On-board Obstacle Avoidance in the Teleoperation of Unmanned Aerial Vehicles

On-board Obstacle Avoidance in the Teleoperation of Unmanned Aerial Vehicles

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

Marcin M. Odelga

aus Warschau (Polen)

Tübingen

2018

Tag der mündlichen Qualifikation:	16.04.2019
Dekan:	Prof. Dr. Wolfgang Rosenstiel
1. Berichterstatter:	Prof. Dr. Heinrich H. Bülthoff, Department of Human Perception, Cognition and Action, MPI for Biological Cybernetics, Tübingen
2. Berichterstatter:	Prof. Dr. Andreas Zell

To all the people that inspired and supported me, teachers, supervisors, fellow researchers and novelists. To my family and foremost to my wife, Aneta Peszko.

Abstract

The teleoperation of unmanned aerial vehicles (UAVs), especially in cramped, GPS-restricted, environments, poses many challenges. The presence of obstacles in an unfamiliar environment requires reliable state estimation and active algorithms to prevent collisions.

In this dissertation, we present a collision-free indoor navigation system for a teleoperated quadrotor UAV. The platform is equipped with an on-board miniature computer and a minimal set of sensors for this task and is self-sufficient with respect to external tracking systems and computation. The platform is capable of highly accurate state-estimation, tracking of the velocity commanded by the user and collision-free navigation. The robot estimates its state in a cascade architecture. The attitude of the platform is calculated with a complementary filter and its linear velocity through a Kalman filter integration of inertial and optical flow measurements.

An RGB-D camera serves the purpose of providing visual feedback to the operator and depth measurements to build a probabilistic, robot-centric obstacle state with a bin-occupancy filter. The algorithm tracks the obstacles when they leave the field of view of the sensor by updating their positions with the estimate of the robot's motion. The avoidance part of our navigation system is based on the Model Predictive Control approach. By predicting the possible future obstacles states, the UAV filters the operator commands by altering them to prevent collisions. Experiments in obstacle-rich indoor and outdoor environments validate the efficiency of the proposed setup.

Flying robots are highly prone to damage in cases of control errors, as these most likely will cause them to fall to the ground. Therefore, the development of algorithm for UAVs entails considerable amount of time and resources. In this dissertation we present two simulation methods, i.e. software- and hardware-in-the-loop simulations, to facilitate this process. The software-in-the-loop testing was used for the development and tuning of the state estimator for our robot using both the simulated sensors and pre-recorded datasets of sensor measurements, e.g., from real robotic experiments. With hardware-in-the-loop simulations, we are able to command the robot simulated in Gazebo, a popular open source ROS-enabled physical simulator, using computational units that are embedded on our quadrotor UAVs. Hence, we can test in simulation not only the correct execution of algorithms, but also the computational feasibility directly on the robot's hardware.

Lastly, we analyze the influence of the robot's motion on the visual feedback pro-

vided to the operator. While some UAVs have the capacity to carry mechanically stabilized camera equipment, weight limits or other problems may make mechanical stabilization impractical. With a fixed camera, the video stream is often unsteady due to the multirotor's movement and can impair the operator's situation awareness. There has been significant research on how to stabilize videos using feature tracking to determine camera movement, which in turn is used to manipulate frames and stabilize the camera stream. However, we believe that this process could be greatly simplified by using data from a UAVs on-board inertial measurement unit to stabilize the camera feed. Our results show that our algorithm successfully stabilizes the camera stream with the added benefit of requiring less computational power.

We also propose a novel quadrotor design concept to decouple its orientation from the lateral motion of the quadrotor. In our design the tilt angles of the propellers with respect to the quadrotor body are being simultaneously controlled with two additional actuators by employing the parallelogram principle. After deriving the dynamic model of this design, we propose a controller for this platform based on feedback linearization. Simulation results confirm our theoretical findings, highlighting the improved motion capabilities of this novel design with respect to standard quadrotors.

Kurzfassung

Teleoperation von Drohnen in Umgebungen ohne GPS-Verbindung und wenig Bewegungsspielraum stellt den Operator vor besondere Herausforderungen. Hindernisse in einer unbekannten Umgebung erfordern eine zuverlässige Zustandsschätzung und Algorithmen zur Vermeidung von Kollisionen.

In dieser Dissertation präsentieren wir ein System zur kollisionsfreien Navigation einer ferngesteuerten Drohne mit vier Propellern (Quadcopter) in abgeschlossenen Räumen. Die Plattform ist mit einem Miniaturcomputer und dem Minimum an Sensoren ausgestattet. Diese Ausstattung genügt den Anforderungen an die Rechenleistung. Dieses Setup ermöglicht des Weiteren eine hochgenaue Zustandsschätzung mit Hilfe einer Kaskaden-Architektur, sehr gutes Folgeverhalten bezüglich der kommandierten Geschwindigkeit, sowie eine kollisionsfreie Navigation. Ein Komplementärfilter berechnet die Höhe der Drohne, während ein Kalman-Filter Beschleunigung durch eine IMU und Messungen eines Optical-Flow Sensors fusioniert und in die Softwarearchitektur integriert.

Eine RGB-D Kamera stellt dem Operator ein visuelles Feedback, sowie Distanzmessungen zur Verfügung, um ein Roboter-zentriertes Modell umliegender Hindernisse mit Hilfe eines Bin-Occupancy-Filters zu erstellen. Der Algorithmus speichert die Position dieser Hindernisse, auch wenn sie das Sehfeld des Sensors verlassen, mit Hilfe des geschätzten Zustandes des Roboters. Das Prinzip des Ausweich-Algorithmus basiert auf dem Ansatz einer modell-prädiktiven Regelung. Durch Vorhersage der wahrscheinlichen Position eines Hindernisses werden die durch den Operator kommandierten Sollwerte gefiltert, um eine mögliche Kollision mit einem Hindernis zu vermeiden. Die Plattform wurde experimentell sowohl in einer räumlich abgeschlossenen Umgebung mit zahlreichen Hindernissen als auch bei Testflügen in offener Umgebung mit natürlichen Hindernissen wie z.B. Bäume getestet.

Fliegende Roboter bergen das Risiko, im Fall eines Fehlers, sei es ein Bedienungs- oder Berechnungsfehler, durch einen Aufprall am Boden oder an Hindernissen Schaden zu nehmen. Aus diesem Grund nimmt die Entwicklung von Algorithmen dieser Roboter ein hohes Maß an Zeit und Ressourcen in Anspruch. In dieser Arbeit präsentieren wir zwei Methoden (Software-in-the-loop- und Hardware-in-the-loop-Simulation) um den Entwicklungsprozess zu vereinfachen. Via Software-in-the-loop-Simulation konnte der Zustandsschätzer mit Hilfe simulierter Sensoren und zuvor aufgenommener Datensätze verbessert werden. Eine Hardware-in-the-loop Simulation ermöglichte uns, den Roboter in Gazebo (ein bekannter frei verfügbarer ROS-Simulator) mit zusätzlicher auf dem Roboter installierter Hardware in Simulation

zu bewegen. Ebenso können wir damit die Echtzeitfähigkeit der Algorithmen direkt auf der Hardware validieren und verifizieren.

Zu guter Letzt analysierten wir den Einfluss der Roboterbewegung auf das visuelle Feedback des Operators. Obwohl einige Drohnen die Möglichkeit einer mechanischen Stabilisierung der Kamera besitzen, können unsere Drohnen aufgrund von Gewichtsbeschränkungen nicht auf diese Unterstützung zurückgreifen. Eine Fixierung der Kamera verursacht, während der Roboter sich bewegt, oft unstetige Bewegungen des Bildes und beeinträchtigt damit negativ die Manövrierbarkeit des Roboters. Viele wissenschaftliche Arbeiten beschäftigen sich mit der Lösung dieses Problems durch Feature-Tracking. Damit kann die Bewegung der Kamera rekonstruiert und das Videosignal stabilisiert werden. Wir zeigen, dass diese Methode stark vereinfacht werden kann, durch die Verwendung der Roboter-internen IMU. Unsere Ergebnisse belegen, dass unser Algorithmus das Kamerabild erfolgreich stabilisieren und der rechnerische Aufwand deutlich reduziert werden kann. Ebenso präsentieren wir ein neues Design eines Quadcopters, um dessen Ausrichtung von der lateralen Bewegung zu entkoppeln. Unser Konzept erlaubt die Neigung der Propellerblätter unabhängig von der Ausrichtung des Roboters mit Hilfe zweier zusätzlicher Aktuatoren. Nachdem wir das dynamische Modell dieses Systems hergeleitet haben, synthetisierten wir einen auf Feedback-Linearisierung basierten Regler. Simulationen bestätigen unsere Überlegungen und heben die Verbesserung der Manövrierfähigkeit dieses neuartigen Designs hervor.

Contents

Chapter 1

Introduction

1.1 Teleoperation in the Context of UAVs

Development of various robotics systems has a common goal - increase of efficiency in execution of tasks from the wide range of human interest. Subsequently, it extends this spectrum by enabling tasks that would not be available without the tools of robotics. It can be informally assumed that the higher the autonomy level of a given robotic system the higher the efficiency of the system. In teleoperation, however, when a robot is directly controlled by the user, the presence of a human operator can be seen as a limiting factor. Nevertheless, the human supervision can provide supreme reliability, not yet achievable for fully autonomous robots.

Operation in unstructured environments is arguably one of the largest impediments to full autonomy, with additional challenges arising from close interaction, noise and ambiguity in sensing. Variability of consecutive cases (case-to-case variability) of a given application leads to a lack of generalization, which in turn translates to a difficulty in categorization and in defining robot behaviors. For example, in applications such as medical robotics, or the decommissioning of a nuclear plant, the great variability between cases makes defining robot behaviors extremely hard.

A trade-off between high reliability in teleoperation and autonomous systems' efficiency can be achieved through the concept of shared autonomy. Such a resolution may be also desired as a compromise in situations when legal limits are imposed on the use of autonomous systems or because of practical reasons. By enabling certain autonomous behaviors the system's performance can be increased while simultaneously reducing the operator's mental workload. To achieve this, however, it is also important to maintain good situational awareness as operator throughout the whole process. The operator should be well aware of any autonomous behavior of the platform and be able to supervise it, such that their decision making and supervision is not limited.

From the point of view of the operator, a robot with appropriate capabilities enables them to remotely study the environment or physically interact with it. Multirotor unmanned aerial vehicles (UAVs) are especially suitable for the former in applications such as visual inspection, thermal imaging, aerial photography and

mapping, search and rescue missions etc. Unlike fixed-wing aircraft, multirotors can hover, consequently enabling control over more degrees of freedom (DOF), and in contrast to common helicopters they have simpler dynamics, and do not require a tail rotor to counteract the torque-induced control issues. Smaller propellers are also safer and cheaper in production.

Manual piloting of a multirotor would require, however, a very high workload if a pilot were to control each motor individually. To facilitate the control task, multirotors are equipped with flight controllers that drive the motors given the user input. To realize the control task, the controller has to be equipped with sensors to estimate the state of the controlled quantities.

Quadcopters and other unmanned vehicles come with added bonuses but also new challenges. In order to be easier to operate, they require reliable controllers and state estimation. Nevertheless, piloting of UAVs can be still challenging due to the limited situation awareness of the pilot who controls the platform remotely, not from on-board the aircraft. Many quadcopters are outfitted with on-board cameras to give the pilot a first person perspective when flying. However, even with improved optics, many precautions still must be taken to ensure a safe flight, such as thorough mission planning, analysis of potential obstacles, and being within the line of sight in order to make the best judgment calls.

Until relatively recently most work on multirotors, with quadrotors being the most prominent members of that family, have utilized external tracking systems to develop new software and hardware solutions for UAVs. With the rapidly growing interest in this field of research, and a booming number of publications on these flying robots over the past decade, scientists started to shift towards more advanced topics and applications. Lately, thanks to progress in miniature computers and new sensors, this has become feasible. Tasks like on-board state estimation, environment sensing and object recognition have become executable on the platforms themselves, enabling significantly improved UAV autonomy.

1.2 Components of Teleoperation

Teleoparation indicates the operation of a robot, or machine, at a distance. From the functional point of view it has two distinctive subsystems, the robot and the operator's desk, coupled with a wired or wireless interface. The interface between the user and the robot has to enable proper communication between these two systems, which includes sending commands to the robot and receiving feedback that informs the user about the execution of the task. Hence, the teleoperated robotic UAV in addition to its standard components, i.e., mechanical structure, actuators and sensors, has to be equipped with devices related to its task and components to provide meaningful data about the robot's state and surroundings to the operator.

As vision is the dominant sense in humans, visual sensors of different types are the major source of information for the operator, starting from simple monocular cameras, through stereoscopic and multiple sensor setups to the use of specialized cameras that measure different spectra, e.g., thermal or infrared sensors. Other sensors can gather additional information about the object in the view, e.g., distance measurement through depth estimation. Additional sensor channels can include, but are not limited to, auditory and tactile channels. Haptic controllers, i.e., devices that enable force and torque feedback, have proved beneficial in teloperation tasks (Franchi *et al.*, 2012; Omari *et al.*, 2014). Through haptic sensing, the operator's awareness about the dynamics of the robot or the presence of obstacles can be greatly improved.

1.2.1 The Platform

From the hardware point of view, a UAV robotic platform has to fulfill various requirements. On one hand, its mechanical structure, i.e., the frame with rotors and other basic components, has to provide enough payload to carry the necessary sensors and computation units for the desired tasks while ensuring certain standards of maneuverability and flight time. On the other hand, as we mostly target indoor applications, size limits must also be considered. Larger propellers provide more lift and enable room for additional equipment but increase the overall footprint of the robot which might make it unsuitable given our assumptions and working environment.

From the software perspective, we need to endow the platform with sufficient computational power while keeping the size and weight limits. Especially for algorithms allowing autonomous behaviours and that require the processing of large amounts of sensor data, a relatively high computational power is necessary. However, because of the limited payload of the platform, this requires a balance between the complexity of the implemented algorithms with a high computational power-to-weight ratio of the on-board computer.

Hardware Overview

As the main part of this thesis is a continuation of the work by Stegagno *et al.* (2014), we begun with the same UAV setup which we later modified and adjusted in terms of hardware and software during the development of the presented algorithms. Our UAV platform is based on a MK-Quadro quadcopter from MikroKopter[1], it consists of a frame with four 10 inch propellers powered by brushless motors with motor controllers, in its default configuration it weights less than 1 kg and can carry up to 0.5 kg of payload.

[1] http://www.mikrokopter.de

(a) (b)

Figure 1.1: Our experimental platforms, (a) initial version and (b) final version.

The platform is equipped with a flight controller, a small microprocessor control board with its own firmware, which serves as the interface with the motor controllers and low-level sensors. Although the on-board computer would be highly capable of performing the aforementioned tasks, the dedicated controller provides higher reliability and robustness. Given the minimal functionalities of the flight controller firmware (with a comprehensive and thorough debugging thanks to the development environment), this embedded system ensures stable and consistent performance. The technical details of our flight controller can be seen in App. A.1.

The main computational unit is a smartphone-grade single-board computer (Odroid-XU3), it communicates wirelessly with a ground station PC, transmitting visual feedback to the operator, i.e., color images from the camera, and receiving joypad and Omega.6 haptic device inputs. Using the joypad buttons, the operator can easily switch between different modes of operation (e.g., turn on the motors, initiate lift-off, switch to the haptic control mode, etc.) and give the desired velocity with the haptic device while receiving force feedback.

The platform has two optical sensors, an RGB-D camera (Asus Xtion Pro Live) that is used as the source of data for obstacle detection and to provide RGB visual feedback to the operator, and an optical flow sensor with a built-in echo sonar (PX4Flow, Honegger *et al.* (2013)) oriented downward and used in the state estimation process.

Additionally, the platform is equiped with IR markers for an external tracking system, which provided *ground truth* pose information for our initial experiments, tuning of the on-board state estimator, and evaluation of the performance of the tracking and obstacle avoidance algorithms independently from state estimation errors.

Two versions of the platform are depicted in Fig. 1.1. Both of the platforms, with additional hardware, weigh approximately 1.3 kg. The additional components are attached rigidly to the frame with 3D printed parts. The platform is powered by a 2600 mAh LiPo battery that provides approximately 10 min of flight.

RGB-D Camera

The on-board RGB-D sensor (Asus Xtion Pro Live) is used as the source of data for obstacle detection providing 640×480 RGB and depth images at around 30 fps. Fig. 1.1 depicts two configurations of the camera that we used throughout this research.

The platform in Fig. 1.1a was used in the initial experiments with single obstacles. The vertical orientation of the camera was motivated by the previous work by Stegagno *et al.* (2014) where the authors performed "pan-scanning", alternating left and right yaw rotations to extend the horizontal field of view. The camera in this setup is also rotated downward at about 20° to increase the number of visual features inside the FOV by framing a bigger portion of the ground, while simultaneously offering a horizontal line of sight to the operator. In the second setup, shown in Fig. 1.1b, the camera is oriented horizontally, in its default orientation. As we decided to forgo the pan-scanning motion in favor of better obstacle tracking, this standard orientation provides a better depth field of view and visual feedback to the operator. It both configurations the camera is mounted approximately 45° to the right with respect to the front propeller, with this new direction also defining the forward direction of motion of the robot in our experiments.

RGB-D cameras are optical sensors that combine functionalities of color images (RGB) and depth sensors in one device. Classically, the distance from an object to the camera in computer vision can be estimated through stereo vision and triangulation. Having at least two images of the same object enables estimation of its position in the camera frame if the relative pose of both cameras with respect to each other is known. This, however, is a computationally expensive process and often results in a sparse depth estimation, especially when dealing with objects of uniform color and low contrast.

Structured light sensors, like the Microsoft Kinect or the Asus Xtion Pro Live that we use, approach this problem in another way. Equipped with infrared projectors, these sensors cast a known pattern on the objects in the field of view and estimate the distance to almost every pixel in the image by analyzing the deformation of this pattern. Although there are new problems related to this approach (susceptibility to sunlight, poor detection of highly reflective surfaces and occluded areas), structured light sensors are highly reliable in indoor applications and provide direct dense depth information.

As stated above, depth cameras with their relatively wide fields of view and dense measurements are perfect sensors for obstacle detection. These sensors provide accurate information about obstacles in space, but in order to express detected points in the robot frame they need proper modeling and calibration.

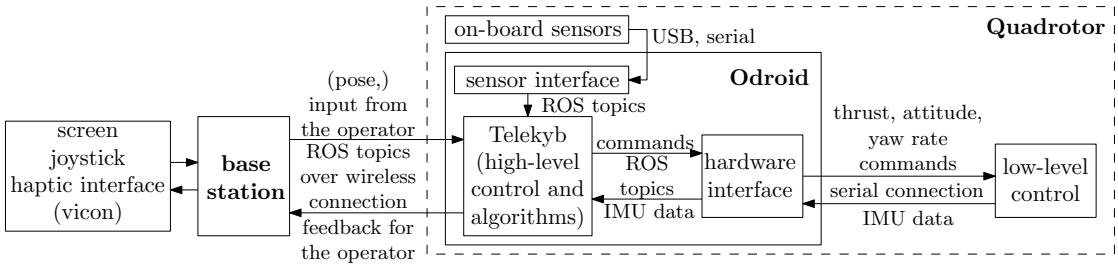


Figure 1.2: A block scheme of our UAV hardware/software setup.

1.2.2 Software Components

The software of a robot includes all the necessary components to drive the platform and perform a desired task. It is comprised of of various modules dedicated to the low- and high-level control, state estimation, sensor software drivers, signal processing, communication with the operator's desk, etc. The block diagram of our software setup is shown in Fig. 1.2, this diagram also indicates the hardware components on which the software is running or with which it is interfacing.

The main computational unit is the Odroid board operated by Ubuntu and running the Robot Operating System with the Telekyb framework. The technical details of this miniature computer are presented in App. A.2. We exploit ROS communication topics to connect the robot to a base station equipped with input/feedback devices using wireless communication. In addition, the base station may host appropriate routines to read measurements provided by a motion capture system (Vicon), if present, and translate them into ROS topics.

Similarly, Telekyb modules are interfaced through ROS topics with a ROS node to connect them with the sensors and the quadrotor hardware. The role of this block is to encrypt and send the commands (desired thrust, attitude and yaw rate) to the microcontroller through the serial connection. The low-level controller is then in charge of driving the propellers' motors to follow the received commands. Similarly, the block receives IMU and battery status data from the microcontroller and translates them to ROS messages.

The UAV can be equipped with additional sensors such as an RGB-D sensor or other cameras, laser scanners, GPS modules etc. In general, these sensors can be connected through serial and USB ports which are present on the Odroid board. The interface with the high-level controller and algorithms is obtained once again with specific ROS nodes that translate measurements into ROS topics.

ROS - the Robot Operating System

The Robot Operating System, although called an operating system, is an advanced framework for robotic software, which requires a compatible operating system to work in. It is a collection of tools, libraries and conventions, whose purpose is to

facilitate the development of complex control software for robotic applications in a broad sense. It has the form of a middleware as it provides layers of abstraction between the computing hardware, sensors, actuators with their hardware, and control algorithms, with protocols to standardize/unify communication and data exchange between different components. As such, it allows users (developers, researchers) to develop software oriented towards these components, which, thanks to the ROS framework, can be independent from the selection of other components.

Each executable software portion constitutes a *node* in the ROS structure/framework that can communicate with other nodes using tools defined in the ROS client library: *messages* over *topics* or *service calls*. In general, topics are meant for fast distribution of frequent data to an undefined number of recipients. Services on the other hand, are two way communication channels. They are usually more specific but mostly allow nodes not only to receive data but also send specific requests.

ROS topics are data channels that are identified with their *names* and have specified message types. Any node can create a topic by *publishing* to it or receive data by *subscribing* to an already existing topic. For example, a user can implement a sensor specific driver as a node that will regularly query the sensor about its measurement, transfer the data into a predefined message structure and publish it so that other nodes can utilize the sensor data. In this example, standardized message types allow users to create more general algorithms that will use a specific type of sensors, which share the same message type, but not limited to specific models.

ROS services also have specified names and data types that consist of two components: a *request* and *response*. In the request part, the node that *calls* the service specifies the conditions and receives a corresponding response. For example, in our obstacle avoidance algorithm that is detailed in Chap. 4, we implemented the obstacle mapping as a separate node with the avoidance part as a service. The UAV trajectory processor node can validate its desired velocity command by sending it to the obstacle avoidance node and receiving a response with a command that minimizes the probability of collision given specific constrains.

Telekyb

Telekyb (Grabe *et al.*, 2013) is a teleoperation framework developed initially as a standalone software and migrated later into the ROS architecture. It is a collection of different software components related to the teleoperation of UAVs, it has definitions of its own data structures with inter-communication. The main component of Telekyb is a ROS node called Telekyb Core. It is modular and contains different classes of objects with internal data structures and and inter-communications, e.g.,

- State Estimator (Controller),
- Trajectory Controller,

- Behavior Controller.

The modularity of this design enables easier module modifications and replacement. For example, the State Estimator implements a state-estimation algorithm that, depending on the available sensor measurements, computes the state of the robot. Thus, it is not limited to one sensor configuration, but as the output is a standard ROS message, different algorithms can be implemented and selected at launch time depending on the current configuration. In a laboratory environment an external tracking system can be used, while for flights in unstructured environments we use an implementation of the algorithm presented in Sec. 2.5.1.

The Trajectory Controller is a module containing a higher-level trajectory tracking algorithm. It produces a unified control output for the robot in a hierarchical fashion where the control vector goes through consecutive sub-modules that can modify it. First, the control signal is computed based on a point-to-point trajectory or, as in our case, from the desired velocity command - the user input. Subsequent routines can filter that control signal based, e.g., on a limited flying zone or additional limits. For example, if the user wants to restrict the robot from flying into a certain region, they can add a module that will check the position of the robot and restrict the commanded velocity without the need of modifying the the whole Trajectory module. In our experiments we use the so-called Standard Trajectory Tracker, which is an implementation of the control algorithm described by Martin and Salaün (2010).

The Behavior Controller controls the transition between different modes or stages of operation. For example, the initial power-on, lift-off, hover, teleoperation etc. It ensures that the commands sent to the robot are appropriate for its current state. For example during the lift-off or hovering in place, the robot should not be commanded to fly sideways.

Other components of Telekyb are ROS nodes that handle the specific experiment and communicate with the UAV. The UAV node implements protocols to interface with the UAV, including reading of the on-board sensors and sending commands to the low-level controller. The experiment node handles the user input devices and translates it into commands sent to the Telekyb Core.

1.2.3 Low-level Control

The low-level controller is the software component that directly interfaces with the robot's actuators to control the robot's motion. It is a PID controller for the roll and pitch angles and the yaw angle rate, and it is implemented as a part of the flight controller firmware. Based on the attitude commands from the trajectory controller in Telekyb, it computes the roll, pitch and yaw torques given the current state of the platform from the internal attitude estimator and gains set by the user in the UAV interface node. The internal attitude estimator is a complementary filter

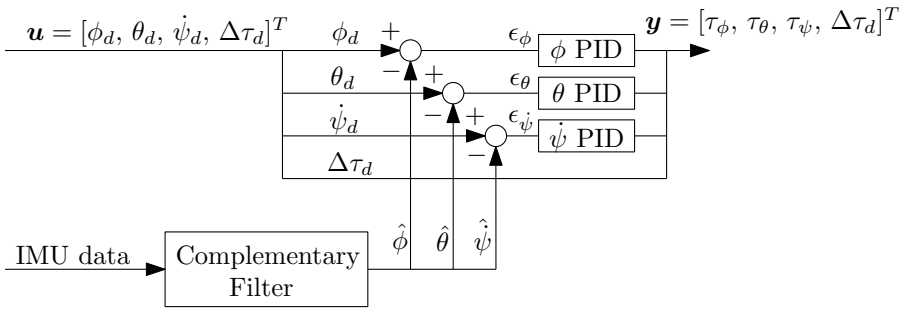


Figure 1.3: Block diagram of the low-level PID controller.

identical to its counterpart in the state estimator of Telekyb presented in Sec. 2.3. The command to each motor is computed based on the obtained roll, pitch and yaw torques and the total desired thrust, fourth component of the attitude command from Telekyb. A simplified diagram of the low-level control scheme is shown in Fig. 1.3.

The commands to the motor controllers are obtained as a weighted distribution of the total thrust

$$\begin{bmatrix} u_{m_1} \\ u_{m_2} \\ u_{m_3} \\ u_{m_4} \end{bmatrix} = \boldsymbol{K}_m \begin{bmatrix} 0 & -1 & -1 & 1/4 \\ -1 & 0 & 1 & 1/4 \\ 0 & 1 & -1 & 1/4 \\ 1 & 0 & 1 & 1/4 \end{bmatrix} \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \\ \Delta\tau_d \end{bmatrix}, \tag{1.1}$$

where u_{m_i} is a generalized i-th motor input, $\boldsymbol{K}_m$ is a matrix with propellers coefficients (Ryll *et al.*, 2015), vector $[\Delta\tau_d,\, \tau_\phi,\, \tau_\theta,\, \tau_\psi]^{\mathrm{T}}$ contains the total thrust $\Delta\tau_d$ and, respectively, the roll, pitch, and yaw torques from the PID controllers, it is multiplied by the torque distribution matrix.

The torque distribution matrix comes from the dynamic model of a quadrotor (Franchi *et al.*, 2012) and its physical meaning can be illustrated with Fig. 1.4. The nominal thrust of each motor is equal to $\frac{1}{4}$ of the total thrust $\Delta\tau_d$. Additional components depend on the desired roll, pitch and yaw torques. Motors m_1 and m_3 lie on the x axis and the difference in their thrust contributes to the pitch rotation, i.e. about the y axis. In the same manner, the motors m_2 and m_4 contribute to the roll rotation. The 4th column of the distribution matrix corresponds to the yaw rotation - torques generated by all motors result in the rotation about the vertical axis, depending on their directions of rotation.

1.2.4 Operator's Desk

The other side of the teleoperation setup is the operator's desk. The main components of the operator's desk are the ground station computer with a screen for

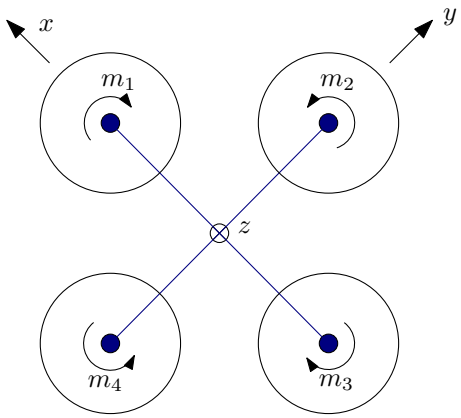


Figure 1.4: Simplified diagram of the quadrotor.

visual information, input device(s), possibly haptic feedback enabled, and speakers if audio information is available. The visual feedback, displayed for the operator on the screen, contains the view from the robot's visual sensors with overlaid additional graphical and textual information, e.g., robot's status. Auxiliary information can be also displayed, for example a 3D view of the robot's surroundings if that is available through robot's sensors. In applications where stereo vision is important, e.g. in teleoperated medical robots, the visual feedback can be presented as stereo pairs of images.

Input Devices

There are a few control strategies in teleoperation. For robots with a limited workspace, e.g., in teleoperation of robotic arms, position control can be used where the workspace of the input device is mapped to the workspace of the robot. For robots with an infinite workspace, for example UAVs, usually velocity control (Stegagno *et al.*, 2014) is used or the desired position is given as way-points on a map (Ahmad *et al.*, 2017). Another control mode is the attitude control of UAVs. In this mode the operator directly controls the roll, pitch, yaw rate and the total thrust. The attitude control requires a skilled operator and involves high workload.

In our experiments we intend to control the robot in the velocity mode and for that we use the following input devices. The main control device is an Omega 6 haptic device shown in Fig. 1.5a. Its three translational degrees of freedom are used to produce a velocity control input as

- forward/backward[2] motion,
- right/left rotation,

[2]backward motion is only conditionally allowed, within the working limits of the obstacle avoidance algorithm, 4.2

(a) (b)

Figure 1.5: Input devices: (a) Omega 6 haptic device and (b) joypad controller. Image sources: (a) `http://www.forcedimension.com` and (b) `https://www.logitechg.com`.

- up/down motion.

We do not allow the user to give lateral translation commands as it would result in motions toward regions outside the visual feedback and could potentially cause collisions. Instead, the user has to rotate the platform first so it faces the desired direction of motion. The haptic device is capable of exerting forces on these degrees of freedom which we use to give a haptic feedback to the operator and increase their situational awareness (Stegagno *et al.*, 2014).

The joypad shown in Fig. 1.5b is used as an auxiliary control device, with its color-coded buttons we can send commands to, e.g., arm the motors, command the robot to lift-off, enter hovering, and then enter into the teleoperation mode. Additionally, in experiments that do not require the precision of control of the haptic device, the joypad knobs and arrow keys can be used to control the robot's motion.

Teleoperation Distance

The distance of remote operation of a robotic system is virtually unlimited. In the field of flying robots close, mid and long range operations are possible, even to the intercontinental distances as presented by Riedel *et al.* (2013). Nevertheless, the important aspect is that the UAV is not in the direct line of sight of the operator and they have to rely on the available feedback.

The limiting factors for long range operations are the interface range, bandwidth and latency. Interfaces can vary from direct cable connection to wireless ones, both using standard bands (WiFi, Bluetooth) or low power dedicated devices (IEEE 802.15.4, such as XBee). Long range communications usually rely on the Internet with LAN/WAN extension on either end.

1.3 Thesis Outline

In this work we tackle the following aspects of vision-based UAV teleoperation. In Chap. 2 we talk about state estimation, a critical task in any robotic application. State estimation is the process of obtaining knowledge about the dynamic state of the robot through measurements and observations. After defining the formal aspects of reference frames and notation related to UAVs, we discuss Kalman filtering (KF), a common sensor fusion algorithm. We analyze non-standard cases of Kalman filtering in terms of available measurements and possible delays in the system. We also show how KF can be used to estimate sensor bias and we define the models of the sensors that we use on our system. Our main contribution in Chap. 2 is a dual state KF design for UAV velocity estimation. Based on the inertial measurement unit (IMU) and optical flow (OF) integration, it estimates the state of the platform and the sensor bias.

The main goal of this thesis is to present a method for obstacle detection, tracking and avoidance for a UAV teleoperated in the velocity mode. In Chap. 3 we discuss different approaches to the obstacle detection and tracking in existing literature, and we analyze them in the context of this thesis. We propose our solution based on the bin-space representation of obstacles in the local frame of the robot using depth camera measurements. As our intention is that our platform is independent from external computation, we pay special attention to the efficiency of our approach in terms of execution on the on-board computer. We specify in detail our approach, which includes the definition of the obstacle state and the process how it is updated with new measurements and the motion of the robot.

In Chap. 4 we define our approach to obstacle avoidance. Our method is inspired by Model Predictive Control. With the obstacle state obtained with the algorithm from Chap. 3 we can predict possible collisions given the user input. The principle of the avoidance algorithm is based on a passive approach, i.e., the algorithm only reacts to the user input, altering it when necessary, however, should not perform actions on its own. The user is informed about any alteration of their input through haptic feedback. This approach guarantees that the user's control over the process is not being limited while facilitating the teleoperation task. In Chap. 4 we also talk about hardware-in-the-loop simulations (HIL), a method that enables testing of an algorithm with simulated sensor measurements on the actual hardware. We show how by using this method, the development time and cost can be greatly improved which is especially important in experiments with high risk of damage to the system. To prove the validity of our navigation system with obstacle avoidance, we show the results of numerous experiments in different scenarios.

In Chap. 5 we analyze the impact of the underactuation of multirotors on teleoperation. As this class of UAVs has to tilt in order to exert lateral accelerations, the visual feedback from a camera rigidly attached to the robot is subject to the same motion. Therefore, it can impair the quality of the visual feedback presented to the

operator and in turn impact their situation awareness. We present three different approaches for visual feedback stabilization, which are

- active stabilization with camera gimbals,
- digital stabilization using IMU data,
- stabilization with a novel, fully actuated platform.

The first approach is a common solution in aerial photography and in this work we analyze how, and if, it can be adopted to our non-standard RGB-D camera. The other two approaches consist of our original work and have been successfully published in the works Odelga *et al.* (2017) and Odelga *et al.* (2016b). Our approach to digital image stabilization is based on using only IMU data and the platform's estimated attitude to stabilize the image. It is a lightweight solution that does not require computationally expensive feature tracking. Next, we present the concept of a novel quadrotor design with two additional actuators for simultaneous manipulation of the propellers' orientation. Thanks to the mechanical links, our platform requires only two servo motors to regain the missing degrees of freedom (DOFs) and have a one-to-one relation between the number of control inputs and DOFs.

Chapter 2

State Estimation

2.1 Introduction

State estimation is a process in which the probable value of certain quantities can be assessed based on available information, both partial and indirect. Thus, by using information from different sources, the calculated estimate is typically more precise and accurate than each information source individually. The source of data are usually sensor measurements, predictions based on the model of the system, and estimates from secondary algorithms. Proper statistical modelling of these data enables calculation not only of the approximate value of the estimated quantity but also of its probabilistic distribution and certainty.

In robotics, state estimation is one of the most important tasks. Variables defining the state of a dynamic system, together with its model, are essential in any control process. The choice of state variables is arbitrary, as long as the corresponding transformation describing the behaviour of the system can be determined. For example, a typical state of an UAV includes its orientation, angular velocity (the rate of change of orientation), 3D velocity and position in the world reference frame. Such a set of quantities comprises a base state vector in the position control of a UAV.

2.1.1 Literature Overview

Depending on the quantities that we want to estimate, there are different approaches to state estimation for UAVs. The state of an UAV can be defined as its pose, i.e., its linear position and angular orientation and the corresponding derivatives: linear and angular velocities and accelerations. However, depending on the control mode, not all of these quantities have to be estimated on-board.

The most fundamental is the estimation of a platform's attitude, the orientation with respect to the gravity vector and the rate of rotation around that direction, i.e., the roll and pitch angles and the yaw rate. Together with an attitude controller such an approach can be implemented on a low-power microcontroller system, which makes it a popular solution among radio control (RC) hobbyists. However, as

the microcontroller knows neither its position nor linear velocity, it is the user's responsibility to assess and control these quantities with the tilt and total thrust of the platform. Therefore, as introduced in Sec. 1.2.4, the attitude control mode requires a skillful and well-trained operator. The standard solution in attitude estimation is the complementary filter (Mahony *et al.*, 2008; Martin and Salaün, 2010).

Enabling on-board velocity control makes UAVs possible to operate even by an untrained person. Because the commanded velocity is directly dependent on the user input, such a control mode is more forgiving, as it is always possible to stop the platform by releasing the input. Velocity estimation, however, requires additional sensors, the most common being an optical flow sensor (Honegger *et al.*, 2013) paired with a range finder, as the estimation of the linear velocity from the optical flow measurements needs the corresponding distance measurements.

To enable the possibility of controlling the robot in the position mode, e.g. through way-points, it is necessary that the robot is able to determine its own position. The most common solution in a laboratory environment is the use of an external tracking system, e.g. Vicon. This approach has been used in multiple works related to UAVs and other robots mainly thanks to its accuracy, low noise, and relatively high frequency of position updates, for example in the work by Stegagno *et al.* (2014) and the work describing our initial results in obstacle avoidance (Odelga *et al.*, 2016b). The main advantage of such setups is obvious, it enables separation of the estimation problem from the investigated problem and focus on higher-level algorithms and applications.

For outdoor scenarios, a possible reference position can be obtained through the Global Positioning System (GPS) and its integration with other sensor measurements, for example in a inertia-GPS sensor fusion as shown by Schinstock (2013). Researchers, however, tend to forgo the use of GPS, mostly due to its limitations and low accuracy in obstacle-rich environments which are typical for small size robots and their applications. Therefore, the main goal of the research on position estimation for UAVs (and other robots) is a fully on-board system, independent from external sources of information, that is robust and accurate in various scenarios.

The main approach to the on-board position estimation is visual odometry (VO) (Nister *et al.*, 2004; Se *et al.*, 2005), where the position of the robot is determined based on the position of the camera computed from the comparison of consecutive images. This estimation can be either based on feature tracking (indirect approaches), as in the notable work by Mur-Artal *et al.* (2015) that uses ORB features, or directly, by analyzing the photometric error between frames (Engel *et al.*, 2013). In summary, the direct approaches estimate the pose of the camera through an optimization problem, by finding a frame-to-frame transformation that align images and minimizes pixel intensity difference directly in the sensor space. Feature-based, or indirect, approaches perform an additional step of feature ex-

traction and matching. The robot's pose can be then determined through bundle adjustment (Lourakis and Argyros, 2005) by finding the transformation that aligns the matched features in consecutive frames.

As in every dead reckoning problem, as the pose of the camera is based on the integration of subsequent transformations, visual odometry approaches are subject to accumulating errors. One of the solutions to this issue is loop closure (Se *et al.*, 2005). The algorithm cross compares the camera's current view with the frames (or features) stored in memory, and consequently minimizes the accumulated error when the robot-camera system reaches a previously known location.

Visual odometry methods are very computationally intensive and although the current state-of-the-art algorithms achieve (sub-)centimeter precision, running them on a lightweight, on-board computer is challenging (Delmerico and Scaramuzza, 2018). Additionally, these methods usually require laborious tuning to optimize their performance to accuracy ratio and often requires a trade-off between robustness and execution time.

Classical visual odometry algorithms, such as in the work presented by Nister *et al.* (2004), were first developed by computer vision researchers. Hence, the focus was purely oriented on determining the camera pose (and subsequently environment mapping) from the images only. As the perspective from a monocular camera does not provide metric correlations between the features, methods that rely solely on a single-lens sensor provide estimates with an unknown scale. To overcome that issue, VO algorithms based on stereo (Engel *et al.*, 2013) and depth cameras (Kerl *et al.*, 2013) have since been developed.

The newest trend is visual-inertial fusion, wherein the motion estimated from a visual sensor is enhanced with the estimate inferred from an inertial sensor. Within this class of visual-inertial odometry (VIO) methods, two classes can be distinguished, so called, loosely and tightly coupled approaches. In loosely coupled algorithms the motion is first determined from the different sensors and then the estimate is refined through sensor fusion algorithms, e.g. Kalman filtering. In tightly coupled algorithms, the inertial data is used within the VO algorithm. A detailed work by Delmerico and Scaramuzza (2018) presents an overview of the most recent VIO algorithms from both of these classes.

The main focus of the work by Delmerico and Scaramuzza (2018), however, is a broad performance comparison of different VIO algorithms run on a few popular on-board computers. It shows that these algorithms can be satisfactorily run on these platforms, but as can be seen from the presented analysis, the algorithms usually consume a large portion of the system's resources, leaving little room for additional functionality.

Among other works, Stegagno *et al.* (2014) shows that the full knowledge of drift-free position is not critical for safe teleoperation, however, the robot must be able to reliably estimate its own velocity. Compared to full-fledged VO or simultaneous localization and mapping (SLAM) algorithms, the velocity estimation,

even when based on feature tracking, is computationally much less intensive and can be relatively easily run on-board allowing space for additional algorithms (Bonnin-Pascual *et al.*, 2015; Oleynikova *et al.*, 2015).

2.1.2 Problem Statement

In the context of this thesis, the goal is to develop a reliable and robust state estimation algorithm for the purpose of indoor teleoperation of UAVs. As the mentioned algorithm has to run on-board a small size quadrotor platform it must be optimized in terms of computational requirements and also in terms of used sensors as the platform is limited both in payload and computational power.

We prioritize the algorithms related to the teleoperation facilitation in this work, namely obstacle detection and avoidance, and the self-sufficiency of the platform with respect to external sensors and computation. Therefore, the state-estimation part of our work must not only ensure on-board feasibility but also allow execution of other, possibly computationally intensive, algorithms.

2.1.3 Methodology

The assumptions mentioned in the previous section convince us to refrain from a full VO or SLAM and seek an alternative solution instead. We developed a velocity estimation algorithm that is based on the classical optical flow-inertial integration (Bonnin-Pascual *et al.*, 2015; Oleynikova *et al.*, 2015), but has been tailored specifically for our platform. Some features are distinctive to our implementation, which we hope is presented with a strong attention to detail, and constitute a clear contribution.

In Sec. 2.2 we present the formal description of our system with associated frames, transformations, and notation, and introduce models of the sensors that we use on our platform. In Sec. 2.3 we introduce equations and describe the complementary filter for orientation estimation. In Sec. 2.4 we discuss Kalman filtering and its variations in different scenarios with examples of our own implementations. In Sec. 2.5 we detail the implementation of our on-board state estimator with example results. In Sec. 2.6 we briefly discuss software-in-the-loop simulations and how we benefited from it in the development of our algorithms. We conclude this chapter in Sec. 2.7.

2.2 Foundations

2.2.1 Frames and Notation

Throughout this thesis we represent relevant quantities with respect to their reference frames. A cartesian coordinate frame A can be defined as $A : \{O_A, X_A, Y_A, Z_A\}$, where O_A defines the origin of A, and X_A, Y_A and Z_A are its three axes as per the right-hand rule.

To express the relation between the two frames, we use the following notation: ${}^A\boldsymbol{p}_B \in \mathbb{R}^3$ to represent the origin of the frame $B : \{O_B, X_B, Y_B, Z_B\}$, in frame A and $\boldsymbol{R}_B^A \in SO(3)$ represent the rotation matrix expressing the orientation of B in A. For an arbitrary point P we can define the following relation for its position in A given its position in B

$$ {}^A\boldsymbol{p}_P = {}^A\boldsymbol{p}_B + \boldsymbol{R}_B^{AB}\boldsymbol{p}_P. \tag{2.1} $$

With ${}^B\phi_A$, ${}^B\theta_A$, and ${}^B\psi_A$ we denote the roll, pitch and yaw angles that represent the rotation of frame B in A according to roll-pitch-yaw (RPY) notation. Thus, the $\boldsymbol{R}_B^A$ matrix is defined as

$$ \boldsymbol{R}_B^A = \boldsymbol{R}_z({}^A\psi_B)\boldsymbol{R}_y({}^A\theta_B)\boldsymbol{R}_x({}^A\phi_B), \tag{2.2} $$

where $\boldsymbol{R}_x(\cdot)$, $\boldsymbol{R}_y(\cdot)$, $\boldsymbol{R}_z(\cdot)$ are the basic rotation matrices and represent the elemental rotations around the X, Y and Z axes respectively. Thus

$$ \begin{aligned} \boldsymbol{R}_B^A &= \begin{bmatrix} c_\psi & -s_\psi & 0 \\ s_\psi & c_\psi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c_\theta & 0 & -s_\theta \\ 0 & 1 & 0 \\ s_\theta & 0 & c_\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & c_\phi & -s_\phi \\ 0 & s_\phi & c_\phi \end{bmatrix} = \\ &= \begin{bmatrix} c_\psi c_\theta & c_\psi s_\theta s_\phi - s_\psi c_\phi & c_\psi s_\theta c_\phi + s_\psi s_\phi \\ s_\psi c_\theta & s_\psi s_\theta s_\phi + c_\psi c_\phi & s_\psi s_\theta c_\phi - c_\psi s_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{bmatrix}, \end{aligned} \tag{2.3} $$

where the notation s_δ, c_δ and t_δ indicates the sin, cos and tan of a generic angle δ.

It is often convenient to determine the RPY angles from a given rotation matrix. If we express an arbitrary rotation matrix as

$$ \boldsymbol{R} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix}, \tag{2.4} $$

from (2.3) we can obtain the following relations

$$\phi = tan^{-1}\left(\frac{r_{32}}{r_{33}}\right), \tag{2.5}$$

$$\theta = tan^{-1}\left(\frac{-r_{31}}{\sqrt{r_{32}^2 + r_{33}^2}}\right), \tag{2.6}$$

$$\psi = tan^{-1}\left(\frac{r_{21}}{r_{11}}\right). \tag{2.7}$$

As the order of rotations matters, it is important to emphasize that the RPY convention assumes extrinsic rotations, i.e., all rotations are about the fixed frame A in the above example. Thus, in order to obtain B, we first perform the rotation of angle ${}^{A}\phi_B$ about X_A, then of angle ${}^{A}\theta_B$ about Y_A, and of angle ${}^{A}\psi_B$ about Z_A. On the other hand, the classical Euler convention defines the rotations as intrinsic, e.g., the Euler ZYX rotation means rotation of ψ, θ, and ϕ about Z - $Y^{'}$ - $X^{''}$ axes, where $Y^{'}$ and $X^{''}$ are the axes of intermediate frames, after the first and the second rotation, respectively. However, the classical Euler ZYX with the same angle values as RPY rotation will produce an identical rotation matrix.

2.2.2 Quadrotor-Camera System

A UAV for teleoperation purposes must be equipped with a camera to provide visual feedback to the operator. A simplified schematic diagram of a UAV-camera system is shown in Fig. 2.1 with reference frames used to represent the relevant quantities related to the state-estimation and control of our platform. In our case, the sensor is rigidly attached to the robot's frame but all of the considerations presented throughout this thesis can be generalized to an actuated camera.

The main reference frame, the quadcopter frame $Q : \{O_Q, X_Q, Y_Q, Z_Q\}$, represented in the common aerospace North-East-Down (NED) notation is used to express the robot's position and orientation in an arbitrary, North-West-Up (NWU), global (world) reference frame $W : \{O_W, X_W, Y_W, Z_W\}$. Q is attached to the middle point of the robot, ideally its center of mass, the X_Q axis defines the front direction and the Z_Q axis points downward.

The robot's position and orientation in the world frame are defined as ${}^{W}\mathbf{p}_Q \in \mathbb{R}^3$ and $\mathbf{R}_Q^W \in SO(3)$, respectively. Following the notation introduced in Sec. 2.2.1, with ${}^{W}\eta_Q = [{}^{W}\phi_Q, {}^{W}\theta_Q, {}^{W}\psi_Q]^{\mathrm{T}}$ we represent the rotation of the robot as RPY angles. Hence, the $\mathbf{R}_Q^W$ matrix can be obtained as

$$\mathbf{R}_Q^W = \mathbf{R}_x(\pi)\mathbf{R}_z({}^{W}\psi_Q)\mathbf{R}_y({}^{W}\theta_Q)\mathbf{R}_x({}^{W}\phi_Q), \tag{2.8}$$

where the $\mathbf{R}_x(\pi)$ matrix describes the transformation from NED to NWU notation.

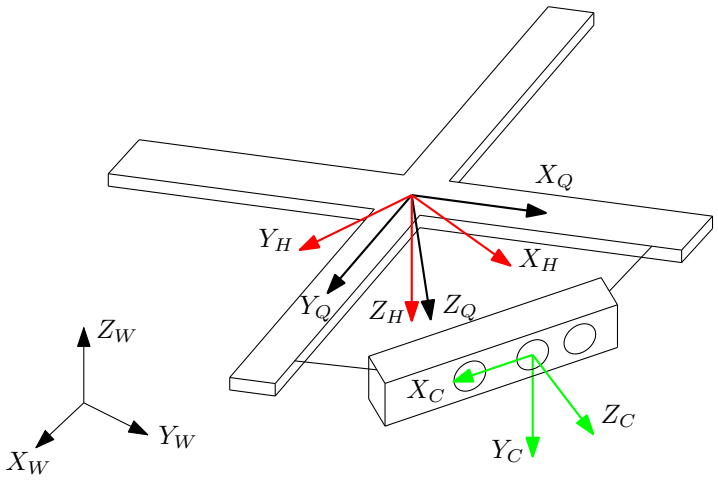


Figure 2.1: The Quadrotor-camera system with relevant coordinate frames.

Stegagno *et al.* (2014) demonstrated that for UAV teleoperation tasks, the robot-centric approach is convenient, i.e. where both, the state of the system and the commands are expressed in a local, horizontal frame. We introduce the horizontal frame $H : \{O_H, X_H, Y_H, Z_H\}$ in which the $X_H Y_H$ plane is parallel to the world $X_W Y_W$ plane and is defined such that $O_H \equiv O_Q$ and its orientation differs from $\mathbf{R}_Q^W$ only by the yaw angle and the NED-NWU conversion. Hence

$$\begin{bmatrix} {}^H\phi_Q & {}^H\theta_Q & {}^H\dot{\psi}_Q \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} {}^W\phi_Q & {}^W\theta_Q & {}^W\dot{\psi}_Q \end{bmatrix}^{\mathrm{T}}, \tag{2.9}$$

and

$$\boldsymbol{R}_Q^H = \boldsymbol{R}_y({}^W\theta_Q)\boldsymbol{R}_x({}^W\phi_Q). \tag{2.10}$$

The position of the quadrotor and its yaw angle expressed in such a frame, ${}^H\mathbf{p}_Q$ and ${}^H\psi_Q$, respectively, are equal to zero.

Lastly, we represent the camera frame, the reference frame for the sensor images, as $C : \{O_C, X_C, Y_C, Z_C\}$. The position and orientation of C in Q, i.e. ${}^Q\mathbf{p}_C$ and $\mathbf{R}_C^Q$, are extrinsic parameters of the camera that depend on the way the camera is attached to the robot. These quantities can be determined or refined through an off-line calibration as shown in Sec. 2.2.4.

In particular, on our platform the camera is mounted with a yaw angle of 45° in order to avoid occlusions by the propellers.

2.2.3 Measurement and Estimation

Measurement describes both the process in which a sensor is used to obtain information about certain quantities and the obtained information itself. As such, measurements are inherently encumbered with errors, usually referred to as ob-

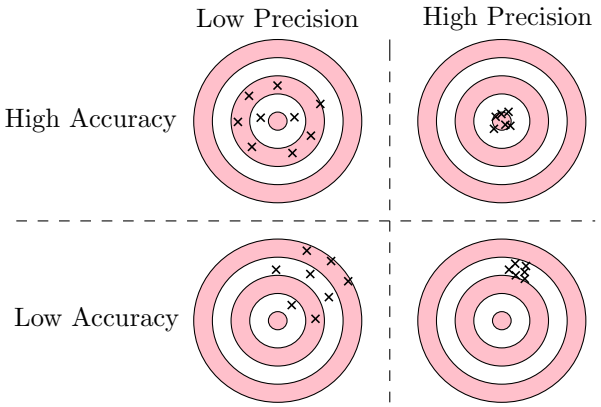


Figure 2.2: Precision and accuracy.

servation or measurement errors. These errors can be classified as *systematic* and *random* errors.

Systematic errors are related to the measurement or the observation process directly, and are strictly related to the way a sensor produces its output . This type of errors is predictable, and can usually be modeled and mitigated in a calibration process. Typical systematic errors are scaling and offset biases. As shown in Fig. 2.2, high proportional errors reduce a sensor's accuracy while measurement offsets contribute to sensor's precision. With the term accuracy we refer to the correctness of the mean value of a quantity, and with the term precision its covariance.

Random errors on the other hand, are random fluctuations of a sensor's readings. They reveal the variation of the output in repeated observations of the same quantity in the same conditions.

Estimation is the process in which uncertain information is used to produce a best approximation of a value. For example, proper statistical modeling of a sensor with its errors allows the reduction of noise and an increase in the overall certainty of the measured value. Methods like Kalman filtering provide a means to fuse information from sources of different precision and accuracy and obtain an estimate better than any single source could provide alone.

2.2.4 Sensor Models

IMU Model

The on-board inertial measurement unit (IMU) consists of an accelerometer and a gyroscope. It provides measurements of the linear acceleration and the angular velocity in the body frame Q. In principle, the IMU can also include a magnetometer to collect absolute heading measurements. However, as shown by Suksakulchai *et al.* (2000), in indoor settings, due to disturbances caused by power lines, ferromagnetic structures, and robotic motors, the measurements provided by digital

compasses are often unreliable. Therefore, in most indoor mobile robotics research, e.g., by Stegagno *et al.* (2016), the heading is considered as an unknown quantity that requires estimation using localization filters.

Accelerometers measure *proper* acceleration, i.e., with respect to their own instantaneous rest frame. A sensor placed in a rest will measure upward acceleration equal to Earth's gravity in the absolute value. Considering sensor imperfections and measurement errors, we model the measurements from the accelerometer as

$$\widetilde{\boldsymbol{a}} = \boldsymbol{\alpha}_a(\boldsymbol{a} + \boldsymbol{\nu}_a) + \boldsymbol{\delta}_a, \tag{2.11}$$

and the measurements from the gyroscope as

$$\widetilde{\boldsymbol{\omega}} = \boldsymbol{\alpha}_\omega(\boldsymbol{\omega} + \boldsymbol{\nu}_\omega) + \boldsymbol{\delta}_\omega, \tag{2.12}$$

where, for $\boldsymbol{i} = \{\boldsymbol{a}, \boldsymbol{\omega}\}$, $\widetilde{\boldsymbol{i}}$ denotes the measured value, $\boldsymbol{i}$ the real value of acceleration and angular rate respectively, $\boldsymbol{\alpha}_i$ and $\boldsymbol{\delta}_i$ represent the scaling and offset biases, and $\boldsymbol{\nu}_i$ is a zero-mean random error.

Although the zero-mean error $\boldsymbol{\nu}_i$ can be filtered out in the estimation process, the offset and scaling biases must be estimated in order to decrease their negative effect on the state estimate. There are various methods to calibrate inertial sensors, in further examples in this chapter we detail our off-line and on-line calibration procedures in different scenarios.

We introduce a set of bias-corrected measurements, $\bar{\boldsymbol{a}}$ and $\bar{\boldsymbol{\omega}}$, for the acceleration and angular velocity, respectively. Hence, these new quantities, assuming an accurate bias estimation, should be affected only by the random, zero-mean noise:

$$\bar{\boldsymbol{a}} = \frac{\widetilde{\boldsymbol{a}} - \boldsymbol{\delta}_a}{\boldsymbol{\alpha}_a} = \boldsymbol{a} + \boldsymbol{\nu}_a, \tag{2.13a}$$

$$\bar{\boldsymbol{\omega}} = \frac{\widetilde{\boldsymbol{\omega}} - \boldsymbol{\delta}_\omega}{\boldsymbol{\alpha}_\omega} = \boldsymbol{\omega} + \boldsymbol{\nu}_\omega. \tag{2.13b}$$

In addition the measured acceleration consists not only of the quadrotor's self-acceleration $\boldsymbol{a}_q$ but also the gravity term $\boldsymbol{g}$. Thus, the following relation completes the IMU model:

$$\boldsymbol{a} = \boldsymbol{a}_q - \boldsymbol{g} = \boldsymbol{a}_q - \boldsymbol{R}_W^Q \boldsymbol{g}_W, \tag{2.14}$$

where $\boldsymbol{g}_W = [0,\ 0,\ g]^{\mathrm{T}}$.

Camera Model

The camera model provides a relation that links pixel coordinates in the image plane with their position in the world frame. This process requires two sets of parameters, intrinsic parameters that allow transformation from the 2D image plane to 3D coordinates in the camera frame, and extrinsic parameters that describe the

relative pose of the camera in the world frame.

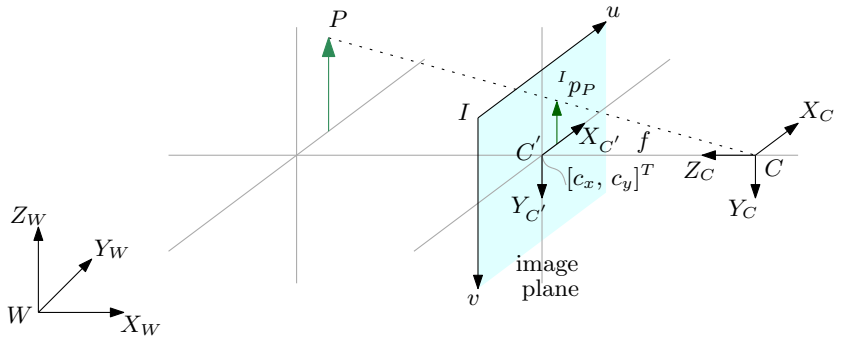


Figure 2.3: Pinhole camera model, projection of an arbitrary point P on the image plane.

The projection of a point in the camera frame onto the image plane can be described with the pinhole camera model (Weng *et al.*, 1992). Let us take an arbitrary point P with its coordinates in the world reference frame as ${}^{W}\boldsymbol{p}_P = [x_W, y_W, z_W]^{\mathrm{T}}$. The same point can be expressed in the coordinates of the camera as ${}^{C}\boldsymbol{p}_P = [x_C, y_C, z_C]^{\mathrm{T}}$, and in the camera frame projected onto the image plane as ${}^{C'}\boldsymbol{p}_P = [x_{C'}, y_{C'}]^{\mathrm{T}}$. If the point P is in the field of view of the camera, assuming no distortions, it will appear in the image with its pixel coordinates as ${}^{I}\boldsymbol{p}_P = [u, v]^{\mathrm{T}}$. The principle of the camera model is shown in Fig. 2.3 and given the camera focal length $\boldsymbol{f} = [f_x, f_y]^{\mathrm{T}}$ we can formulate the following geometric relation

$$\begin{cases} x_{C'} = f_x \dfrac{x_C}{z_C} \\ y_{C'} = f_y \dfrac{y_C}{z_C} \end{cases}, \tag{2.15}$$

and, with respect to the pixel coordinates

$$\begin{cases} x_{C'} = u + c_x \\ y_{C'} = v + c_y \end{cases}, \tag{2.16}$$

where $[c_x, c_y]^{\mathrm{T}}$ is the principal point, i.e. the camera's optical center projected onto the image plane. Combining Eq. (2.15) and (2.16) we define the relation between pixel coordinates and objects in the camera frame as

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_C/z_C \\ y_C/z_C \\ 1 \end{bmatrix} = \boldsymbol{K} \frac{{}^{C}\boldsymbol{p}_P}{z_C}, \tag{2.17}$$

where $\boldsymbol{K}$ is the camera projection matrix.

The model in Eq. (2.17) can be extended with the point coordinates in the world frame given the camera's extrinsic parameters, i.e., its position and orientation in the world frame

$$^{C}\boldsymbol{p}_P = \boldsymbol{R}_W^{C\,W}\boldsymbol{p}_P + {}^{C}\boldsymbol{p}_W. \tag{2.18}$$

Using homogeneous coordinates

$$\begin{bmatrix} {}^{C'}\boldsymbol{p}_P \\ 1 \end{bmatrix} = \left[\begin{array}{c|c} \boldsymbol{R}_W^C & {}^{C}\boldsymbol{p}_W \end{array}\right] \begin{bmatrix} {}^{W}\boldsymbol{p}_P \\ 1 \end{bmatrix} = \boldsymbol{T}_W^C \begin{bmatrix} {}^{W}\boldsymbol{p}_P \\ 1 \end{bmatrix}, \tag{2.19}$$

Hence, the following relation completes the camera model

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \boldsymbol{K}\boldsymbol{T}_W^C \begin{bmatrix} {}^{W}\boldsymbol{p}_P \\ 1 \end{bmatrix}, \tag{2.20}$$

which describes the projection of an arbitrary point in the world frame ${}^{W}\boldsymbol{p}_P$ onto the image plane, where s is an unknown scale factor.

Image Distortions

The model in Eq. (2.20) is a distortion-free camera model, i.e., it describes a perfect camera without potential image distortions. Although these distortions can be irregular, due to the circular symmetry of a camera's lens, they are usually radially symmetric and as such can be modeled and compensated for.

We assume the distortion model

$$\begin{cases} u' = u + \delta_u(u, v) \\ v' = v + \delta_v(u, v) \end{cases}, \tag{2.21}$$

where u' and v' are distorted coordinates of ideal u and v coordinates, $\delta_u(u, v)$ and $\delta_v(u, v)$ are distortion functions. Based on the geometric relation between ideal and actual point positions in the image plane, image distortions can be classified as radial and tangential. These distortions result in the shift of the ideal pixel position $[u,\, v]^{\mathrm{T}}$ to the distorted position $[u',\, v']^{\mathrm{T}}$ as shown in Fig. 2.4a.

Radial distortion results in an offset of the point's position along the radial direction, i.e., away or towards the center of an image. A negative radial distortion is typical for wide angle lenses. It is a barrel distortion and causes the points located farther from the image center to move inward and decrease of the scale. In pincushion distortion, points spread away from the image center, increasingly with the radial distance. It is a positive radial displacement and increases the scale. An example of these distortions is shown in Fig. 2.4b, the object, originally a square, is viewed in the image as either positively or negatively distorted.

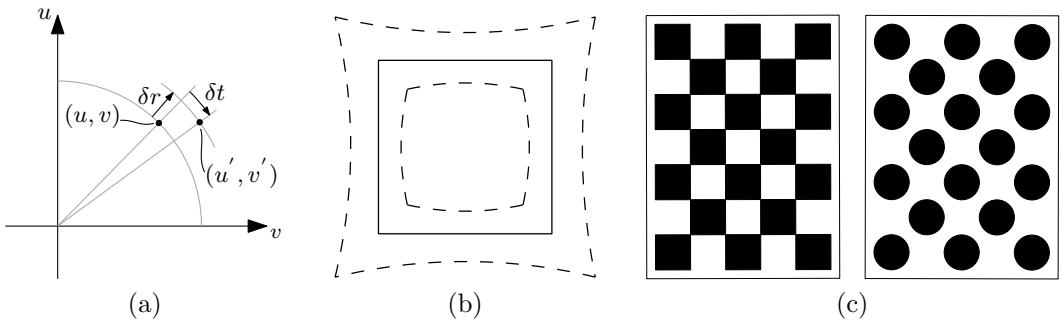


Figure 2.4: Camera calibration: (a) radial and tangential distortions, (b) pincushion and barrel distortions of a square, and (c) typical calibration patterns.

As the displacement in a radial distortion depends on the distance from the image center it is best modeled in polar coordinates (ρ, ϕ) with a polynomial (Weng *et al.*, 1992)

$$\delta_{\rho r} = k_1\rho^3 + k_2\rho^5 + k_3\rho^7 + \dots, \tag{2.22}$$

where ρ is the radial distance from the image center, $k_1, k_2, k_3, \dots$ are the radial distortion coefficients, and

$$\begin{cases} u = \rho cos(\phi) \\ v = \rho sin(\phi) \end{cases} \tag{2.23}$$

defines the polar to Cartesian system transformation.

Tangential distortion causes translation of the ideal point position along the direction perpendicular to the line between the point and image center, or, in other words along the angular coordinate ϕ in the polar system. The reason of this distortion is usually a misalignment of the lens or lens assembly.

Camera Calibration

As image distortion is a common and well-known problem in computer vision there are numerous tools available for camera calibration (Bouguet, 2015). The principle of these methods is similar, the user records a number of images of a known pattern, e.g. as shown in Fig. 2.4c, and using a calibration algorithm based on optimization finds the camera matrix and distortion coefficients that match the observed projection and deformation of the pattern in the images.

Optical Flow sensor model

Optical flow (OF) is the apparent motion of objects in the field of view of a camera caused by the relative motion between the sensor and the environment. Because of the perspective projection of objects in the field of view onto the image plane,

optical flow has the following properties a) it depends on the distance to the object, and b) translation along the image u axis is indistinguishable from the rotation of the sensor around its v axis, and vice versa.

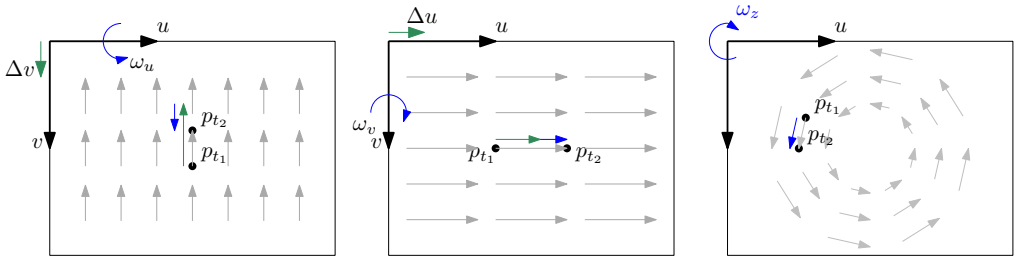


Figure 2.5: Effect of translations and rotations of the camera frame on the optical flow in the image plane.

An example of the effect of positive translations and rotations of the camera on the optical flow is shown in Fig. 2.5. In this section we present a simplified optical flow sensor model under the following assumptions

a) the sensor provides optical flow averaged to the image center, and

b) rotations about the axis perpendicular to the image plane are always around the image center,

which allow us to neglect the effect of the rotation around the camera's optical axis. Thus, we can model the optical flow sensor as

$$\begin{aligned} v_x &= \left(\frac{\Delta u}{\Delta t} - \omega_y d\right)\frac{f}{d}, \\ v_y &= \left(\frac{\Delta v}{\Delta t} + \omega_x d\right)\frac{f}{d}, \end{aligned} \tag{2.24}$$

where v_x and v_y are optical flow based velocities along the image axes, Δu and Δv are measured flows, i.e. the difference in the object's position in the image between consecutive frames expressed in the number of pixel. Δt is the time interval between the frames, ω_x and ω_y are angular rates around the camera axes, f is the sensor focal length, and d is the distance to the object that produced the optical flow.

The distance d has to be estimated in order to have the proper reference metric for the optical flow based velocities. The distance is usually measured with an ultrasonic or infrared/laser distance sensor but can be also estimated from a stereo or depth camera. As shown later in this chapter, for better estimates of OF based velocities, the precision of measurements from a distance sensor can be improved by filtering this variable, e.g., by including it in the state of a Kalman filter.

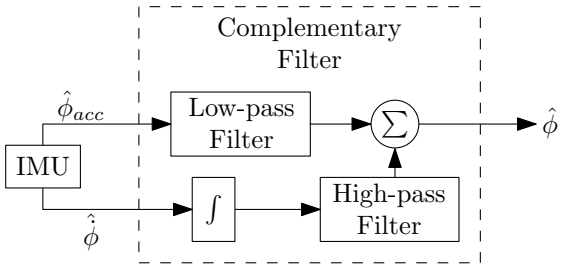


Figure 2.6: Block diagram of the complementary filter.

2.3 Complementary Filter

A complementary filter is an estimator that can be used to estimate the platform's orientation, i.e., the roll and pitch angles (ϕ, θ), using IMU measurements (Mahony *et al.*, 2008; Martin and Salaün, 2010). In this algorithm the angles are first integrated using a proper projection of the gyro readings and corrected using the orientation of the gravity vector, estimated from the filtered accelerometer readings. The block diagram depicting this algorithm is shown in Fig. 2.6.

First, at every instant k, the angular velocity $\boldsymbol{\omega}_k$ in the sensor (body) frame, read from the gyroscope, must be properly transformed to obtain the roll, pitch and yaw rates $\dot{\boldsymbol{\eta}}_k = [\dot{\phi}_k, \dot{\theta}_k, \dot{\psi}_k]^{\mathrm{T}}$,

$$\dot{\boldsymbol{\eta}}_k = \boldsymbol{D}_k(\boldsymbol{\eta}_k)\boldsymbol{\omega}_k, \tag{2.25}$$

where

$$\boldsymbol{D}_k(\boldsymbol{\eta}_k) = \begin{bmatrix} 1 & sin(\phi_k)tan(\theta_k) & cos(\phi_k)tan(\theta_k) \\ 0 & cos(\phi_k) & -sin(\phi_k) \\ 0 & sin(\phi_k)/cos(\theta_k) & cos(\phi_k)/cos(\theta_k) \end{bmatrix}, \tag{2.26}$$

defines the transformation between the angular velocity read by the sensor and the RPY rates. The values obtained in Eq. (2.25) can be integrated to obtain the platform's attitude

$$\begin{aligned} \hat{\phi}_{gyro} &= \hat{\phi}_{k-1} + \dot{\phi}_k \Delta t, \\ \hat{\theta}_{gyro} &= \hat{\theta}_{k-1} + \dot{\theta}_k \Delta t. \end{aligned} \tag{2.27}$$

The complementary values can be calculated from the accelerometer measurement

$$\boldsymbol{a} = \boldsymbol{a}_q - \boldsymbol{R}_W^Q \boldsymbol{g}_W = \boldsymbol{a}_q - \begin{bmatrix} sin(\theta_k) \\ -cos(\theta_k)sin(\phi_k) \\ -cos(\theta_k)cos(\phi_k) \end{bmatrix} \boldsymbol{g}_W, \tag{2.28}$$

where $\boldsymbol{a} = [a_x, a_y, a_z]^{\mathrm{T}}$ is the measured acceleration, $\boldsymbol{a}_q$ is the acceleration of the platform and $\boldsymbol{g}_W = [0, 0, g]^{\mathrm{T}}$ is the gravity vector in the world frame. Assuming that the acceleration of the platform $\boldsymbol{a}_q$ is negligible with respect to the gravitational

acceleration $\boldsymbol{g}_W$, the attitude can be estimated as

$$\begin{aligned} \hat{\phi}_{acc} &= \text{atan}_2(-a_y, -a_z), \text{ and} \\ \hat{\theta}_{acc} &= \text{atan}_2\left(a_x, \sqrt{a_y^2 + a_z^2}\right). \end{aligned} \tag{2.29}$$

Combining Eq. (2.27) and (2.28), the complementary filter equations with gain α are then

$$\begin{aligned} \hat{\phi}_k &= (1-\alpha)(\hat{\phi}_{k-1} + \dot{\phi}_k \Delta t) + \alpha \hat{\phi}_{acc}, \text{ and} \\ \hat{\theta}_k &= (1-\alpha)(\hat{\theta}_{k-1} + \dot{\theta}_k \Delta t) + \alpha \hat{\theta}_{acc}. \end{aligned} \tag{2.30}$$

This algorithm has the form of a low-pass filter and the gain α must be properly tuned such that the gyro drift is compensated for without introducing additional acceleration components coming from the platform's own acceleration $\boldsymbol{a}_q$.

2.4 Kalman Filter

Kalman filtering (KF) is an iterative estimation algorithm that uses different sources of information in order to obtain an approximate estimate of a system's state in the form of state variables. This estimate is represented as a joint probability distribution together with its covariance, i.e., a measure of uncertainty, and through the use of multiple data it is more accurate than each information sources individually.

Using the system's dynamic model, series of measurements, statistical noise and other uncertainties associated with these processes, results in an optimal estimate in terms of minimum mean square error (Kalman (1960)) for a linear system. The extended Kalman filter (EKF) is an extension for nonlinear systems in which, for every new time interval, the system and measurement models are linearized about the current estimate of the state.

The algorithm consists of two steps in every iteration, the state estimate of the current iteration is firstly obtained as a transform of the previous one using the system's dynamical model. In this **prediction** step, because of uncertainties related to the model, the resulting *a piori* estimate has a higher covariance. In the second step, the sensor measurement is used to **update** the estimate and lower the uncertainty, using a weighted average, called the Kalman gain. Hence, the final estimate of the system's state lies between the predicted value and the measurement, and has a better estimated uncertainty than either alone. The weights are calculated from the covariance, and thus, are distributed accordingly to the certainty associated with the system and sensor models.

This process is repeated at every instance that a new measurement and its covariance is available. As the system's state and model describe the dynamics of the system, the filter requires only the last estimate and its covariance to calculate

the new estimate with no past information required. Although in the classic implementation the frequency of this process depends on the rate of measurement, as we explain later in this section, this is not mandatory. Thus, the updates based on the system model can be virtually performed at any desired rate with the cost of lower certainty in the produced estimate.

Kalman filtering provides an effective way to handle noisy sensor data and uncertainties related to the system model, which are usually due to simplifications. Nevertheless, the correctness of the model, accuracy of measurements and other unaccounted factors place limits on the estimation performance and reliability of the results. KF it is a common sensor fusion and data fusion algorithm with numerous applications such as in guidance and navigation technologies. Is is widely used in the control of robots and in the estimation of their state.

2.4.1 General Equations

A discrete-time linear dynamic system can be described with the following matrix difference equation:

$$\boldsymbol{q}_k = \boldsymbol{F}_k \boldsymbol{q}_{k-1} + \boldsymbol{G}_k(\boldsymbol{u}_k + \boldsymbol{\nu}_{u_k}), \qquad k = 0, 1, \ldots \tag{2.31}$$

where $\boldsymbol{q}_k$ is a state vector, $\boldsymbol{F}_k$ is a state transition matrix, $\boldsymbol{u}_k$ is a known input vector, e.g., control signal, $\boldsymbol{G}_k$ is an input matrix, and $\boldsymbol{\nu}_{u_k}$ is a zero-mean white Gaussian process noise, $\boldsymbol{\nu}_{u_k} \sim N(0, \boldsymbol{R}_k)$, with covariance

$$\boldsymbol{Q}_k = E[\boldsymbol{\nu}_{u_k} \boldsymbol{\nu}_{u_k}^{'}]. \tag{2.32}$$

A general measurement equation can be written as

$$\boldsymbol{z}_k = \boldsymbol{H}_k \boldsymbol{q}_k + \boldsymbol{\nu}_{z_k}, \qquad k = 0, 1, \ldots \tag{2.33}$$

where $\boldsymbol{z}_k$ is the measurement vector, $\boldsymbol{H}_k$ is a measurement (observation) matrix, and $\boldsymbol{\nu}_{z_k}$ is a zero-mean measurement noise, $\boldsymbol{\nu}_{z_k} \sim N(0, \boldsymbol{R}_k)$, with covariance

$$\boldsymbol{R}_k = E[\boldsymbol{\nu}_{z_k} \boldsymbol{\nu}_{z_k}^{'}]. \tag{2.34}$$

The matrices $\boldsymbol{F}_k$, $\boldsymbol{G}_k$, $\boldsymbol{H}_k$, $\boldsymbol{Q}_k$, and $\boldsymbol{R}_k$ are assumed known and can be either time-varying or constant, and in the later case the time indices k can be dropped. The state vector is modeled as a random variables with Gaussian distribution, and the process and measurement noise are assumed independent.

Step 1. Prediction

As mentioned before, Kalman filtering consists of two estimation steps, in the first step the dynamic model of the system, eq. (2.31), is used to propagate the previous estimate as

$$\hat{\boldsymbol{q}}_k^- = \boldsymbol{F}_k \hat{\boldsymbol{q}}_{k-1} + \boldsymbol{G}_k \boldsymbol{u}_k, \tag{2.35}$$

where $\hat{\boldsymbol{q}}_k^-$ is the *a priori* estimate at instant k, and $\hat{\boldsymbol{q}}_{k-1}$ is the state estimate at instant $k-1$. The above $\hat{\cdot}$ notation represents the mean value of a random variable $\boldsymbol{q}_k$, whose *a priori* covariance can be expressed as

$$\boldsymbol{P}_k^- = \boldsymbol{F}_k \boldsymbol{P}_{k-1} \boldsymbol{F}_k^{\mathrm{T}} + \boldsymbol{G}_k \boldsymbol{Q}_k \tag{2.36}$$

Step 2. Correction

With every prediction step the covariance $\boldsymbol{P}_k^-$ will inherently grow, i.e., propagation of the mean value in eq. (2.35) always entails an increase of uncertainty. The estimate in eq. (2.35) can be updated given a measurement $\boldsymbol{z}_k$. First, the residual value is computed as

$$\tilde{\boldsymbol{y}}_k = \boldsymbol{z}_k - \boldsymbol{H}_k \hat{\boldsymbol{q}}_k^-, \tag{2.37}$$

and the residual covariance as

$$\boldsymbol{S}_k = \boldsymbol{H}_k \boldsymbol{P}_k^- \boldsymbol{H}_k^{\mathrm{T}} + \boldsymbol{R}_k, \tag{2.38}$$

then, with the Kalman filter gain

$$\boldsymbol{K}_k = \boldsymbol{P}_k^- \boldsymbol{H}_k^{\mathrm{T}} \boldsymbol{S}_k^{-1}, \tag{2.39}$$

the *a posteriori* state estimate and estimate covariance can be obtained as

$$\hat{\boldsymbol{q}}_k = \hat{\boldsymbol{q}}_k^- + \boldsymbol{K}_k \tilde{\boldsymbol{y}}_k, \tag{2.40}$$

$$\boldsymbol{P}_k = (\boldsymbol{I} - \boldsymbol{K}_k \boldsymbol{H}_k) \boldsymbol{P}_k^-. \tag{2.41}$$

Remark 2.1 *Initial values of $\hat{\boldsymbol{q}}_0$ and $\boldsymbol{P}_0$ are needed at initial step $k = 1$, if $\hat{\boldsymbol{q}}_0$ is known then $\boldsymbol{P}_0 = \boldsymbol{0}$, $\boldsymbol{P}_0 = \lambda \boldsymbol{I}$ otherwise, λ sufficiently large.*

If we expand Eq. (2.40) using the residual formulation from Eq. (2.37), we get

$$\hat{\boldsymbol{q}}_k = (\boldsymbol{1} - \boldsymbol{K}_k \boldsymbol{H}_k) \hat{\boldsymbol{q}}_k^- + \boldsymbol{K}_k \boldsymbol{z}_k, \tag{2.42}$$

which represents a weighted average of the *a priori* estimate $\hat{\boldsymbol{q}}_k^-$ and the measurement $\boldsymbol{z}_k$. Hence, the values with better estimated uncertainty are trusted more.

In order to achieve the desired behavior of the filter the certainty of the measurements and predictions based on the system model, have to be properly determined.

As the the values of these quantities usually have to be assumed, at least to a certain extent, the performance of the filter can be tuned by altering them. Thus, by changing the respective covariances the filter's responsiveness to the model and to the measurements can be tuned. A high value of the resulting Kalman gain will result in a fluctuating output while a lower gain will follow the prediction more closely but reduce the filter's responsiveness.

Extended Kalman Filter

To generalize this to nonlinear systems, let us consider the following model and measurement equations:

$$\boldsymbol{q}_k = f(\boldsymbol{q}_{k-1}, \boldsymbol{u}_k) + \boldsymbol{\nu}_{u_k}, \tag{2.43}$$

$$\boldsymbol{z}_k = h(\boldsymbol{q}_{k-1}) + \boldsymbol{\nu}_{z_k}, \tag{2.44}$$

where f and h are nonlinear functions of the state vector. Although eq. (2.43) can be used directly in eq. (2.35) to obtain the *a priori* estimate $\boldsymbol{q}_k^-$, and, respectively, eq. (2.44) in eq. (2.37), matrices $\boldsymbol{F}_k$, $\boldsymbol{G}_k$, and $\boldsymbol{H}_k$ are needed in the consecutive steps. Assuming that f and h are differentiable, the required matrices can be calculated as partial derivatives, which essentially linearizes f and h around the current estimate

$$\boldsymbol{F}_k = \left.\frac{\partial f}{\partial \boldsymbol{q}_k}\right|_{\hat{\boldsymbol{q}}_{k-1}, \boldsymbol{u}_k}, \quad \boldsymbol{G}_k = \left.\frac{\partial f}{\partial \boldsymbol{u}_k}\right|_{\hat{\boldsymbol{q}}_{k-1}, \boldsymbol{u}_k}, \quad \boldsymbol{H}_k = \left.\frac{\partial h}{\partial \boldsymbol{q}_k}\right|_{\hat{\boldsymbol{q}}_{k-1}} \tag{2.45}$$

2.4.2 Variations in the Implementation

The standard state progression of a Kalman filter is presented in Fig. 2.7, where at every time instance k (iteration) the *a priori* estimate $\hat{\boldsymbol{q}}_k^-$ is obtained based on the system model, the estimate from the previous step $\hat{\boldsymbol{q}}_{k-1}$, and the input to the system $\boldsymbol{u}_{k-1}$. Consequently, the *a posteriori* estimate $\hat{\boldsymbol{q}}_k$ is calculated given the current measurement $\tilde{\boldsymbol{z}}_k$ using the measurement model.

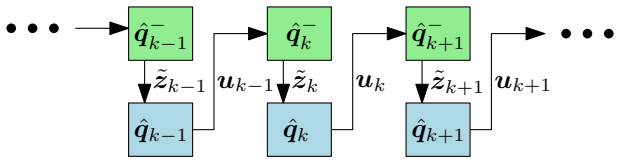


Figure 2.7: State progression in a classical Kalman filter.

In this classical implementation, Kalman filtering is an iterative, discrete-time process, where the frequency of new estimates depends on the frequency of observations. An example of such a case can be a situation when we want to get

information of a systems that we do not directly control or when the observation frequency is high enough for the control purposes. If higher rate estimation is required, new estimates can be calculated using the system model alone and updated with measurements when the later are available. This process, however, is subject to cumulative errors which is reflected in covariance growth over consecutive steps. Examples of such cases are path integration or dead reckoning in general, where the object's position can be updated knowing its previous location and velocity. Over time, however, the accuracy of such an estimate decreases.

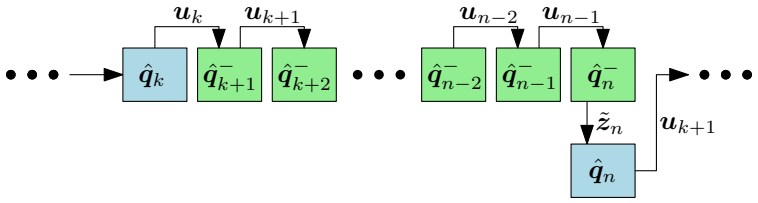


Figure 2.8: State progression of a KF with sparse measurements or when higher frequency of the estimate is needed.

Fig. 2.8 depicts the case with multiple state updates in between sparse measurements. Consecutive system inputs $\boldsymbol{u}_k$ to $\boldsymbol{u}_{n-1}$, from time instant k to n, respectively, are used to update the estimate until a new observation $\tilde{\boldsymbol{z}}_n$ is available and the estimate can be corrected.

In order to increase the estimate's accuracy or in a case when multiple sensors are available (or necessary, if individual sensors do not cover enough of the state vector's components) multiple updates per iteration step are possible. An example block diagram of this case is shown in Fig. 2.9, where at every time instant k two independent measurements, $\tilde{\boldsymbol{z}}^1_k$ and $\tilde{\boldsymbol{z}}^2_k$, are available. In this example, $\hat{\boldsymbol{q}}'_k$ represents an intermediate estimate after an update with measurement $\tilde{\boldsymbol{z}}^1_k$, and $\hat{\boldsymbol{q}}''_k$ the *a posteriori* estimate for the given time instant.

In general, variations of cases from Fig. 2.8 and 2.9 are possible, i.e., when more than one measurement is available although not simultaneously with other measurements, and with multiple state updates between measurements.

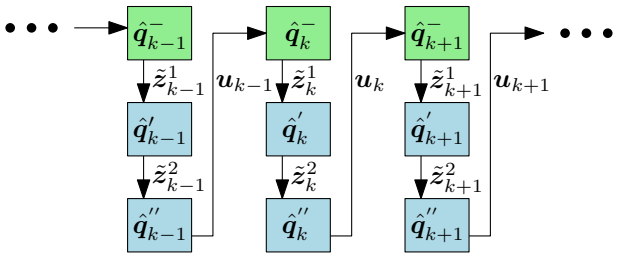


Figure 2.9: State progression of a KF with multiple measurements per iteration.

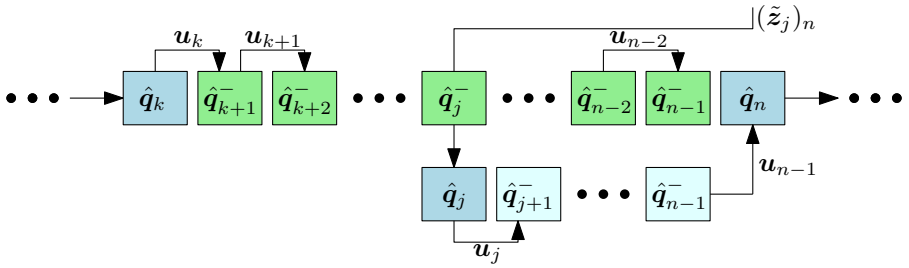


Figure 2.10: State progression of a Kalman filter with asynchronous measurements.

Asynchronous (E)KF

Ideally, when dealing with an estimation problem using Kalman filtering, as demonstrated in the case shown in Fig. 2.7, reliable measurement is available with a high enough frequency to generate state estimates for control purposes. Although variations of the KF (e.g., Fig. 2.8 and 2.9) can provide solutions in non-standard cases, they do not cover all real world scenarios.

Sensors are also dynamic systems and the output state (measurement) does not change instantly when an input change occurs (measured quantity). It will rather adjust the output value over a period of time - the response time T_r. The response time can be defined as the time that is required for a sensor to change its output value from a previous state to a new, steady value within a tolerance band of the correct output.

This problem can be neglected if a sensor's response time is smaller than half of the desired estimation period T_e

$$T_r < T_e/2, \quad T_e = 1/f_e. \tag{2.46}$$

A delayed measurement with respect to an *a priori* estimate at instant k, by less than half of the estimation period, still corresponds to that same time instant.

A Kalman filter requires that measurements are used to update estimates from the corresponding time instances. Therefore, delayed observations do not meet the criteria in Eq. (2.46) and should not be used directly in the time step they are available.This is especially true when the observation is not a raw sensor output but involves additional processing of sensory data, such as when the measurement is an output from a computer vision algorithm, for example, visual odometry or object recognition and detection.

One of possible solutions to this problem is shown in Fig. 2.10 as another KF block diagram. Assuming a system with infrequent and delayed measurements it is a similar case to the example shown in Fig. 2.8, where the system model is used between consecutive measurements to obtain a higher rate of estimates. The difference is that the measurement $(\tilde{\boldsymbol{z}}_j)_n$ available at instant n, is delayed, and thus

corresponds to a past time instant j.

As explained before, in a proper Kalman filtering process, $(\tilde{\boldsymbol{z}}_j)_n$ should be used to update the corresponding estimate $\hat{\boldsymbol{q}}_j^-$. In order to do that, sequences of past control inputs and corresponding estimates should be stored. This allows our algorithm to calculate the update at the proper time instant and use the stored values of the control input $\boldsymbol{u}_j$, $j = \{i, ..., n-1\}$, to propagate this estimate until the current time instant n.

A more specific example of a KF in this configuration is presented in Sec. 2.4.3 describing our Kalman filter design for the European Robotics Challenges (EuRoC).

Measurements as the Input

Normally, the Kalman filter equations shown in the previous section, for state updates and corrections using measurements, can be used directly, either in their standard or extended version. This is true as long as the measurement model shown in Eq. (2.33) holds, that is, the measurement is a linear (or non-linear in EKF) function of the state vector. Many sensors that output position or distance, fall into that category, however, sensor measurements of higher differential order than in the state vector can not be used directly.

The classical examples are inertial navigation systems (INS) in which inertia measurement unit (IMU) readings are integrated in the system model to estimate, consecutively, system's velocity and position. Because of simplifications and inaccuracies of the dynamic model of a system, acceleration and angular rate provided by an IMU tend to be more accurate than the values calculated from the system's dynamic model.

For example, there are a few steps involved to obtain the same quantities using the dynamic model of a quadrotor. Firstly, the motor and propeller models are used to estimate generated forces and torques, which are consequently used in the dynamic model of the quadrotor to obtain its acceleration and angular rate. This process contains simplifications (motor dynamics, higher order aerodynamically effects, approximated values of the quadrotor's mass and inertia) that can be potentially neglected for control purposes but result in a higher uncertainty than the propagation of IMU measurements.

In such a case, IMU measurements are used as the input to a simplified model, describing integration of the inertia values to velocities and position of the system. Depending on the reference frame, this model has to contain proper transformation of the values read in the sensor/body frame.

Bias Estimation

As mentioned before, Kalman filtering ensures optimal estimation results in terms of the mean squared error. This is valid, however, when the noise associated with

the state update and measurement has the form of white noise, i.e., it has a zero-mean Gaussian distribution. Thus, any offset, either constant or drifting, will result in an inaccurate estimation.

If sensor or input biases can not be eliminated through calibration, additional estimation of these quantities is required. If a complementary measurement to the biased value is available, this can be performed on-line with the same KF by extending its state with new quantities. Examples of bias estimation within the Kalman filter can be seen in the examples presented later in this chapter.

2.4.3 EuRoC Kalman filter

The European Robotics Challenges[1] (EuRoC) initiative is a set of industry-relevant challenges that aim to speed up the process of bringing innovative technologies from research labs to industry. It consisted of three challenges with multiple tasks and stages from the wide spectrum of industry related robotics topics such as: reconfigurable manufacturing cells, logistics and manipulation, and plant inspection and servicing.

One of the challenges[2] aimed at targeting the open problems in the existing UAV solutions, especially multicopters, to enable their deployment in real life scenarios. It involved the development of a localization and state estimation algorithm without any external positioning systems which was aligned with the topic of this thesis. The organizers provided datasets, sensor measurements recorded in real UAV flights, and comprehensive benchmarking tools, simulation environment with ground truth data for evaluation. We found it of great use in the development of algorithms for our teleoperated platform.

The main tasks that we were interested in were tasks 3 and 4 of challenge 3:

- UAV control (hovering) in different conditions (wind gusts),
- UAV way point navigation in different conditions (sensors, obstacles).

One of the goals in EuRoC was to enable teleoperation inspection tasks by inspection experts, even those untrained in piloting UAVs. As until recently the complexity of UAV control has required a skilled operator, applications such as UAV inspection have not been possible. On-board localization and state estimation also enable further capabilities, such as path planning and obstacle avoidance, which makes the deployment of UAVs in real life applications possible.

Problem Statement

The goal of the aforementioned tasks was to develop a control strategy for a UAV, simulated together with its sensors in the provided software environment. The UAV,

[1] `http://www.euroc-project.eu/`

[2] `http://www.euroc-project.eu/index.php?id=challenge_3`

whose body frame we denote as Q, was equipped with an IMU and depending on the task either one or two optical pose sensors. The sensors were fixed with respect to the body frame Q, the IMU at its center and with its own frame aligned to Q, and the pose sensors, P_1 and P_2, rotated and shifted with respect to Q by known extrinsic parameters.

We aimed at developing a state estimator to compute the full 6 degrees of freedom pose of the platform, an essential component of this task as its accuracy directly influences the accuracy of the control part. The simulated sensors provided inertial measurements at 100 Hz, pose measurements at 10 Hz, and ground truth data for evaluation.

Sensor Models

The provided IMU data contained measurements of the linear acceleration and the angular velocity of the UAV in the sensor frame. These measurements were affected by errors, a white noise and a time-varying offset. Except with respect to the model presented in Sec. 2.2.4, the measurements were not affected by scaling biases. Considering the non-inertial character of the sensor frame, the measured acceleration also consists of the gravity term $\boldsymbol{g}_Q$.

Hence in this example, we model the measurements as

$$\widetilde{\boldsymbol{a}} = \boldsymbol{a} + \boldsymbol{\nu}_a + \boldsymbol{\delta}_a, \tag{2.47}$$

$$\widetilde{\boldsymbol{\omega}} = \boldsymbol{\omega} + \boldsymbol{\nu}_\omega + \boldsymbol{\delta}_\omega, \tag{2.48}$$

where again, for $\boldsymbol{i} = \{\boldsymbol{a}, \boldsymbol{\omega}\}$, $\widetilde{\boldsymbol{i}}$ denotes the measured value, $\boldsymbol{i}$ the real value of the acceleration and angular rate respectively, $\boldsymbol{\nu}_i$ is a zero-mean random error, and $\boldsymbol{\delta}_i$ is a time-varying offset. With respect to the IMU model in Eq. (2.11) and (2.12), this model does not contain scaling factors. Considering the additional gravity part, Eq. (2.14) holds

$$\boldsymbol{a} = \boldsymbol{a}_Q - \boldsymbol{g}_Q = \boldsymbol{a}_Q - \boldsymbol{R}_W^Q \boldsymbol{g}_W. \tag{2.49}$$

The pose sensor's data was 10 Hz pose measurements, i.e. position and orientation,. Although being encumbered only by a white noise error, this data was elaborately delayed by about 0.1 s with respect to the IMU measurements to reflect the processing time of optical pose sensors. These sensors can be modeled as

$${}^W\widetilde{\boldsymbol{p}}_{P_i} = {}^W\boldsymbol{p}_{P_i} + \boldsymbol{\nu}_{p_i}, \tag{2.50}$$

$${}^W\widetilde{\boldsymbol{\eta}}_{P_i} = {}^W\boldsymbol{\eta}_{P_i} + \boldsymbol{\nu}_{\eta_i}, \tag{2.51}$$

where ${}^W\boldsymbol{p}_{P_i}$ is the position of the i-th sensor in the world frame, $\boldsymbol{p} = [x,\, y,\, z]$, and ${}^W\boldsymbol{\eta}_{P_i}$ is its orientation in the form of the roll, pitch and yaw angles, $\boldsymbol{\eta} = [\phi,\, \theta,\, \psi]$.

To obtain the corresponding pose of the robot given ${}^{W}\widetilde{\boldsymbol{p}}_{P_i}$ and ${}^{W}\widetilde{\boldsymbol{\eta}}_{P_i}$ we use

$$ {}^{W}\widetilde{\boldsymbol{p}}_{Q_i} = \boldsymbol{R}_{Q}^{W} \cdot {}^{Q}\boldsymbol{p}_{P_i} - {}^{W}\widetilde{\boldsymbol{p}}_{P_i}, \tag{2.52} $$

$$ \widetilde{\boldsymbol{R}}_{Q_i}^{W} = \widetilde{\boldsymbol{R}}_{P_i}^{W} \left({}^{W}\widetilde{\boldsymbol{\eta}}_{P_i} \right) \boldsymbol{R}_{P_i}^{Q\,\mathrm{T}}, \tag{2.53} $$

where ${}^{Q}\boldsymbol{p}_{P_i}$ and $\boldsymbol{R}_{P_i}^{Q}$ are the position and orientation of the i-th sensor in the robot's frame Q, respectively, and they are known extrinsic parameters. The corresponding roll, pitch and yaw angles, ${}^{W}\boldsymbol{\eta}_{Q_i}$, can be calculated from the rotation matrix in Eq. (2.53) as shown in Sec. 2.2.

Kalman Filter Design

The formal definition of the state vector

$$ \boldsymbol{q} = [{}^{W}\boldsymbol{p}_{Q}^{\mathrm{T}},\ {}^{W}\dot{\boldsymbol{p}}_{Q}^{\mathrm{T}},\ {}^{W}\boldsymbol{\eta}_{Q}^{\mathrm{T}}]^{\mathrm{T}}, \tag{2.54} $$

contains the position of the robot ${}^{W}\boldsymbol{p}_{Q}$, its velocity ${}^{W}\dot{\boldsymbol{p}}_{Q}$, and orientation ${}^{W}\boldsymbol{\eta}_{Q}$ in the world reference frame W. For the sake of clarity, we omit the frame indices and write Eq. (2.54) as

$$ \boldsymbol{q} = [x,\ y,\ z,\ \dot{x},\ \dot{y},\ \dot{z},\ \phi,\ \theta,\ \psi]^{\mathrm{T}}. \tag{2.55} $$

Following the notation introduced in Sec. 2.4.1, we modeled the system as

$$ \boldsymbol{q}_k = \boldsymbol{F}\boldsymbol{q}_{k-1} + \boldsymbol{G}\boldsymbol{u}_k, \tag{2.56} $$

where

$$ \boldsymbol{F} = \begin{bmatrix} \boldsymbol{I}_{3x3} & \Delta t\boldsymbol{I}_{3x3} & \boldsymbol{0}_{3x3} \\ \boldsymbol{0}_{3x3} & \boldsymbol{I}_{3x3} & \boldsymbol{0}_{3x3} \\ \boldsymbol{0}_{3x3} & \boldsymbol{0}_{3x3} & \boldsymbol{I}_{3x3} \end{bmatrix}, \quad \boldsymbol{G} = \begin{bmatrix} \boldsymbol{0}_{3x3} & \boldsymbol{0}_{3x3} \\ \Delta t\boldsymbol{R}_{Q_k}^{W} & \boldsymbol{0}_{3x3} \\ \boldsymbol{0}_{3x3} & \Delta t\boldsymbol{D}_k \end{bmatrix}, \tag{2.57} $$

and

$$ \boldsymbol{u}_k = [\widetilde{\boldsymbol{a}}_k^{\mathrm{T}},\ \widetilde{\boldsymbol{\omega}}_k^{\mathrm{T}}]^{\mathrm{T}} \tag{2.58} $$

represent the IMU measurement, (2.47) and (2.48) at instant k and includes noise as introduced earlier. The rotation between the sensor (robot) frame and the world frame is represented by $\boldsymbol{R}_{Q_k}^{W}$ in matrix $\boldsymbol{G}$. $\boldsymbol{D}_k$ defines the transformation between the angular velocity read by the sensor and the RPY rates as defined in Eq. (2.26).

As in Eq. (2.33) in Sec. 2.4.1 we modeled the pose sensor's measurements as

$$ \boldsymbol{z}_{i_k} = \boldsymbol{H}_k\boldsymbol{q}_k + \boldsymbol{\nu}_{z_k}, \tag{2.59} $$

where

$$ \boldsymbol{z}_{i_k} = [{}^{W}\widetilde{\boldsymbol{p}}_{Q_i}^{\mathrm{T}},\ {}^{W}\widetilde{\boldsymbol{\eta}}_{Q_i}^{\mathrm{T}}]^{\mathrm{T}} \tag{2.60} $$

is the pose from the i-th pose sensor, given relations in (2.52) and (2.53), respec-

tively, and

$$\boldsymbol{H} = \begin{bmatrix} \boldsymbol{I}_{3x3} & \boldsymbol{0}_{3x3} & \boldsymbol{0}_{3x3} \\ \boldsymbol{0}_{3x3} & \boldsymbol{0}_{3x3} & \boldsymbol{I}_{3x3} \end{bmatrix}. \tag{2.61}$$

The above quantities enable consecutive state estimations in the iterative KF process described in Sec. 2.4.1. As the pose sensor's readings were delayed, and at a much lower frequency than the IMU measurements, we utilized the asynchronous KF strategy as shown earlier in Fig. 2.10. The main update loop with predictions based on Eq. (2.35) with matrices defined in this section is run with the rate of the IMU measurements and the updates are performed whenever information from the pose sensor becomes available.

Bias Estimation

Although the zero-mean error can be filtered out in the estimation process, the IMU offsets, $\boldsymbol{\delta}_a$ and $\boldsymbol{\delta}_\omega$, must be estimated in order to decrease their negative effect on the state estimate. In order to do this we introduce an extended state that now includes these additional quantities

$$\boldsymbol{q}^\star = [x,\, y,\, z,\, \dot{x},\, \dot{y},\, \dot{z},\, \delta_{\ddot{x}},\, \delta_{\ddot{y}},\, \delta_{\ddot{z}},\, \phi,\, \theta,\, \psi,\, \delta_{\omega_x},\, \delta_{\omega_y},\, \delta_{\omega_z}]^{\mathrm{T}}, \tag{2.62}$$

where $[\delta_{\ddot{x}},\, \delta_{\ddot{y}},\, \delta_{\ddot{z}}] = \boldsymbol{\delta}_a^{\mathrm{T}}$ and $[\delta_{\omega_x},\, \delta_{\omega_y},\, \delta_{\omega_z}] = \boldsymbol{\delta}_\omega^{\mathrm{T}}$.

The extended state $\boldsymbol{q}^\star$ requires additional extensions of the matrices $\boldsymbol{F}$, $\boldsymbol{G}$ and $\boldsymbol{H}$ to accommodate the new quantities and their relation with the other state variables. Thus

$$\boldsymbol{F}^\star = \begin{bmatrix} \boldsymbol{I}_{3x3} & \Delta t\boldsymbol{I}_{3x3} & \boldsymbol{0}_{3x3} & \boldsymbol{0}_{3x3} & \boldsymbol{0}_{3x3} \\ \boldsymbol{0}_{3x3} & \boldsymbol{I}_{3x3} & \boldsymbol{0}_{3x3} & \boldsymbol{0}_{3x3} & \boldsymbol{0}_{3x3} \\ \boldsymbol{0}_{3x3} & \boldsymbol{0}_{3x3} & -\Delta t\boldsymbol{R}_{Q_k}^W & \boldsymbol{0}_{3x3} & \boldsymbol{0}_{3x3} \\ \boldsymbol{0}_{3x3} & \boldsymbol{0}_{3x3} & \boldsymbol{0}_{3x3} & \boldsymbol{I}_{3x3} & \boldsymbol{0}_{3x3} \\ \boldsymbol{0}_{3x3} & \boldsymbol{0}_{3x3} & \boldsymbol{0}_{3x3} & \boldsymbol{0}_{3x3} & -\Delta t\boldsymbol{D}_k^W \end{bmatrix}, \quad \boldsymbol{G}^\star = \begin{bmatrix} \boldsymbol{0}_{3x3} & \boldsymbol{0}_{3x3} \\ \Delta t\boldsymbol{R}_{Q_k}^W & \boldsymbol{0}_{3x3} \\ \boldsymbol{0}_{3x3} & \boldsymbol{0}_{3x3} \\ \boldsymbol{0}_{3x3} & \Delta t\boldsymbol{D}_k \\ \boldsymbol{0}_{3x3} & \boldsymbol{0}_{3x3} \end{bmatrix},$$

$$\boldsymbol{H}^\star = \begin{bmatrix} \boldsymbol{I}_{3x3} & \boldsymbol{0}_{3x3} & \boldsymbol{0}_{3x3} & \boldsymbol{0}_{3x3} & \boldsymbol{0}_{3x3} \\ \boldsymbol{0}_{3x3} & \boldsymbol{0}_{3x3} & \boldsymbol{0}_{3x3} & \boldsymbol{I}_{3x3} & \boldsymbol{0}_{3x3} \end{bmatrix}. \tag{2.63}$$

Estimation Results

We used the presented Kalman filter design to control the simulated UAV in tasks 3 and 4 of EuRoC challenge 3. As explained before, the robot was equipped with an IMU and two pose sensors with extrinsic parameters (sensor poses in the robot's

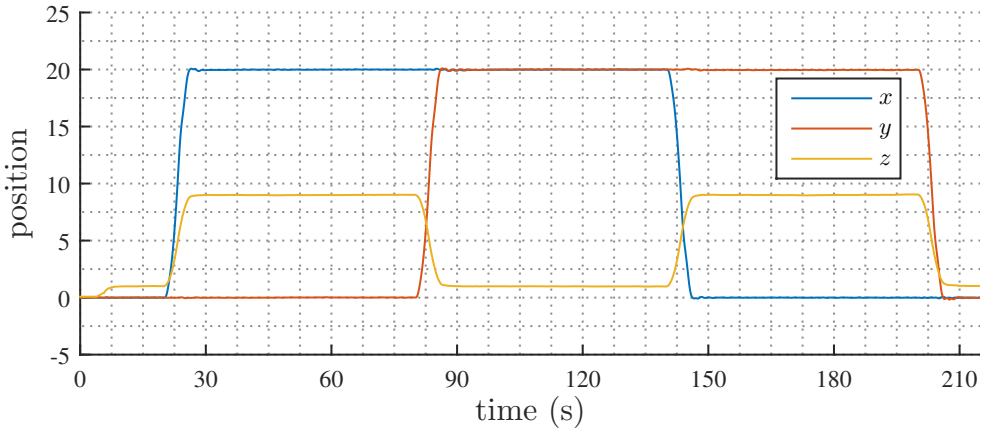


Figure 2.11: EuRoC Kalman filter estimation results, estimated position.

	x	y	z
$\boldsymbol{e}_p$ (m)	0.0116	0.0295	0.0161
$\boldsymbol{e}_\eta$ (rad)	0.0088	0.0057	0.0084

Table 2.1: Pose estimation in the EuRoC task 4.2, RMS errors.

frame)

$$
\begin{aligned}
{}^{Q}\boldsymbol{p}_{P_1} &= \begin{bmatrix} 0.03 \\ -0.07 \\ 0.1 \end{bmatrix}, \quad {}^{Q}\boldsymbol{p}_{P_2} = \begin{bmatrix} -0.05 \\ 0.08 \\ 0.1 \end{bmatrix}, \\
{}^{Q}\boldsymbol{\eta}_{P_1} &= \begin{bmatrix} 0.2 \\ -0.1 \\ 0.3 \end{bmatrix}, \quad {}^{Q}\boldsymbol{\eta}_{P_2} = \begin{bmatrix} -0.1 \\ 0.2 \\ -0.3 \end{bmatrix}.
\end{aligned} \tag{2.64}
$$

Here, we present the estimation results from task 4.2. The goal in this task was to perform way-point tracking within the given time limits. The resulting estimated position of the robot is shown in Fig. 2.11 and the estimation error, i.e., the difference between the estimated value and the ground truth data, is presented in Fig. 2.12.

The error of the estimated orientation is shown in Fig. 2.13. The overall performance of our estimator is presented in the form of the RMS errors in Table 2.1. Fig. 2.14 and 2.15 show the time evolution of the estimated biases, offsets on the accelerometer and gyroscope, respectively.

2.4.4 Vicon-IMU Integration

Vicon is an external tracking system (also called a motion capture system), which uses multiple cameras to track infrared reflective markers attached to an object

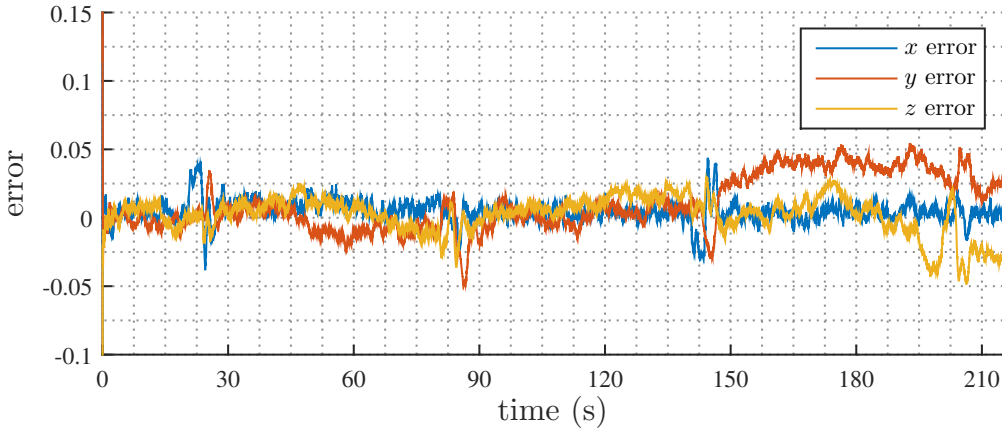


Figure 2.12: EuRoC Kalman filter estimation results, the estimated position error.

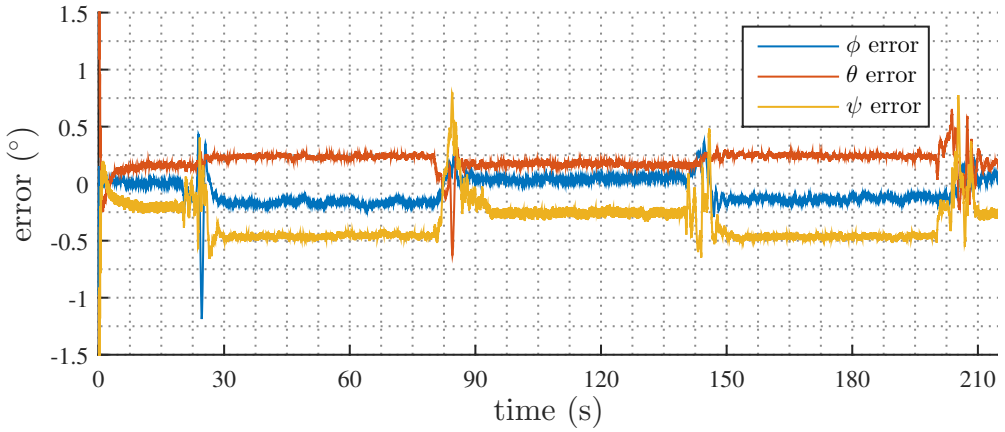


Figure 2.13: EuRoC Kalman filter estimation results, the estimated orientation error.

in a rigid configuration. The markers, attached to the object of interest, form the so-called vicon-tree, a geometrical shape with uniquely defined position and orientation in the Vicon frame of reference. Assuming enough camera coverage and visibility of the markers, the system reports the object's pose with high accuracy and relatively high frequency.

Vicon, and other motion capture systems, are very popular in robotics experiments as they separate the problem of state estimation from the control algorithms allowing researchers to focus on other aspects of robotics applications. One of many works that used such a system, is the work by Stegagno *et al.* (2014) with more references therein. We also utilized Vicon in the initial experiments of our obstacle avoidance algorithm (Odelga *et al.*, 2016b).

In our setup, the tracking system provided pose measurements at 120 Hz. Such a rate is usually sufficient for most of the control problems but in some cases more

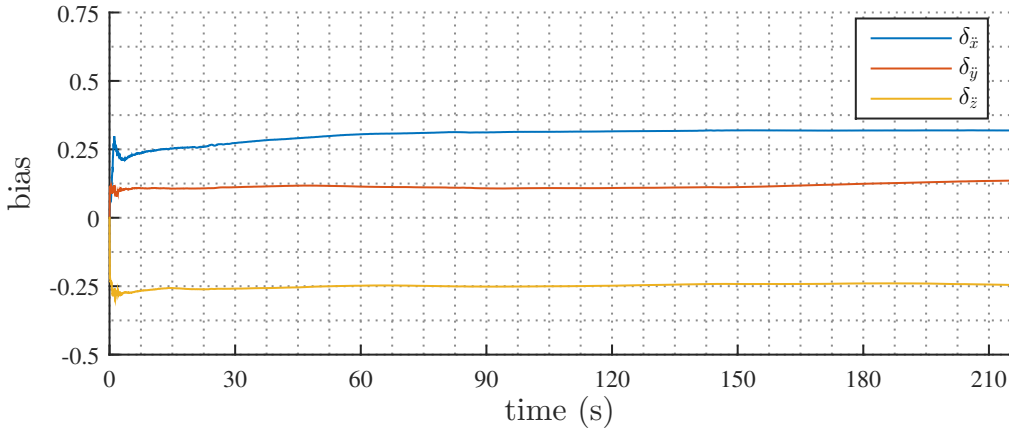


Figure 2.14: EuRoC Kalman filter estimation results, offset bias on acceleration.

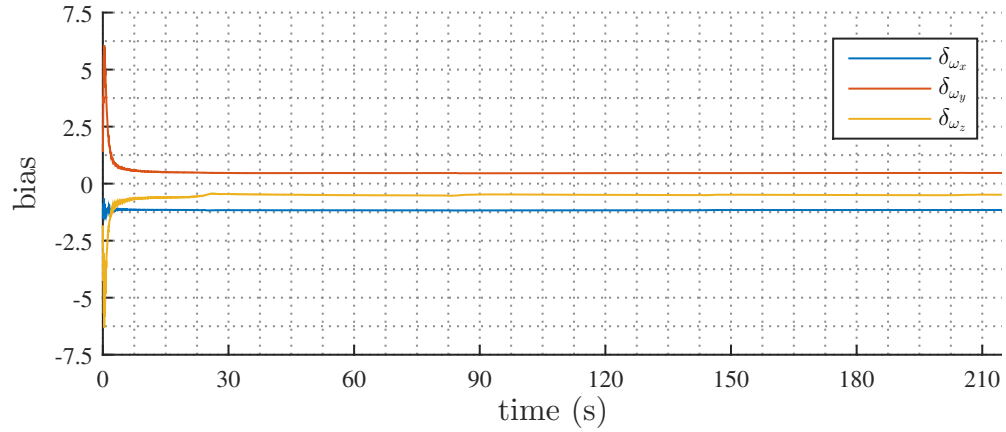


Figure 2.15: EuRoC Kalman filter estimation results, offset bias on angular rate.

precise information is required. For example, in controllers that require higher order differential components of the pose, like jerk or snap (Ryll *et al.*, 2015; Mellinger and Kumar, 2011; Odelga *et al.*, 2016a), additional sensors are necessary.

One of the strategies to provide higher frequency and more precise estimate is Vicon-IMU integration through Kalman filtering. Virtually identical in its configuration to the example provided in Sec. 2.4.3, such a system can not only provide higher differential components of the pose at higher frequencies but also increases the estimation robustness in the case of communication delays or the loss of Vicon packets. Additionally, using the extended state as in Eq. (2.62), the system can be used to calibrate the inertial sensor by estimating its offset biases.

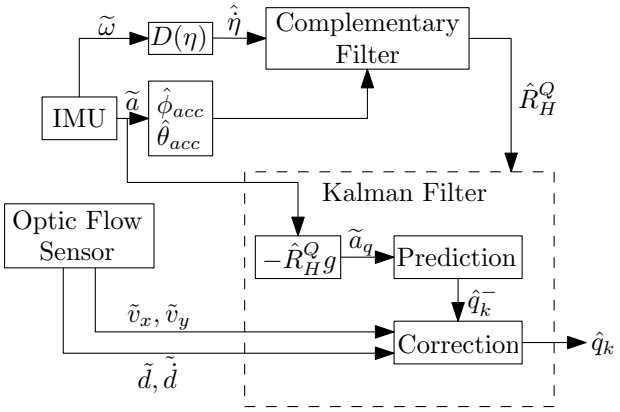


Figure 2.16: Block diagram of the estimation scheme.

2.5 On-board Velocity Estimation

The quantities that we estimate depend on two aspects. First of all, they reflect the control purposes as we need the direct or indirect observation of values that are being controlled. Secondly, depending on the robot's set of sensors, we can extend the state with variables that can improve the overall estimation and control performance, for example the biases in Eq. (2.62).

For a teleoperated UAV, we have a choice of three main modes of operation (Sec. 1.2.4): the attitude mode, velocity mode, and position- control mode. The most basic mode of operation, the attitude mode, although the simplest in terms of estimation is the most challenging for the pilot. Although the complementary filter alone would be sufficient in this mode, the limitations of this approach makes it inadequate in the context of this work. In terms of estimation, the position mode is the most challenging one, especially under the assumption of a GPS-denied environment and the use of on-board sensors only.

The state estimation algorithm that we present in this section and utilize in our teleoperation experiments is a Kalman filter-based velocity estimator. It estimates the robot's velocity in the horizontal frame, i.e., in a frame that moves and yaws with the robot but its orientation with respect to the gravity vector is constant (Sec. 2.2.2).

2.5.1 Estimator Design

Our algorithm implements the IMU-optical flow integration using the cascade architecture shown in Fig. 2.16 with complementary and Kalman filters. The complementary filter estimates the roll and pitch angles which define the platform's orientation with respect to the horizontal frame $\boldsymbol{R}_Q^H$. This orientation is used to project the IMU readings onto the desired directions. The projected measurements

are then integrated to obtain the platform's velocity ${}^{H}\dot{\boldsymbol{p}}_{Q}$. The use of an optical flow sensor enables compensation of the drift accumulating in the integration process. Since the optical flow sensor is also equipped with an echo sonar sensor to provide the metric reference for the flow based velocity estimate, we have extended the filter's state to also include the distance to the ground d.

The state vector of our Kalman filter is

$$\boldsymbol{q} = \begin{bmatrix} {}^{H}\dot{\boldsymbol{p}}_{Q}^{\mathrm{T}} & d \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} {}^{H}\dot{x}_{Q} & {}^{H}\dot{y}_{Q} & {}^{H}\dot{z}_{Q} & d \end{bmatrix}^{\mathrm{T}} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} & d \end{bmatrix}^{\mathrm{T}}. \tag{2.65}$$

We model the system with Eq. (2.31) , where

$$\begin{gathered} \boldsymbol{F}_{k} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & \Delta t & 1 \end{bmatrix}, \boldsymbol{G}_{k} = \begin{bmatrix} \Delta t \hat{\boldsymbol{R}}_{Q_k}^{H_k} \\ 0 \quad 0 \quad 0 \end{bmatrix} \\ \boldsymbol{u}_{k} = \bar{\boldsymbol{a}}_{k} - (\hat{\boldsymbol{R}}_{Q_k}^{H_k})^{\mathrm{T}} \boldsymbol{g}_{W}. \end{gathered} \tag{2.66}$$

The resulting estimate is inherently encumbered with accumulating error and noise resulting from inaccuracy in the estimation of the system's orientation $\hat{\boldsymbol{R}}_{Q_k}^{H_k}$. Using the optical flow sensor in the Kalman filter's update (Eq. (2.40)) we can compensate for this error.

We note the optical flow measurement as

$$\widetilde{\boldsymbol{z}}_{flow} = \begin{bmatrix} \tilde{v}_{x} \\ \tilde{v}_{y} \end{bmatrix}, \tag{2.67}$$

where $\tilde{v}_x$ and $\tilde{v}_y$ are the OF-based velocities obtained with the OF sensor model in Eq. (2.24). We use the following measurement model,

$$\widetilde{\boldsymbol{z}}_{flow} = \begin{bmatrix} \tilde{v}_{x} \\ \tilde{v}_{y} \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \boldsymbol{q}_{k} = \boldsymbol{C}_{flow} \boldsymbol{q}_{k}. \tag{2.68}$$

To correct the estimates of the horizontal velocity $\dot{z}$ and distance to the ground d. We use measurements from the echo sonar with the corresponding measurement model,

$$\widetilde{\boldsymbol{z}}_{dist} = \begin{bmatrix} \tilde{d}_{k} \\ \frac{\tilde{d}_{k} - \tilde{d}_{k-1}}{\Delta t} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix} \boldsymbol{q}_{k} = \boldsymbol{C}_{dist} \boldsymbol{q}_{k}, \tag{2.69}$$

It is important to note that the OF velocity measurement (2.67) is computed using the model in Eq. (2.24) with the estimate of the ground distance $\hat{d}$ that, thanks to the filtering process, has a lower covariance than the direct measurement from the echo sonar $\tilde{d}$.

	$\boldsymbol{\alpha}_a$	$\boldsymbol{\delta}_a$
x	0.9963	0.0028
y	1.0053	0.0096
z	1.0098	0.0517

Table 2.2: Estimated accelerometer biases

2.5.2 Bias Estimation

The IMU measurements used in the presented estimation scheme are assumed to be bias-corrected, as shown in Sec. 2.2.4. The EuRoC KF example shows that sensor calibration and bias estimation are important to achieve high estimation performance. Especially when dealing with real sensors, improper calibration can diminish the results. To define the bias of our sensor, we employed a two step process, an offline calibration and a pre-flight bias estimation with the KF with an extended state.

Offline Calibration

To calibrate the accelerometer, we have employed the least squares method based on the ellipsoid fitting described by (Bektas, 2015). This process requires samples of a known acceleration, e.g., gravity, in at least six widely-spread orientations and was performed off-line. The calibration results are used to compute the bias-corrected estimates (2.13). An example set of obtained values is presented in Table 2.2.

In principle, the same method could be applied to estimate the gyroscope bias. That would require, however, to rotate the sensor with a known angular velocity around different axes. Our experiments showed that the factory calibration of the gyroscope is accurate enough to neglect the scaling factor (i.e., $\boldsymbol{\alpha}_\omega \simeq 1$). The gyroscope offset $\boldsymbol{\delta}_\omega$ can be estimated by averaging steady measurements over a sampling period τ_s, which, for the platform presented in this work, is performed every time during the on-ground phase prior to take-off with motors switched off. The sampling period is set to 2 s, which corresponds to 1000 samples at 500 Hz.

On-ground Bias Estimation

Throughout our experiments, we noticed that the accelerometer biases (2.11) tend to drift slightly between consecutive tests, probably due to vibrations and impacts during landings. To mitigate this issue, we employed an additional on-ground bias

estimation based on an extension to our Kalman filter's state model (2.65):

$$\boldsymbol{q}^\star = \begin{bmatrix}\boldsymbol{q}^\mathrm{T} & \boldsymbol{\delta}_a^\mathrm{T}\end{bmatrix}^\mathrm{T} = \begin{bmatrix}\dot{x} & \dot{y} & \dot{z} & d & \delta_{\dot{x}} & \delta_{\dot{y}} & \delta_{\dot{z}}\end{bmatrix}^\mathrm{T}, \tag{2.70a}$$

$$\boldsymbol{F}_k^\star = \left[\begin{array}{cccc|ccc} 1 & 0 & 0 & 0 & & & \\ 0 & 1 & 0 & 0 & & -\Delta t\hat{\boldsymbol{R}}_{Q_k}^{H_k} & \\ 0 & 0 & 1 & 0 & & & \\ 0 & 0 & \Delta t & 1 & 0 & 0 & 0 \\ \hline & \boldsymbol{0}_{3\times4} & & & & \boldsymbol{I}_{3\times3} & \end{array}\right], \quad \boldsymbol{G}_k^\star = \begin{bmatrix} \Delta t\hat{\boldsymbol{R}}_{Q_k}^{H_k} \\ \\ \boldsymbol{0}_{4\times3} \end{bmatrix}. \tag{2.70b}$$

This bias estimation utilizes the fact that when the platform is on the ground its state is constant (in particular, its velocity is equal to zero) to correct the biases with assumed zero measurements in Eq. (2.67) and (2.69).

Once the robot is commanded to lift-off, the estimation system switches from Eq. (2.70) to the reduced form (2.65).

2.5.3 Estimation Results

In Figures 2.17-2.23, shown in this section, we present the results of our state estimation from one of our experiments. We recorded both the sensor measurements and the output from the filter, i.e., the estimate. As the reference value, we recorded the corresponding values using Vicon. In this experiment the robot was commanded to lift-off and then fly consecutively: forward, left, backward, right, forward-left, backward-right.

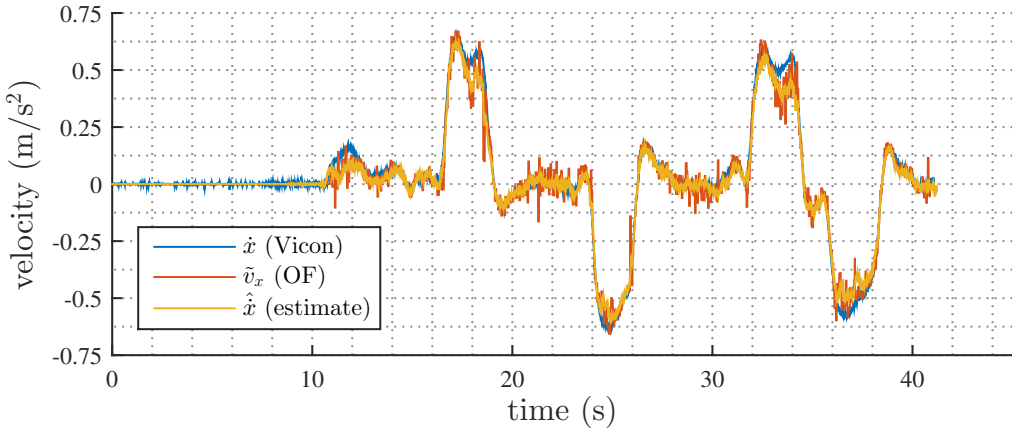


Figure 2.17: IMU-OF state estimation results, forward velocity.

Fig. 2.17 and 2.18 show the horizontal velocities of the robot. The vertical velocity and the distance to the ground are shown in Fig. 2.19 and 2.20, respectively. Please note that the distance to the ground d is not identical to the robot's z position

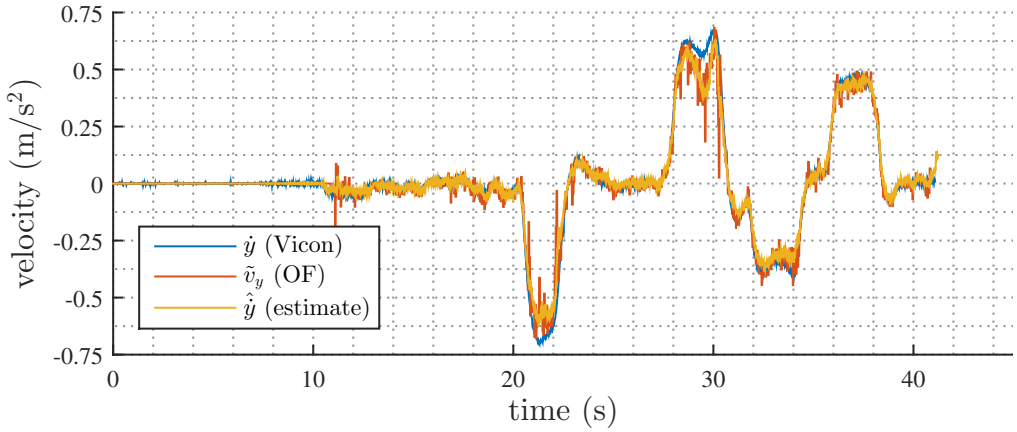


Figure 2.18: IMU-OF state estimation results, lateral velocity.

in the Vicon frame, as the sensor is attached with an offset from the center of the robot. This offset is visible in the plot in Fig. 2.20.

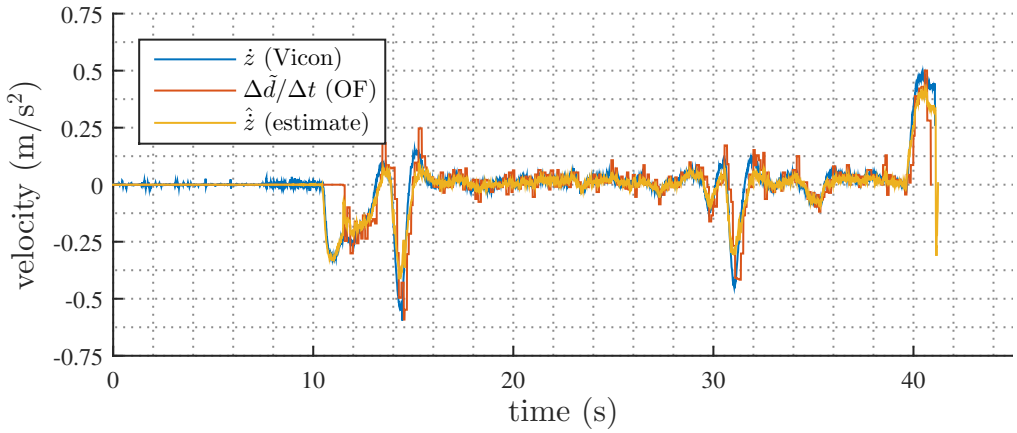


Figure 2.19: IMU-OF state estimation results, vertical velocity.

The on-ground accelerometer bias estimation result is shown in Fig. 2.21. Because of the correlation with the orientation estimation, the algorithm requires around 10 s to stabilize the bias value. After the robot is commanded to lift-off, our estimator switches to the simpler state and biases are no longer updated.

Finally, in the plots shown in Fig. 2.22 and 2.23 we present the results of the complementary filter, where the orientation of the robot is expressed in terms of the roll and pitch angles.

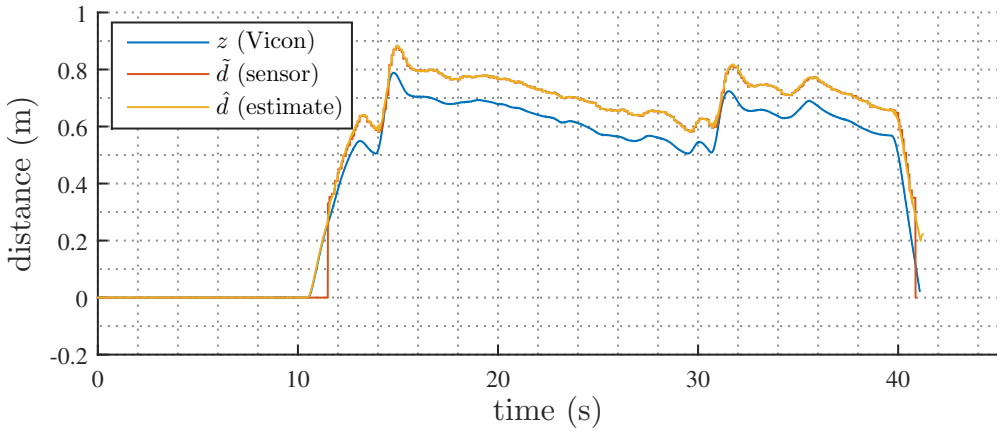


Figure 2.20: IMU-OF state estimation results, distance to the ground.

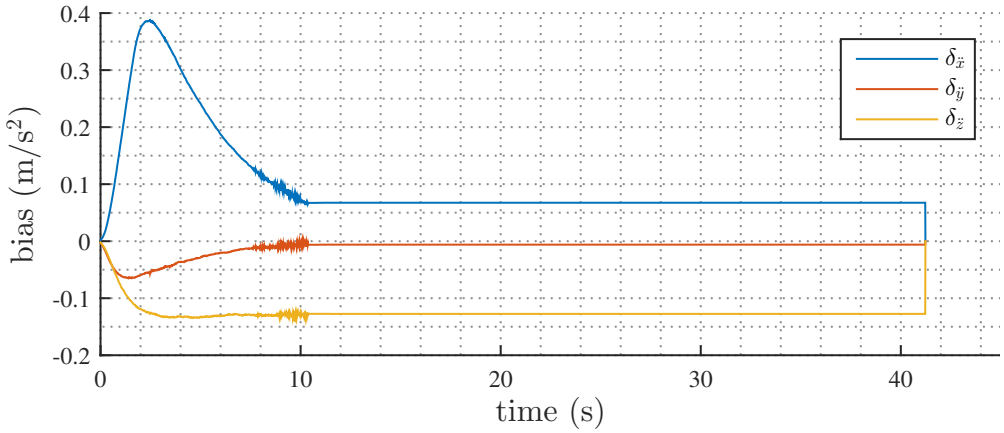


Figure 2.21: IMU-OF state estimation results, accelerometer offset bias.

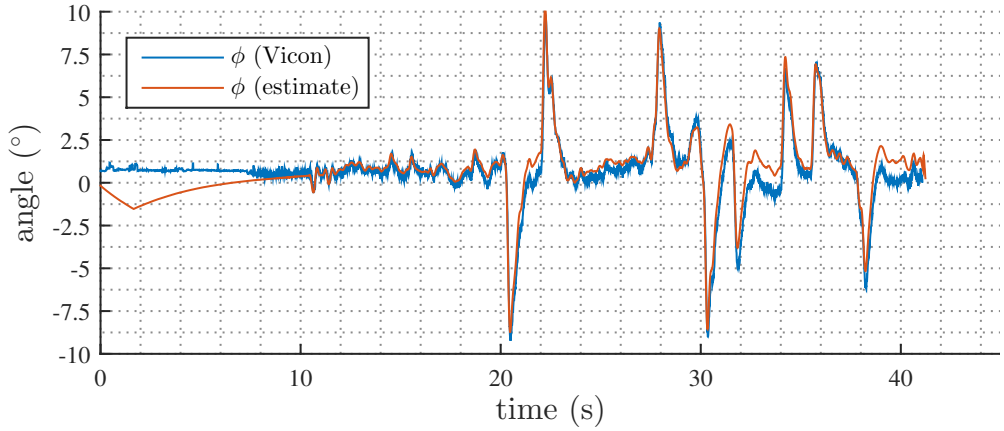


Figure 2.22: IMU-OF state estimation results, complementary filter: the roll angle.

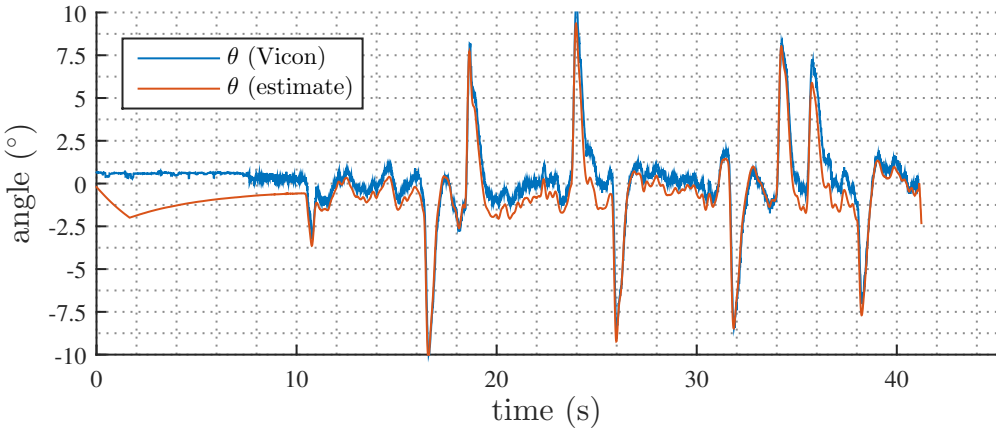


Figure 2.23: IMU-OF state estimation results, complementary filter: the pitch angle.

2.6 Software-in-the-loop Simulations

Software-in-the-loop simulations (SIL) are methods for testing algorithms, or entire code structures, in simulated environments. As SIL testing usually refers to real-time and dynamic systems, SIL simulations can greatly improve the time and cost of robotics development by enabling algorithm validation on a simulated system before deployment in the real world. For example, during development of a software for motion control, failures of the algorithm can result is crashes or collisions, broken components of the robot or damages to its surroundings and can endanger people as well. Hence, SIL simulations provide a convenient and safe method for code testing.

In our work we utilized SIL simulations to test and validate our state estimator. The EuRoC competition enabled twofold SIL testing, first, the organizers provided various datasets - recordings of sensor readings in different UAV flights together with the ground truth pose. The second testing modality was the provided physical simulation environment that included a controllable model of the UAV with sensors. The datasets allowed us to validate our KF state estimator against the provided ground truth data, fine tune its parameters to minimize the error between the estimated state and the real one, and subsequently develop further functionalities, e.g., bias estimation.

Although the real flight data enabled testing with real-life sensor readings, they did not test the stability of our state estimator when used in a control loop. To test that scenario, it is necessary to simulate not only the sensors but also the platform with its dynamics. The simulation environment provided in EuRoC consisted of an arbitrary UAV with its mass, inertia, propellers, and sensors in a ROS-compatible physical simulator - Gazebo. This enabled a proper SIL testing, i.e., UAV trajectory

control with Telekyb using our KF-based state estimator.

Subsequently we used a similar approach in the testing of our on-board state estimator described in Sec. 2.5.1. After retrofitting our quadrotor platform with an optical flow sensor, we performed a number of test flights with different levels of aggressive maneuvers in our lab with an external tracking system. Thus, we created our own datasets with recorded motion and sensor readings of our actual platform.

In principle we could follow the same procedure and simulate the whole platform in the simulator, however, the results with the recorded datasets were accurate enough to allow testing with our real platform directly.

2.7 Summary

In this chapter we showed in detail the development process of our on-board state estimation algorithm. Although it is based on a classical sensor fusion algorithm - Kalman filtering, we paid special attention to the particularity of our application. We discussed the use of KF in non-standard scenarios, in different sensor setups and for sensor bias estimation. Our final approach is based on IMU-OF integration and has proven reliable in the experiments with our platform, i.a., in those shown later in Chap. 4. Additionally, we have extended our algorithm to position estimation which allowed us to take part in the EuRoC competition. The EuRoC simulator and datasets allowed us to also demonstrate how SIL simulations can be used in the process of software development and debugging.

Chapter 3

Obstacle Detection and Tracking

3.1 Introduction

A variety of sensors can give a robot the ability to sense its surroundings. From the most basic mechanical bumpers and non-contact, close-proximity distance sensors, through optical infrared or sonar rangefinders to laser scanners and depth cameras, the robot gains the ability to sense the environment. Analysis of sensor measurements allows the robot to take actions depending on its task and situation, e.g., the detection of obstacles around the robot enables further actions such as path planning or obstacle avoidance. Given the limited sensing range and field of view of many sensors, obstacle tracking provides the means to improve and maintain knowledge about the state of obstacles despite the motion of the platform or the obstacles themselves.

In UAV teleoperation, especially in an unknown, obstacle-rich environment, obstacle detection and tracking is of great importance as a part of the assisting algorithms for the operator, enabling functionalities such as collision-free navigation. Thus, it can facilitate the task of the human operator and improve efficiency of the teleoperation process.

3.1.1 Literature Overview

Although tracking is inseparably linked with detection, there are systems that forgo tracking, and, for the purpose of collision-free navigation, use reactive algorithms based only on the current state of detected obstacles. As a reference for this thesis we analyze different classes of approaches in order to make a thorough overview of existing algorithms. For clarity, we divide this section into two parts, dedicated to detection and tracking approaches, respectively.

Detection

The most basic approach is the use of distance sensors emulating the behavior of mechanical bumpers that limit motion in the sensing directions (Nieuwenhuisen *et al.*, 2013; Houskamp, 1978). Gageik *et al.* (2015) proposed an array of complementary

ultrasonic and infrared distance sensors to enhance their system's robustness. Although it does not require complicated processing of measurements, this approach has certain limitations. The obstacle detection, and hence avoidance, is limited to the area covered by the sensors, thus, require many dedicated sensors to achieve effective results. This in turn, can be not feasible for small robotics platforms as it can add excess weight.

Instead of using multiple one-directional sensors, Ferrick *et al.* (2012) used a 360° laser range finder to create a 2D occupancy grid map to determine obstacle-free paths for their UAV. A natural 3D extension can also be obtained by rotating the sensor via servomotors as shown by Nieuwenhuisen *et al.* (2013). The laser range finders, especially with the additional actuators, can again be too heavy or bulky for small size UAVs.

For the purpose of obstacle avoidance, a popular approach is to use monocular cameras, for example in algorithms based on optical flow (Merrell *et al.*, 2004), or by determining the size expansion ratios of obstacles from feature tracking in the field of view of the sensor (Al-Kaff *et al.*, 2016). The perspective from a monocular camera, however, does not provide any correlation between the detected features other than the relative direction. This in turn makes building a 3D model of the environment and obstacle tracking cumbersome. On the other hand, stereo cameras enable generation of disparity images and the distance estimation based on the epipolar geometry, which consequently enables mapping of the observed obstacles from the image plane to the robot/camera frame (Ait-Jellal and Zell, 2015).

Tracking

In order for a robot to be able to process information about its environment, it is first necessary to properly model it and store that information in the memory. Different ways of modeling have certain advantages and disadvantages and differently influence processing efficiency. Irrespective of the actual implementation, a representation of the robot's surroundings contains virtual objects representing their counterparts in the real world together with the encoded information about their position, size, etc.

Depending on the reference frame, such a representation can be classified either as global or local, i.e., expressed with respect to a global reference frame (extrinsic representation) or in relation to the robot (intrinsic representation). While the former approach is typically related to the localization problem, or broadly speaking simultaneous localization and mapping problem (SLAM), the later has certain advantages, especially in tasks like teleoperation, as it ensures fixed memory size and bounded processing time (Odelga *et al.*, 2016b).

Regardless of the reference frame, the information about the environment can be mapped in different ways. Information about objects can be stored as sets of points (point clouds), lines, or polygons, as continuous data containing their

positions, dimensions, colors, etc. The representation of laser sensor readings and stereo or depth cameras as point clouds has been used in several SLAM algorithms, for instance by Cole and Newman (2006) or Nüchter *et al.* (2007).

The second representation in this category is the discrete mapping, in which space is modeled as a grid of cells of equal size, i.e., volume elements - voxels. Objects are mapped into these grids such that each cell contains information about its occupancy. This category introduces an abstraction layer between the object and its representation such that the data (mainly position) is associated with the grid, not the object (Roth-Tabak and Jain, 1989; Moravec, 1996).

One of the most notable examples of this method is the work by Hornung *et al.* (2013). The authors propose an efficient way to store large-scale environments as 3D voxel grids using an *octree* representation. The cells in a map of predefined size are recursively divided into eight smaller cells - nodes. The main advantage of this representation is that whenever all the children of a node contain the same information they can be *pruned*, hence, it enables dynamic resolutions. For example, a large, empty (or occupied) space can be fully described with a few big cells as finer resolution does not provide more information. The main drawback, however, is that accessing specific regions of the map requires iteration along the tree, while in maps of fixed resolution, the voxels can be queried directly by their position.

The cells occupancy information can be also represented discretely, with deterministic binary (empty/full) information, or in a stochastic manner, where cells contain a given probability of being occupied, which enables storing uncertainty related to the measurement and the representation. On the other hand, continuous data contains inherently more information about objects, especially their boundaries, thus it is seldom represented in a stochastic way. Nevertheless, statistical analysis is still widely used with such data, especially in registration of different measurements to a common reference frame (Agamennoni *et al.*, 2016).

3.1.2 Problem Statement

In most of the aforementioned works, the limiting factor is the on-board computational power. The transition from sensor arrays to frame-to-frame image analysis and occupancy grids was possible not only thanks to the development of more advanced algorithms, but also because of to the recent progress in mobile CPUs.

Obstacle detection and tracking is a computationally expensive task due to the amount of data that needs to be processed, especially given the availability of lightweight sensors that have extended the data gathering abillities of UAVs with limited payload. Since we are performing all the computation on-board, in our development we have given particular attention to the computational efficiency of our method.

In this doctoral thesis, obstacle avoidance in the context of UAV teleoperation is the main thread. Therefore, in previous chapters we defined the platform, a quadro-

tor unmanned aerial vehicle, and the particularity of teleoperation with respect to other control modes, Sec. 1.2.4. While obstacle avoidance is a broad subject with many application and different approaches, the context of teleoperation implies certain constraints and assumptions.

In order to avoid any obstacle, the obstacles must be reliably detected first, then saved into the memory to preserve information about their state (namely their size and position). Additionally, given the limited number of sensors that the platform can carry and their range of detection, a tracking algorithm can be employed to maintain the state of detected obstacles knowing an estimate of the robot's motion. Finally, the obstacle's possible interference with the desired trajectory of the robot can be analyzed and prevented.

3.1.3 Methodology

As obstacle detection (with tracking) and avoidance can be partially analyzed independently, we dedicate this chapter to the investigation of the former problem. Motivated by the work by Stegagno *et al.* (2014), we decided to develop an obstacle detection and tracking algorithm based on a single RGB-D camera as this sensor provides a relatively large amount of information with a large field of view compared to its size.

Constrained by the limited computational power of the on-board computer, we decided to adopt a local and bounded obstacle system, i.e., one represented in a frame that moves with the robot. Therefore, in order to preserve the relative position of the obstacles, this local state has to be updated given the motion of the robot.

In Sec. 3.2 we summarize the bin-occupancy filter, a multitarget tracking algorithm that provides theoretical formulations for our problem.

In Sec. 3.3 we detail our implementation using the bin-occupancy filter. We start from a comparison of different coordinate frames and their pros and cons in terms of being used to represent the obstacle state. Next, we fully characterize our *surveillance* region within which we will detect and track obstacles. Finally, we detail the state and measurement updates of our implementation, i.e., how we update the state with depth measurements and how we transform it given the estimate of the robot's motion. We summarize and conclude this chapter in Sec. 3.4.

3.2 Bin-Occupancy Filter

The bin-occupancy filter is a multitarget tracking (MTT) algorithm introduced by Erdinc *et al.* (2009). It approaches the problem of target tracking by investigating if there is a target at a given point in space. It models the robot's surroundings as

a surveillance region S partitioned into *bins*, which may or may not be occupied by a target. Hence, it is an opposite approach to most MTT algorithms, which instead estimate the number of targets and simultaneously track them. In the case of obstacles that might collide with the platform, our interest is to define free and occupied regions in the vicinity of the robot rather than to know the exact number of potential obstacles. Therefore, the use of the first approach is more reasonable.

The algorithm has the structure of a recursive filter whose state is the probability of occupancy of each bin in the region of interest S. The bins in the surveillance region S are assumed to be sufficiently small such that each of them is potentially occupied by at most one target. If two targets (regardless of their dimension) happen to be located close together, the bin volume can be reduced so that this assumption remains valid. Moreover, one target should give raise to only one measurement, and should generate it independently of other targets.

The algorithm comprises of two phases that we summarize in this section: prediction and correction, also referred to as the state and measurement updates.

3.2.1 Prediction

The prediction step of the bin-occupancy filter takes the knowledge of the state of the bins at time $k-1$ and makes an *a priori* estimate based on the probabilities of two events. Given two generic bins i and j, at a given instant k the event *bin i contains a target* is possible if:

a) a new target appears in bin i,

b) a target from bin j moves to bin i.

These events are mutually exclusive - two targets cannot occupy the same bin considering the aforementioned assumptions. The second case also includes the case when $j = i$, i.e., the target in bin i stays in this bin. We also call this step the state update, as we will show later that these events correspond to the motion of the robot.

The state update equation computes the probability of bin i being occupied at time k, given the set of measurements from time 1 to $k-1$

$$\begin{aligned} p_{k|k-1}(U_k(i)) &= b(i) \\ &\quad + \sum_j p(i|x_j) P_s(x_j) p_{k-1|k-1}(U_{k-1}(j)) \,, \end{aligned} \tag{3.1}$$

where $U_k(i) \in \{0, 1\}$ is a random variable which is 1 if bin i contains a target at time k and 0 otherwise, $b(i)$ is the probability that a new target appears in bin i, x_j is the middle point of bin j, $P_s(x_j)$ is the probability of survival of a target located at x_j to the next time step, $p(i|x_j)$ is the probability that a target located at x_j

moves to bin i, and $p_{k-1|k-1}(U_{k-1}(j))$ is the probability of bin i being occupied at time $k-1$. Equation (3.1) describes the sum of probabilities of the two independent events a) and b).

3.2.2 Correction

The correction phase of the algorithm is based on sensor measurement. Whenever new information is available the state of the observed bins can be updated. In the bin-occupancy filter method, a generalized sensor measurement is modeled as z_s, $s \in \{1, \ldots, m\}$, where m is the number of detected targets. At instant k, and the measurement update of bin i is defined as

$$p_{k|k}(U_k(i)) = p_{k|k-1}(U_k(i)) \Bigg[(1 - P_d(x_i)) \tag{3.2a}$$

$$+ \sum_{s=1}^{m} \frac{P_d(x_i) f(z_s|x_i)}{\sum_j P_d(x_j) f(z_s|x_j) p_{k|k-1}(U_k(j))} \Bigg], \tag{3.2b}$$

which updates the probability from Eq. (3.1) with the measurement at instant k. The other components of Eq. (3.2) are:

- $P_d(x_i)$ is a *visibility factor* describing the probability of detection of a target located at x_i independent from the sensor measurement,
- $f(z_s|x_i)$ is the sensor's probability density function (PDF) of a measurement z_s given a target at x_i.

In Erdinc *et al.* (2009), Eq. (3.2) was derived using Bayes' formula using the *a priori* estimate $p_{k|k-1}(U_k(i))$, current measurement z_s, $s \in \{1, \ldots, m\}$ and probabilities related to the visibility of the target $P_d(x_i)$. The second part, Eq. (3.2b) corresponds to the influence of measurement $s \in \{1, \ldots, m\}$ on the updated bin i, weighted by the PDF function $f(z_s|x_i)$, which depends on the measurement location. This PDF represents the sensor noise, and thus allows the introduction of false positive and false negative measurements.

3.3 Implementation

Although the bin-occupancy filter inherently imposes a probabilistic representation with the surveillance region partitioned into discrete cells, it does not specify other aspects of the representation. The choice of the size of the bins or even their shape is implementation specific, and at the user's discretion as long as it fulfills the filter's assumptions. The general prediction and correction equations, shown in Sec. 3.2, must also be particularized given the implementation specifics.

3.3.1 Depth Measurement Model and Calibration

The pinhole camera model presented in Sec. 2.2.4 describes only the perspective geometry of a point. Thus, by inverting the model in Eq. (2.17) we can only obtain (x_c, y_c) coordinates in relation to the distance z_C. The depth measurement, i.e., the z_C value, complements the pinhole camera model allowing a full 3D position mapping from the image plane to the camera and, consequently, the world (or robot) frame.

In a general form the depth measurement can be modeled as

$$\tilde{z}_{uv} = f(u, v, z_C), \tag{3.3}$$

where $\tilde{z}_{uv}$ is the measured distance to an object (pixel) with (u, v) coordinates in the image plane, and f is an unknown sensor model. Assuming that the image distortions can be sufficiently well compensated for with the previous distortion models, Eq. (2.21), we use the following quadratic depth sensor model

$$\tilde{z}_{uv} = \alpha_{d_2} z_C^2 + \alpha_{d_1} z_C + \delta_d \tag{3.4}$$

where δ_d is an unknown offset, and α_{d_2} and α_{d_1} are unknown scaling coefficients.

The calibration algorithm principle is similar to the image calibration presented in Sec. 2.2.4. Knowing the true distance to the object, the coefficients of Eq. (3.4) are obtained using the least squares method with a sufficient number of measurements.

The actual distance can be estimated in the same fashion, by using an object with a known, easily detectable pattern. For example, a checkerboard, as shown in Fig. 2.4c can be easily detected using corner detection. Known dimensions of the squares in the checkerboard enable estimation of the distance.

Remark 3.1 *In order to calibrate the depth sensor one has to use the corresponding images from the IR sensor as the depth sensor does not allow recognition of the calibration patter. Nevertheless, the calibration is possible assuming proper lighting conditions and access to the IR images used for the depth estimation.*

Point Cloud

Each pair of color and depth measurements can be mapped to the world (or camera) frame into a point with 3D coordinates and color information. The set of resulting points representing the surface of objects in the field of view is often reffered to as a point cloud. Even though we have not used the color information in the presented obstacle detection and tracking algorithm, for the sake of completeness, it is worth to mention that assigning color to the depth points requires another calibration process which can improve the factory setting of the sensor. The color and the depth sensor create a stereo pair that needs to be registered to a common reference

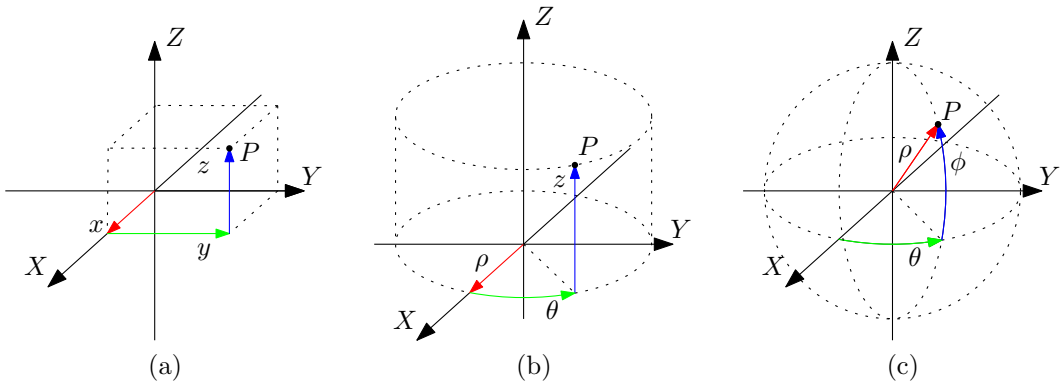


Figure 3.1: The three main coordinate systems with coordinates of an arbitrary point.

frame, such that pixels of the same image coordinate match each other in both of the images.

The stereo camera calibration procedure again requires a calibration pattern and a set of corresponding images from both sensors. This process allows estimation of the transformation matrix from one camera to the other, and registration of both of the images.

3.3.2 Coordinate System

As explained earlier in this chapter, the map or representation of the robot's surroundings can be expressed either with respect to a global frame or in relation to the robot in its local frame. In the global representation both the position of the robot and obstacles are defined with respect to a common, stationary frame of reference. A Cartesian coordinate system seems a natural choice for the global reference frame as it directly corresponds to (x, y, z) linear translations - natural movement for humans in our 3D world. In Fig. 3.1 we summarize the three main coordinate systems with the position of an arbitrary point P.

In local representations where the position of every obstacle is defined with respect to the robot, other representations such as cylindrical or spherical coordinate systems, have certain advantages. For instance, the radial coordinate of either of these systems directly represents the distance to an object, which can be beneficial when determining the range of possible collision-free motion. Moreover, local representations can have a fixed size, i.e., we only preserve information about objects within a certain range. This ensures constant memory demand and bounded processing time of such a representation while also not restricting the range of the robot's motion.

The main drawback of global representations is their large memory requirements

as sufficient computational power and memory has to be available to store and process the mapped obstacles. Especially in large scale environments this is usually not feasible with the limited power of on-board computers. Therefore, in this work about the teloperation of UAVs, we decided to employ a local representation of the robot's surroundings for the purpose of its navigation. In this section we analyze the usefulness of the main coordinate systems in the context of obstacle representation for our task. The summary of our findings is shown in Table 3.1.

Partitioning into Bins

The bin-occupancy filter assumes the specification of a surveillance region S partitioned into bins (cells). Target detection and tracking is performed only over this predefined region, i.e., even when a target is detected outside of this region it is disregarded. Hence, at every instant k, as we only take into consideration obstacles within S, the size of this region defines the instantaneous possible range of motion.

Furthermore, we assume that the cells have a constant size and that the surveillance region is defined as the number of cells along every axis. This ensures a three-dimensional array structure of S with a uniform distribution of cells. Hence, similarly to the voxel representation, the position of every cell is linearly dependent on its number (or coordinate) and does not have to be explicitly stored. The reverse operation is also of low complexity. Cells associated with an arbitrary location (or region) within S can be easily determined.

The shape of the surveillance region will differ in every coordinate system: cuboid in the Cartesian system, cylinder and sphere in the cylindrical and spherical systems, respectively.

Transformation Complexity

	Cartesian	Cylindrical	Spherical
x/y transformation complexity	low	moderate	high
z transformation complexity	low	low	high
ψ transformation complexity	moderate	low	low
sensor model matching	poor	partially good	good
space complexity	high	low	high

Table 3.1: Comparison of the three main coordinate systems as local frames for obstacle representation.

Given the inherent properties of local representations, the information about objects stored in memory must be updated to accommodate for the robot's motion. Different coordinate systems involve different levels of transformation complexity - the mapping between the motion of the platform and the coordinates of a given system. For example, the rotation around the platform's vertical axis, i.e., yaw rotation of ψ angle about z_H (see Fig. 2.1 for reference), directly corresponds to the azimuthal coordinate of both cylindrical and spherical systems. Thus, in order to update the position of mapped obstacles, the value of $\Delta\psi$ (total rotation) can be used directly.

On the other hand, the same transformation in a Cartesian system requires additional computation of the corresponding Δx and Δy values given the rotation of $\Delta\psi$. Additionally, as the corresponding linear translation is proportional to the radial distance from the system's origin in the yaw rotation, the $\{\Delta x,\ \Delta y\}$ value is different depending on the object's (x, y) location. The transformation values do not depend, however, on the cell's z coordinate. Thus, cells of the same $\{x, y\}$ coordinate are transformed by the same $\{\Delta x,\ \Delta y\}$ value in every *layer* of the cylindrical system given a pure ψ rotation.

Similar complexity will affect the x/y transformation in the cylindrical system. For a given $\{\Delta x,\ \Delta y\}$ translation of the platform, a corresponding $\{\Delta\rho,\ \Delta\varphi\}$ value has to be calculated. Although this value differs for different $\{\rho,\ \psi\}$ coordinates, it is again the same for every *layer*, i.e., z coordinate.

The spherical system has the highest transformation complexity, except the yaw rotation for every translation of the platform, either lateral or horizontal, a full set of $\{\Delta\rho, \Delta\theta, \Delta\phi\}$ translations in the spherical system has to be computed.

Sensor Matching

The depth camera that we use to detect obstacles can be described with the pinhole camera model (Sec. 2.2.4). Although it uses Cartesian coordinates, the sensor readings can be easily transformed to any other system. The probability of detection, however, depends on the relative size of the bins. As the objects farther from the camera appear smaller, they have also a lower probability of detection assuming a constant cell size and volume in a Cartesian system.

On the other hand, a spherical system ensures a constant relative size of its bins, as their metric volume grows together with the radial distance, and their dimensions along $\{\theta, \phi\}$ (azimuth and polar angles) directions are constant after projection on the image frame.

Space Complexity

With the space complexity we mean the general efficiency of a system in terms of the number of cells needed to cover a given range. The Cartesian representation requires

the highest number of cells for the same volume. As the metric volume of cells in both the cylindrical and spherical representations grows with radial distance, they require fewer cells for the same volume. Hence, the Cartesian system has higher space complexity as more cells have to be updated in every iteration of the filter.

Although the spherical system covers more space with the same number of cells, (assuming comparable metric volume of cells) the surveillance region must be larger to enable the same range of motion. Considering the motion of a quadrotor, especially in the context of teleoperation, the platform mainly moves forward, rotates and changes its altitude. Hence, a cylindrical surveillance region offers the highest (instantaneous) range of motion with a lower cell number than other coordinate systems.

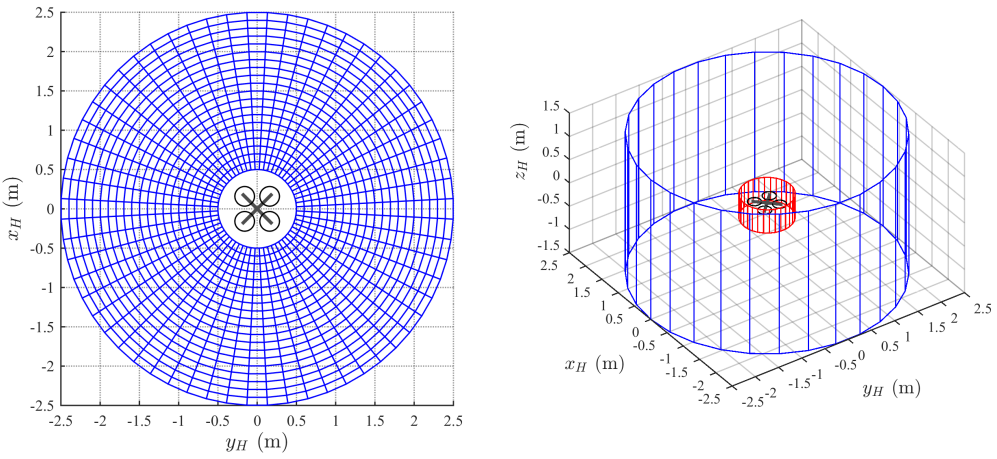
Conclusion

Considering the above discussion, summarized in Table 3.1, we decided to employ a cylindrical coordinate system as it provides the best performance in our situation. Given the limited computational power related to on-board computation capabilities, the average low complexity of the operations on the cylindrical system should ensure the feasibility of this approach. Additionally, the parameters of such a system can be further tuned to accommodate for limited computational resources. For example, the size and resolution of the obstacle representation can be changed to optimize the complexity of the operations on this system.

3.3.3 Robot-Centric Obstacle State

The bin-occupancy algorithm performs tracking and assumes target detection in a quantized surveillance region S. As we are interested in tracking obstacles in the near vicinity of the UAV, we define S as a bounded region in a local cylindrical coordinate frame $M : \{O_M, P_M, \Psi_M, Z_M\}$, wherein obstacles will be tracked. The P_M and Ψ_M coordinates are, respectively: the radial and the azimuth distances, and a generic point is expressed in M with (ρ, ψ, z) coordinates. The frame is defined with respect to the horizontal frame H such that $O_M \equiv O_H$ and $Z_M \equiv Z_H$. The azimuth reference plane, $\Psi_M = 0$, is defined at an angle to the $X_H Z_H$ plane, such that it intersects with the camera optical center. The surveillance region boundaries are defined as $(\rho_{S_{min}}, \rho_{S_{max}})$ and $(z_{S_{min}}, z_{S_{max}})$ on the P_M and Z_M axes, respectively.

The surveillance region S is defined as a set of bins b_i, whose size in M is $\Delta\rho_b \times \Delta\psi_b \times \Delta z_b$, and the obstacle state is represented as the probability of bins $b_i \in S$ being occupied. In Fig. 3.2a an example *layer* of S and its partition into bins are shown. The inner circle, clear from partitioning for clarity, represents the restricted area S_R, which is considered occupied by the quadrotor, hence no obstacle should get inside S_R. In Fig. 3.2b, we show a 3D view of S and S_R with a



(a) (b)

Figure 3.2: Surveillance region: (a) top view of the surveillance region divided into partitions, (b) in blue a 3D view of the surveillance region; in red the restricted area.

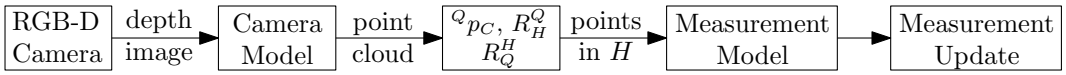


Figure 3.3: Diagram of the measurement update of the Bin-Occupancy filter.

simplified contour of the quadrotor and its relative size.

The dimension of bins ($\Delta\rho_b \times \Delta\psi_b \times \Delta z_b$) is constant in cylindrical coordinates, but considering the non-Euclidean property of this coordinate system, the absolute volume of cells differ, i.e., their volume grows proportionally to the radial distance from the center. The bin-occupancy filter assumes the same volume for all bins, or rather, the same probability of target detection. Considering the model of an ideal pinhole camera, each pixel in a depth frame from the sensor represents the distance to an area that grows with the distance from the sensor. Although the ideal would be a spherical representation, our cylindrical surveillance region ensures more comparable probability of detection than a Cartesian system, when we take into account the simultaneous change in the volume of the cells and the pixels relative size.

3.3.4 Measurement Updates

Obstacle state corrections (Sec. 3.2.2) have the form of measurement updates. Whenever a new depth frame from the camera is available, a measurement update can be performed. First, the image from the camera is scaled down such that the new pixel dimension is adjusted to the dimensions of the bins. This process

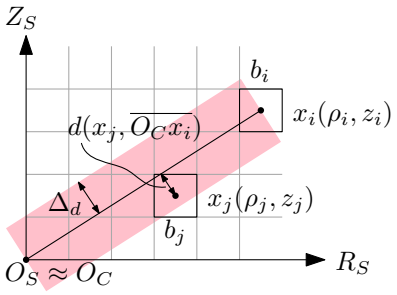


Figure 3.4: Illustration of the visibility factor computation.

significantly reduces the amount of data that must be processed while ensuring that every pixel corresponds to at most one bin. The scaled depth image is then transformed into a point-cloud using the camera's intrinsic parameters. Using the model described in Sec. 3.3.1, the location of every depth point (pixel) with a non-zero depth value in the horizontal frame is calculated. The points are expressed in the frames Q and H using the camera's extrinsic parameters (i.e. ${}^{Q}\mathbf{p}_C$ and $\mathbf{R}_C^Q$) and the roll and pitch angles. Then, using the standard Cartesian-cylindrical transformation, the corresponding cylindrical coordinates are calculated forming a set of obstacle points in M. The set of obstacle points is then mapped into the bin-space representation, i.e., each point is assigned to the bin that corresponds to its position. A scheme of the whole process is presented in Fig. 3.3.

Visibility calculation

The measurement update equation of the bin-occupancy filter given in Eq. (3.2) requires the computation (or definition) of the visibility factor $P_d(x_i)$ for every i-th bin with its center at x_i. For every bin with a positive target detection this parameter is assumed to be $P_d(x_i) = 1$, as the detection of the corresponding obstacle point ensures its visibility.

All the other bins, without positive target detections, go through a two-step process to determine their $P_d(x_i)$ factor. In the first step we perform an inverse procedure with respect to the previously described one. For every bin we calculate its corresponding pixel coordinates (u, v) in order to check if that bin is in the field of view of the camera. If the bin is outside the field of view $P_d(x_i) = 0$, otherwise we compute the occlusion rate based on the occupancy of other cells between the investigated one and the camera.

Hence, in the second step we compute the visibility factor based on the occlusion rate of bins with positive detections along the line segment $\overline{O_C x_i}$. As illustrated in Fig. 3.4, the occlusion rate is defined based on the distance $d(x_j, \overline{O_C x_i})$ between

the potentially occluding bin j (with its center at x_j) and $\overline{O_C x_i}$, as

$$\gamma_i(j) := \begin{cases} \frac{d(x_j, \overline{O_C x_i})}{\Delta_d} & \text{if } d(x_j, \overline{O_C x_i}) \leq \Delta_d \\ 0 & \text{if } d(x_j, \overline{O_C x_i}) > \Delta_d \end{cases}, \tag{3.5}$$

$$\Gamma_i = \{\gamma_i(j) \mid \gamma_i(j) > 0\}, \tag{3.6}$$

where Δ_d is a threshold value and Γ_i is the set of all bins j that occlude bin i with their factors $\gamma_i(j)$. The visibility factor $P_d(x_i)$, for bins without a positive target detection, is defined as

$$P_d(x_i) = V_i \prod_j \gamma_i(j) \left(1 - p_{k|k-1}(U_k(i))\right), \tag{3.7}$$

where V_i is the visibility indicator for cell i: 1 for cells inside the camera's FOV, 0 otherwise.

As empty space is detected by the camera indirectly, the visibility factor allows the algorithm to update the cells that are occupied in the state but where there is no corresponding positive detection in the new measurement. The probability of occupancy of such cells is consequently decreased as the lack of a positive measurement suggests that the corresponding cell is now empty.

3.3.5 State Updates

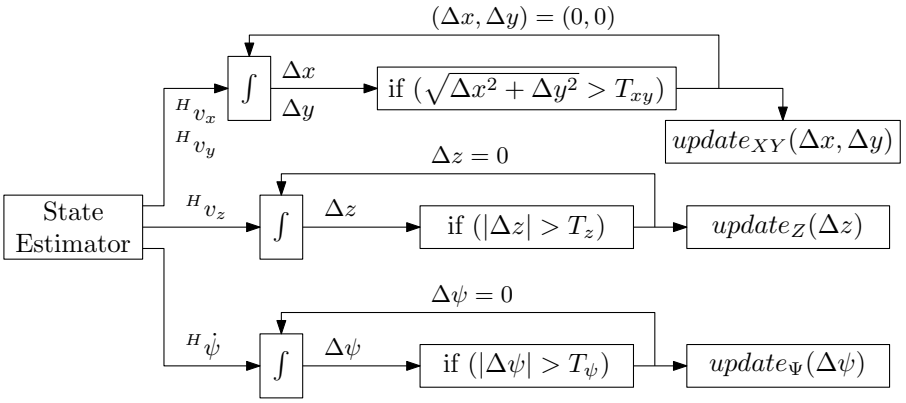


Figure 3.5: Block diagram of the state update logic.

The obstacle state is local and bounded, meaning that whenever the robot moves the obstacles stored in memory should be moved as well in order to maintain their relative location with respect to the real world objects. This also allows us to extend the knowledge of the obstacles to the region of S outside the field of view of the camera. The predefined, limited size of S with a constant number of bins

ensures finite computational time and memory demand. Because of the probabilistic representation, where each cell in the grid contains not a binary value but a probability of being occupied, the uncertainty related to the estimate of the robot's motion should also be reflected in the obstacle state. Hence, when obstacles are no longer in the field of view of the sensor, the certainty related to their position decreases over time with consecutive motion updates.

On the other hand, because of the discrete character of the representation, the obstacle state is partitioned into fixed size bins, each cell, besides of its position and size, contains only one piece of information - the probability of being occupied. Therefore, in order to represent a loss of certainty, this probability is spread to adjacent cells with consecutive updates.

The prediction stage of our algorithm is performed based on the estimated state of the robot. The obstacles in S are updated using three independent transformations:

- a translation along the Z_M axis, given the UAV velocity ${}^{H}v_z$,
- a rotation around the Z_M axis, given the UAV yaw rate ${}^{H}\dot{\psi}$,
- a translation on the $P_M\Psi_M$ plane, given the UAV velocities ${}^{H}v_x$ and ${}^{H}v_y$.

In order to keep the computational time bounded, and limit the spread of the probabilities in S due to the quantization noise, these three transformations are not performed every time that a new velocity estimate is available. Instead, the velocities are integrated and a transformation is performed only when the corresponding accumulated value reaches the corresponding threshold. This process is illustrated in Fig. 3.5, where T_i, $i = \{xy, z, \psi\}$ is the threshold value for the respective updates. The actual implementation of the update functions is shown in Appendix B.

Similarly, obstacles that leave the surveillance region are erased from the memory. This also keeps the necessary memory bounded and independent from the duration of experiments without limiting the area of exploration. Please note that to compute Eq. (3.1) the probability of survival $P_s(i)$ and the probability of the birth of new targets $b(i)$ are needed. These parameters depend on the environment and should be set such that a boundary zones should have a relatively high $b(i)$, while a value of $P_s(i) < 1$ can be used to reduce artifacts.

For the first two updates listed above, i.e., the translation and rotation around the Z_M axis, the uncertainty spread can be controlled with the corresponding threshold value. As in the example shown in Fig. 3.6, in these two updates the probability of the occupancy of a cell will always be transferred towards a directly adjacent bin, along the Z_M and Ψ_M direction, respectively.

In the example of probability of occupancy evolution over time in Fig. 3.6, the threshold value was set to half the cell size $1/2\Delta x$. The first iteration of this example contains high probability of occupancy in the middle cell after a previous measurement. The state is then subsequently updated six times, three times in the positive

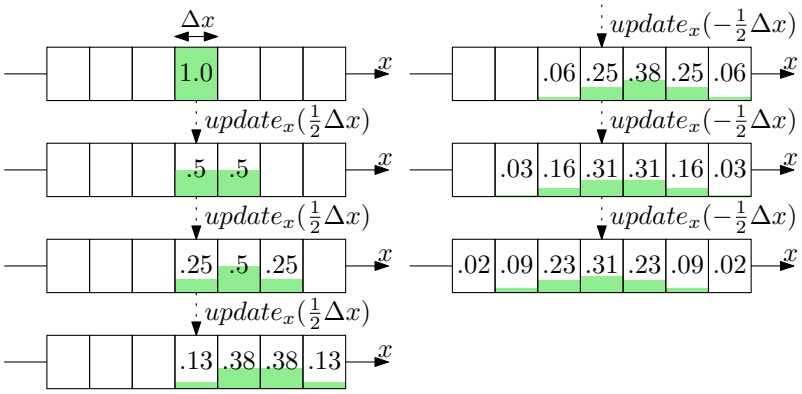


Figure 3.6: An example of the state update on a simple, one dimensional, obstacle state.

direction and then three times in the negative direction. Thus, for every cell, at each step half of the probability remains in that cell and half moves to the adjacent cell. As can be seen from this example, this approach preserves the mean value of the obstacle's position, after the last update the highest probability is again in the middle cell. The variance of the obstacle's position, however, increases with every update as there is no new measurement.

The obstacle state update on the $P_M\Psi_M$ plane, as these directions are coupled, is more complicated. As explained earlier during the transformation complexity analysis in different coordinate systems, $\{\Delta\rho, \Delta\varphi\}$ translations depend not only on the corresponding $\{\Delta x, \Delta y\}$ motion of the platform but also on the cell's (ρ, φ) position. Fig. 3.7 shows an example of such a transformation. A $\{\Delta\rho, \Delta\varphi\}$ translation of one cell is shown based on the corresponding $\{\Delta x, \Delta y\}$ pair. As these two directions can not be updated independently we also can not directly control the occupancy spread and therefore the spread of related uncertainty.

The example in Fig. 3.7 also depicts a limitation of our approach, the rate of the uncertainty growth on the $P_M\Psi_M$ plane can not be controlled independently, i.e., it depends indirectly on the number of updates and the update threshold size. A higher threshold size can limit the spread of uncertainty by limiting the number of updates.

3.4 Summary

In this chapter we presented our algorithm for a local bin-occupancy system to store information about detected obstacles. Implementation of the bin-occupancy filter allowed us not only to store new measurements, but to also update the existing state

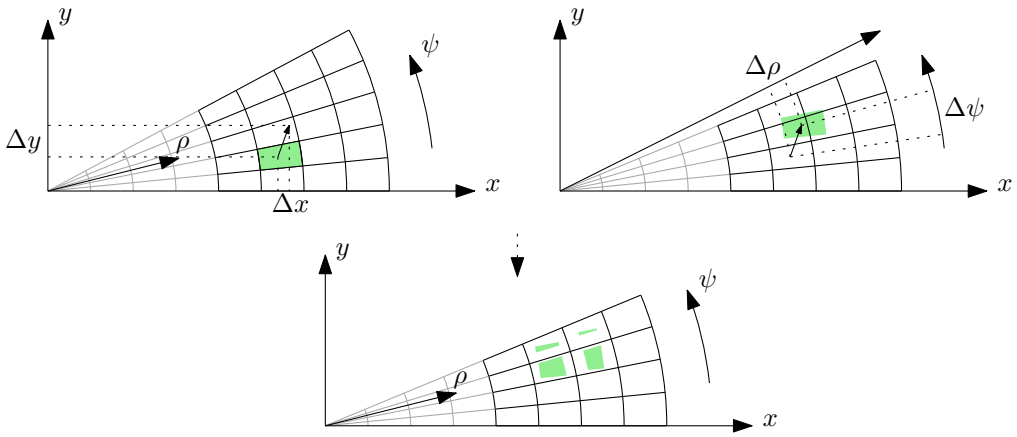


Figure 3.7: Example of a state update on the $P_M\Psi_M$ plane.

with new information and estimate the location of previously seen obstacles with the estimated motion of the robot. The use of a cylindrical system increased the efficiency of our approach with respect to other representations as was shown in the complexity analysis. A bounded surveillance region partitioned into a fixed number of bins ensures finite computational time and memory demand, which is especially important when dealing with the limited resources of on-board computation.

As the main purpose of obstacle detection and tracking is the following use of this algorithm in obstacle avoidance, the next chapter will show further advantages of our system with respect to a teleoperated UAV.

Chapter 4

Obstacle Avoidance

4.1 Introduction

Unconstrained by ground conditions, flying robots are desired in applications such as exploration, inspection, and surveillance, as they can reach places that ground robots can not. However, the operation of UAVs in an unstructured, obstacle rich environment with limited space to maneuver is a challenging task. As we explained in Chap. 2, state estimation is essential in order to ensure drift-free control and enable further algorithms for collision-free navigation. In order to further assist the operator and increase the number of potential applications of this technology, the robot needs to be able to properly process the user's input and correct for errors resulting from the limited situational awareness of the operator.

Obstacle avoidance, enabled by environment sensing, detection and tracking of potential obstacles, can greatly enhance the teleoperation performance of UAVs by filtering the user's commands to prevent collisions. Additionally, by improving safety, obstacle avoidance measures can increase the operator's confidence and allow for more effective task execution. Although human supervision is often required due to legal and safety reasons, operating a UAV can be greatly simplified by endowing the UAV with autonomy (Odelga *et al.*, 2016b; Grabe *et al.*, 2013). Autonomous obstacle avoidance, especially when operating in tight and unstructured GPS-denied environments, can enhance visual inspection or search and rescue missions. Thus, allowing for operation closer to the objects of interest, and in more cluttered environments, leaving the operator free to focus on higher level goals (Nieuwenhuisen *et al.*, 2013; Bonnin-Pascual *et al.*, 2015; Mebarki *et al.*, 2015).

4.1.1 Literature Overview

Depending on the task's objective and the amount of a priori knowledge of the system, we can distinguish two main formulations of the collision-free navigation problem, path-planning and obstacle avoidance. In path-planning the focus is usually on the optimization of a robot's trajectory given the desired position that the robot should reach (Ferguson *et al.*, 2005). Assuming a high amount of information

about the environment, including obstacle sizes and positions, these algorithms aim to minimize the path length or optimize energy efficiency (Mellinger and Kumar, 2011) while ensuring no collision and obstacle avoidance. Teleoperation applications, however, consider mostly unknown and/or non-static environments. Moreover, when a UAV has only a generic goal (e.g., exploration of a given space), a path planning approach becomes ineffective. In such scenarios, a reactive control law is usually implemented that adjusts the system's behavior given exteroceptive measurements (Borenstein and Koren, 1991; Becker *et al.*, 2006; Zohaib *et al.*, 2013).

Some previous works have dealt with autonomous obstacle avoidance for teleoperated UAVs. One of the most basic approaches is the potential field (Ge and Cui, 2002), in which directional distance sensors are used and a repulsive force is generated whenever a sensor's measurement is below a certain threshold (Nieuwenhuisen *et al.*, 2013). Hua and Rifa (2010) and later Omari *et al.* (2014) proposed a solution using measurements from laser scanners and other range finders. Obstacles are represented as surfaces, and the commanded velocity of the robot is modified based on the robot's distance to these surfaces. In the work by Gageik *et al.* (2015), the authors propose an array of complementary ultrasonic and infrared distance sensors to enhance their system's robustness.

Although UAVs operate in 3D environments, some works simplify the case to a plane. Ferrick *et al.* (2012) developed a method based on a 360° laser range finder to create *an image* of the walls around the robot at its altitude. Using standard image processing techniques they extract the boundaries of the obstacles and implement the obstacle avoidance using the concept of wall-following.

Another popular approach is the use of optical flow generated from a monocular camera. Although in these methods the distance information can not be directly extracted, they provide a relative measure of the obstacle's proximity and enable avoidance actions (Merrell *et al.*, 2004; Serres *et al.*, 2006; Mahony *et al.*, 2009). Al-Kaff *et al.* (2016) performed obstacle avoidance by determining the size expansion ratios of objects in the field of view of the sensor using a feature tracking-based optical flow.

Oleynikova *et al.* (2015) presented an algorithm for a reactive obstacle avoidance based on stereo camera images. Reconstruction of the obstacles in the field of view of the sensor is based on epipolar geometry and the resulting disparity images. The avoidance algorithm then performs path-planning that relies on a short-term map. It does not require accurate odometry and does not keep a globally consistent map over the whole experiment. The obstacle map is created from a U-map, an accumulated histogram along the columns of the disparity images and the obstacles are modeled as ellipses. The 3D disparity information is reduced to a 2D representation, where every data point contains a sum of occupancies from the corresponding column. Therefore, although the method uses 3D stereo vision, the avoidance algorithm, called in the work *a local path-planning*, is limited to the

planar case.

More recent works take advantage of the smaller size and higher performance of miniature computers by employing more sophisticated methods not only for map building but also for obstacle avoidance. In the work by Xu *et al.* (2015), an octree-based 3D path planning algorithm using a state lattice concept is employed to find an optimal trajectory. The path-planning problem is reduced to the graph search problem. By using the octree concept for both the map and the state lattice, the memory requirement is kept to a minimum and the authors claim that the graph search is made more efficient. Gohl *et al.* (2015) equipped their platform with four stereo cameras allowing the robot to have an omnidirectional view of the environment. Detected obstacles are stored as points in a local, spherical coordinate frame that is transformed as the robot moves with the estimate of its state. Although their platform can see in all directions the avoidance algorithm is reduced to determining the possible range of motion in different directions (bearings).

4.1.2 Problem Statement

One of the main goals of this work is to develop an obstacle avoidance algorithm for a teleoperated multirotor UAV. Our intention is that the resulting algorithm should allow an operator to pilot a robot relying only on visual and haptic feedback without fearing of possible collisions. As one of the main applications of teleoperated UAVs is inspection, we assumed that our algorithm should allow the robot to come relatively close to objects of interest and help the user to fly through obstacle-rich environments. Moreover, the user should always be aware when the algorithm changes their command and the algorithm should not perform undesired motions. Furthermore, the analysis of potential directions of motion should not be limited to 2D.

One of the limiting factors of the avoidance capabilities of our system is the narrow field of view of the camera. In vision based teleoperation the operator should not be allowed to command the robot in directions that are not instantly visible if the navigation system can not ensure safety.

In order to realize the avoidance algorithm, we assumed that a relatively accurate representation of the robot's surroundings is needed. In the previous chapter, we proposed an algorithm for obstacle detection and tracking based on the bin-occupancy filter and measurements from a depth camera, that creates and updates a robot-centric and bounded obstacle state, partitioned into bins. The obstacles are represented as the probability of occupancy assigned to every bin of the state. The resolution of the cells can be tuned to achieve a trade-off between the processing time and the accuracy of obstacles representation.

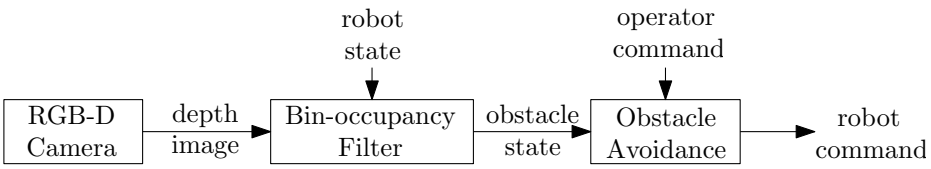


Figure 4.1: Block diagram of the navigation system.

4.1.3 Methodology

The aim of the work presented in this chapter is to develop an algorithm that will alter the user input to avoid collisions in a way optimized for teleoperation. Therefore, having had the obstacle state created with the bin-occupancy filter, our algorithm checks for possible obstacle interference with the desired trajectory of the robot to prevent collisions. The algorithm can be tuned to achieve the desired behavior of the system. By changing the value of weights in the cost function, different avoidance actions can be favored. The user input can be limited to stop the robot in front of an obstacle, or the direction of motion can be changed in order to fly around an object on the robot's path by either passing it on left, on the right, or by going above it.

Since we are using an RGB-D camera with a limited field of view, we have integrated the obstacle tracking algorithm with a Model Predictive Control (Camacho and Bordons (2007)) inspired avoidance to modify the velocities commanded by the operator. We detail the avoidance algorithm in Sec. 4.2.

In Sec. 4.3 we present a hardware-in-the-loop simulation setup for quadrotors. With our setup we are able to command the robots in a Gazebo simulation, a popular open source ROS-enabled physical simulator, using computational units that are embedded on our quadrotor UAVs. Hence, we can test in simulation not only the correct execution of algorithms, but also the computational feasibility directly on the robot's hardware. In addition, since Telekyb is inherently designed for multi-robot systems, with our setup we can also test the communication flow among multiple robots. We provide two use cases to show the characteristics of our setup.

In Sec. 4.4 we show a thorough validation of our algorithm over multiple experiments in different obstacle setups. We conclude this chapter in Sec. 4.5.

4.2 Avoidance Algorithm

The architecture of our navigation system is as follows. First, the obstacle detection and tracking module creates a local obstacle state using the bin-occupancy filter. The role of this part of the system is not only to detect obstacles but also to extend the limited field of view of the camera by including regions that are not instantly

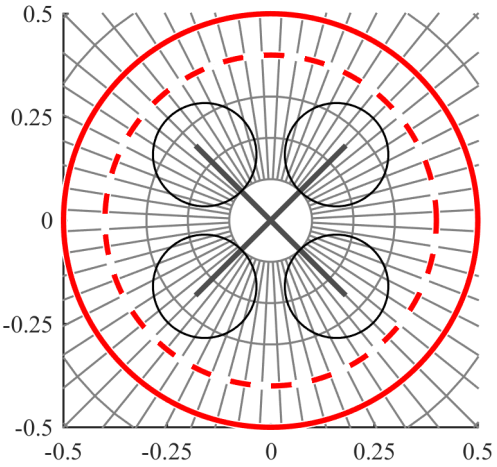


Figure 4.2: Top view of a segment of the bin representation and, in red, boundaries of a possible restricted area given the presented size of the robot.

visible. Using filtering, we can extend the knowledge of obstacles in the vicinity of the robot and reduce measurement noise as well. The second module in the navigation system is the obstacle avoidance. At every time step, the algorithm tries to predict potential collisions by analyzing the possible future obstacle states by performing a sequence of simulated state updates (Sec. 3.3.5) in different directions in the range of possible motions. Its working principle is inspired by MPC, where the system state is described with the obstacle state and the dynamics of the system with the user input and state updates.

The block diagram of the navigation system is shown in Fig. 4.1. The key idea is that the operator's commands are altered by the obstacle avoidance block given the knowledge of the estimated obstacle state by our detection and tracking algorithm fed with the robot's state and depth images.

4.2.1 Probability of Collision

To ensure no collisions, the obstacle avoidance part of our navigation system ought to prevent obstacles from entering the region of S occupied by the robot. We define the probability of collision as a joint probability of occupancy over a subset of bins $b_i \in S_R \subset S$, where S_R represents the part of the surveillance region associated to the robot. The boundary of the restricted area S_R is represented in the Fig. 4.2 with the red continuous circle.

Assuming that collisions with obstacles in different bins are independent, we

define the probability of collision with obstacles in S_R at instant k as

$$1 - \prod_{i \in S_R} \Big(1 - p(U_k(i))\Big), \tag{4.1}$$

where $p(U_k(i))$ is the probability of bin i being occupied.

In general, the two main sources of possible collision (i.e., an obstacle entering the region S_R in the local obstacle state) are

a) the operator driving the robot towards an occupied area,

b) uncommanded drift of the robot.

4.2.2 Model Predictive Control

In MPC, a model of the system is used to calculate a control input that minimizes an objective function. The system state is predicted at a finite-time horizon. To determine the control input, an Optimal Control Problem (OCP) is solved sequentially at each instance of the system. Then, the first control input is applied to the system, while taking future steps into account.

An example of such an objective function is

$$J(x, u) = p(x_k) + \sum_{t=0}^{N-1} q(x_k, u_k)\,, \tag{4.2}$$

where the time horizon is expressed as N computation steps in which the cost, expressed as $q(x_k, u_k)$ and $p(x_k)$, is computed. The optimization problem consists of finding the value of u_k that minimizes the cost function (4.2).

4.2.3 Commanded Velocity

The quadcopter is commanded by the operator in the horizontal frame H, i.e. the desired velocity is given in directions parallel and perpendicular to the gravity vector (Sec. 2.2.1). Since the platform is intended for teleoperation, the range of possible commanded velocities is limited. In particular, we have allowed the following movements

- forward/backward translation,
- along the Z_H axis (up-/downward),
- rotation around the Z_H axis (left/right).

The forward direction is defined in H by the orientation of the camera, i.e., with the unit vector

$$\boldsymbol{R}_z({}^{Q}\psi_C)\begin{bmatrix}1 & 0 & 0\end{bmatrix}^{\mathrm{T}} = \begin{bmatrix}cos({}^{Q}\psi_C) & sin({}^{Q}\psi_C) & 0\end{bmatrix}^{\mathrm{T}}, \tag{4.3}$$

where ${}^{Q}\psi_C$ is the yaw angle of C in Q.

The values sent to the robot are the commanded velocity

$${}^{H}\boldsymbol{v}_D = \begin{bmatrix}v_{Dx}\\ v_{Dy}\\ v_{Dz}\end{bmatrix} = \begin{bmatrix}cos({}^{Q}\psi_C)v_D\\ sin({}^{Q}\psi_C)v_D\\ v_{Dz}\end{bmatrix} \tag{4.4}$$

and the commanded yaw rate ${}^{H}\dot{\psi}_D$, where v_D, v_{Dz} and ${}^{H}\dot{\psi}_D$ are three inputs provided by the operator using an input device. The mapping between the input device's workspace to the velocity command was based on the one presented by Stegagno *et al.* (2016).

4.2.4 Obstacle Avoidance

The obstacle avoidance algorithm foresees possible collisions based on the user input and the current obstacle state. To achieve collision-free teleoperation, our algorithm solves an optimization problem over N time-steps

$$\begin{aligned}\underset{\boldsymbol{u}_k}{\operatorname{argmin}}\, J = &\sum_{k=1}^{N} w_k \left(1 - \prod_{i\in S_R}\Big(1 - p_i(ku_\rho, u_\psi, u_z)\Big)\right)\\ &+ w_\psi|u_\psi|\\ &+ w_z|u_z|\,,\end{aligned} \tag{4.5}$$

where

$$p_i(ku_\rho, u_\psi, u_z) = p(U_k(i)|(ku_\rho, u_\psi, u_z)), \tag{4.6}$$

is the probability of bin i being occupied in step k, it is determined using the model of the system defined in Sec. 3.3.5, where (ku_ρ, u_ψ, u_z) define the values of the state updates. The optimization problem in Eq. (4.5) produces the optimal control input $\boldsymbol{u}_k^\star = (u_\rho, u_\psi, u_z)^{\mathrm{T}}$ and gives the number $N_s \leq N$ of collision-free steps. The parameters w_k, w_ψ and w_z are weights that define the cost of the translation limit because of the probability of collision, the change of the desired direction of motion, and the change of elevation of the robot, respectively. Since by the earlier definition, the operator only commands the robot to go forward on the X_HY_H plane, we penalize the change of direction and elevation by adding weighted costs.

The number of collision-free steps N_s is defined as the maximum value of k for

which the collision probability is lower than the safety threshold given the obtained value of $\boldsymbol{u}_k^\star$.

The avoidance algorithm produces a control input that ensures the avoidance of obstacles and minimizes the change of magnitude of the user's input, direction, and altitude of the robot. It is a *passive* approach, i.e., no action is performed in the absence of an operator's command. It is valid under the assumption of an accurate state estimation and fast control, i.e., assuming no uncommanded drift of the robot.

The control input $\boldsymbol{u}_k$, expressed in the cylindrical coordinates M, describe the transformation of the obstacle state, as defined in Sec. 3.3.5, and has the following form

$$\boldsymbol{u}_k = \begin{bmatrix} u_\rho \\ u_\psi \\ u_z \end{bmatrix} = \begin{bmatrix} \Delta\rho \\ n\Delta\psi \\ m\Delta z \end{bmatrix}, \begin{matrix} n \in Z : \{n\Delta\psi \in [-\frac{\pi}{2}, \frac{\pi}{2}]\} \\ m \in [m_{min}, m_{max}], \end{matrix} \tag{4.7}$$

where $\Delta\rho$, $\Delta\psi$ and Δz are the dimensions of the bins in S. The above relation also constrains our optimization problem in Eq. (4.5) by defining the feasible region of $\boldsymbol{u}_k$.

This control input describes a translation of the size of a cell $\Delta\rho$ in each prediction step k, in the azimuth direction of $n\Delta\psi$ and a potential change in elevation along the Z_M axis of $m\Delta z$. The parameters n and m define the maximum deviation from the desired values given by the user.

The number of prediction steps N is not predefined, but is decided based on the forward velocity commanded by the operator v_D and the time horizon T_h of the prediction. In particular, it corresponds to the number of u_ρ updates needed to achieve the desired translation of $v_D T_h$. The value of N can be computed as

$$N = \left\lceil \frac{v_D T_h}{\Delta\rho} \right\rceil. \tag{4.8}$$

Lastly, the reference velocity in the frame H sent to the robot is

$${}^H\boldsymbol{v}_{ref} = f(\boldsymbol{u}_k^\star, v_D) = \frac{N_s}{N} \mathbf{R}_z \left({}^Q\psi_C\right) \begin{bmatrix} v_D \cos(u_\psi) \\ v_D \sin(u_\psi) \\ u_z / T_h \end{bmatrix} \tag{4.9}$$

Please note that the coefficient N_s/N is meant to limit the velocity when there is no collision-free path over the whole time horizon T_h. In the extreme case when $N_s = 0$, the reference velocity is set to zero. This case corresponds to the situation in which obstacles are in front of the robot in all feasible directions.

To summarize, our obstacle avoidance algorithm produces a reference velocity (4.9) as a set of translations of the obstacles state by minimizing the difference between the commanded velocity (4.4) and the output reference signal. The possible

cases are

- $\boldsymbol{v}_{ref} = \boldsymbol{v}_D$ when there is no predicted collision on the desired direction,
- lateral or vertical avoidance by projecting the desired velocity $\boldsymbol{v}_D$ on a new, collision-free direction,
- limit of the commanded velocity when there is no fully collision-free direction,
- additional component on the Z_H direction for vertical avoidance.

Remark 4.1 *Although we do not allow the operator to command any lateral motion, the algorithm, knowing the obstacle state, can perform such a motion.*

Remark 4.2 *The algorithm cannot perform any motion by itself, any obstacle avoidance action is only possible under the presence of operator input.*

Remark 4.3 *In order to compute Eq. (4.5), it is not necessary to perform the prediction of the obstacle state of the whole region S. In fact, to reduce the computational time, it is possible to compute only the future state of the bins that belong to S_R.*

Backward Motion

So far, we have covered the case of forward motion of the robot. When the operator gives a positive input, the velocity command is extrapolated in the form of simulated obstacle state updates, the probability of collision is evaluated, and an alternative input is chosen if needed. In the case of yaw rotation no action is needed, as the overall joint probability of collision in S_R does not change due to the cylindrical shape of this region. In order to command the robot in a different direction, the operator must first rotate the platform and then command the robot to travel in the new forward direction.

As one of the objectives of our teleoperated setup is that we allow the user to come very close to objects, we also needed to enable backward motion to avoid tedious maneuvers close to obstacles. The algorithm, however, will limit avoidance action to the reduction of the commanded velocity and would stop the platform if a potential collision was detected. Hence, in the case of a negative input, the reference velocity has the following form

$$ {}^{H}\boldsymbol{v}_{ref} = \frac{N_s}{N}\mathbf{R}_z\left({}^{Q}\psi_C\right)\begin{bmatrix} v_D \\ 0 \\ 0 \end{bmatrix} \tag{4.10} $$

where the ratio ${}^{N_s}/_{N}$ converges to zero as the robot gets closer to an occupied region.

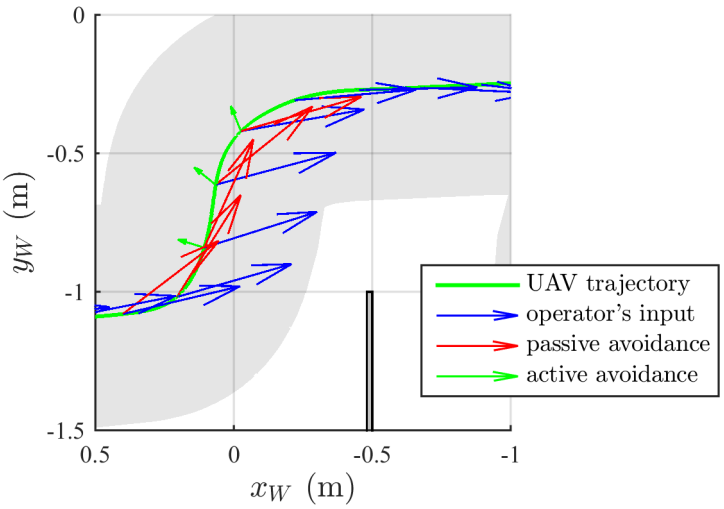


Figure 4.3: Top view of a part of robot's trajectory, in gray, a projection of the restricted area.

Additionally, as we do not differentiate between empty and unknown cells, we imply a second limit to backwards motion. The robot can only be commanded to go back up to a predefined distance, lower than the radius of the surveillance region S, and only backtrack its previous forward motion. In the case of a significant change in bearing, the backward motion is restricted.

4.2.5 Active Avoidance

Our avoidance algorithm, described in the previous section is a passive (or reactive) approach, i.e., any avoidance action can be performed only under the presence of the user's input. This is true, as long as the estimation system and the controller are accurate enough to prevent the robot from drifting. However, when that happens, it is possible that some of the obstacles can get too close to the robot and the algorithm can not correct for that.

In the current on-board implementation there are two main factors that contribute to the potential drift of the robot. These factors are related to the presumably lower quality of the on-board velocity estimate with respect to the estimate provided by an external motion capture system. First, the on-board estimate is affected by larger noise and may be biased for some time. As a result the quadrotor will naturally drift in the direction of the error. The second source of uncommanded drift is related to the delay in the velocity control loop. As a lower quality estimate of the velocity is now available, we need to reduce the gains of the velocity controller, making the system slower to respond to the reference velocity.

In order to tackle this issue we have isolated a subset of S_R at its boundary

S_{Rb} (between the solid and dashed red lines in Fig. 4.2), thus, we can extract the occupied bins in S_{Rb} as $S_O = \{b_i \in S_{Rb} : p(U_k(i) > 0)\}$. The algorithm actively checks this region of the obstacle state for possible obstacles and in a case of detection performs a repulsive avoidance action.

The active avoidance velocity command $\boldsymbol{u}_A$ is computed in the opposite direction than the detected obstacles, proportionally to the possibility of collision according to

$$\boldsymbol{u}_A = \begin{bmatrix} u_{Ax} \\ u_{Ay} \end{bmatrix} = \sum_{b_i \in S_O} \frac{-ap(U_k(i))}{1 - \prod\limits_{b_i \in S_O} (1 - p(U_k(i)))} \frac{\boldsymbol{x}_{b_i}}{|\boldsymbol{x}_{b_i}|}, \tag{4.11}$$

where $\boldsymbol{x}_{b_i}$ is the (x, y) coordinate of bin i in H and a is a parameter that defines the magnitude of the repulsive velocity, and the vector $\boldsymbol{u}_A$ is then added to the reference command obtained from Eq. (4.9) and the new command is sent to the robot. In the worst case, when there is no motion that clears S_R from the obstacles, the produced command minimizes the possibility of collision (4.1).

An example of the active avoidance is shown in Fig. 4.3 as green arrows. It is a top view of the robot's trajectory in one of our experiments. In the presented example, the robot was commanded towards an edge of an obstacle, it can be seen how the operator's input (in blue) command was changed to circumvent the obstacle (in red). The controller, however, was not fast enough to limit the forward velocity of the robot, and the additional active component was generated (in green) to assure collision-free operation.

4.3 Hardware-in-the-loop Simulations

In this section, we present our setup for multirotor hardware-in-the-loop simulations. As the development of an obstacle avoidance algorithm entails testing in obstacle rich environments, the risk of failures and damage to the system is very high. To overcome this issue at least partially, we designed a setup in which the algorithm could be tested in a simulated environment. Hence, there is no risk of damage to the actual hardware. Moreover, as we use the same computational unit, the method allows us to test the feasibility of execution of our algorithm directly on the on-board computer.

Motivated by the clear advantages of this type of testing, we subsequently extend this approach to multi-robot scenarios to cover a wider spectrum of UAV related experiments, and shared our result with the community in the work Odelga *et al.* (2015).

4.3.1 About HIL Simulations

Research on computationally power-demanding algorithms has required the development of new, faster, more power efficient and miniaturized CPUs. As in the system of communicating vessels, demand on computational power drives the progress in computing hardware, while the development of new hardware enables more advanced, higher fidelity algorithms.

In robotics, the latest trends in research have pushed for on-board integration of highly informative sensors such as laser scanners (Omari *et al.*, 2013), cameras (Forster *et al.*, 2014) and RGB-D devices (Stegagno *et al.*, 2014; Fang and Scherer, 2015), where the quantity of available information demands high processing bandwidth. This is especially true in the field of computer vision, as can be seen in algorithms related to visual odometry (Delmerico and Scaramuzza, 2018), object detection, recognition and tracking (Ahmad *et al.*, 2017; Price *et al.*, 2018), where in many cases the available processing power is the limiting factor and algorithms are tuned to balance computational time and performance.

The increase in available information and the need to process it on-board also requires more computational power embedded directly on the robot. However, due to payload and power consumption constraints, the computational power available on-board is not yet comparable to the one of a normal desktop PC.

Another trend is the use of physical simulations to test algorithms for robotics before the hardware implementation phase. Simulations are particularly useful when considering UAVs as each experiment can be time consuming and even result in a crash. However, whenever porting an algorithm from simulation to physical hardware, frequency synchronization issues may arise due to limited on-board computational power. The main problem is that simulations usually run on a different hardware than the one equipped on-board, hence it is impossible to check the real execution time of the software.

Literature Overview

Hardware-in-the-loop (HIL) simulations (Burbank *et al.*, 2011) are a good way to test these aspects without a need of real robot experiments. In the work by Chandhrasekaran and Choi (2010) the authors present a UAV system with HIL simulations for testing the platform with real-time data. Their system consists of a reliable platform for testing critical safety properties with special attention. Thus, through HIL simulation they greatly reduce experimental costs. Another HIL setup, for a UAV helicopter, can be seen in the work by Cai *et al.* (2008). The authors show its cost-efficiency in terms of verification of the overall control performance and safety. Their system, capable of simulating flight tests, including basic flight motions and full-envelope flights, confirms the high effectiveness and usefulness of HIL simulations.

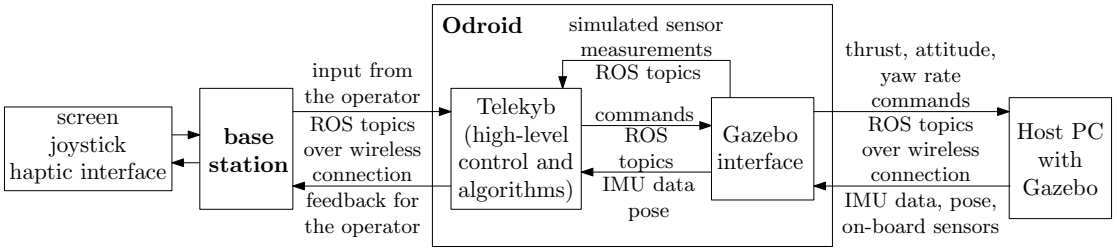


Figure 4.4: A block scheme of the hardware-in-the-loop simulation setup.

Methodology

In this section, we present our setup for HIL simulations for UAVs. This approach not only enables tests of the functioning of our algorithms, but, in contrast to only software simulations presented in Sec. 2.6, also allows tests of the hardware constraints of our computational units. It is a scalable approach, and thus allows testing with a single or multiple robots.

Our setup for HIL simulations consist of two main parts. The first is Gazebo, a popular open source ROS-enabled simulator, which provides the dynamical simulation of one or more UAVs and the corresponding sensor readings (IMU, cameras, etc.). On the other side, each of the simulated UAVs is driven using an ARM-based Odroid board, which is the high level control board installed on our quadrotors. The interfacing between the components is provided by a ROS node based on the Telekyb software (Grabe *et al.* (2013)).

Using this simulation scheme, we obtain two major benefits with respect to software simulation. First, we can test algorithms directly on the on-board computational units, hence also testing the computational times and the feasibility in real-time. In addition, since our setup is scalable to multi-robot systems, we can also test the communication among multiple boards.

In Sec. 4.3.2 we show the software setup for HIL simulations in detail and talk about specific aspects of our approach for both single and multi-robot scenarios. In Sec. 4.3.3 we show two case studies in which we demonstrate the feasibility of hardware-in-the-loop simulations in testing of our obstacle avoidance algorithm for teleoperated UAVs, and an extension of HIL simulations to multi-robot systems.

4.3.2 Simulation Setup

In Sec. 1.2.2 we described the software setup used to drive our UAV platform. Here, we show the necessary modifications in order to run HIL simulations and we point out the required steps to extend the system to perform multi-robot HIL simulations.

In order to perform HIL simulations, we can exploit the standardization of the input/output provided by Telekyb, whose interfacing with other blocks is run solely

through ROS topics. A block scheme of our setup to perform hardware-in-the-loop simulations is depicted in Fig. 4.4. With respect to the scheme in Fig. 1.2, the hardware and sensor interface block has been replaced with a Gazebo interface block. The main functionality of this setup is to provide an interfacing layer between Gazebo and the high-level control algorithms.

In particular, this approach uses data provided by the sensors and ground truth data provided by Gazebo to emulate the ROS topics provided by the Vicon tracking system (if required), IMU, cameras and other sensors present both in the simulation and the real robot setup. The emulation not only comprises of the specific topics and format on which the data are provided by Gazebo, but also handles frequency synchronization. For example, our actual Vicon system provides data at 120 Hz, while the physical simulation is performed with a timestep of 1 ms, so that ground truth pose information about the UAV is available at 1000 Hz. Nevertheless, the pose provided to the Telekyb blocks is temporized at 120 Hz by the Gazebo interface.

On the other hand, the Gazebo interface translates the thrust, attitude and yaw rate commands provided by the high-level controller into commands that can be read from the simulator and applied to the simulated UAV model.

Interfacing with the base station and operator does not change with respect to the real robot system. The Gazebo simulator can be hosted either on the base station or on a different machine. In order to also test the communication link between the robot and the base station, the latter is preferred, while the first can be implemented if the objective of the simulation is only to test the functionality and the execution time of the implemented algorithms. In both cases, the communication between the Odroid board and Gazebo is achieved through ROS topics over wireless IEEE 802.11 connection.

Multi-robot Extension

Since all components of the hardware-in-the-loop simulation, and in particular Telekyb, are inherently thought to be for multi-robot applications, only a few adaptations are required to perform multi-robot HIL simulations, which are as follows.

The most important adaptation is that each robot simulated in Gazebo is endowed with a unique identifier, which must be inserted in the name of all ROS nodes and topics relative to such a robot. Hence, for each robot in Gazebo, one Odroid board is set up to host a Gazebo interface that reads the appropriate topics (i.e., the topics containing the own ID) and a Telekyb instance connected to such ID. Each Odroid is also connected to the base station in the same way as in the previous case. The resulting scheme is presented in Fig. 4.5.

In addition to the single robot case, it is also possible to test the inter-robot communication network. At the current stage of development, the communication between robots is yet again performed exploiting ROS topics on a wireless IEEE

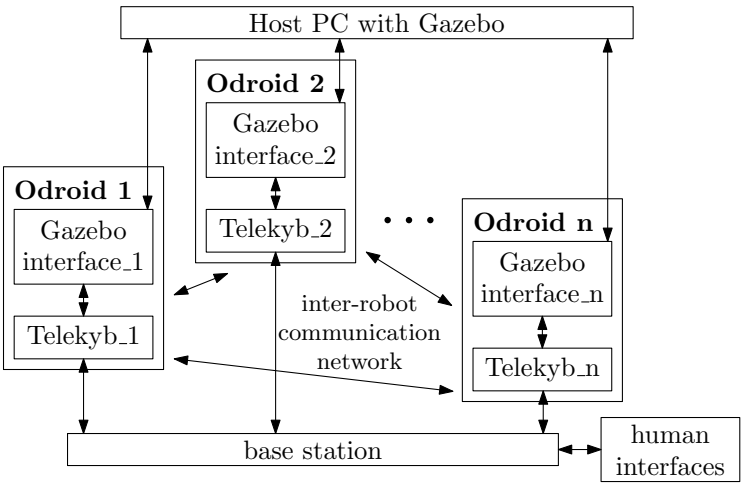


Figure 4.5: A block scheme of the multi-robot hardware-in-the-loop simulation setup.

802.11 channel. Nevertheless, it is also possible to include and test custom communication networks among the robots equipping Odroid boards with appropriate hardware (e.g., bluetooth antennas).

4.3.3 Experiments

To illustrate usability of HIL simulations we present two case studies in this section. Our main interest in this type of testing, in the context of this thesis, is to test our navigation system architecture with obstacle detection, tracking and avoidance in a single robot case. Therefore, we present our results in a HIL setting with a simulated quadrotor and a single obstacle that can be approached multiple times as the robot is teleoperated.

In the second case study, we highlight the extended multi-robot capabilities of our hardware-in-the-loop simulation scheme by performing formation control with three simulated UAVs driven by three Odroid boards.

Obstacle Avoidance

In order to conduct the experiment with simulated UAV and sensor readings, we recreated our quadrotor setup presented in Sec. 2.2.2 in Gazebo using a generic dynamic model of a multirotor[1]. Color and depth images are generated thanks to an RGB-D sensor plug-in. The intrinsic and extrinsic parameters, i.e. camera parameters and sensor pose, respectively, are set to match the real hardware.

[1] `https://github.com/ethz-asl/rotors_simulator`

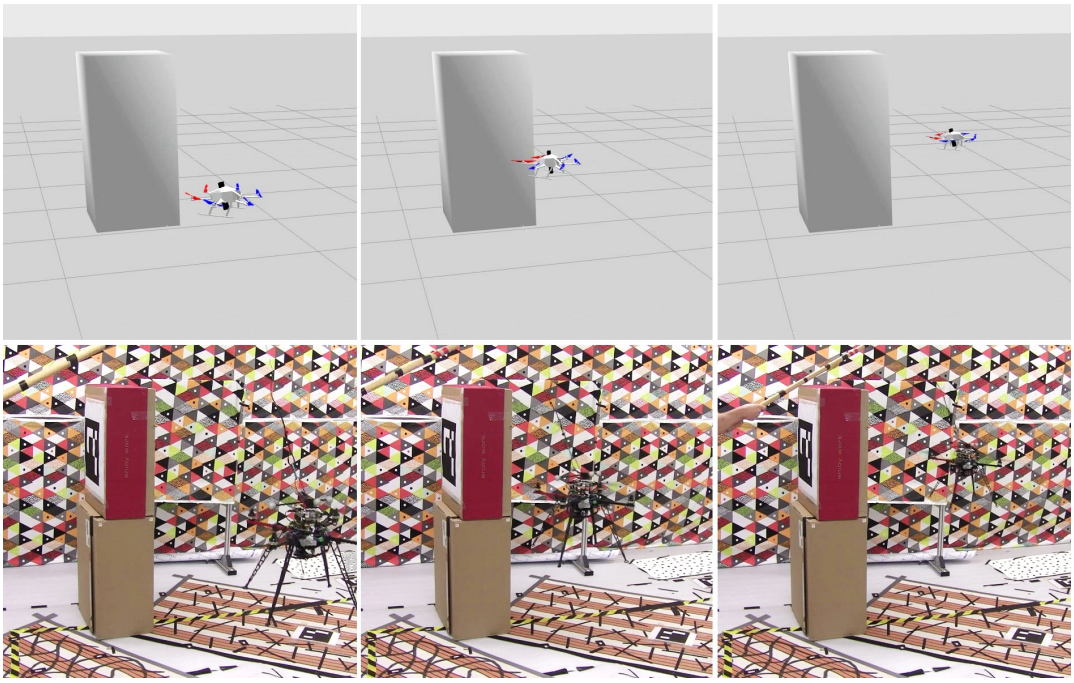
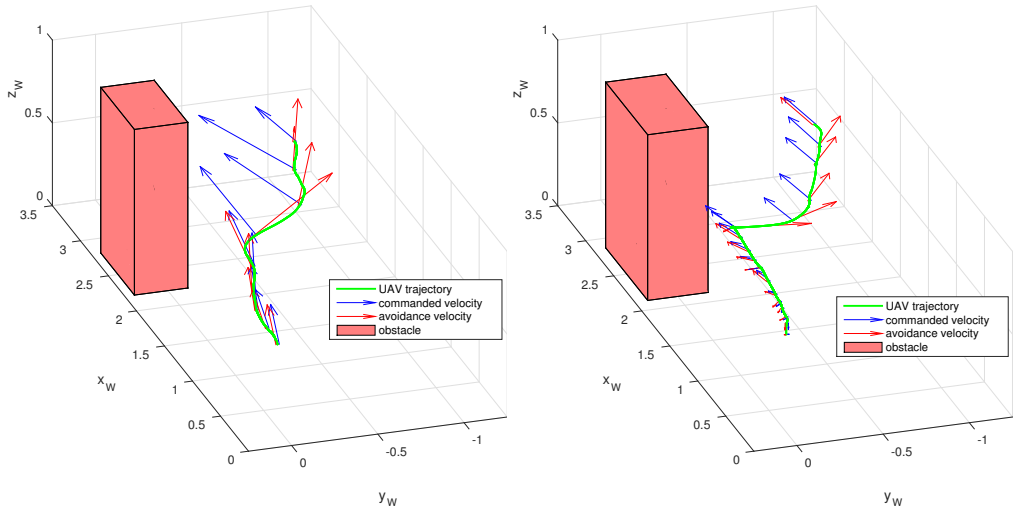


Figure 4.6: Three comparative snapshots of the same obstacle avoidance algorithm performed in hardware-in-the-loop simulation (top) and a real UAV experiment (bottom).



(a) (b)

Figure 4.7: The comparison of trajectories performed by the UAV in a real experiment (left) and in hardware-in-the-loop simulations (right).

	Module 1		Module 2	
	μ	σ	μ	σ
PC	<1 ms	-	<1 ms	-
HIL	1.74 ms	0.302 ms	5.3 ms	1.42 ms
Robot	1.95 ms	0.339 ms	5.65 ms	0.78 ms

Table 4.1: Execution time comparison of the measurement update (module 1) and the state update (module 2) on different platforms.

To control the simulation we used the control scheme shown in Fig. 4.4 with appropriate components to interface with Gazebo instead of the real quadrotor. The trajectory controller part of Telekyb was tuned accordingly to adjust to the slightly different dynamics of the simulated UAV.

We have conducted simple obstacle avoidance experiments performed both in simulation and on a real robot. In both cases, the human operator was commanding the robot to go towards the edge of the obstacle. In Fig. 4.6 snapshots from the experiments are presented and show consecutive positions of the robot during the flight. Full trajectories, together with the commanded and reference velocities, are presented in Fig. 4.7. Both figures show comparable behavior of the real and the simulated robot in the experiments.

In Table 4.1 we show the mean values μ and standard deviations σ of the execution time of two modules of the algorithm. These modules are related to the measurement and state updates of the obstacle state, respectively. We have run the same experiment on three different computers: a standard desktop PC, hardware-in-the-loop simulation, and the real platform. Table 4.1 shows that the computation times obtained in the PC simulation are very different with respect to the timing obtained in the other two experiments. On the contrary, the computation times in the HIL simulation and in the experiment with a real robot are comparable. Small differences in the execution times in HIL and real experiments are due to the real sensor acquisition time, which is, however, negligible compared to the rest of the algorithm's execution time. It is clear from the table that the timing obtained on the PC does not provide any meaningful insight into the evaluation of the real-time feasibility on the real hardware. However, using HIL simulation we were able to evaluate the real-time execution time which was comparable to the real experiment's execution time.

Multi-robot Formation Control

In order to test the multi-robot capabilities of our HIL setup, we have performed a multi-robot teleoperation experiment with three simulated UAVs driven by three separate Odroid-XU3 boards following the scheme of Fig. 4.5. The formation con-

Figure 4.8: A snapshot of formation control performed in a hardware-in-the-loop simulation.

trol algorithm for the robots is the one proposed by Franchi *et al.* (2012).

The test algorithm was specifically chosen because it requires some exchange of information among the robots in order to achieve consensus on the status of the system. This inter-robot communication was performed using the wireless IEEE 802.11 capabilities of the Odroid-XU3 boards. In Fig. 4.8 we show one snapshot of this simulation with three robots in a simple formation. This setup was later used in experiments in the work by Ahmad *et al.* (2016).

4.3.4 Conclusion

In this section, we presented a UAV hardware-in-the-loop simulation scheme which allows for the testing of the computational requirements of our algorithms directly on the computational unit that is equipped on the robots, while simultaneously enjoying the safety of a simulation. As we used the same on-board computational unit, HIL simulations provided a quantitative evaluation of the execution time, memory demand and the capability of our CPU to run all the system's software components simultaneously.

The presented examples show clear benefits of the simulation environment and similarly to the SIL shown in Sec. 2.6, it enabled the testing and tuning of our algorithm without the risk of damages to the platform. The comparable behavior of our system in simulations and real experiments proves the possibility of the generalization of the simulated results to the real experiments. Additionally, we can perform multi-robot HIL simulations to test the communication among the

robots.

The use of HIL simulations was very useful during the development of our obstacle avoidance algorithm, especially in terms of time needed to perform experiments. In fact, it was possible to test various parameters without the need to run real experiments. The use of the actual hardware (Odroid-XU3) in the simulations allowed us to check if the execution times of the various modules were compatible with the on-line execution.

4.4 Experimental Validation

In this section, we present selected results of our obstacle avoidance experiments. As we described in Sec. 4.3, we started from HIL simulations which enabled us to debug and tune every part of our algorithm (obstacle detection, tracking and avoidance) before deployment on an actual platform. After successful validation in the simulation environment, we performed avoidance experiments of simple obstacles in our laboratory with an external tracking system. The use of the tracking system allowed us to test the avoidance algorithm independently from state estimation, leaving more computational power for the former.

After the initial testing of the avoidance algorithm we shifted our focus to the development of the on-board state estimation algorithm, which we detailed in Sec. 2.5. The presented solution, thanks to its low computational power requirements, complemented our avoidance algorithm, enabling state estimation in a GPS-denied environment and made the platform self-contained in terms of sensor equipment and computations.

In order to fully validate our method, we have successfully performed multiple experiments in several obstacle setups including both horizontal and vertical obstacles. In the set of 50 experiments, the operator was able to guide the robot along the desired path with a success rate of over 94%. Excessive motion of the platform, due to drift in the state estimation, caused early termination of two experiments, which were aborted as safety measure to prevent crashes. Although the robot drifted, it merely touched an obstacle, as our algorithm was still able to limit the platform's velocity and prevent major collisions.

4.4.1 Indoor Experiments

The experiments presented in this section were performed with the following setting of the algorithm's parameters. The size of the surveillance region S (Sec. 3.3.3) was set such that

$$\rho \in [0, 2.5\,m], \quad z \in [-1.5\,m, 1.5\,m],$$
$$\Delta\rho_b = 0.1\,m, \quad \Delta\psi_b = \frac{1}{30}\pi\,rad, \quad \Delta z_b = 0.1\,m\,,$$

and the restricted area S_R occupied by the robot as

$$\rho \in [0, 0.5\,m], \quad z \in [-0.3\,m, 0.2\,m]\,.$$

The weights used in the cost calculation in Eq. (4.5) were set to

$$w_t = 10,\ w_\psi = 3,\ w_z = 3.$$

These values have been hand-tuned in a heuristic technique by the operator to adjust the avoidance behavior. The magnitude of the user input, i.e. the commanded velocity, was limited to 0.5 $^{\mathrm{m}}/_{\mathrm{s}}$ and the time horizon T_h for the collision prediction to 1 s.

All the velocity plots presented in this section, show the velocities in a horizontal frame in NWU orientation (Sec. 2.1). Hence, the forward (or longitudinal) velocity refers to the velocity component along the x axis and the lateral velocity to its component along the y axis. With the term vertical velocity we describe the velocity component along the direction of the gravity vector.

All the experiments presented in this section were performed in a laboratory with a Vicon tracking system and the world coordinates in the plots showing the robot's trajectories refer to the coordinates of the tracking system. The information from the tracking system was only used as a ground truth, this reference data was not sent to the robot. In all the experiments, in order to well illustrate the avoidance capabilities of our approach, the operator was instructed to fly towards the edges of the obstacles.

Zig-zag Flight

The first experiment presented here, consisted of a corridor with four walls perpendicular to the desired trajectory. The gap between the walls was set such that it did not allow for a straight, collision-free motion from the one end to the other. The operator commanded the robot to fly along the corridor, make a u-turn after the last obstacle, and return.

A full 3D view of the obstacle setup and the robot's trajectory together with the operator's commands and the avoidance commands can be seen in Fig. 4.9. Additionally, to illustrate the compactness of the setting, we show the ground projection of the trajectory with the gray shaded area corresponding to the robot's size. Fig. 4.3, shown earlier in this chapter, presents a close view of the avoidance of one of the walls with a detailed presentation of corresponding commands. Fig. 4.10 and 4.11 show the forward and lateral components of the commanded velocity. The avoidance algorithm added additional lateral components to navigate between the obstacles, while limiting the longitudinal component when necessary.

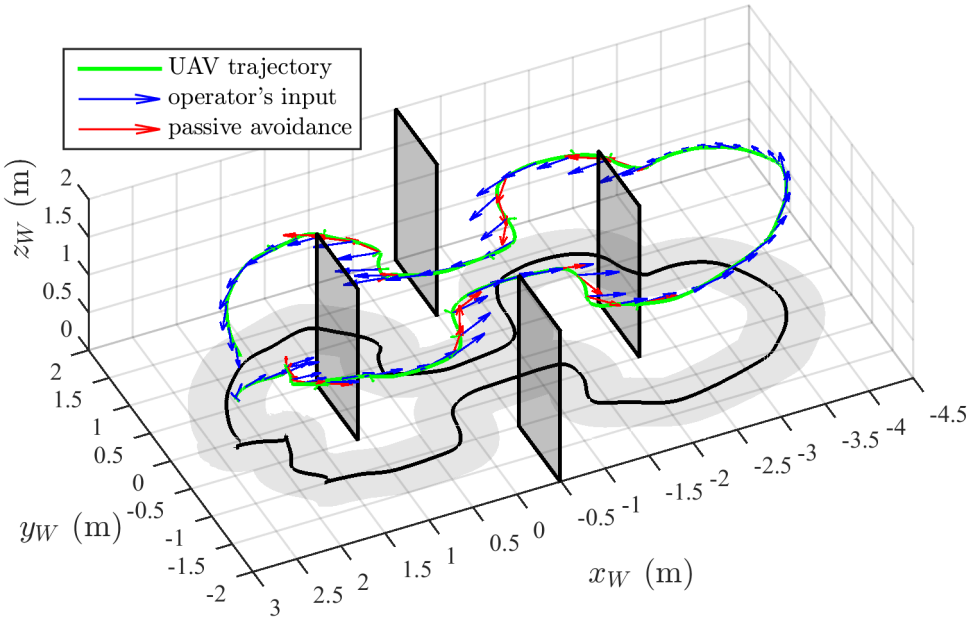


Figure 4.9: Zig-zag flight: the trajectory of the robot (in green) in the 4-wall avoidance experiment with its projection on the ground (in black), the commanded velocity (in blue) and the avoidance velocity (in red). The obstacles are represented as the gray cuboids.

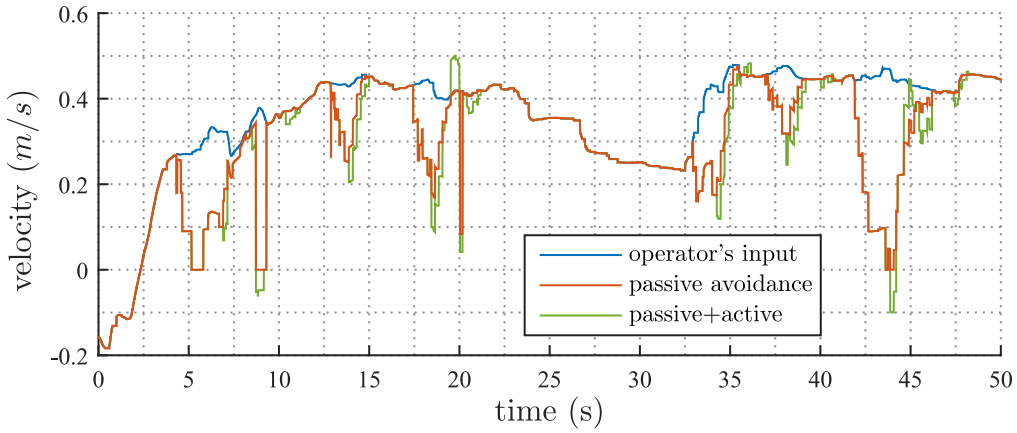


Figure 4.10: Forward velocities in the 4-wall avoidance experiment.

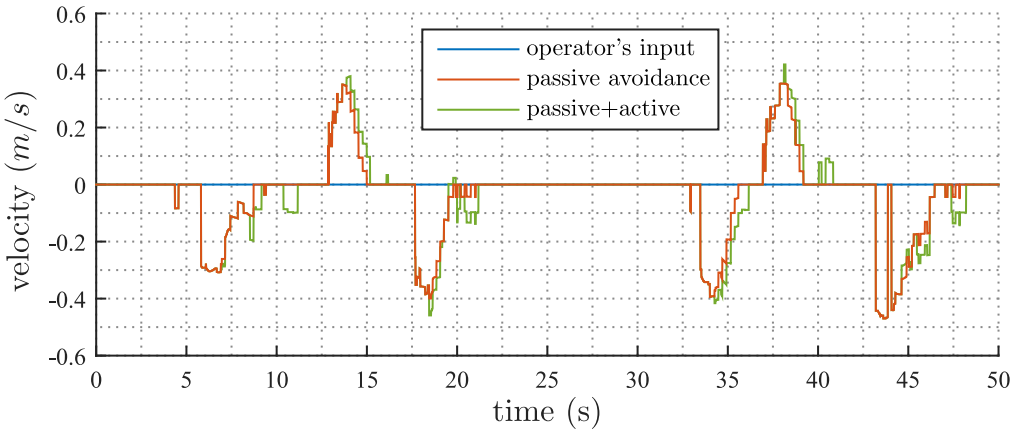


Figure 4.11: Lateral velocities in the 4-wall avoidance experiment.

Wall Following

In the experiment in Fig. 4.12, our robot was commanded to fly at an angle towards a straight wall. This simple example shows how the commanded velocity was altered by adding the additional lateral component (Fig. 4.13) resulting in a reference velocity that guided the robot along the surface of the wall, while the forward command was reduced to keep the absolute value of the velocity within the imposed limits. This example depicts the versatility of our approach, which resulted in the predicted and desired behavior, without utilizing additional algorithms, e.g., edge or surface detection.

Vertical Avoidance

The third experimental setup presented here, consisted of two horizontal obstacles along the desired path of the robot. In this case, the user commanded the robot to fly straight at the level of the obstacles' edges and our algorithm altered the commanded velocity in the vertical direction resulting in motions below and above the corresponding obstacles. The 3D trajectory of the robot can be seen in Fig. 4.15 and the forward and the lateral velocities in Fig. 4.16 and 4.17.

Backward Avoidance

The last experiment that we present in this section shows the case in which the robot was commanded to fly backward towards a previously detected obstacle. From the trajectory of the robot, shown in Fig. 4.18, can be seen that when the robot was below the top edge of the obstacle, the algorithm limited the backward motion to avoid a collision. However, after the robot was commanded to increase its altitude above the obstacle, the algorithm allowed the robot to fly further back,

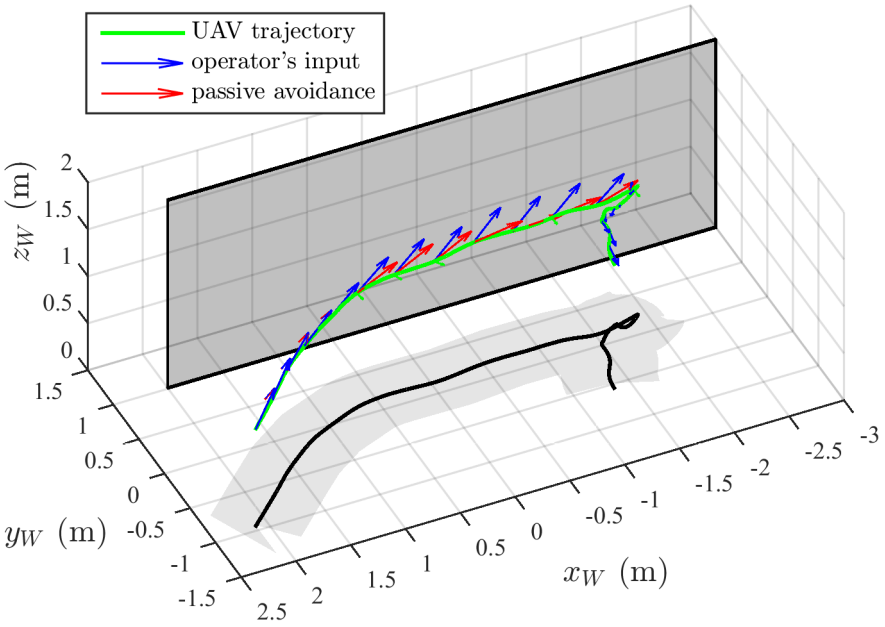


Figure 4.12: Wall following experiment: the trajectory of the robot (in green) with its projection on the ground (in black), the commanded velocity (in blue) and the avoidance velocity (in red). The wall is represented as the gray cuboid.

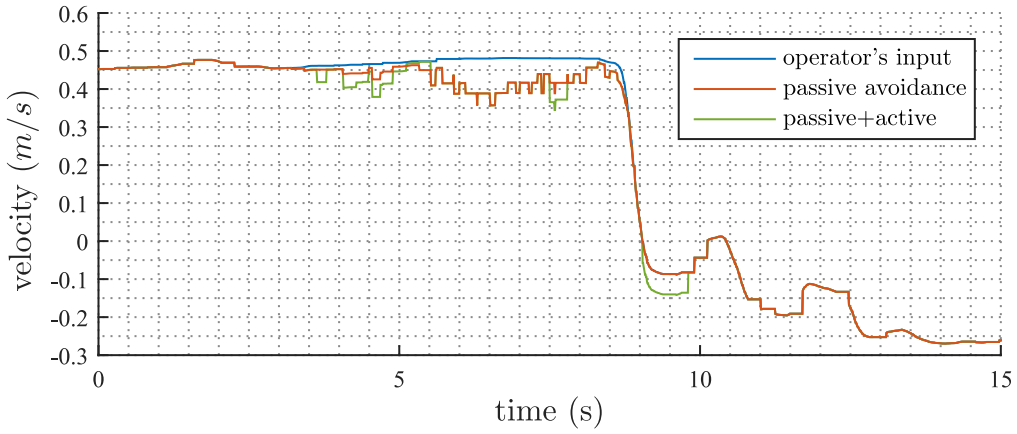


Figure 4.13: Forward velocities in the wall following experiment.

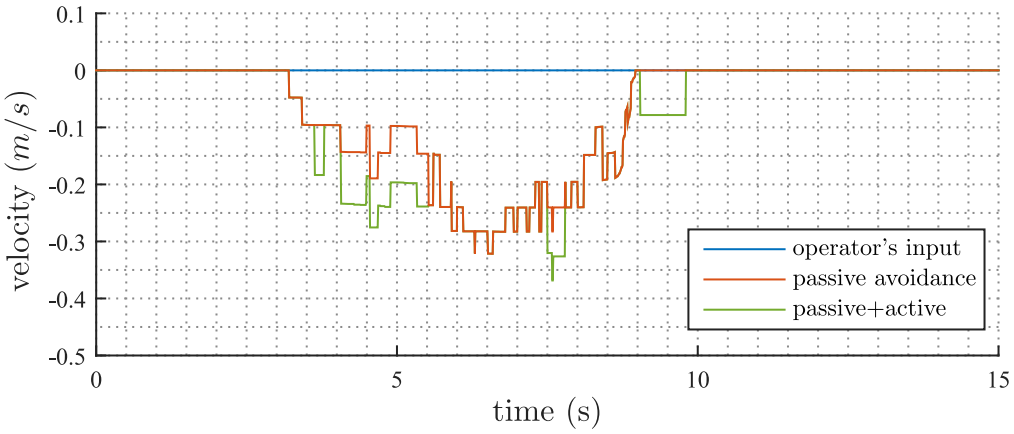


Figure 4.14: Lateral velocities in the wall following experiment.

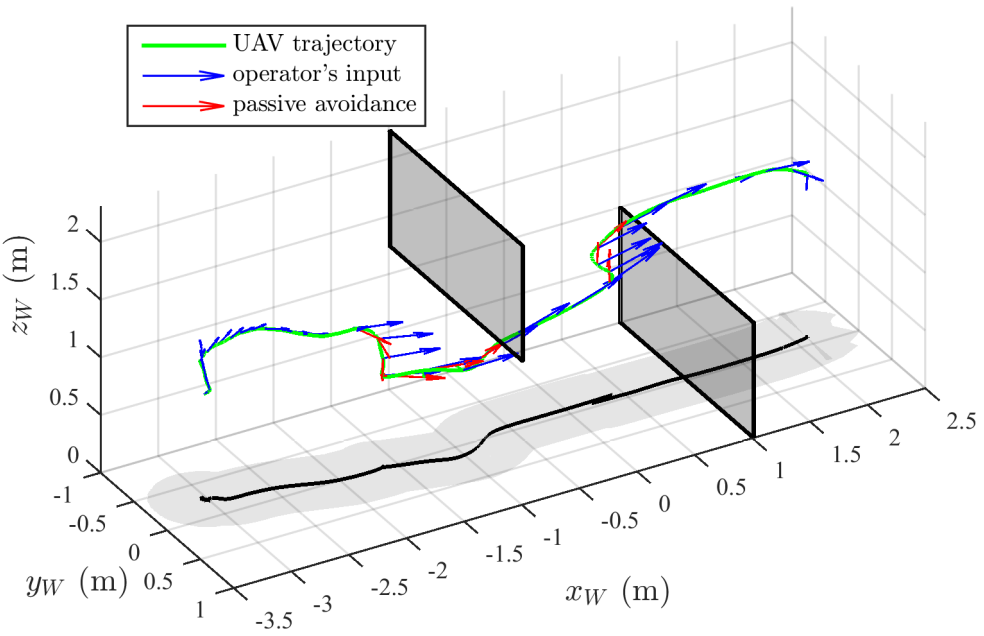


Figure 4.15: Vertical avoidance experiment: the trajectory of the robot (in green) in the avoidance of horizontal obstacles with its projection on the ground (in black), the commanded velocity (in blue) and the avoidance velocity (in red). The obstacles are represented as the gray cuboids.

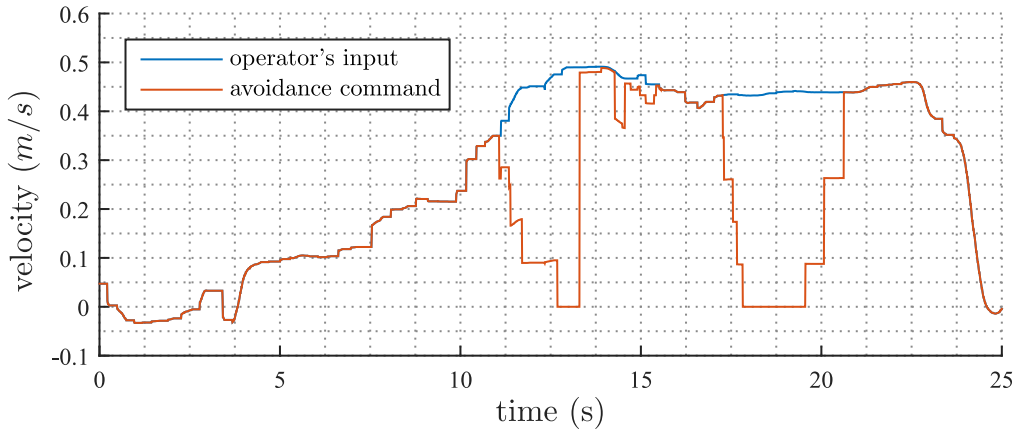


Figure 4.16: Forward velocities in the avoidance of horizontal obstacles.

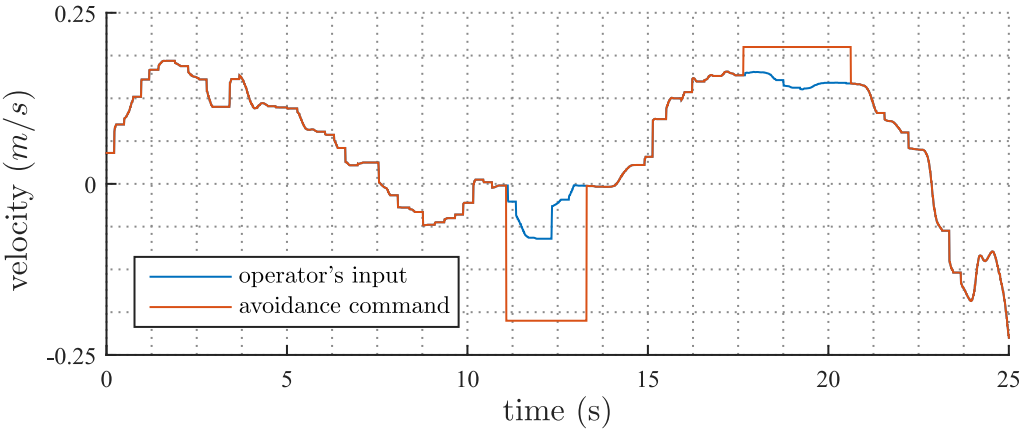


Figure 4.17: Vertical velocities in the avoidance of horizontal obstacles.

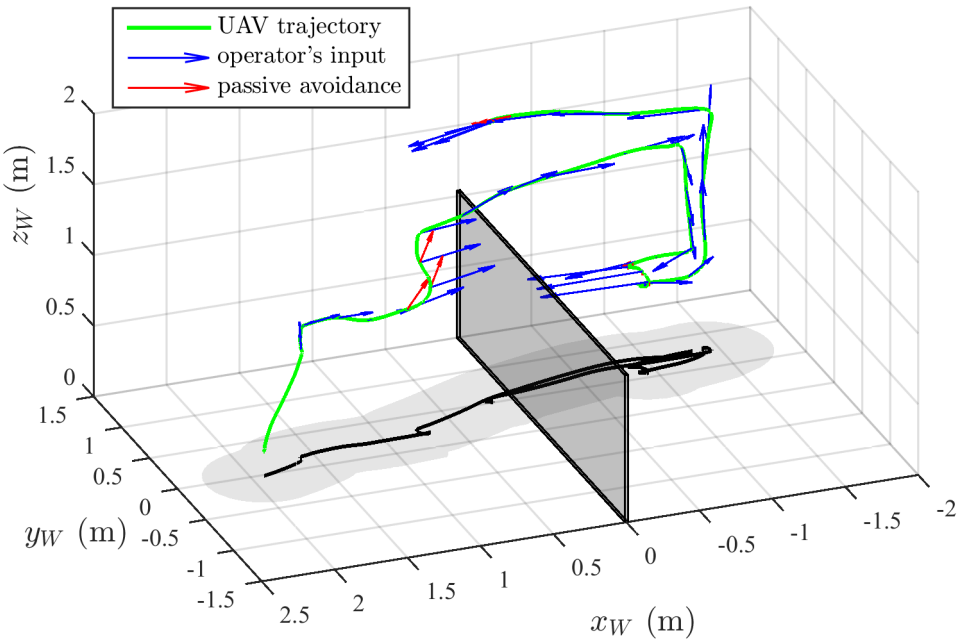


Figure 4.18: Backward avoidance experiment: the trajectory of the robot (in green) with its projection on the ground (in black), the commanded velocity (in blue) and the avoidance velocity (in red). The obstacle is represented as the gray cuboid.

up to the limit imposed by the size of the surveillance region. The corresponding velocities can be seen in Fig. 4.19 and 4.20.

4.4.2 Outdoor Experiments

The main limiting factor of utilizing our platform in outdoor settings is the depth sensing technology. The camera that we have used uses the structured light technique that is very susceptible to lighting conditions. The strong IR component of sunlight effectively blinds the sensor. In order to show that other aspects of our method do not limit its usability to indoor scenarios, we performed a few initial experiments in a forest during a cloudy day.

Thanks to the cloud cover and the canopy, the platform was shaded from the direct sunlight and the sensor was able to detect major obstacles as tree trunks and larger bushes. During these several minutes long experiments the operator tried to purposely fly into trees and our algorithm successfully prevented collisions, a snapshot of one of these flights is shown in Fig. 4.21. These initial outdoor testings also proved that our method, in terms of both the state estimation and obstacle avoidance is not limited to a structured laboratory environment and flat obstacles.

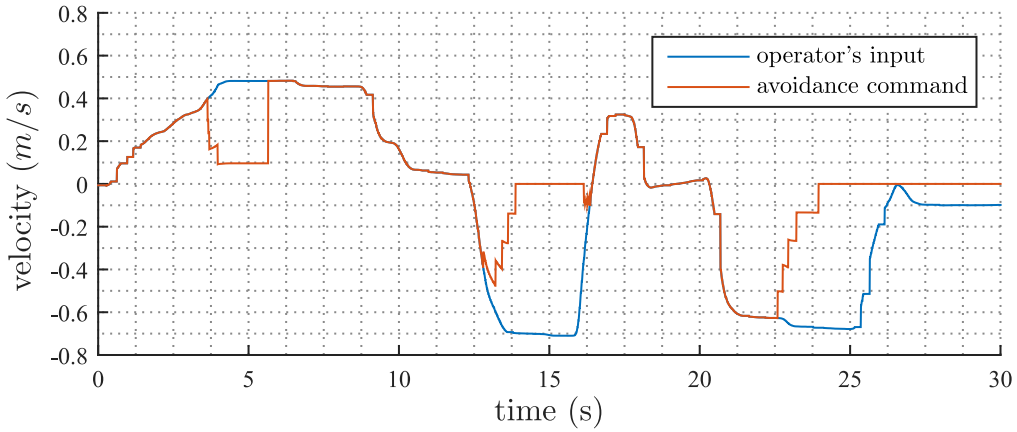


Figure 4.19: Forward velocities in the backward avoidance experiment.

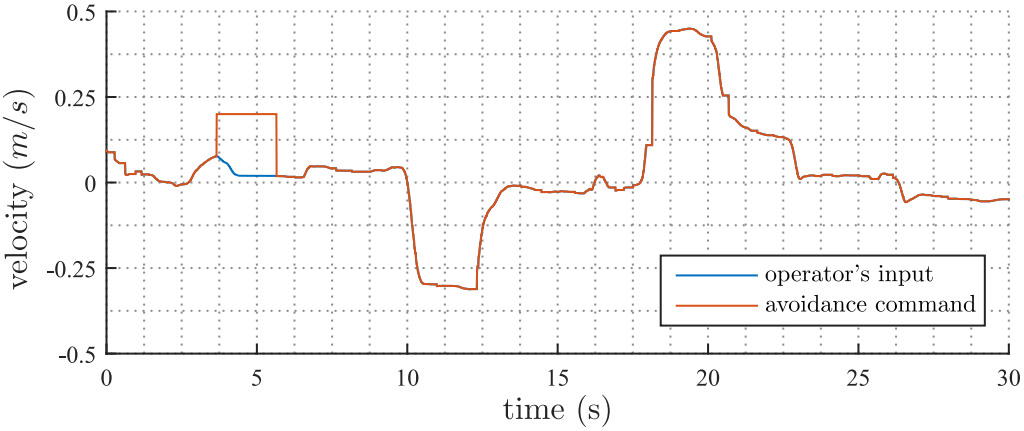


Figure 4.20: Vertical velocities in the backward avoidance experiment.

Figure 4.21: Snapshot from the recording of the forest experiment with our platform avoiding a tree and showing our safety equipment.

4.5 Summary

In this chapter, we presented our obstacle avoidance algorithm for a teleoperated UAV. Based on a local obstacle state, created using a bin-occupancy filter with measurements from a depth camera and the robot's state, the algorithm filters the operator's input and alters it when necessary. The estimated obstacle state is used to predict possible collisions and to modify the velocity commanded by the operator to avoid obstacles. Additionally, we added an active avoidance component to compensate for any possible drift of the platform. Through the experiments presented in this chapter, we not only validated our navigation system but also the on-board state estimator and the self-sufficiency of the platform. The platform is able to estimate its state in an indoor, GPS-restricted environment, using IMU and optical flow integration and is independent from external tracking systems and computations. The initial outdoor testing shows that the algorithm is not limited to structured environments and can handle obstacles of different sizes.

In this chapter, we also presented a UAV hardware-in-the-loop simulation scheme which allows us to test the computational requirements of our algorithms directly on the computational unit that is equipped on the robot, while simultaneously enjoying the safety of a simulation. Additionally, we can perform multi-robot HIL simulations to test the communication between robots.

The use of HIL simulations was very useful during the development of our obstacle

avoidance algorithm, especially in terms of time needed to perform experiments. In fact, it was possible to test various parameters without the need to run real-world experiments. The use of the actual hardware (Odroid-XU3) in the simulations allowed us to check if the execution times of the various modules were compatible with the on-line execution.

Chapter 5

Underactuation of UAVs in Teleoperation

5.1 Introduction

Unmanned aerial vehicles, being not constrained by ground conditions, can operate in places that are out of reach of other classical mobile robots. They offer high maneuverability, vertical take-off and landing, a hovering mode, and other features that make them popular platforms for many robotic applications such as inspection, exploration, surveillance, data collection, and recently also physical interaction with their environment (Ryll *et al.*, 2017) or even humans (Rajappa *et al.*, 2017).

5.1.1 Problem Statement

One of the limitations of multirotors with co-planar propellers, including quadrotors, is their intrinsic underactuation. A change in position or disturbance counteraction of such UAVs involve a change in their orientation. Although this may not impose any significant restraint in open-air flights, it might be crucial when the precision of control matters or in presence of external disturbances, e.g., an abrupt or unwanted change in orientation might involve the need of incommodious rectification of directional sensor measurements.

In the context of teleoperation, stabilization of the visual feedback from the UAV is an important factor affecting task performance, especially in obstacle rich environments as presented in Chap. 4 and in works by Stegagno *et al.* (2014); Odelga *et al.* (2016b).

In physical interaction involving either a direct contact (Gioioso *et al.*, 2014) or additional robotic arms attached to the multirotor body (Yüksel *et al.*, 2015, 2016), the lack of controllability over some degrees of freedom can significantly complicate the control task, e.g., by requiring higher order differentiation of the system model, and consequently estimation of higher order derivatives of the system state.

(a) (b)

Figure 5.1: Examples of camera gimbals for the use with UAVs for (a) small cameras, and (b) more professional equipment.

5.1.2 Methodology

One of the common solutions, mostly in aerial photography, is the use of additional stabilization devices (e.g., camera gimbals, Hilkert (2008)) that decouple the sensor rotation from the orientation of the platform. In Sec. 5.2 we present a selection of solutions of gimbal systems, comment on them and show our own design for the depth camera used in our work.

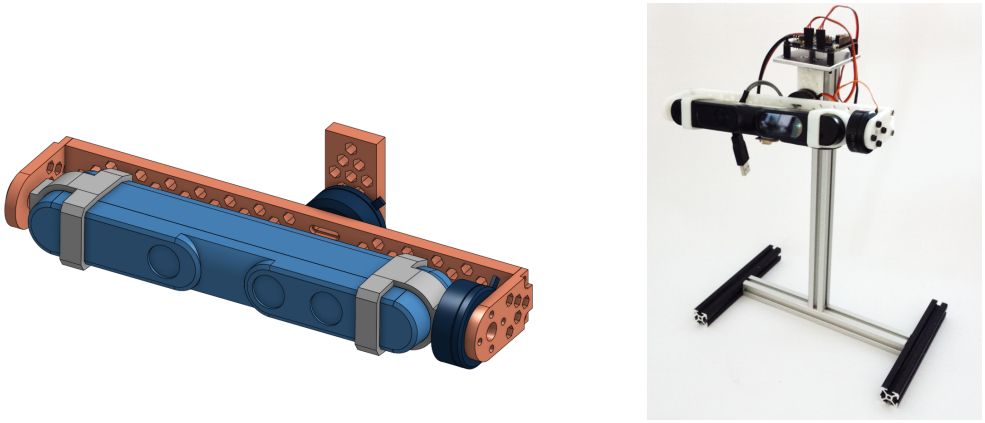
In many cases, however, mechanical methods are inadequate or impractical and their additional weight limits flight time. In Sec. 5.3 we present our algorithm for digital image stabilization using only IMU data. Since it is a software solution it adds no physical weight to the system, which is especially beneficial for small aerial vehicles.

In Sec. 5.4 we propose a novel contept of a fully-actuated UAV in which we expand the controllability with additional actuated DOFs.

5.2 Camera Gimbals

Stabilizing the video feed from UAVs is extremely important in a wide variety of applications, from aerial photography to security monitoring. As most UAVs are still operated by a human, e.g. in teleoperation tasks, it is important to be able to provide the operator with an easily viewable video feed to facilitate controlling the UAV or for the purpose of cinematography and aerial photography.

Camera gimbals are mechatronic devices with 2 or 3 rotational degrees of freedom that allow decoupling of the rotation of the camera from the system it is attached to. This main system can be a robot or any moving object including a simple handle, or a mount in general, for the camera operator. Gimbals for UAVs are usually



(a) (b)

Figure 5.2: Prototype gimbal for our RGB-D camera, (a) CAD model and (b) mechanical prototype.

mounted below the platform, often with an additional layer of passive damping for high frequency vibrations from the UAV's motors. Examples of such devices are depicted in Fig. 5.1.

The main purpose of camera gimbals is video stabilization, i.e., elimination of the negative effect caused by the motion of the main system on the video feed. Additionally, depending on the gimbal's construction and range of motion, it can enable active control of the camera's orientation allowing the user to change the direction of view.

Modern camera gimbals utilize specially designed brushless motors that, in contrast to simple servo motors, allow smooth motion control and rapid reactions to unwanted motions. Gimbals usually require separate controllers but UAV flight controllers also exist with built-in gimbal functionality that can control additional brushless motors directly. In any case, in order to properly control rotation, the camera's orientation must be estimated. The easiest and most common solution is the use of an additional IMU sensor attached to the camera frame. This approach enables relatively precise estimation of the camera's roll and pitch angles, but as explained earlier in Sec. 2.2.4, without additional measurements the estimated yaw angle value is subject to drift. Alternatively, rotary encoders can be used on the motors to determine the relative orientation of the camera with respect to the main system. This, however, adds cost and weight and might not be feasible for low-weight systems.

5.2.1 Depth Camera Gimbal

In Chap. 3 we showed how to expand the limited field of view of a sensor with tracking algorithms. Nevertheless, it is only possible to track objects previously visible by the sensor and to a certain extent. When the camera is fixed to the robot's frame, the range of possible directions of view is limited to the feasible platform orientations. Hence, it is impossible to orient the sensor in directions that the platform cannot face. For example, in the work by Stegagno *et al.* (2014), the authors employed an active "pan-scanning", alternative left and right rotations of their quadrotor platform, to increase the FOV in the horizontal direction.

To test the positive impact of a stabilized video feedback on the performance of UAV teleoperation and additional benefits related to the ability of controllable direction of view, we designed and built the device depicted in Fig. 5.2 for our depth camera. It comprises of a relatively lightweight, glass fiber reinforced 3D printed frame and appropriate gimbal-designated brushless motors. It is important to properly balance such a device, i.e., mount the camera in a way that all the rotation axes intersect close to the system's center of mass and the motors effort is optimized. As most modern CAD software can estimate the total mass and inertia assuming properly assigned material properties to every part, this is not a complicated task.

To control our device we used a BaseCam SimpleBGC[1] 3-axis controller. This controller also has analog and digital (serial) connections to enable inputs of the desired camera orientation. The serial port can also be used to read the status of the camera orientation and controller parameters.

We used the software provided with the controller to adjust its PID gains to the dynamics of our components (the camera and frame inertia, and motors) with the trial-and-error method. In tests with the prototype shown in Fig. 5.2b, where the structure was rotated by hand to simulate the motion of a quadrotor, the stabilized video was almost indistinguishable from a still camera view (except for the yaw rotation, which was not stabilized).

5.2.2 Conclusions

In the final design of our platform we decided to forgo the additional stabilization equipment. The obstacle avoidance experiments presented in Chap. 4 proved that within the assumptions related to the platform's motion in the teleoperation tasks, the performance with a rigid camera and an appropriate tracking algorithm is very satisfactory. The bulkiness, additional weight and problematic cable management outweighted the benefits of the active stabilization device.

In terms of video feedback improvement, we propose an alternative approach based on software stabilization. As presented in the following section, that approach

[1] `https://www.basecamelectronics.com/simplebgc32bit/`

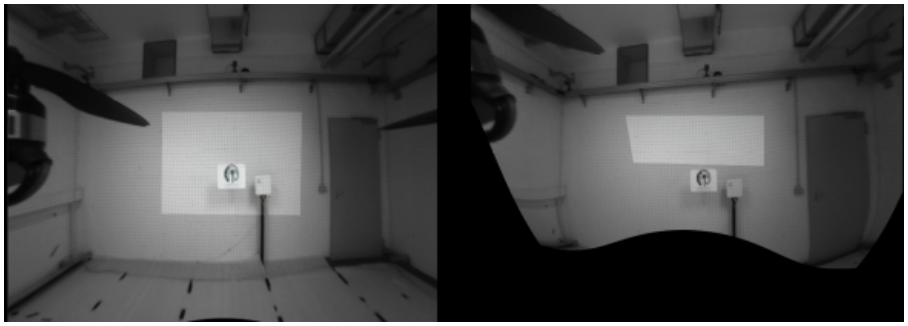


Figure 5.3: The RGB camera's narrow FOV (bright) overlaid on the fisheye's wide FOV, with little (left) and large (right) compensations for quadcopter movement.

adds no physical weight to the system, and therefore offers clear benefits with respect to low computational power requirements.

5.3 IMU-based Digital Image Stabilization

While some UAVs have the capacity to carry mechanically stabilized camera equipment, weight limits or other problems may make mechanical stabilization impractical. As a result many UAVs rely on fixed cameras to provide a video stream to an operator or observer. With a fixed camera, the video stream is often unsteady due to the multirotor's movement from wind and acceleration. These video streams are often analyzed by both humans and machines, and the unwanted camera movement can cause problems for both. For a human observer, unwanted movement may simply make it harder to follow the video, while for computer algorithms, it may severely impair an algorithm's intended function.

There has been significant research on how to stabilize videos using feature tracking to determine camera movement, which in turn is used to manipulate frames and stabilize the camera stream. We believe, however, that this process could be greatly simplified by using data from a UAVs on-board inertial measurement unit to stabilize the camera feed. In this section we present an algorithm for video stabilization based only on the IMU data from a UAV platform. Our results show that our algorithm successfully stabilizes the camera stream with the added benefit of requiring less computational power than a feature-tracking approach.

5.3.1 Literature Overview

The logical solution for video stabilization on smaller aerial vehicles is using software since it adds no physical weight to the UAV and many stabilization algorithms already exist. Most existing software used for video stabilization, however, relies on complex and computationally expensive feature tracking algorithms (Thillainayagi

and Kumar, 2016; Shen *et al.*, 2009; Mingkhwan and Khawsuk, 2017; Ryu *et al.*, 2009). In these algorithms, image features in consecutive frames are matched with the features in a reference frame. A comparison of the location of known features between frames is then used to determine how the camera has moved between frames, and to stabilize the video. Research has focused on efficient and effective feature detection algorithms such as SIFT (Thillainayagi and Kumar, 2016) and SURF (Mingkhwan and Khawsuk, 2017), as well as methods which allow for the calculation of an affine-transform matrix comparing separate frames in the video (Thillainayagi and Kumar, 2016; Shen *et al.*, 2009; Mai *et al.*, 2012; Mingkhwan and Khawsuk, 2017; Schwertfeger *et al.*, 2011). Other approaches have included algorithms based on particle filters (Zhu *et al.*, 2015), linear and curve filters (Wang *et al.*, 2012), and iFMI spectral registration (Schwertfeger *et al.*, 2011).

In this work, however, we propose that the use of feature detection is unnecessarily complex and computationally expensive for video stabilization given the limited computational power on-board most UAVs. The goal of every stabilization algorithm is to determine the type and magnitude of unwanted camera movement. Once camera movement is determined, individual frames can be manipulated to stabilize the video. As virtually all UAVs carry IMU units, we believe it is easier and faster to calculate the orientation of the camera from the IMU's accelerometer and gyroscope rather than by using feature detection.

IMU data is frequently used in active optical and mechanical stabilization devices to manipulate either the camera's orientation, or the position of the camera's lens. The use of IMU data, however, has not generally been used in digital stabilization algorithms. Although some studies have used IMU data for image stabilization, they tend to be hybrid approaches that combine feature detection with IMU data for faster and more accurate results. Most notably Ryu *et al.* (2009) created an algorithm that uses IMU data to assist, speed up, and improve the accuracy of the KTL tracker algorithm. It does not seem, however, that anyone has entirely forgone feature recognition for digital video stabilization on UAVs.

Further examples of hybrid stabilization approaches are seen in industry. For example, Axis Communications, which develops, and sells security cameras, has developed a hybrid approach using gyroscope data in combination with feature tracking to stabilize security camera footage in real time[2]. Karpenko *et al.* (2011) created an algorithm for stabilizing iPhone videos, and adjusting for rolling shutter distortion, in real time using data from the iPhone's gyroscope. By calculating the difference in rotation between frames, the algorithm utilizes an iterative method, however, rather than simply calculating the orientation of the camera at any given time with respect to a reference frame.

To the best of our knowledge, there is only one work that proposes the use of rotation obtained from an attitude and heading reference system (AHRS) by

[2]https://www.axis.com/fi/en/technologies/axis-electronic-image-stabilization

Wiriyaprasat and Ruchanurucks (2015). Although the presented method is similar in its core to our method, it lacks real world experiments and evaluation.

We believe that the only work that has demonstrated an IMU based stabilization method on-board a quadcopter was the limited horizontal and vertical compensation algorithm created by Stegagno *et al.* (2014). This work involved a teleoperated quadcopter that would actively "pan-scan" by rotating left and right to increase the camera's limited field of view (FOV) for the operator. By gathering data from the camera's on-board IMU, the quadcopter could display each image frame in the correct position along a horizontal reference line, while the images were also stabilized vertically. Since the quadcopter was limited to three degrees of freedom (it was unable to "roll" left or right) there was no compensation for roll movement in the algorithm, a correction that is very important for flights outside of the lab. Additionally, the image was only translated, and not warped to adjust for the perspective of the operator.

5.3.2 Motivation and Methodology

Our platform setup consists of a quadcopter operated remotely by a human operator, and was developed to engage in obstacle avoidance using an RGB-D camera. During experiments described in more depth in Sec. 4.4, the camera footage was quite shaky and consequently limited the teleoperation performance of the robot. To solve this problem we wanted to create a system that stabilizes the video feed in real time. The solution is digital stabilization software since it is lightweight and, as long as the quadcopter has the necessary computing power, can be added onto an existing platform with relatively little effort. Rather than using a feature-tracking algorithm, we decided instead to use the quadrotor's orientation estimated from IMU data to manipulate video frames.

While the implementation by Karpenko *et al.* (2011) based the video stabilization on an iterative method using gyroscopes, we use only the quadcopter's immediate orientation relative to the horizontal frame to stabilize the image. Because our method does not require any feature detection or iterative methods, it uses very little computing power, and so can easily run on-board our existing quadrotor platform. In contrast to typical digital stabilization algorithms, which frequently have narrow fields of view, even discarding data to gain stability, we demonstrate our approach with a wide angle camera as illustrated in Figs. 5.3.

In contrast to Wiriyaprasat and Ruchanurucks (2015), we propose a method that uses a quadrotor's IMU to compute the camera rotation without a dedicated AHRS unit. Furthermore, we provide an implementation of the algorithm on an actual UAV with real world experiments and numerical evaluation.

In the following sections we first describe our method for image stabilization using IMU data. We then discuss the effectiveness of this form of stabilization and draw conclusions regarding the success and ultimate usefulness of this approach.

Figure 5.4: Stabilization results for the quadrotor's roll rotation. Showing the reference view (left), non-stabilized (middle) and stabilized (right) frames.

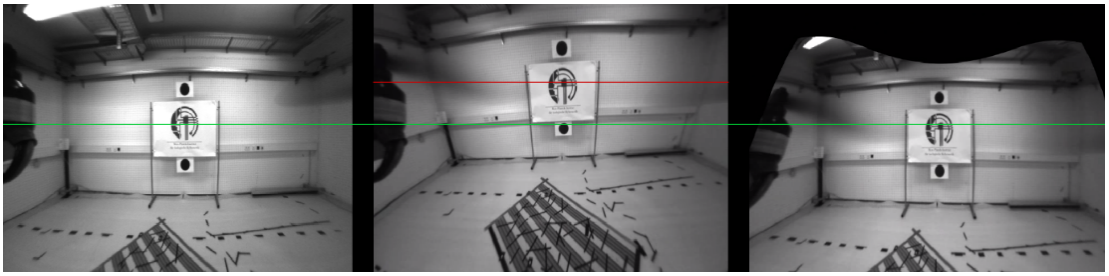


Figure 5.5: Stabilization results for the quadrotor's pitch rotation. Showing the reference view (left), non-stabilized (middle) and stabilized (right) frames.

5.3.3 Stabilization Algorithm

In this section, we consider an arbitrary UAV-camera system in which the sensor is rigidly attached to the robot's frame. In order to stabilize the image, the camera's orientation with respect to a reference frame is needed, but, as the orientation estimation is essential for control of any flying robot, this can be considered to be already known.

Quadrotor-camera System

In particular, we can take the system presented earlier in Sec. 2.2.1 as an example, however, we will show that the algorithm can be generalized to an arbitrary selection of reference frames. To summarize, the relevant coordinate frames of the system were shown in Fig. 2.1, where Q denotes the quadrotor's frame of reference, H is the corresponding horizontal frame, and C is the frame associated with the camera. In the robot-centric approach, X_H and Y_H define the forward and lateral directions of motion, respectively.

The robot's attitude is expressed as the rotation matrix between Q and H,

$$\mathbf{R}_Q^H = \mathbf{R}_y(\theta)\mathbf{R}_x(\phi), \tag{5.1}$$

where ϕ, θ denote the roll and pitch angles. Because the frame H rotates together with the robot the yaw angle ψ is set to zero. When $(\phi, \theta) = (0, 0)$, the horizontal frame H aligns with Q.

Since the camera is rigidly attached to the robot, $\mathbf{R}_C^Q$ – the rotation matrix between frame C and the quadroter frame Q, is a constant extrinsic parameter of the camera.

Desired Reference Frame

The goal of video stabilization is to make consecutive frames of a video stream appear as they would from a desired, still camera frame. Because the camera and its frame C rotate together with the quadrotor, we introduce a new horizontal camera frame C_H, such that $C_H = C$ when $(\phi, \theta) = (0, 0)$. Moreover, we define I and I_H to be image planes associated with the camera frames C and C_H, respectively.

In order to stabilize the image, it is necessary to find a mapping function that will transform points from the current image plane I_C to the desired, stabilized horizontal image plane I_H, $f_{map} : I \mapsto I_H$.

In the pinhole camera model, described in Sec. 2.2.4, the transformation

$$\begin{aligned} \begin{bmatrix} u_I \\ v_I \\ 1 \end{bmatrix} &= \mathbf{K} \left[\mathbf{R}_W^C | {}^C\mathbf{t}_W \right] \begin{bmatrix} x_W \\ y_W \\ z_W \\ 1 \end{bmatrix}, \\ \mathbf{p}'_I &= \mathbf{K} \left[\mathbf{R}_W^C | {}^C\mathbf{t}_W \right] \mathbf{p}'_W, \end{aligned} \tag{5.2}$$

defines the projection between homogeneous coordinates of an arbitrary world point $\mathbf{p}'_W$ and the homogeneous coordinates of the corresponding image point $\mathbf{p}'_I$. $\mathbf{K}$ represents the camera matrix (intrinsic parameters) and $\left[\mathbf{R}_W^C | {}^C\mathbf{t}_W \right]$ is a joint rotation-translation matrix.

Similarly, the camera-horizontal frame is related to $\mathbf{p}'_W$ by

$$\mathbf{p}'_{I_H} = \mathbf{K} \left[\mathbf{R}_W^{C_H} | {}^{C_H}\mathbf{t}_W \right] \mathbf{p}'_W. \tag{5.3}$$

Next, given that

$$\mathbf{R}_W^C = \mathbf{R}_{C_H}^C \mathbf{R}_W^{C_H}, \tag{5.4}$$

we get

$$\mathbf{p}'_I = \mathbf{K} \left[\mathbf{R}_{C_H}^C \mathbf{R}_W^{C_H} | {}^C\mathbf{t}_W \right] \mathbf{p}'_W, \tag{5.5}$$

and consequently

$$\begin{aligned}\mathbf{p}'_W &= \left[\mathbf{R}^{C}_{C_H}\mathbf{R}^{C_H}_{W}|{}^{C}\mathbf{t}_W\right]^{-1}\mathbf{K}^{-1}\mathbf{p}'_I \\ &= \left[\left(\mathbf{R}^{C}_{C_H}\mathbf{R}^{C_H}_{W}\right)^{-1}|{}^{C}\mathbf{t}_W^{-1}\right]\mathbf{K}^{-1}\mathbf{p}'_I \\ &= \left[\mathbf{R}_W^{C_H\,-1}\mathbf{R}_{C_H}^{C\;\;-1}|{}^{C}\mathbf{t}_W^{-1}\right]\mathbf{K}^{-1}\mathbf{p}'_I.\end{aligned} \tag{5.6}$$

Using standard relations for $\mathbf{R} \in SO(3)$ rotation matrices

$$\begin{aligned}\mathbf{R}_A^{B^{-1}} = \mathbf{R}_A^{B^{\mathrm{T}}} = \mathbf{R}_B^A, \\ \mathbf{R}_B^A\mathbf{R}_C^B = \mathbf{R}_C^A,\end{aligned} \tag{5.7}$$

and translation vectors

$${}^{A}\mathbf{t}_B^{-1} = -{}^{A}\mathbf{t}_B, \tag{5.8}$$

we can obtain

$$\mathbf{p}'_W = \left[\mathbf{R}^{W}_{C_H}\mathbf{R}^{C_H}_{C}| - {}^{C}\mathbf{t}_W\right]\mathbf{K}^{-1}\mathbf{p}'_I, \tag{5.9}$$

and then substitute ((5.9)) into ((5.3)) to get

$$\begin{aligned}\mathbf{p}'_{I_H} &= \mathbf{K}\left[\mathbf{R}^{C_H}_{W}|{}^{C_H}\mathbf{t}_W\right]\left[\mathbf{R}^{W}_{C_H}\mathbf{R}^{C_H}_{C}| - {}^{C}\mathbf{t}_W\right]\mathbf{K}^{-1}\mathbf{p}'_I \\ &= \mathbf{K}\left[\mathbf{R}^{C_H}_{W}\mathbf{R}^{W}_{C_H}\mathbf{R}^{C_H}_{C}|{}^{C_H}\mathbf{t}_W - {}^{C}\mathbf{t}_W\right]\mathbf{K}^{-1}\mathbf{p}'_I \\ &= \mathbf{K}\left[\mathbf{R}^{C_H}_{C}|\mathbf{0}\right]\mathbf{K}^{-1}\mathbf{p}'_I.\end{aligned} \tag{5.10}$$

Equation (5.10) shows that since $O_C \equiv O_{C_H}$, we only need the relative orientation between frames $\mathbf{R}_C^{C_H}$ and the camera matrix $\mathbf{K}$ in order to re-project points from image plane I onto I_H.

Furthermore, we can define

$$\mathbf{K}\left[\mathbf{R}^{C_H}_{C}|\mathbf{0}\right]\mathbf{K}^{-1} = \mathbf{H}_I^{I_H}, \tag{5.11}$$

where $\mathbf{H}_I^{I_H}$ is the homography matrix that represents the difference in perspective between two cameras that are both viewing the same object.

Construction of the Homography Matrix

The homography matrix in Eq. (5.10) can account for rotation and translation, however we are only interested in compensating for the rotation of the camera. The rotation of the robot can be obtained from Eq. (5.1) using the angles calculated by the complementary filter, Eq. (2.30). Matrix $\mathbf{R}_Q^H$, however, only represents the orientation of the robot relative to the horizontal frame, and we need to calculate the orientation of the camera to the camera's horizontal frame $\mathbf{R}_C^{C_H}$ in order to stabilize the image. Using Eq. (5.7) we can define the projection of quadrotor's

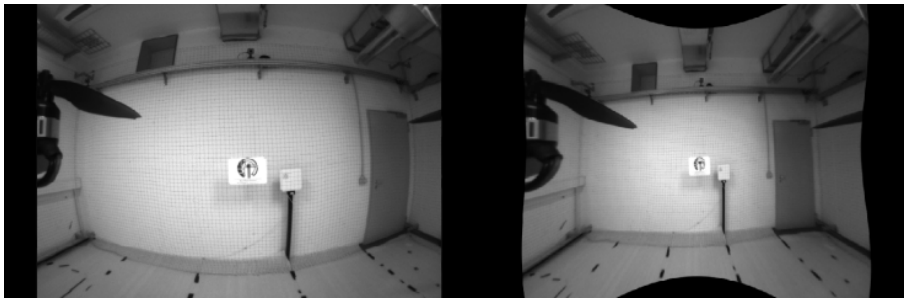


Figure 5.6: An example image (left) before and (right) after distortion rectification.

rotation to the camera frame

$$\mathbf{R}_C^{C_H} = \mathbf{R}_H^{C_H}\mathbf{R}_Q^H\mathbf{R}_C^Q, \tag{5.12}$$

where $\mathbf{R}_H^{C_H}$ is the only unknown, and $\mathbf{R}_C^Q$, as defined earlier, represents the extrinsic rotation of the camera.

As defined earlier, when $(\phi, \theta) = (0, 0)$ $Q = H$, $C = C_H$ and consequently, $\mathbf{R}_H^{C_H} = (\mathbf{R}_C^Q)^{\mathrm{T}}$. Because the camera is fixed in relation to the quadcopter, $\mathbf{R}_C^Q$ is constant, thus, this relation also holds for $(\phi, \theta) \neq (0, 0)$. Hence

$$\mathbf{R}_C^{C_H} = \left(\mathbf{R}_C^Q\right)^{\mathrm{T}}\mathbf{R}_Q^H\mathbf{R}_C^Q. \tag{5.13}$$

Once we have constructed the homography matrix we are able to use it to reproject the current frame onto the stabilized image plane I_H.

Camera Calibration and Distortion Correction

In order to obtain the camera matrix, we used a classical calibration method based on a chess board pattern of a known size to estimate the camera's intrinsic parameters and the lens distortion coefficients. Based on the pinhole camera model, our projection matrix is

$$\mathbf{K} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{5.14}$$

where f_x and f_y are the camera focal lengths, and c_x and c_y represent the optical center in pixel coordinates.

Because the wide field of view we use introduces barrel distortions onto images, the first step of our algorithm is a distortion correction. Since this is a standard process (Hartley and Zisserman (2004)) we do not review the method. An example of the distortion correction for one frame is depicted in Fig. 5.6.

Figure 5.7: Our quadrotor platform with a new camera with a wide field of view lens.

5.3.4 Experimental Setup and Results

For any large rotations of the quadcopter and camera, objects of importance can leave the field of view of a standard camera. The effects of this problem can be minimized by using a wide angle lens. For these reasons, we equipped our robot with the Intel RealSense ZR300 camera which, among other features, provides color and grayscale images with standard and wide angle fields of view, respectively. Our quadrotor setup with the new camera is shown in Fig. 5.7, the camera is fixed to the robot's frame such that it's optical axis Z_C points toward the front of the robot.

A comparison of the two fields of view is presented in Fig. 5.3 as overlaid images. The fisheye lens greatly increases the area visible to the operator and a central target is still within the field of view even after large rotations.

For performance validation purposes, we put two circular targets in the view of the camera. By detecting the circles' centers we can relate the camera roll and pitch rotations to the corresponding rotation of the line between the circles and the vertical translation of the point between the circles, respectively.

Experiments

In order to validate our algorithm, we have performed several test flights with aggressive changes in the direction of motion, resulting in rapid and large changes to the robot's orientation. In each of these tests, the camera feed together with the IMU measurements were recorded. Real flight examples of the quadrotor's roll and pitch rotations and their effect on the camera images are shown in Figs. 5.4 and 5.5. It is clear by inspection alone that our algorithm successfully stabilizes the video feed.

For quantitative evaluation, we recorded a non-flight set of data in which the roll and pitch angles could be decoupled from the other movements of the robot. It also allowed us to obtain higher rotation angles than would occur in a real flight. In

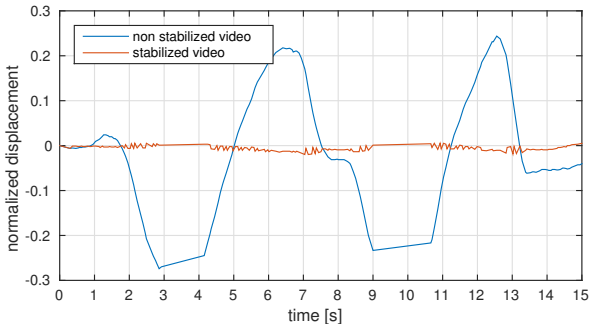


Figure 5.8: Illustration of pitch stabilization using the normalized vertical target translation of the video without (blue) and with (red) stabilization.

these tests, we were able to evaluate the functioning of our algorithm in a controlled manner. The quality of the roll and pitch stabilization is shown in Figs. 5.8 and 5.9, respectively. The displacement in Fig. 5.8 is normalized as a fraction of the frame height.

While there is some residual noise in the stabilized results displayed in Figs. 5.8 and 5.9, it is negligible when compared to the non-stabilized video. For pitch correction, the maximum displacement of the target center in the stabilized image was 1.9% of the frame height, while the maximum displacement of the non-stabilized frame was 27.4% of the frame height. The results for the rotational correction are similar. In the stabilized video, the target rotates a maximum of 3.9°, while the maximum rotation in the non-stabilized video is 64.0°.

It is also important to note that in both of these experiments the robot was manually rotated to pitch and roll angles that would never be seen in normal flight in order to test the limits of the IMU data and algorithm. Thus the residual noise experienced in flight would likely be much smaller than the maxima we see in these tests. In fact, we measured the pitch angle to the point that one of the circle-targets partially left the field of view and so data was not collected until it returned within the field of view. This explains the straight lines from seconds 3-4 and 9-10.5 in Fig. 5.8.

5.3.5 Discussion

For our purposes, the idea of digital stabilization is to manipulate a video stream so that every frame appears from the same perspective, without any shaking or swaying. Ideally this would mean that if a target were in the horizontal frame, the UAV could be rotated any amount around the roll and pitch axis and we would still see the same image.

It is clear that our stabilization algorithm has created a much more viewable image for the operator in the sense that we have removed much of the jitter and

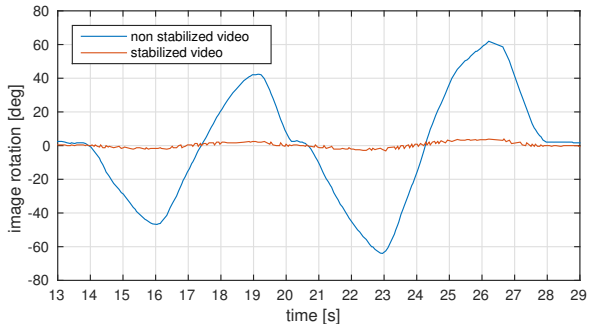


Figure 5.9: Illustration of roll stabilization using the target rotation of the video without (blue) and with (red) stabilization.

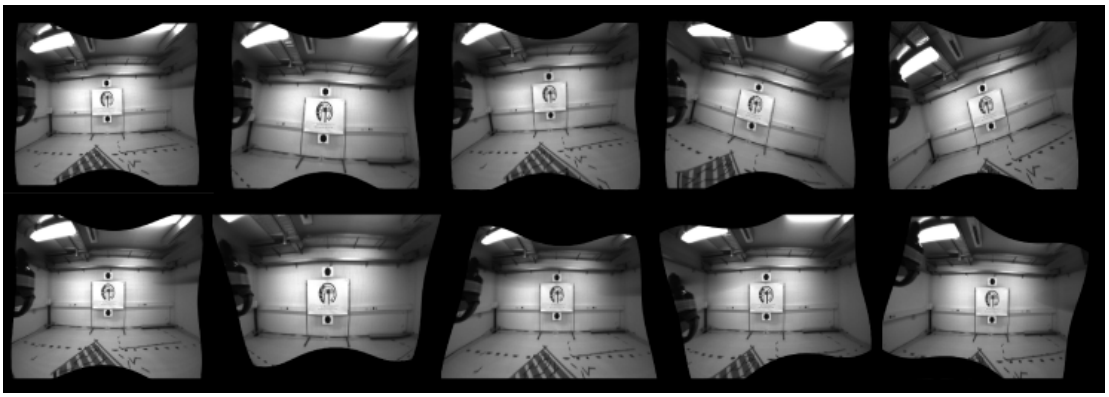


Figure 5.10: A set of five frames from our test flight with the non-stabilized (top) and the corresponding stabilized (bottom) frames.

rotational errors that come from robot movement. As seen in Fig. 5.10, even with large roll and pitch changes, the central target remains in the same orientation and vertical position throughout the video. Not only does our algorithm successfully stabilize the video, but it does so relying only on the on-board IMU data of our quadcopter. Just as important is that the approach works equally well for the expanded field of view provided by the fisheye camera. The limitation of digital stabilization to small fields of view in order to maintain stability is a major problem of the field. Theoretically, an expanded field of view leads to more features that a feature-based algorithm must process for each frame, slowing the video-processing time. Because our algorithm is not based on feature tracking, an expanded field of view does not slow the algorithm, and we can arbitrarily scale the field of view with the only limitations on processing speed being the actual image size.

Having seen the success of using a wide angle lens, it is also interesting to consider the possibilities that could come from gathering an even wider FOV. In the past few years, companies, such as Ricoh and GoPro, have begun to manufacture cheap

and lightweight 360° cameras that can capture video of the entire world around them. These cameras usually function by stitching together the images from two opposite facing fisheye cameras. If it is possible to capture the entire world around the quadcopter, then theoretically we can stabilize for any magnitude of roll and pitch that the quadcopter undergoes, while the field of view passed to the operator remains unchanged.

5.3.6 Conclusions and Future Works

Relying only on IMU data, our algorithm successfully stabilizes the video feed for large roll and pitch variations of the UAV. Thanks to its low computational requirements, our algorithm can also be employed on-board UAVs more easily than feature tracking methods. Lastly, using a wide angle lens enables the retention of a larger field of view after stabilization than is provided by our RGB camera's raw image.

Further improvements could be achieved by using a camera with an even larger field of view, such as a 360° camera. This could allow for stabilization in nearly any orientation of the UAV with no data loss. Additionally, a comparison between our work and a feature tracking method on the same dataset could be performed with a more thorough method for performance evaluation, e.g., the inter-frame transformation fidelity Aguilar and Angulo (2014).

5.4 6 DOF Quadrotor

In this section, we propose a novel quadrotor design in which the tilt angles of the propellers with respect to the quadrotor body are simultaneously controlled by two additional actuators by employing the parallelogram principle. Since the velocity of the controlled tilt angles of the propellers does not appear directly in the derived dynamic model, the system cannot be static feedback linearized. Nevertheless, the system is linearizable at a higher differential order, leading to a dynamic feedback linearization controller. Simulations confirm the theoretical findings, highlighting the improved motion capabilities with respect to standard quadrotors.

One of the limitations of quadrotor UAVs (and other multirotors with coplanar propellers) is their intrinsic underactuation. Belonging to the family of underactuated systems the lateral motion of such platforms is strongly coupled with their orientation and, consequently, it is not possible to track an arbitrary 6D trajectory in space. A change in position or disturbance counteraction of such a UAV involves a change in its orientation. Although this may not impose any significant restraint in open-air flights, it might be crucial when the precision of control matters, or in presence of external disturbances, e.g., an abrupt or unwanted change in orientation might involve the need of incommodious rectification of directional sensor

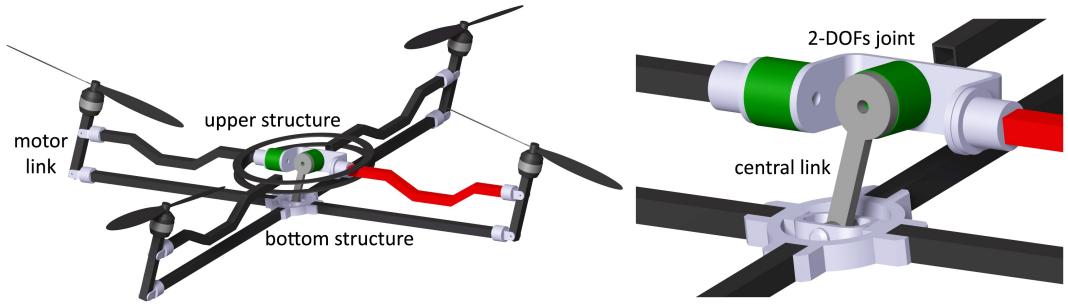


Figure 5.11: Top: conceptual CAD model of the proposed platform; bottom: the tilting mechanism in detail; in green, the two additional actuators; in gray, the joints of the platform; the red arm indicates the forward direction.

measurements. As stated before, stabilization of the visual feedback from a vehicle is an important factor affecting performance in teleoperation tasks, especially in obstacle rich environments as described in Chap. 4 of this thesis.

5.4.1 Literature Overview

Fully actuated multirotors are UAVs with additional actuators that enable control over all six degrees of freedom of such platforms. The two main concepts of this approach can be found in the literature, which are as follows.

The first solution is the employment of fixed tilted propellers (Rajappa *et al.*, 2015; Brescianini and D'Andrea, 2016), which however, requires constant counterbalance and energy dissipation of extra forces and torques. For this reason, the values of the tilt angles are usually optimized for certain range of trajectories. Another approach is to add auxiliary propellers (Gentili *et al.*, 2009) or a wing/rotor tilting mechanism, as by Ryll *et al.* (2015); Mikami and Uchiyama (2015); Kendoul *et al.* (2006). Platforms equipped with additional actuators share some drawbacks, such as loss of efficiency from the increased weight, and higher power consumption.

5.4.2 Motivation and Methodology

Inspired by the aforementioned research, we propose a novel platform with only two additional actuators that allows the simultaneous change of the orientation of all propellers. In Fig. 5.11, we show a conceptual 3D design of the platform and, in Fig. 5.12, its kinematic diagram. The main novelty, and in our opinion the most interesting property of this design, is that it allows the control of six degrees of freedom with six inputs. Thus it does not require optimization due to redundant control inputs, nor does it suffer from energy dissipation due to an internal wrench caused by counteracting actuators.

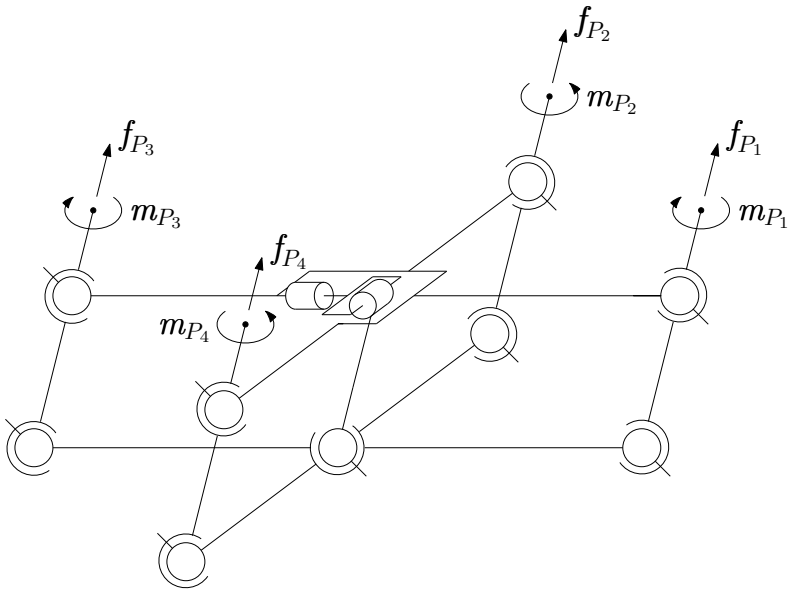


Figure 5.12: Kinematic diagram of the platform.

In Sec. 5.4.3 we describe the proposed mechanism and derive the dynamical model of our fully actuated quadrotor. In Sec. 5.4.4 we derive the equations for the dynamic feedback linearization and propose a controller for the platform. Sec. 5.4.5 presents simulation results which highlight the improved motion capabilities with respect to a standard quadrotor and Sec. 5.5 concludes this chapter.

5.4.3 Platform Design

In order to control a classical quadrotor to perform a motion along a given trajectory, the accumulated thrust of the four propellers must be directed towards the desired direction while simultaneously counteracting the gravity force and other external disturbances (Mahony *et al.*, 2012). Although the rotation of the platform can be induced with the resulting torque, only the angle around the vertical axis (usually Z) can be controlled independently whilst the other two are used to orient the thrust as stated above.

In order to decouple the lateral motion from the tilt angles of the platform and thus regain the control over the two missing degrees of freedom, we have designed a model with a propeller tilting mechanism that can change the orientation of the produced thrust. This tilting mechanism, depicted in Figs. 5.11 and 5.12, is made of a 2-DOF actuated joint, a central link, and a bottom structure (the four bottom arms in the figures) that transfer the motion to the motor links. The central link, bottom structure, and the platform's arms and motor links form four parallelograms coupled with the central link allowing the simultaneous tilt of all the propellers.

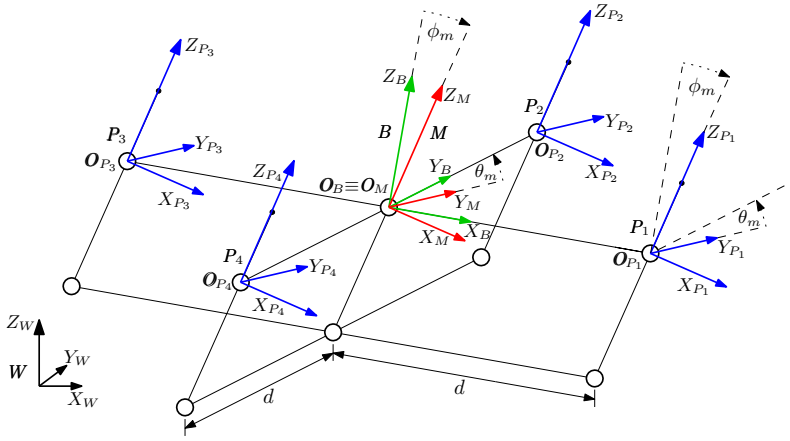


Figure 5.13: Simplified diagram of the platform with depicted coordinate frames: W in black, B in green, M in red and $P_i, i = 1, \ldots, 4$ in blue.

In the derivation process of the dynamic model of our platform, presented later in this section, we assume that the motion of the bottom structure is negligible with respect to the dynamics of the UAV. This assumption is motivated by the fact that the broad majority of the system's components (e.g., battery, computational unit, etc.), and thus its mass, can be located around the center of the upper structure. Due to the mechanical constraints of the design presented in Fig. 5.11, the tilting angles are limited to the interval $[-\frac{\pi}{6}, \frac{\pi}{6}]$. The effect of this constraint will be further discussed in Sec. 5.4.5.

Notation and Definitions

We represent the relevant quantities in the following reference frames: depicted in Fig. 5.13, a global inertial frame of reference $W : \{O_W, X_W, Y_W, Z_W\}$; a moving body frame $B : \{O_B, X_B, Y_B, Z_B\}$ attached to the quadrotor at its center of mass, ideally the middle point between the propellers; the tilting mechanism frame $M : \{O_M, X_M, Y_M, Z_M\}$ with $O_M \equiv O_B$; and a set of four frames attached to the centers of propellers $P_i : \{O_{P_i}, X_{P_i}, Y_{P_i}, Z_{P_i}\},\ i = 1 \ldots 4$.

To represent the relative pose of two arbitrary frames we use the notation presented in Sec. 2.2.1. Therefore, the platform's position and orientation in the global frame W is

$$\begin{aligned} \boldsymbol{p} &= {}^W\boldsymbol{p}_B = [p_x,\ p_y,\ p_z]^{\mathrm{T}}, \\ \boldsymbol{R}_B &= \boldsymbol{R}_B^W = \boldsymbol{R}_z({}^W\psi_B)\boldsymbol{R}_y({}^W\theta_B)\boldsymbol{R}_x({}^W\phi_B), \end{aligned} \tag{5.15}$$

where ${}^W\phi_B = \phi_B$, ${}^W\theta_B = \theta_B$, ${}^W\psi_B = \psi_B$ denote the roll, pitch and yaw rotation angles of the platform, and $\boldsymbol{R}_x(\cdot)$, $\boldsymbol{R}_y(\cdot)$, $\boldsymbol{R}_z(\cdot)$, the canonical rotation matrices representing the elemental rotations around the X, Y and Z axes.

In a classical quadrotor the thrust is directed along the vertical axis Z_B of the

body frame. In our design, by reorienting the propellers with a tilting mechanism, the force can be applied independently of the body rotation along the new direction Z_M defined such that

$$\boldsymbol{R}_M^B = \boldsymbol{R}_y(\theta_m)\boldsymbol{R}_x(\phi_m), \tag{5.16}$$

where ${}^B\phi_M = \phi_m$ and ${}^B\theta_M = \theta_m$ are the roll and pitch angles of the tilting mechanism. As a result of the four coupled parallelograms that link the tilting mechanism with the motor links, the orientation of each i-th propeller $\boldsymbol{R}_{P_i}^B$ is identical and equal to the orientation of the tilting mechanism

$$\boldsymbol{R}_{P_i}^B = \boldsymbol{R}_M^B, \qquad i = 1 \dots 4, \tag{5.17}$$

and their positions are defined as

$$\boldsymbol{o}_{P_i} = {}^B\boldsymbol{o}_{P_i} = \boldsymbol{R}_z\left((i-1)\pi/2\right)\begin{bmatrix} d \\ 0 \\ 0 \end{bmatrix}, \qquad i = 1 \dots 4, \tag{5.18}$$

with d being the arm length - the distance from the centers of the propellers to the center of mass of the UAV.

Dynamic Model

Exploiting the common Newton-Euler approach, we start with defining the forces and torques acting on the platform. We assume the following simplified model of a propeller:

$$\begin{cases} \boldsymbol{f}_{P_i} = (0\ 0\ k_f \bar{w}_i|\bar{w}_i|)^{\mathrm{T}}, & k_f > 0, \\ \boldsymbol{m}_{P_i} = (0\ 0\ -k_m \bar{w}_i|\bar{w}_i|)^{\mathrm{T}}, & k_m > 0, \end{cases} \tag{5.19}$$

in which the produced thrust $\boldsymbol{f}_{P_i}$ and the reaction moment $\boldsymbol{m}_{P_i}$ in the propeller frame $\boldsymbol{P}_i$, $i = 1 \dots 4$ are proportional to the signed square spinning velocity of the rotor $w_i = \bar{w}_i|\bar{w}_i|$, $i = 1 \dots 4$ with factors k_f and $-k_m$, respectively. The factors k_f and k_m are the propeller's thrust and torque coefficients, which can be obtained experimentally as shown by (Ryll *et al.*, 2015). Although this approach does not model any first- or second-order aerodynamic effects (e.g., the different thrust between the advancing and retreating blades or blade flapping), as validated by Ryll *et al.* (2015), Eq. (5.19) sufficiently accurately captures the dynamics of a propeller. In addition we assume that the actuators of the tilting mechanism have high gain and fast dynamics.

Hence, we can obtain the propellers total thrust ${}^B\boldsymbol{t} \in \mathbb{R}^3$ and torque ${}^B\boldsymbol{\tau} \in \mathbb{R}^3$

acting on the center of mass

$$ {}^{B}\boldsymbol{t} = \sum_{i=1}^{4} \boldsymbol{R}_{P_i}^{B} \boldsymbol{f}_{P_i}, \tag{5.20} $$

$$ {}^{B}\boldsymbol{\tau} = \sum_{i=1}^{4} \boldsymbol{R}_{P_i}^{B} \boldsymbol{m}_{P_i} + \sum_{i=1}^{4} \left({}^{B}\boldsymbol{o}_{P_i} \times \boldsymbol{R}_{P_i}^{B} \boldsymbol{f}_{P_i} \right). \tag{5.21} $$

Substituting (5.20) and (5.21) to the Newton-Euler set of equations

$$ \begin{aligned} \boldsymbol{R}_B \, {}^{B}\boldsymbol{t} &= m \left(\ddot{\boldsymbol{p}} - \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} \right), \\ {}^{B}\boldsymbol{\tau} &= \boldsymbol{I}_B \dot{\boldsymbol{\omega}}_B + \boldsymbol{\omega}_B \times \boldsymbol{I}_B \boldsymbol{\omega}_B, \end{aligned} \tag{5.22} $$

yields the dynamic model of our system:

$$ \begin{aligned} \begin{bmatrix} \ddot{\boldsymbol{p}} \\ \dot{\boldsymbol{\omega}}_B \end{bmatrix} &= \begin{bmatrix} \boldsymbol{g} \\ \mathbf{c} \end{bmatrix} + \begin{bmatrix} \frac{1}{m}\boldsymbol{R}_B & \mathbf{0} \\ \mathbf{0} & \boldsymbol{I}_B^{-1} \end{bmatrix} \begin{bmatrix} \frac{\partial\, {}^{B}\boldsymbol{t}}{\partial \boldsymbol{w}} & \frac{\partial\, {}^{B}\boldsymbol{t}}{\partial \boldsymbol{\omega}_m} \\ \frac{\partial\, {}^{B}\boldsymbol{\tau}}{\partial \boldsymbol{w}} & \frac{\partial\, {}^{B}\boldsymbol{\tau}}{\partial \boldsymbol{\omega}_m} \end{bmatrix} \begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{\omega}_m \end{bmatrix} = \\ &= \begin{bmatrix} \boldsymbol{g} \\ \mathbf{c} \end{bmatrix} + \begin{bmatrix} \frac{1}{m}\boldsymbol{R}_B & \mathbf{0} \\ \mathbf{0} & \boldsymbol{I}_B^{-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{F}_m & \mathbf{0} \\ \boldsymbol{\tau}_m & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{\omega}_m \end{bmatrix} = \\ &= \boldsymbol{f} + \boldsymbol{J}_R \begin{bmatrix} \boldsymbol{J}_m & \mathbf{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{\omega}_m \end{bmatrix} = \boldsymbol{f} + \boldsymbol{J} \begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{\omega}_m \end{bmatrix}, \end{aligned} \tag{5.23} $$

where m is the total mass of the platform, $\boldsymbol{I}_B \in \mathbb{R}^{3\times3}$ is the constant, diagonal and positive-definite inertia matrix, and $\boldsymbol{\omega}_B = [\omega_{B_x}\ \omega_{B_y}\ \omega_{B_z}]^{\mathrm{T}}$ is the angular velocity of the UAV in B,

$$ \boldsymbol{f} = [\boldsymbol{g}^{\mathrm{T}}\ \boldsymbol{c}^{\mathrm{T}}]^{\mathrm{T}} = [(0\ 0\ -g)^{\mathrm{T}}\ (-\boldsymbol{I}_B^{-1}(\boldsymbol{\omega}_B \times \boldsymbol{I}_B \boldsymbol{\omega}_B))^{\mathrm{T}}]^{\mathrm{T}} \tag{5.24} $$

expresses the gravity force acting on the system and the Coriolis term, $[\boldsymbol{w}^{\mathrm{T}}\ \boldsymbol{\omega}_m^{\mathrm{T}}]^{\mathrm{T}} = [(w_1\ w_2\ w_3\ w_4)^{\mathrm{T}}\ (\dot{\phi}_m\ \dot{\theta}_m)^{\mathrm{T}}]^{\mathrm{T}}$ is the control input of the system, and

$$ \boldsymbol{F}_m = \frac{\partial^{B}\boldsymbol{t}}{\partial\, \boldsymbol{w}} = \begin{bmatrix} k_f c_{\phi_m} s_{\theta_m} & k_f c_{\phi_m} s_{\theta_m} & k_f c_{\phi_m} s_{\theta_m} & k_f c_{\phi_m} s_{\theta_m} \\ -k_f s_{\phi_m} & -k_f s_{\phi_m} & -k_f s_{\phi_m} & -k_f s_{\phi_m} \\ k_f c_{\phi_m} c_{\theta_m} & k_f c_{\phi_m} c_{\theta_m} & k_f c_{\phi_m} c_{\theta_m} & k_f c_{\phi_m} c_{\theta_m} \end{bmatrix}, \tag{5.25} $$

$$\boldsymbol{\tau}_m = \frac{\partial^B \boldsymbol{\tau}}{\partial \boldsymbol{w}} = \begin{bmatrix} -k_m c_{\theta_m} s_{\phi_m} & dk_f c_{\theta_m} c_{\phi_m} + k_m c_{\theta_m} s_{\phi_m} & \\ -dk_f c_{\phi_m} c_{\theta_m} + k_m s_{\phi_m} & -k_m s_{\phi_m} & \dots \\ -dk_f s_{\phi_m} - k_m c_{\phi_m} c_{\theta_m} & -dk_f c_{\phi_m} s_{\theta_m} + k_m c_{\phi_m} c_{\theta_m} & \\ & -k_m c_{\theta_m} s_{\phi_m} & -dk_f c_{\theta_m} c_{\phi_m} + k_m c_{\theta_m} s_{\phi_m} \\ \dots & dk_f c_{\phi_m} c_{\theta_m} + k_m s_{\phi_m} & -k_m s_{\phi_m} \\ & dk_f s_{\phi_m} - k_m c_{\phi_m} c_{\theta_m} & dk_f c_{\phi_m} s_{\theta_m} + k_m c_{\phi_m} c_{\theta_m} \end{bmatrix}, \tag{5.26}$$

where $c_\alpha = cos(\alpha)$ and $s_\alpha = sin(\alpha)$ for $\alpha = \{\phi_m, \theta_m\}$.

5.4.4 Control

The system obtained in (5.23) is a non-linear dynamic model, as detailed by Isidori (1995) it is always possible to statically feedback linearize such a system if the Jacobian matrix

$$\boldsymbol{J} = \begin{bmatrix} \frac{1}{m}\boldsymbol{R}_B & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{I}_B^{-1} \end{bmatrix} \begin{bmatrix} \boldsymbol{F}_m & \boldsymbol{0} \\ \boldsymbol{\tau}_m & \boldsymbol{0} \end{bmatrix} = \begin{bmatrix} \frac{1}{m}\boldsymbol{R}_B \boldsymbol{F}_m & \boldsymbol{0} \\ \boldsymbol{I}_B^{-1}\boldsymbol{\tau}_m & \boldsymbol{0} \end{bmatrix} \in \mathbb{R}^{6\times 6},$$

is invertible. Because of the two rightmost null columns of $\boldsymbol{J}$, corresponding to the input $\boldsymbol{\omega}_m$, this condition is not satisfied. Thus, we seek to invert the system at a higher differential order. Derivation of Eq. (5.23) with respect to time gives

$$\begin{aligned} \begin{bmatrix} \dddot{\boldsymbol{p}} \\ \ddot{\boldsymbol{\omega}}_B \end{bmatrix} &= \dot{\boldsymbol{f}} + \dot{\boldsymbol{J}}_R \boldsymbol{J}_{\boldsymbol{m}} \boldsymbol{w} + \boldsymbol{J}_R \dot{\boldsymbol{J}}_m \boldsymbol{w} + \boldsymbol{J}_R \boldsymbol{J}_m \dot{\boldsymbol{w}} \\ &= \begin{bmatrix} \frac{1}{m}\dot{\boldsymbol{R}}_B \boldsymbol{F}_m \boldsymbol{w} \\ \dot{\boldsymbol{c}} \end{bmatrix} + \boldsymbol{J}_R \begin{bmatrix} \frac{\partial \boldsymbol{F}_m}{\partial \phi_m}\boldsymbol{w} & \frac{\partial \boldsymbol{F}_m}{\partial \theta_m}\boldsymbol{w} \\ \frac{\partial \boldsymbol{\tau}_m}{\partial \phi_m}\boldsymbol{w} & \frac{\partial \boldsymbol{\tau}_m}{\partial \theta_m}\boldsymbol{w} \end{bmatrix} + \boldsymbol{J}_R \boldsymbol{J}_m \dot{\boldsymbol{w}} \\ &= \boldsymbol{J}_R \begin{bmatrix} \boldsymbol{F}_m & \frac{\partial \boldsymbol{F}_m}{\partial \phi_m}\boldsymbol{w} & \frac{\partial \boldsymbol{F}_m}{\partial \theta_m}\boldsymbol{w} \\ \boldsymbol{\tau}_m & \frac{\partial \boldsymbol{\tau}_m}{\partial \phi_m}\boldsymbol{w} & \frac{\partial \boldsymbol{\tau}_m}{\partial \theta_m}\boldsymbol{w} \end{bmatrix} \begin{bmatrix} \dot{\boldsymbol{w}} \\ \boldsymbol{\omega}_m \end{bmatrix} + \begin{bmatrix} \frac{1}{m}\dot{\boldsymbol{R}}_B \boldsymbol{F}_m \boldsymbol{w} \\ \dot{\boldsymbol{c}} \end{bmatrix} \\ &= \boldsymbol{J}^\star \begin{bmatrix} \dot{\boldsymbol{w}} \\ \boldsymbol{\omega}_m \end{bmatrix} + \begin{bmatrix} \frac{1}{m}\dot{\boldsymbol{R}}_B \boldsymbol{F}_m \boldsymbol{w} \\ \dot{\boldsymbol{c}} \end{bmatrix}, \end{aligned} \tag{5.27}$$

where $\boldsymbol{J}^\star$ is the extended Jacobian, $\boldsymbol{w}$ becomes an internal state of the system, and $\dot{\boldsymbol{R}}_B = [\boldsymbol{\omega}_B]_\times \boldsymbol{R}_B$ with $[\boldsymbol{\omega}_B]_\times \in so(3)$ such that $[\boldsymbol{a}]_\times \boldsymbol{b} = \boldsymbol{a} \times \boldsymbol{b}$ for $\boldsymbol{a} \in \mathbb{R}^3$, $\boldsymbol{b} \in \mathbb{R}^3$.

The system (5.27) can be feedback linearized with the reference input $[\dddot{\boldsymbol{p}}_r^{\mathrm{T}} \ \ddot{\boldsymbol{\omega}}_r^{\mathrm{T}}]^{\mathrm{T}}$ by means of the law

$$\begin{bmatrix} \dot{\boldsymbol{w}} \\ \boldsymbol{\omega}_m \end{bmatrix} = \boldsymbol{J}^{\star -1} \left(\begin{bmatrix} \dddot{\boldsymbol{p}}_r \\ \ddot{\boldsymbol{\omega}}_r \end{bmatrix} - \begin{bmatrix} \frac{1}{m}\dot{\boldsymbol{R}}_B \boldsymbol{F}_m \boldsymbol{w} \\ \dot{\boldsymbol{c}} \end{bmatrix} \right), \tag{5.28}$$

which has a solution if $\rho_{J^\star} = rank(\boldsymbol{J}^\star) = 6$, i.e.,

$$\det(\boldsymbol{J}^\star) = 8d^2 k_f^5 k_m c_{\phi_m}^2 c_{\theta_m} (w_1 + w_2 + w_3 + w_4)^2 \neq 0 \tag{5.29}$$

Condition (5.29) is satisfied for $\phi_m, \theta_m \neq \pm\frac{\pi}{2}$ and $w_i > 0,\ i = 1 \dots 4$, which is always fulfilled due to the mechanical constraints of the platform ($|\phi_m|, |\theta_m| < \frac{\pi}{6}$) and the assumption that during flight $w_i > 0,\ i = 1 \dots 4$.

In order to validate the derived system we have employed a linear trajectory tracking controller for position

$$\dddot{\boldsymbol{p}}_r = \dddot{\boldsymbol{p}}_d + \boldsymbol{K}_{p_1}(\ddot{\boldsymbol{p}}_d - \ddot{\boldsymbol{p}}) + \boldsymbol{K}_{p_2}(\dot{\boldsymbol{p}}_d - \dot{\boldsymbol{p}}) + \boldsymbol{K}_{p_3}(\boldsymbol{p}_d - \boldsymbol{p}), \tag{5.30}$$

and orientation

$$\ddot{\boldsymbol{\omega}}_r = \ddot{\boldsymbol{\omega}}_d + \boldsymbol{K}_{\omega_1}(\dot{\boldsymbol{\omega}}_d - \dot{\boldsymbol{\omega}}_B) + \boldsymbol{K}_{\omega_2}(\boldsymbol{\omega}_d - \boldsymbol{\omega}_B) + \boldsymbol{K}_{\omega_3}\boldsymbol{e}_R, \tag{5.31}$$

as a common approach for feedback linearized systems. The orientation error $\boldsymbol{e}_R$ is defined as

$$\boldsymbol{e}_R = \frac{1}{2}\left[\boldsymbol{R}_B^{\mathrm{T}}\boldsymbol{R}_d - \boldsymbol{R}_d^{\mathrm{T}}\boldsymbol{R}_B\right]_\vee, \tag{5.32}$$

with $[\cdot]_\vee$ being the inverse map from $so(3)$ to $\mathbb{R}^3$.

Controllers in Eq. (5.30) and (5.31) ensure (Isidori, 1995) an exponential convergence of the tracking error to $\mathbf{0}$ for all the aforementioned derivatives if the gain matrices $\boldsymbol{K}_{p_1}$, $\boldsymbol{K}_{p_2}$, $\boldsymbol{K}_{p_3}$, $\boldsymbol{K}_{\omega_1}$, $\boldsymbol{K}_{\omega_2}$, $\boldsymbol{K}_{\omega_3}$ assure a proper pole placement. The control problem is now defined as an output tracking problem given the desired trajectory in position $\boldsymbol{p}_d$, orientation $\boldsymbol{R}_d$, and their derivatives. It is therefore advisable that the desired trajectories are continuous and differentiable up to the 3rd order.

It is necessary, however, to estimate the platform position $\boldsymbol{p}$ and orientation $\boldsymbol{R}_B$, linear and angular velocities, $\dot{\boldsymbol{p}}$ and $\boldsymbol{\omega}_B$, and linear and angular accelerations, $\ddot{\boldsymbol{p}}$ and $\dot{\boldsymbol{\omega}}_B$. Because of the differentiation needed for the dynamic feedback linearization of the model, the actual control inputs $\boldsymbol{w}$, must be numerically integrated from the output of system (5.28), i.e., $\dot{\boldsymbol{w}}$.

The closed loop system is similar to the one described by Ryll *et al.* (2015). Due to the one-to-one correspondence between the number of inputs and outputs it is not redundant, and thus does not require optimization.

5.4.5 Simulations

In order to validate our model and controller, we have performed numerical simulations of the system in closed loop. In particular, we have tested the ability of our system to perform motions that are not feasible with a normal quadrotor, such as pure translational lateral motion and hovering with non-zero roll and pitch angles.

The trajectory controller designed in Sec. 5.4.4 requires a trajectory defined in position and its derivatives up to the third order (jerk). Therefore, the reference trajectories are computed using a fifth-order spline interpolation which generates a

smooth trajectory based on start and end poses (Khalil and Dombre, 2002).

For all of the presented simulations we used the following values of platform parameters, $m = 1\,\mathrm{kg}$,

$$\boldsymbol{I}_B = \begin{bmatrix} 0.015\,\mathrm{kg}\cdot\mathrm{m}^2 & 0 & 0 \\ 0 & 0.015\,\mathrm{kg}\cdot\mathrm{m}^2 & 0 \\ 0 & 0 & 0.025\,\mathrm{kg}\cdot\mathrm{m}^2 \end{bmatrix}, \tag{5.33}$$

$k_f = 1.6 \cdot 10^{-5}$, $k_m = 2.5 \cdot 10^{-7}$, $d = 0.3\,\mathrm{m}$. The gains of the linear feedback controllers, (5.30) and (5.31), were obtained heuristically and set to $\boldsymbol{K}_{p_1} = \boldsymbol{K}_{\omega_1} = 30\boldsymbol{I}_3$, $\boldsymbol{K}_{p_2} = \boldsymbol{K}_{\omega_2} = 600\boldsymbol{I}_3$, and $\boldsymbol{K}_{p_3} = \boldsymbol{K}_{\omega_3} = 900\boldsymbol{I}_3$, where $\boldsymbol{I}_3$ is the 3×3 identity matrix.

Here, we have selected three simulations in which the UAV performs maneuvers which are not feasible with a classical quadrotor.

Lateral motion

Figure 5.14 shows the simulation of a lateral motion with null orientation of the UAV. The desired trajectory is defined as three points such that the platform moves first 5 m along the X_W direction, then 5 m along the Y_W direction, and lastly go back to the initial position $\boldsymbol{p}_0 = [0\ 0\ 0]^{\mathrm{T}}$, while maintaining the orientation $\phi_B = \theta_B = \psi_B = 0°$. As expected, the roll and pitch angles of the mechanism, ϕ_m and θ_m respectively, behave as if they were the roll and pitch angles of a classical quadrotor, thus the body orientation can remain constant. The fluctuation of the spinning velocities w_i, $\in 1 \ldots 4$, compensates the additional effects imposed by the movement of the tilting mechanism.

Hovering with non-zero orientation

In the second simulation, whose results are shown in Fig. 5.15, the platform is set to hover in place ($\boldsymbol{p}_d = [0\ 0\ 0]^{\mathrm{T}}$) while the UAV's orientation angles, ϕ_B, θ_B and ψ_B, are manipulated. The angles, initially at ϕ_B, θ_B, $\psi_B = 0°$, are in turn brought first to 10° and then back to 0°. The change of the roll and pitch of the UAV results in a symmetric behavior of the tilting mechanism in order to compensate for them. Instead, by analogy with a classical system, the yaw angle ψ_B can be controlled independently through the input $\boldsymbol{w}$.

Complex motion

In the third simulation, Fig. 5.16, we present a point to point trajectory in which both the position and orientation of the platform are being changed. It shows that the pitch angle of the body θ_B can be set to a negative value during forward motion (and vice versa) which is not possible with a standard quadrotor.

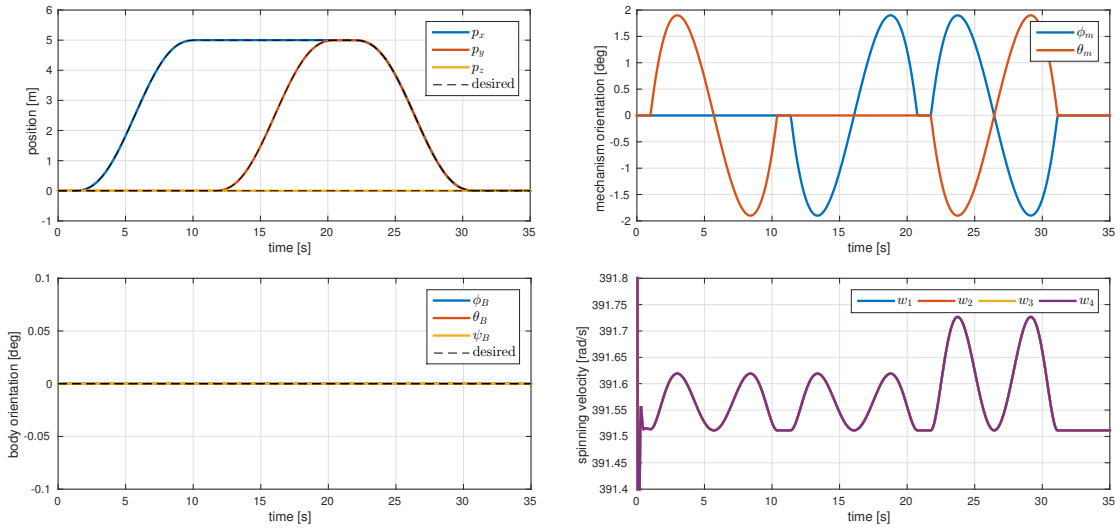


Figure 5.14: Results of the first simulation (lateral motion). Top-left: position of the UAV; top-right: orientation of the tilting mechanism; bottom-left: orientation of the UAV; bottom-right: spinning velocities of the propellers.

Remark: The tilt angles of the mechanism, ϕ_m and θ_m, depend on the desired lateral acceleration, as seen in simulation 1), and UAV orientation, ϕ_B and θ_B, as seen in simulation 2). Hence, their limit due to the mechanical constraints also limits the feasible motion. For example, it is not possible to hover with any $\phi_B > \max(\phi_m) = \frac{\pi}{6}$.

5.4.6 Conclusions

One of the main drawbacks of quadrotors is their underactuation, which causes two rotational degrees of freedom to be strongly coupled with the translational degrees of freedom. In this section, we have presented a novel design of a fully actuated quadrotor with tilting propellers. After deriving the dynamical model of the platform, we showed that it can be feedback linearized in order to design an appropriate control law. Simulations highlight the six DOFs capabilities of the platform.

5.5 Summary

Underactuation of quadrotors can have a negative impact on an operator's situational awareness when the provided visual feedback is subject to the robot's motion. In this chapter we showed three approaches to regain the missing degrees of freedom from the point of view of the camera. The first solution is to use an additional stabilization device that decouples the rotation of the camera from the motion of

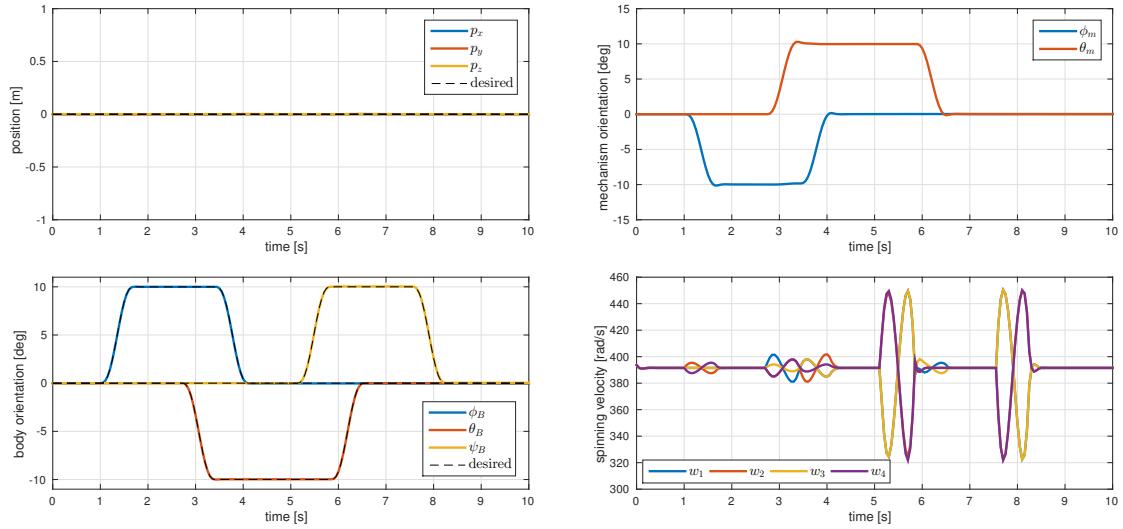


Figure 5.15: Results of the second simulation (hovering with non-zero orientation). Top-left: position of the UAV; top-right: orientation of the tilting mechanism; bottom-left: orientation of the UAV; bottom-right: spinning velocities of the propellers.

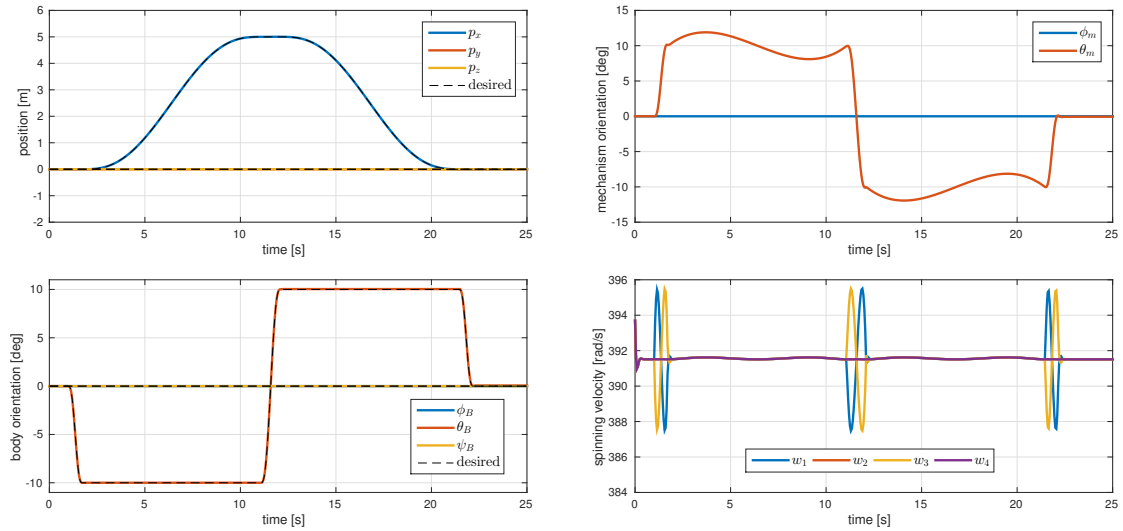


Figure 5.16: Results of the third simulation (complex motion). Top-left: position of the UAV; top-right: orientation of the tilting mechanism; bottom-left: orientation of the UAV; bottom-right: spinning velocities of the propellers.

the platform. While additional equipment increases the overall size and weight of the platform, it could be also used to reorient the sensor and extend the feasible measurement range.

The second approach is the easiest to implement as it is a software solution. It does not increase the weight of the platform, and as it uses the state of the robot which is already available it has low computational power requirements. However, as the principle of software image stabilization is based on a change of image perspective, it requires a camera with a wide field of view.

The last concept that we presented in this chapter was a novel design of a quadrotor with two additional actuators to tilt its propellers. Although our simulation results proved the validity of the design, the higher mechanical complexity of this system requires further development to transfer the results to an actual flying robot.

Chapter 6

Discussion

In this dissertation, we have focused on algorithms to aid UAV operators in their required tasks. Arguably, one of the most difficult tasks in teloperation is the navigation of UAVs in the vicinity of obstacles. The limited field of view of visual feedback and the difficult assessment of depth in monocular vision limits the operator's dimensional awareness of the robot's surroundings. In bilateral teleoperation, haptic feedback is used to complement the visual information and enhance the user's awareness with tactile feeling.

Our main motivation throughout this work was the belief that teleoperation can be simplified by endowing a robot with autonomy. By enabling the robot to sense its environment and detect objects that could possibly interfere with its trajectory and cause collisions, we have developed a navigation scheme, based on the concept of shared autonomy. In this approach, the function of the autonomous behaviours of the robot is to help the user while not limit their supervision. Moreover, the user must always be aware of the robot's actions and should be able to stop any undesired behavior. Therefore, our obstacle avoidance algorithm is primarily passive. When the user commands the robot to fly towards an obstacle, the algorithm will try to modify the commanded velocity by changing its direction or magnitude. Only in the presence of uncommanded drift of the platform will the algorithm actively compensate for this motion to minimize the chance of collision. In our approach, we give feedback about the algorithm's action to the operator through the haptic device. Hence, in the case of unwanted behavior, the user can change its input and halt the avoidance action.

The avoidance algorithm is inspired by Model Predictive Control. The algorithm uses a weighted cost function to optimize its behavior. The operator can tune the algorithm to their needs by simply adjusting the values of the weights assigned to different criteria, i.e., prioritize either the change of the desired velocity direction or its magnitude. For example, in visual inspection, the operator may wish to safely approach objects of interest rather than avoid them by flying around. Moreover, the user can further prioritize vertical over horizontal avoidance, which will result in a different behaviour of the platform in the case of obstacles of more complex shape.

The obstacle state is a grid based probabilistic representation of the obstacles in a local, cylindrical coordinate frame. The state is updated with measurements from the depth camera and, as it is a robot-centric representation, with the estimated motion of the robot. The obstacle state is limited in size to minimize memory and computational demand, thus ensuring on-board execution feasibility. When occupied cells in the obstacle state move outside the field of view of the camera, we spread their occupancy probability to adjacent cells to reflect to the lower certainty of these regions. However, we do not have the full control of that spread, as we use a cylindrical representation, in which radial and azimuth coordinates are coupled and can not be updated independently. This results in the platform not being able to pass through tight openings, e.g., windows, as we need a larger opening to accommodate for the uncertainty growth.

Supervised by human operators, teleoperated UAV platforms with additional autonomous behaviours can achieve reliability levels beyond that of fully autonomous robots given the current state of research. Obstacle avoidance algorithms, designed for close operation to obstacles and in cramped spaces, enable applications such as specialized inspection and search and rescue missions.

6.1 Future Work

The main shortcoming of our approach is that we can not ensure collision-free navigation in all directions. For example, with the camera rigidly fixed to the platform, it is not possible to look directly above the platform and ensure obstacle avoidance when commanding the robot to go upward. Additionally, as we do not discriminate between occupied/empty cells and unknown cells (not previously visible), we were forced to put additional constraints on the allowed user input. Further analysis of this issue could allow us to improve our avoidance algorithm and increase the safety of teleoperation.

In this thesis, we assumed that our approach should work with any controller for the robots velocity. Therefore, we imposed rather low limits on the platform's maximum velocity in our experiments. We also gradually decrease the platform's velocity as it gets closer to obstacles without investigating the platform's dynamics. Although our experiments were successful and proved the validity of our approach, the method could be further improved by including the dynamic model of the robot in the collision prediction and avoidance process.

Our approach was intended to be used in close proximity of static obstacles. Although the algorithm can detect and update moving obstacles, by design, it can not ensure safe navigation with highly dynamic objects. Because of the properties of the grid representation, e.g. its resolution, it is not well suited for accurate representation of motion. In order to avoid dynamic objects, our navigation system would need to be complemented with an algorithm with a different working prin-

ciple, for example, one based on optical flow estimation. As we are already using a camera that can provide depth information, it should be possible to detect the full 3D motions of objects in the field of view by combining their optical flow and depth motion.

The RGB-D camera is also a suitable sensor for visual odometry. However, because of the limited computational power of the on-board computer, we proposed to use an alternative method with a dedicated optical flow sensor. Nevertheless, we believe that with a proper software optimization, velocity estimation using the color and depth images could be potentially more reliable and versatile than the current approach.

Lastly, depth sensing technology based on structured light is the only limiting factor for outdoor applications with our navigation system. Transition to a stereo vision based depth estimation would allow us to perform experiments in outdoor settings as well as extend the usability of our method.

To summarize, potential future works could include

- further development of the obstacle state to include and discriminate between unknown cells/regions around the robot,
- modeling of robot's dynamics to improve its velocity limits,
- an extension of the obstacle avoidance algorithm to include support for dynamic obstacles,
- improvement of the sensing technology to enable outdoor applications and better state estimation.

Appendix A

Experimental Hardware

A.1 Flight Controller

A flight controller is a microcontroller board that performs the low level control of the motors as explained in Sec. 1.2.3. In our initial works (Odelga *et al.*, 2016b) we used a low-level controller board provided by the quadrotor's manufacturer. Equipped with an 8-bit microcontroller running at 16 MHz, the board included two 3-axis, 10-bit analog sensors, an accelerometer (with a range of ± 2 g) and a gyroscope (± 300 deg/s range), both read with an analog to digital converter.

The proprietary firmware, which allowed control of the quadcopter with a standard radio controller, was replaced with our own software with a PID controller to track the desired attitude (the roll and pitch angles, and the yaw rate) and thrust commands driving the brushless motor controllers over a standard I^2C bus. Commands are sent to the board through two serial channels. The first channel is used to send control commands to the microcontroller at around 100 Hz, to receive status data (e.g. battery level) and change setting (e.g. controller gains) at low-frequency (˜20 Hz). The second channel, strictly unidirectional, is used to retrieve high frequency IMU readings at 500 Hz.

Later, we retrofitted the robot with a low-level flight controller based on a CopterControl3D (CC3D) board, developed in the OpenPilot[1] project. Equipped with a faster, 32-bit microcontroller running at 72 MHz, this board allowed for better firmware development with its advanced debugging options. The main advantage, however, is its 16-bit digital IMU - MPU-6000 sensor configured to provide measurements at 500 Hz over the range of 500°/s and 4g for the built-in gyroscope and accelerometer, respectively. The IMU measurements are used for the on-board state estimation presented in Sec. 2.5.1.

Data exchange with the on-board computer running ROS and Telekyb is achieved through the asynchronous serial communication port of the microcontroller.

[1] http://opwiki.readthedocs.io/en/latest/index.html

(a) (b)

Figure A.1: Flight controller boards (a) from MikroKopter (v2.5) and (b) CC3D board from the OpenPilot project. Image sources: (a) `http://wiki.mikrokopter.de/en/FlightCtrl_ME_2_5` and (b) `https://openpilotwiki.readthedocs.io/`.

A.2 On-board Computer

The main computational unit of our robot is an Odroid-XU3 shown in Fig. A.2, having a large selection of ports it can interface with numerous sensors and other logic boards, e.g., our flight controller. This single board computer supports the Ubuntu 14.04/16.04 ARM distribution and is ROS enabled, hence we can run the Telekyb (Grabe *et al.*, 2013) software and its high-level controller and algorithms. It is a double quad core[2] ARM microprocessor board that provides enough computational power to make the system independent of external computational units.

The high computational power to weight and size ratio, and the low cost make this board relatively popular among the robotics community. It enables the use of computationally demanding algorithms, e.g., vision based odometry and mapping (Fang and Scherer (2015); Forster *et al.* (2014); Mohanarajah *et al.* (2015)) or advanced control methods such as model predictive control (Liu *et al.* (2015)), directly on the platform.

Communication with the low-level flight controller is carried out over the serial connection as described in the previous section. Power to the board and its peripherals is provided by a 5 V step-down voltage regulator connected to the LiPo battery. The board can exchange data with the fixed operator desk using a USB Wi-Fi adapter.

[2]four Cortex-A15 at 2.0 Ghz, four Cortex-A7 and Heterogeneous Multi-Processing (HMP) solution for tasks management

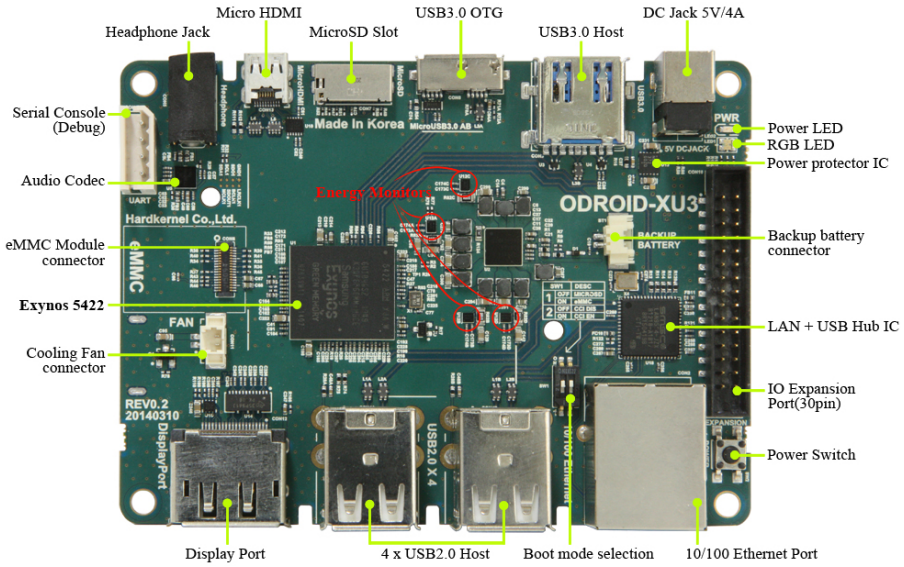


Figure A.2: Odroid-XU3 single board computer with its numerous interfaces. Image source: `https://www.hardkernel.com/main/products/prdt_info.php?g_code=G140448267127&tab_idx=2`.

Appendix B

State Update Code

Algorithm 1: Obstacle state update given a lateral translation $(\Delta x, \Delta y)$

function update$_{XY}$ $(\Delta x, \Delta y)$
input : Integrated translations in the horizontal frame Δx and Δy
parameters: Cell dimensions $\Delta\rho_b$, $\Delta\psi_b$, and Δz_b
data : Obstacle state, an array $S[R][\Psi][Z]$ of the size $R \times \Psi \times Z$

for $\rho_i \leftarrow 1$ **to** R **do**
  for $\psi_i \leftarrow 1$ **to** Ψ **do**
    `// real values of the coordinates`
    $\rho \leftarrow (\rho_i - .5)\Delta\rho_b$
    $\psi \leftarrow (\psi_i - .5)\Delta\psi_b$
    $x \leftarrow \rho cos(\psi) + \Delta x$
    $y \leftarrow \rho sin(\psi) + \Delta y$
    `// change of the cylindrical coordinates`
    $\Delta\rho \leftarrow \sqrt{x^2 + y^2} - \rho$
    $\Delta\psi \leftarrow \text{atan}_2(y, x) - \psi$
    `// change expressed in the discrete coordinates`
    $\Delta\rho_i \leftarrow \lfloor \Delta\rho / \Delta\rho_b \rfloor$
    $\Delta\psi_i \leftarrow \lfloor \Delta\psi / \Delta\psi_b \rfloor$
    $\Delta\rho_r \leftarrow \Delta\rho - \Delta\rho_i \Delta\rho_b$
    $\Delta\psi_r \leftarrow \Delta\psi - \Delta\psi_i \Delta\psi_b$
    `// fer every cell`
    for $z_i \leftarrow 1$ **to** Z **do**
      if $(\rho_i + \Delta\rho_i + 1) \leq R$ ***and*** $(\rho_i + \Delta\rho_i) > 0$ **then**
        $S'[\rho_i][\psi_i][z_i] \leftarrow (1 - \Delta\rho_r)(1 - \Delta\psi_r) \cdot S[\rho_i + \Delta\rho_i][(\psi_i + \Delta\psi_i)\%\Psi][z_i]$
        $+\Delta\rho_r(1 - \Delta\psi_r) \cdot S[\rho_i + \Delta\rho_i + 1][(\psi_i + \Delta\psi_i)\%\Psi][z_i]$
        $+(1 - \Delta\rho_r)\Delta\psi_r \cdot S[\rho_i + \Delta\rho_i][(\psi_i + \Delta\psi_i + 1)\%\Psi][z_i]$
        $+\Delta\rho_r\Delta\psi_r \cdot S[\rho_i + \Delta\rho_i + 1][(\psi_i + \Delta\psi_i + 1)\%\Psi][z_i]$
      else
        $S'[\rho_i][\psi_i][z_i] \leftarrow 0$
      end
    end
  end
end
for $\rho_i \leftarrow 1$ **to** R **do**
  for $\psi_i \leftarrow 1$ **to** Ψ **do**
    for $z_i \leftarrow 1$ **to** Z **do**
      $S[\rho_i][\psi_i][z_i] \leftarrow S'[\rho_i][\psi_i][z_i]$
    end
  end
end

Bibliography

Agamennoni, G., Fontana, S., Siegwart, R. Y., and Sorrenti, D. G. (2016). Point clouds registration with probabilistic data association. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4092–4098.

Aguilar, W. G. and Angulo, C. (2014). Real-time video stabilization without phantom movements for micro aerial vehicles. *EURASIP Journal on Image and Video Processing*, **2014**(1), 46.

Ahmad, A., Ruff, E., and Bülthoff, H. (2016). Dynamic baseline stereo vision-based cooperative target tracking. In *19th International Conference on Information Fusion*, pages 1728–1734.

Ahmad, A., Lawless, G., and Lima, P. (2017). An online scalable approach to unified multirobot cooperative localization and object tracking. *IEEE Transactions on Robotics (T-RO)*, **33**, 1184 – 1199.

Ait-Jellal, R. and Zell, A. (2015). A fast dense stereo matching algorithm with an application to 3d occupancy mapping using quadrocopters. In *2015 International Conference on Advanced Robotics (ICAR)*, pages 587–592.

Al-Kaff, A., Meng, Q., Martn, D., de la Escalera, A., and Armingol, J. M. (2016). Monocular vision-based obstacle detection/avoidance for unmanned aerial vehicles. In *2016 IEEE Intelligent Vehicles Symposium (IV)*, pages 92–97.

Becker, M., Meirelles Dantas, C., and Perdigo Macedo, W. (2006). Obstacle avoidance procedure for mobile robots. *ABCM Symposium Series in Mechatronics*, **2**, 250–257.

Bektas, S. (2015). Least squares fitting of ellipsoid using orthogonal distances. *Boletim de Ciências Geodésicas*, **21**, 329–339.

Bonnin-Pascual, F., Ortiz, A., Garcia-Fidalgo, E., and Company, J. P. (2015). A micro-aerial platform for vessel visual inspection based on supervised autonomy. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 46–52.

Borenstein, J. and Koren, Y. (1991). The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, **7**(3), 278–288.

Bouguet, J.-Y. (2015). Camera calibration toolbox for matlab. `http://www.vision.caltech.edu/bouguetj/calib_doc/`.

Brescianini, D. and D'Andrea, R. (2016). Design, modeling and control of an omni-directional aerial vehicle. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3261–3266.

Burbank, J. L., Kasch, W., and Ward, J. (2011). *Hardware-in-the-Loop Simulations*, chapter 6, pages 114–142. Wiley-Blackwell.

Cai, G., Chen, B. M., Lee, T. H., and Dong, M. (2008). Design and implementation of a hardware-in-the-loop simulation system for small-scale uav helicopters. In *2008 IEEE International Conference on Automation and Logistics*, pages 29–34.

Camacho, E. F. and Bordons, C. (2007). *Introduction to Model Predictive Control*, pages 1–11. Springer London, London.

Chandhrasekaran, V. K. and Choi, E. (2010). Fault tolerance system for uav using hardware in the loop simulation. In *4th International Conference on New Trends in Information Science and Service Science*, pages 293–300.

Cole, D. M. and Newman, P. M. (2006). Using laser range data for 3d slam in outdoor environments. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 1556–1563.

Delmerico, J. A. and Scaramuzza, D. (2018). A benchmark comparison of monocular visual-inertial odometry algorithms for flying robots. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*.

Engel, J., Sturm, J., and Cremers, D. (2013). Semi-dense visual odometry for a monocular camera. In *IEEE International Conference on Computer Vision (ICCV)*, Sydney, Australia.

Erdinc, O., Willett, P., and Bar-Shalom, Y. (2009). The bin-occupancy filter and its connection to the phd filters. *IEEE Transactions on Signal Processing*, **57**(11), 4232–4246.

Fang, Z. and Scherer, S. (2015). Real-time onboard 6dof localization of an indoor mav in degraded visual environments using a rgb-d camera. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5253–5259.

Ferguson, D., Likhachev, M., and Stentz, A. T. (2005). A guide to heuristic-based path planning. In *Proceedings of the International Workshop on Planning under Uncertainty for Autonomous Systems, International Conference on Automated Planning and Scheduling (ICAPS)*.

Ferrick, A., Fish, J., Venator, E., and Lee, G. S. (2012). Uav obstacle avoidance using image processing techniques. In *2012 IEEE International Conference on Technologies for Practical Robot Applications (TePRA)*, pages 73–78.

Forster, C., Pizzoli, M., and Scaramuzza, D. (2014). Svo: Fast semi-direct monocular visual odometry. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 15–22.

Franchi, A., Secchi, C., Son, H. I., Bülthoff, H. H., and Giordano, P. R. (2012). Bilateral teleoperation of groups of mobile robots with time-varying topology. *IEEE Transactions on Robotics*, **28**(5), 1019–1033.

Gageik, N., Benz, P., and Montenegro, S. (2015). Obstacle detection and collision avoidance for a uav with complementary low-cost sensors. *IEEE Access*, **3**, 599–609.

Ge, S. and Cui, Y. (2002). Dynamic motion planning for mobile robots using potential field method. *Autonomous Robots*, **13**(3), 207–222.

Gentili, L., Marconi, L., Naldi, R., and Sala, A. (2009). Experimental framework for a ducted-fan miniature aerial vehicle. In *2009 American Control Conference*, pages 4159–4164.

Gioioso, G., Ryll, M., Prattichizzo, D., Bülthoff, H. H., and Franchi, A. (2014). Turning a near-hovering controlled quadrotor into a 3d force effector. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 6278–6284.

Gohl, P., Honegger, D., Omari, S., Achtelik, M., Pollefeys, M., and Siegwart, R. (2015). Omnidirectional visual obstacle detection using embedded fpga. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3938–3943.

Grabe, V., Riedel, M., Bülthoff, H. H., Giordano, P. R., and Franchi, A. (2013). The telekyb framework for a modular and extendible ros-based quadrotor control. In *2013 European Conference on Mobile Robots*, pages 19–25.

Hartley, R. I. and Zisserman, A. (2004). *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition.

Hilkert, J. M. (2008). Inertially stabilized platform technology concepts and principles. *IEEE Control Systems*, **28**(1), 26–46.

Honegger, D., Meier, L., Tanskanen, P., and Pollefeys, M. (2013). An open source and open hardware embedded metric optical flow cmos camera for indoor and outdoor applications. In *2013 IEEE International Conference on Robotics and Automation*, pages 1736–1741.

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., and Burgard, W. (2013). OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*. Software available at `http://octomap.github.com`.

Houskamp, R. W. (1978). Obstacle protection with unmanned vehicles. In *28th IEEE Vehicular Technology Conference*, volume 28, pages 153–155.

Hua, M. and Rifa, H. (2010). Obstacle avoidance for teleoperated underactuated aerial vehicles using telemetric measurements. In *49th IEEE Conference on Decision and Control (CDC)*, pages 262–267.

Isidori, A. (1995). *Nonlinear Control Systems*. Springer-Verlag, Berlin, Heidelberg, 3rd edition.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, **82**(1), 35–45.

Karpenko, A., Jacobs, D., Baek, J., and Levoy, M. (2011). Digital video stabilization and rolling shutter correction using gyroscopes. Technical report, Stanford University.

Kendoul, F., Fantoni, I., and Lozano, R. (2006). Modeling and control of a small autonomous aircraft having two tilting rotors. *IEEE Transactions on Robotics*, **22**(6), 1297–1302.

Kerl, C., Sturm, J., and Cremers, D. (2013). Robust odometry estimation for rgb-d cameras. In *2013 IEEE International Conference on Robotics and Automation*, pages 3748–3754.

Khalil, W. and Dombre, E. (2002). Chapter 13 - trajectory generation. In W. Khalil and E. Dombre, editors, *Modeling, Identification and Control of Robots*, pages 313 – 345. Butterworth-Heinemann, Oxford.

Liu, Y., Montenbruck, J. M., Stegagno, P., Allgwer, F., and Zell, A. (2015). A robust nonlinear controller for nontrivial quadrotor maneuvers: Approach and verification. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5410–5416.

Lourakis, M. L. A. and Argyros, A. A. (2005). Is levenberg-marquardt the most efficient optimization algorithm for implementing bundle adjustment? In *Tenth IEEE International Conference on Computer Vision (ICCV'05) Volume 1*, volume 2, pages 1526–1531 Vol. 2.

Mahony, R., Hamel, T., and Pflimlin, J. M. (2008). Nonlinear complementary filters on the special orthogonal group. *IEEE Transactions on Automatic Control*, **53**(5), 1203–1218.

Mahony, R., Schill, F., Corke, P., and Oh, Y. S. (2009). A new framework for force feedback teleoperation of robotic vehicles based on optical flow. In *2009 IEEE International Conference on Robotics and Automation*, pages 1079–1085.

Mahony, R., Kumar, V., and Corke, P. (2012). Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *IEEE Robotics Automation Magazine*, **19**(3), 20–32.

Mai, Y., Zhao, H., and Guo, S. (2012). The analysis of image stabilization technology based on small-uav airborne video. In *2012 International Conference on Computer Science and Electronics Engineering*, volume 3, pages 586–589.

Martin, P. and Salaün, E. (2010). The true role of accelerometer feedback in quadrotor control. In *2010 IEEE International Conference on Robotics and Automation*, pages 1623–1629.

Mebarki, R., Lippiello, V., and Siciliano, B. (2015). Nonlinear visual control of unmanned aerial vehicles in gps-denied environments. *IEEE Transactions on Robotics*, **31**(4), 1004–1017.

Mellinger, D. and Kumar, V. (2011). Minimum snap trajectory generation and control for quadrotors. In *2011 IEEE International Conference on Robotics and Automation*, pages 2520–2525.

Merrell, P. C., Lee, D.-J., and Beard, R. W. (2004). Obstacle avoidance for unmanned air vehicles using optical flow probability distributions. *Mobile Robots XVII*, **5609**, 13–22.

Mikami, T. and Uchiyama, K. (2015). Design of flight control system for quad tilt-wing uav. In *2015 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 801–805.

Mingkhwan, E. and Khawsuk, W. (2017). Digital image stabilization technique for fixed camera on small size drone. In *2017 Third Asian Conference on Defence Technology (ACDT)*, pages 12–19.

Mohanarajah, G., Usenko, V., Singh, M., D'Andrea, R., and Waibel, M. (2015). Cloud-based collaborative 3d mapping in real-time with low-cost robots. *IEEE Transactions on Automation Science and Engineering*, **12**(2), 423–431.

Moravec, H. P. (1996). Robot spatial perception by stereoscopic vision and 3d evidence grids. Technical report, Carnegie Mellon University.

Mur-Artal, R., Montiel, J. M. M., and Tards, J. D. (2015). Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, **31**(5), 1147–1163.

Nieuwenhuisen, M., Droeschel, D., Schneider, J., Holz, D., Lbe, T., and Behnke, S. (2013). Multimodal obstacle detection and collision avoidance for micro aerial vehicles. In *2013 European Conference on Mobile Robots*, pages 7–12.

Nister, D., Naroditsky, O., and Bergen, J. (2004). Visual odometry. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, volume 1, pages I–652–I–659 Vol.1.

Nüchter, A., Lingemann, K., Hertzberg, J., and Surmann, H. (2007). 6d slam-3d mapping outdoor environments: Research articles. *J. Field Robot.*, **24**(8-9), 699–722.

Odelga, M., Stegagno, P., Bülthoff, H. H., and Ahmad, A. (2015). A setup for multi-uav hardware-in-the-loop simulations. In *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, pages 204–210.

Odelga, M., Stegagno, P., and Bülthoff, H. H. (2016a). A fully actuated quadrotor uav with a propeller tilting mechanism: Modeling and control. In *2016 IEEE International Conference on Advanced Intelligent Mechatronics (AIM)*, pages 306–311.

Odelga, M., Stegagno, P., and Bülthoff, H. H. (2016b). Obstacle detection, tracking and avoidance for a teleoperated uav. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 2984–2990.

Odelga, M., Kochanek, N., and Bülthoff, H. H. (2017). Efficient real-time video stabilization for uavs using only imu data. In *2017 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, pages 210–215.

Oleynikova, H., Honegger, D., and Pollefeys, M. (2015). Reactive avoidance using embedded stereo vision for mav flight. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 50–56.

Omari, S., Hua, M. D., Ducard, G., and Hamel, T. (2013). Bilateral haptic teleoperation of vtol uavs. In *2013 IEEE International Conference on Robotics and Automation*, pages 2393–2399.

Omari, S., Hua, M.-D., Ducard, G., and Hamel, T. (2014). Bilateral haptic teleoperation of an industrial multirotor uav. In F. Röhrbein, G. Veiga, and C. Natale, editors, *Gearing Up and Accelerating Crossfertilization between Academic and Industrial Robotics Research in Europe:*, pages 301–320, Cham. Springer International Publishing.

Price, E., Lawless, G., Ludwig, R., Martinovic, I., Buelthoff, H. H., Black, M. J., and Ahmad, A. (2018). Deep neural network-based cooperative visual tracking through multiple micro aerial vehicles. *IEEE Robotics and Automation Letters*, **3**, 3193–3200.

Rajappa, S., Ryll, M., Bülthoff, H. H., and Franchi, A. (2015). Modeling, control and design optimization for a fully-actuated hexarotor aerial vehicle with tilted propellers. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4006–4013.

Rajappa, S., Bülthoff, H. H., Odelga, M., and Stegagno, P. (2017). A control architecture for physical human-uav interaction with a fully actuated hexarotor. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 4618–4625.

Riedel, M., Franchi, A., Giordano, P. R., Bülthoff, H. H., and Son, H. I. (2013). Experiments on intercontinental haptic control of multiple uavs. In S. Lee, H. Cho, K.-J. Yoon, and J. Lee, editors, *Intelligent Autonomous Systems 12*, pages 227–238, Berlin, Heidelberg. Springer Berlin Heidelberg.

Roth-Tabak, Y. and Jain, R. (1989). Building an environment model using depth information. *Computer*, **22**(6), 85–90.

Ryll, M., Bülthoff, H. H., and Giordano, P. R. (2015). A novel overactuated quadrotor unmanned aerial vehicle: Modeling, control, and experimental validation. *IEEE Transactions on Control Systems Technology*, **23**(2), 540–556.

Ryll, M., Muscio, G., Pierri, F., Cataldi, E., Antonelli, G., Caccavale, F., and Franchi, A. (2017). 6d physical interaction with a fully actuated aerial robot. In *2017 IEEE Int. Conf. on Robotics and Automation*, pages 5190–5195, Singapore.

Ryu, Y. G., Roh, H. C., Kim, S. J., An, K. H., and Chung, M. J. (2009). Digital image stabilization for humanoid eyes inspired by human vor system. In *2009 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 2301–2306.

Schinstock, D. E. (2013). Gps-aided ins solution for openpilot. `http://wiki.openpilot.org/download/attachments/950387/INSGPSAlg.pdf`.

Schwertfeger, S., Birk, A., and Bülow, H. (2011). Using ifmi spectral registration for video stabilization and motion detection by an unmanned aerial vehicle (uav). In *2011 IEEE International Symposium on Safety, Security, and Rescue Robotics*, pages 61–67.

Se, S., Lowe, D. G., and Little, J. J. (2005). Vision-based global localization and mapping for mobile robots. *IEEE Transactions on Robotics*, **21**(3), 364–375.

Serres, J., Ruffier, F., and Franceschini, N. (2006). Two optic flow regulators for speed control and obstacle avoidance. In *The First IEEE/RAS-EMBS International Conference on Biomedical Robotics and Biomechatronics, 2006. BioRob 2006.*, pages 750–757.

Shen, H., Pan, Q., Cheng, Y., and Yu, Y. (2009). Fast video stabilization algorithm for uav. In *2009 IEEE International Conference on Intelligent Computing and Intelligent Systems*, volume 4, pages 542–546.

Stegagno, P., Basile, M., Bülthoff, H. H., and Franchi, A. (2014). A semi-autonomous uav platform for indoor remote operation with visual and haptic feedback. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3862–3869.

Stegagno, P., Cognetti, M., Oriolo, G., Bülthoff, H. H., and Franchi, A. (2016). Ground and aerial mutual localization using anonymous relative-bearing measurements. *IEEE Transactions on Robotics*, **32**(5), 1133–1151.

Suksakulchai, S., Thongchai, S., Wilkes, D. M., and Kawamura, K. (2000). Mobile robot localization using an electronic compass for corridor environment. In *Systems, Man, and Cybernetics, 2000 IEEE International Conference on*, volume 5, pages 3354–3359 vol.5.

Thillainayagi, R. and Kumar, K. S. (2016). Video stabilization technique for thermal infrared aerial surveillance. In *2016 Online International Conference on Green Engineering and Technologies (IC-GET)*, pages 1–6.

Wang, L., Zhao, H., Guo, S., Mai, Y., and Liu, S. (2012). The adaptive compensation algorithm for small uav image stabilization. In *2012 IEEE International Geoscience and Remote Sensing Symposium*, pages 4391–4394.

Weng, J., Cohen, P., and Herniou, M. (1992). Camera calibration with distortion models and accuracy evaluation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **14**(10), 965–980.

Wiriyaprasat, K. and Ruchanurucks, M. (2015). Realtime vdo stabilizer for small uavs using a modified homography method. In *2015 International Conference on Science and Technology (TICST)*, pages 40–43.

Xu, S., Honegger, D., Pollefeys, M., and Heng, L. (2015). Real-time 3d navigation for autonomous vision-guided mavs. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 53–59.

Yüksel, B., Mahboubi, S., Secchi, C., Bülthoff, H. H., and Franchi, A. (2015). Design, identification and experimental testing of a light-weight flexible-joint arm for aerial physical interaction. In *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pages 870–876.

Yüksel, B., Buondonno, G., and Franchi, A. (2016). Differential flatness and control of protocentric aerial manipulators with any number of arms and mixed rigid-/elastic-joints. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 561–566.

Zhu, J., Li, C., and Xu, J. (2015). Digital image stabilization for cameras on moving platform. In *2015 International Conference on Intelligent Information Hiding and Multimedia Signal Processing (IIH-MSP)*, pages 255–258.

Zohaib, M., Pasha, M., Riaz, R. A., Javaid, N., Ilahi, M., and Khan, R. D. (2013). Control strategies for mobile robot with obstacle avoidance. *CoRR*, **abs/1306.1144**.