware Parameters

mass m	1.28	kg
inertia (J_{xx}, J_{yy}, J_{zz})	$(6.9E^{-3}, 7.0E^{-3}, 12.4E^{-3})$	Ns^2
propeller diameter d_p	0.165	m
propeller positions $\mathbf{r}^B_{\text{prop}}$	$\begin{bmatrix}\pm 0.117 & \pm 0.117 & -0.012\end{bmatrix}^T$	m

A standard Wifi module, *Intel 7260*, provides the Wifi connection to a ground station computer for real-time data visualization and user-input capabilities. No real-time-critical data is streamed between the ground station and the quadrotor.

On the sensor side, the flight scenario that is being considered for this thesis requires an IMU for high-bandwidth orientation estimation and a monocular camera for

lower-rate localization and position estimation. On the actuation side, four brushless motors were used to provide thrust with off-the-shelf propellers. For data-collection and safety purposes, it is possible to switch between autonomous flight mode and human-controlled flight mode. A separate microcontroller, an *Arduino nano*, generates the PWM signals that command the motors. This setup enables the operator to use a switch on the RC controller to switch between motor commands generated by either the onboard computing unit or by the entirely self-contained, off-the-shelf quadrotor flight controller (*AfroFlight Naze32*).

To prevent bandwidth and interference complications in data transfer all components are connected through individual buses to the central *TK1* computing unit. Figure 2-2 details the connections.

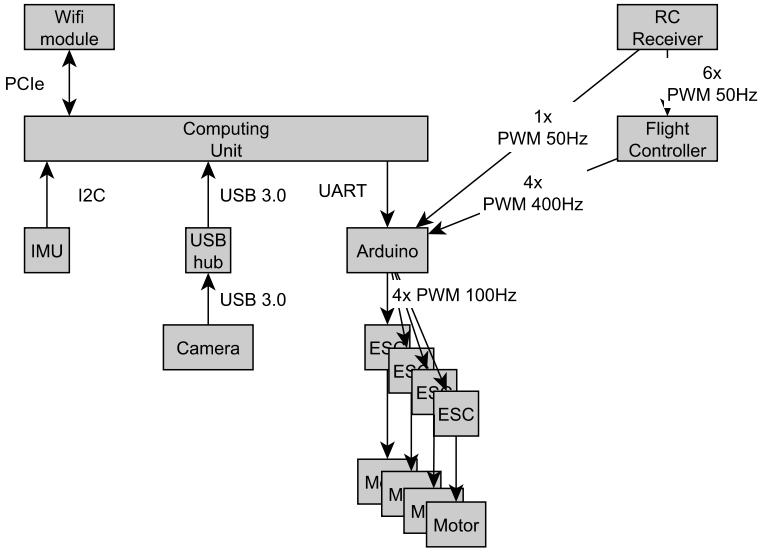


Figure 2-2: Electronics Diagram

2.2 Computation Unit

While there are many options to choose from for onboard-computing power, the *NVIDIA TK1* computing unit poses a noticeable step-up in computational capabilities in comparison to often used microcontrollers. The *TK1* features an ARM-Cortex A15 Quadcore CPU with 2.32 Ghz, 2Gb RAM, 16GB memory, an *NVIDIA* Kepler

GPU with 192 cores and various interfaces like HDMI, I2C, UART, USB 3.0 or Ethernet. It delivers up to 300 gflops. Its power consumption rarely exceeds 5W, but can go up to 15W under highly demanding tasks. These specifications allow for complex computations at high rates. The manufacturer showcases feature tracking at 720p at 40 fps. This thesis exploited this computational power to run high-resolution computer vision for pose estimation and real-time controller optimization at a sufficiently high rate to allow for agile maneuvering of the quadrotor. The TK1's low weight of about 130g allowed to mount it onto the quadrotor, thus running all processes *on board* the platform.

The platforms's power supply is split into two separate circuits as shown in Fig. 2-4: a 4S Lithium-Polymer-battery provides power to the computing unit as well as the USB-hub that powers the camera, the *Naze32* flight controller and the *Arduino*; a 3S Lithium-Polymer-battery powers the motors. This setup minimizes electrical noise induced by power spikes from varying motor load.

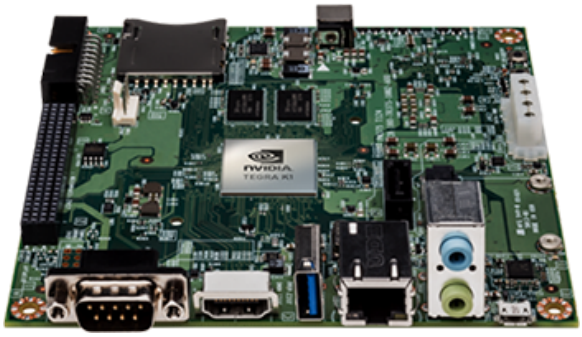


Figure 2-3: *NVIDIA*'s Jetson TK1 [47]

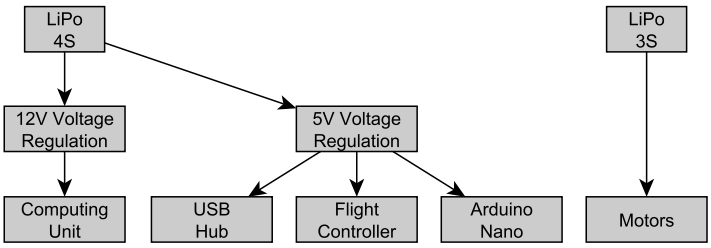


Figure 2-4: Power Supply Diagram

2.3 Sensor Setup

This section discusses the details of the sensor setup. An IMU provides measurements for angular rates and acceleration and can therefore provide high-bandwidth information for position and orientation estimates. The camera is used for lower-rate updates of pose estimates with respect to an identified scene target.
Future versions of the platform will contain a battery voltage sensor and possibly measurements of motor speeds. Programmable ESCs like the VESC [58] could offer a handy solution to this (although the current version is too large).

2.3.1 Inertial Measurement Unit

The current configuration features a BOSCH BNO055 IMU on a Sparkfun breakoutboard. The IMU-chip offers onboard filter capabilities. Using a custom-extended version of [50] the gyroscope's and accelerometer's bandwidth were set to 41Hz. The angular rates and acceleration are then read through a dedicated thread on the *TK1* computing unit via an I2C-connection at 100Hz. It turned out that the propellers induce significant mechanical vibrations into the frame. With some IMUs this caused extremely noisy accelerometer readings and sometimes even a shift in accelerometer bias. The use of damping mounts to dampen out mechanical vibrations greatly improved the sensor data.

2.3.2 Camera

The platform's camera, a *Flea 3 FL3-U3-13Y3M-C* from *Point Grey Research* (Figure 2-5) offers up to 150 FPS at 1280x1024 pixels. While for pure navigation purposes a high-resolution high-rate camera, as chosen for this project, might not be necessary, it does improve the localization accuracy with respect to a visual scene target. Importantly, it features a global shutter, simplifying the use of raw images for estimation purposes.
It is connected via USB 3.0 to a powered, dedicated USB-3.0 hub.
The installed fish-eye lens (DSL219D-650-F2.0 [55]) provides a 178 deg field of view.

Figure 2-5: *PointGreyResearch*'s Flea 3 Camera [48]

[51] provides a model and toolbox to calibrate omni-directional and fish-eye cameras. It maps a 2D-image point $\mathbf{p}^{\text{fish}} = [u\, v]^T$ to a 3D-unit-vector $\mathbf{m}^{\text{cam}}$ emanating from the camera system's 'single effective viewpoint'. All real-world points lying on this ray result in the same $\mathbf{p}^{\text{fish}}$.

Assuming perfectly aligned lenses, [51] shows that

$$\mathbf{m}^{\text{cam}} = \begin{bmatrix} X \\ Y \\ Z \end{bmatrix}^{\text{cam}} = \begin{bmatrix} u \\ v \\ f(u,v) \end{bmatrix}$$

With a perfectly symmetric lens, $f(u,v)$ can further be simplified to $f(u,v) = f(\rho)$ with ρ being the pixel-distance of a point $\mathbf{p}^{\text{fish}}$ from the image center. The function $f(\rho)$ is approximated by an n-th order polynomial. The toolbox suggests a default order of 4. In our experiments this choice of parameters yielded reasonable results. The calibration run for the lens used on the quadrotor resulted in parameters

$$\alpha_0 = -5.518E^2 \qquad \alpha_1 = 0 \qquad \alpha_2 = 8.372E^{-4} \qquad \alpha_3 = -6.474789E^{-7} \qquad \alpha_4 = 1.236E^{-9}$$

following

$$f(\rho) = \alpha_0 + \alpha_1\rho + \alpha_2\rho^2 + \alpha_3\rho^3 + \alpha_4\rho^4$$

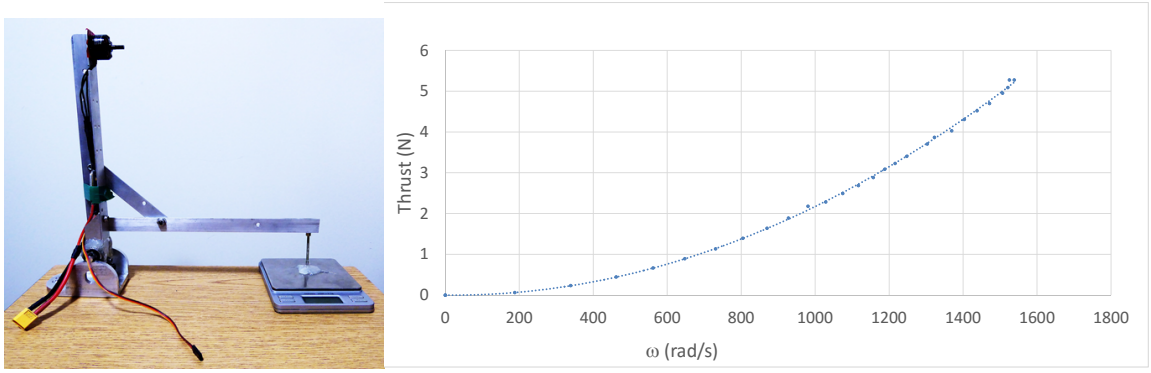
2.4 Actuator Characteristics

The quadrotor's four propellers are driven by *Sunnysky 2210TS* brushless-DC-motors. They are commanded by *ZTW Spider Opto* electronic speed controllers (ESC). These ESCs are capable of handling up to 30A and are controlled using PWM-signals with a length between 1ms and 2ms at up to 400Hz update-rate. The power supply is separate from the supply to the computation and sensing architecture.
Standard 6" quadrotor propellers are installed.

2.4.1 Propellers

Using a custom-made rig pictured in Figure 2-6 the relation from propeller speed/motor speed to generated thrust was identified. Motor speeds were measured using a *Extech* digital laser tachometer. While this identification does not account for in-flight aerodynamic effects caused by, *e.g.*, relative wind velocities, it does not suffer from the influence of ground effects because the design features a horizontal propeller axis, resulting in horizontal air flow.
The ESCs were commanded through the same software framework that was used to later fly the quadrotor in autonomous mode.

The recorded data follows a quadratic relation, thus complying with the classic



(a) Rig for Static Thrust Estimation

(b) Data Points Static Thrust

Figure 2-6: Propeller Parameter Identification

model for propeller thrust:

$$T = \alpha_T \omega^2$$

with T being the thrust generated and ω being the motor speed in rad/s. The identification resulted in

$$\alpha_T = 2.26E^{-6}$$

2.4.2 Speedcontrollers and Battery

The software flight controllers output desired motor speeds which need to be converted into PWM-signals with pulse length κ microseconds that can be sent to the ESCs. An identification of the relation between κ and motor speed is therefore necessary.
This relation is, however, heavily influenced by a decreasing battery charge state since the terminal voltage drops not only with load (which is covered by the identification procedure), but also over time with an increasingly depleted battery. The ESCs do not compensate for this effect.
To mitigate this effect, the identification is conducted with a fully charged battery, quick enough to not noticeably deplete the battery. ESCs are oftentimes tuned to reproduce a near linear relation between generated thrust and κ (in steady-state). Since $T \propto \omega^2$, an approximate relation $\sqrt{\omega} \propto \kappa$ could be expected. Indeed, the resulting identification revealed a clear square-root relation for lower motor speeds, and a rather linear relation for high speeds (Figure 2-7).
This procedure neglects the fact that three *additional* motors will draw current from the same battery during flight, especially during high motor speeds - further reducing the battery's terminal voltage, which, in turn, additionally reduces the thrust and renders the identification less accurate. Section 4.2.3 sets up a parametrized model

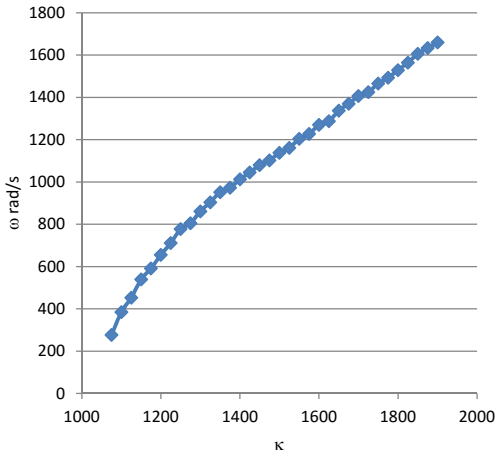


Figure 2-7: Data Points PWM κ to Motor Speed ω: Near-square root-relation

to account for this effect and presents the identified values.

2.4.3 Motors

Neglecting electrical time scales, electric motors generally follow *approximate*, first-order delay dynamics. Figure 2-8 shows a short-time-Fourier-transformed (STFT) audio recording of an ESC-controlled motor being ramped up from below-hover to take-off motor speed. It reveals that this first-order delay dynamics persist with an ESC in the loop. 4.2.4 elaborates on the modeling of this effect.

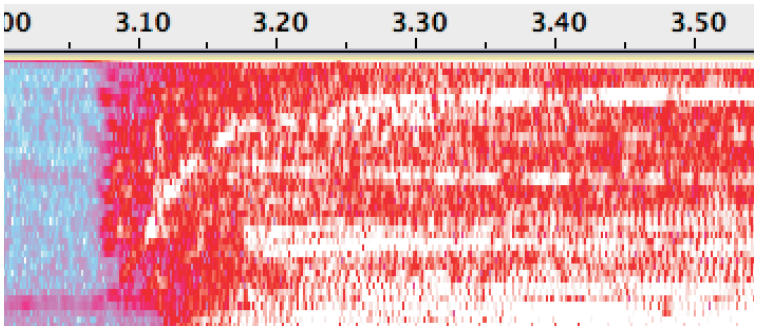


Figure 2-8: Short-Time-Fourier-Analysis of Motor Sound on Stepinput: Plot reveals first-order delay dynamics.

2.5 Motion Capture System

A motion capture system provided position estimates for initial flight experiments and ground truth to evaluate onboard position estimates. This system was newly set up for these experiments. The *OptiTrak*-System [46] features 6 infrared *Flex 13W* cameras, capable of providing full 6D-pose estimates of rigid-bodies at up to 360Hz. Infrared-reflective markers are mounted onto the drone in an asymmetric way to provide unambiguous identification to the *OptiTrak* pose-estimation-algorithm. Fishing nets were used to build a safety cage around the flying area. Figure 2-9 shows the setup of the motion capture area.

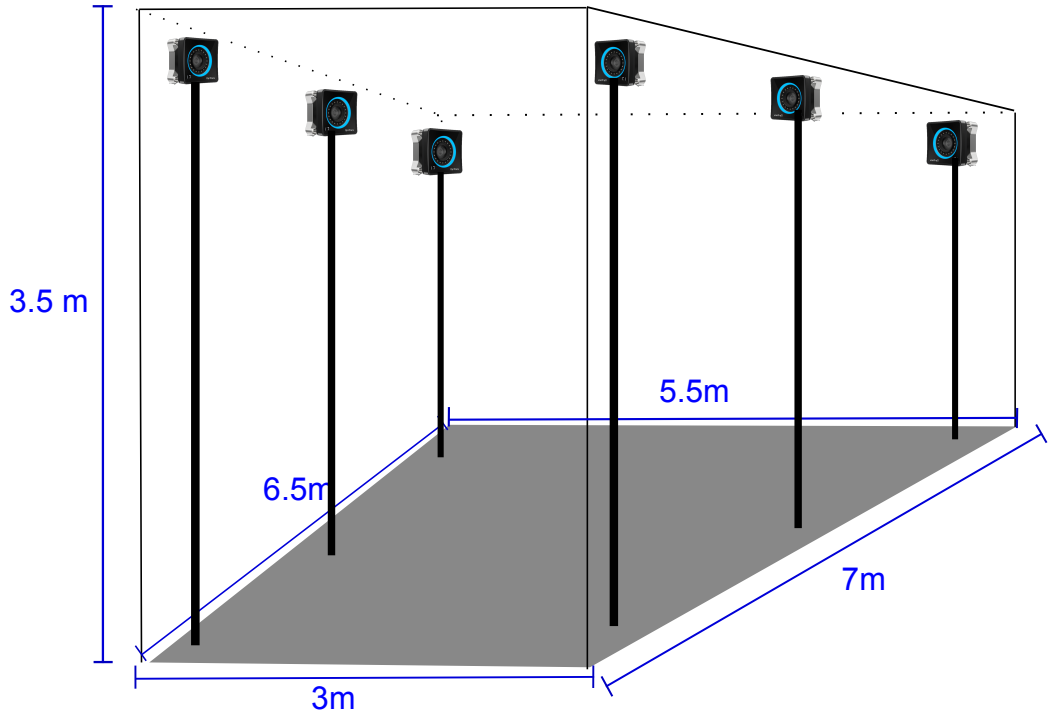


Figure 2-9: Motion Capture Area

Chapter 3

Software Implementation

This chapter discusses the software architecture. Since the onboard computing unit is capable of running of full Linux system (Linux4Tegra L4T), the software design was inspired by architectures that have previously been designed to run on standard desktop computers. All code was written in C++ and the *PODs*-framework [5] was used as software development guideline. In the remainder, threads - or dedicated processes - are called *PODs* themselves. These resemble *nodes* from the known RobotOpertingSystem ROS.

3.1 Multithread Architecture

Full autonomy requires various computational tasks to be completed simultaneously: Sensor data is acquired, filtered, combined and augmented through estimators; controllers decide on the desired action to be executed by actuators. These different processes have been distributed into separate PODs. They exchange data by passing messages. Again, this design closely resembles a ROS-design. The message passing system, however, is implemented using LCM. Section 3.2 gives a brief description. Figure 3-1 illustrates the resulting network of PODs and the required messages being passed between them. The PODs are run at three different update rates: Visual feature detection runs at 50Hz, IMU data acquisition, estimation, control and safety checks (*watchdog*-POD) at 100Hz and the *motorCommander* at 200Hz.

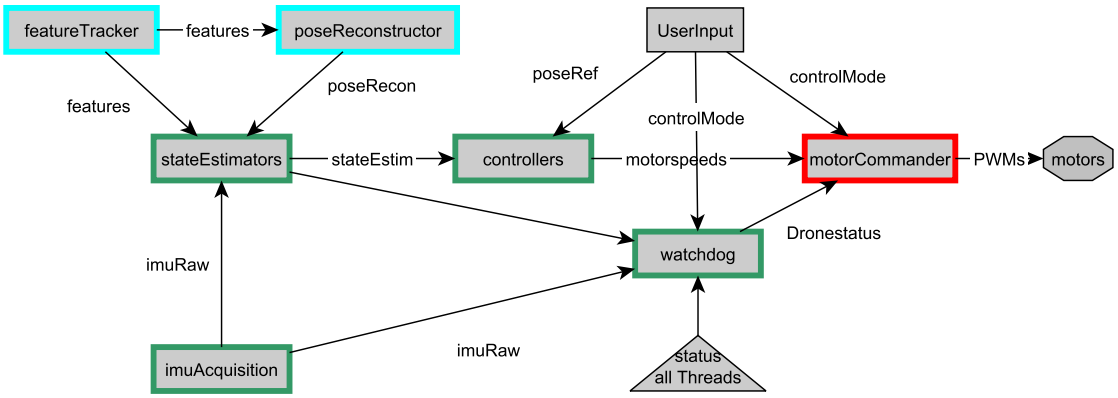


Figure 3-1: Multithread Architecture with Messages and Update rates: cyan (50Hz), green(100Hz), red(200Hz).

The *featureTracker* detects the scene target and publishes calibrated 3D rays emanating from the camera to the features. The detection algorithm is detailed in Section 5.1.1.

The *poseReconstrutor* reconstructs the full 6D-pose, consisting of orientation and position. It uses a standard openCV-pnp-algorithm and is discussed in more detail in 5.1.2.

The *stateEstimators* take IMU-raw data, visual features and potentially the visually reconstructed pose to estimate the full 12-dimensional state of the quadrotor. Details can be found in Chapter 5.

The *controllers* convert the current state estimate into desired motor speeds. Chapter 6 elaborates on the design of various controllers.

The *watchdog* runs safety checks to shut down the quadrotor in case of emergencies or software faults. Figure 3-3 illustrates the safety checks with a state machine.

The *motorCommander* selects which controller's desired motor speeds are being converted into PWM-values that are then sent out to the *Arduino nano* that generates the PWM-signals to command the motors via the ESCs.

3.2 Thread Communication using LCM

The communication between different PODs is handled by the *Lightweight Communications and Marshalling* framework (LCM) [32]. The LCM-libraries were developed by MIT's DARPA Urban Challenge team to provide a high-bandwidth, low latency tool for message passing with bindings to various programming languages. Similarly to ROS, PODs can publish messages to and receive from user-specified channels using the LCM-libraries. This thesis chose LCM over ROS to avoid computational overhead and minimize latency in message handling.

3.3 Design of Base Thread

Every POD thread instantiates a worker object from a class that is derived from a POD-base-class *podBase*. These worker objects store data needed for the computations they do. For example, the full-pose position controller thread *controllerPDPose* instantiates a *controllerPDPose*-object derived from the *podBase*-class. This section describes how the base class *podBase* offers functionality to do computations and receive and publish messages in a multithreaded way.
Figure 3-2 shows its UML diagram.
The member variables:

- *podName* represents the name of the POD. It is used to publish a status message about its "health" on a channel "status*podName*".
- *onMsgRecompuation* is a flag to enable rerunning the POD's computation-method on receiving any new message.
- *statusPOD* stores the POD's status.
- *statusWatchdog* represents the POD's local copy of the watchdog's current status. The *podBase*-class constructor autosubscribes every POD to the message channel that contains the watchdog-POD's status; *i.e.*, every POD knows about the watchdog's status.

- same for *statusDrone*.

- *callInterval*: the interval which the POD's computation loop is called at by a gLib-loop.

- *timestampJetsonLastComputation*: a timestamp when the most recent computation loop was completed.

- *computationInterval*: the duration of the most recent computation.

- *lcm*: the lcm-object. This member object offers LCM-functionalities like subscribing or actively listening to an LCM channel.

- *messageAdmin*: A map that stores information about all the messages that the POD is subscribed to. It maps from the channel name to a *messageContainer* object which is derived from a virtual base class *messageContainerBase*. This allows to iterate through the map, thus iterating through all messages that the POD is subscribed to and *e.g.* executing operations on all messages.

The member methods:

- *subscribe*: subscribes the POD to a channel. The POD will use *receiveIntervalExpected* to check whether it is up-to-date on messages from this channel. A new messageContainer with key "channel" is added to the *messageAdmin*. This function also calls the actual LCM-subscribe-function and passes the handler-method that is called upon receiving a new LCM-message. This handler method is per default the *handleMessage*-function.

- *unsubscribe* unsubscribes the POD from an LCM-message-channel by removing the corresponding item from the *messageAdmin*-map and by calling the LCM object's unsubscribe-function.

- *initComputationInterval* initializes variables related to checking the execution time of the computation method.

- *listen* is a simple while loop that calls LCM's handle function which makes it listen for new messages coming in over the LCM network. LCM's handle() waits until it gets a new message and then calls the function handle *handlerMethod* which was passed on subscribing an LCM object to a channel. The *listen* function is run in a separate, dedicated thread. This requires taking care of multithreaded variable access.

- *handleMessage* is the default handler method that is being called by the LCM object on receiving a new message. It blocks the access to the *messageAdmin* and stores the new message in the corresponding *messageContainer* in the *messageAdmin*. It also updates the meta information about receive intervals stored in the *messageContainers* by calling *updateMessageLastReceived*. In case *onMsgRecomputation* is enabled (in general or for this specific channel) and the POD's computation loop is currently not running, it runs the computation loop.

- *gtimerFuncComputation* is the static method whose handle is fed into a gLib-timer function. On being called it first copies all messages stored in the *messageAdmin* to the POD's corresponding member variables by calling *updateMessageMemberVariables*. *UpdateMessageMemberVariables* uses the *messageAdmin* to iterate through all messages that the POD is subscribed to and make them update their corresponding POD's member variables. Then, the POD's specific doComputation-function is run, executing the actual computations. After being done it updates the computation interval the computation took. With this setup, the doComputation-method can simply use the worker object's member variables to access the most recent LCM messages that the POD received. Mutex-locking the access to the messageAdmin and the member variables only occurs during the copy and update-process. This setup handles the fact that the listener-method runs in a different thread simultaneously and has write-access to the messages stored in the *messageContainers*.

- *gtimerFuncStatusPod* is the static function fed into a second gLib-loop. It calls the POD's specific *updateStatus*-function after copying the most recent

messages from the messageAdmin to the local member variables by calling *updateMesageMemberVariables*. *updateStatus* checks if messages are up-to-date by calling *checkMessageUptodate* and implements POD-specific health checks before it publishes the POD's status by calling *publishStatus*.

On the messageContainerBase-class:
This abstract class provides a base class that holds meta information about a subscribed channel and the most recent message received from that channel.

- *timestampJetsonReceived* stores when a new message was received last from this channel.

- *receiveIntervalExpected* stores the interval which new messages from this channel are expected to arrive at

- *subscription* holds the pointer to the lcm-subscription that corresponds to the message. This objects can be used to *e.g.* unsubscribe.

- *messageReceived* is typeless pointer to the most recently received message.

- *updateMessageMemberVariable()* is a virtual method that is implemented in a child class and calls the child's *updateMesageMemberVariable()*. This function copies the message in *messageReceived* into the corresponding member variable of the POD object while mutex-locking the access.

A templated class derived from *messageContainerBase* is then used to instantiate messageContainers that "know" about the actual type of the message (instead of merely having typeless void pointers to stored messages) and that contain a *messageMemberVariable* pointer that points to the POD's member variable that receives a copy of the most recently received corresponding message and can be used to do computations with.

3.4 System Status and Safety Handling

Since the overall system has multiple subsystems, each with different initialization procedures, a system status variable is introduced. This *droneStatus* is being used to handle both the initialization status and the emergency status. The emergency status aspect of this implementation can be considered the software-sided safety handing, in addition to the experimental setup (the net cage) and the electronic safety switch using a separate Arduino to switch between human-RC-controlled mode and autonomous mode.

Figure 3-3 illustrates the state machine with transition conditions and *motorCommander* (the POD that communicates with the ESCs through the Arduino) action taken in each state. The state machine runs in the *watchdog*-POD.

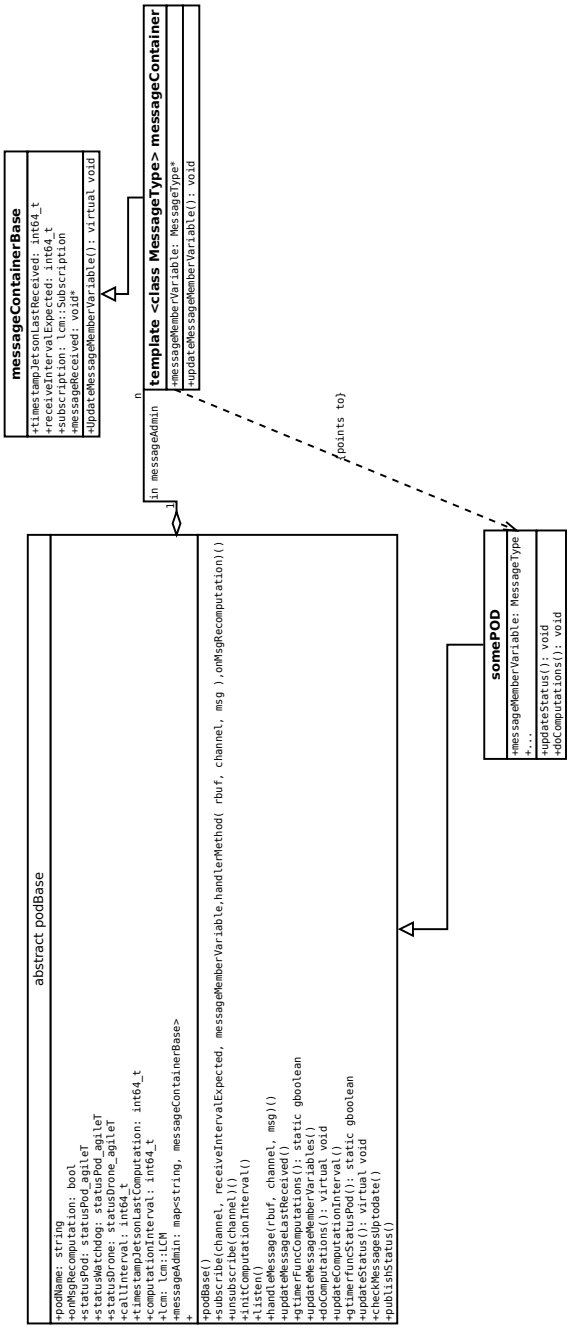


Figure 3-2: *podBase*-Class UML diagram

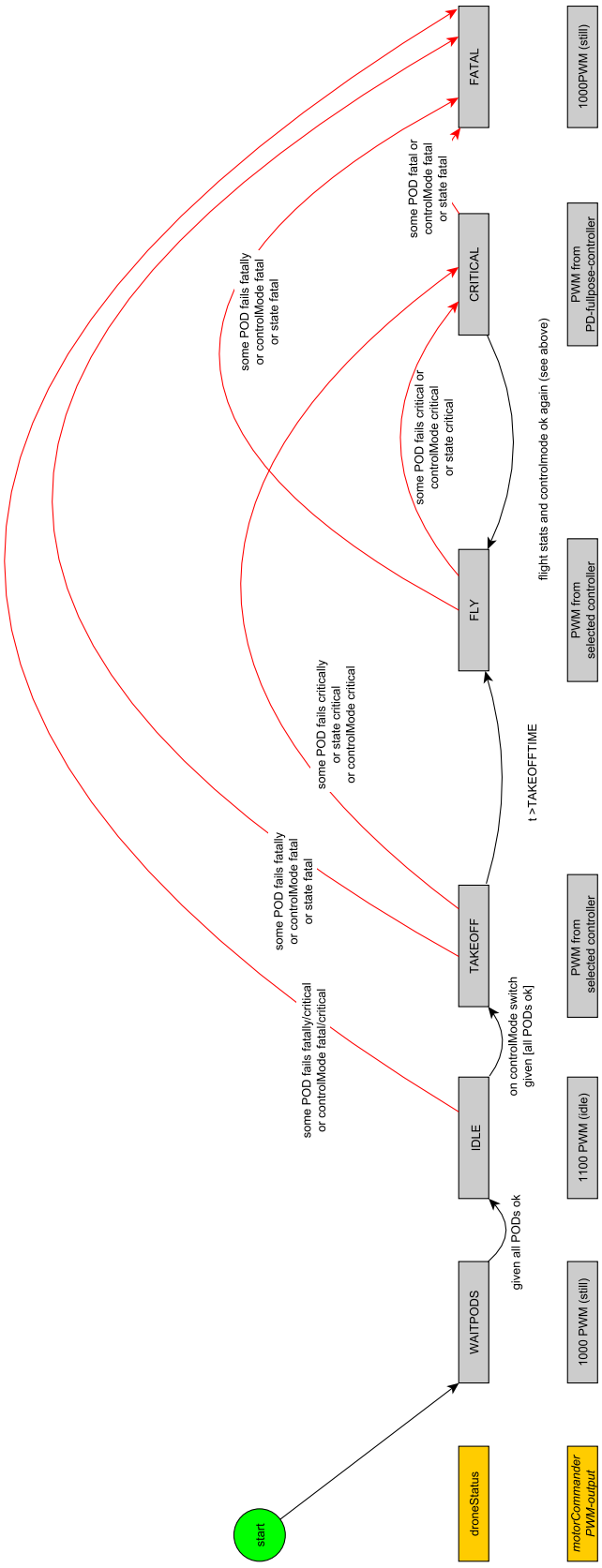


Figure 3-3: Quadrotor Status: State Machine

3.5 Remote Communication

This section addresses the remaining need for remote communication for a quadrotor that does not require offboard computation or processing.

The quadrotor sends all its flight telemetry data through an LCM tunnel using a standard Wifi module to a central router.

To acquire ground truth data, a motion capture system was installed. Its software runs on a separate Windows computer and sends pose estimates (position, orientation and translational velocities computed with filtered finite differences) as UDP multicast packages over a Gigabit Ethernet to the central router. Note that the quadrotor can receive these packages. It timestamps them and converts them into LCM messages.

A groundstation desktop computer was used to receive all flight telemetry data through an Ethernet LCM tunnel for visualization purposes. Additionally, user-input for different flight-modes or target position can be streamed to the quadrotor.

To manually fly the drone for data-recording and safety purposes the quadrotor features the *AfroFlight Naze32*, an off-the-shelf flight controller. This flight controller receives a desired quadrotor orientation from an *FrSky Taranis Plus* RC-flight controller over radio signal.

Based on one channel of the radio signal, the Arduino switches between creating the PWM-signal from PWM-values received from the *TK1* computing unit or received from the *Naze32* Flight Controller. This switch-channel is linked to a button on the RC control.
Figure 3-4 illustrates this communication architecture.

3.6 Simulation and Visualization Environment

For efficient testing and tuning of control parameters, a simulation environment was setup in MATLAB/Simulink. [36] was extended to model additional phenomena (discussed in Chapter 4). [36] is an extended version of Peter Corke's toolbox [14].

This simulator was then translated into c-code by means of Simulink's auto-code-

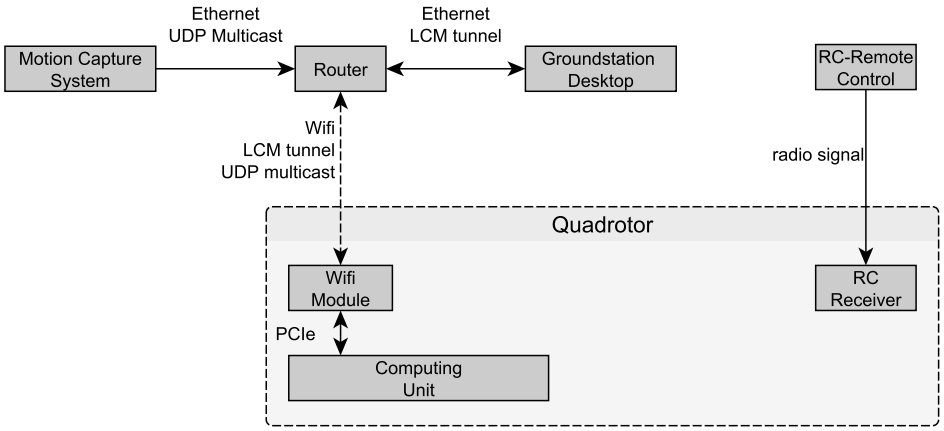


Figure 3-4: Remote Communication Diagram

generation and interfaced with a POD. This POD feeds the simulation with motor speed inputs and publishes simulation outputs like the full, simulated quadrotor state, sensor readings and visual features. The full software framework including estimation and control PODs can therefore simply be run "simulator in the loop".

LCM provides real-time viewers to display data published over the LCM network. For analysis purposes, this data can be converted into MATLAB format using a tool provided by the libbot-library [31]. A libbot-based viewer-POD subscribes to relevant quadrotor telemetry and visualizes the actual and estimated quadrotor states together with thrust in 3D in real-time (Figure 3-5). Note that the channels subscribed to to acquire the visualization data can either be generated from a simultaneous simulation, or from recorded, real in-flight data.

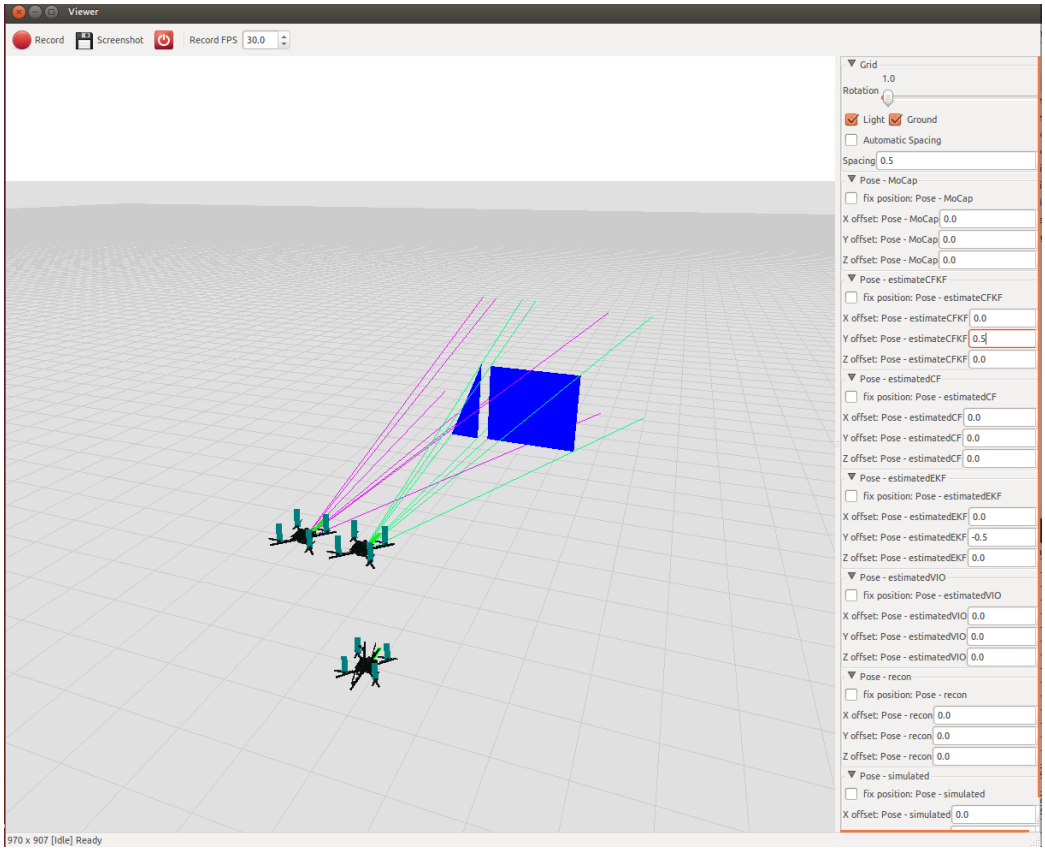


Figure 3-5: 3D Viewer for Flight Telemetry: Simulated state (green), and EKF-estimated state (purple; with offset for better visibility).

Summary

This chapter described the software architecture and introduced the roles of various system components like estimators, controllers and vision-related software pieces. The setup revealed the ease of substituting in new or additional software parts. It also presented the communication architecture between quadrotor, motion capture system and ground station.

The next chapter presents the mathematical models for the quadrotor's dynamics including actuator and battery dynamics.

Chapter 4

System Dynamics Modeling

This section introduces the mathematical models used to describe the quadrotor's dynamics. While these models can be arbitrarily complex to account for various effects, the following two models were chosen: The simplest model that still fully describes all six degrees of freedom of a quadrotor. This model will be used to derive controllers. The model used for simulation takes into account additional phenomena like battery effects, motor dynamics and simple aerodynamics.

4.1 Quadrotor Model

4.1.1 Coordinate Frames

Figure 4-1 shows the major coordinate systems used in the derivations of the dynamic equations:

Let $\mathbf{r}^I$ be the position of the quadrotor's center of mass in an inertial global coordinate frame

$$\mathbf{r}^I = \begin{bmatrix} x^I & y^I & z^I \end{bmatrix}^T$$

$\mathbf{v}^I$ denotes the quadrotor's velocity with respect to the global coordinate frame ex-

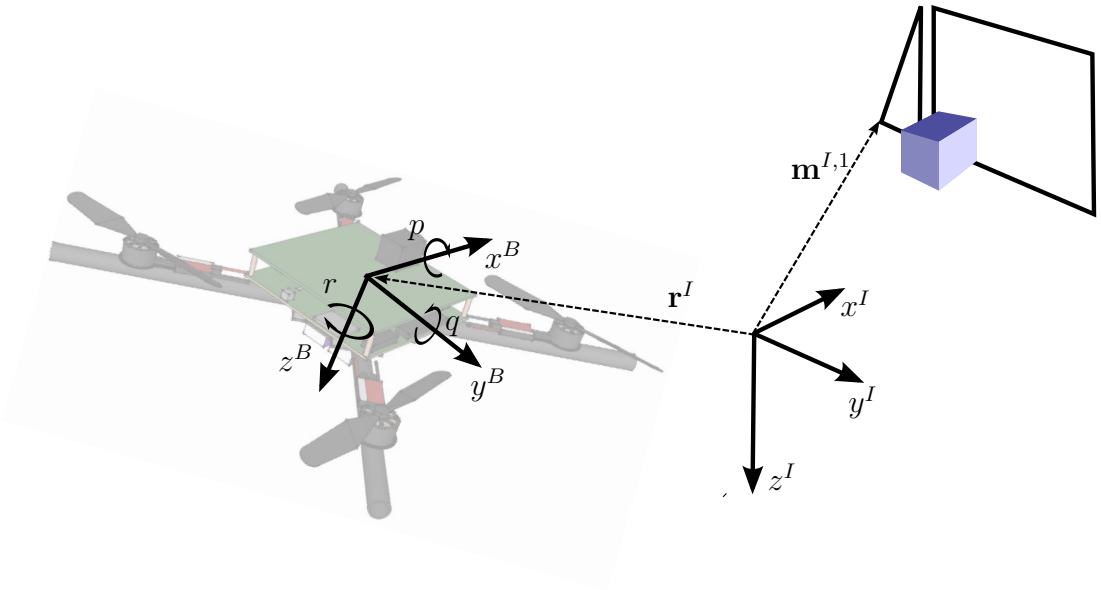


Figure 4-1: Coordinate Frames with Marked Window and Target Region

pressed in the global frame

$$\mathbf{v}^I = \dot{\mathbf{r}}^I = \begin{bmatrix} \dot{x}^I & \dot{y}^I & \dot{z}^I \end{bmatrix}^T$$

The origin of a body-frame coordinate system $\{x^B, y^B, z^B\}$ is fixed to the quadrotor's center of mass and the axes line up with the principal axes as illustrated.
The quadrotor's orientation η is expressed in euler-angles (yaw, pitch, roll)

$$\eta = \begin{bmatrix} \psi & \theta & \phi \end{bmatrix}^T$$

relating to a rotation first about the global Z axis (yaw ψ), then a pitch-rotation θ about the new y-axis, followed by a roll-rotation ϕ about the new x-axis.
The matrix $\mathbf{W}^{-1}$ transforms the body-angular rates

$$\Omega = (p, q, r)^T$$

about local x-y-z-axes to euler-rates

$$\dot{\eta} = \begin{bmatrix} \dot{\psi} & \dot{\theta} & \dot{\phi} \end{bmatrix}^T$$

Let $\mathbf{T}^B$ be the total thrust generated by the rotors expressed in the body frame. Let τ_{yaw} be the total torque about the (body-frame) z^B-axis resulting from propeller drag. Motor speed acceleration is neglected. Let τ_{pitch} and τ_{roll} be the resulting torque about y^B-axis and x^B-axis, respectively. Let $\mathbf{J}$ be the quadrotor's inertia expressed in the body frame.

4.1.2 Basic Quadrotor Model

This section now introduces the simplest dynamic model to describe the quadrotor's dynamics with its full 6-DOF.
Let the state $\mathbf{x}$ be composed of global position $\mathbf{r}^I$, orientation η, global velocities $\mathbf{v}^I$ and body-frame angular rates $\mathbf{\Omega}$:

$$\mathbf{x} = \begin{bmatrix} \mathbf{r}^I & \mathbf{v}^I & \eta & \mathbf{\Omega} \end{bmatrix}^T$$

Derived from from a standard Newtonian approach with total thrust $\mathbf{T}^B$ acting on the center of mass, and body-frame torques τ_i as the plant inputs, the quadrotor dynamics result in (as derived in *e.g.* [21], with neglected gyroscopic effects)

$$\begin{aligned} \dot{\mathbf{r}}^I &= \mathbf{v}^I \\ \dot{\mathbf{v}}^\mathbf{I} &= \mathbf{G}^I + \mathbf{D}_B^I(\eta)\frac{\mathbf{T}^B}{m} \\ \dot{\eta} &= \mathbf{W}^{-1}\mathbf{\Omega} \\ \mathbf{J}\dot{\mathbf{\Omega}} &= \begin{bmatrix} \tau_{roll} \\ \tau_{pitch} \\ \tau_{yaw} \end{bmatrix} - \mathbf{\Omega} \times \mathbf{J}\mathbf{\Omega} \end{aligned}$$

with $\mathbf{D}_B^I$ denoting the rotation matrix from body-frame axes to the inertial frame axes, either as function of euler-angles η

$$\mathbf{D}_B^I(\eta) = \mathbf{D}_\psi \mathbf{D}_\theta \mathbf{D}_\phi$$

where

$$\mathbf{D}_\psi = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ -\sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \mathbf{D}_\theta = \begin{bmatrix} \cos(\theta) & 0 & \cos(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \quad \mathbf{D}_\phi = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}$$

or as function of a quaternion $\mathbf{q}$ that represents the same orientation:

$$\mathbf{D}_B^I(\mathbf{q}) = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1q_2 - q_3q_0) & 2(q_1q_3 + q_2q_0) \\ 2(q_1q_2 + q_3q_0) & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2(q_2q_3 - q_1q_0) \\ 2(q_1q_3 - q_2q_0) & 2(q_2q_3 + q_1q_0) & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix}$$

The quaternion representation follows

$$\mathbf{q} = \begin{bmatrix} q \\ \mathbf{e} \end{bmatrix} = \begin{bmatrix} q_0 \\ \vdots \\ q_3 \end{bmatrix}$$

where $q = q_0$ represents the scalar and $\mathbf{e}$ the complex part of the orientation quaternion. Note that $||\mathbf{q}|| = 1$.

Furthermore,

$$\mathbf{W}^{-1} = \begin{bmatrix} 0 & \frac{\sin(\phi)}{\cos(\theta)} & \frac{\cos(\phi)}{\cos(\theta)} \\ 0 & \cos(\phi) & -\sin(\phi) \\ 1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta) \end{bmatrix}$$

It is to be noticed that this system is underactuated since it has six degrees of freedom but only 4 actual control inputs (the four motors speeds ω_i. The conversion from motor speeds ω_i to $\begin{bmatrix} \mathbf{T}^B & \tau_{\text{roll}} & \tau_{\text{pitch}} & \tau_{\text{yaw}} \end{bmatrix}^T$ is detailed in 4.2.2). However, due to coupling effects, any position $\mathbf{r}^I$ and yaw-angle ψ can be achieved as steady-state. Differentially flat models use exactly these four coordinates $[\mathbf{r}^I\ \psi]$ as output space [45].

Parameters for the dynamic equations above can be found in Table 2.1.

4.2 Actuator Models

The model introduced above uses physical torques τ_i and total thrust $\mathbf{T}^B$ as plant inputs. However, these inputs result from motor speeds which are themselves subject to additional dynamics (*motor-, battery and ESC-dynamics*). Additionally, the introductory quadrotor tutorial [42] points out and summarizes relevant *aerodynamic* effects that affect the generation of thrust $\mathbf{T}^{B,i}$ on each propeller. This section describes these phenomena. The simulation environment introduced in Section 3.6 features all these effects.

4.2.1 Aerodynamic Effects

Mahoney et al. [42] provide an overview of quadrotor estimation and control topics, including a brief introduction to the effects of blade flapping and induced drag. These effects are considered to have significant influence on the dynamics, especially since they generate forces occurring in the body-frame x-y-plane - which is under-actuated since the propeller thrust - the only *force*-control input (without these effects) - is fully aligned with the body-frame z-axis.

When the quadrotor moves in its x-y-plane, the advancing propeller blade has a higher absolute tip velocity than the retreating blade. This causes the propeller blades to bend. Due to the high rotor velocity, gyroscopic effects occur and induce a torque perpendicular to the apparent wind direction. Consequently, the thrust is not aligned with the motor axes any longer and points backwards with respect to the velocity direction of the quadrotor. Induced drag also results from different relative airspeeds of advancing vs retreating propeller blades: Since the advancing blade moves faster, it produces more lift, but also more drag than the retreating blade. The net effect is an additional, the "induced", drag. [42] also demonstrates how both effects can be modeled in a lumped model. Peter Corke's toolbox [14] models these effects. (Note that this toolbox is part of the simulator set up for this thesis (cf. Section 3.6)).

4.2.2 Motor Speed-Thrust Conversion

This section describes how [14] modeled the conversion from the four motor speeds to the plant input $\begin{bmatrix}\mathbf{T}^B & \tau_{\text{roll}} & \tau_{\text{pitch}} & \tau_{\text{yaw}}\end{bmatrix}^T$ used in the system model above.

The total thrust $\mathbf{T}^B$ results from the sum of four propeller thrusts $\mathbf{T}^{B,i}$, where the thrust vectors' directions are determined by above mentioned aerodynamic effects:

$$\mathbf{T}^B = \sum_{i=0}^{3} \mathbf{T}^{B,i}$$

where

$$\mathbf{T}^{B,i} = T^i \begin{bmatrix} -\cos(\beta_{\text{aero},i})\sin(\alpha_{\text{aero},i}) \\ \sin(\beta_{\text{aero},i}) \\ -\cos(\alpha_{\text{aero},i})\cos(\beta_{\text{aero},i}) \end{bmatrix}^B$$

with $\beta_{\text{aero},i}$ and $\alpha_{\text{aero},i}$ being detailed in Table 4.1; additional parameters can be found in the Appendix in Table A.1.

Table 4.1: Aerodynamic Effects on Thrust as in [14]

Relative air speed at propeller i: $\mathbf{v}^B_{\text{ra},i}$	$\Omega \times \mathbf{r}^B_{\text{prop},i} + \mathbf{D}^B_I(\eta)\mathbf{v}^I$
Planar components: μ_i	$\sqrt{{v^B_{\text{x,ra},i}}^2 + {v^B_{\text{y,ra},i}}^2} / \lvert\omega_i \frac{d_p}{2}\rvert$
Non-dimensionalized normal inflow: l_i	${v^B_{\text{z,ra},i}}^2 / \lvert\omega_i \frac{d_p}{2}\rvert$
Sideslip azimuth relative to x-axis: j_i	$\text{atan2}(v^B_{\text{y,ra},i}, v^B_{\text{x,ra},i}$
Sideslip rotation matrix: $\mathbf{J}_{\text{ss},i}$	$\begin{bmatrix}\cos(j_i) & -\sin(j_i) \\ \sin(j_i) & \cos(j_i)\end{bmatrix}$
Longitudinal flapping: $\beta_{\text{lf},i}$	$\mathbf{J}^T_{\text{ss},i}\begin{bmatrix}((\frac{8}{3}\theta_{\text{b0}} + 2\theta_{\text{b1}}) - 2l_i/(1/\mu_i - \mu_i/2) \\ 0\end{bmatrix}$
$\alpha_{\text{aero},i}$	$\beta_{x,\text{lf},i} - 16q/(\gamma_{\text{aero}}\lvert\omega_i\rvert)$
$\beta_{\text{aero},i}$	$\beta_{y,\text{lf},i} - 16p/(\gamma_{\text{aero}}\lvert\omega_i\rvert)$

For the body-frame torques $\tau = [\tau_\phi \ \tau_\theta \ \tau_\psi]$ it is

$$\tau = \sum_{i=0}^{3} \mathbf{r}^B_{\text{prop},i} \times \mathbf{T}^B_i + Q_i \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

with $\mathbf{r}^B_{\text{prop},i}$ being propeller i's position with respect to the quadrotor's center of mass and $Q_i = \alpha_Q \omega_i |\omega_i|$ being the propeller-drag-induced torque.

Most basic models neglect these aerodynamic effects and assume $\alpha_{\text{aero}} = \beta_{\text{aero}} = 0$, thus assuming the thrust vectors to be fully aligned with the body-frame z-axis. This results in $\mathbf{T}^B_i = [0 \ 0 \ T_i]^T$ and

$$\mathbf{T}^B = \begin{bmatrix} 0 & 0 & T \end{bmatrix}^T$$

where $T = \sum T_i$. Under these assumptions, the plant input reduces from the six dimensional $\begin{bmatrix} \mathbf{T}^B & \tau_\phi & \tau_\theta & \tau_\psi \end{bmatrix}^T$ to a four dimensional $\mathbf{u} = \begin{bmatrix} T & \tau_\phi & \tau_\theta & \tau_\psi \end{bmatrix}^T$.
With $T_i = \alpha_T \omega_i^2$ from 2.4.1, above equations can be reformulated as

$$\mathbf{u} = \begin{bmatrix} T \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \mathbf{M}^{\mathbf{u}}_{\mathbf{T}} \begin{bmatrix} \omega_0^2 \\ \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \end{bmatrix} = \alpha_T \begin{bmatrix} 1 & 1 & 1 & 1 \\ r_{\text{y},0} & r_{\text{y},1} & r_{\text{y},2} & r_{\text{y},3} \\ r_{\text{x},0} & r_{\text{x},1} & r_{\text{x},2} & r_{\text{x},3} \\ \frac{\alpha_Q}{\alpha_T} & -\frac{\alpha_Q}{\alpha_T} & \frac{\alpha_Q}{\alpha_T} & -\frac{\alpha_Q}{\alpha_T} \end{bmatrix} \begin{bmatrix} \omega_0^2 \\ \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \end{bmatrix}$$

This gives an *invertible* relation between squared motor speeds ω_i and the plant input $\mathbf{u} = [T \ \tau_\phi \ \tau_\theta \ \tau_\psi]^T$. This allows to design controllers on the basic quadrotor model with input $\mathbf{u}$ and, during flight, translate a given desired actuator action $\mathbf{u}^*(t)$ into desired motor speeds $\omega^*(t)$ to command the motors. Parameters for this equation can be found in Table 2.1 and Section 2.4.1.

4.2.3 Speed Controller-Battery Model

The hardware section 2.4.2 on battery and ESC characteristics mentioned the fact that changing terminal voltage changes the steady-state motor speeds even when PWM-signals that are being sent to the ESC remain constant. While a dropping terminal voltage as a result of a depleting battery state was not modeled, a simplified parametrized battery model was used to account for the effect of the terminal voltage dropping with increased current that is drawn from the battery. The following assumptions are made: the battery follows a Thevenin Equivalent Circuit with constant voltage source and an internal resistance; this results in the terminal voltage dropping proportionally to current drawn; the current drawn is proportional to the torque required from the motors to overcome rotational propeller drag; this torque is proportional to the thrust produced; thrust is proportional to the squared motor speeds; under ideal battery conditions, the ESC achieves a linear relationship between PWM value κ_i and the squared motor speeds; squared motor speeds therefore drop linearly with dropping terminal voltage.
A simple, approximate model derived under these assumptions is

$$\kappa_i = \frac{\alpha_{\text{ESC}}\omega_i^{*2}}{U_0 - \sum_{\text{motors }_j}(\alpha_{\text{Bat}}\omega_j^*)^2} + \kappa_0$$

A nonlinear least-squares resulted in estimated parameters:

$$\alpha_{\text{ESC}} = 0.0033 \qquad U_0 = 11.5$$
$$\alpha_{\text{Bat}} = 6.5310^{-4} \qquad \kappa_0 = 1074$$

Note that this equation can be inverted when taking into account all four ω_i^* and κ_i: The simulator used to simulate the quadrotor's dynamics takes the commanded κ_i and outputs the corresponding steady-state ω_i^*.
This model holds for the steady-state relation after transient. The next section ad-

dresses the noticeable transient in achieving this steady-state.

4.2.4 Motor Model

The hardware description in Section 2.4.3 pointed out that the closed-loop system of motor and ESC - with the PWM-value κ being the input signal and actual motor speed ω the output signal - roughly follows first-order delay dynamics when feeding a step-input.
From the STFT plot in Figure 2-8, the time constant is approximated as

$$t_m = 0.06$$

for both upwards and downwards steps, resulting in a transfer function from a *desired* motor speed ω^* (corresponding to some PWM value κ) to an *actual* motor speed ω:

$$\omega(s) = \frac{1}{t_M s + 1}\omega^*(s)$$

Summary

This chapter presented the mathematical models for the quadrotor's dynamics including actuator and battery dynamics.

The following chapter introduces the visual detection of a scene target and two estimation algorithms to generate fullstate estimates that comply with the dynamical models presented in this chapter. The estimators' performance is evaluated with experimental data.

Chapter 5

Visual-inertial State Estimation

This chapter discusses vision-based *scene target* detection and two approaches to synthesize a fullstate estimate with respect to that *scene target*. An experimental evaluation of the estimation performance is presented. Unlike *e.g.* image-based visual-servoing, the controller solution in this thesis separate the estimation and control problem. Therefore, a fullstate estimate is required. Recalling that a possible scenario for a quadrotor of this design could be to explore buildings in disaster relief scenarios, entering through openings like doors and windows becomes a crucial task. While a vision-based window detection is not the focus of this thesis, a detection is still needed to present a full proof-of-concept. For this purpose, the visual detection problem was simplified as much as possible.

5.1 Vision-based Localization

In this section, two algorithms are described: first, a simple approach to reliably find a scene target at high-rates. A marked window forms this scene target. Second, the known pnp-pose reconstruction algorithm (where *pnp* stands for *perspective-n-Point*) that is commonly used to reconstruct a camera pose from identified image features with known 3D real-world locations.

5.1.1 Scene Target Detection

Since the detection problem is not focus of this thesis, the experimental setup was designed such that the real-world markers (oftentimes referred to as landmarks) and their corresponding image features are most easily detectable and unambiguously identifiable at high rates.

In the chosen experimental setup, a window-like structure was imitated by a dark rectangle and triangle on a white wall (Figure 5-1). It is positioned at the back end of the flying area, around $x^I = 3.0m$. This structure enabled a reliably unambiguous identification of all corners on an image level - independent from any other estimates - and therefore proved to be a convincing choice as image features. The global 3D-positions of these corners have been measured using the motion capture system. AprilTags or colored dots were considered as an alternative. However, the first approach showed robust results at rates above 50Hz.

The algorithm to find and identify the seven corners of the marker setup is described in the following paragraphs. First, the gray-scale image is thresholded, resulting in an image shown in Figure 5-1. This thresholded image is then fed into openCV's

Figure 5-1: Thresholded Image of View on Marker Setup

findContours-function. A *Ramer-Douglas-Peucker*-algorithm (RDP) [29] is applied to fit polygons to the contours, effectively reducing the number of points of each contour. The approximation tolerance is chosen such that the marker triangle and rectangle are robustly approximated as three and four-point polygons for a wide range of camera poses. The resulting image with fitted polygons is shown in Figure 5-2. The set of all three- and four-point polygons is then searched for a pair with

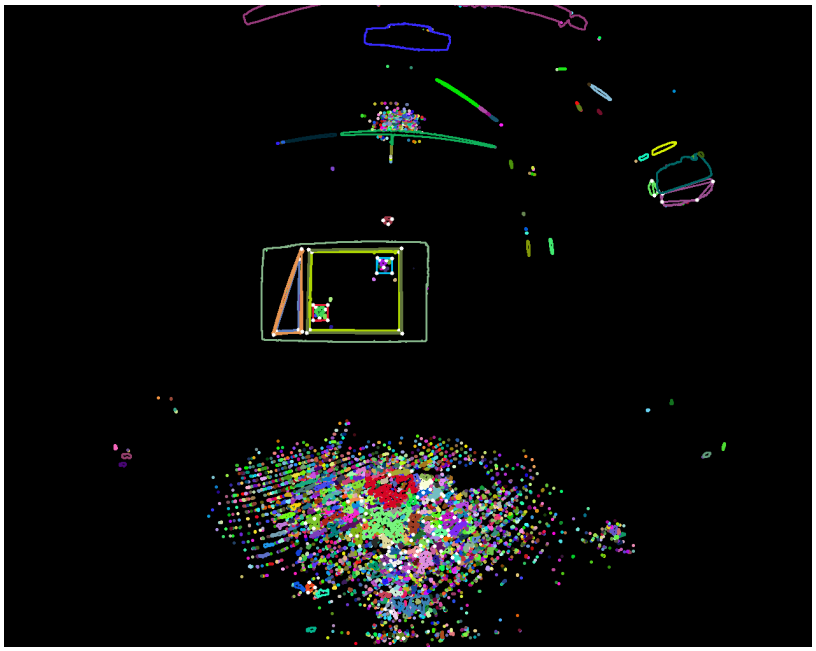


Figure 5-2: Polygons found by RDP

a ratio $\zeta = \frac{|\mathbf{p}_i^{\text{fish,T}} - \mathbf{p}_j^{\text{fish,R}}|_{\min}}{|\mathbf{p}_i^{\text{fish,T}} - \mathbf{p}_j^{\text{fish,R}}|_{\max}}$ with $\underline{\zeta} < \zeta < \bar{\zeta}$ (where $\mathbf{p}_i^{\text{fish,T}}$,$\mathbf{p}_j^{\text{fish,R}}$ denote the pixel locations in the original image of the triangle and rectangle corners, respectively), and $\underline{A} < \frac{A_{\text{rectangle}}}{A_{\text{triangle}}} < \bar{A}$, with A denoting the area of the polygons. Parameters can be found in Table A.2.

This approach proved to be robust enough to uniquely and unambiguously identify the 7 corners of the triangle and rectangle. Figure 5-3 shows an overlay of original image and identified markers.

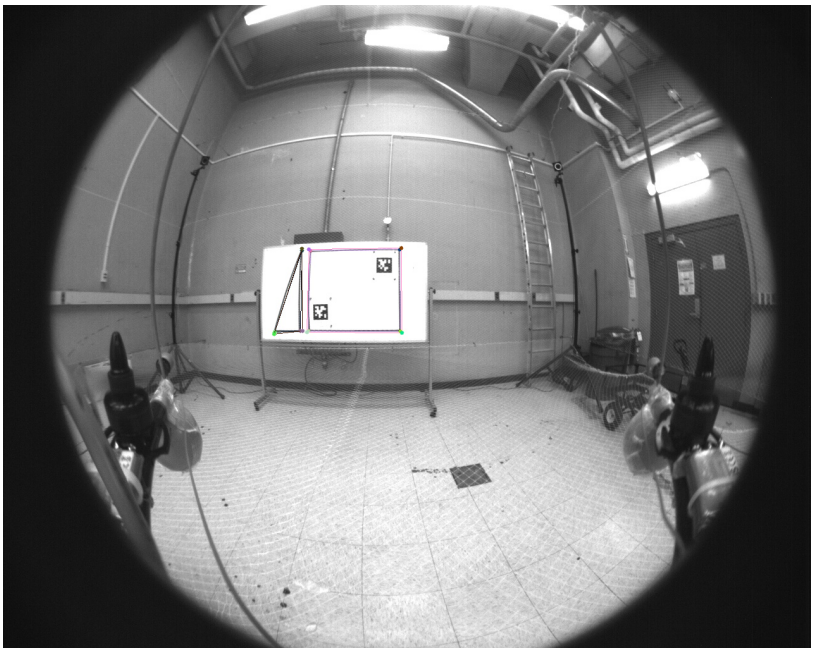


Figure 5-3: Identified markers: Triangle (black), rectangle (purple).

5.1.2 Pose Reconstruction

Given that the features (the detected corners) come with known global 3D coordinates $\mathbf{m}^I$, it is possible to fully reconstruct the camera's 3D-pose. This problem is called *Perspective-n-Point*. With 7 3D-points available to be used for reconstruction, this becomes an optimization problem that aims to find the optimal camera pose (consisting of a translation $\mathbf{r}^{I,\text{cam}}$ and an orientation η^{cam}) that minimizes the reprojection error

$$e = \sum_{i=1}^{7} \left\| s\mathbf{p}^{\text{pin},i} - \mathbf{K}[\mathbf{D}_W^{\text{cam}}(\eta^{\text{cam}})|\mathbf{r}^{I,\text{cam}}]\mathbf{m}^{I,i} \right\|$$

where $\mathbf{p}_i^{\text{pin}}$ is the undistorted pinhole-image pixel location of feature i, s a scale factor, $\mathbf{K}$ the matrix of generic intrinsic camera parameters, $\mathbf{m}^{I,i}$ the 3D-world coordinates of feature i expressed in the inertial frame and where $\mathbf{K}[\mathbf{D}_W^{\text{cam}}(\eta^{\text{cam}})|\mathbf{r}^{I,\text{cam}}]\mathbf{m}^{I,i}$ corresponds to the standard projection model for pinhole camera models that projects a 3D-point onto the image plane.

openCV provides an implementation for this setup. The optimizer used in the implementation is a standard Levenberg-Marquardt algorithm [60].
The camera used in this thesis was equipped with a fish-eye lens. Therefore, pinhole camera models do not hold. As described in 2.3.2, a camera model for fish-eye lenses allows to reconstruct a 3D-unit ray $\tilde{\mathbf{m}}^{B,i}$(emanating from the camera) for a marker with detected image pixel location $\tilde{\mathbf{p}}^{\text{fish},i}$. This 3D-unit ray is then projected onto a generic image plane using a generic pinhole camera model to compute the undistorted, pinhole-camera pixel location $\tilde{\mathbf{p}}^{\text{pin},i}$.
This approach provides a full reconstruction of the quadrotor's pose which is used for one of the suggested sensor fusion models that is described in the following sections.

5.2 Sensor Fusion

In this thesis' setup, the purpose of sensor fusion is to estimate the quadrotor's full state $\mathbf{x}$, consisting of the position $\mathbf{r}^I$, the velocities $\mathbf{v}^I$, the orientation η and the angular rates $\mathbf{\Omega}$. The sensor setup provides measurements of the body rates $\tilde{\mathbf{\Omega}}^{B,\text{gyro}}$ through the gyroscope, *specific* acceleration $\tilde{\mathbf{a}}^{B,\text{acc}}$ through the accelerometer (body-frame acceleration overlaid by gravity) and unit rays $\tilde{\mathbf{m}}^{B,i}$ emanating from the camera to real-world markers with known 3D-coordinates. The following sections introduce two approaches to generate fullstate estimates. As Shen et al. note, unobservable modes can occur during flight maneuvers with zero linear acceleration when utilizing simple filter approaches [52]. However, the main application case of this thesis are agile maneuvers approaching target regions, so non-zero acceleration can be expected most of the time. Additionally, since the flight maneuvers are short (<3sec) highly accurate drift-free estimation is not necessary.

5.2.1 Hybrid Filter

The hybrid filter approach described in this section combines a complimentary filter (CF) for estimating orientation and a standard Kalman filter (KF) for estimating the translational components of the full state. It requires the pnp-pose-reconstruction

algorithm to run simultaneously with the estimator since this hybrid estimator takes a fully reconstructed 3D-pose as measurement update.

Inputs into the complimentary filter are measurements from the IMU's gyroscope $\tilde{\mathbf{\Omega}}^{B,\text{gyro}}$ and accelerometer $\tilde{\mathbf{a}}^{B,\text{acc}}$, and a yaw-estimate ψ^{pnp} reconstructed purely from vision via the pnp-pose-reconstruction algorithm described above. The Kalman-filter uses accelerometer data $\tilde{\mathbf{a}}^{B,\text{acc}}$ as input to the state propagation model and the reconstructed 3D-pose $\hat{\mathbf{r}}^{I,\text{pnp}}$ as measurement update. The orientation estimate from the complimentary filter is used to transform the accelerometer data into global acceleration.

While this approach is not optimal since it does not exploit the full coupling between orientation, position and observed visual features, it offers the advantage that the pitch- and roll- estimates are separate from visual inputs. In case of a (possibly unnoticed) failure of the vision pipeline, a reliable orientation estimate is still available, so the quadrotor can be stabilized (without preventing drift in position though). Figure 5-4 shows the signal flow.

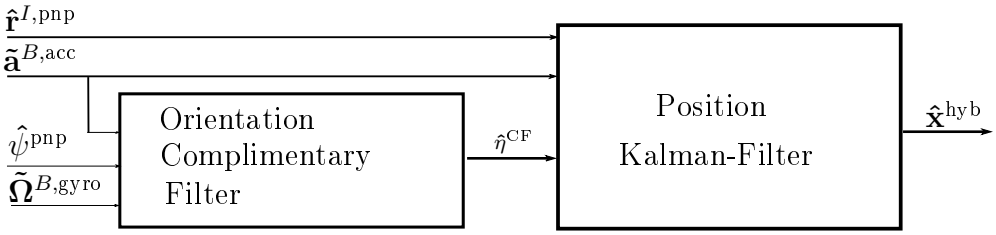


Figure 5-4: Hybrid Fullstate Filter with CF and KF: Signal Flow

The complimentary filter is a simple, but widely used filter to estimate the quadrotor's orientation. In essence, its simplest, linear version simply integrates the angular rates measured by the gyroscope to obtain an orientation estimate [9]. In steady-state flight, an estimate for pitch θ and roll ϕ can be inferred from the direction of the acceleration vector measured by the accelerometer. This can be used to cope with drift in the orientation estimate resulting from the simple integration of angular rates.

The estimate $\hat{\eta}^{\int}$ from the integration approach results from

$$\hat{\eta}^{\int} = \int \mathbf{W}^{-1}\tilde{\mathbf{\Omega}}_{\text{gyro}}$$

The sensor model for the accelerometer measurement $\tilde{\mathbf{a}}^{\mathrm{acc}}$ is assumed as

$$\tilde{\mathbf{a}}^{B,\mathrm{acc}}(t) = \mathbf{D}_I^B(t)(\mathbf{a}^I(t) - \mathbf{G}^I) + \mathbf{b}^{B,\mathrm{acc}} + \delta\mathbf{a}^{B,\mathrm{acc}}(t)$$

with $\mathbf{a}^I$ being the acceleration of the quadrotor's center of mass in the inertial frame, $\mathbf{G}^I$ the gravity vector, a bias $\mathbf{b}^{B,\mathrm{acc}}$ which is assumed to be constant during the short experiments and zero after calibrating the sensors, and finally, zero-mean Gaussian noise $\delta\mathbf{a}^{B,\mathrm{acc}}(t)$.
In steady-state flight $\mathbf{a}^I \approx 0$ and, with the chosen representation of the rotation matrix $\mathbf{D}_I^B$, it follows

$$E[-\frac{\tilde{\mathbf{a}}^{B,\mathrm{acc}}}{g}] = E(\mathbf{D}_I^B(t)(:,3)) = \begin{bmatrix} -\sin(\theta) \\ \cos(\theta)\sin(\phi) \\ \cos(\theta)\cos(\phi) \end{bmatrix}$$

with $E[\cdot]$ being the expectation operator.
This results in estimates $\hat{\phi}^{\mathrm{acc}}$ and $\hat{\theta}^{\mathrm{acc}}$.

$$\hat{\theta}^{\mathrm{acc}} = \sin^{-1}(\tilde{a}^{B,x,\mathrm{acc}}/g)$$
$$\hat{\phi}^{\mathrm{acc}} = \tan^{-1}(\tilde{a}^{B,y,\mathrm{acc}}/\tilde{a}^{B,z,\mathrm{acc}})$$

In practice, these accelerometer measurements are very noisy and thus have to be low-pass-filtered. This does not introduce rate-issues, since the high-bandwidth components of the angle estimates come from angular rates. Additionally, as pointed out, these estimates only hold for $\mathbf{a}^I \approx 0$. In usual flight scenarios, the low-pass filter takes care of this restriction, too, since, most of the time, quadrotors are flown in steady-state. However, in more agile flight scenarios, this does not hold. Restricting the use of $\hat{\phi}^{\mathrm{acc}}$ and $\hat{\theta}^{\mathrm{acc}}$ to time intervals where $|\tilde{\mathbf{a}}^{\mathrm{acc}}| \approx g$ (which holds in steady-state, as well as some agile maneuvers where the quadrotor loses altitude and accelerates in X-Y-plane) further excludes measurements acquired during dynamic flight states that do not meet $\mathbf{a}^I \approx 0$. [18] presents an augmented complementary filter that in-

cludes dynamic models for centripetal forces and air speed measurements to better cope with drift and bias.

Note that an estimate for yaw (ψ) cannot be inferred from the accelerometer. The estimate $\hat{\psi}^{\text{pnp}}$ from the visual reconstruction is fed into the complimentary filter instead. This approach does not strongly oppose the idea of designing the estimation architecture in such a way that a stable flight is still possible (with drift) even with a broken vision-pipeline since a drift in yaw does not heavily affect position dynamics. The complete, discretized filter is then

$$\hat{\eta}_{k+1}^{\text{CF}} = \left\{ \begin{array}{cc} \hat{\eta}_k^{\text{CF}} + (\mathbf{I}_{3x3} - \mathbf{w}^{\text{CF}})\mathbf{W}^{-1}\tilde{\mathbf{\Omega}}^{B,\text{gyro}}\triangle t + \mathbf{w}^{\text{CF}}[\hat{\psi}^{\text{pnp}}\ \hat{\theta}^{\text{acc}}\ \hat{\phi}^{\text{acc}}]^T; & \text{if } |\tilde{\mathbf{a}}^{B,\text{acc}}| \approx g \\ \hat{\eta}_k^{\text{CF}} + \mathbf{W}^{-1}\tilde{\mathbf{\Omega}}^{B,\text{gyro}}\triangle t; & \text{else} \end{array} \right\}$$

Note that this setup assumes $\mathbf{W}^{-1}$ to be constant during $\triangle t$ although $\mathbf{W}^{-1} = f(\eta)$. Also note that it acts as the mentioned low-pass filter on $\hat{\phi}^{\text{acc}}$ and $\hat{\theta}^{\text{acc}}$ through the weighting.

The complimentary filter provides its output, the orientation estimate $\hat{\eta}^{\text{CF}}$, to the linear discrete Kalman-filter so it can derive the estimated, inertial acceleration $\hat{\mathbf{a}}^I$ from the measured acceleration $\tilde{\mathbf{a}}^{\text{acc}}$ and, with the help of the pnp-reconstructed position $\hat{\mathbf{r}}^{I,\text{pnp}}$ as measurement update, output an estimate $\hat{\mathbf{x}}^{\text{KF}}$ containing estimates for the global inertial position $\mathbf{r}^I$ and velocity $\mathbf{v}^I$.

Following the standard Kalman-filter model from literature, *e.g.* [12], the filter's

Table 5.1: Hybrid Filter: Noise covariances

$\mathbf{Q}^{\mathrm{KF}}$	$= E[(\mathbf{V}^{\mathrm{KF}})^T\mathbf{V}^{\mathrm{KF}}]$	$= \mathrm{diag}([1\,1\,1\,50\,4\,4])$
$\mathbf{R}^{\mathrm{KF}}$	$= E[(\mathbf{W}^{\mathrm{KF}})^T\mathbf{W}^{\mathrm{KF}}]$	$= 100\cdot\mathbf{I}_3$

discrete state propagation and measurements equation are given by

$$\mathbf{x}_{k+1}^{\mathrm{KF}} = \begin{bmatrix} {\mathbf{r}^{\mathbf{I}}}_{k+1}^{\mathrm{KF}} \\ {\mathbf{v}^{\mathbf{I}}}_{k+1}^{\mathrm{KF}} \end{bmatrix} = \mathbf{F}_k^{\mathrm{KF}}\mathbf{x}_k^{\mathrm{KF}} + \mathbf{G}_k^{\mathrm{KF}}\mathbf{u}_k^{\mathrm{KF}} + \mathbf{V}_k^{\mathrm{KF}}$$

$$= \begin{bmatrix} 1 & 0 & 0 & \triangle t & 0 & 0 \\ 0 & 1 & 0 & 0 & \triangle t & 0 \\ 0 & 0 & 1 & 0 & 0 & \triangle t \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{x}_k^{\mathrm{KF}} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ \triangle t & 0 & 0 \\ 0 & \triangle t & 0 \\ 0 & 0 & \triangle t \end{bmatrix} (\mathbf{D}_B^I(\eta_k)\,\tilde{\mathbf{a}}_k^{B,\mathrm{acc}} + \mathbf{G}^I) + \mathbf{V}_k^{\mathrm{KF}}$$

$$\tilde{\mathbf{z}}_{k+1}^{\mathrm{KF}} = {\mathbf{r}^{\mathbf{I}}}_{k+1}^{\mathrm{KF}} = \mathbf{H}^{\mathrm{KF}}\mathbf{x}_{k+1}^{\mathrm{KF}} + \mathbf{W}_{k+1}^{\mathrm{KF}} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{x}_{k+1}^{\mathrm{KF}} + \mathbf{W}_{k+1}^{\mathrm{KF}}$$

with $\mathbf{V}^{\mathrm{KF}}$ and $\mathbf{W}^{\mathrm{KF}}$ being zero-mean Gaussian white-noise with covariance matrices detailed in Table 5.1:

The full set of Kalman equations is given by

$$\begin{aligned} \hat{\mathbf{x}}_{k+1|k}^{\mathrm{KF}} &= \mathbf{F}_k^{\mathrm{KF}}\hat{\mathbf{x}}_{k|k}^{\mathrm{KF}} + \mathbf{G}_k^{\mathrm{KF}}\mathbf{u}_k \\ \mathbf{P}_{k+1|k}^{\mathrm{KF}} &= \mathbf{F}_k^{\mathrm{KF}}\mathbf{P}_{k|k}(\mathbf{F}_k^{\mathrm{KF}})^T + \mathbf{Q}^{\mathrm{KF}} \\ \hat{\mathbf{x}}_{k+1|k+1}^{\mathrm{KF}} &= \hat{\mathbf{x}}_{k+1|k}^{\mathrm{KF}} + \mathbf{K}_{k+1}^{\mathrm{KF}}[\tilde{\mathbf{z}}_{k+1}^{\mathrm{KF}} - \mathbf{H}^{\mathrm{KF}}\hat{\mathbf{x}}_{k+1|k}^{\mathrm{KF}}] \\ \mathbf{P}_{k+1|k+1}^{\mathrm{KF}} &= (\mathbf{I} - \mathbf{K}_{k+1}^{\mathrm{KF}}\mathbf{H}^{\mathrm{KF}})\mathbf{P}_{k+1|k}^{\mathrm{KF}}(\mathbf{I} - \mathbf{K}_{k+1}^{\mathrm{KF}}\mathbf{H}^{\mathrm{KF}})^T + \mathbf{K}_{k+1}^{\mathrm{KF}}\mathbf{R}^{\mathrm{KF}}(\mathbf{K}_{k+1}^{\mathrm{KF}})^T \end{aligned}$$

with

$$\mathbf{K}_{k+1}^{\mathrm{KF}} = \mathbf{P}_{k+1|k}^{\mathrm{KF}}(\mathbf{H}^{\mathrm{KF}})^T[\mathbf{H}^{\mathrm{KF}}\mathbf{P}_{k+1|k}^{\mathrm{KF}}(\mathbf{H}^{\mathrm{KF}})^T + \mathbf{R}^{\mathrm{KF}}]^{-1}$$

The hybrid filter's estimate $\hat{\mathbf{x}}^{\mathrm{hyb}}$ is then composed of

$$\hat{\mathbf{x}}^{\mathrm{hyb}} = \begin{bmatrix} \mathbf{x}^{\mathrm{KF}} & \hat{\eta}^{\mathrm{CF}} \end{bmatrix}^T = \begin{bmatrix} \mathbf{r}^{I,\mathrm{KF}} & \mathbf{v}^{I,\mathrm{KF}} & \hat{\eta}^{\mathrm{CF}} \end{bmatrix}^T$$

Note that the angular rates $\tilde{\mathbf{\Omega}}^{B,\mathrm{gyro}}$ can be appended to $\hat{\mathbf{x}}^{\mathrm{hyb}}$ in order to obtain a full state estimate $\hat{\mathbf{x}}$ corresponding to the modeling in 4.1.2. Onboard-filters on the IMU-chip take care of filtering the gyroscope's sensor noise.

5.2.2 Extended Kalman Filter

This section describes a second approach for fullstate estimation. In contrast to the hybrid-approach, it does not require a fully reconstructed 3D-pose from vision since the state propagation model models the dynamics between IMU measurements as filter input and unit rays (to the visual markers) as measurement update. This approach lowers delays induced by the pose-reconstruction and increases accuracy since it exploits the coupling between orientation, position and observed visual features. Bad, unnoticed vision measurements can, however, destabilize the quadrotor significantly since vision measurements are used to update pitch- and roll-estimates. Note that the estimation is run at 100Hz and features as measurement updates arrive at 50Hz.

Due to the underlying nonlinear dynamics, a standard extended Kalman filter [12] is used. The resulting signal flow is illustrated in Figure 5-5. The state $\mathbf{x}^{\mathrm{EKF}}$ uses a quaternion representation $\mathbf{q}$ of the quadrotor's orientation to prevent the estimation from running into singularities as it can happen with euler-angles. Therefore

$$\mathbf{x}^{\mathrm{EKF}} = \begin{bmatrix} \mathbf{q} & \mathbf{r}^I & \mathbf{v}^I \end{bmatrix}^T$$

Figure 5-5: Fullstate EKF: Signal Flow

Note that the angular rates $\tilde{\boldsymbol{\Omega}}^{B,\text{gyro}}$ can be appended to $\hat{\mathbf{x}}^{\text{EKF}}$ and the quaternion representation $\hat{\mathbf{q}}^{\text{EKF}}$ can be transformed to euler-representation $\hat{\eta}^{\text{EKF}}$ in order to obtain a full state estimate $\hat{\mathbf{x}}$ corresponding to the modeling in 4.1.2. Onboard-filters on the IMU-chip take care of filtering the gyroscope's sensor noise.

General EKF equations

This section briefly introduces the general setup of an EKF, as found in e.g. [12]. Assume a discrete-time, nonlinear state-space model for state propagation and measurement dynamics:

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) + \delta\mathbf{f}_k$$
$$\mathbf{z}_{k+1} = \mathbf{h}(\mathbf{x}_{k+1}) + \delta\mathbf{h}_{k+1}$$

with $\mathbf{x_k}$ being the state at time t_k, $\mathbf{u}_k$ the input, $\mathbf{z}_k$ the measurements and $\delta\mathbf{f}_k$ and $\delta\mathbf{h}_k$ Gaussian noise sources with covariances $\mathbf{Q}_k$ and $\mathbf{R}_k$ for state propagation and measurement noise, respectively.
The EKF utilizes linearized versions of the state and measurement equations to update the estimation covariances. The functions $\mathbf{f}$ and $\mathbf{h}$ are approximated as

$$\mathbf{f}(\mathbf{x}_k, \mathbf{u}_k) \approx \mathbf{f}(\hat{\mathbf{x}}_{k|k}, \mathbf{u}_k) + \mathbf{F}_k \cdot (\mathbf{x}_k - \hat{\mathbf{x}}_{k|k})$$
$$\mathbf{h}(\mathbf{x}_{k+1}) \approx \mathbf{h}(\hat{\mathbf{x}}_{k+1}) + \mathbf{H}_{k+1} \cdot (\mathbf{x}_{k+1} - \hat{\mathbf{x}}_{k+1|k})$$

where $\mathbf{x}_{k+1|k}$ denotes the state estimate at t_{k+1} from k measurements. The matrices $\mathbf{F}$ and $\mathbf{H}$ are derived as

$$\mathbf{F}_k = \frac{\partial \mathbf{f}}{\partial \mathbf{x}}|_{\mathbf{x}=\hat{\mathbf{x}}_k, \mathbf{u}=\mathbf{u}_k}$$
$$\mathbf{H}_{k+1} = \frac{\partial \mathbf{h}}{\partial \mathbf{x}}|_{\mathbf{x}=\hat{\mathbf{x}}_{k+1|k}}$$

The EKF algorithm then follows as:

1. State propagation

$$\hat{\mathbf{x}}_{k+1} = \mathbf{f}(\hat{\mathbf{x}}_k, \mathbf{u}_k)$$

2. Covariance propagation

$$\mathbf{P}_{k+1|k} = \mathbf{F}_k \mathbf{P}_{k|k} \mathbf{F}_k^T + \mathbf{Q}_k$$

3. Expected measurement computation

$$\hat{\mathbf{z}}_{k+1} = \mathbf{h}(\hat{\mathbf{x}}_{k+1|k})$$

4. Residual covariance update

$$\mathbf{S}_{k+1|k} = \mathbf{H}_{k+1} \mathbf{P}_{k+1|k} \mathbf{H}_{k+1}^T + \mathbf{R}_{k+1}$$

5. Kalman gain update

$$\mathbf{K}_{k+1} = \mathbf{P}_{k+1|k} \mathbf{H}_{k+1}^T \mathbf{S}_{k+1|k}^{-1}$$

6. State estimate update

$$\hat{\mathbf{x}}_{k+1|k+1} = \hat{\mathbf{x}}_{k+1|k} + \mathbf{K}_{k+1}(\mathbf{z_{k+1}} - \hat{\mathbf{z}}_{k+1|k})$$

7. Covariance estimate update

$$\mathbf{P}_{k+1|k+1} = \mathbf{P}_{k+1|k} - \mathbf{P}_{k+1|k}\mathbf{H}_{k+1}^T\mathbf{K}_{k+1}^T$$

Problem-specific EKF setup

With the specific state $\mathbf{x} = \mathbf{x}^{\text{EKF}} = \begin{bmatrix}\mathbf{q} & \mathbf{r}^I & \mathbf{v}^I\end{bmatrix}^T$, the state propagation function $\mathbf{f}(\mathbf{x}, \mathbf{u})$ is composed of $\mathbf{f}(\mathbf{x}, \mathbf{u}) = \begin{bmatrix}\mathbf{f}^{\mathbf{q}}(\mathbf{x}, \mathbf{u}) & \mathbf{f}^{\mathbf{r}}(\mathbf{x}, \mathbf{u}) & \mathbf{f}^{\mathbf{v}}(\mathbf{x}, \mathbf{u})\end{bmatrix}^T$, so also

$$\mathbf{F}(\mathbf{x}, \mathbf{u}) = \begin{bmatrix}\mathbf{F}^{\mathbf{q}}(\mathbf{x}, \mathbf{u}) \\ \mathbf{F}^{\mathbf{r}}(\mathbf{x}, \mathbf{u}) \\ \mathbf{F}^{\mathbf{v}}(\mathbf{x}, \mathbf{u})\end{bmatrix}$$

and state propagation noise $\delta\mathbf{f}$ is composed of $\delta\mathbf{f} = \begin{bmatrix}\delta\mathbf{f}^{\mathbf{q}} & \delta\mathbf{f}^{\mathbf{r}} & \delta\mathbf{f}^{\mathbf{v}}\end{bmatrix}^T$.
The state propagation noise covariance matrix follows with

$$\mathbf{Q} = \text{diag}(\begin{bmatrix}\mathbf{Q}^{\mathbf{q}} & \mathbf{Q}^{\mathbf{r}} & \mathbf{Q}^{\mathbf{v}}\end{bmatrix})$$

with 'diag' creating a 10x10 matrix with the three covariance matrices on its diagonal. The input vector $\mathbf{u}$ comprises measured angular rates $\tilde{\mathbf{\Omega}}^{B,\text{gyro}} = \tilde{\mathbf{\Omega}}$ and accelerometer data $\tilde{\mathbf{a}}^{B,\text{acc}} = \tilde{\mathbf{a}}$, such that $\mathbf{u} = \begin{bmatrix}\tilde{\mathbf{\Omega}} & \tilde{\mathbf{a}}\end{bmatrix}^T$.

The following sections derive the functions and matrices necessary to run the EKF algorithm. The first section focuses on the orientation dynamics, the next on the translational (*i.e.*, position and velocity) dynamics.

Orientation dynamics

The orientation-part of the presented full-state EKF closely follows [6] and [59].
[59] presents the discrete state propagation dynamics:

$$\mathbf{q}_{k+1} = \mathbf{\Phi}(\underline{\underline{\mathbf{\Omega}}}_k, \triangle t)\mathbf{q}_k$$

with $\triangle t$ denoting the sample time and $\mathbf{\Phi} \in \Re^{4x4}$ denoting the state transition matrix and

$$\underline{\underline{\mathbf{\Omega}}}_k = \frac{1}{2}\begin{bmatrix} 0 & -\mathbf{\Omega}_k^T \\ \mathbf{\Omega}_k & -[\mathbf{\Omega}_k]_\times \end{bmatrix} \qquad \text{where} \quad [\mathbf{\Omega}_k]_\times = \begin{bmatrix} 0 & -r & q \\ r & 0 & -p \\ -q & p & 0 \end{bmatrix}$$

$\mathbf{\Phi}$ can be written as

$$\mathbf{\Phi}_k = e^{\underline{\underline{\mathbf{\Omega}}}_k \triangle t} = \cos(\|\mathbf{\Omega}_k\| \frac{\triangle t}{2})\mathbf{I}_4 + \frac{1}{\|\mathbf{\Omega}_k\|}\sin(\|\mathbf{\Omega}_k\| \frac{\triangle t}{2})\underline{\underline{\mathbf{\Omega}}}_k$$

Linearized state propagation dynamics Let the measurement $\tilde{\mathbf{\Omega}}$ of the angular rates (note that this is used as input into the state propagation dynamics not as measurement updates) be subject to zero-mean Gaussian white noise $\delta\mathbf{\Omega}$ with covariance $\mathbf{Q}^{\text{gyro}}$

$$\hat{\mathbf{\Omega}}_k = \mathbf{\Omega}_k + \delta\mathbf{\Omega}$$

After plugging in this noise model into the above presented state propagation dynamics, a first order Taylor-approximation yields

$$\mathbf{q}_{k+1} \approx e^{\tilde{\underline{\underline{\mathbf{\Omega}}}}_k \triangle t}\mathbf{q}_k + e^{\tilde{\underline{\underline{\mathbf{\Omega}}}}_k \triangle t}\delta\underline{\underline{\mathbf{\Omega}}}_k \triangle t\mathbf{q}_k$$

which can be rewritten as

$$\mathbf{q}_{k+1} \approx e^{\tilde{\underline{\underline{\mathbf{\Omega}}}}_k \triangle t}\mathbf{q}_k + \mathbf{\Gamma}_k\delta\mathbf{\Omega}_k$$

with

$$\mathbf{\Gamma}_k = \frac{\triangle t}{2}e^{\tilde{\underline{\underline{\mathbf{\Omega}}}}_k \triangle t}\mathbf{\Xi}$$

where

$$\mathbf{\Xi} = \begin{bmatrix} -\mathbf{e}_k^T \\ [\mathbf{e}_k x] + q_k \mathbf{I}_3 \end{bmatrix}$$

The state propagation noise covariance can therefore be approximated as

$$\mathbf{Q}_k^q = \mathbf{\Gamma}_k \mathbf{Q}^{\text{gyro}} \mathbf{\Gamma}_k^T$$

Note that the state propagation is already linear in state $\mathbf{q}$ with $e^{\underline{\underline{\tilde{\mathbf{\Omega}}}}_k \triangle t}\mathbf{q}_k$. It follows

$$\mathbf{F^q}_k = \text{diag}(\begin{bmatrix} e^{\underline{\underline{\tilde{\mathbf{\Omega}}}}_k \triangle t} & \mathbf{0}_{4x3} & \mathbf{0}_{4x3} \end{bmatrix})$$

Translational dynamics

Correcting measured, *specific* acceleration to global acceleration in the inertial frame and integrating twice over a time step $\triangle t$ yields

$$\mathbf{r}_{k+1}^I = \mathbf{r}_k^I + \mathbf{v}_k^I \triangle t + \frac{\triangle t^2}{2}(\mathbf{D}_B^I(\mathbf{q}_k)\mathbf{a}_k^B + \mathbf{G})$$

Linearized position propagation dynamics Assuming a noise model of

$$\tilde{\mathbf{a}}_k^B = \mathbf{a}_k^B + \delta \mathbf{a}_k^B$$

with noise covariance $\mathbf{Q^a}$ we get

$$\mathbf{r}_{k+1}^I = \mathbf{r}_k^I + \mathbf{v}_k^I \triangle t + \frac{\triangle t^2}{2}(\mathbf{D}_B^I(\mathbf{q}_k)(\tilde{\mathbf{a}}_k^B + \delta \mathbf{a}_k^B) + \mathbf{G})$$

It follows

$$\mathbf{F^r}_k = \frac{\partial \mathbf{f^r}}{\partial \mathbf{x}}|_{\mathbf{x}=\hat{\mathbf{x}}_\mathbf{k}, \mathbf{u}=\mathbf{u}_k} = \begin{bmatrix} \frac{\triangle t^2}{2} \frac{d\mathbf{D}_B^I(\hat{\mathbf{q}}_k)}{d\mathbf{q}} \tilde{\mathbf{a}}_k & \mathbf{I}_3 & \mathbf{I}_3 \triangle t \end{bmatrix}$$

with

$$\frac{d\mathbf{D}_B^I(\hat{\mathbf{q}}_k)}{d\mathbf{q}}\tilde{\mathbf{a}}_k = \begin{bmatrix} \frac{d\mathbf{D}_B^I(\hat{\mathbf{q}}_k)}{dq_0}\tilde{\mathbf{a}}_k & \frac{d\mathbf{D}_B^I(\hat{\mathbf{q}}_k)}{dq_1}\tilde{\mathbf{a}}_k & \frac{d\mathbf{D}_B^I(\hat{\mathbf{q}}_k)}{dq_2}\tilde{\mathbf{a}}_k & \frac{d\mathbf{D}_B^I(\hat{\mathbf{q}}_k)}{dq_3}\tilde{\mathbf{a}}_k \end{bmatrix}$$

where

$$\frac{\partial \mathbf{D}_B^I(\mathbf{q})}{\partial q_0}|_{\mathbf{q}=\hat{\mathbf{q}}_{\mathbf{k+1|k}}} = 2\begin{bmatrix} \hat{q}_0 & -\hat{q}_3 & \hat{q}_2 \\ \hat{q}_3 & \hat{q}_0 & -\hat{q}_1 \\ -\hat{q}_2 & \hat{q}_1 & \hat{q}_0 \end{bmatrix}_{k+1|k} \qquad \frac{\partial \mathbf{D}_B^I(\mathbf{q})}{\partial q_1}|_{\mathbf{q}=\hat{\mathbf{q}}_{\mathbf{k+1|k}}} = 2\begin{bmatrix} \hat{q}_1 & \hat{q}_2 & \hat{q}_3 \\ \hat{q}_2 & -\hat{q}_1 & -\hat{q}_0 \\ \hat{q}_3 & \hat{q}_0 & -\hat{q}_1 \end{bmatrix}_{k+1|k}$$

$$\frac{\partial \mathbf{D}_B^I(\mathbf{q})}{\partial q_2}|_{\mathbf{q}=\hat{\mathbf{q}}_{\mathbf{k+1|k}}} = 2\begin{bmatrix} -\hat{q}_2 & \hat{q}_1 & \hat{q}_0 \\ \hat{q}_1 & \hat{q}_2 & \hat{q}_3 \\ -\hat{q}_0 & \hat{q}_3 & -\hat{q}_2 \end{bmatrix}_{k+1|k} \qquad \frac{\partial \mathbf{D}_B^I(\mathbf{q})}{\partial q_3}|_{\mathbf{q}=\hat{\mathbf{q}}_{\mathbf{k+1|k}}} = 2\begin{bmatrix} -\hat{q}_3 & -\hat{q}_0 & \hat{q}_1 \\ \hat{q}_0 & -\hat{q}_3 & \hat{q}_2 \\ \hat{q}_1 & \hat{q}_2 & \hat{q}_3 \end{bmatrix}_{k+1|k}$$

Regarding noise:

$$\delta\mathbf{f^r}_k = \frac{\triangle t^2}{2}\mathbf{D}_B^I(\mathbf{q}_k)\delta\mathbf{a}_k^B$$

results in

$$\mathbf{Q^r}_k = \frac{\triangle t^4}{4}\mathbf{D}_B^I(\hat{\mathbf{q}}_k)\mathbf{Q^a}\mathbf{D}_B^{I\,T}(\hat{\mathbf{q}}_k)$$

Linearized velocity propagation dynamics

Similar steps as in the previous section lead to

$$\mathbf{v}_{k+1}^I = \mathbf{v}_k^I + \triangle t(\mathbf{D}_B^I(\mathbf{q}_k)(\tilde{\mathbf{a}}_k^B + \delta\mathbf{a}_k^B) + \mathbf{G})$$

It follows

$$\mathbf{F^v}_k = \frac{\partial \mathbf{f^v}}{\partial \mathbf{x}}|_{\mathbf{x}=\hat{\mathbf{x}}_\mathbf{k},\mathbf{u}=\mathbf{u}_k} = \begin{bmatrix} \triangle t\frac{d\mathbf{D}_B^I(\hat{\mathbf{q}}_k)}{d\mathbf{q}}\tilde{\mathbf{a}}_k & \mathbf{0}_3 & \mathbf{I}_3 \end{bmatrix}$$

and

$$\delta \mathbf{f^v}_k = \triangle t \mathbf{D}_B^I(\mathbf{q}_k)\delta \mathbf{a}_k^B$$

State propagation noise covariance then follows as

$$\mathbf{Q^v}_k = \triangle t^2 \mathbf{D}_B^I(\hat{\mathbf{q}}_k)\mathbf{Q^a}{\mathbf{D}_B^I}^{-T}(\hat{\mathbf{q}}_k)$$

Measurement Model

The discrete-time measurement model relates the observed/measured unit rays $\tilde{\mathbf{m}}^{B,i}$ (emanating from the camera origin towards the world markers) to the current position and orientation of the quadrotor. It is assumed that the observed unit rays $\tilde{\mathbf{m}}^{\mathrm{cam},i}$ expressed in the camera frame have been transformed into the body-frame and normalized to $\tilde{\mathbf{m}}_x^{B,i} = 1$. Note that this transformation is independent of state estimates.

The measurement model follows as

$$\tilde{\mathbf{m}}_{k+1}^{B,i} = \frac{\mathbf{D}_I^B(\mathbf{q}_{k+1})(\mathbf{m}^{I,i} - \mathbf{r}_{k+1}^I)}{[1\,0\,0]\,\mathbf{D}_I^B(\mathbf{q}_{k+1})(\mathbf{m}^{I,i} - \mathbf{r}_{k+1}^I)} + \delta \mathbf{m}_{k+1}^{B,i}$$

where $\delta \mathbf{m}^{B,i}$ denotes zero-mean Gaussian noise as measurement noise with covariance matrix $\mathbf{R}_{3x3}^{\mathrm{feat}}$. This function can be represented as

$$\tilde{\mathbf{m}}_{k+1}^{B,i} = \frac{\mathbf{h}'}{\mathbf{L}\mathbf{h}'}|_{k+1} + \delta \mathbf{m}_{k+1}^{B,i}$$

with $\mathbf{L} = [1\,0\,0]$ and $\mathbf{h}'|_{k+1} = \mathbf{D}_I^B(\mathbf{q}_{k+1})(\mathbf{m}^{I,i} - \mathbf{r}_{k+1}^I)$.

Computing the linearization using quotient rule results in

$$\mathbf{H}_{k+1}^i = \frac{\partial \tilde{\mathbf{m}}^{B,i}}{\partial \mathbf{x}}|_{k+1} = \frac{\frac{\partial \mathbf{h}'}{\partial \mathbf{x}}\mathbf{L}\mathbf{h}' - \mathbf{h}'\mathbf{L}\frac{\partial \mathbf{h}'}{\partial \mathbf{x}}}{(\mathbf{L}\mathbf{h}')^2}|_{k+1}$$

which only requires to compute partial derivatives of $\mathbf{h}'$:

$$\frac{\partial \mathbf{h}'}{\partial \mathbf{x}}|_{k+1} = \left[\frac{d\mathbf{D}_I^B(\hat{\mathbf{q}}_k)}{d\mathbf{q}}(\mathbf{m}^{I,i} - \hat{\mathbf{r}}_{k+1}^I) \quad -\mathbf{D}_I^B(\mathbf{q}_{k+1}) \quad \mathbf{0}_3 \right]$$

Note that the full $\mathbf{H}_{k+1}$-matrix stacks the 7 measurement matrices $\mathbf{H}_{k+1}^i$ corresponding to every marker with its observed unit vector $\tilde{\mathbf{m}}^{B,i}$:

$$\mathbf{H} = \begin{bmatrix} \mathbf{H}^1 \\ \vdots \\ \mathbf{H}^7 \end{bmatrix}$$

The full measurement $\tilde{\mathbf{m}}^B$ stacks the measurements of all 7 markers:

$$\tilde{\mathbf{m}}^B = \begin{bmatrix} \tilde{\mathbf{m}}^{B,1} \\ \vdots \\ \tilde{\mathbf{m}}^{B,7} \end{bmatrix}$$

For the full measurement noise covariance matrix $\mathbf{R} = \mathbf{R}_{21x21}$ for the stacked measurement $\tilde{\mathbf{m}}$ we have $\mathbf{R} = \text{diag}([\mathbf{R}_{3x3}^{\text{feat}} \cdots \mathbf{R}_{3x3}^{\text{feat}}])$.

Actual gyroscope and accelerometer signals were analyzed for noise variance. Under the assumption of uncorrelated noise, the values detailed in Table 5.2 were chosen.

Table 5.2: EKF: Noise Covariances

$\mathbf{Q}^{\text{gyro}}$	$= \text{diag}_3([4E^{-6}, 2E^{-6}, 2E^{-6}])$
$\mathbf{Q}^{\text{acc}}$	$= \text{diag}_3([0.25, 0.25, 0.25])$
$\mathbf{R}_{3x3}^{\text{feat}}$	$= \text{diag}_3([0.001, 0.001, 0.001])$

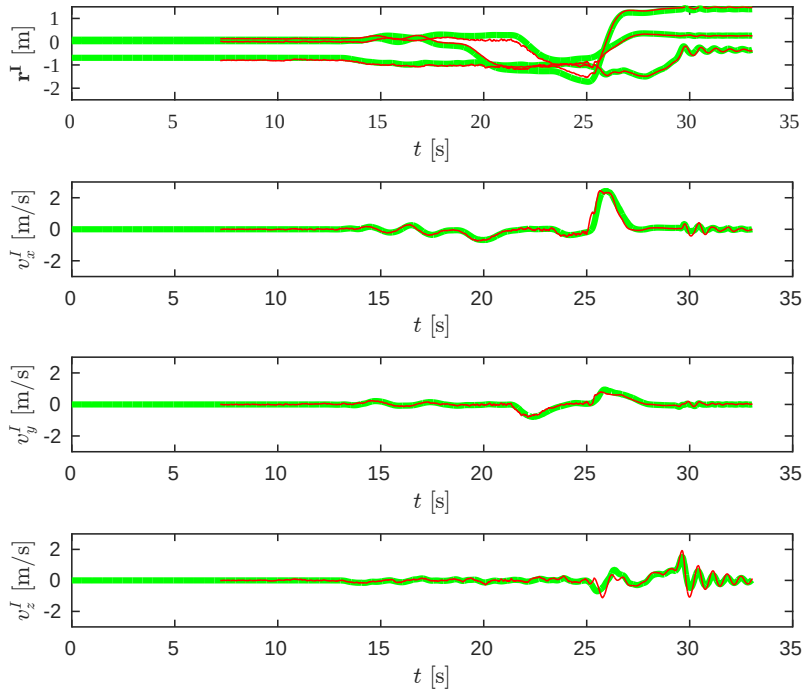
5.2.3 Performance Results

This section discusses the estimators' performance. The quadrotor was flown in the motion capture area to acquire ground truth data. The maneuvers closely resemble

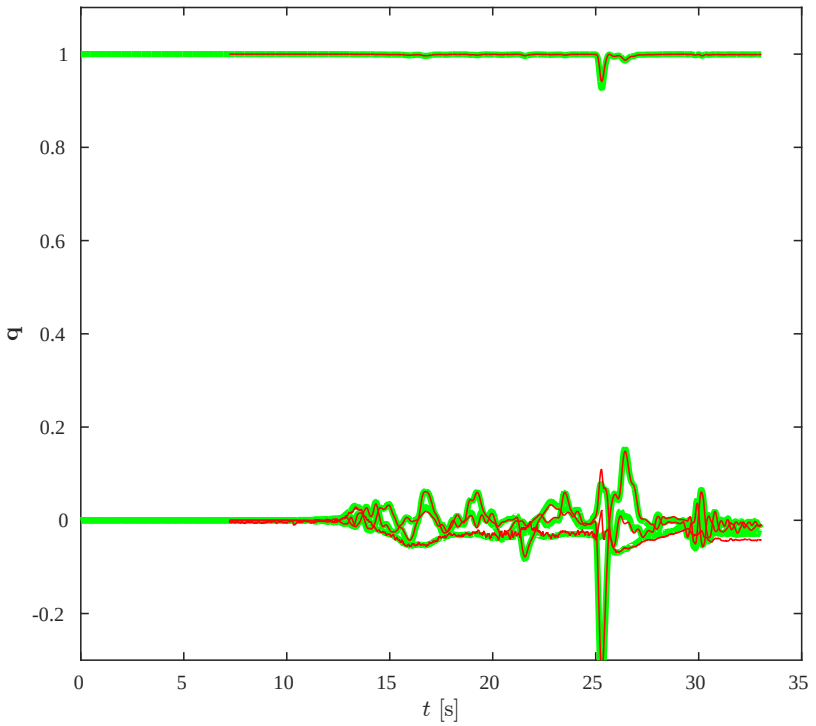
approaching a target region.

Hybrid filter Figure 5-6 shows estimates generated by the hybrid filter for positions, velocities and orientation against ground truth data acquired through the motion capture system. It reveals a mostly smooth and accurate tracking of the actual states, apart from small but noticeable errors in z-estimates during aggressive acceleration at about $t = 25s$. The filtering process does not seem to lead to noticeable phase-delays.

EKF Figure 5-7 plots the EKF's estimates against ground-truth data acquired through the motion capture system. Including the small but noticeable errors in z-estimates during aggressive acceleration at about $t = 25s$, it reveals a similar estimation quality compared to the hybrid filter, while posing less computational load since, unlike with the hybrid filter, no full pose reconstruction is necessary prior to the kalman filtering.
Both estimators were also tested with the overall integrated system (estimator and controller in the loop). The EKF proved to perform slightly better in these experiments. Chapter 7 presents these experiments.

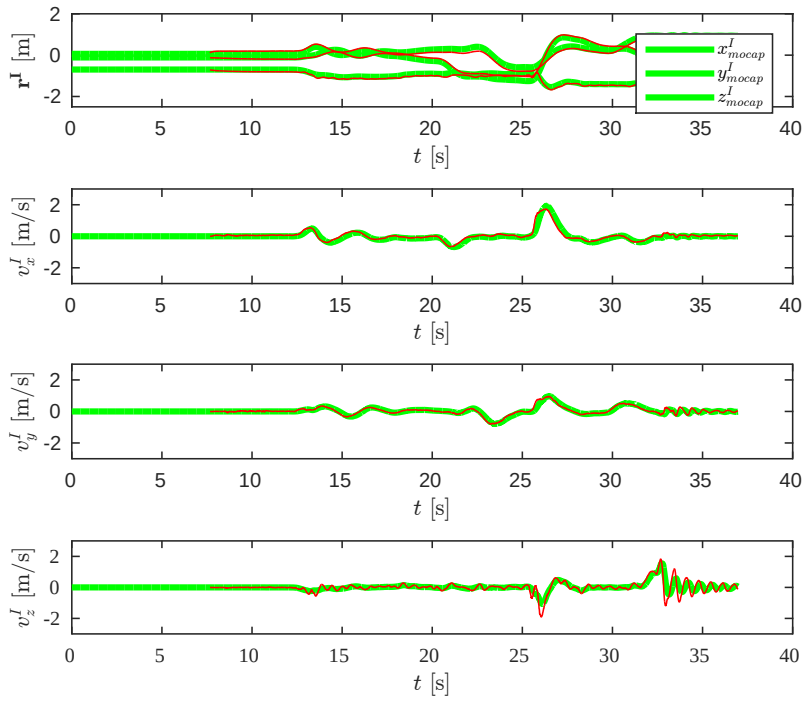


(a) Position (top plot: (x^I,y^I,z^I) end at (1,0,-1.5) respectively) and velocity estimates.

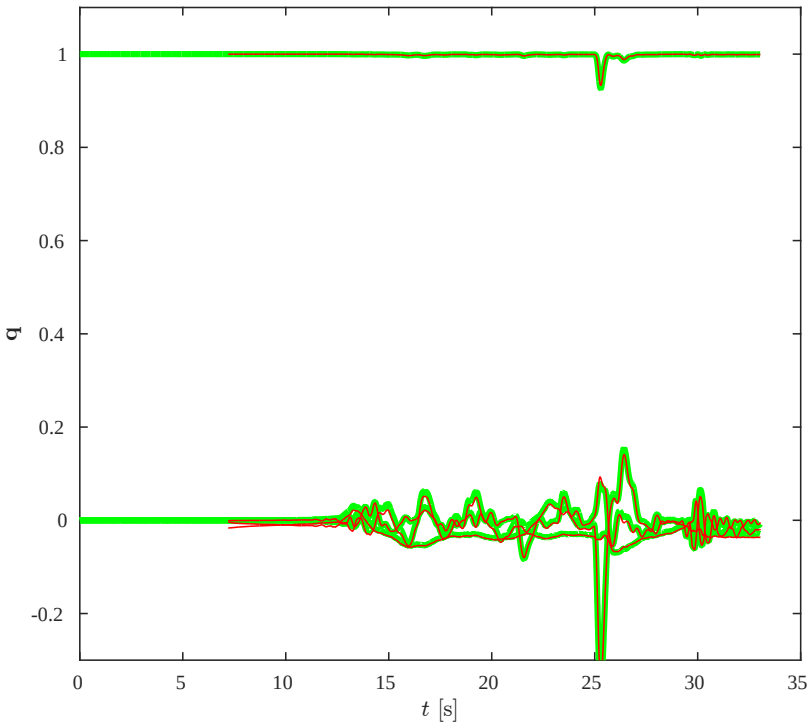


(b) Orientation Estimates: $q_0..q_3$

Figure 5-6: Hybrid Fullstate Filter Performance. Motion capture reference (green) vs hybrid filter estimate (red). Data reveals smooth and accurate estimates.



(a) Position (top plot: (x^I,y^I,z^I) end at (1,0,-1.5) respectively) and velocity estimates.



(b) Orientation Estimates: $q_0..q_3$

Figure 5-7: Fullstate EKF Performance: motion capture reference (green) vs EKF (red). Data reveals smooth and accurate estimates.

Summary

This chapter presented the visual detection of a scene target and two estimation algorithms to generate fullstate estimates. Both use priorly known visual features in the environment and onboard accelerometer and gyroscopic measurements. The estimators were evaluated with experimental data and proved to provide smooth and accurate fullstate estimates.

The following chapter presents multiple linear and nonlinear control schemes to control the quadrotor. The controller performance is evaluated in simulation.

Chapter 6

Control Systems

This chapter introduces the control algorithms used to guide the quadrotor to achieve two goals: (a) hover in front of visual cues and (b) fly towards a target region.

First, a control scheme is discussed that cascades the 6DOF-dynamics into a position- and an orientation subsystem. Then, three control algorithms are presented: a nonlinear orientation controller and two position controllers. While both of them operate on a fullstate estimate, they differ considerably: The first position controller acts on linearized dynamics and follows a classical PD approach. This approach requires separate path planning when more complex flight maneuvers are required. The second position control approach uses tensor-train-decomposition-based compressed computation to solve a nonlinear stochastic optimal control problem globally for the entire state space, taking the system's stochasticity, nonlinearity and actuator constraints into account and computing optimal controller outputs for every point in state-space.

6.1 Controller Goals

The control algorithms presented in this chapter aim to control the quadrotor to: (a) hover at a specified 3D-position with given yaw-angle and (b) fly through a specified target region.

Hover During the hover flight, the quadrotor aims to stabilize itself at a desired point in 3D-position space $\mathbf{r}^{I*}$ with given yaw-angle ψ^* and zero velocities (steady state). We therefore define the 4-dimensional vector $\mathbf{x}^*_{4,\text{hover}}$:

$$\mathbf{x}^*_{4,\text{hover}} = \begin{bmatrix} \mathbf{r}^{I*} \\ \psi^* \end{bmatrix}$$

Target Region With the scenario of approaching and passing on opening in mind, we define a *target region* in the 8-dimensional translational- and yaw-angle state space with *center* $\mathbf{x}^*_{8,\text{target}}$:

$$\mathbf{x}^*_{8,\text{target}} = \begin{bmatrix} \mathbf{r}^{I*} & \psi^* & \mathbf{v}^{I*} & r^* \end{bmatrix}^T$$

and a *width* of

$$\triangle\mathbf{x}^*_{8,\text{target}} = \begin{bmatrix} \triangle\mathbf{r}^{I} & \triangle\psi & \triangle\mathbf{v}^{I} & \triangle r \end{bmatrix}^T$$

Note that this defines a position- *and velocity*-subspace Ƀ of the full 12-dimensional state space X_{12} where $\mathbf{x} = [\mathbf{r}^I\ \eta^I\ \mathbf{v}^I\ \mathbf{\Omega}]^T \in X_{12}$:

$$Ƀ = \{\mathbf{x} \in X_{12} : (|x^{I*} - x^I| < \frac{\triangle x^I}{2}) \wedge \cdots \wedge (|v_x^{I*} - v_x| < \triangle v_x^I) \wedge \cdots\}$$

This effectively sets desired terminal values for both position- and velocity-subspace and can thus be compared to trajectory planning setups with boundary conditions on the position- and velocity-space. This region can be used to define a positional-target region, with small "entering" velocities v_y^I and v_z^I and a positive v_x^I, thus perfectly suiting the scenario setup of approaching and passing a target region in front of a scene target.

6.2 Cascaded Control Scheme for Fullstate Control

This section introduces the popular cascaded control scheme for quadrotors as e.g. described in [42] and presents modifications to suit the use cases of this thesis. The cascaded control scheme casacdes the 6DOF-dynamics into translation- and orientation-dynamics. Starting from the basic model introduced in 4.1.2 the cascaded control scheme is presented, underlying assumptions are discussed and an input reparametrization is introduced.

Recalling the basic dynamic model for a quadrotor model with states

$$\mathbf{x} = \begin{bmatrix} \mathbf{r}^I & \mathbf{v}^I & \eta & \mathbf{\Omega} \end{bmatrix}^T$$

given by

$$\begin{aligned} \dot{\mathbf{r}}^I &= \mathbf{v}^I \\ \dot{\mathbf{v}}^I &= \mathbf{G} + \mathbf{D}_\psi \mathbf{D}_\theta \mathbf{D}_\phi \frac{\mathbf{T}^B}{m} \\ \dot{\eta} &= \mathbf{W}^{-1} \mathbf{\Omega} \\ \mathbf{J}\dot{\mathbf{\Omega}} &= \begin{bmatrix} \tau_{roll} \\ \tau_{pitch} \\ \tau_{yaw} \end{bmatrix} - \mathbf{\Omega} \times \mathbf{J}\mathbf{\Omega} \end{aligned}$$

it becomes clear that a simple cascaded control scheme can be derived under minor assumptions. This cascaded scheme can handle the system's underactuation when trying to control the 3D-position $\mathbf{r}^I$ and yaw-angle ψ of the quadrotor.

$\mathbf{T}^B$ is assumed to be aligned with the body-frame z-axis and can be represented as $\mathbf{T}^B = \begin{bmatrix} 0 & 0 & T \end{bmatrix}^T$. This neglects aerodynamic effects mentioned in section 4.2.1, but still gives a good approximation since the major force acting on the quadrotor body is the thrust produced by the propellers - whose axes are aligned with the body-frame z-axis.

The resulting model shows *hierarchical* dynamics where the three body-frame torques

τ_i fully drive the 3DOF-second-order dynamics of η and the four $\begin{bmatrix} T & \theta & \phi & \psi \end{bmatrix}^T$, total body-frame thrust and orientation, drive the 3DOF-second-order dynamics of $\mathbf{r^I}$.
Note that the input to the $\mathbf{r}^I$-dynamics, $\mathbf{D}_\psi\mathbf{D}_\theta\mathbf{D}_\phi \begin{bmatrix} 0 & 0 & T \end{bmatrix}^T$, is a 3D-vector in the inertial frame. A 3D-vector can be fully parametrized by two angles and its magnitude. We reparametrize

$$\mathbf{D}_\psi\mathbf{D}_\theta\mathbf{D}_\phi \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} = \mathbf{D}_{\theta'}\mathbf{D}_{\phi'} \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix}$$

The dynamics can then be written as

$$\begin{aligned} \dot{\mathbf{r}}^\mathbf{I} &= \mathbf{v}^\mathbf{I} \\ \dot{\mathbf{v}}^\mathbf{I} &= \mathbf{G} + \frac{\mathbf{D}_{\theta'}\mathbf{D}_{\phi'}}{m} \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} \\ \dot{\eta} &= \mathbf{W}^{-1}\mathbf{\Omega} \\ \mathbf{J}\dot{\mathbf{\Omega}} &= \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} - \mathbf{\Omega} \times \mathbf{J}\mathbf{\Omega} \end{aligned}$$

With this reformulation, a cascaded control scheme can be applied:

The 3DOF-η-dynamics are controlled by a closed-loop feedback controller with the three τ_i. This structure constitutes the *inner-orientation-loop* that controls the system to achieve a reference orientation $\eta^* = \begin{bmatrix} \psi^* & \theta^* & \phi^* \end{bmatrix}^T$.

For controlling the dynamics of $\mathbf{r^I}$, it is assumed that the inner-orientation-loop-controller achieves sufficiently faster closed-loop η-dynamics than the $\mathbf{r^I}$-dynamics. It can then be assumed that, in the $\mathbf{r}'$-dynamics, $\theta = \theta^*$ and $\phi = \phi^*$.
With this assumption the 3DOF-second-order $\mathbf{r^I}$-dynamics can be controlled by the three $\begin{bmatrix} T^* & \theta'^* & \phi'^* \end{bmatrix}^T$. (Note that with given ψ, (ϕ^*, θ^*) can be derived from (ϕ'^*, θ'^*),

cf. Appendix B.1). This structure constitutes the *outer-position-loop*.
The resulting scheme is illustrated in Figure 6-1. Note that both controllers each control a 6-dimensional model.

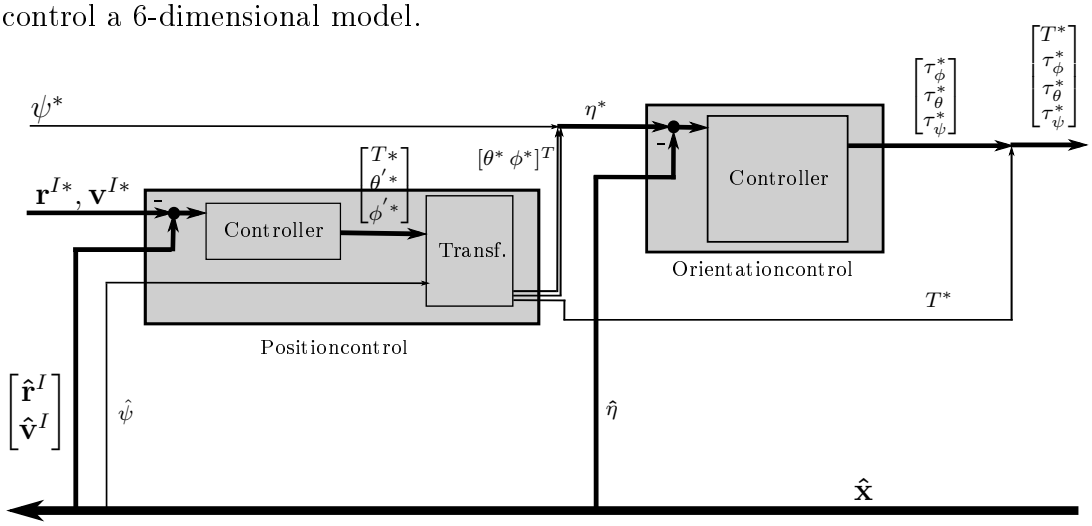


Figure 6-1: Cascaded Control Scheme

Oftentimes, the dynamics are oftentimes further simplified by linearizing around hover-conditions $\tau_i = \theta = \phi = p = q = r = v_x^I = v_y^I = v_z^I$ and $T = -m \cdot g$. This removes all higher-order coupling terms, which in turn removes the coupling between the DOF. Simple PID-control can then be applied to both position- and orientation control.
In this thesis, linearization is only applied to the $\mathbf{r}^I$-dynamics in order to derive a simple outer-loop-PD-position-controller (6.3). For the inner-orientation-loop, a basic nonlinear control concept is used.

Final controller output The overall controller output variables are total thrust T^* and body-frame torques τ_i^* (cf. Figure 6-1) which eventually need to be converted into PWM-signals. This is done in two steps: First, the controller output is transformed into four desired motor speeds using the relation described in 4.2.2. In software, this still happens in the same controller-POD. The *motorCommander*-POD then converts the desired motor speeds into PWM-values which are sent to the Arduino. The algorithms running in the *motorCommander* are described in more detail in Section 6.5.

Inner-orientation-loop This section presents the nonlinear orientation controller. It follows a basic nonlinear control concept known as computed torque as *e.g.* presented in [54]. From the η-dynamics

$$\dot{\eta} = \mathbf{W}^{-1}\boldsymbol{\Omega}$$
$$\mathbf{J}\dot{\boldsymbol{\Omega}} = \tau - \boldsymbol{\Omega} \times \mathbf{J}\boldsymbol{\Omega}$$

it follows

$$\begin{aligned}\ddot{\eta} =& \dot{\mathbf{W}}^{-1}\boldsymbol{\Omega} + \mathbf{W}^{-1}\dot{\boldsymbol{\Omega}} = \\ & \dot{\mathbf{W}}^{-1}\boldsymbol{\Omega} + \mathbf{W}^{-1}\mathbf{J}^{-1}(\tau - \boldsymbol{\Omega} \times \mathbf{J}\boldsymbol{\Omega})\end{aligned}$$

Following computed torque approaches with PD control, the controller output τ results as

$$\tau = \boldsymbol{\Omega} \times \mathbf{J}\boldsymbol{\Omega} + \mathbf{J}\mathbf{W}\left(-\dot{\mathbf{W}}^{-1}\boldsymbol{\Omega} - (\mathbf{K}_{\mathrm{P},\eta}\mathbf{e}_{\eta} + \mathbf{K}_{\mathrm{D},\eta}\mathbf{e}_{\dot{\eta}})\right)$$

with $\mathbf{K}_{\mathbf{P},\eta}$ and $\mathbf{K}_{\mathbf{D},\eta}$ being diagonal gain matrices and $\mathbf{e}$ denoting the control error. The chosen controller gains can be found in Table A.3.

6.3 PD Position Control

This section and the following now discuss two different *position* controllers: First, a simple PD controller that works on linearized translational $\mathbf{r}^I$-dynamics. Second, a nonlinear stochastic optimal controller that addresses the nonlinearity, stochasticity and actuator constraints of the system. The PD controller is introduced in this section.

This controller is used as a hover controller and for comparison purposes against the stochastic optimal controller to guide to quadrotor to the target region center.

The linearized $\mathbf{r}^I$-dynamics follow as

$$\dot{\mathbf{r}}^{\mathbf{I}} = \mathbf{v}^{\mathbf{I}}$$
$$\dot{\mathbf{v}}^{\mathbf{I}} = \begin{bmatrix} \triangle\theta' g \\ -\triangle\phi' g \\ \triangle T/m \end{bmatrix}$$

A simple PD-control scheme can then be applied:

$$\begin{bmatrix} \theta'^* \\ \phi'^* \\ T^* \end{bmatrix} = -\mathbf{K}_{\mathbf{P},\mathbf{r}^\mathbf{I}}\, \mathbf{e}_{\mathbf{r}^\mathbf{I}} - \mathbf{K}_{\mathbf{D},\mathbf{r}^\mathbf{I}}\, \mathbf{e}_{\mathbf{v}^\mathbf{I}} + \begin{bmatrix} 0 \\ 0 \\ -m \cdot g \end{bmatrix}$$

with $\mathbf{K}_{\mathbf{P},\mathbf{r}^\mathbf{I}}$ and $\mathbf{K}_{\mathbf{D},\mathbf{r}^\mathbf{I}}$ being diagonal gain matrices and $\mathbf{e}$ denoting the control error. The chosen values can be found in Table A.3. They are chosen to primarily allow accurate hover performance, but also such that the system converges faster without overshooting in y^I- and z^I-direction than in x^I-direction. For some initial conditions this setup allows to mimic the maneuver of flying towards the target region center-position with low v_y, v_z, but a positive v_x-velocity when entering the target region by simply moving the position reference $\mathbf{r}^{I*}$.

The pitch- and roll-angles were saturated at 0.4 rad. Note that with this setup, the quadrotor's orientation is considered an *input/actuator* to the $\mathbf{r}^I$-dynamics. A saturation of the orientation therefore poses an actuator constraint which falls outside the realm of linear, unconstrained control.

6.4 Nonlinear Stochastic Optimal Motion Control

The simple PD position controller presented in 6.3 suffers from major simplifications: It does not address the nonlinear translational-dynamics, it is not optimal by any meaningful measure and it necessitates separate trajectory planning if more involved flight tasks are required. These flight tasks might require obstacle-avoiding trajecto-

ries or a non-zero final velocities or orientation. Shen et al. [52] introduce a trajectory planning approach that takes into account dynamic constraints. A nonlinear tracking controller is then used to track the trajectory.

A promising approach to *simultaneously* tackle the issues of dynamics-constrained trajectory planning and tracking control is nonlinear stochastic optimal control: The goal of optimal control is to find a control policy that minimizes the cost incurred by running a stochastic system from an initial state $\mathbf{x}_0$ under a control process $\mathbf{u}$. The following two sections pose a short introduction to the approach presented in [23]. This paper is recommended for detailed background information.

Let an autonomous - time-independent- system follow the following differential form:

$$d\mathbf{x}(t) = \mathbf{f}(\mathbf{x}(t), \mathbf{u}(t))dt + \sigma(\mathbf{x}(t))d\mathbf{w}(t)$$

where $\mathbf{x} \in X$, $O \subset X$ an open subset of X and $\mathbf{w}(t)$ denotes a Brownian motion. As control policy a Markovian control policy was chosen that only depends on the current state: $\mathbf{u} = \mathbf{u}(\mathbf{x})$.
Furthermore, be τ either $\tau = \infty$ or the exit time when $\mathbf{x}(t)$ leaves O. With this setup, a meaningful cost to evaluate a control policy is:

$$J_{\mathbf{u}}(\mathbf{x}_0) = \mathbb{E}\left[\int_0^{\tau} e^{-\beta t} L(t, \mathbf{x}(t), \mathbf{u}(\mathbf{x}(t)))dt + \chi_{\tau<\infty} e^{-\beta\tau} g(\mathbf{x}(\tau))\right]$$

where χ represents that characteristic function that is 1 if the process comes to an end and 0 otherwise.

This setup allows to penalize time, state-deviation and control effort while taking into account the stochasticity of the process.

The solution to this stochastic optimal control problem is an optimal control policy $\mathbf{u}^*$ that minimizes $J_{\mathbf{u}}(\mathbf{x}_0)$ for all $\mathbf{x_0}$. Note the optimal control policy $\mathbf{u}^*$ represents a solution that solves the stochastic optimal control problem globally for the entire state space.

6.4.1 Tensor-train-decomposition-based Solution

Common techniques to solve this setup involve discretizing the problem. To this end, the continuous time-domain and state-space are discretized. Due to the stochastic nature of the system, the problem translates into a *discrete Markov process*. This method is known as *Markov-chain approximation* (MCA). Above motioned problem setup of finding the optimal control action then translates into a *Markovian Decision Process* (MDP).

A popular approach to solve these seeks an optimal value function $V(\mathbf{x})$ with $V(\mathbf{x}) = \inf_{\mathbf{u}} J_{\mathbf{u}}(\mathbf{x})$. The optimal control at each state can then be inferred by finding the control action that minimizes the expected value function after applying the control action. This approach is known as dynamic programming and can be approximately solved by value iteration or policy iteration. Guarantees do exist that, with finer discretization, the Markov-chain approximation converges to the original continuous process and importantly, the value function does so, too. The optimal control policy inferred from these approximations therefore converges to the optimal control for the continuous setup.
However, this approach suffers from the well-known curse of dimensionality since the number of discretization nodes is exponential in dimensionality.

Recent research utilizes compression techniques to execute computations on high-dimensional functions. Gorodetsky et al. introduce the use of tensor-train decomposition to exploit low-rank structures occurring in the data that represents the discretized value function [23].

Tensor-train decomposition resembles the commonly known singular value decomposition (SVD), but is applied to tensors instead of matrices. It was demonstrated to be able to solve 7-dimensional nonlinear stochastic optimal control problems with actuator constraints in simulation [24].

The following section introduces the setup used to generate an optimal controller that flies a quadrotor towards a target region in an optimal way. The setup addresses

the stochasticity, nonlinearity and the quadrotor's actuator constraints.

6.4.2 Problem Formulation

Although tensor-train-decomposition-based solution approaches to optimal control problems have shown to be able to solve higher dimensions than naive, exact approaches, the proof of concept in this thesis limits itself to work with a 6-dimensional model. Section 6.2 on the cascaded control scheme presented an option to work with two 6-state models instead of the full 12-state quadrotor model: A position-controller controls the 6-dimensional $\mathbf{r}'$-position-dynamics; while an orientation-controller controls the 6-dimensional η-orientation-dynamics.
The orientation-controller needs to be able to track reference trajectories (more specifically, orientation trajectories) since the position-controller outputs reference trajectories and assumes these to be tracked perfectly. The general formulation of a stochastic optimal control problem that drives the process into some goal region in state-space cannot be used for tracking problems. For the position-controller, however, it is nicely applicable if the reference position and velocity is fixed and can therefore be interpreted as static target region in state space. The approach presented in the following sections replaces the PD position controller from 6.3 with a stochastic optimal controller.

Dynamic Equations Recalling the equations for the $\mathbf{r}^I$-translational dynamics with $\mathbf{x} = [\mathbf{r}^I \ \mathbf{v}^I]^T$, a stochastic form follows as:

$$
\begin{aligned}
d\mathbf{r}^I &= \mathbf{v}^I dt + \sigma_r d\mathbf{w}_r(t) \\
d\mathbf{v}^I &= \left(\mathbf{G} + \frac{\mathbf{D}_{\theta'}\mathbf{D}_{\phi'}}{m} dt \begin{bmatrix} 0 \\ 0 \\ T \end{bmatrix} \right) dt + \sigma_v d\mathbf{w}_v(t)
\end{aligned}
$$

where

$$\sigma_r = \text{diag}([\sigma_x \ \sigma_y \ \sigma_z]) \qquad \sigma_v = \text{diag}([\sigma_{v_x} \ \sigma_{v_y} \ \sigma_{v_z}])$$
$$\mathbf{w}_r = \begin{bmatrix} w_x(t) & w_y(t) & w_z(t) \end{bmatrix}^T \qquad \mathbf{w}_v = \begin{bmatrix} w_{v_x}(t) & w_{v_y}(t) & w_{v_z}(t) \end{bmatrix}^T$$

Expanding the vector equations results in

$$\begin{aligned}
dx^I &= v_X^I dt + dw_x \\
dy^I &= v_Y^I dt + dw_y \\
dz^I &= v_Z^I dt + dw_z \\
dv_x^I &= c(\phi')s(\theta')\frac{T}{m}dt + dw_{v_x} \\
dv_y^I &= -\, s(\phi')\frac{T}{m}dt + dw_{v_y} \\
dv_z^I &= \left(g + c(\theta')c(\phi')\frac{T}{m}\right)dt + dw_{v_z}
\end{aligned}$$

and reveals the multiple nonlinear couplings of the (bounded) control input $\mathbf{u} = \begin{bmatrix} T & \phi' & \theta' \end{bmatrix}^T$.
Note that position-variables do not show up in the right-hand side of the equation. Therefore, the dynamics are independent of the position. This allows to compute a controller that drives the system towards, *e.g.*, the position-origin $\mathbf{r}^{I*} = \mathbf{0}$. In case that, during runtime, the system should be controlled not into the origin but into a non-origin/non-zero position reference $\mathbf{r}'^*$, the position coordinate system can simply be statically shifted ($\mathbf{r}' = \mathbf{r}^I - \mathbf{r}'^*$). When computing the optimal control policy $\mathbf{u}*$, a target region can therefore be set up around the position-origin.

Cost A meaningful setup aims to minimize the time the quadrotor needs to reach a target region. Time is therefore penalized. To smooth the control action, control effort is also penalized. Quadratic cost on *position*-states is added to enable faster convergence of the value- and policy iterations without introducing a trade-off between cost incurred by non-zero position and cost incurred by non-zero velocity (non-zero

velocity is needed to drive position errors to zero). In a pure min-time formulation the cost slowly propagates from a "zero cost region" into the entire state space. With quadratic cost on the position-subspace, every node in state-space is already assigned some cost, which sped up convergence.

The cost function that is to be minimized over $\mathbf{u}$ subject to the dynamic- and actuator constraints found above is therefore set up as follows:

$$J_{\mathbf{u}}(\mathbf{x}_0) = \mathbb{E}\left[\int_0^{\tau} e^{-\beta t} L(t, \mathbf{x}(t), \mathbf{u}(\mathbf{x}(t)))dt + \chi_{\tau<\infty} e^{-\beta\tau} g(\mathbf{x}(\tau))\right]$$

with

$$L = \begin{Bmatrix} \varrho_t + \mathbf{r}^T \mathbf{S}^r \mathbf{r} + \mathbf{u'}^T \mathbf{S}^u \mathbf{u'}, & \text{for } \mathbf{x} \notin \text{Ƀ} \\ 0, & \text{else} \end{Bmatrix} \qquad g = 0$$

$$\mathbf{S}^r = \text{diag}([\varrho_x \ \varrho_y \ \varrho_z]) \qquad \mathbf{S}^u = \text{diag}([\varrho_T \ \varrho_\phi \ \varrho_\theta])$$

where $\text{Ƀ} \subset O \subset X$ denotes the target region as introduced in 6.1. Table A.4 lists the chosen actuator limits, costs and noise parameters.

Reflecting boundary conditions are used for the derivation of the MDP from the continuous stochastic system since they resulted in smoother approximations of the value function near the boundaries [23]. With this approach the process is a true infinite-time horizon problem (thus $\chi = 0$) since the process never comes to an end (which would happen when the state leaves O, i.e. the process leaves the state-space bounds. This is, however, prevented by reflecting boundary conditions). With the chosen cost function, the process only stops to accumulate cost when it is within Ƀ. The optimal control policy will therefore aim to bring the system into Ƀ. This justifies the way the target region Ƀ (as a region to fly towards, as described in Section 6.1) is used in the cost function.

Note that there are initial states near the boundaries where no admissible control policies exist that would reliably drive the process towards the target region (*e.g.* a pose right at the position-boundaries with the quadrotor's velocity vector pointing

straight towards that very boundary). However, the use of reflecting boundary conditions results in tweaked transition probabilities, s.t. the probabilities to transition to neighboring states that are outside of the boundaries are added to the probability of staying at the same state. Consequently, in these regions, it is very likely to stay at the current state. However, stochasticity introduces a small but non-zero likelihood to transition away from the boundary and leave these uncontrollable regions where (without stochasticity) no control policy exists that can drive the system towards the target region. Due to the very low likelihood the cost of these states is very high (intuitively because the system stays there for a long time before it, by stochasticity, transitions towards controllable states). This prevents the controller from considering to drive the system into the state-space boundaries. Recalling the overarching goal of flying the quadrotor through a target region that could represent an opening in a wall, we can exploit this characteristic to simulate the wall around the fictional opening (see Section 6.4.4 for details).

6.4.3 Controller Synthesis

Computation

This nonlinear stochastic optimal control problem has been solved with code [22] accompanying the work by Gorodetsky et al. [23] that implements the above introduced tensor-train-decomposition-based value- and policy iteration of an approximate MDP representing the continuous stochastic optimal control setup.

The computation of the value function $V(\mathbf{x})$ happens over multiple iterations until convergence is reached. At each iteration the computation approximates the value function tensor in low-rank tensor-train format with a specified rank, runs one value-iteration on this low-rank representation, runs policy-iterations in between and possibly adjusts the specified rank if an approximation tolerance is not met. Figure 6-2 plots the computation diagnostics. The controller computation was run for 8 iterations on a coarse discretization level of $n = 20$-nodes per dimension first. After about 50 iterations on a discretization level of $n = 60$, $||V||$ converged and oscillated

around its final value. $||V||$ denotes the integral of the value function over the entire state-space. The diagnostics also show that this tensor-train-decomposition based computation of the optimal control policy evaluates five orders of magnitude fewer nodes (of the $n = 60$-discretized state-space) than a naive solution would do - thus successfully addressing the curse of dimensionality. This computation took about 15 hours on a Jetson TK1. A naive solution evaluating five orders of magnitude more nodes would take about 15 years. Noticeably, after only 4 iterations at a discretization level of $n = 20$ the controller could fly the quadrotor successfully towards the target region for simple initial conditions (in front of the target region, no fast initial velocity that requires aggressive deceleration). The computation of this non-converged cost function took about 3 hours on the TK1. Note that this finding also indicates that an accurate approximation of the value function through the tensor-decomposition-algorithms more difficult near the bounds than away from the bounds. Gorodetsky et al. [23] present a more thorough analysis of this.

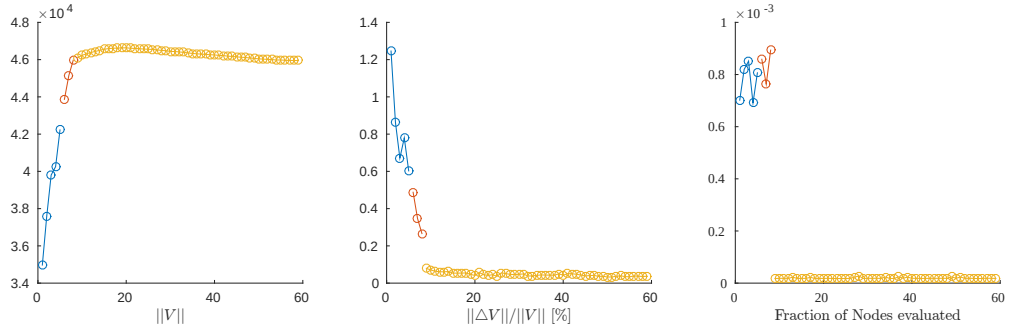


Figure 6-2: Controller Synthesis Diagnostics: 5 iterations on $n = 20$ (blue), 3 on $n = 20$ (red), 51 on $n = 60$. Plot on the left illustrates how $||V||$ converges. Right plot shows that the algorithm evaluates five orders of magnitude fewer nodes than a naive solution operating on all nodes in a linearly discretized state-space would do. This reduces computational time from about 15 years to 15 hours on a Jetson TK1.

Deployment

The resulting value function $V(\mathbf{x}) = \min_{\mathbf{u}} J_{\mathbf{u}}(\mathbf{x})$ is stored in tensor-train-decomposition compressed format. During flight, the optimal control policy

$$\mathbf{u}^*(\hat{\mathbf{x}}_k) = \begin{bmatrix} T^*(\hat{\mathbf{x}}_k) & \phi'^*(\hat{\mathbf{x}}_k) & \theta'^*(\hat{\mathbf{x}}_k) \end{bmatrix}^T$$

for the current state estimate $\hat{\mathbf{x}}_k$ is derived at every time step by finding the control input that results in the least expected value of the value function. In practical implementation, once the target region is reached, the quadrotor system switches back to the standard PD position-controller.
Like the PD-position controller, this controller outputs $\begin{bmatrix} T^* & \phi^* & \theta^* \end{bmatrix}^T$ and can therefore simply be switched in for the outer-loop-PD-position-controller described in 6.3.

6.4.4 Performance Results

Simulation on Simple Dynamics

Since the controller was synthesized using the simple dynamics that do not model aerodynamic or motor-dynamic related effects, it is advisable to evaluate its performance using a simulation based on these simple dynamics first.
The target region Ƀ was set to

$$\begin{aligned} Ƀ = \{\mathbf{x} \in X : (&-0.35 < x^I < 0.15, \quad -0.25 < y^I < 0.25, \quad -0.25 < z^I < 0.25 \\ &1 < v_x < 2, \quad -0.15 < v_y < 0.15, \quad -0.15 < v_z < 0.15)\} \end{aligned}$$

The state space bounds are chosen as

$$\begin{aligned} &-5.0 < x^I < 0, \quad -2.5 < y^I < 2.5, \quad -2.0 < z^I < 2.0 \\ &-5.0 < v_x < 5.0, \quad -5.0 < v_y < 5.0, \quad -5.0 < v_z < 5.0 \end{aligned}$$

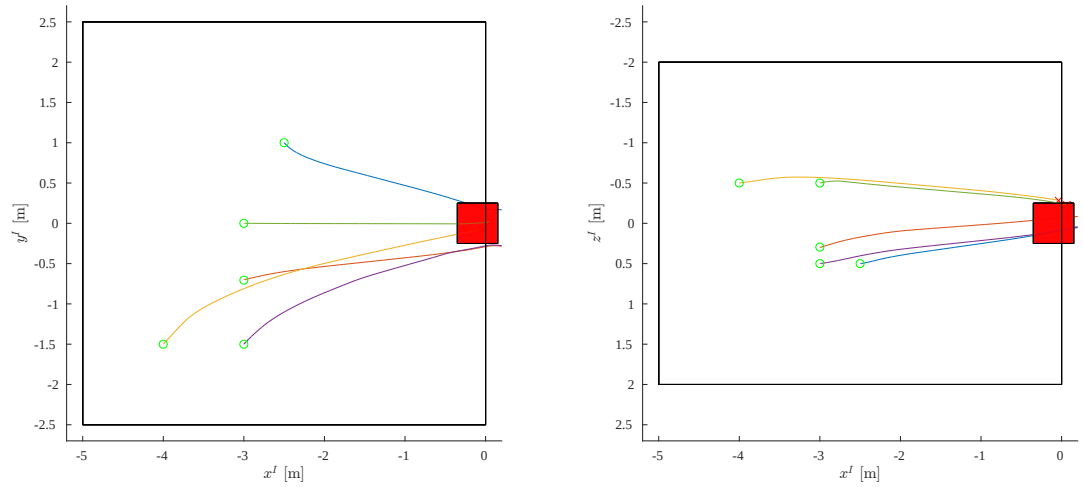
with controller limits

$$-mg - 3 < T < -mg + 3$$
$$-0.4 < \phi', \theta' < 0.4$$

This setup defines a positional-target region around the origin, with small "entering" velocities v_y^I and v_z^I and a positive v_x^I. As discussed in Section 6.4.2 the bounds on x^I are chosen to align with the target region. A slight offset in x^I-direction is introduced to ensure that states that lie both in the target region Ƀ as well as on the state-space bounds are not penalized. This setup perfectly suits the scenario of approaching and passing through a target region in a defined direction (here: x-direction).
Figure 6-3 gives a top- and side view of this setup with the target region in red and state space bounds in black. Trajectories from simulations starting from various initial conditions are shown. They illustrate how the controller successfully drives the quadrotor towards the target region for a wide range of initial conditions. The resulting 3D trajectories are shown on the right-hand side of Figure 6-4.

On the left, Figure 6-4 picks one trajectory and plots the quadrotor's velocities and controller output. It reveals that the controller accelerates the quadrotor and decelerates it if necessary to drive the system's fullstate, including positions *and velocities*, into the position- and velocity-target region Ƀ. The controller chooses smoothed, near-bang-bang controller action. The smoothed bang-bang control action with a coasting phase in-between is expected for min-time-problems with control costs and actuator constraints [38].

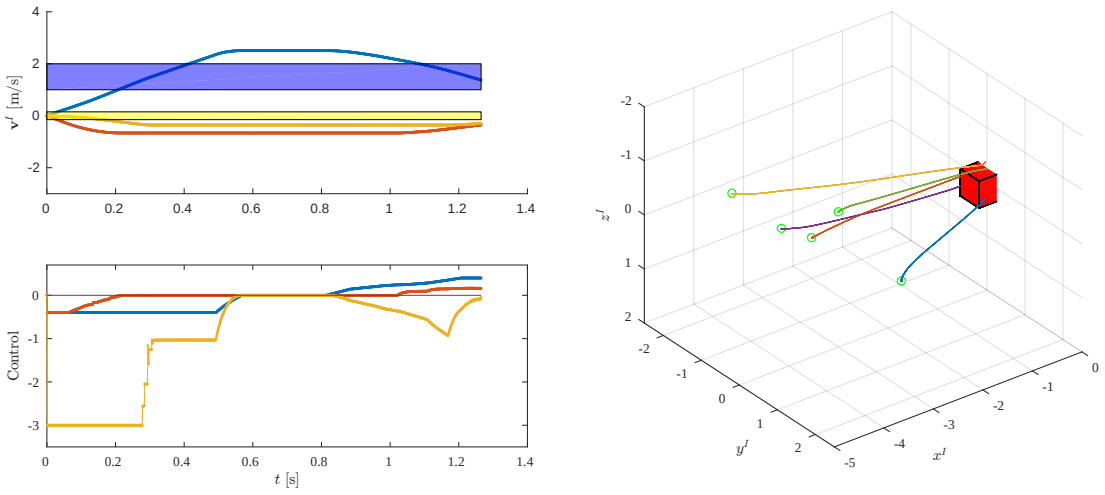
It is to be noted that the system is robustly driven into the position-sub space of the target region, while the velocity-target is not reached from all initial conditions accurately.



(a) Top view: X-Y motion

(b) Side view: X-Z motion

Figure 6-3: Stochastic Optimal Controller on Simple Dynamics: 2D Trajectories. Controller drives quadrotor into target region (red) in 6-dimensional state space. State-space bounds in black.



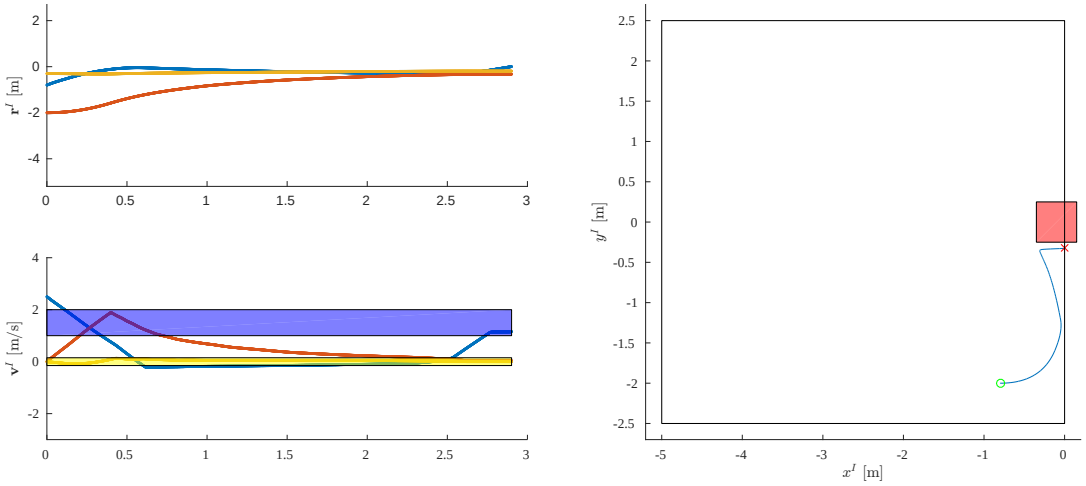
(a) Velocity (blue: v_x, red: v_y, yellow: v_z) and control action (blue: θ'^*, red: ϕ'^*, yellow: $T^* + mg$) that drives velocities into target regions.

(b) 3D Trajectories with target region (red)

Figure 6-4: Stochastic Optimal Controller on Simple Dynamics: Velocity, control and 3D trajectories.

Comparison to PD-position controller Figure 6-4 illustrates how the thrust is increased during pitch- and roll-maneuvers to compensate for the effective loss of gravity-balancing, vertical thrust. A linearized controller (like the PD position

controller in Section 6.3) is not aware of this effect. Additionally, Figure 6-5 reveals how the controller is capable of handling situations where most standard controllers would need trajectory planning since the only way to reach the target region (in both position and velocity-space) is a complex maneuver that includes backing up. The simulation was started at an initial position $\mathbf{r}^I_{\text{init}} = [-1\ -2\ -0.3]^T$ with an initial velocity of $\mathbf{v}^I_{\text{init}} = [2.5\,0\,0]^T$. The controller first decelerates the quadrotor in x^I-direction, backs it up while positioning it in front of the target region and eventually accelerates into the target region to enter it with the desired forward velocity $1\text{m/s} < v^I_x < 2\text{m/s}$.



(a) Position and velocity trajectories (blue: x^I, v^I_x, red: y^I, v^I_y, yellow: z^I, v^I_z)

(b) Top view: X-Y motion

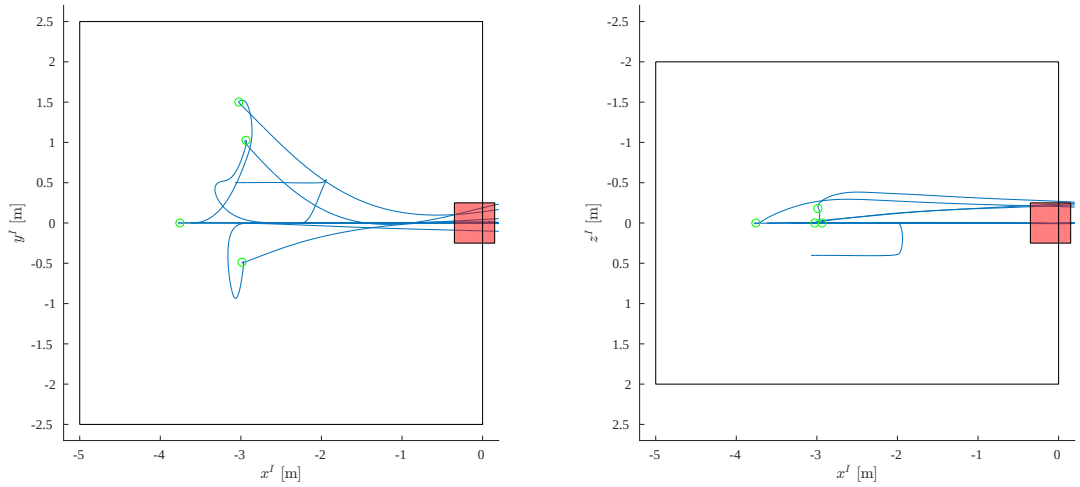
Figure 6-5: Stochastic Optimal Controller under Challenging Initial Condition. Controller drives quadrotor into target region (red) in 6-dimensional state space with complex maneuver.

Simulation on Full Dynamics in LCM-framework

The same controller was then run in the full LCM-PODs-software-framework, with the simulator introduced in Section 3.6 in-the-loop instead of the real quadrotor. To evaluate the controller performance independent of estimation performance, the controller was run directly on the simulated states.

Figure 6-6 and 6-7 show a single simulation run where the quadrotor was flown to various positions (marked with 'o ') using the PD position-controller. The operator then disabled the PD position-controller and enabled the stochastic optimal position controller. After passing the target region, the PD position-controller was reactivated and the quadrotor was flown to a new starting position. The plots illustrate how the controller can successfully handle the more complex, more realistic dynamics simulated with the full simulator and drives the quadrotor into the target region.

Since this simulation was run in the full software framework, the simulation can be turned into a real-world experiment by turning off the simulator-POD, rerouting *motorCommander*-commands to the actual motors instead of the simulator and acquiring IMU and vision data through the IMU acquisition- and vision-PODs, respectively. The next chapter gives experimental results of flying the quadrotor with this stochastic optimal controller towards a target region.



(a) Top view: X-Y motion

(b) Side view: X-Z motion

Figure 6-6: Stochastic Optimal Controller on Full Dynamics in LCM-framework: 2D Trajectories. Controller activated at 'o'. Controller drives quadrotor into target region (red) in 6-dimensional state space. State-space bounds (of computed stochastic optimal controller) in black.

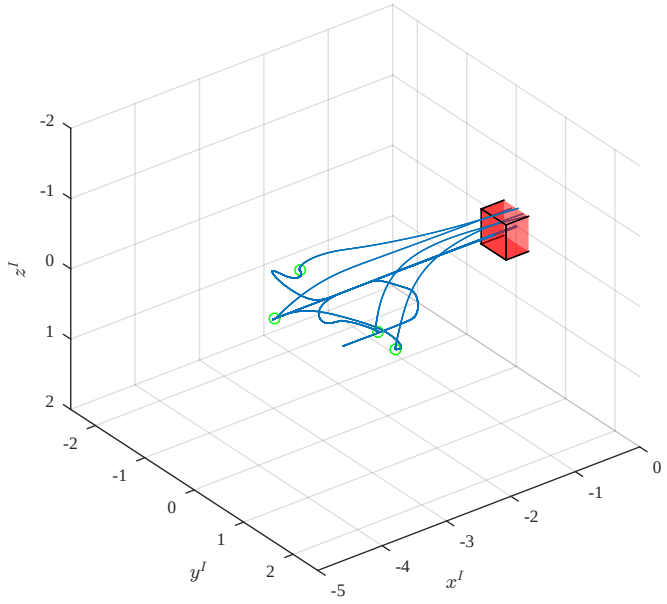


Figure 6-7: Stochastic Optimal Controller on Full Dynamics in LCM-framework: 3D Trajectories. Controller activated at 'o'.

6.5 Motor Control

This section describes the algorithm running in the *motorCommander*. Its task is to convert desired motor speeds ω_i^* into PWM-values κ_i. These PWM-values are then sent to the Arduino which generates PWM-signals that are sent to the ESCs to control the motors.

Using the battery-ESC-model described in 4.2.3 the *motorCommander* can compensate for otherwise occurring drops in motor speeds caused by dropping terminal voltage due to increased current draw on the battery. This model is purely geometric without dynamics.

Additionally, 4.2.4 showed the first-order delay that occurs in the actual motor speeds. This effect could be counteracted using lead-lag-compensators. Ideally, these would be implemented in the *motorCommander* as well. A saturation-block limits the PWM-values κ_i. The final signal flow in the motorCommander is shown in Figure 6-8.

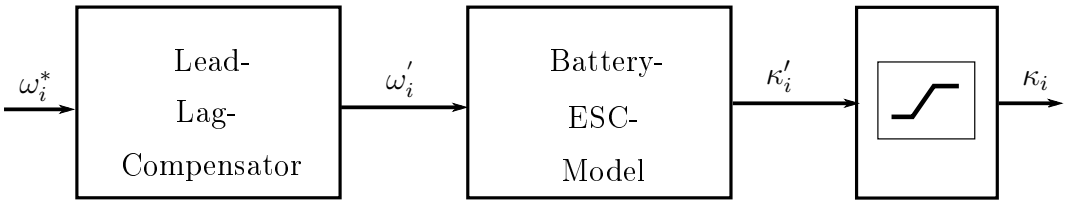


Figure 6-8: Motorcommander Signal Flow

Summary

This chapter described a control structure to control the quadrotor. The cascaded scheme cascades the underactuated dynamics into a position- and an orientation subsystem. Two controllers were discussed for the position subsystem: first, a commonly used PD controller that acts on linearized dynamics. Second, a globally optimal, stochastic optimal controller that takes into account the nonlinearity of the quadrotor dynamics and actuator constraints. In this chapter, the controller performance was evaluated in simulation.

An experimental performance evaluation of the overall integrated systems is presented in the next chapter. It shows the platform's hover capabilities and demonstrates that tensor-decomposition-based nonlinear stochastic optimal controllers can control the quadrotor to approach a target region.

Chapter 7

Experimental Flight Performance

This chapter presents experimental results to evaluate the system's integrated flight performance, thus evaluating the joint estimation- and control-performance. With the idea of exploratory disaster relief scenarios in mind, two useful experimental procedures to benchmark the usability are (a) hover flight and (b) target region flight. Both are described with their results in the following sections.

7.1 Experimental Scene Setup

Figure 7-1 visualizes the experimental scene setup: The (visually marked) scene target (blue) is located in the background. The visual-inertial estimation discussed in Chapter 5 uses this scene target. The virtual target region where the quadrotor will aim to fly through is placed in front of the markers. Note that eventually an opening like a window will replace the scene target and the target region will be placed right in the center of this window. This setup, however, necessitates a different estimation approach. A detailed discussion on this can be found in 8.2.2.

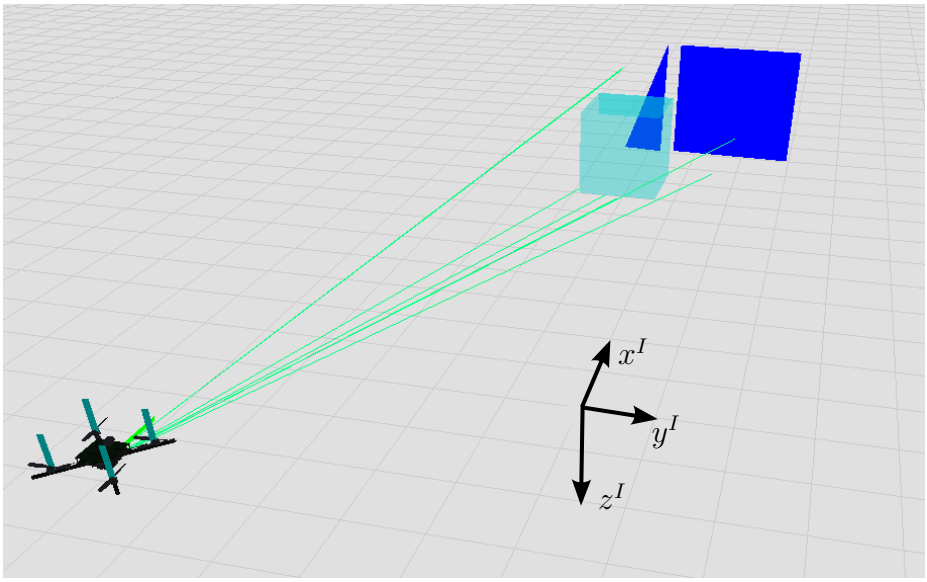


Figure 7-1: Experimental Scene Setup with Scene Target, Target Region and Quadrotor. Rays emanating from the quadrotors camera towards the 7 corners of the visual markers are displayed.

7.2 Hover Flight

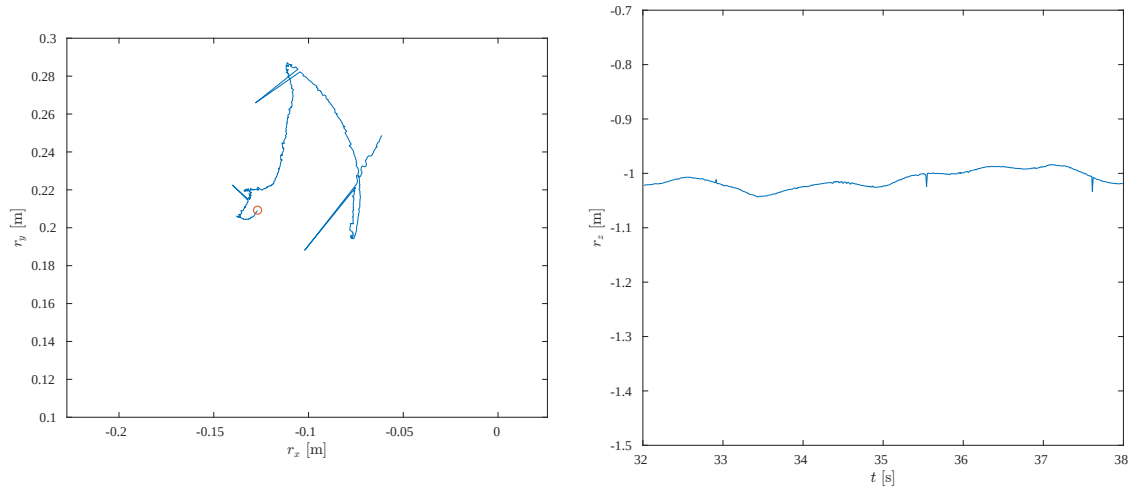
During the hover flight test, the quadrotor aims to stabilize itself at a desired point in 3D-position space $\mathbf{r}^{I*}_{\text{hover}}$ in front of the scene target with given yaw-angle ψ^*_{hover} and zero velocities (steady state). Section 6.1 defined the desired hover vector $\mathbf{x}^*_{4,\text{hover}}$. Recall that the markers $\mathbf{m}^{I,i}$ on the scene target are positioned at $\mathbf{m}^{I,i} = [3 \ * \ *]^T$. For this experiment, $\mathbf{x}^*_{4,\text{hover}}$ is chosen to be in front of the marker setup with

$$\mathbf{x}^*_{4,\text{hover}} = \begin{bmatrix} \mathbf{r}^I \\ \psi \end{bmatrix}_{\text{hover}} = \begin{bmatrix} 0 & 0 & -1 & 0 \end{bmatrix}^T$$

The controller used for hover is the simple cascaded PD-fullpose-controller setup described in 6.3 since this controller was specifically linearized and tuned for states around hover. $\mathbf{x}^*_{4,\text{hover}}$ was fed as reference into the cascaded PD-fullpose-controller. Hover performance with both the hybrid filter as well as the EKF in-the-loop are presented in the next section. The motion capture system provided data for ground truth comparison only.

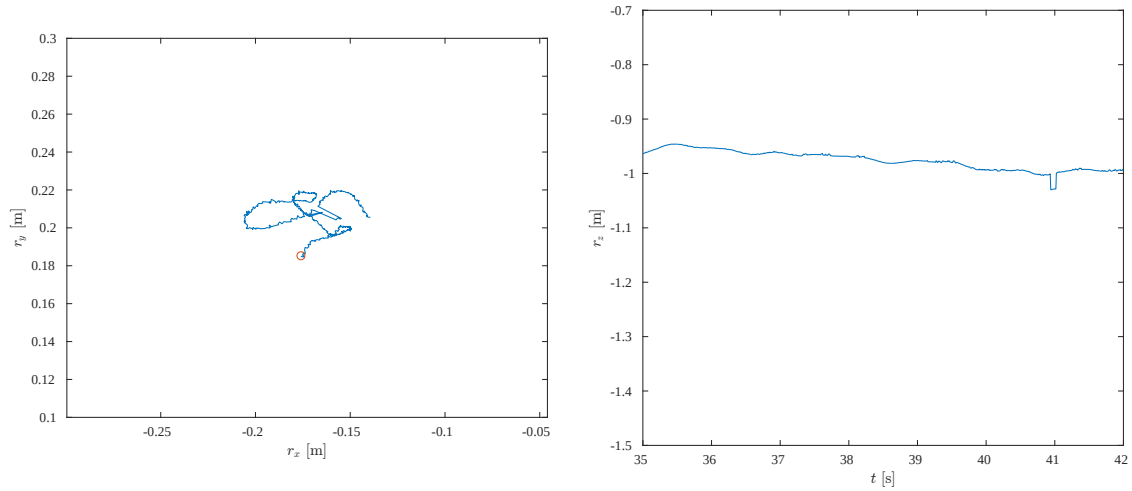
Performance Results Figure 7-2 shows the trajectories resulting from hovering in front of the markers with the cascaded PD-fullpose-controller, based on estimates from the hybrid filter and the EKF, respectively. The left plot shows the motion in the X-Y-plane while the right plots show the quadrotor's altitude over time; the shown data was acquired from the motion capture system and can be considered ground truth. Both trajectories show a constant offset of about .2m, but low position variability.

While both the hybrid filter as well as the EKF can keep the quadrotor stable around the hover point, the EKF shows lower variability and keeps the quadrotor within .1m. For the experiments described in the following section, the EKF was used.



(a) X-Y motion, hybrid filter

(b) Z-motion, hybrid filter



(c) X-Y motion, EKF

(d) Z-motion, EKF

Figure 7-2: Hover Flight Trajectories with hybrid filter (top) and EKF (bottom), data from motion capture system.

7.3 Target Region Flight

The second experimental benchmark consists of a more dynamic flight maneuver where the quadrotor tries to reach a target region in the 8-dimensional translational- and yaw-angle state space with center $\mathbf{x}^*_{8,\text{target}}$, starting from on initial, full steady-state $\mathbf{x}^*_{12,\text{init}}$.
We chose

$$\mathbf{x}^*_{12,\text{init}} = \begin{bmatrix}\mathbf{r}^{I*} & \eta^* & \mathbf{v}^{I*} & \mathbf{\Omega}^*\end{bmatrix}^T = \begin{bmatrix}-2.0 & 1.0 & -1.2 & 0_{1x9}\end{bmatrix}^T$$
$$\mathbf{x}^*_{8,\text{target}} = \begin{bmatrix}\mathbf{r}^{I*} & \psi^* & \mathbf{v}^{I*} & r^*\end{bmatrix}^T = \begin{bmatrix}0.5 & 0 & -1.35 & 0 & 1.5 & 0_{1x3}\end{bmatrix}^T$$

The target region's center $\mathbf{x}^*_{8,\text{target}}$ is chosen in front of the scene target to simulate the mission of approaching an opening with positive forward velocity, ready to fly through it. The initial state $\mathbf{x}^*_{12,\text{init}}$ is chosen at a distance to and off-centered to the target region center. The target *region* has width of

$$\triangle\mathbf{x}^*_{8,\text{target}} = \begin{bmatrix}0.5 & 0.5 & 0.5 & 0 & 1.0 & 0.3 & 0.3 & 0\end{bmatrix}^T$$

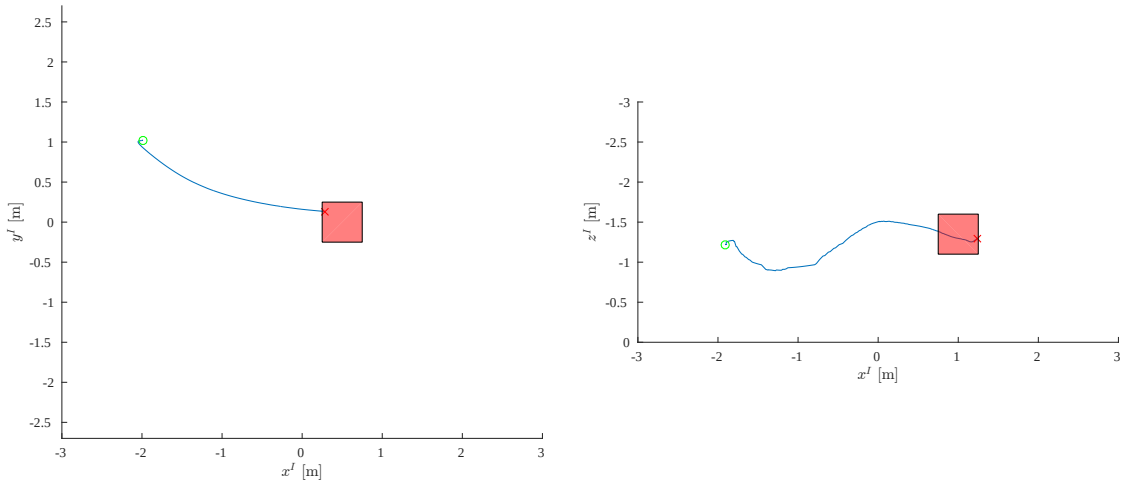
Note that this target region complies with the Ƀ chosen in Section 6.4.4 where a nonlinear stochastic controller was computed and evaluated in simulation. The regions only differ in a static shift of the target regions' position center which does not require to recompute a new controller (cf. 6.4.2).

7.3.1 PD Position Control

Using the cascaded PD controller the quadrotor was flown to $\mathbf{x}^*_{12,\text{init}}$ and after stable hover the position-yaw-reference for the outerloop-position-controller was switched to $\mathbf{x}^*_{8,\text{target}}(1:4)$. Note that this controller controls on positions and yaw-angle only, and uses velocities/rates as damping.
This experiment was run with the EKF in-the-loop. The motion capture system was used to acquire ground truth data only.

Performance Results

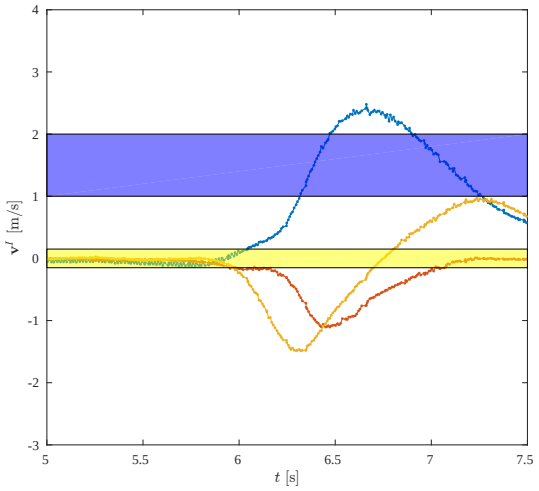
Figures 7-3 to 7-5 show the resulting trajectories. The plots on position-trajectories show that the quadrotor successfully and aggressively flies towards the target region. However, the quadrotor fails to properly keep altitude. Since this controller is designed on linearized dynamics, it is not aware that it effectively loses (globally-) vertical thrust that balances gravity once it tilts (factor $\cos(\phi')\cos(\theta')$). This can easily be solved by introducing $T' = T\cos(\theta')\cos(\phi')$ before linearizing: the linearization results in the same equations (with T' instead of T), but the thrust to be applied T is $T = T'/(\cos(\theta')\cos(\phi'))$, effectively taking care of the loss of vertical thrust when tilting. This solution is not considered here to stick to simple linearized controllers. Figure 7-4 reveals a maximum absolute velocity above $2\frac{m}{s}$. Figure 7-5 shows the aggressive orientation of up to 30 degrees and a resulting acceleration of up to $6m/s^2$. Note that the controller does have a forward velocity ($v_x^I > 0$) when entering the (position) target region, but also a considerably non-zero z^I-velocity v_z^I.



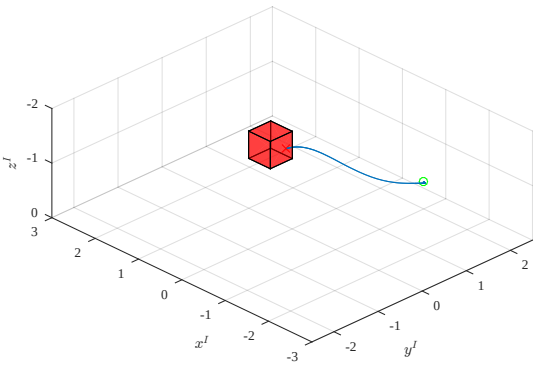
(a) X-Y motion

(b) X-Z-motion

Figure 7-3: Target Region Flight with EKF and PD Control, translational data from motion capture system. Quadrotor reaches target region (red).

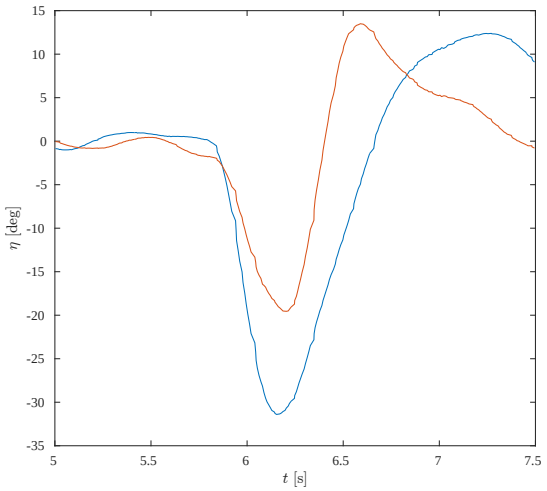


(a) velocities. blue: v_x^I, red: v_y^I, yellow v_z^I with corresponding velocity-target regions

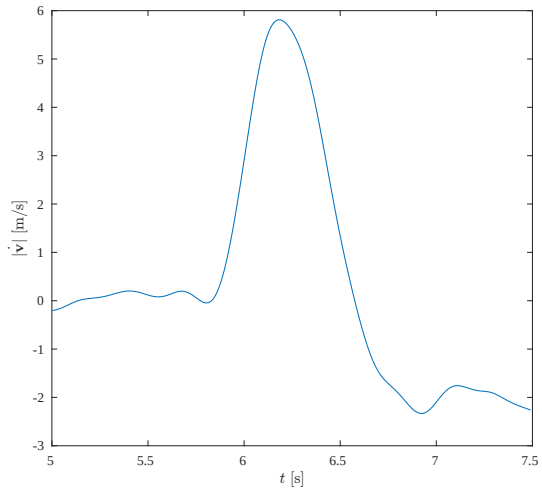


(b) 3D trajectory

Figure 7-4: Target Region Flight with EKF and PD Control, translational data from motion capture system. Quadrotor reaches target region.



(a) orientation: red: roll ϕ, blue: pitch θ



(b) acceleration of absolute velocity

Figure 7-5: Target Region Flight with EKF and PD Control, orientation and acceleration data from motion capture data.

7.3.2 Nonlinear Stochastic Optimal Motion Control

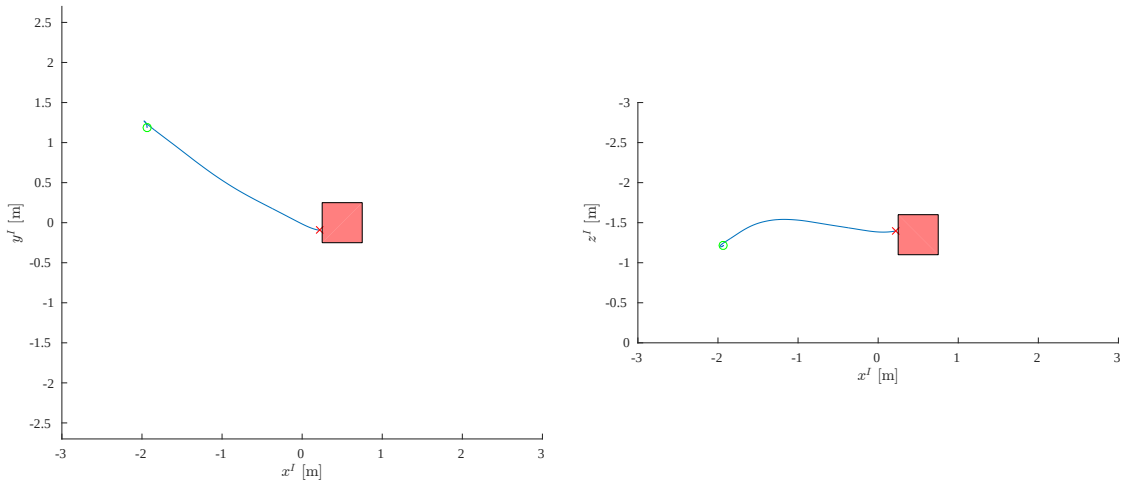
This section discusses to performance of the nonlinear stochastic optimal controller introduced in Section 6.4. Using the PD position-controller the quadrotor was flown to $\mathbf{x}^*_{12,\text{init}}$ and after stable hover, the PD position-controller was switched out for the

nonlinear stochastic controller with Ƀ specified above.
This experiment was run with the EKF in-the-loop. The motion capture system was used to acquire ground truth data only.

Performance Results

Figures 7-6 to 7-8 show the resulting trajectories. The plots on position-trajectories show that the quadrotor successfully and aggressively flies towards the target region. The resulting altitude-trajectory demonstrates how the controller's knowledge of the nonlinear dynamics helps to control altitude slightly better than the PD controller. Figure 7-4 shows maximum velocities above $2\frac{m}{s}$. Figure 7-5 shows the aggressive orientation of up to 25 degrees and a resulting acceleration of up to $5.5m/s^2$. Raw camera footage that reveals the aggressive angles chosen during the flight maneuver is shown in Figure 7-9.

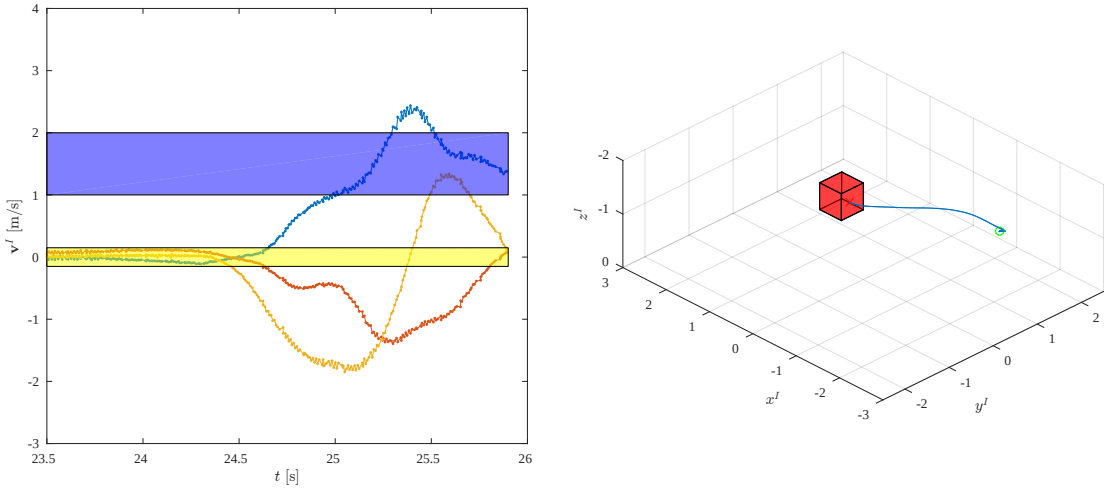
The velocities when entering the (position-) target region comply with the velocity-subspace of the target region: small velocity v_y^I, v_z^I, positive forward velocity v_x^I. This demonstrates one of the benefits the stochastic optimal controller approach: While a similar result (reaching a target region in position *and velocity*-subspace) can be achieved with carefully tuned PD control or trajectory planning methods, the stochastic optimal controller tries to achieve this "terminal condition" by design.



(a) X-Y motion

(b) X-Z-motion

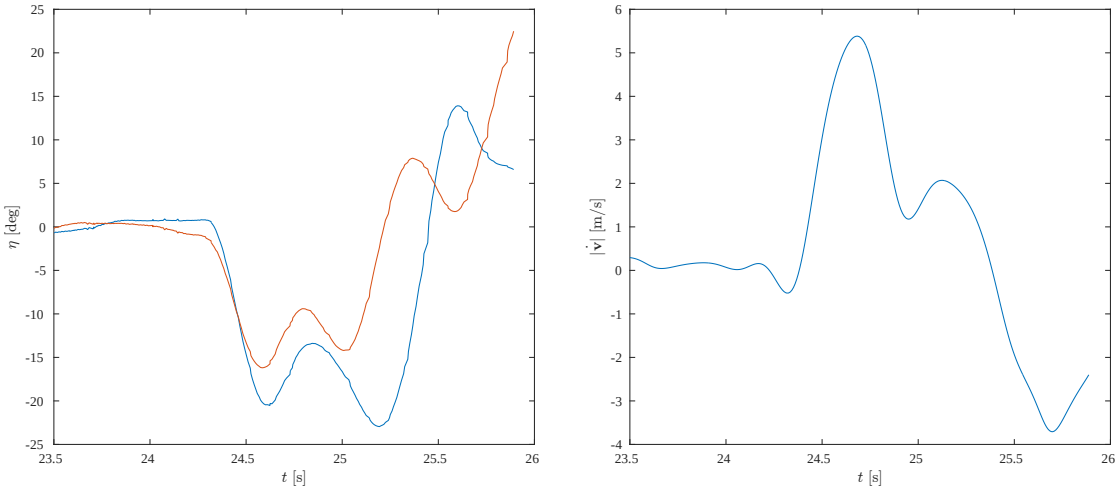
Figure 7-6: Target Region Flight with EKF and Stochastic Optimal Control, translational data from motion capture system. Quadrotor reaches target region.



(a) velocities. blue: v_x^I, red: v_y^I, yellow v_z^I with corresponding velocity-target regions

(b) 3D trajectory

Figure 7-7: Target Region Flight with EKF and Stochastic Optimal Control, translational data from motion capture system.



(a) orientation: red: roll ϕ, blue: pitch θ

(b) acceleration of absolute velocity

Figure 7-8: Target Region Flight with EKF and Stochastic Optimal Control, orientation and acceleration data from motion capture data.

Figure 7-9: Raw Footage of Target Region Approach: Quadrotor flies aggressive angles; visual markers can be seen on the right.

The CPU-load on the Jetson TK1 with all PODs running was about 60% with the visual detection taking up around 22%. This leaves computational reserves for additional onboard processing.

While this proof-of-concept demonstrates the usability and potential of tensor-decomposition-based approaches to compute stochastic globally optimal feedback controllers for high-dimensional real systems drawbacks do exist that require further research. When aiming for a reliable, deployable solution that works for a wide range of initial conditions on the real system extensive tuning of cost parameters, discretization levels and optimizer settings is required.

In the specific problem setup of approaching and passing a target region, the stochastic optimal controller demonstrates its strengths for initial conditions that require complex maneuvers. These initial conditions especially comprise initial conditions close to the state-space bounds. The tensor-decomposition-based approximations of the value function seem to be least accurate near these bounds, too. While one of these complex control maneuvers near the bounds was successfully demonstrated on the simplified dynamics for some initial conditions (cf. 6.4.4), it could be reproduced less reliably on the full simulator and unsuccessfully on the real system. For the problem setup of passing a target region without obstacles in the state-space, tracking controllers combined with fast trajectory-planning as in [45] are assumed to promise at least comparable capabilities. However, with a scene setup that comprises obstacles, the stochastic optimal control approach could display great strength since in this case complex trajectory planning is required with standard controllers, while cost-based optimization-approaches solved over the entire state space, like the stochastic optimal controller used in this thesis, would integrate the obstacles (as high costs) into the control design. Therefore this control design concept seems promising for further research. A discussion on this follows in Section 8.2.1.

Summary

An experimental performance evaluation of the overall integrated systems was presented in this chapter. It showed the platform's hover capabilities and demonstrated that tensor-decomposition-based nonlinear stochastic optimal controllers can control the quadrotor to approach a target region.

The next and concluding chapter summarizes this thesis, discusses limitations and suggests starting points for future work.

Chapter 8

Conclusion

8.1 Summary

In this work a quadrotor platform was built that enables fast, vision-based autonomous maneuvering, bridging the gap between slow but fully autonomous and fast, but dependent platforms. To this end, the platform utilizes a high-performance embedded computing unit and demonstrates a proof-of-concept use of tensor-decomposition-based, stochastic globally optimal feedback controller. During this work it was demonstrated that:

1. the high-performance embedded computing unit with a GPU enables the application of fast, high-resolution computer vision, estimation and control *on board*.

2. the developed platform offers sufficient computing reserves to *add* scene understanding, feature tracking or obstacle detection using standard approaches from e.g. the openCV libraries that can exploit the onboard GPU unit.

3. the modular software design allows to easily switch in and out new estimators, controllers, feature trackers, etc. Due to the use of the LCM-message-handling-framework low data exchange latency is achieved and signals from high-level tasks down to motor-level commands can be handled by the same infrastruc-

ture.

4. tensor-train-decomposition based approaches can be used to synthesize a stochastic globally optimal feedback controller that complies with nonlinear 6-dimensional quadrotor dynamics and actuator constraints, and shows trajectory planning capabilities to drive the system into a desired final state.

5. these recent developments in utilizing compressed computation can be used to synthesize controllers that work in simulation and on real systems. Extensive tuning is required to generate reliable, deployable controllers.

   In this thesis' specific problem setup of approaching and passing a target region, the stochastic optimal controller demonstrates its strength for initial conditions that require complex maneuvers. These initial conditions especially comprise initial conditions close to the state-space bounds. While one of these complex control maneuvers near the bounds was successfully demonstrated on simplified dynamics for some initial conditions, it could be reproduced less reliably on more complex dynamics and unsuccessfully on the real system. For the problem setup of passing a target region without obstacles in the state-space, tracking controllers combined with fast trajectory planning as presented by Mellinger et al. [45] are assumed to promise at least comparable capabilities. However, with a scene setup that comprises obstacles, the stochastic optimal control approach could display great strength since, in this case, complex trajectory planning is required with standard controllers; cost-based optimization-approaches on the other hand, like the stochastic globally optimal controller used in this thesis, would integrate the obstacles (as high costs) into the control design.

8.2 Future Work

This section suggests future work to build on the results of this thesis. While this work presents a working, prototypical system that incorporates visual-inertial estimation

and tensor-decomposition based optimal feedback control techniques there are clear avenues how to advance the current implementation.

8.2.1 Controller libraries, Non-cascaded Controllers and Image-based Visual Servoing

As mentioned in Section 8.1, stochastic globally optimal controllers can fully display their strength for more complex scene settings, including obstacle fields. This could be explored through building value function-libraries for a variety of obstacle scenes. Then, for every scene in that library, an optimal control output exists for every point in a scene's state space. During runtime, a quadrotor would then select the scene from the library that suits the actual scene the quadrotor finds itself in.

On quadrotor model side, more complex models should be integrated. The current implementation synthesizes optimal controllers for a reduced quadrotor model that assumes the quadrotor's orientation as a direct actuator. This concept results from cascading the quadrotor dynamics into an outer position loop and an inner orientation loop. Ideally, the controller synthesis should be based on the full 12-dimensional quadrotor model such that the orientation dynamics are taken into account instead of being assumed as perfectly controlled by the inner-loop orientation controller. Solving the 6-dimensional nonlinear stochastic optimal control problem as done in this work demonstrated the potential of tensor-decomposition based methods. However, while the solution approach theoretically scales well with dimensions, a 12-dimensional model will require additional investigations, especially if the goal is to generate robust controllers that work for a large state-space and arbitrary initial conditions. As an intermediate step, adding two states to the 6-dimensional position model could approximate the actual, 4-dimensional pitch- and roll-dynamics with two first-order delay systems.

While these approaches model the mechanical dynamics from end to end, an additional avenue to synthesize optimal controllers is to incorporate the entire visual-mechanical pipeline into the models. Work on image-based visual servoing does that:

Instead of controlling on fullstate errors, errors in the observed image are controlled. This challenging approach would render estimators redundant.

8.2.2 Visual-inertial Odometry for Robust State Estimation

The visual-inertial estimation presented as proof-of-concept setup in this thesis suffers from the necessity of priorly known visual cues. Once lost, the estimation purely relies on accelerometer measurements which is known for its unreliable drift. This prevented to actual fly *through* a window (which would result in losing track of the known visual cues).

Visual-inertial odometry promises to be a great remedy to this issue. Both openCV as well as NVIDIA's *visionworks*-toolbox offer out-of-the-box feature tracking. These features can be fed into SLAM-algorithms that estimate the features' 3D-positions as well as the camera's pose.

GTSAM [56] offers a factor graph-based optimizer to that problem. [19] extends and evaluate this framework for improved use with IMU measurements and their manifold structure of the rotational components.

Initial tests of this framework with preliminary modifications running on the TK1 showed promising results in terms of accuracy. This removes the need for priorly known visual cues being detected at high rates, which, in turn, allows to integrate target region detectors that run at considerably lower frame rate than the estimation.

It can also be used as a starting point to build sparse 3D maps of the environment. Kleiner et al. follow a semi-direct SLAM-approach that does not solely rely on extracted and tracked features and present a solution for live dense 3D mapping [39].

8.2.3 Target Region Detection in Unknown Environments

As Section 8.2.2 points out, the detection of the target region can and should be enhanced. Given the scenario of exploring indoor-environments quickly without priorly known visual cues, the tasks reduces to finding doors, windows, openings, or generally, open alleys in an unknown environment. Since, when using visual-inertial-

odometry-based state estimation, this detection can run at considerably lower rates than the estimation, many options come to mind: standard classification techniques like HoG-classifiers [17] could be tested to spot distinct openings or approximate, dense 3D-vision could reveal open alleys to fly through.

Appendix A

Tables

Table A.1: Aerodynamic Parameters as in [14], with adapted parameters

Air density ρ_{aero}	1.204
Lock number γ_{aero}	0.839
Blade root angle θ_{b0}	0.52
Blade twist angle θ_{b1}	-0.35

Table A.2: Marker Detection Parameters

Minimum area ratio $\underline{A}$	2.5
Maximum area ratio $\bar{A}$	7
Minimum distance ratio $\underline{\zeta}$	0.02
Maximum distance ratio $\bar{\zeta}$	0.2

Table A.3: PD-controller Gains

$\mathbf{K_{P,r'}}$	$\text{diag}(\begin{bmatrix}0.2 & 0.25 & 5\end{bmatrix})$
$\mathbf{K_{D,r'}}$	$\text{diag}(\begin{bmatrix}0.15 & 0.22 & 3\end{bmatrix})$
$\mathbf{K_{P,\eta}}$	$\text{diag}(\begin{bmatrix}6.3 & 31.5 & 31.5\end{bmatrix})$
$\mathbf{K_{D,\eta}}$	$\text{diag}(\begin{bmatrix}4.5 & 7.2 & 7.2\end{bmatrix})$

Table A.4: Stochastic Optimal Control Parameters

State bounds		
x^I	$-5 < x^I < 0.0$	m
y^I	± 2.5	m
z^I	± 2	m
v_x^I, v_y^I, v_z^I	± 5	m/s
Controller Limits		
Thrust T	$-mg \pm 3$	N
Orientation θ', ϕ'	± 0.4	rad
Costs		
Control Costs $\mathbf{S^u}$	$\text{diag}([0 \quad 0.03 \quad 0.2])$	
Position Costs $\mathbf{S^r}$	$\text{diag}([8 \quad 4 \quad 8])$	
Time Cost ϱ_t	60	
Noise		
Position Noise σ_r	$2E^{-2}\mathbf{I}_3$	
Velocity Noise σ_v	$15E^{-1}\mathbf{I}_3$	
Discount factor		
β	1	

Appendix B

Transformations

B.1 Orientation

From Position-Controller Orientation To Quadrotor Orientation

$$\mathbf{D}_{\psi}^{-1}\mathbf{D}_{\theta'}\mathbf{D}_{\phi'}\begin{bmatrix}0\\0\\1\end{bmatrix} = \mathbf{D}_{\theta}\mathbf{D}_{\phi}\begin{bmatrix}0\\0\\1\end{bmatrix}$$

$$\Leftrightarrow$$

$$\begin{bmatrix}\cos(\phi')\cos(\psi)\sin(\theta') - \sin(\phi')\sin(\psi)\\ -\cos(\psi)\sin(\phi') - \cos(\phi')\sin(\theta')\sin(\psi)\\ \cos(\theta')\cos(\phi')\end{bmatrix} = \begin{bmatrix}b_0\\b_1\\b_2\end{bmatrix} = \mathbf{D}_{\theta}\mathbf{D}_{\phi} = \begin{bmatrix}\cos(\phi)\sin(\theta)\\ -\sin(\phi)\\ \cos(\theta)\cos(\phi)\end{bmatrix}$$

$$\Rightarrow$$

$$\phi = \sin^{-1}(-b_1)$$

$$\theta = \sin^{-1}(b_0/\cos(\phi))$$

Bibliography

[1] M. Achtelik, M. Achtelik, S. Weiss, and R. Siegwart, "Onboard imu and monocular vision based control for mavs in unknown in- and outdoor environments," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 3056–3063.

[2] M. Achtelik, Tianguang Zhang, K. Kuhnlenz, and M. Buss, "Visual tracking and control of a quadcopter using a stereo camera system and inertial sensors," in *2009 International Conference on Mechatronics and Automation*. IEEE, 2009, pp. 2863–2869.

[3] S. Ahrens, D. Levine, G. Andrews, and J. How, "Vision-based guidance and control of a hovering vehicle in unknown, gps-denied environments," in *2009 IEEE International Conference on Robotics and Automation*. IEEE, 2009, pp. 2643–2648.

[4] Ascending Technologies GmbH. Asctec amazing technology. [Online]. Available: http://www.asctec.de/

[5] A. Bachrach and A. Huan. Pods guidelines. [Online]. Available: https://sourceforge.net/p/pods/home/Home/

[6] I. Bar-Itzhack and Y. Oshman, "Attitude determination from vector observations: Quaternion estimation," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 21, no. 1, pp. 128–136, 1985.

[7] A. Barry and R. Tedrake, "Pushbroom stereo for high-speed navigation in cluttered environments."

[8] D. Brescianini, M. Hehn, and R. D'Andrea, "Quadrocopter pole acrobatics," in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2013, pp. 3472–3479.

[9] G. Buskey, J. Roberts, P. Corke, P. Ridley, and G. Wyeth, "Sensing and control for a small-size helicopter," in *Experimental Robotics VIII*, B. Siciliano and P. Dario, Eds. Springer Berlin Heidelberg, 2003, pp. 476–486.

[10] C. Cadena, L. Carlone, H. Carrillo, Y. Latif, D. Scaramuzza, J. Neira, I. D. Reid, and J. J. Leonard, "Simultaneous localization and mapping: Present, future, and the robust-perception age," *ArXiv e-prints*, 2016.

[11] J. Canny, *The complexity of robot motion planning*, ser. ACM doctoral dissertation awards. Cambridge and Mass: MIT Press, ©1988, vol. 1987.

[12] D. E. Catlin, *Estimation, Control, and the Discrete Kalman Filter*, ser. Applied Mathematical Sciences. New York and NY: Springer New York, 1989, vol. 71.

[13] F. Chaumette and S. Hutchinson, "Visual servo control. i. basic approaches," *IEEE Robotics & Automation Magazine*, vol. 13, no. 4, pp. 82–90, 2006.

[14] P. Corke, *Robotics, Vision & Control: Fundamental Algorithms in MATLAB*. Berlin and Heidelberg: Springer Verlag, 2011.

[15] G. Costante, C. Forster, J. A. Delmerico, P. Valigi, and D. Scaramuzza, "Perception-aware path planning," *ArXiv e-prints*, 2016.

[16] M. Cutler and J. P. How, "Analysis and control of a variable-pitch quadrotor for agile flight," *Journal of Dynamic Systems, Measurement, and Control*, vol. 137, no. 10, p. 101002, 2015.

[17] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*. IEEE, 2005, pp. 886–893.

[18] M. Euston, P. Coote, R. Mahony, Jonghyuk Kim, and T. Hamel, "A complementary filter for attitude estimation of a fixed-wing uav," in *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2008, pp. 340–345.

[19] C. Forster, L. Carlone, F. Dellaert, and D. Scaramuzza, "Imu preintegration on manifold for efiňĄcient visual-inertial maximum-a-posteriori estimation."

[20] F. Fraundorfer, L. Heng, D. Honegger, G. H. Lee, L. Meier, P. Tanskanen, and M. Pollefeys, "Vision-based autonomous mapping and exploration using a quadrotor mav," in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*. IEEE, 2012, pp. 4557–4564.

[21] L. R. García Carrillo, A. E. Dzul López, R. Lozano, and C. Pégard, "Modeling the quad-rotor mini-rotorcraft," in *Advances in Industrial Control*, L. R. García Carrillo, A. E. Dzul López, R. Lozano, and C. Pégard, Eds. Springer London, 2013, pp. 23–34.

[22] A. Gorodetsky. (2016) Compressed continuous computation. [Online]. Available: http://www.alexgorodetsky.com/c3/html/

[23] A. Gorodetsky, S. Karaman, and Y. Marzouk, "Tensor-based optimal control," to be submitted.

[24] A. Gorodetsky, S. Karaman, and Y. Marzouk, "Efficient high-dimensional stochastic optimal motion control using tensor-train decomposition," in *Robotics: Science and Systems XI*. Robotics: Science and Systems Foundation, 2015.

[25] V. Grabe, H. H. Bulthoff, and P. R. Giordano, "On-board velocity estimation and closed-loop control of a quadrotor uav based on optical flow," in *2012 IEEE International Conference on Robotics and Automation*. IEEE, 2012, pp. 491–497.

[26] D. Gurdan, J. Stumpf, M. Achtelik, K.-M. Doth, G. Hirzinger, and D. Rus, "Energy-efficient autonomous four-rotor flying robot controlled at 1 khz," in *Proceedings 2007 IEEE International Conference on Robotics and Automation*. IEEE, 2007, pp. 361–366.

[27] T. Hamel and R. Mahony, "Visual servoing of an under-actuated dynamic rigid-body system: an image-based approach," *IEEE Transactions on Robotics and Automation*, vol. 18, no. 2, pp. 187–198, 2002.

[28] T. Hamel, R. Mahony, and A. Chriette, "Visual servo trajectory tracking for a four rotor vtol aerial vehicle," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*. IEEE, 2002, pp. 2781–2786.

[29] P. Heckbert and M. Garland, "Survey of polygonal surface simplification algorithms."

[30] M. B. Horowitz, A. Damle, and J. W. Burdick, "Linear hamilton jacobi bellman equations in high dimensions," in *53rd IEEE Conference on Decision and Control*. IEEE, 2014, pp. 5880–5887.

[31] A. Huang. libbot2. [Online]. Available: https://github.com/libbot2/libbot2

[32] A. Huang, E. Olson, and D. Moore. Lcm. [Online]. Available: https://lcm-proj.github.io/

[33] S. Hutchinson, G. Hager, and P. Corke, "A tutorial on visual servo control," *IEEE Transactions on Robotics and Automation*, vol. 12, no. 5, pp. 651–670, 1996.

[34] J. Engel, J. Sturm, and D. Cremers, "Accurate figure flying with a quadrocopter using onboard visual and inertial sensing," in *Proc. of the Workshop on Visual Control of Mobile Robots (ViCoMoR) at the IEEE/RJS International Conference on Intelligent Robot Systems (IROS)*, 2012.

[35] J. Sturm, E. Bylow, F. Kahl, and D. Cremers, "Dense tracking and mapping with a quadrocopter," in *Unmanned Aerial Vehicle in Geomatics (UAV-g)*, 2013.

[36] S. Karaman and F. Riether. (2015) 16.30 feedback control systems. [Online]. Available: dronecontrol.mit.edu

[37] S. Kim, D. Lee, and H. Kim, "Image based visual servoing for an autonomous quadrotor with adaptive backstepping control," 2011.

[38] D. E. Kirk, *Optimal control theory; An introduction*, ser. Prentice-Hall networks series. Englewood Cliffs and N.J: Prentice-Hall, 1970.

[39] A. Kleiner, F. Heintz, S. Tadokoro, M. Faessler, F. Fontana, C. Forster, E. Mueggler, M. Pizzoli, and D. Scaramuzza, "Autonomous, vision-based flight and live dense 3d mapping with a quadrotor micro aerial vehicle," *Journal of Field Robotics*, vol. 33, no. 4, pp. 431–450, 2016.

[40] H. Lim, J. Park, D. Lee, and H. Kim, "Build your own quadrotor: Open-source projects on unmanned aerial vehicles," *IEEE Robotics & Automation Magazine*, vol. 19, no. 3, pp. 33–45, 2012.

[41] Linux Foundation/Collaborative Projects. (2016) Dronecode/ardupilot. [Online]. Available: https://www.dronecode.org/about

[42] R. Mahony, V. Kumar, and P. Corke, "Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor," *IEEE Robotics & Automation Magazine*, vol. 19, no. 3, pp. 20–32, 2012.

[43] L. Meier, P. Tanskanen, F. Fraundorfer, and M. Pollefeys, "Pixhawk: A system for autonomous flight using onboard computer vision," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 2992–2997.

[44] D. Mellinger, N. Michael, and V. Kumar, "Trajectory generation and control for precise aggressive maneuvers with quadrotors."

[45] D. Mellinger and V. Kumar, "Minimum snap trajectory generation and control for quadrotors," in *2011 IEEE International Conference on Robotics and Automation*. IEEE, 2011, pp. 2520–2525.

[46] I. NaturalPoint. Optitrak motion capture systems. [Online]. Available: http://optitrack.com/

[47] NVIDIA Corp. Jetson tk1. [Online]. Available: http://www.nvidia.com/object/jetson-tk1- embedded-dev-kit.html

[48] PointGreyResearch. Flea3 usb3.0 camera. [Online]. Available: https://www.ptgrey.com/ Content/Images/ uploaded/case-studies/ flea3-usb3-camera.jpg

[49] M. Popp, G. Scholz, S. Prophet, and G. F. Trommer, "A laser and image based navigation and guidance system for autonomous outdoor-indoor transition flights of mavs," in *2015 DGON Inertial Sensors and Systems Symposium (ISS)*. IEEE, 2015, pp. 1–18.

[50] richards tech. [Online]. Available: https://github.com/RTIMULib/RTIMULib-Arduino

[51] D. Scaramuzza. Ocamcalib: Omnidirectional camera calibration toolbox for matlab. [Online]. Available: https://sites.google.com/site/scarabotix/ocamcalib-toolbox

[52] S. Shen, Y. Mulgaonkar, N. Michael, and V. Kumar, "Vision-based state estimation and trajectory control towards high-speed flight with a quadrotor," 2013.

[53] S. Shen, Y. Mulgaonkar, N. Michael, and V. Kumar, "Vision-based state estimation for autonomous rotorcraft mavs in complex environments," in *2013 IEEE International Conference on Robotics and Automation*. IEEE, 2013, pp. 1758–1764.

[54] J.-J. E. Slotine and W. Li, *Applied nonlinear control*. Englewood Cliffs and N.J: Prentice Hall, ©1991.

[55] Sunex. Optics online lenses. [Online]. Available: http://www.optics-online.com/dsl_half.asp

[56] G. T. The Borg Lab. Gtsam. [Online]. Available: https://collab.cc.gatech.edu/borg/gtsam/

[57] C. Troiani, A. Martinelli, C. Laugier, and D. Scaramuzza, "Low computational-complexity algorithms for vision-aided inertial navigation of micro aerial vehicles," *Robotics and Autonomous Systems*, vol. 69, pp. 80–97, 2015.

[58] B. Vedder. Vesc – open source esc. [Online]. Available: http://vedder.se/2015/01/ vesc-open-source-esc/

[59] J. R. Wertz, *Spacecraft Attitude Determination and Control*, ser. Astrophysics and Space Science Library, 0067-0057. Dordrecht: Springer Netherlands, 1978, vol. 73.

[60] Willow Garage and contributors. Opencv camera calibration and 3d reconstruction. [Online]. Available: http://docs.opencv.org/2.4/ modules/calib3d/doc/camera_calibration _and_3d _reconstruction.html #solvepnp

[61] S. Yang, S. A. Scherer, and A. Zell, "An onboard monocular vision system for autonomous takeoff, hovering and landing of a micro aerial vehicle," *Journal of Intelligent & Robotic Systems*, vol. 69, no. 1-4, pp. 499–515, 2013.