UNIVERSITÉ LIBRE DE BRUXELLES

Bruface Brussels Faculty of Engineering

VRIJE UNIVERSITEIT BRUSSEL

Constrained Control of a UAV in a World with Obstacles

Elie Hermand

Master thesis submitted under the supervision of
Prof. dr. ir. Emanuele Garone

The co-supervision of
Dr. Mehdi Hosseinzadeh
Ir. Tam Nguyen

In order to be awarded the Master's Degree in
Electro-Mechanical Engineering
Mechatronics - Mechanical Constructions

Academic year
2017–2018

Abstract

***Constrained Control of a UAV in a World with Obstacles*, Elie Hermand, Electro-Mechanical Engineering, Mechatronics - Mechanical Constructions, 2017–2018**

Nowadays, unmanned aerial vehicles (UAV) are brought to accomplish an ever-growing variety of missions in the presence of challenging obstacles. Interest in simple and systematic methods to handle state and input constraints is raising rapidly. Among others, this led to the development of the explicit reference governor (ERG). This master thesis' work develops a complete and fully functional controller for small UAVs, including the ERG in the navigation part of its architecture. Besides the actual implementation, contribution of the present work is to demonstrate – for the first time – that an ERG augmented controller effectively navigates in a laboratory environment with arbitrary positional constraints. The first step develops a control layer for the UAV stabilization through a cascade scheme. An inner loop, embedded in the UAV, controls its attitude, while an outer loop, implemented on a ground-based computer, controls its position, the position and attitude being measured by a motion capture system. The controller accounts for the UAV dynamics by means of a state space model. Since motion capture solely provides position and attitude, a Kalman filter, tuned according the experimental configuration, filters position and estimates velocity. Preparatory experiments with a pendulum assess performance. In a second step, a navigation layer is supplemented to enabl obstacle avoidance through the ERG. After adequate constraints are defined, its original formulation is extended and specialized. In a third step, the architecture – combining the primary control, Kalman estimator and ERG – is implemented in a Python program. In a last step, several experiments are carried out with constraints representing multiple vertical walls. The UAV response is logged and assessed for arbitrary reference positions, violating or not the constraints. The results confirm that the ERG modified reference behaves well navigating the UAV toward the reference and preventing collision. Independent tests show the system robustness and reliability. This thesis' work successfully implements in practice the ERG concept. The experiments demonstrate the use of ERG as a systematic tool to provide constraint handling capability to a small UAV. Besides this achievement, the products of this work includes a documented, python program which is fast and computationally-efficient, serving as a solid basis for further application. Next steps include performance improvement through Lyapunov function optimization, evaluation of alternative ERG forms, testing of other rotorcraft types and introduction of more and other forms of obstacles.

Keywords: Explicit Reference Governor, Unmanned Aerial Vehicle, Constrained Control, Obstacle Avoidance, Python

Summary

Nowadays, unmanned aerial vehicles (UAV) are brought to accomplish an ever-growing variety of missions in the presence of challenging obstacles. Small vehicles are able to execute a wide range of tasks as diversified as search and rescue tasks, environmental, crop and traffic monitoring, mapping, inspection, aerial transportation and manipulation, and surveillance. In this context, the interest in simple and systematic methods to handle state and input constraints is raising rapidly. Among others, this led to the development of the explicit reference governor (ERG). This master thesis' work develops a complete and fully functional controller for small UAVs, incorporating the ERG in the navigation part of its architecture. Besides the actual implementation, contribution of the present work is to demonstrate – for the first time – that an ERG-augmented controller effectively navigates in a laboratory environment with arbitrary positional constraints which can be imposed in real time. The first step of our work develops a control layer for the stabilization of the vehicles through a simple and robust cascade scheme of two nested loops. An inner loop, embedded in the vehicle, controls its attitude, while an outer loop, implemented on a ground-based computer, controls its position, the actual position and attitude being measured by a motion capture setup overlooking the navigation space. The controller requires a description of the UAVs' dynamics by means of a state space model relating the full pose, velocities and angular rates to generalized input forces and torque. To handle the cameras' measurements that provide solely position and attitude signals subject to noise, a Kalman filter is based on a linear kinematic model to provide filtered position and estimate of the velocity. According to the experimental configuration, the filter is subsequently tuned through adjustment of its process and measurement covariance matrices. Preparatory experiments with a pendulum and the camera system validate the state-space estimator and assess its performance. In a second step, a navigation layer is supplemented to the above pre-stabilizing control layer to provide the UAVs with an obstacle avoidance capability. To this end, the explicit reference governor framework is used. After adequate constraints are defined for a scenario of obstacle avoidance, its original formulation is extended and specialized. In a third step, the whole system architecture – combining the primary control, the Kalman estimator and the ERG – is implemented in a Python-language program. In a last step, several experiments are carried out with single and multiple constraints representing the presence of up to four vertical walls bounding the navigation space. The response of the UAV governed system is logged and assessed for arbitrary positions of the reference, violating or not the constraints. The experimental results confirm that the ERG modified reference behaves as expected navigating the UAV toward the reference and preventing collision into the obstacles. Independent realizations of the tests with multiple scenarios show the system robustness and reliability. A sphere obstacle was also coded but not tested yet. In conclusion, this thesis' work successfully implements in practice the concept of reference governor. The experiments demonstrate the use of the explicit reference governor framework as a systematic tool to provide constraint handling capability to a small UAV. Besides this achievement, the products of this work includes a documented, python program which is fast and computationally-efficient, serving as a solid basis for further application. Next steps of this work include improvement of the controller performance through optimization of the Lyapunov function underlying the current ERG navigation layer, evaluation of alternative forms of the ERG, testing of other types of rotorcrafts, introduction of more and other forms of realistic obstacles in the experimental testing, and enhancement of the graphical user interface.

Contents

List of Figures

Chapter 1

Introduction

In the last three years, a new concept for the constrained control of dynamical systems has been developed: the explicit reference governor (ERG). The main goal application of this framework is the constrained control of unmanned aerial vehicles (UAVs).

1.1 Motivation

The field of unmanned aerial vehicles (UAVs), commonly known as drones, or more generally the field of aerial robotics is expanding with huge opportunities for civil and military applications. More specifically, small unmanned aerial vehicles (sizing between 0.1-0.5 meters in length and 0.1-0.5 kilograms in mass) are able to execute a large range of tasks as diversified as search and rescue tasks, environmental, crop and traffic monitoring, mapping, inspection, aerial transportation and manipulation, and surveillance [1].

Among the various rotorcraft designs, the quadrotors whose propulsion is provided by four motors represent the most flexible and adaptable platforms for such purposes. Hexacopters or octocopters improve fault tolerance and, in the case of re-oriented propellers, overcome the underactuation problem present in quadrotors but they pay a penalty for cost, efficiency and weight. Quads instead cannot be considered fully fault tolerant. Nevertheless they have key advantages: less expensive, less complicated, smaller and lighter per kilogram of payload. A typical quadrotor design is easy to build and reasonably robust. Multi-rotor UAVs are highly manoeuvrable in 3-dimensional space: able to hover in place, take off and land vertically if needed. They generally have sufficient load capacity and flight autonomy to support a number of indoor and outdoor utilizations: small quadrotors have been demonstrated for mapping three-dimensional environments, transporting and manipulating payloads, assembling structures, and for autonomous exploration.

An autonomous UAV will be typically able to fly without human interaction, to monitor and assess its health, status and configuration and, command and control assets onboard the vehicle within the limits of its programmed and built-in intelligence. This could be achieved interacting with a ground based controller. Intelligent autonomous architecture (IAA) is a combination of on-board and ground-based automated systems for controlling the vehicle and its payload. The onboard autonomous executive will execute the flight plan, along with performing other basic tasks associated with flying the vehicle. As an alternative to the IAA, there is a growing interest in fully autonomous UAVs, greatly reducing the computational force available.

In this thesis we consider UAV quadcopters having blades with fixed pitch and no hinges or flaps; the propellers operate in a cross configuration with two pairs of opposite rotors rotating clockwise and the other rotor pair rotating counter-clockwise to cancel the gyroscopic effect. Quadrotors could seem quite easy to model and control as they use no complex mechanical systems and basically only the variation in motor speed for vehicle control. However this is at the cost of complex coupled non-linear and underactuated dynamics that makes accurate flight control a challenge. As shown by Zulu and John [2] who reviewed most of the common control algorithms used on these type of UAVs no single algorithm presents the best of the required features. Therefore getting the best performance usually requires hybrid control schemes that optimize for the desired mission the best combination of robustness, adaptability, optimality, simplicity, tracking ability, fast response and disturbance rejection among other factors. But guaranteeing a level of performance for a specific application is always a compromise. In addition, when the flight itself is mastered it remains that the basics of pilots' ability is to sense and avoid obstacles.

In industrial applications there is an increasing demand to handle requirements that have the form of pointwise-in-time state and control constraints. Non-linear control design mostly use model predictive control (MPC) strategies. In MPC schemes, the control action is computed in real-time by solving an optimization problem on the predicted state trajectories. Although MPC schemes are particularly effective and outperform other solutions in terms of control performances, their applicability can be limited by the elevated computational cost. For nonlinear problems MPC advancement is still unsatisfactory.

Another optimization-based solution consists in stabilizing the system using classical feedback control (primal control unit), and then augmenting the controlled system with a secondary control unit or add-on, called reference governor (RG) able to enforce the constraints by acting on the reference of the closed-loop system [3]. An alternative approach to optimization-based strategies consists in designing a closed-form control law able to ensure constraint satisfaction. Although this typically introduces a loss of performance, explicit formulations are simpler to implement on low-cost hardware. This work used a systematic method previously presented by Nicotra and Garone [3] to design a closed-form control scheme for nonlinear constrained systems based on the RG scheme: the Explicit Reference Governor (ERG) which is based on the utilization of Lyapunov level-sets to obtain a single constraint value that is then enforced by acting on the velocity of the applied reference. The principal benefit of these governors is that the modification of the command is performed only when necessary, the intent being to preserve, whenever possible, the original reference command as far as it satisfies the constraints. The solution provided by such framework naturally enforces both state and input constraints.

Future applications, in factory automation for example, with this type of quadrotors would include more complex missions including a set of task which suppose additional embedded equipment and a ground-based decision making or expert system and will justify continuing research into a synergistic combination of an upper level reference governor and lower level MPC-based controller in order to manage increasing complexity of constraints.

1.2 Project Goal

This master thesis focuses on the practical development of a fully functional controller for a UAV, implementing an explicit reference governor in order to enforce flight constraints. The main initial goal is to propose an experimental validation of the ERG through a practical application of the constrained control of a UAV.

Two main stages were needed to achieve this goal: first, a primary controller had to be designed in order to stabilize the UAV, then, the ERG had to be implemented to supplement the UAV with obstacle avoidance capability. As an auxiliary step, the experimental setup, composed of a motion capture system and a UAV has to be fully operational. This last part becoming a lot more tedious than foreseen, one of the main products of this work became to propose a solid working implementation on which future students could base their work.

1.3 State of the Art

1.3.1 Overview of UAV control

Over the years, a wide variety of schemes have been proposed to tackle the challenging problem of the control of a UAV. Those can be roughly categorized as linear and non-linear. [2, 4, 5]

Linear Control

Linear control techniques are based on linear approximations of the dynamic model around the desired state trajectories. Those techniques are widely used in airplanes where the attitude variations are pretty low, yielding accurate linearized models. [4] This category includes classical PID controllers [6, 7], linear quadratic regulators (LQR) [8, 9], linear quadratic Gaussian controllers (LQG) [10] and other optimal controllers like H_∞ [11, 12].

The PID controller is the most applied controller in industry for a broad range of applications. The classical PID linear controller is simple to design with gains parameter easy to adjust, and resulting in reasonable robustness. However in applying PID controller to the quadrotor, the inaccurate mathematical modeling of the non-linear dynamics limits severely its performance.

The LQR approach for a dynamic system is based on minimizing a suitable cost function. It demonstrates better performance than the PID even under perturbations but it tends to loose tracking after avoiding an obstacle and its performances in the presences of many obstacles is to be proven.

The main advantage of linear controls is their robustness and simplicity. However, their local nature raises important limitations mainly regarding performance and robustness with respect to aerodynamic disturbances and uncertainties [2, 4].

Non-Linear Control

In order to extend the stability domain and increase the robustness in the case of highly non-linear dynamics, non-linear methods have been increasingly developed [4].

Those mainly include sliding mode controllers (SDC), backstepping control approaches, dynamic inversion techniques and model predictive control (MPC) schemes.

Sliding mode control works by applying a discontinuous control signal to the system in order to make it slide along a prescribed path. Its main advantage is that it does not simplify the dynamics through linearization and has good tracking. The main drawback is that the actuators which have to cope with the high frequency control actions could wear or even break.

Backstepping control consists of a recursive algorithm that breaks down the controller into steps, each one being progressively stabilized [2]. The algorithm has the advantage to converge fast using less computational resources and can well handle disturbances. The main limitation is its lack of sufficient robustness. An integrator was added to the general backstepping algorithm in an attempt to improve robustness to external disturbances (Integrator backstepping control) but like other hybrid combinations it makes the whole controller a lot more complex [13].

1.3.2 Constrained Control

In order to satisfy constraints, there are different approaches possible. The most "classical" way to deal with constraints are anti-windup techniques. Anti-windup approaches are based on the saturation of the input issued by the closed-loop law submitted to constraints followed by the modification of the internal states of the controller to prevent windup effect.

Model Predictive Control

Model predictive control (MPC) provides a framework to impose input constraints as part of the control architecture. The MPC framework allow to effectively implement state and input constraints along with goal states, as well as trajectory constraints. Model predictive control is an approach to controller design that involves on-line optimization calculations at each sampling time for predicted state trajectories. The online optimization problem takes into account system dynamics, constraints and control objectives. It is able to handle hard constraints on controls and states that arise in most industrial applications [14]. The requirement in computational resources is high and the development of more efficient solvers to the optimization problem is a constant requisite especially for fast and reliable nonlinear MPCs.

Reference and Command Governors

The reference governor scheme is comparable to the anti-windup approach as an add-on scheme that instead of saturating the control input, is acting on the error with respect to the reference. The saturation applies before the calculation of the control input without touching the internal states of the controller for constraint satisfaction. Classical RG presented discrete time domain reformulation for performing trajectory based optimization. In order to improve the performances several variants of the reference and command governor schemes have been proposed in the literature often leading to heavy computational costs. For non-linear problems, the RG formulation is still simpler than MPCs and permit direct estimation of the state trajectories that ensure constraints satisfaction. It is also possible to use Lyapunov level-sets to lower computational needs at the cost of output performances.

The selection between the different approaches is conditioned by the priority of the concerned applications: the MPC approach for high performances, anti-windup for limited computational resources and the reference governor as a compromise between loss of performances and reasonable computational costs (depending on the type of RG). Another decision factor could be the pre-existence of a primary control scheme that a reference governor could supplement while an MPC approach would impose to re-design the entire control.

1.3.3 Explicit Reference Governor

The explicit reference governor (ERG) introduced in [3, 15] is presented as a closed-form reference manager able to enforce constraints without the need to solve an online optimization problem. This is achieved by manipulating the derivative of the applied reference instead of its actual value as in the classical RG scheme. One of the main goals of the ERG is to be able to propose a systematic approach to constraint handling. The typical design steps would be: first, to design a primary control to pre-stabilize the system while providing a Lyapunov function for the closed-loop, then to translate the state constraints into maximum admissible values for the Lyapunov function, and finally, to implement the ERG as an add-on scheme.

This recently introduced framework has already developed a quite strong theoretical basis but still suffers from a lack of experimental results. In [15], the use of a combined control, ERG and LQR, has been used successfully to control a system consisting in an inverted pendulum on a cart.

Chapter 2

Stabilization of the UAV

This chapter presents a stabilizing control layer for a vertical take-off and landing UAV. The development is done focusing on the quadrotor design.

The control of a quadrotor is a challenging problem because of its non-linear nature and under-actuated design [16]. As previously stated, this problem has been largely addressed and thus a number of solution schemes are available. A common approach is to stabilize the UAV using a PD cascade control with two distinct loops for the control of the attitude and position. This is the control scheme that will be presented in this thesis. This control scheme has been chosen mainly for its simplicity and robustness. Furthermore, this type of controller is the one that is usually implemented in commercial quadrotors that can be bought off-the-shelf.

The first section will present the basics of quaternion algebra and of the use of quaternions for attitude representation. Next, a dynamical model of the position and attitude of a UAV will be presented. Last, a nested control strategy to stabilize a UAV at any equilibrium point will be proposed.

2.1 Background: Quaternions

Quaternions are hypercomplex numbers belonging to the quaternion space $\mathbb{H}$. A quaternion can be represented in many ways. Here, a quaternion q is represented as a 4-components vector:

$$q = [q_w \; q_x \; q_y \; q_z]^T = \begin{bmatrix} q_w \\ q_{xyz} \end{bmatrix} \tag{2.1}$$

where $q_w, q_x, q_y, q_z \in \mathbb{R}$ and $q_{xyz} \in \mathbb{R}^3$

The conjugate and norm of a quaternion q are given respectively by

$$q^* = \begin{bmatrix} q_w \\ -q_{xyz} \end{bmatrix} \quad \text{and} \quad \|q\| = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2}. \tag{2.2}$$

The main operation of quaternion algebra is the non-commutative quaternion product. The product of two quaternions q and h is defined as

$$q \cdot h = \begin{bmatrix} q_w h_w - q_{xyz} h_{xyz} \\ h_w q_{xyz} + q_w h_{xyz} + q_{xyz} \times h_{xyz} \end{bmatrix}. \tag{2.3}$$

Unit quaternions (or versors), i.e. quaternions of unitary norm, are a convenient way to represent rotations and orientations in three dimensions. Compared to Euler angles, their main advantage is that they avoid the problem of gimbal lock while being simpler to use. They are also more compact than rotation matrices. A simple way to their interpretation is using the axis angle representation. For this purpose, consider a rotation of θ degrees around a unitary vector u, the corresponding quaternion is given by

$$q = \begin{bmatrix} \cos(\frac{\theta}{2}) \\ \sin(\frac{\theta}{2}) u \end{bmatrix}. \tag{2.4}$$

The vector u' resulting from the rotation of a vector u about a quaternion q is given by the following quaternion multiplication:

$$u' = q \cdot u \cdot q^* \tag{2.5}$$

where the vector u is equivalent to the quaternion $[0 \; u^T]^T$, i.e. with a scalar part equal to zero.

Further informations on attitude representation and quaternion algebra can be found in [17] and on [18].

2.2 Dynamic Model

This section introduces a general mathematical model in order to describe the dynamics of a UAV. Although we will focus on the quadrotor design, the state-space dynamic model presented below is common to the most typical VTOL aircrafts, more precisely to the ones where the thrust direction is constant and pointing in the negative z direction (e.g. multi-rotor copters, ducted fans). Equivalent models from the one proposed in this section are described in e.g. [19–21]

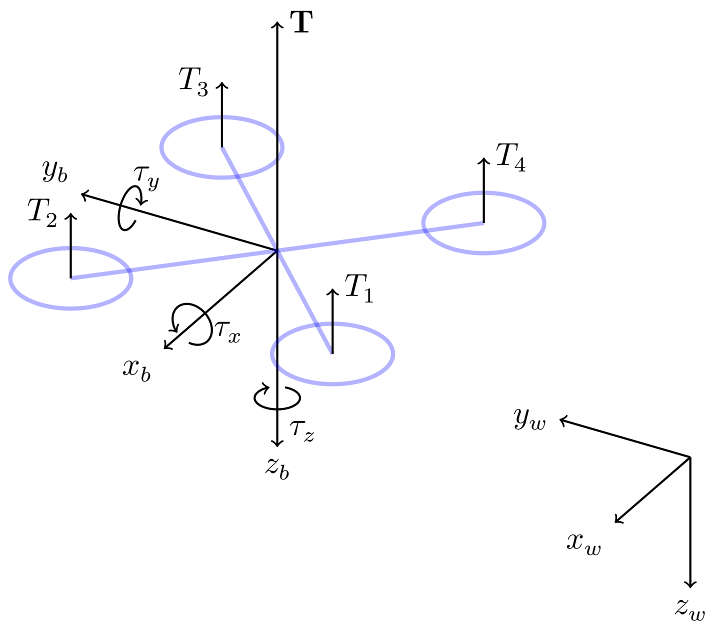


Figure 2.1: World S_w and body S_b reference frames; the quadrotor frame and four rotors; the four input thrusts T_i; and the resulting thrust T and torque $[\tau_x \tau_y \tau_z]^T$ (based on [19]).

To provide a convenient description of the aircraft dynamics, two reference frames are defined: a world reference frame ($S_\text{w} = \{O_\text{w}, (e_1, e_2, e_3)\}$), inertial and fixed , and a body reference frame ($S_\text{b} = \{O_\text{b}, (b_1, b_2, b_3)\}$), attached to the UAV (Figure 2.1). To follow the aerospace conventions, the x-axis points towards the forward direction and the z-axis points downwards.

In order to describe the body dynamics, lets begin by defining some notations:

- p denotes the position of the UAV in S_w;
- ω, its angular velocity defined in S_b;
- q is the unit quaternion describing the rotation from S_b to S_w, defined in S_w.

In the case of the very common quadrotor configuration, the thrust force is produced by the four propellers mounted at each end of the cross' arms. Two symmetrically opposite rotors (relative to the UAV's center) rotate in a same direction while the two others rotate in the reverse direction to cancel the yawing moment. Each rotor ($i = 1, 2, 3, 4$) produces a thrust force T_i proportional to the square of the rotor speed. The control inputs can be summarized as a resulting thrust force T and a torque vector $\tau = [\tau_x \tau_y \tau_z]^T$ (defined in

S_b) with

$$T = \sum T_i \quad \text{and} \quad \begin{cases} \tau_x = (T_1 - T_2 - T_3 + T_4)d \\ \tau_y = (T_1 + T_2 - T_3 - T_4)d \\ \tau_z = (T_1 - T_2 + T_3 - T_4)\mu \end{cases} \tag{2.6}$$

where d is the half-distance between two consecutive rotors (or the distance between the rotors and the body-axis), and μ is the ratio between lift and drag, depending on the propellers' geometry.

Given a UAV of mass m and symmetric inertia matrix J, the Newton-Euler equations of motion lead to the following state-space model:

$$\begin{cases} m\ddot{p} = mge_3 - Tq \cdot e_3 \cdot q^* \\ \dot{q} = \frac{1}{2} q \cdot \omega \\ J\dot{\omega} = -\omega \times J\omega + \tau \end{cases} \tag{2.7}$$

where $g \approx 9.81\,\mathrm{m/s^2}$ is the gravitational acceleration, $e_3 = [0\ 0\ 1]^T$.

The state vector $\mathbf{x}$ and input vector $\mathbf{u}$ fully describing system (2.7) are

$$\mathbf{x} = \begin{bmatrix} p \\ \dot{p} \\ q \\ \omega \end{bmatrix} \quad \text{and} \quad \mathbf{u} = \begin{bmatrix} T \\ \tau \end{bmatrix}. \tag{2.8}$$

2.3 Control Layer

To determine a suitable control law, the control objectives have to be defined. The objective of the control layer is to stabilize the UAV in a desired equilibrium point. The set of equilibrium points is characterized by identifying when the state-space model (2.7) remains stationary. This corresponds to the quadrotor being in an hovering state. The steady-state requirements of the state-space system are thus given by:

$$\dot{p} = 0, \quad \omega = 0, \quad q \cdot e_3 \cdot q^* = e_3. \tag{2.9}$$

It can be noted that, as expected, there are four degrees of freedom remaining non-imposed: the three position terms $[p_x\ p_y\ p_z]^T$ and the yaw angle ψ. Indeed, the condition $q \cdot e_3 \cdot q^* = e_3$ holds true for any arbitrary rotation around the z-axis.

Based on the dynamic model (2.7), two separate control loops are used to stabilize the UAV's position on the one hand and its attitude on the other hand (Figure 2.2).

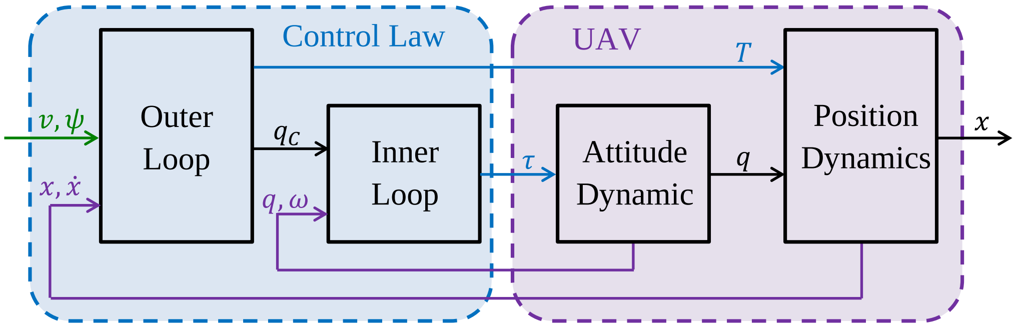


Figure 2.2: Cascade control scheme. [22]

2.3.1 Outer Loop

The aim of the outer loop is thus to control the UAV's position. As previously stated, this will be done under the assumption that the attitude can be used as a control input, i.e. $q = q_c$. This leads to the following position dynamics:

$$m\ddot{p} = mge_3 - Tq_c \cdot e_3 \cdot q_c^*. \tag{2.10}$$

Since both T and q_c are control inputs, the thrust vector $F = -Tq_c \cdot e_3 \cdot q_c^*$ can be directly assigned using a standard PD controller with gravity compensation.

A suitable thrust vector $F = [F_x \ F_y \ F_z]^T$ (defined in S_w) is thus constructed as

$$\begin{cases} F_x = -mk_{px}(p_x - r_x) - mk_{dx}\dot{p}_x \\ F_y = -mk_{py}(p_y - r_y) - mk_{dy}\dot{p}_y \\ F_z = -mk_{pz}(p_z - r_z) - mk_{dz}\dot{p}_z - mg \end{cases} \tag{2.11}$$

where the k_{pi} and k_{di} gains are positive scalars. Usually, the dynamics in the x and y axis are similar and the gains can be chosen with $k_{px} = k_{py}$ and $k_{dx} = k_{dy}$.

Since $\|q_c \cdot e_3 \cdot q_c^*\| = 1 \ \forall q_c$, the input thrust T can be directly computed from the modulus:

$$T = \|F\|. \tag{2.12}$$

As for the input attitude q_c, it is convenient to decompose the rotation in two steps: an arbitrary rotation q_ψ around the yaw axis e_3 and the shortest arc rotation q_α that aligns e_3 with the desired thrust vector.

Indeed, the rotation q_α is such that $q_c \cdot e_3 \cdot q_c^* = q_\alpha \cdot e_3 \cdot q_\alpha^*$ but this does not imply that $q_c = q_\alpha$. The control quaternion is thus given by the following quaternion product:

$$q_c = q_\alpha \cdot q_\psi. \tag{2.13}$$

To complete the control, it remains to simply assign the yaw rotation q_ψ. For a given reference angle ψ, using the axis angle representation, we have

$$q_\psi = [\cos(\psi/2)\ 0\ 0\ \sin(\psi/2)]^T. \tag{2.14}$$

2.3.2 Inner Loop

The outer loop control aimed to the control of the UAV's position while using the attitude q_c as a control input.

The role of the inner loop is to control the UAV's attitude using the attitude q_c as reference. By introducing the error quaternion

$$\tilde{q} = q \cdot q_c^*, \tag{2.15}$$

the attitude dynamics from system (2.7) can be rewritten

$$\begin{cases} \dot{\tilde{q}} = \frac{1}{2}\tilde{q} \cdot \omega \\ J\dot{\omega} = -\omega \times J\omega + \tau \end{cases}. \tag{2.16}$$

Finally, the inner loop can be stabilized using a PD control law:

$$\tau = -h_p \tilde{q}_{xyz} - h_d \omega. \tag{2.17}$$

where the gains h_p and h_d are positive scalars.

For the purpose of this work, since the attitude control was embedded and imposed in the tested UAV's (see section 5.2), it was not required to design the inner loop controller. Nevertheless, for the seek of completeness and for further reference, a compatible control law has been proposed.

Chapter 3

State Estimation

In chapter 2, a control scheme has been described in order to control the position of a UAV. This controller assumed that the absolute position and velocity were known and furthermore, that those measures were exempt of noise. However, when it comes to the practical implementation, the state is usually not fully available through measurements.

As it will be seen in chapter 5, the experimental setup used for this work does not give access to a direct measurement of the aircraft velocity but only to its position. In this chapter, the use of a Kalman filter to estimate the UAV's velocity is presented.

The first section gives an insight of the theory of the Kalman filter. The second section presents the prediction model and its tuning, that will be used for our implementation. The last section aims to assess and validate the resulting estimator.

3.1 Kalman Filter

The Kalman filter algorithm can be viewed as an estimator that produces an output based on a series of noisy measurements. This outputs can be estimates of unmeasured variables of interest or a filtered sequence of measurements. [23, 24]

The Kalman filter model assumes that the system evolves as a stochastic system subject to a zero-mean Gaussian noise. Its state space model is given by the following equations:

$$\begin{cases} x(t) = Ax(t-1) + Bu(t) + w(t) \\ y(t) = Cx(t) + v(t) \end{cases} \tag{3.1}$$

where

- $x(t)$, $u(t)$ and $y(t)$ are respectively the state, input and output vectors;
- A, B and C are respectively the state transition, control input and observation matrices;
- $w(t)$ contains the process disturbance terms for each parameter in the state vector and is assumed a zero-mean Gaussian noise of covariance Q;
- $v(t)$ contains the measurement noise terms for each measurement and is assumed to be a zero-mean Gaussian noise with covariance R.

The Kalman filter operates iteratively to determine a state estimate $\hat{x}(t)$ and its associated error covariance matrix $P(t)$. The algorithm is decomposed in two stages: prediction and measurement update. The prediction stage involves the following equations:

$$\begin{cases} \hat{x}(t|t-1) = A\hat{x}(t-1|t-1) + Bu(t) \\ P(t|t-1) = AP(t-1|t-1)A^T + Q \end{cases} \tag{3.2}$$

The measurement update equations are

$$\begin{cases} \hat{x}(t|t) = \hat{x}(t|t-1) + K[y - C\hat{x}(t|t-1)] \\ P(t|t) = P(t|t-1) - KCP(t|t-1) \end{cases} \tag{3.3}$$

where K is the Kalman gain (or weighting matrix) given by

$$\begin{cases} S = R + CP(t|t-1)C^T \\ K = P(t|t-1)CS^{-1} \end{cases} . \tag{3.4}$$

3.2 Prediction Model

In this section, the Kalman filter is applied to our case. First, the state-space model that is used for the prediction model is determined. Second, the Kalman filter is tuned by choosing the values of the covariance matrices Q and R.

To begin, a suitable state vector is chosen. As previously stated, the objective of the Kalman filter is to estimate the translational position and velocity. The following state vector is thus chosen:

$$\mathbf{x} = [p_x \; p_y \; p_z \; \dot{p}_x \; \dot{p}_y \; \dot{p}_z]^T. \tag{3.5}$$

The state-space model used for the prediction computation is given by the following matrices:

$$A = \begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad B = 0, \quad C = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}. \tag{3.6}$$

This model assumes the position to evolve linearly between two update steps with a constant speed. Thanks to the high update rate of the system, this assumption gives relatively good results.

Alternatively, a more accurate model of the real dynamics could have been used, possibly using an extended Kalman filter to implement the non-linear model. This alternative would have complicated the implementation while only achieving a very slight improvement compared to the chosen model. Here, the model mismatch is taken into account as process disturbance.

Now that the model has been defined, the covariance matrices will be used to tune the Kalman filter.

To be able to tune the covariance parameters, it is important to understand how the process and measurement noise covariances affect the state estimate. As seen in equations (3.2), (3.3) and (3.4), the operation of the estimator depends on the Kalman gain K. For a small K, the estimator "believes" the model while for a large K, the estimator "believes" the measurements. The effect of the covariance matrices on the gain and the estimation is resumed in table 3.1.

Condition	Gain	Parameter
Model trusted	small	P small (adequate model)
		R large (noisy measurement)
Measurement trusted	large	R small (good measurement)
		P large (inadequate model)

Table 3.1: Estimation operation depending on covariance parameters (adapted from [23]).

An initial tuning of the filter was achieved by using a simple pendulum. The matrices were then refined using measurements gathered during a UAV flight. The empirical tuning of the Kalman filter resulted in the following covariance matrices:

$$Q = 1e^{-4} \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad R = 1e^{-6} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

3.3 Experimental Validation

In this section, the resulting state estimator is tested to assess its performance. For this purpose, a pendulum with reflective markers was set-up in the capture arena. The pendulum was placed in the center of the capture volume in a way that it oscillates in the yz-plane.

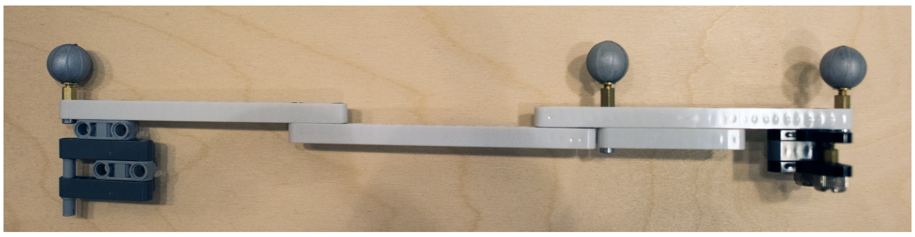
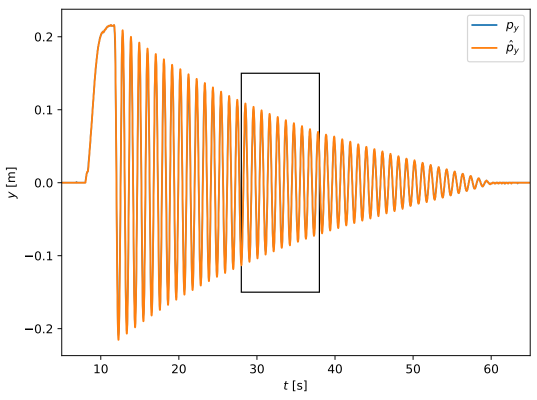


Figure 3.1: Pendulum used for the Kalman filter validation showing the reflective markers for the tracking.

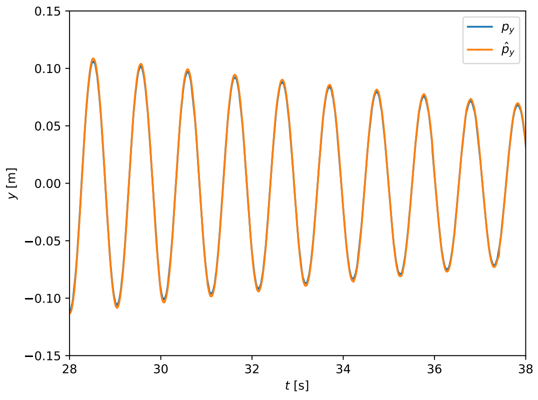
Figure 3.2 compares the position estimated by the Kalman filter to the raw measured one. By analysing the figures, it is clear that the fit is relatively good. Further analysis shows that the difference is always within $1\,\mathrm{mm}$, and thus of the same order of the measurement error.

In Figure 3.3, the estimated velocity is superimposed to the position of the pendulum. It can be seen that the estimation corresponds to the expected curve with the velocity peaks corresponding to the position's zeros and the velocity's zeros corresponding to the position extrema.

Figure 3.4 shows data extracted from consecutive experiments with the fully controlled UAV. The estimated velocity and the directly differentiated values are superimposed. The estimation results in a lot smoother curve where the high frequencies oscillations have been filtered. Those oscillations are clearly due to noise and, therefore, the estimated values better describe the real velocity.

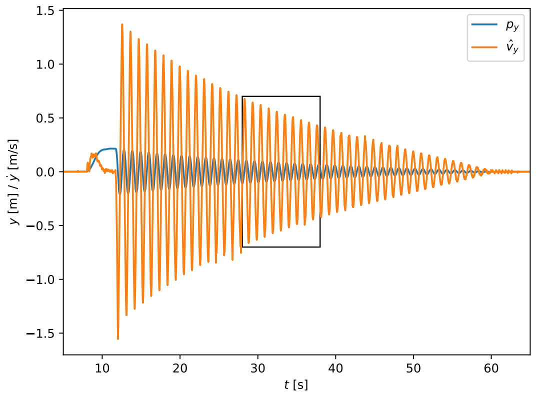


(a) Complete experiment.

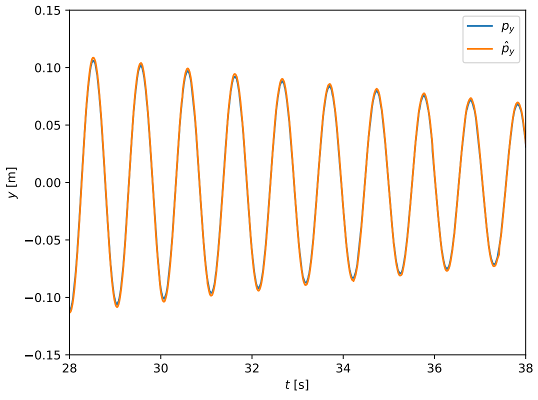


(b) Zoom.

Figure 3.2: Pendulum experiment: measured p vs estimated $\widehat{p}$ position.



(a) Complete experiment.



(b) Zoom.

Figure 3.3: Pendulum experiment: position p and estimated $\widehat{v}$ velocity.

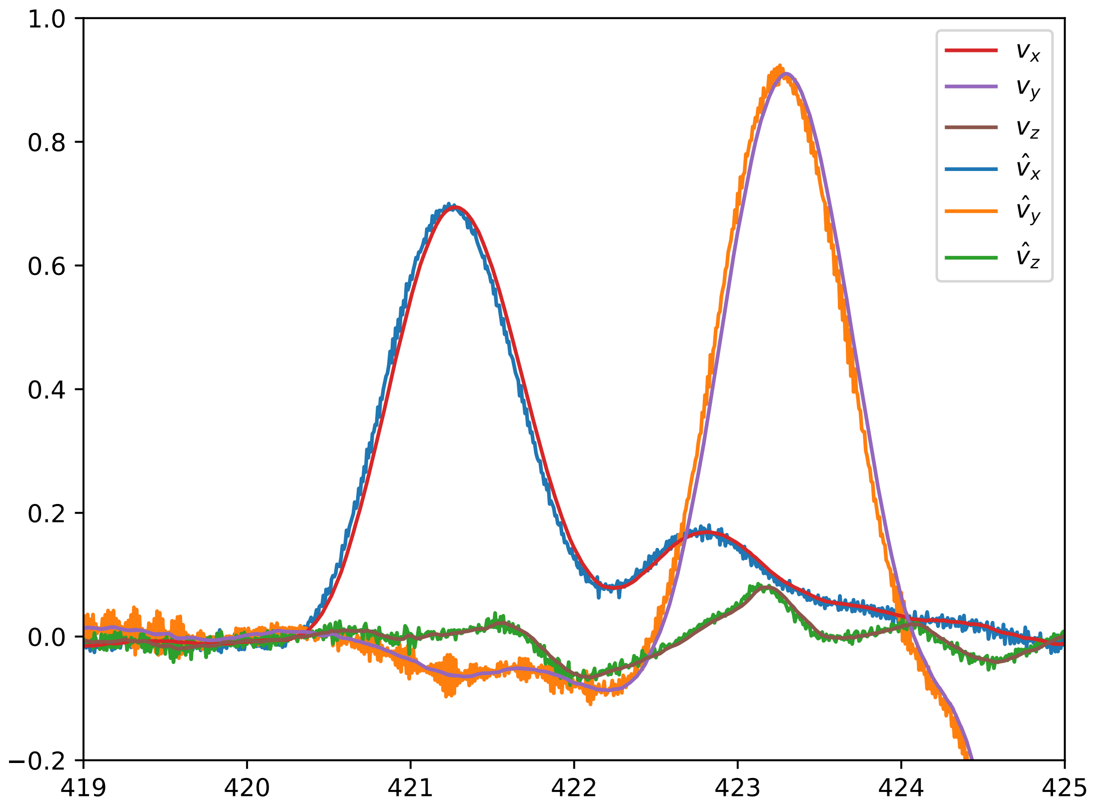


Figure 3.4: Comparison of the estimated velocity $\hat{v}$ with the differentiated one v.

Chapter 4

Navigation Layer: Explicit Reference Governor

In Chapter 2, a pre-stabilizing control law for UAVs has been described. Subsequently, it will be shown in Chapter 5 that this controller has been successfully implemented and applied to stabilize, in real-time, a Parrot AR.Drone UAV in a motion capture setup.

The objective of the navigation layer will be to supplement the already stabilized system with obstacle avoidance capability. To this end, this chapter will treat of the implementation of an explicit reference governor in order to enforce positional constraints by manipulating the auxiliary reference of the closed-loop system.

The explicit reference governor is a non-linear reference manager scheme that is able to provide constraint handling capabilities without the need to solve an online optimization problem. To achieve constraints enforcement, the ERG relies on invariant Lyapunov level-sets.

4.1 Basic Principle

The explicit reference governor deals with pre-compensated continuous time systems in the form

$$\dot{x} = f(x, v) \tag{4.1}$$

that are subject to constraints

$$c_i(x, v) \geq 0. \tag{4.2}$$

The constraint enforcement problem that the ERG aims to solve is to generate, at each

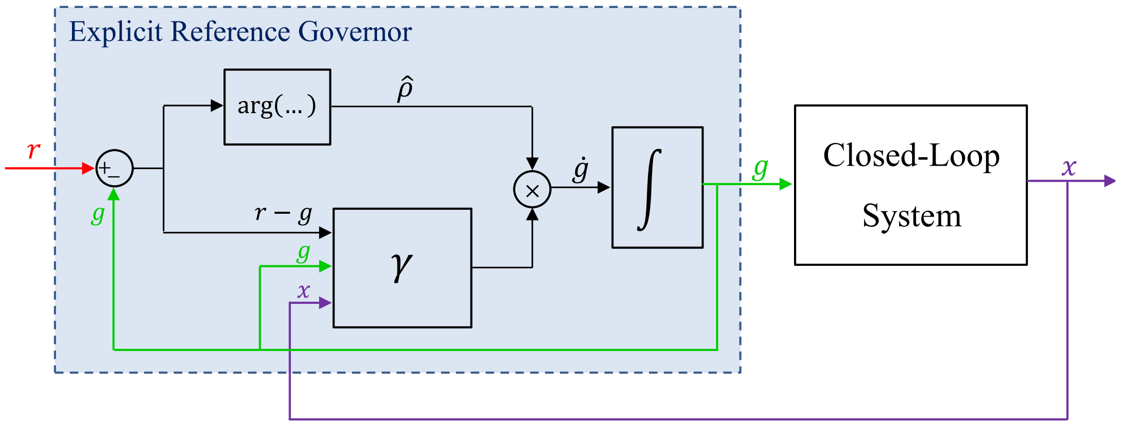


Figure 4.1: Explicit reference governor internal scheme [15].

time t, a suitable reference $v(t)$ such that the constraints are not violated and that $v(t)$ approximates the input reference $r(t)$ as closely as possible.

The ERG framework solves the constraint enforcement problem by assigning

$$\dot{v} = \Delta(x,v)\rho(x,v) \tag{4.3}$$

where $\Delta(x,v)$ is the dynamic safety margin and $\rho(x,v)$ is the attraction field.

It can be noted that, unlike other reference governors, the ERG works by manipulating the derivative of the modified reference and not the reference itself. Figure 4.1 shows the general scheme of the ERG.

The design challenge of the ERG effectively becomes to find suitable $\Delta(x,v)$ and $\rho(x,v)$ based on the nature of the system and its constraints. An intuitive interpretation of those terms is proposed in [22]:

- The dynamic safety margin $\Delta(x,v)$ can be interpreted as the distance between the current state trajectory and the constraints. The role of $\Delta(x,v)$ will thus be to regulate how fast the reference can move, i.e. the modulus of $\dot{v}$.
- The attraction field $\rho(x,v)$ represents the direction of the path leading to the desired reference r from the current reference v. Therefore, $\rho(x,v)$ determines the direction of $\dot{v}$.

4.1.1 Original Formulation

This section presents the original formulation of the ERG as proposed in [3, 15].

In order to describe the basic principle of the ERG, a Lyapunov function V of the pre-stabilized system is considered. This function must satisfy the Lyapunov asymptotic stability conditions.

Given a threshold value $\Gamma(v)$ continuous in v such that

$$V(x,v) \leq \Gamma(v) \Rightarrow c(x,v) \geq 0, \tag{4.4}$$

it follows that, for a constant reference v the Lyapunov invariant set $\{x : V(x,v) \leq \Gamma(v)\}$ fully contains x_v and is entirely contained in the constraints.

This has led, in the original ERG proposition, to the following choice for the dynamic safety margin:

$$\Delta(x,v) = \kappa(\Gamma(v) - V(x,v)) \tag{4.5}$$

where $\kappa > 0$ is an arbitrary scalar.

For what regards the attraction field, it is intuitively chosen as the unit vector pointing from the auxiliary reference v to the desired reference r. It is computed as

$$\rho(r,v) = \frac{r - v}{\max(\|r - v\|, \eta)} \tag{4.6}$$

where the smoothing radius $\eta > 0$ is simply introduced to avoid numerical issues during computation.

4.1.2 Dynamic Safety Margin

Although any threshold value defined by equation (4.4) is a Lyapunov set that does not violate the constraints, the choice of $\Gamma(v)$ determines how conservative the dynamic safety margin is. Consequently, it greatly influences the performance of the governed system.

The optimal choice of the threshold value is thus the largest level-set that does not violates the constrains:

$$\begin{cases} \Gamma(v) = \max_x \mathcal{V}(x,v), \text{ s.t.} \\ V(x,v) \leq \mathcal{V}(x,v) \Rightarrow c(x,v) \geq 0. \end{cases} \tag{4.7}$$

It must be noted that using Lyapunov sets is not the only way to construct a dynamic safety margin. Alternatives comprise other invariant set approaches, trajectory-based approaches or feed-forward approaches that do not require the state knowledge. [22]

The case of multiple constraints $c_i(x, v)$ can easily be treated by assigning the overall safety margin as

$$\Delta(x, v) = \min_i \Delta_i(x, v) \tag{4.8}$$

where $\Delta_i(x, v)$ is the safety margin associated to the constraint $c_i(x, v)$.

4.1.3 Attraction Field

A more general definition of the attraction field can be done by considering the domain of admissible equilibrium states $\mathcal{D}$. The construction of a suitable attraction field is then equivalent to the generation of a continuous path, entirely comprised in $\mathcal{D}$, that connects the current reference v to the desired one r.

4.2 Application to Linear Systems

Although the ERG framework has been initially developed as a tool for non-linear systems, the linear system theory is often preferred because of its simplicity. Additionally, it is very common that non-linear systems are pre-stabilized using feedback linearization and, as the ERG framework treats with the pre-stabilized system, the theory presented in this section can be fully applied. As a result, this section will show how the ERG is specialized to the case of linear systems subject to linear constraints.

As a start, lets consider a pre-stabilized linear system of the form

$$\dot{x} = Ax(t) + Bv \tag{4.9}$$

subject to linear constraints

$$c_i(x, v) = a^T + b \geq 0. \tag{4.10}$$

Given a constant reference v, the system will thus asymptotically tend to the equilibrium point $x_v = -A^{-1}Bv$.

The basic linear system theory teaches us that linear systems are characterized by an infinite choice of quadratic Lyapunov functions of the form

$$V(x, v) = (x + A^{-1}Bv)^T P(x + A^{-1}Bv) \tag{4.11}$$

where $P > 0$ must satisfy the Lyapunov equation

$$A^T P + PA \leq 0. \tag{4.12}$$

It can be proven [22], that for a given Lyapunov function of the form (4.11) and linear constraints (4.10), the optimal threshold value can be computed as

$$\Gamma(v) = \frac{(a^T x_v)^2 + b}{a^T P^{-1} a}. \tag{4.13}$$

4.3 Obstacle Avoidance

In Section 4.1, the general theory behind the explicit reference governor framework has been presented. Subsequently, Section 4.2 showed how to specialize the ERG for linear systems.

This section aims to define suitable state and input constraints, and further develop the basic ERG in order to apply the ERG framework to the obstacle avoidance problem. To this end, two relatively simple cases will be treated:

- Wall avoidance: this constraint will model a planar surface that the UAV must not trespass;
- Spherical obstacle avoidance: this constraint will model a spherical surface that must be circumvented by the UAV.

4.3.1 Wall Avoidance

The constraint definition in order to enforce wall avoidance for a UAV, which position is given by p, is done by modelling a planar surface. This constraint can be written

$$c^T p + d \geq 0 \tag{4.14}$$

where c is the normal vector to the modelled surface (pointing inside the allowed region) and d is the distance of the plane along the normal direction to the origin, effectively placing it into the world.

From equation (4.13), the threshold value associated to a wall constraint is thus [22]

$$\Gamma_W(v) = \frac{(c^T v + d)^2}{c^T P^{-1} c}. \tag{4.15}$$

As the wall constraint $c^T p + d \geq 0$ can be potentially violated at steady-state $p = v$, the attraction field (4.6) is no more sufficient for the correct behaviour of the ERG.

The attraction field is thus extended with a repulsion term as [22]

$$\rho_W(v,r) = \rho(v,r) + \max\left(\frac{\zeta - (c^T v + d)}{\zeta - \delta}, 0\right) c \tag{4.16}$$

where $\zeta > \delta$ is the influence region of the constraint and $\delta > 0$ is the static safety margin.

4.3.2 Spherical Obstacle Avoidance

The obstacle avoidance constraint proposed models a spherical surface that the UAV must circumvent. The equation constraint for a UAV with position p is written as

$$\|p - p_0\| - R \geq 0 \tag{4.17}$$

where $R > 0$ is the radius of the sphere and p_0 is the center of the sphere.

The constraint above defines a non-convex admissible region. For this reason, typical reference governors find this constraint relatively challenging (due to the difficulty to solve the optimization problems). In the case of the ERG framework, the handling of this constraint is not particularly difficult. [22]

The way this constraint is handled is by defining a reference-dependent virtual wall $c(v)^T p + d \leq 0$ that will enforce the constraint [22]. This is done by choosing

$$c(v) = \frac{(p_0 - v)^T}{\|p_0 - v\|} \tag{4.18}$$

and

$$d(v) = \|p_0 - v\| - \frac{(p_0 - v)^T}{\|p_0 - v\|} v - R \tag{4.19}$$

As the admissible domain contains a spherical hole, the attraction field (4.16) should be extended as [22]

$$\rho_O(v,r) = \rho(v,r) + \max\left(\frac{\zeta - (\|p - p_0\| - R)}{\zeta - \delta}, 0\right) (c(v) + \tilde{\rho}_O(v)) \tag{4.20}$$

with

$$\tilde{\rho}_O(v) = \operatorname{sign}\left(\rho_W^T(v,r)(p_0 - v)^\perp\right)(p_0 - v)^\perp \tag{4.21}$$

where $(p_0 - v)^\perp$ denotes the perpendicular vector. It can be noted that this vector is not uniquely defined in $\mathbb{R}^3$ but can be chosen arbitrarily perpendicular to a given vector (e.g. e_3).

Chapter 5

Implementation

After a control law has been selected (see Chapter 2) and a state estimator proposed (see Chapter 3), this section will present their implementation on the experimental testbench available at the SAAS department of ULB.

To begin, as a reminder of previous steps, the coplete and final control architecture is presented.

5.1 Architecture of the Solution

This section presents the final control architecture containing the primary stabilizing control, the reference governor and the state observer.

As it can be seen on Figure 5.1, there are two main physical components: the main, ground-based, controller (corresponding to the lab's PC) and the UAV. The part that will be implemented is the ground-based controller, comprising the Kalman filter (as state estimator), the position controller (as the stabilizing control) and the explicit reference governor (as the navigation layer).

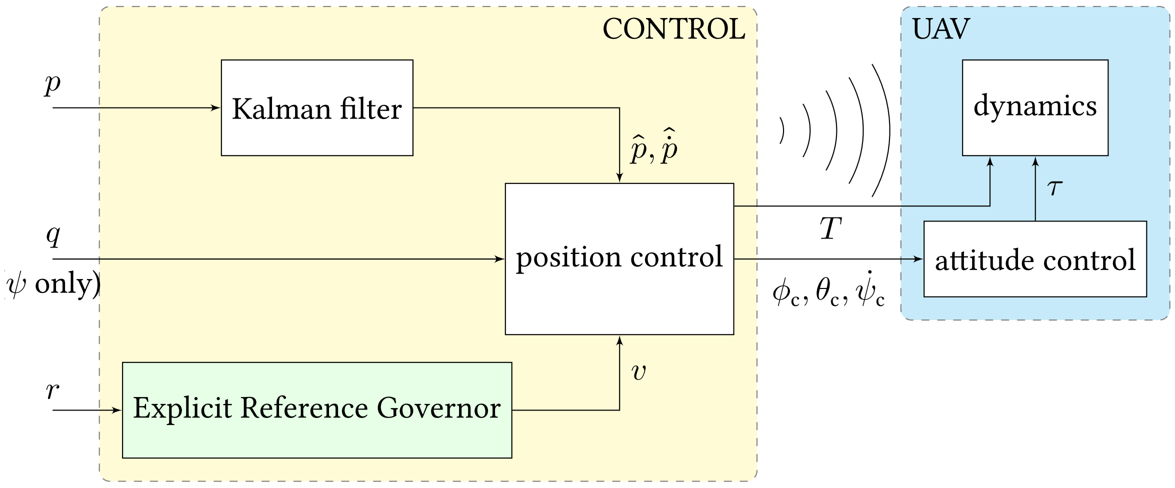


Figure 5.1: Complete control architecture.

5.2 Experimental Setup

In this section, the hardware and software used will be shortly introduced. First, the motion capture system for the position measurement is described. Then the basic workings of the two UAVs that have been successively used are presented.

5.2.1 Motion Capture System

The input data needed for the UAV's control, namely the position and attitude of the rigid body are provided by a motion tracking setup. This setup is an optical motion capture system by *Optitrack* [25]. The motion capture arena is composed of eight *Flex 13* (Figure 5.5a) cameras forming a tracking volume of a maximum of $3.8\,\mathrm{m} \times 3.8\,\mathrm{m} \times 3\,\mathrm{m}$. A render in *Motive* showing the camera placement can be seen in Figure 5.3.

This camera system can be run at an update rate of $120\,\mathrm{Hz}$ with a camera latency of $8.3\,\mathrm{ms}$ and a software latency of under a millisecond. If well calibrated, it can deliver a sub-millimeter accuracy for marker positioning. This characteristics make this system fast and precise and thus ideal for the tracking of a UAV, even during fast manoeuvres.

The *Motive* capture software receives the raw 2D data from the cameras and processes it to compute the 3D spatial position of the reflective markers. From there, it will compute the complete rigid body pose.

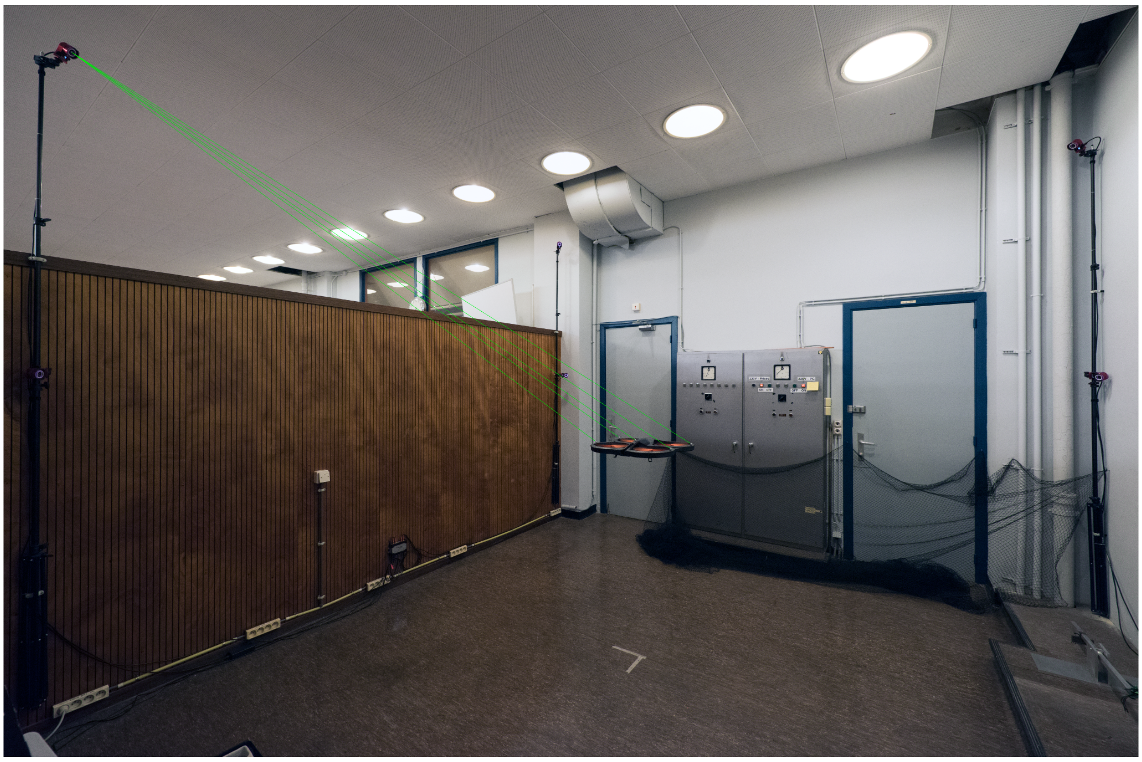


Figure 5.2: Lab setup: six of the eight motion capture cameras and, in the middle of the navigation space, a flying UAV are visible.

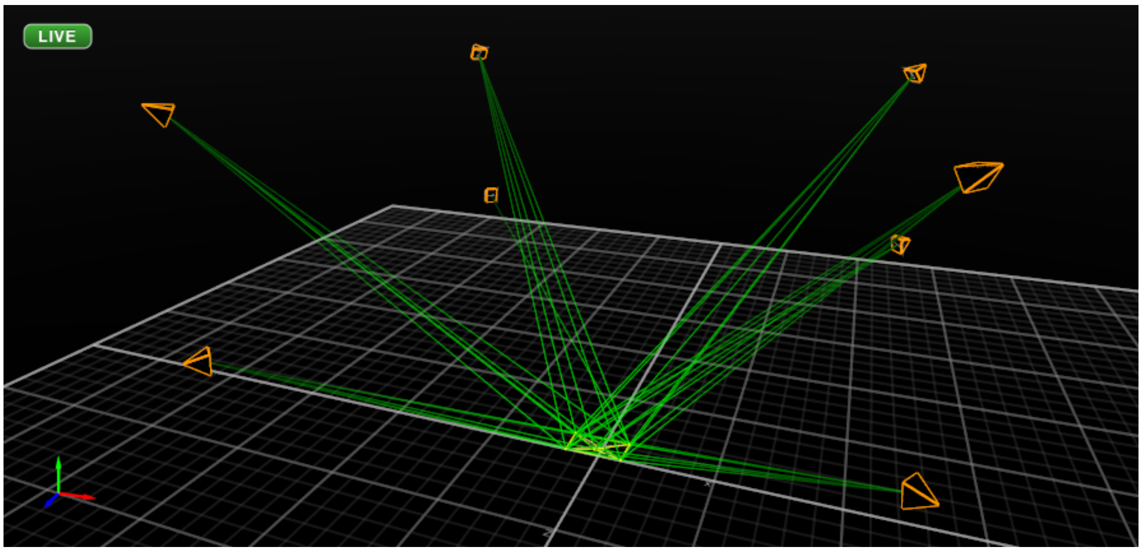


Figure 5.3: Render of the tracking volume in *Motive* showing the cameras (orange) placement and the tracked UAV (yellow). The rays (green) show which markers are visible by each camera.

5.2.2 First UAV: *RC Logger Xtreme*

The first rotorcraft used was a *RC EYE One Xtreme* from *RC Logger*. It has a typical quadrotor design with four 2-blade propellers spaced by about $160\,\mathrm{mm}$ for an approximate weight of $226\,\mathrm{g}$. It is actuated by DC brushless motors on which the propellers are directly mounted.

An on-board microprocessor coupled with an inertial measurement unit (IMU) provides the *Xtreme* with an attitude stabilizing control. It must be noted that this internal controller is very sensitive to the sensors calibration. This, combined to the fact that a good calibration was very difficult to achieve was not sufficient to keep the UAV in hover. As a result, the UAV was always drifting in one direction or another.

The quadrotor is usually operated through pulse-code modulation (PCM) signals using the stock transmitter or the *RC EYE OneLINK*, making it possible to use custom transmitters. The *Xtreme* is controlled using four channels: throttle, rudder, elevator and aileron. The throttle command controls the vertical thrust. The rudder sets the yaw-angle (z-axis) angular speed and, the elevator and aileron denote the rotations respectively around the y-axis and x-axis.

Communication and Command Mapping

To be able to connect the PC to the *RC EYE OneLINK* transmitter, a USB interface, converting the input controls to pulse-position modulation (PPM) signals, was needed. The USB interface[1] had to be controlled by a specific program that could receive the control inputs via the UDP protocol. Each PPM frame[2] has a duration of $20\,\mathrm{ms}$.

Prior to the implementation of the control algorithms, it was needed to map the true commands, expressed in SI units, to the corresponding PPM values.

5.2.3 Second UAV: *Parrot AR.Drone*

As a result to the problems described below (section 5.3), a second UAV was introduced: the *Parrot AR.Drone 2.0*. The *AR.drone* is commonly used by research groups and in academic labs because of its relatively low cost and, more importantly, the availability of an open-source API and extensive documentation for developers.

[1]The PCtoRC interface was used, available at [26].

[2]A frame denotes the complete signal that contains the pulses for all the different channels.

As the *Xtreme*, the *AR.drone* has a typical design with four 2-blade propellers which are actuated by brushless DC motors. Compared to the *Xtreme*, it is slightly bigger and heavier with a size of $517\,\mathrm{mm} \times 517\,\mathrm{mm}$ and a weight of $420\,\mathrm{g}$ with its indoor hull.

The sensory available is also more extensive comprising a 3-DOF accelerometer, a 3-DOF gyroscope, an ultrasound based altimeter, a pressure sensor and a 3-DOF magnetometer for absolute measurement, and two integrated cameras, on the front and bottom.

The *AR.drone* is equipped with an on-board controller which stabilizes its attitude and provides basic manoeuvres, like take-off and landing. Like its *RC Logger* counter-part, the attitude controller is pretty sensitive to the IMU calibration and, as such, depending on the quality of the calibration, it was rarely able to keep the UAV in an hovering state.

Communication

The communication with the *AR.drone* is done through WiFi mainly using the UDP protocol. The control and configuration of the drone is achieved by sending *AT commands*, ASCII-encoded strings that follow a precise format. The documentation recommends that the commands are sent every $30\,\mathrm{ms}$. Navigation data and other information (e.g. battery status, speed, attitude) about the drone status are sent by the drone through *navdata* packets. Those packets are sent at a rate of, depending on the mode, $15\,\mathrm{Hz}$ or $200\,\mathrm{Hz}$.

The commands must be normalized with regard to the maximum admissible values before sending. These maximum values can be fixed by sending the appropriate command.

5.3 First Attempts: *RC Logger Xtreme*

This section presents the different attempts to a working implementation of the stabilizing control algorithm described in Chapter 2 applied to the control of the *RC EYE One Xtreme*.

5.3.1 *Simulink* Implementation

A first implementation using *MATLAB* and *Simulink* was made based on previous work in the lab [27]. The problems with the implementation showed in [27] were:

- the presence of errors in the control algorithms;
- a lack of a good velocity estimation;

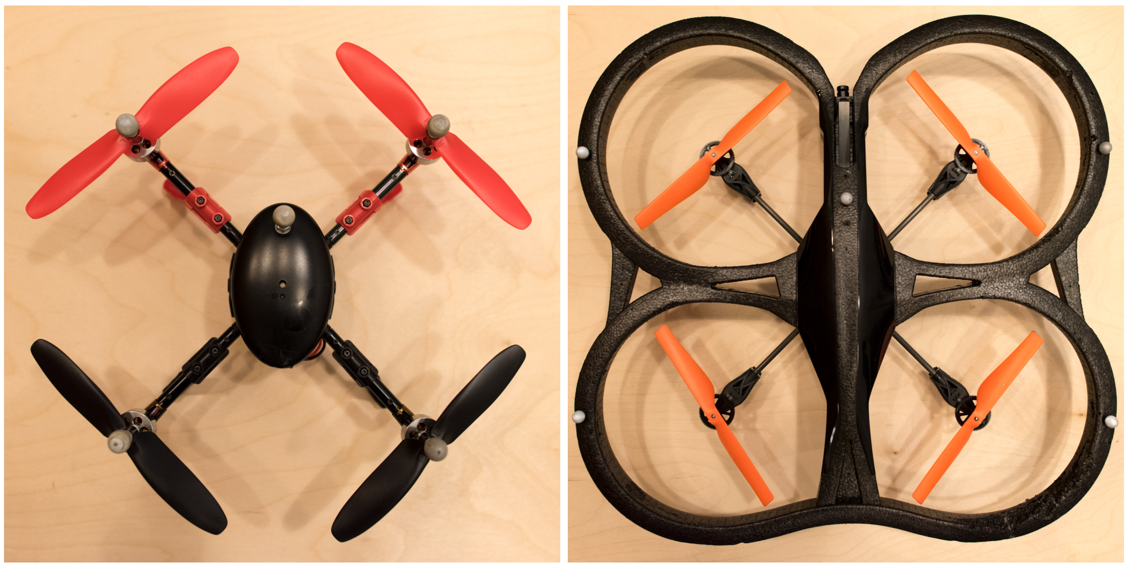

(a) RC Logger Xtreme. (b) Parrot AR.Drone 2.0.

Figure 5.4: Hardware: UAVs ready for tracking. Each one has five reflective markers, one is always asymmetric so that the software can distinguish the front from the back.

- that it was relying on yet another software to make the conversion of the *NatNet* packets and resend them on UDP in a new format to be received by the *Simulink* socket. Therefore limiting the update rate and increasing the latency.

The errors were corrected and the Kalman filter introduced for the velocity estimation. The updated implementation was now relying on the *MATLAB* .NET interface provided by the *Optitrack*'s *NatNet SDK* to receive the position and attitude data from *Motive*. This achieved good but not sufficient improvement on the latency. It was finally discarded because of the limitations of the .NET interface and the difficulties to maintain it.

5.3.2 *Python* Implementation

As a result, it was first considered to work with the Robot Operating System (ROS) for its obvious specialization but was discarded because of the difficulties and limitations arising from the fact that it runs exclusively on Linux while the tracking software *Motive* only runs on MS Windows platforms. It was thus finally decided to work with *Python* because it is widely used and there are a lot of libraries available.

This implementation removed the middle-man in the reception of the *NatNet* packets by using direct depacketization, reducing the measurement latency and allowing full access of the frame data. The details of the *Python* program will be further described in a next section.

Unfortunately, this did not suffice to be able to correctly stabilize the *Xtreme*. With the current implementation, the delays at the measurement side were reduced to a minimum: remaining only the latency of the cameras of 8.3 ms [28], the point-cloud and rigid-body pose computation of under a millisecond and the UDP loopback, that could not be measured but is usually also under a millisecond.

To try to characterize the delays at the command side, a measure of the inner loop response was taken. To realize this measure, the UAV was fixed to a special base that lets the UAV freely rotate while keeping it at a fixed position (Figure 5.5b). This allows the UAV to remain in orientations normally unstable.

With this setup, an input step of $\phi_c = 20°$ on the roll angle is applied. The resulting response can be seen on Figure 5.6. By analyzing the graph, it can be seen that the delay between the reference command and the attitude reaction, i.e. the instant when the UAV begins to move, is about 300 ms. This delay is too long to be able to stabilize the UAV. It must be noted that this delay comprises the measurement loop but as seen previously, the measurement latency is very low.

(a) IR camera: close-up.

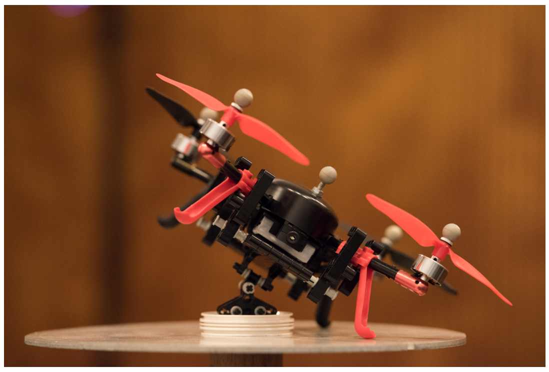

(b) Setup for inner loop identification.

Figure 5.5: Hardware.

As a comparison, the inner loop response of the *AR.Drone* is show in figure 5.7. The same test setup could not be achieved easily, therefore, the data show here is taken from a normal flight. To be noticed, the same communication delay is always under 100 ms.

5.4 Final Implementation

As a result to the delay problems described in the previous section. It was decided to use another platform: the *Parrot AR.Drone*. As alreday shown in section 5.3, the command

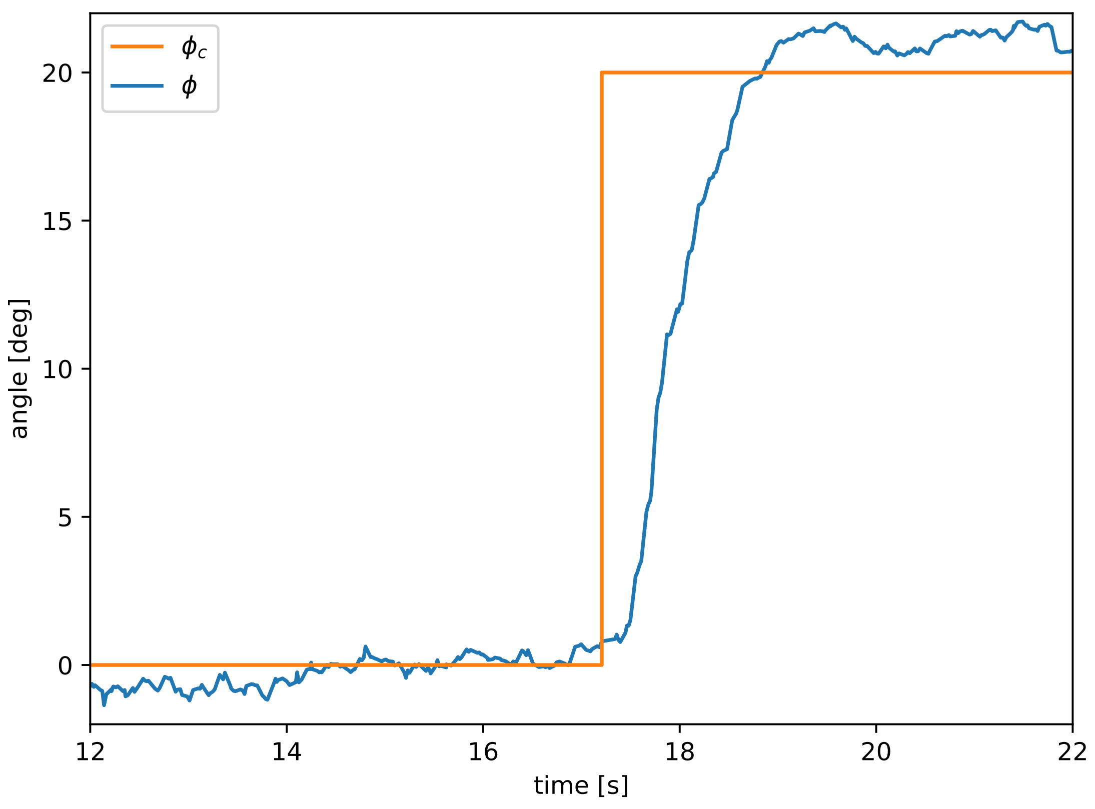


Figure 5.6: *RC Logger Xtreme*: inner loop response to a step input on the roll angle ϕ.

latency was greatly reduced and the same stabilization difficulties were not encountered.

This section will thus focus on the presentation of the final implementation used to control the *Parrot AR.Drone*.

5.4.1 Software Pipeline Overview

The complete control system and data flow can be seen in Figure 5.8.

First, Motive receives the raw image data from the IR cameras through the USB interface. The software will then reconstruct the 3D points and then compute the rigid body position and attitude. Second, the Python program receives the rigid body pose through UDP using the *NatNet* protocol. Using the Kalman filter, the translational velocity is estimated and the position filtered. The outerloop control will then compute the desired attitude and thrust, taking into account the position reference. Finally, the quadrotor receives

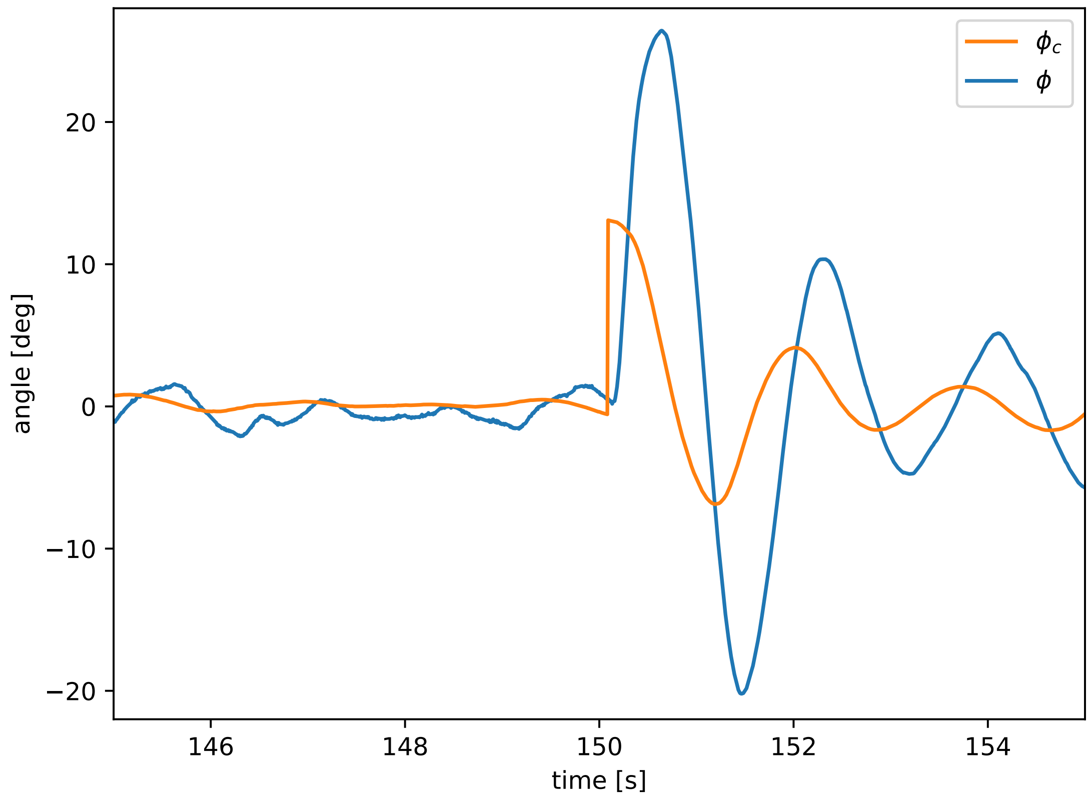


Figure 5.7: *Parrot AR.Drone*: inner loop response to a variable input on the roll angle ϕ.

the input attitude and thrust. Internally, the four rotor speeds necessary to achieve the desired rotation and displacement are set.

5.4.2 Python Program

The Python program has been written to be the more modular possible. Each of the main components being part of its own module. The program does extensive use of classes and relies and multiprocessing to keep the slow graphical user interface (GUI) separated from the main control loop. The main control loop is able to run sufficiently fast as to catch every frame sent by the camera setup (120 Hz).

The main components are:

- mainloop: the main control loop managing the calls to each class and function;
- natnetclient (modified from [29]): it provides a class that manages the input position

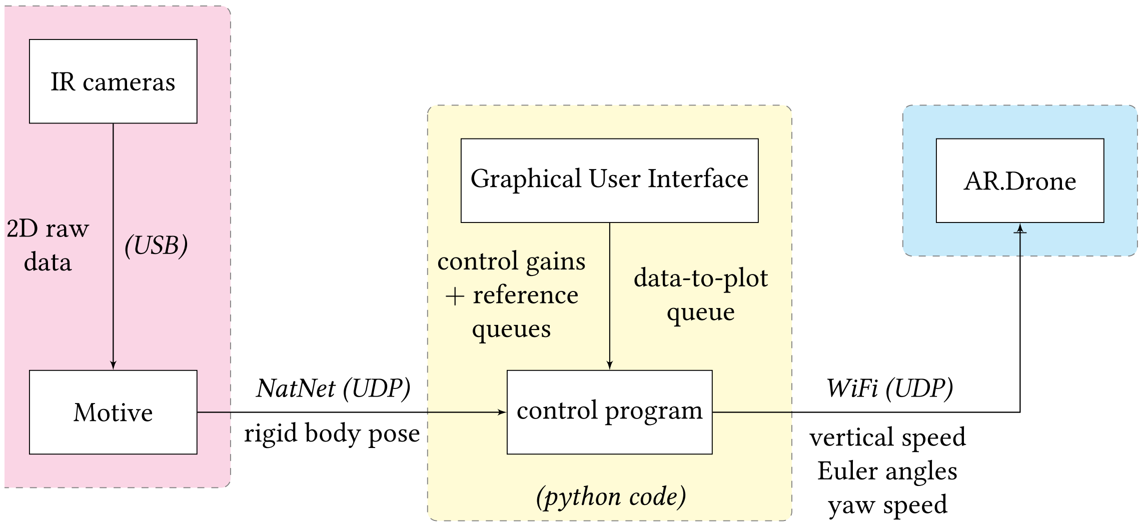


Figure 5.8: Complete control implementation and data flow.

and attitude;

- kalman_filter: implements the Kalman filter update equations and internally contains the states of the prediction model;
- uav_control: implements the outerloop control as well as the yaw controller;
- ardrone_wrapper (extends the pyardrone class from [30]): the class works as a high-level API to communicate with the drone, defining simple commands;
- gui: provides a simple GUI for real time tuning of the gains, modification of the reference and visualisation of the UAV states.

The main achievements of this new implementation are

- to reduce the latency of the input signals;
- to lower the computations delay, being able to run at 120 Hz, eliminating any frame loss;
- to be more easily maintainable and more adaptable.

The last source code can be found at `https://github.com/ehermand/saas-uav-ulb`.

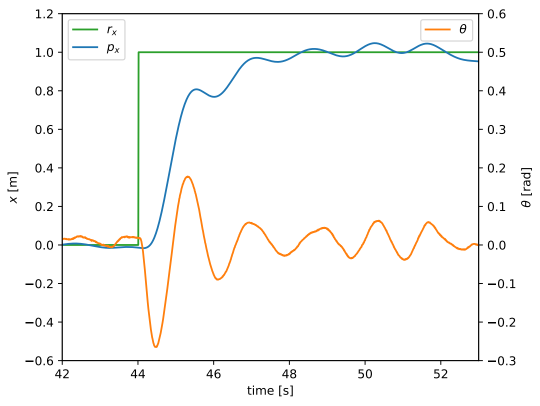
Chapter 6

Experimental Validation

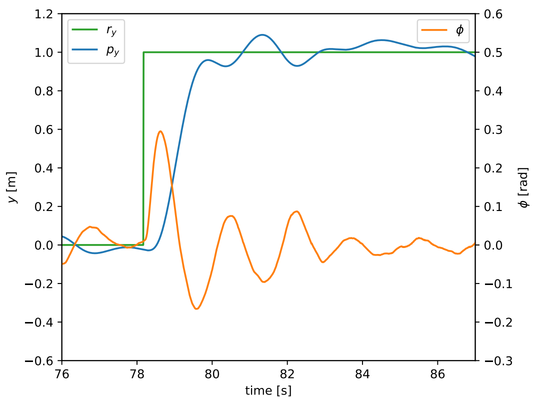
This final chapter focuses on the final results of a practical implementation in order to control and constrain an *AR.Drone* quadrotor. The first section presents the result of the stabilizing control.

6.1 Stabilizing Control

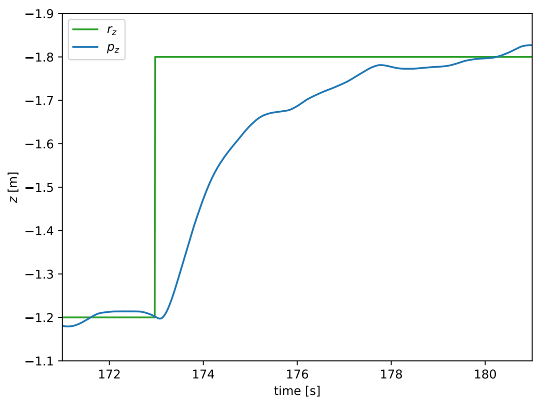
The purpose of this section is to show the results of the implementation of the stabilizing control layer prior to the introduction of the ERG framework.



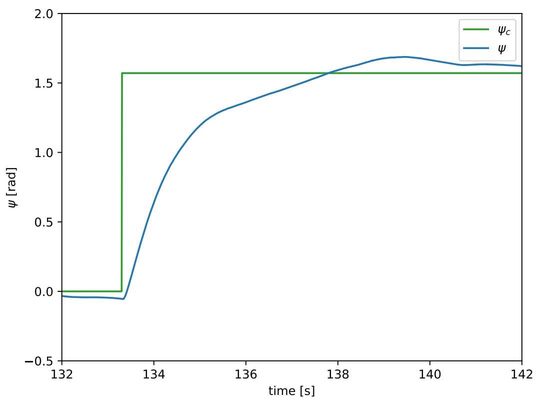
(a) Position x and pitch angle θ.



(b) Position y and roll angle ϕ.



(c) Altitude z.



(d) Yaw angle ψ.

Figure 6.1: Outer loop response to step inputs.

The resulting responses are quite satisfactory. The main focus was put on the x and y position for which the control is relatively fast.

The dynamics of the altitude and yaw are very different from the movement in the xy plane due to the unidirectional thrust of the quadrotor. It is believed that those response time could be improved by changing the input saturations defined in the *AR.Drone*.

6.2 Model Identification

To be able to implement the ERG, it is first needed to compute a model estimate of the stabilized system. It was decided to use a black box modelling approach. This was achieved by using the *N4SID* algorithm.

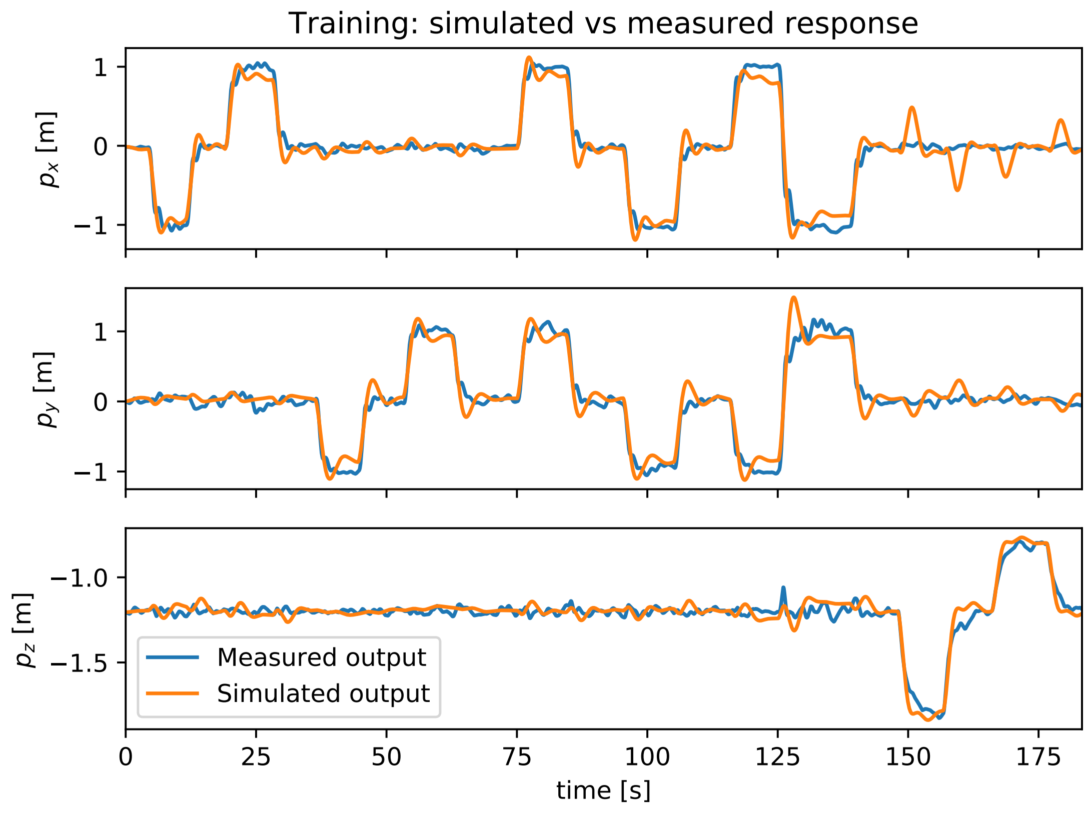


Figure 6.2: Training data for the decoupled system: measured vs simulated output.

Two sets of data were recorded: the first is used to train the model and the second one is used to validate it.

To realize the model, two different approaches are tested and compared. In the first case, the system position dynamics are considered decoupled in the three inertial axis. This leads to three 2$^{\text{nd}}$ order systems that can be combined to describe the entire position dynamics. In the second case, the potential coupling effects are taken into account by considering the entire set of inputs and outputs during the estimation process.

Model type	Learning data	Validation data
Coupled dynamics	[75.99% 74.05% 73.19%]	[71.64% 68.95% 78.37%]
Decoupled dynamics	[80.15% 77.70% 72.58%]	[82.84% 75.61% 73.41%]

Table 6.1: Fit quality assessment: normalized root mean square error.

After having constructed a satisfying model, a similarity transform must be performed

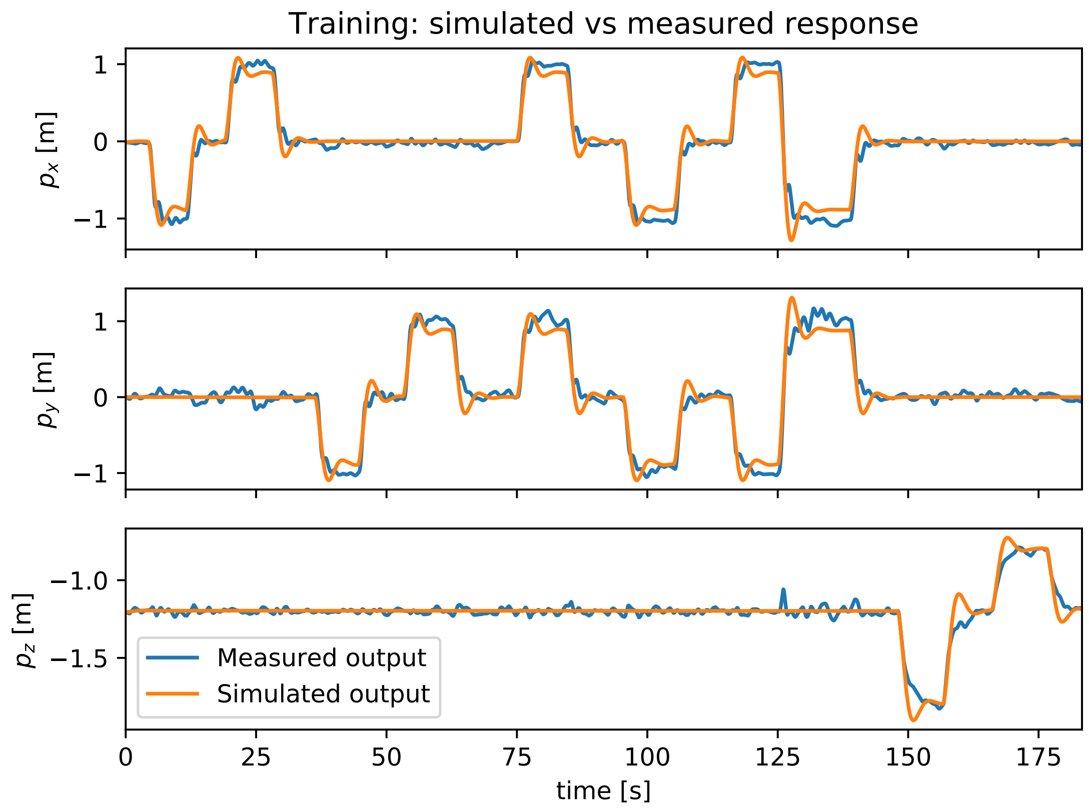


Figure 6.3: Training data for the coupled system: measured vs simulated output.

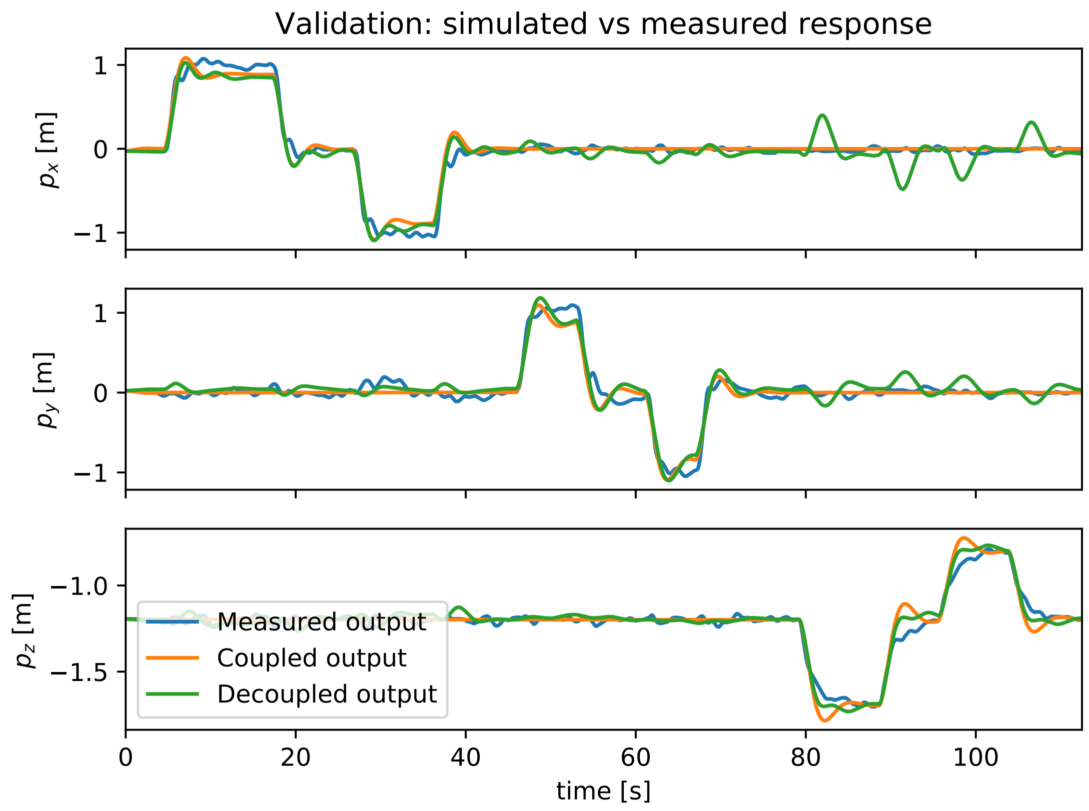


Figure 6.4: Validation data: measured vs simulated output for the coupled and decoupled case.

in order to produce an equivalent state-space model with as state vector:

$$\mathbf{x} = [p_x \; p_y \; p_z \; \dot{p}_x \; \dot{p}_y \; \dot{p}_z]^T. \tag{6.1}$$

6.3 Navigation Layer

This section presents the results of the implementation of an explicit reference governor on a pre-stabilized UAV system subject to a position constraint. Two main aspects are studied: the effect of the ERG on the system performance and the constraints enforcement of the ERG.

The ERG has been implemented given the wall constraint (4.14) with

$$c^T = \frac{1}{\sqrt{2}}[1\;1\;0] \quad \text{and} \quad d = \frac{1}{\sqrt{2}}, \tag{6.2}$$

the influence margin $\zeta = 0.5\,\text{m}$ and the static safety margin $\delta = 0.2\,\text{m}$.

The κ gain was chosen as 1, to give the better control performance while avoiding any numerical issues. The smoothing radius η is 0.01.

6.3.1 Effect on Closed-Loop Performance

The closed-loop performance is analyzed far from the positional constraints.

In Figures 6.5 and 6.6, the response of the controller with ERG is compared to the one of the stabilized controller without ERG. The performances are greatly reduced in the case with the ERG. This is due to the fact that, in this case, an arbitrary Lyapunov function has been chosen and thus, the ERG working is not optimal.

6.3.2 Constraints enforcement

Figure 6.8 shows the behaviour of the governed system for a static reference input that violates the constraints. As it is expected, the UAV navigates towards the best steady-state admissible position.

In figure 6.9, three consecutive steps are applied to the reference. The resulting position response is as expected and kept at all times in the constraints.

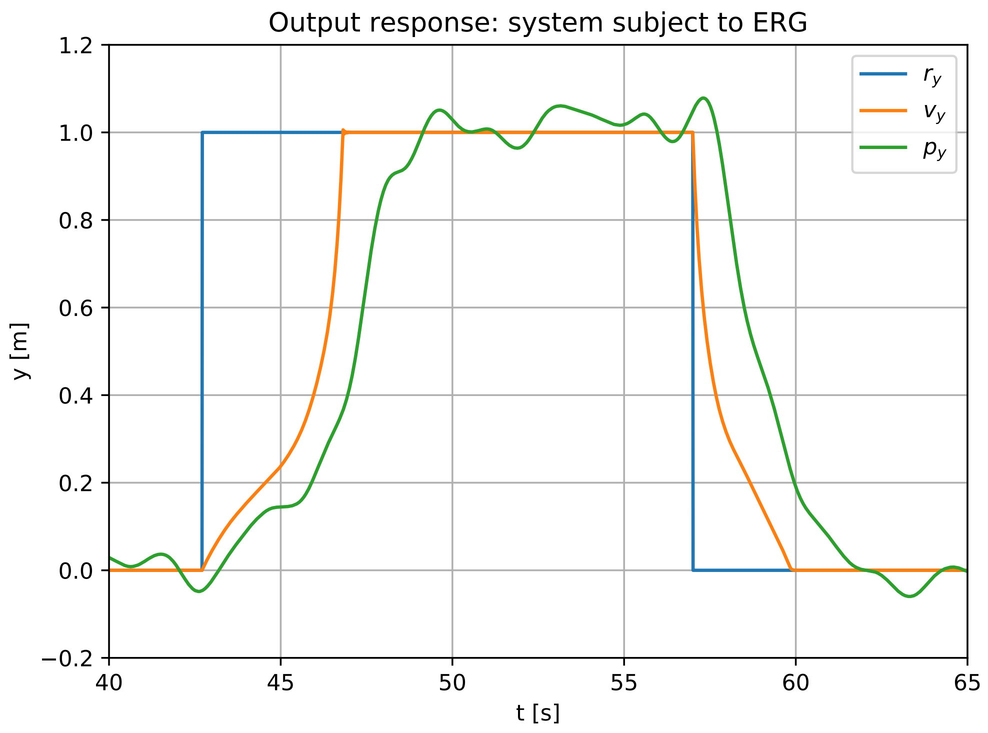


Figure 6.5: Step response in the y-axis: system subject to ERG (far from constraints)

To further test the robustness of the constraints' enforcement, a test flight simulating a more aggressive usage is illustrated on figure 6.10.

Figure 6.11 shows the UAV's trajectory in the case of multiple constraints. For this experiment, four wall constraints forming a fully closed convex admissible region are used. As it can be seen, the reference governor manages pretty well the corner cases where it must take into account multiple constraints.

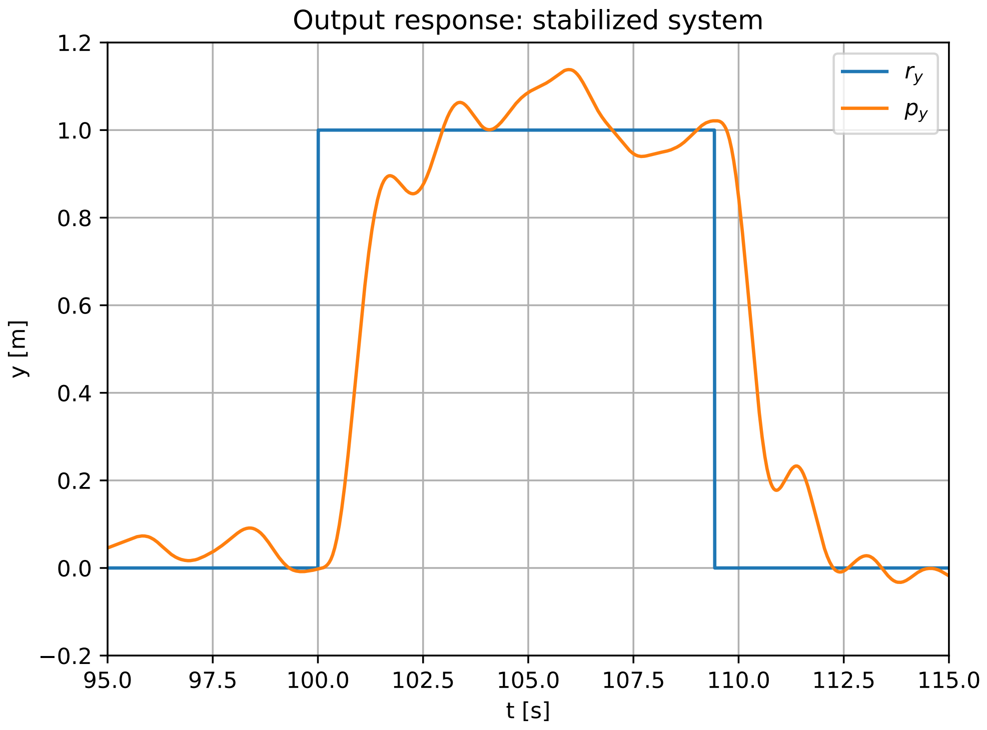


Figure 6.6: Step response in the y-axis: stabilized system

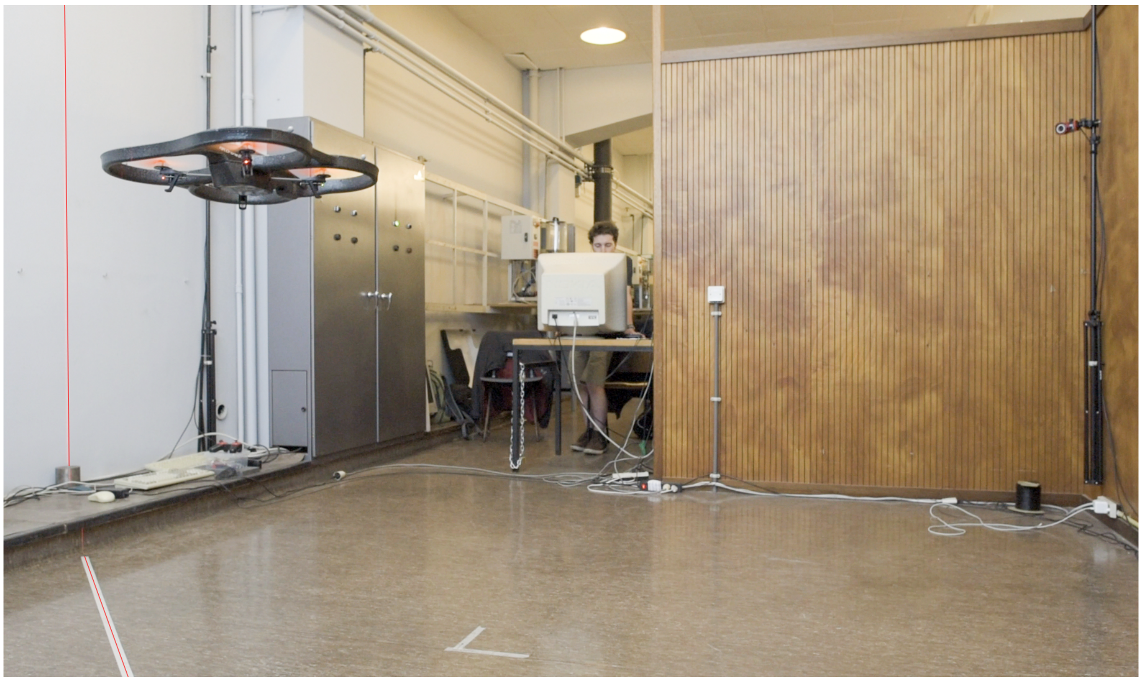


Figure 6.7: Experimental setup showing the virtual wall with a laser.

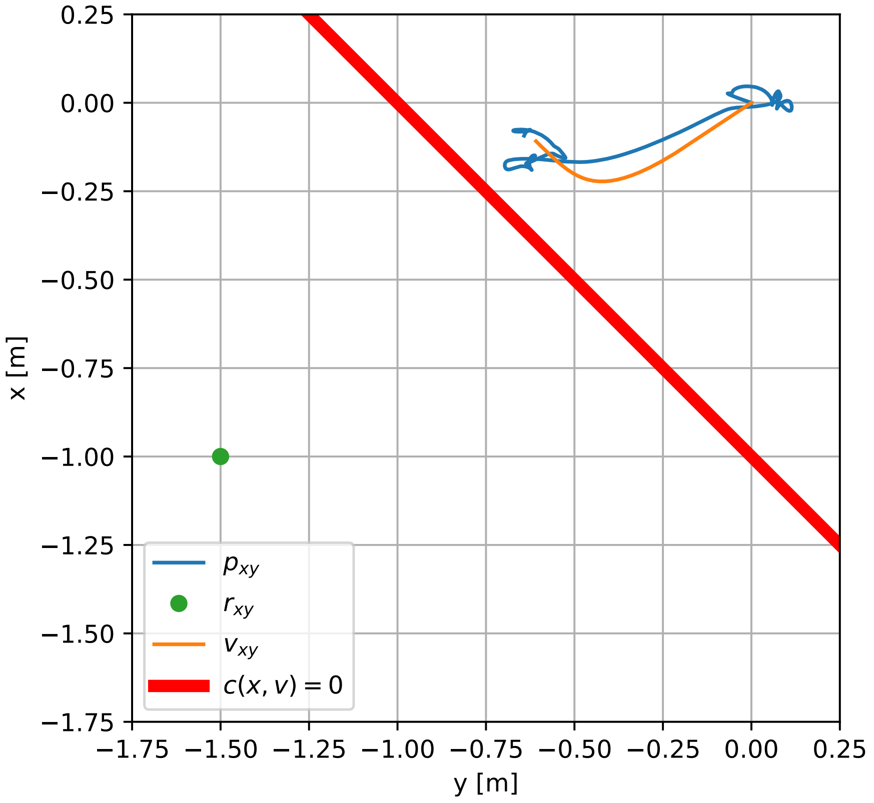


Figure 6.8: System response in the xy-plane in presence of a wall subject to a reference step.

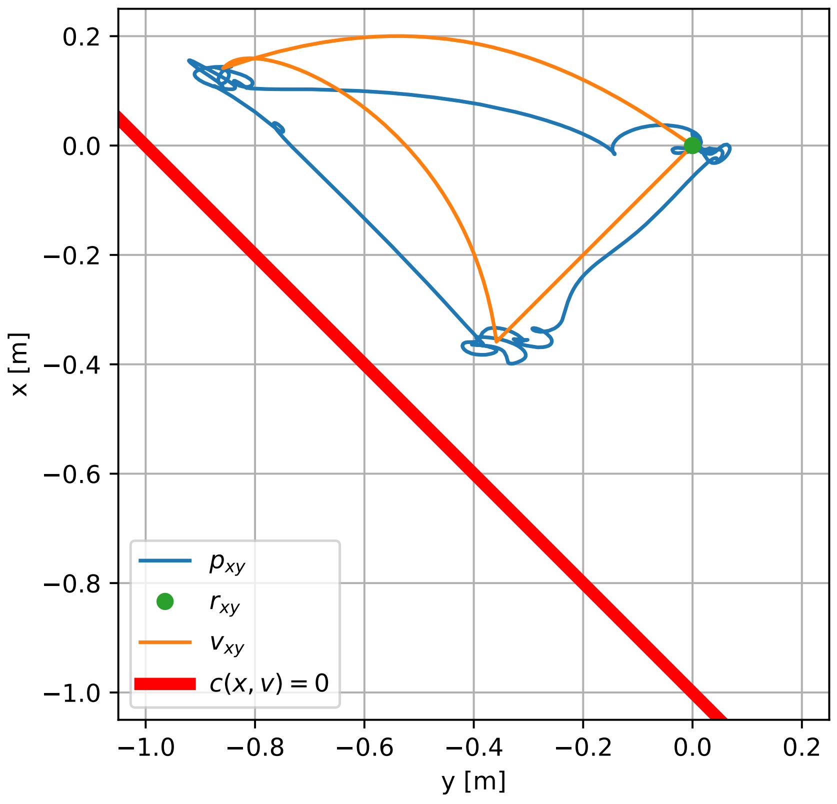


Figure 6.9: System response in the xy-plane in presence of a wall subject to 3 consecutive reference steps.

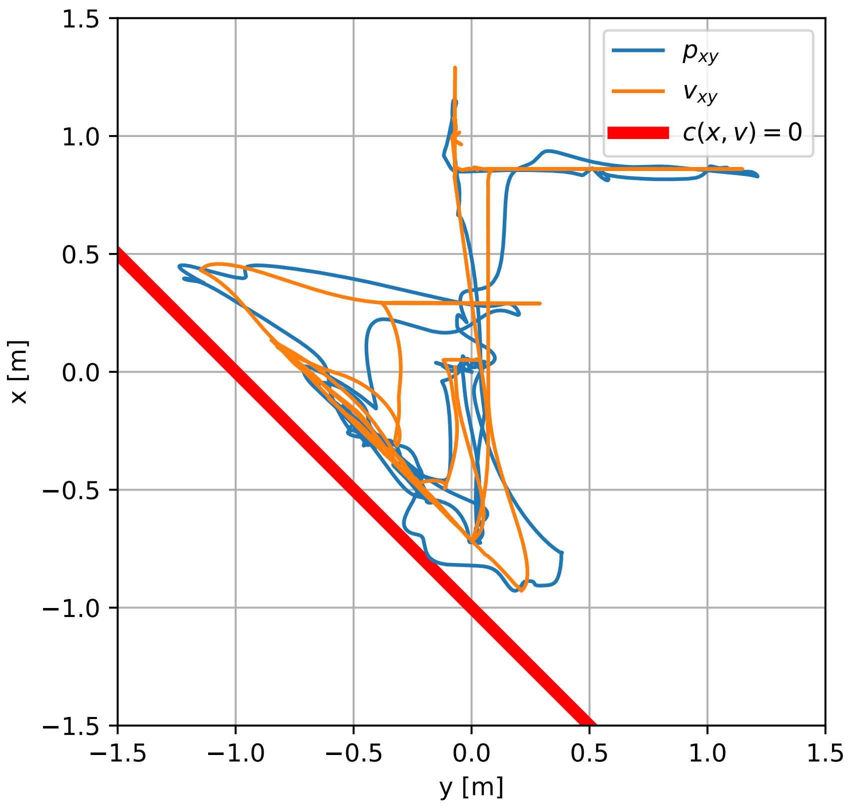


Figure 6.10: System response in the xy-plane.

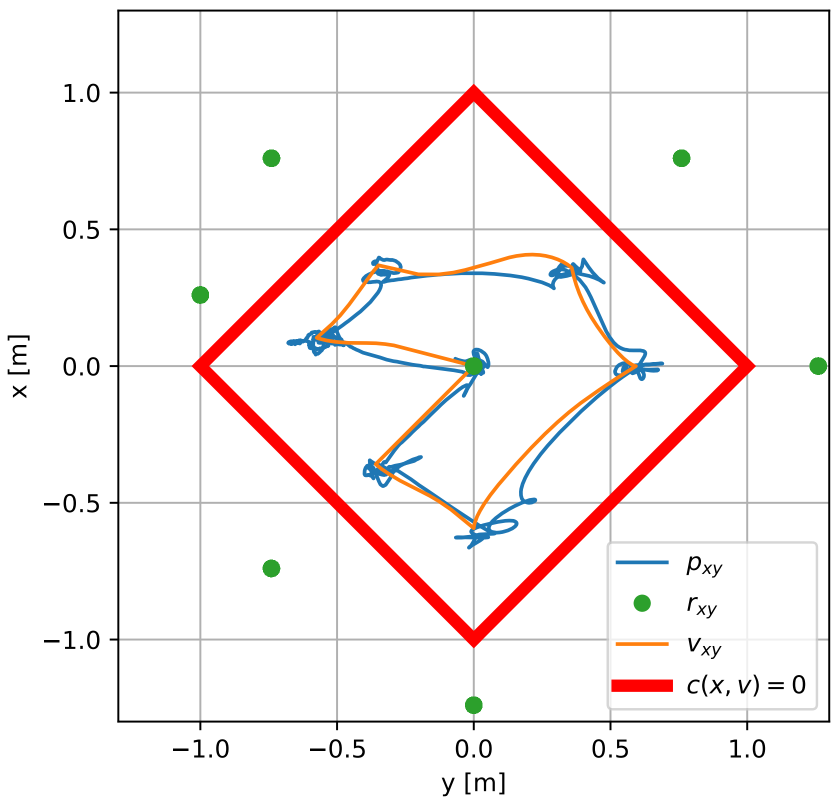


Figure 6.11: System response in the xy-plane in presence of four walls.

Chapter 7

Conclusion

This master thesis proposes a practical development and validation of a fully functional controller, implementing an explicit reference governor in order to enforce flight constraints in a world with obstacles.

In Chapter 1, the state of the art of the different UAV control systems has been analyzed, focusing on the constrained control, and the current available development of the explicit reference governor scheme described.

The first part of the controller, the stabilization control layer is presented in Chapter 2. As a preliminary step, a dynamic model which describes the behavior of the UAV has been presented. Then, a cascade control scheme was described with an inner loop for the attitude control and an outer loop for the position control.

In order to implement the control scheme presented in Chapter 2, a state estimator had to be constructed. As the controller works on the position and velocity, and the experimental setup does not allow direct measurement of the velocity, the implementation of a Kalman filter was proposed. The prediction model used and its tuning and validation through a pendulum experiment have been described.

Chapter 4 dealt with the navigation layer or explicit reference governor add-on. The role of the navigation layer is to supplement the already stabilized system with obstacle avoidance capability. The ERG developed relies on Invariant Lyapunov level-sets to achieve constraints enforcement without solving an online optimization problem. Two type of obstacles were considered: wall and spherical obstacles.

Chapter 5 presented the real-time implementation. To begin, an overview of the experimental setup was given, presenting the two commercially available quadrotors used. A first rotorcraft, the *RC Logger Xtreme* relying on a USB interface coupled with a middle-ware for the interface with the PC, presented some timing difficulties in order to be stabilized. Multiple attempts to stabilize this quadrotor were done, using first a *Simulink* implementation which presented too much latency as well as difficulties to be maintained. As a result, a new approach with a Python program was then implemented which reduced drastically the latency at the measurement side while greatly reducing the computation delays. However, the only caveat remaining was the middle-ware interface. Some testing showed that the remaining latency, due to the interface (more likely) or the on-board controller (less likely), was still to high for the correct stabilization of the *Xtreme*. Another UAV platform was then considered: the *Parrot AR.Drone* which resulted relatively easy to stabilize using the Python software already developed.

Finally, the experimental results have been presented in Chapter 6. First, the case of a single wall constraint has been studied. Multiple flight missions where presented with desired reference violating the constraints or not. The behaviour of the drone showed a good working of the ERG. The modified reference each time converging to the better admissible steady-state point. Experiments implementing multiple constraints, four walls fully enclosing the UAV, also showed a good behaviour of the modified reference and subsequently, of the drone.

All the different steps that must include the UAV's system controller were declined for our specific case as outlined in general terms above. The novelty lies in the validation tests in real scale obstacle avoidance scenarios (wall and sphere) within a defined tracking volume. This main achievement will set the basis to consider further this solution as a viable alternative control design method avoiding to solve the on-line optimization problem. This thesis has shown and demonstrated in practice how the explicit reference governor can be used as an add-on tool to provide constraint handling capabilities to an unmanned aerial vehicle.

In the author's opinion, this experimental validation demonstrated the great simplicity and good performance of the explicit reference governor. The proposed framework is therefore believed to be of real interest for many applications that require a simple, systematic, and computationally inexpensive add-on tool for non-linear constraint enforcement.

Another important product of this work is the Python implementation that can be used as a basis for other experimental studies using the motion capture lab.

Future development dedicated to improve performance of the ERG scheme developed and validated in this work might concern using genetic algorithms to optimize control Lyapunov function to increase the performances of the ERG.

Appendix A

MoCap Lab: User Guide

A.1 Motive

A.1.1 Calibration

The use[1] and set-up of the Motive software by Optitrack is also explained in the detailed video tutorial: `https://www.youtube.com/watch?v=cNZaFEghTBU`.
⚠ The cameras must always be turned on before Motive is launched. If Motive is launched before the cameras, they will not be able to synchronize correctly. Also, if Motive is opened and then closed, it will be necessary for the user to restart the cameras.

1. Launch Motive and choose "Perform camera calibration". A menu opens on the right of the window.
2. Click on the "Mask Visible" button. Red dots will appear on the cameras view, masking the white dots which could be, for example, sources of light in the environment.
3. Take the "Wand" (it is a T-like stick with 3 markers) and click on "Start Wanding". Then, walk around the space covered by the cameras while moving the wand around. Be aware to cover the entire volume.
4. When the view of each cameras on the screen is well covered, enough data has been acquired for the calibration. Click on "Calculate". As soon as the calculation are completed, some statistics of the tracking, like volume accuracy, will be displayed. If those are not sufficient, restart the process from step 3. Otherwise, click on "Apply results" and save the calibration settings.

[1]This section of the User Guide is based on [27]

5. To define the ground plane, the L-frame provided with the system must be used. Note that the z direction indicated on it corresponds to the x-axis on the ground. Once the L-frame has been positioned in the center of the room, the user can press on the “Set ground plane” button.
6. Finally, save the project for future use.

Note: The following steps must only be followed if it is the user’s first use of Motive. The next time Motive will be opened, the user can choose the “Open already existing project” option. However, if a camera is moved or, if the tracking has degraded, the calibration should be redone.

A.1.2 Project Set-Up

To be able to track the vehicle, it is necessary to create a rigid body:

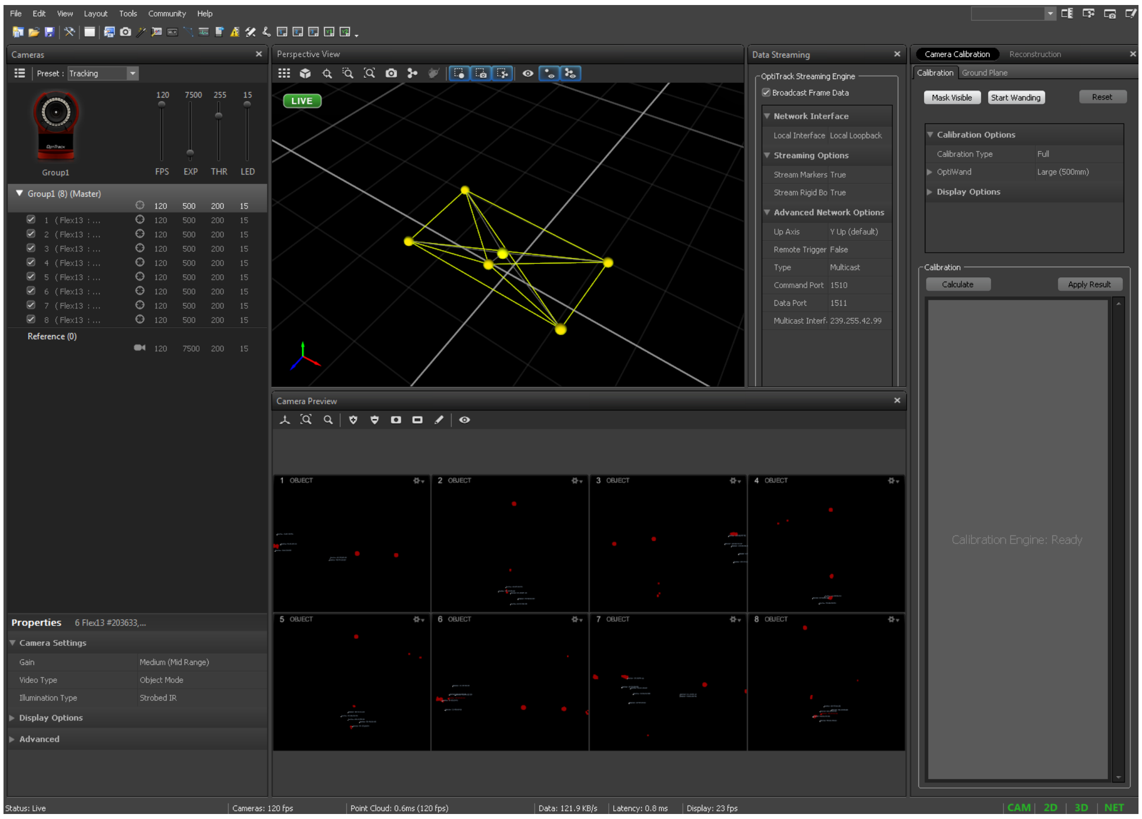
1. Make sure that the UAV is placed as flat as possible (and at level) in the space covered by the cameras.
2. White dots corresponding to the markers appear on the screen. The user must select them, then right click and choose “Create rigid body from selected markers”.
3. When this step is performed, the attitude of the UAV is initialized.

To send the data through the NatNet protocol, the data streaming must be activated:

1. To open the data streaming option panel (see figure A.1), click on “View”, then on “Data streaming”.
2. To actually stream data, tick “Broadcast Frame data”.
3. Specify the local interface by selecting “Local Loopback” (or eventually the local IP: 192.168.0.65). Make sure that this corresponds to the value set in main.py (Local Loopback is the same as 127.0.0.1).
4. In “Advanced Network Options”, select “Multicast” as type.

A.2 Drone Set-Up

Depending on the UAV you will use, different steps are needed. The first section presents the set-up of the Parrot AR.Drone, the second, the one of the RC Logger Xtreme.



Data Streaming
OptiTrack Streaming Engine
Broadcast Frame Data
Network Interface
Local Interface Local Loopback
Streaming Options
Stream Markers True
Stream Rigid Bo True
Advanced Network Options
Up Axis Y Up (default)
Remote Trigger False
Type Multicast
Command Port 1510
Data Port 1511
Multicast Interf. 239.255.42.99

Calibration Options
Calibration Type Full
OptiWand Large (500mm)
Calibration Engine: Ready

Camera Settings
Gain Medium (Mid Range)
Video Type Object Mode
Illumination Type Strobed IR

Status: Live | Cameras: 120 fps | Point Cloud: 0.6ms (120 fps) | Data: 121.9 KB/s | Latency: 0.8 ms | Display: 23 fps

Figure A.1: Motive Interface.

A.2.1 Parrot AR.Drone

To initialize the AR.Drone, the user must:

1. Connect the battery.
2. Place the UAV in the center of the room, on a flat surface.
3. Connect the PC to WiFi: ardrone2_108176.

⚠ If the Python program is not launched quickly enough, the UAV goes in timeout. In this case, disconnect the WiFi and reconnect.

A.2.2 RC Logger Xtreme

To initialize the RC Logger Xtreme, the user must:

1. Launch PCtoRC.exe which is in the “PCtoRC (latest)” folder.

2. Connect the USB cable of the RC EYE oneLINK to the computer and turn it one ("PCtoRC connected" should appear in green in the PCtoRC window).
3. Connect the battery of the Xtreme. If the connection is established, the led on the top of the Xtreme should blink, the color depending on the mode (beginner in green, sport in orange).

For more details, refer to the user manual of the RC Logger Xtreme.

A.3 Python Program

1. Use the desktop link to launch the WinPython program prompt.
2. Go to the main code directory (where the main.py file is found) by typing "cd directory_name".
3. Launch the main program by typing "python main.py". The GUI (see figure A.2) will open.

Note: When the program is launched, the UAV is calibrated and should be on a flat surface.

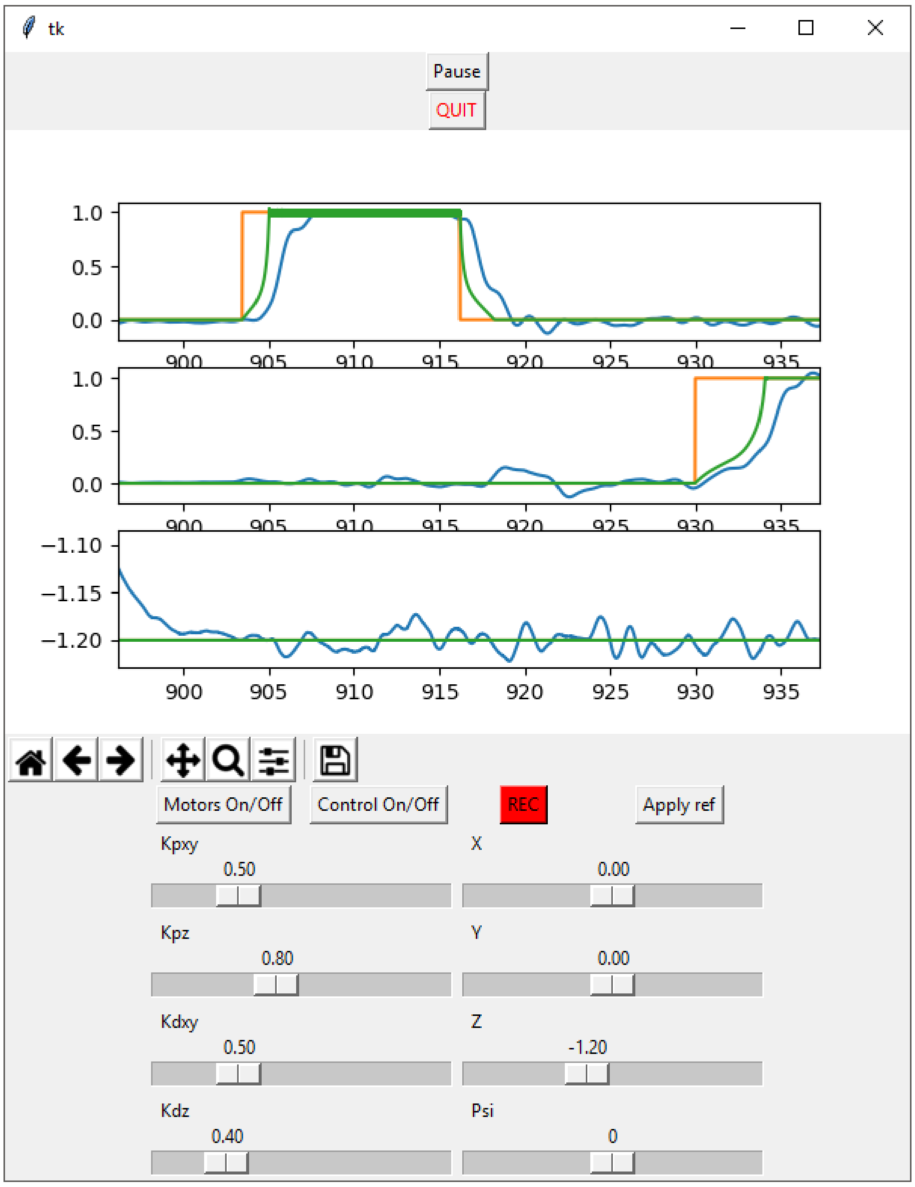


Figure A.2: Graphical User Interface (GUI) of the Python program

Motors On/Off: makes the UAV take off or land autonomously

Control On/Off: activates/deactivates the control strategy. If not in control, the UAV is in hovering mode. In this case, pay attention because the drone could be drifting.

REC: starts/stops the recording and automatically creates a file named with the date and time in "./recordings" directory.

Apply ref: toggles the update of the reference with current sliders' values.

Bibliography

[1] G. J. Vachtsevanos and K. P. Valavanis, "Military and civilian unmanned aircraft," in *Handbook of Unmanned Aerial Vehicles*, DOI: 10.1007/978-90-481-9707-1_96, Springer, Dordrecht, 2015, pp. 93–103.

[2] A. Zulu and S. John, "A review of control algorithms for autonomous quadrotors," *Open Journal of Applied Sciences*, vol. 04, no. 14, pp. 547–556, 2014.

[3] E. Garone and M. Nicotra, "Explicit reference governor for constrained nonlinear systems," *IEEE Transactions on Automatic Control*, vol. 61, no. 5, pp. 1379–1384, 2016.

[4] M. D. Hua, T. Hamel, P. Morin, and C. Samson, "Introduction to feedback control of underactuated VTOL vehicles: A review of basic control design ideas and principles," *IEEE Control Systems*, vol. 33, no. 1, pp. 61–75, Feb. 2013.

[5] N. S. Özbek, M. Önkol, and M. Ö. Efe, "Feedback control strategies for quadrotor-type aerial robots: A survey," *Transactions of the Institute of Measurement and Control*, vol. 38, no. 5, pp. 529–554, May 1, 2016.

[6] J. Li and Y. Li, "Dynamic analysis and PID control for a quadrotor," in *2011 IEEE International Conference on Mechatronics and Automation*, Aug. 2011, pp. 573–578.

[7] P. Pounds, R. Mahony, and P. Corke, "Modelling and control of a large quadrotor robot," *Control Engineering Practice*, Special Issue on Aerial Robotics, vol. 18, no. 7, pp. 691–699, Jul. 1, 2010.

[8] S. Bouabdallah, A. Noth, and R. Siegwart, "PID vs LQ control techniques applied to an indoor micro quadrotor," in *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, vol. 3, Sep. 2004, pp. 2451–2456.

[9] P. Castillo, R. Lozano, and A. Dzul, "Stabilization of a mini rotorcraft with four rotors," *IEEE Control Systems*, vol. 25, no. 6, pp. 45–55, Dec. 2005.

[10] A. Mokhtari, A. Benallegue, and A. Belaidi, "Polynomial linear quadratic gaussian and sliding mode observer for a quadrotor unmanned aerial vehicle," *Journal of Robotics and Mechatronics*, vol. 17, no. 4, pp. 483–495, Aug. 20, 2005.

[11] E. Prempain and I. Postlethwaite, "Static h∞ loop shaping control of a fly-by-wire helicopter," *Automatica*, vol. 41, no. 9, pp. 1517–1528, Sep. 1, 2005.

[12] S. Patra, S. Sen, and G. Ray, “Design of static h∞ loop shaping controller in four-block framework using LMI approach,” *Automatica*, vol. 44, no. 8, pp. 2214–2220, Aug. 1, 2008.

[13] Z. Fang and W. Gao, “Adaptive integral backstepping control of a micro-quadrotor,” in *2011 2nd International Conference on Intelligent Control and Information Processing*, vol. 2, Jul. 2011, pp. 910–915.

[14] P. Ru and K. Subbarao, “Nonlinear model predictive control for unmanned aerial vehicles,” *Aerospace*, vol. 4, no. 2, p. 31, Jun. 17, 2017.

[15] M. M. Nicotra and E. Garone, “Explicit reference governor for continuous time nonlinear systems subject to convex constraints,” in *2015 American Control Conference (ACC)*, Jul. 2015, pp. 4561–4566.

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

[17] J. Diebel, “Representing attitude: Euler angles, unit quaternions, and rotation vectors,” *Matrix*, vol. 58, no. 15, pp. 1–35, 2006.

[18] M. J. Baker. (). Euclideanspace - mathematics and computing, [Online]. Available: `https://www.euclideanspace.com/` (visited on 12/31/2017).

[19] C. Powers, D. Mellinger, and V. Kumar, “Quadrotor kinematics and dynamics,” in *Handbook of Unmanned Aerial Vehicles*, DOI: 10.1007/978-90-481-9707-1_71, Springer, Dordrecht, 2015, pp. 307–328.

[20] X. Zhang, X. Li, K. Wang, and Y. Lu, “A survey of modelling and identification of quadrotor robot,” *Abstract and Applied Analysis*, vol. 2014, pp. 1–16, 2014.

[21] P. Corke, *Robotics, vision and control - fundamental algorithms in MATLAB®*. 2017.

[22] M. Nicotra, “Constrained control of nonlinear systems: The explicit reference governor and its application to unmanned aerial vehicles,” PhD thesis, Université libre de Bruxelles, Sep. 13, 2016.

[23] J. V. Candy, *Signal processing: Model based approach.* New York, NY, USA: McGraw-Hill, Inc., 1986.

[24] R. Faragher, “Understanding the basis of the kalman filter via a simple and intuitive derivation [lecture notes],” *IEEE Signal Processing Magazine*, vol. 29, no. 5, pp. 128–132, Sep. 2012.

[25] Optitrack. (). Motion capture systems, OptiTrack, [Online]. Available: `http://www.optitrack.com/index.html` (visited on 01/14/2018).

[26] RC-ELEC. (2006). RC-ELEC radiocommande & électronique, [Online]. Available: `http://rc-elec.com/` (visited on 01/14/2018).

[27] C. Pochet, *Flight lab: User guide*, Apr. 25, 2015.

[28] Optitrack, *Flex 13: Technical specifications*, 2012.

[29] N. A. D. Grosso, *Natnetclient: Python NatNet client for retrieving NaturalPoint's optitrack data from their motive software using depacketization*, original-date: 2015-10-22T10:57:02Z, Oct. 12, 2017.

[30] Y. Li-Yu, *Pyardrone: Parrot AR.drone client library for python 3*, original-date: 2015-08-14T11:44:14Z, Oct. 31, 2017.