MPC-based Visual Servo Control for UAVs

ELISA BIN

Master in Systems, Control and Robotics
Date: July 9, 2020
Supervisor: Pedro Miraldo and Pedro Roque
Examiner: Dimos Dimarogonas
School of Electrical Engineering and Computer Science
Swedish title: Detta är den svenska översättningen av titeln

Abstract

Vision information is essential for planning and control of autonomous systems. Vision-based control systems leverage rich visual input for motion planning and manipulation tasks. This thesis studies the problem of Image-Based Visual Servo (IBVS) control for quadrotor UAVs. Despite the effectiveness of vision-based systems, the control of quadrotors with IBVS presents the non-trivial challenge of matching the 6 DoF control output obtained by the IBVS with the 4DoF of the quadrotor. The novelty of this work lies in addressing the under-actuation problem of quadrotors using linear Model Predictive Control (MPC). MPC is a well-known optimization control technique that leverages a model of the system to predict its future behaviour as a function of the input signal. We extensively evaluate the performance of the designed solution on both simulated environment and real-world experiments.

Sammanfattning

Visuell information är grundläggande för planering och kontroll av autonoma system. Visionsbaserade kontrollsystem drar nytta av rik visuell inmatning för rörelseplanerings- och manipuleringsuppgifter. Den här avhandlingen studerar problemet med Image-Based Visual Servo (IBVS) -kontroll för quadrotor UAVs. Trots effektiviteten hos visionsbaserade system utgör kontrollen av quadrotorer med IBVS den icke-triviala utmaningen att matcha 6 DoF-kontrollutgång som erhållits av IBVS med 4DoF från quadrotorn. Nyheten i detta arbete ligger i en ny formulering av underaktiveringsproblemet för quadrotorer med linjär Model Predictive Control (MPC). MPC är en välkänd optimeringskontrollteknik som utnyttjar en modell av systemet för att förutsäga dess framtida beteende som en funktion av insignalen. Vi utvärderar omfattande prestandan för den designade lösningen i både simulerad miljö och verkliga experiment.

Contents

Chapter 1

Introduction

In the last decades, automation and robotics have become an important field of study to shape the future of society. From intelligent home assistants to autonomous vacuum cleaners, this technological revolution is already around us. Autonomous vehicles, operated without any direct human intervention, are slowing becoming a reality. Because of its potential, this technology is being applied not only for transportation but also in agriculture, in rescuing people and in substitution of human operators performing tasks in dangerous environments. In robotics vehicles operated autonomously are typically referred to as Unmanned Vehicles (UV). When referring to an autonomous vehicle one can immediately think of a self-driving car or an autonomous rover, but self-piloted planes and drones belong to this category as well. Being more specific: the firsts are Unmanned Ground Vehicles (UGVs) while the seconds are called Unmanned Aerial Vehicles (UAVs). The use of autonomous areal vehicles is a very effective tool to substitute or support human operators in dangerous environments while performing monitoring tasks, rescuing straggler or delivering medical devices. To successfully perform its task, the UAV needs to gain knowledge and understanding of the surroundings, guessing its position and realizing which direction should be taken to reach its goal. To be able to *"see"* the environment around, the vehicle needs to be equipped with camera sensors that act as *"eyes"* for it.

UAVs are already in use in many different contexts such as photography, movies production, inspection and agriculture. Interesting research areas are trying to investigate the use of drones for noble causes such as helping to rescue people in need, trapped after an avalanche or after an accident in a dangerous environment. In particular, on this topic it is worth mentioning the SHERPA

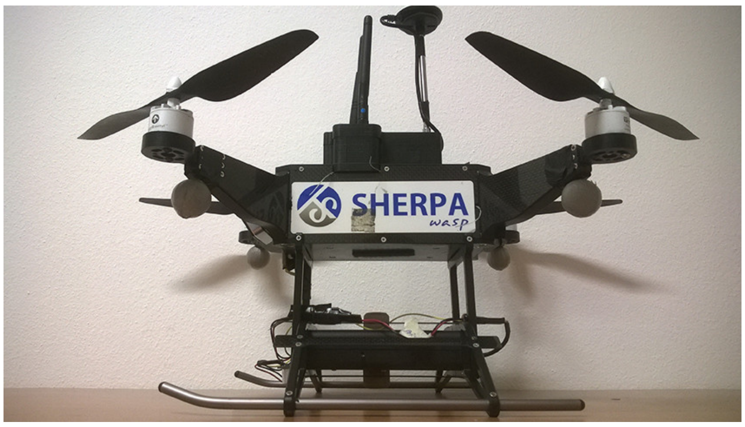

Figure 1.1: Sherpa drone designed to support search and rescue activities in a real-world hostile environment like the alpine scenario. [3]

project [1] [2] which is a UE funded project under the supervision of the Alma Mater Studiorum of Bologna. SHERPA's goal is to develop a mixed ground and aerial robotic platform to support search and rescue activities in a real-world hostile environment like the alpine scenario. In particular, the strategy adopted to require a fleet of small UAVs to fly around looking for people to be rescued. The small dimension of the areal vehicles allows them to be agile and fast in the search, cutting down the time needed to identify the position of stragglers. On the other hand, the UAVs' battery life is limited, therefore they need a ground robot companion carrying spare batteries. Moreover, the rover is equipped with a camera and a robotic arm able to replace the drone's battery when needed. To do so, the drones need to be able to land in a specific position with respect to the rover.

Another good example is the study on ambulance drones [4] from Delft University, where they tested the use of a drone solution to promptly deliver defibrillator in case of an emergency. This is can potentially save a lot of lives since it could be easily called using an app from anyone, referring to the GPS coordinates of the caller could reach the specific position easily and much faster than a human-operated ambulance van. When the ambulance drone gets close

[1] Video link: https://www.youtube.com/watch?v=dKThQJ3VAl8

[2] Project link: https://www.unibo.it/en/research/projects-and-initiatives/Unibo-Projects-under-7th-Framework-Programme/cooperation-1/information-and-communication-technology-ict-1/sherpa

[3] Article link: https://www.tomshw.it/altro/sherpa-i-droni-al-servizio-del-soccorso-alpino/

[4] Websitelink link: https://www.tudelft.nl/en/ide/research/research-labs/applied-labs/ambulance-drone/

enough to the patient, it could identify them and land close enough such that it gets easy for the people nearby to use the defibrillator.

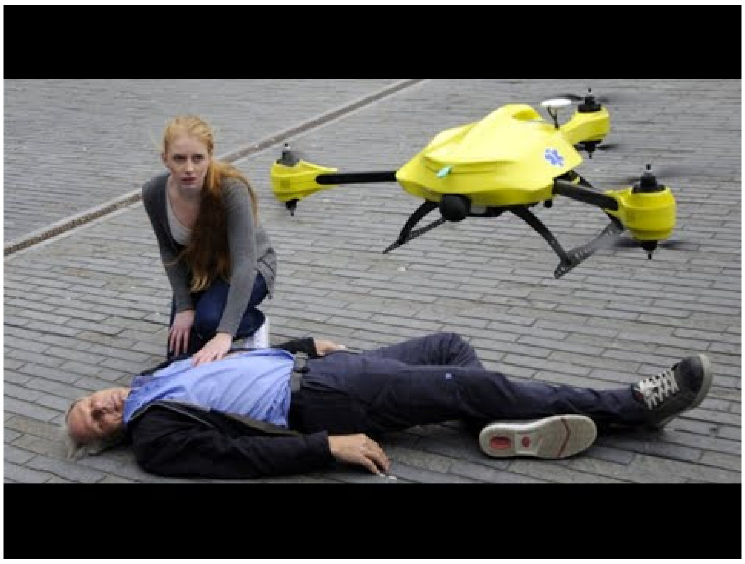

Figure 1.2: Ambulance drone developed in TU Delft carrying and Automated Defibrillator (AED) to provide fast first aid assistance to people suffer from a cardiac arrest. [4]

Agricultural applications for UAV are also largely explored recently where researchers are trying to bridge the gap between the current and desired capabilities of agricultural robots. In this kind of applications, the motion of the UAV needs to follow the vineyards, both for inspection and for intervention porpoises. For example, the drone in Fig. 1.3 has been developed at ETH for the Flourish Project [5]. The drone is able to selectively applying pesticides to the plants that need it.

1.1 Motivation

In this work, we will investigate the usage of vision sensors to close the control loop in UAVs' applications. Vision is essential for humans and living beings to survive, as well as the ability to perceive and understand the environment is crucial for an autonomous robot to accomplish any given task. Robot vision

[5]Project link: http://flourish-project.eu/

[6]Wesite link: https://worldfoodsystem.ethz.ch/news/wfsc-media/2018/08/robots-or-drones-helping-precision-pesticide-application.html

Figure 1.3: Agricultural drone developed in ETH in order to help in precise pesticide applications. [6]

refers to the capability of a robot to visually perceive the environment and interact with it, [20]. Robot vision typically is applied to perform tasks such as navigation in a known environment or exploration in an unknown one, avoiding obstacles, looking for a given target, interaction with a human operator or another robot.

In particular, we are going to focus on visual servoing that is a well-known control strategy that uses visual information as feedback to control the motion of a robot with respect to a given reference without the need of knowing its global position. Despite the many advantages that this method provides, applying it for the control of drones is not trivial. In particular the problem of the system being under-actuated needs to be faced: visual servoing algorithms output a required velocity to be tracked by the vehicle for each of the six degrees of freedom, but the UAV is actuated just along four of them. For this reason, it is essential that the tracking takes into account the actual dynamics of the system and its physical limitations that are not taken into account by the visual servoing. We investigate the use of linear Model Predictive Control (MPC) to accomplish this task. MPC is an effective advanced control method that uses the dynamic model of the system to control it while satisfying a set of constraints.

1.2 Literature review

As previously mentioned, visual information is crucial for motion planning and control of robots in an unknown environment. As a matter of fact vision allows robotic systems to obtain geometrical and qualitative information of the surrounding space essential to accomplish any given task, [35]. In this paragraph, we are going to discuss previous works done in this field that we are going to consider as the background of this thesis.

The first experiments that use visual information to successfully correct the robot position came from 1973, [33], and from then visual feedbacks has been extensively used in robot navigation, obstacle avoidance and manipulation of objects. We refer to *Look-and-move* algorithms when the visual information is used in open loop and we call *vision-based control* or *visual servoing* in the case of closed-loop. *Visual servoing* is based on techniques from different subjects like image processing, computer vision and control theory. It consists of two distinct processes: feature tracking and control, in this thesis we will focus on the control part.

There are two fundamental types of visual servoing: PBVS - *Position-Based Visual Servoing* and IBVS - *Image-Based Visual Servoing*. Despite the potential problems in convergence and stability, [5], IBVS overcome many of the calibration and robustness problems of PBVS, [12]. Moreover, depending on the position of the camera with respect to the robot, we can distinguish between *eye-in-hand* and *eye-to-hand* configuration. As the name suggests, in the *eye-in-hand* configuration the camera is allocated on the robot itself, in opposition in the *eye-to-hand* configuration the camera is not positioned on the robot itself, but it is in the surrounding environment pointing to the robot. Depending on the application, each configuration may have advantages or disadvantage. As a matter of fact, with the *eye-to-hand* configuration one could have a more general view on the robot and its surroundings, but, while accomplishing the task, the robot itself could occlude the view. Moreover and *eye-to-hand* configuration require the robot to operate in an equipped environment. This can find a good application for industrial robots, such as automatic manipulators or smart carrying systems. Furthermore, the *eye-in-hand* configuration is better applicable to mobile robots in a partially or unknown environment.

Since 1992, autonomous agents use vision systems and the are inserted in the automatic control loop as dedicated sensors, [8]. IBVS was first used in the control of a class of under-actuated rigid body in 2002, [14]. For the first time, the full dynamic system with all degrees of freedom was considered. They exploited the passivity-like properties of the system to obtain a Lyapunov control algorithm using robust backstepping techniques. A novel control law based on computer vision for quasi-stationary flights above a planar target is presented two years later, [28]. The focus of this work is on the dynamics of an Unmanned Aerial Vehicle (UAV) for monitoring of structures and maintenance of bridges. Furthermore, in 2010, an image-based visual servo control for an unmanned aerial vehicle(UAV) capable of stationary or quasi-stationary flight with a camera mounted on board was proposed, [11]. The authors consider as target-features a set of stationary and disjoint points on a plane. Before, there were few integrated IBVS control designs for fully dynamic under-actuated system models, for example, [27, 14, 15], where, as in previous works, they used a nonlinear controller based on backstepping techniques. Finally, more recent works on the topic concern attitude estimation and stabilization entirely based on image feedback from a pan and tilt camera and biased rate gyros, [3].

In this work we are using MPC as a lower-level controller. MPC is an advanced control strategy that aims to find an optimal control signal for a process while satisfying a set of constraints. MPC uses the dynamic model of the system to predict its future behaviour and uses that to determine the input signal u that minimize the cost function. Initially, it was used in chemical applications, to control the transients of dynamic systems with hundreds of inputs and outputs, subject to constraints, [29].

Historically modern control came from Kalman in the early 1960s who worked on determine when a linear control system could be considered optimal, [16, 17]. LQR-*linear quadratic regulator* minimizes an unconstrained quadratic objective function of states and inputs, but, even with powerful stabilizing properties because of the infinite horizon, it had a low impact on the industrial control applications because of the lack of constraints in its formulation and the nonlinearities of the real system, [32]. Linear MPC is a special case the unconstrained infinite horizon LQR, where the horizon $N = \inf$ and the stage cost is given by the quadratic expression.

In the last decades, a lot of work has been done on MPC and it has been applied in a lot of new fields. Quoting Camacho and Bordons in the preface of their book *Model Predictive Control*, [4]: "The reason of this success can be attributed to the fact that MPC is, perhaps, the most general way of posing the process control problem in the time domain". In particular, here we are going to focus on its application to control UAVs.

In 2007 the first model predictive controller to control UAVs was designed. The purpose of the work was to see if a good tracking controller was achievable when dealing with a highly nonlinear aircraft system. For the simulations, a MATLAB Simulink model of the UAV 'Ariel' is used, [30]. In the same year an autonomous vision-based Landing and Terrain Mapping Using an MPC-controlled Unmanned Rotorcraft was proposed, [36]. Two years late, in 2009 a model predictive control strategy was presented for the visual servoing of a robot manipulator with eye-in-hand configuration, [21].

In 2010 predictive control was used with the image-based visual servoing (IBVS) to deal with constraints such as robot workspace limitations, visibility constraints and actuators limitations. These constraints are expressed into the MPC formulation as state, output, and input constraints, respectively. Based on the predictive-control strategy, the IBVS task is written into a nonlinear optimization problem in the image plane, where the constraints can be easily and explicitly taken into account [1].

More recently, in 2014 a real-time solution to onboard trajectory tracking control of quadrotors was presented. The proposed approach combines the standard hierarchical control paradigm that separates the control into low-level motor control, mid-level attitude dynamics control, and a high-level trajectory tracking with a model predictive control strategy, [2]. In 2015 an explicit solution of model predictive control (MPC) for trajectory tracking of quadrotors has been proposed. Here they represented the reference trajectory, system outputs and inputs using Bezier curves by using the differential flatness property of the quadrotor. Thus, the formulated optimisation problem can be parameterised and converted into standard quadratic programming, that then can be further formulated into multiparametric quadratic programming which is then solved off-line as a piecewise affine function, [22].

Finally, in 2017 a classical Linear Model Predictive Controller (LMPC) is presented and compared against a more advanced Nonlinear Model Predictive Controller (NMPC) that considers the full system model, [18].

More recent works on Model Predictive Visual Servoing has been done for fully actuated underwater vehicles. In 2019 Gao, Zhang, Wu, Zhao, Wang and Yan presented a sliding-mode observer-based model predictive control (SMO-MPC) strategy for image-based visual servoing (IBVS) of fully-actuated underwater vehicles subject to the field of view and actuator constraints and model uncertainties. With the consideration of system uncertainties, including external disturbances and unknown dynamic parameters, a sliding-mode observer is designed to estimate the modelling mismatch, which is feedforward to the dynamic model in MPC, [10].

1.3 Thesis Outline

We are going to start in Chapter 2 by presenting the system we are going to work with and illustrating how we build its dynamical model. In Chapter 3 we will continue by presenting some theory on visual servo control to give some context to the work, followed by the results of some preliminary simulations performed on MATLAB. Then, in Chapter 4 we are going to briefly introduce MPC and present how we are using it as a low-level controller to address the issue that the system is in fact under-actuated. We are going to explain the optimization problem we aim to solve and the model of the system we are using. Furthermore in Chapter 5 we are going to describe the whole software architecture we implemented. Results from simulations and experiments will be discussed. Finally in Chapter 6 we are going to go thought some conclusions and future works.

Chapter 2

System Dynamics

2.1 Mathematical Notation

In this section, we are going to introduce the mathematical notation that is used in this work.

- x: lower case letter indicates a scalar
- **x**: lower case bold letter indicates a vector
- **X**: upper case bold letter indicates a matrix

Moreover we will use the following abbreviations:

- $c(\theta)$ instead of $\cos\theta$
- $s(\theta)$ instead of $\sin\theta$

2.2 Coordinate Systems

To describe a quadrotor's dynamical model we have to first introduce the reference frames we are going to use in the description. In particular, we are going to introduce two coordinates systems: *Inertial Reference Frame* $\{F_I\}$ and *Body Reference Frame* $\{F_B\}$. The inertial frame is considered fixed with respect to the world, while the body frame is fixed with the robot's barycenter therefore it moves with respect to the inertial frame with the UAV. In Fig. 2.1 is shown a representation of the different coordinate systems.

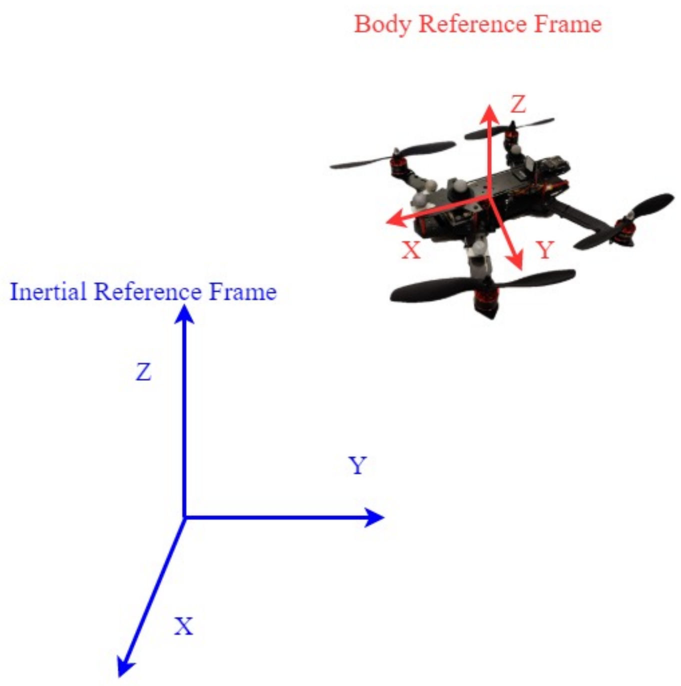

Figure 2.1: Quadrotor model connected with its body frame represented in red and the inertial frame in blue color. Body frame is fixed with respect to the quadrotor, so it moves with it while the inertial frame is fixed with respect to the external world.

2.3 Euler Angles

To describe the UAV's orientation with respect to the Inertial Coordinate System, we need a way for describing 3D rotations. In this work, we will parametrize 3D rotations with the use of Euler Angles. Euler Angles have been named after Leonhard Euler, who was the first one using them in the 18th century. His theory is based on the principle that each 3D rotation can be unequivocally defined using three angles. We define the Euler angles as follows:

- ϕ: define rotation around the x axis
- θ: define rotation around the y axis
- ψ: define rotation around the z axis

The resulting elementary rotational matrices are:

$$\mathbf{R}_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & c(\phi) & -s(\phi) \\ 0 & s(\phi) & c(\phi) \end{bmatrix}, \tag{2.1}$$

$$\mathbf{R}_y(\theta) = \begin{bmatrix} c(\theta) & 0 & s(\theta) \\ 0 & 1 & 0 \\ -s(\theta) & 0 & c(\theta) \end{bmatrix}, \tag{2.2}$$

$$\mathbf{R}_z(\psi) = \begin{bmatrix} c(\psi) & -s(\psi) & 0 \\ s(\psi) & c(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{2.3}$$

The final 3D rotation is defined as the combination of three elementary rotations around axis x, y and z, respectively $\mathbf{R}_x$, $\mathbf{R}_y$ and $\mathbf{R}_z$. There are different configurations, we are using ZYX notation, [35]. Therefore, a general rotation is defined as follows:

$$\mathbf{R}_{zyx}(\phi, \theta, \psi) = \mathbf{R}_z(\psi)\mathbf{R}_y(\theta)\mathbf{R}_x(\phi), \tag{2.4}$$

$$\mathbf{R}_{zyx}(\phi, \theta, \psi) = \begin{bmatrix} c(\theta)c(\psi) & s(\phi)s(\theta)c(\psi) - c(\theta)s(\psi) & c(\phi)s(\theta)c(\psi) + s(\theta)s(\psi) \\ c(\theta)s(\psi) & s(\phi)s(\theta)s(\psi) + c(\theta)c(\psi) & c(\phi)s(\theta)s(\psi) - s(\theta)c(\psi) \\ -s(\theta) & s(\phi)c(\theta) & c(\phi)c(\theta) \end{bmatrix}. \tag{2.5}$$

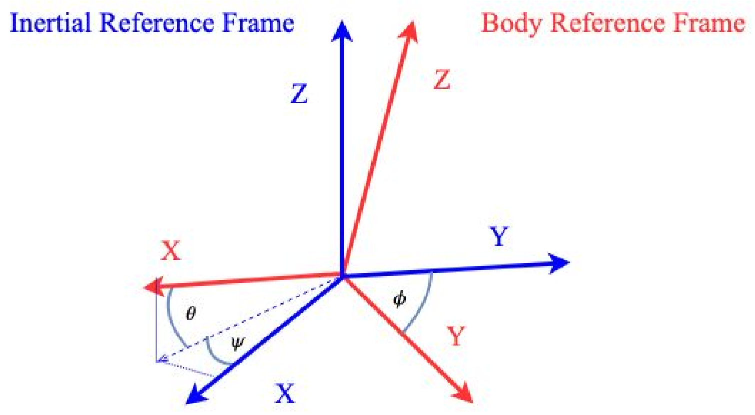

Figure 2.2: Euler Angles representation. Inertial frame represented in blue while body frame represented in red. ϕ define rotation around the x axis, θ around the y axis and ψ the z axis.

2.4 UAVs

There are several kinds of UAV depending on the dimension and number of actuators. In particular, we are going to focus on quadrotors that are characterized by four actuators and propellers. Quadrotors have four individual rotors connected to the rigid cross airframe, as shown in Fig. 2.3. Note that the system is underactuated because no actuator is directly providing motion on the XY plane. Therefore that translation needs to be controlled with the available four degrees of freedom, [24].

The UAV is therefore actuated for roll, pitch, yaw and thrust. In Fig. 2.4, each degree of freedom is illustrated with respect to the body frame of the vehicle. In particular, thrust represents a motion along the z axis, roll represents a rotation around the x axis, pitch represents a rotation around the y axis and yaw represents a rotation around the z axis. All axes here are considered in the body frame.

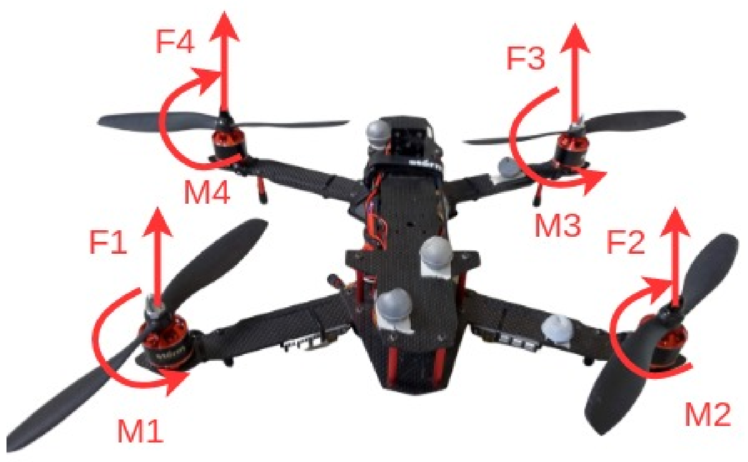

Figure 2.3: Quadrotor model with schematics of its actuation system. A quadrotor is a UAV with four actuators and for propellers.

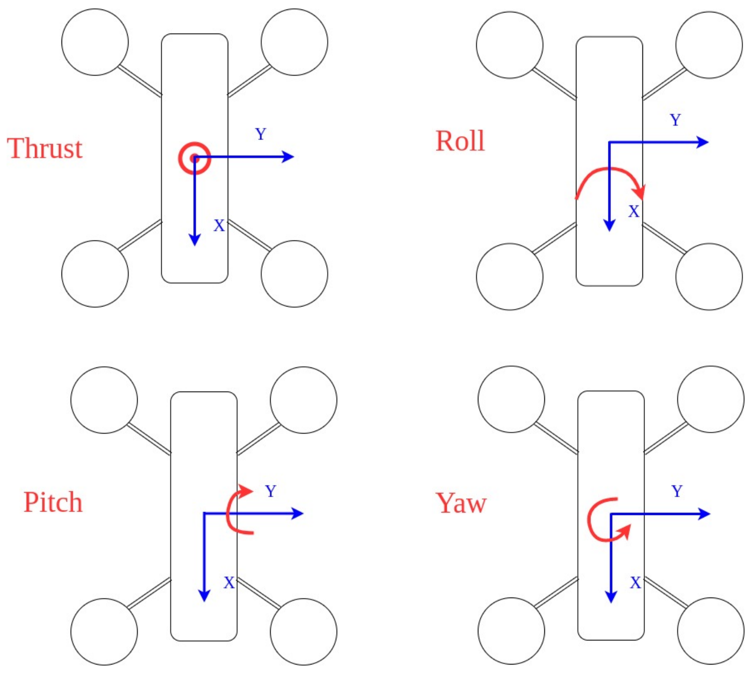

Figure 2.4: Schematics of the four actuated degrees of freedom of a quadrotor. Thrust represent the translation along the z axis in the body frame, roll refers to the rotation around the x axis, pitch around the y axis and yaw around z.

2.5 Nonlinear Dynamical Model of the System

As a first approximation, we can consider the nonlinear model of a quadrotor as described by [25]

$$\dot{\mathbf{p}} = \mathbf{v}, \tag{2.6}$$

$$m\dot{\mathbf{v}} = mg\vec{\mathbf{z}} + \mathbf{R}f, \tag{2.7}$$

$$\dot{\mathbf{R}} = \mathbf{R}\Omega_\times, \tag{2.8}$$

$$\mathbf{J}\dot{\Omega} = -\Omega \times \mathbf{J}\Omega + \tau, \tag{2.9}$$

where $\mathbf{p} \in \mathbb{R}^3$ and $\mathbf{v} \in \mathbb{R}^3$ represent the UAV position and velocity in the inertial frame, m its mass, g is the gravity contribution, the vector $\vec{\mathbf{z}}$ gives the direction of the z axis of the inertial frame, R is the rotational matrix, f and τ are forces and moments expressed in body frame, $\Omega \in \mathbb{R}^3$ denotes the angular velocities in the body frame, $\Omega\times$ denotes the skew-symmetric matrix and $\mathbf{J} \in \mathbb{R}^{3\times3}$ is the inertial matrix expressed in the body frame.

As previously mentioned the quadrotor has four actuators: one for each propeller. By controlling the power in each one of them it is possible to directly control the vehicle into four different types of motions: vertical motion and rotations around x, y and z-axis. Namely, it is possible to direct actuate thrust on the vertical direction and the three torques. It is clear now that the system is under-actuates because horizontal motion along x and y can be achieved with combinations of the other allowed motions. This is crucial for our implementation since the higher-level visual servo control does not take into account the under-actuation of the system. So the nonlinear state-space model we are considering now is composed of twelve states and four inputs signal to control the system. The states are defined as follows: linear positions along x-y-z, linear velocities along x-y-z, angular positions roll-pitch-yaw and angular velocities along roll-pitch-yaw

$$\mathbf{x} = \begin{bmatrix} p_x & p_y & p_z & v_x & v_y & v_z & \phi & \theta & \psi & \omega_\phi & \omega_\theta & \omega_\psi \end{bmatrix}^T. \tag{2.10}$$

While the inputs are: $\tau_\phi, \tau_\theta, \tau_\psi$ respectively moment with respect x, y, z axis and f_{thrust} is the force applied along z:

$$\mathbf{u} = \begin{bmatrix} \tau_\phi & \tau_\theta & \tau_\psi & f_{thrust} \end{bmatrix}^T. \tag{2.11}$$

In the next section, we are going to describe how we are linearizing and than discretizing the model of the system in equations (2.6),(2.7),(2.8) and (2.9).

2.6 Linearization and Discretization

Solving non-linear optimization problem is very computationally expensive. Therefore we linearize the dynamic of the system around the hovering equilibrium point. Starting from the nonlinear model in equations (2.6),(2.7),(2.8) and (2.9) we linearize around the hovering equilibrium configuration:

$$\mathbf{u}_0 = \begin{bmatrix} 0 & 0 & 0 & mg \end{bmatrix}^T, \tag{2.12}$$

$$\mathbf{x}_0 = \begin{bmatrix} p_{x0} & p_{y0} & p_{z0} & v_{x0} & v_{y0} & v_{z0} & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}^T. \tag{2.13}$$

The final linearized state space model obtained is $\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$, Where the obtained matrices $\mathbf{A}$ and $\mathbf{B}$ are:

$$\mathbf{A} = \begin{bmatrix} \mathbf{I}^{3\times3} & \mathbf{0}^{3\times3} & \mathbf{0}^{3\times3} \\ \mathbf{0}^{3\times3} & [\mathbf{g}]_\times & \mathbf{0}^{3\times3} \\ \mathbf{0}^{3\times3} & \mathbf{0}^{3\times3} & \mathbf{I}^{3\times3} \\ \mathbf{0}^{3\times3} & \mathbf{0}^{3\times3} & \mathbf{0}^{3\times3} \end{bmatrix}, [\mathbf{g}]_\times = \begin{bmatrix} 0 & -g & 0 \\ g & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}, \tag{2.14}$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{0}^{3\times3} & \mathbf{0}^{3\times1} \\ \mathbf{0}^{3\times3} & \frac{1}{m}\vec{\mathbf{z}} \\ \mathbf{0}^{3\times3} & \mathbf{0}^{3\times1} \\ \mathbf{J}^{-1} & \mathbf{0}^{3\times1} \end{bmatrix}, \tag{2.15}$$

where the vector $\vec{\mathbf{z}}$ gives the direction of the z axis of the inertial frame, $\mathbf{J}^{-1}$ is the inverse of the inertial matrix expressed in the body frame and $\mathbf{0}^{3\times3}$ is a 3 by 3 matrix with all zeros. Since the goal we have is to control the UAV's velocities around a given setpoint, the three states linked to the linear positions **p** are not needed for our purpose. Therefore, we remove them from the model used by the MPC. The final state-space model we are considering is composed of remaining nine states: linear velocities along $x - y - z$, angular positions roll-pitch-yaw and angular velocities along roll-pitch-yaw:

$$\mathbf{A} = \begin{bmatrix} \mathbf{0}^{3\times3} & [\mathbf{g}]_\times & \mathbf{0}^{3\times3} \\ \mathbf{0}^{3\times3} & \mathbf{0}^{3\times3} & \mathbf{I}^{3\times3} \\ \mathbf{0}^{3\times3} & \mathbf{0}^{3\times3} & \mathbf{0}^{3\times3} \end{bmatrix}, [\mathbf{g}]_\times = \begin{bmatrix} 0 & -g & 0 \\ g & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{2.16}$$

$$\mathbf{B} = \begin{bmatrix} \mathbf{0}^{3\times3} & \frac{1}{m}\vec{\mathbf{z}} \\ \mathbf{0}^{3\times3} & \mathbf{0}^{3\times1} \\ \mathbf{J}^{-1} & \mathbf{0}^{3\times1} \end{bmatrix}. \tag{2.17}$$

Moreover, we discretize the model as zero-order-hold and we consider a sampling time of T_s. The discretized state space model is $\mathbf{x}(k+1) = \mathbf{A}_d\mathbf{x}(k)+$

$\mathbf{B}_d\mathbf{u}(k)$, where the matrices $\mathbf{A}_d$ and $\mathbf{B}_d$ are defined as follows:

$$\mathbf{A}_d = e^{\mathbf{A}T_s} = \mathcal{L}^{-1}\{(t\mathbf{I} - \mathbf{A})^{-1}\}_{t=T_s}, \tag{2.18}$$

$$\mathbf{B}_d = \left(\int_{t=0}^{T_s} e^{\mathbf{A}t} dt\right)\mathbf{B}. \tag{2.19}$$

$\mathcal{L}$ represents the inverse Laplace transform and $e^{\mathbf{A}T_s}$ is the matrix exponential of the system.

Chapter 3

Vision-based control

Visual information is crucial for motion planning and control of robots in an unknown and unstructured environment. As a matter of fact, vision allows robotic systems to obtain geometrical and qualitative information on the surrounding environment, [35]. There are many vision-based control algorithms, in particular, the first one comes from 1973, [34]. This control strategy has been introduced by Shrai and Inoue in order to solve an assembling problem. In general we refer to *Look-and-move* algorithms when the visual information is used in open loop and we call *vision-based control* or *visual servoing* if vision sensors' information, such as cameras, is used to close the loop. *Visual servoing* is based on techniques from different subjects like image processing, computer vision and control theory. It consists of two distinct processes: feature tracking and control, in this thesis we will mainly focus on the control part, an example is shown in Figure 3.1.

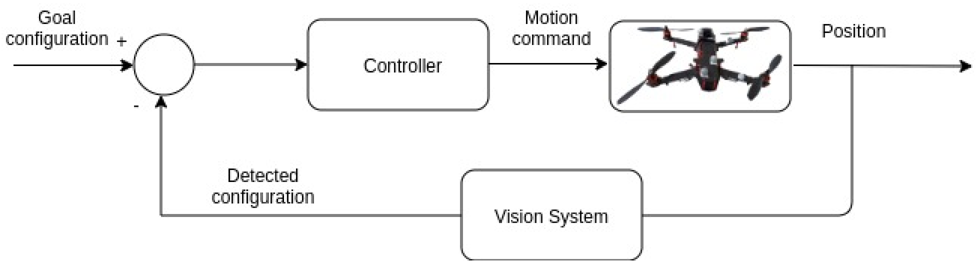

Figure 3.1: Visual control scheme. The motion command to the system (in this case a quadrotor) is computed by the controller from the error in the features. In particular, it consists of the difference between the goal configuration and the one that has been detected by the vision sensor.

3.1 Visual servoing

Visual serving is a control strategy that uses computer vision data from a camera sensor to control the motion of a robot. The camera sensor can be placed on the end effector of the robot or can be fixed in a static position in the workspace. These two main configurations are respectively called: *eye-in-hand* and *eye-to-hand*, [6].

The visual servoing controller is designed to minimize an error between features position. That error is defined as follows:

$$\mathbf{e}(t) = \mathbf{s}(\mathbf{m}(t), \mathbf{a}) - \mathbf{s}^*, \tag{3.1}$$

where $\mathbf{m}(t)$ is a set of image measurements, $\mathbf{s}(\mathbf{m}(t), \mathbf{a})$ is a vector of k visual features computed using the image measurements, $\mathbf{a}$ is the set of camera intrinsic parameters and $\mathbf{s}^*$ represents the desired values of the features, [6]. We are considering $\mathbf{s}^*$ is constant, since in our setup the features are stationary with respect of the workspace and there is a fixed goal position. Therefore the changes in $\mathbf{s}$ depends only on the camera motion.

Depending on how $\mathbf{s}$ is designed there are two different visual servoing schemes: image-based visual servo control (IBVS) and position-based visual servo control (PBVS). In IBVS, $\mathbf{s}$ consists of a set of features that are immediately available from the image data, on the other hand, in PBVS $\mathbf{s}$ consists of a set of 3-D parameters that must be estimated from image measurements. This means that IBVS performs the visual servoing in the *image space*, while PBVS in the *operational space*. Some other advanced visual servoing control schemes exist, such as: *hybrid visual servoing* and *partitioned visual servoing*, [7]. In *hybrid visual servoing*, the control error is defined in the operational space for some components and in the image space for others for this reason this approach combines the advantages of PBVS and IBVS, [35]. By choosing adequate visual features, the hybrid control scheme is able to decouple the translational from the rotational motions. On the other hand, in *partitioned visual servoing*, the goal is to create a one-to-one relationship between feature and degree of freedom. This means having six features each one related to one of the degrees of freedom of the system.

After having selected $\mathbf{s}$, the control law should be designed. We start by expressing the velocity of the camera as follows:

$$\mathbf{v}_c = (\boldsymbol{v}_c, \boldsymbol{\omega}_c), \tag{3.2}$$

where v_c is the instantaneous linear velocity of the origin of the camera frame and $\boldsymbol{\omega}_c$ is the instantaneous angular velocity of the camera frame. Then we

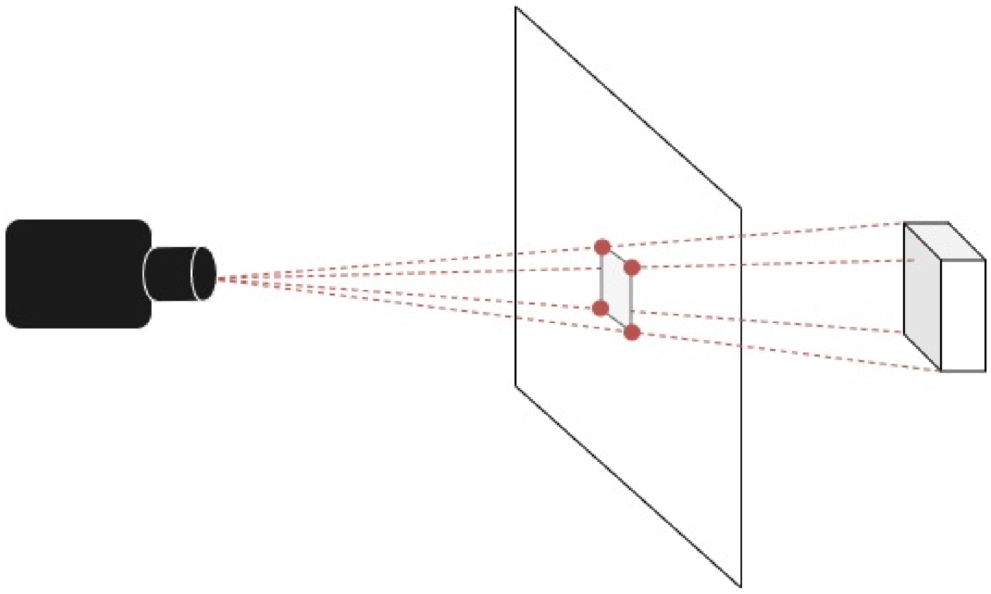

Figure 3.2: Projection of features on image plane. In this case we consider as features the four corners of the base of a parallelepiped.

have that

$$\dot{\mathbf{s}} = \mathbf{L_s}\mathbf{v}_c, \tag{3.3}$$

where $\mathbf{L_s} \in \mathbb{R}^{k\times 6}$ is the *interaction matrix*, or *feature Jacobian*, related to s, where k is the number of visual features. Considering both equations 3.1 and 3.3, we obtain that

$$\dot{\mathbf{e}} = \mathbf{L_e}\mathbf{v}_c, \tag{3.4}$$

where $\mathbf{L_e} = \mathbf{L_s}$. Moreover

$$\mathbf{v}_c = -\lambda\mathbf{L}_e^{+}\mathbf{e}, \tag{3.5}$$

where $\mathbf{L}_e^{+} \in \mathbb{R}^{6\times k}$ is chosen as the Moore-Penrose pseudo-inverse of $\mathbf{L_s}$, that is $\mathbf{L}_e^{+} = (\mathbf{L}_e^T\mathbf{L}_e)^{-1}\mathbf{L}_e^T$.

Considering that in real visual servoing applications, it is impossible to know the exact values for $\mathbf{L}_e$ and $\mathbf{L}_e^{+}$, and approximation or estimation of them should be realized. Therefore, by adding the correct notation the control law becomes:

$$\mathbf{v}_c = -\lambda\widehat{\mathbf{L}_e^{+}}\mathbf{e}. \tag{3.6}$$

3.2 The Interaction Matrix

Considering a 3-D point $\mathbf{P} = (X, Y, Z)$ in the camera frame, which projects in the image as a 2-D point with coordinates $\mathbf{p} = (x, y)$.

We obtain that:

$$\begin{cases} x = X/Z = (u - c_u)/f\alpha \\ y = Y/Z = (v - c_v)/f \end{cases} \tag{3.7}$$

where $\mathbf{m} = (u, v)$ gives the coordinate of the image point expressed in pixels and $\mathbf{a} = (c_u, c_v, f, \alpha)$ is the set of camera intrinsic parameters (c_u and c_v are the coordinate of the principal point, f is the focal length and α is the ratio of the pixel dimension).

By combining the derivative of equation 3.7 and

$$\begin{cases} \dot{X} = -v_x - \omega_y Z + \omega_z Y \\ \dot{Y} = -v_y - \omega_z X + \omega_x Z \\ \dot{Z} = -v_z - \omega_x Y + \omega_y X \end{cases} \tag{3.8}$$

we obtain that

$$\begin{cases} \dot{x} = -v_x/Z + x v_z/Z + xy\omega_x - (1 + x^2)\omega_y + y\omega_z \\ \dot{y} = -v_y/Z + y v_z/Z + (1 + y^2)\omega_x - xy\omega_y - x\omega_z \end{cases} \tag{3.9}$$

from which it is possible to extract the interaction matrix easily

$$\mathbf{L}_x = \begin{bmatrix} -1/Z & 0 & x/Z & xy & -(1+x^2) & y \\ 0 & -1/Z & y/Z & 1+y^2 & -xy & -x \end{bmatrix}, \tag{3.10}$$

as shown in [6]. In order to be able to control a 6 DOF robot, at least 3 points are necessary, therefore we got:

$$\mathbf{L}_x = \begin{bmatrix} \mathbf{L}_{x_1} \\ \mathbf{L}_{x_2} \\ \mathbf{L}_{x_3} \end{bmatrix}. \tag{3.11}$$

Note that with three points there exists multiple local minima poses, therefore the algorithm will give as output one of them while $\mathbf{e}$ converges, [6]. We want to identify a unique goal position for the camera, therefore we will have four non-collinear features. Therefore $\mathbf{s}$ is build as follows:

$$\mathbf{s} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ Z_1 & Z_2 & Z_3 & Z_4 \end{bmatrix}. \tag{3.12}$$

3.3 Stability Analysis

If the number of features is equal to the number of camera degrees of freedom, and if the features are chosen and the control scheme designed so that $\mathbf{L}_e$ and $\widehat{\mathbf{L}_e^+}$ are full rank, then the condition $\mathbf{L}_e\widehat{\mathbf{L}_e^+} > 0$ is ensured if the approximations involved in $\widehat{\mathbf{L}_e^+}$ are not too coarse, [6].

Note that for IBVS only local asymptotic stability around a small neighbourhood of the desired position is ensured, even with perfect knowledge of the interaction matrix, [6]. It is well-known that if s is composed of three collinear points, then the Interaction matrix will be singular. Even if the three points are not collinear, problems may occur due to the not unicity of the goal camera pose. This issue is solved by simply using four points as visual features.

Chapter 4

Model Predictive Control

Model Predictive Control (MPC) is a well known optimization-based control technique. The intuition behind this control strategy is to use a dynamic model for the system, to predict its future behaviour as a function of an input signal. In this way, it is able to identify the best possible outcome and which action performed at the current time instant will lead to it, [31]. A generic dynamic model can be expressed in state space form through differential equations,

$$\frac{d\mathbf{x}}{dt} = f(\mathbf{x}, \mathbf{u}, t) \tag{4.1}$$

$$\mathbf{y} = h(\mathbf{x}, \mathbf{u}, t) \tag{4.2}$$

$$\mathbf{x}(t_0) = \mathbf{x}_0 \tag{4.3}$$

where $\mathbf{x} \in \mathbb{R}^n$ is called *state vector*, $\mathbf{u} \in \mathbb{R}^m$ is the *input*, $\mathbf{y} \in \mathbb{R}^p$ is the *output vector* and $t \in \mathbb{R}$ is the *time*. Moreover, $t = t_0$ represents the initial time instant and $\mathbf{x}_0$ is the initial configuration of the system.

A model is an abstract representation of the real system. In particular, we are going to work with state-space representation, where the relationship between input, output and state variables is described through differential equations. The more generic state-space model is nonlinear, as the one used in equations (4.1), (4.2) and (4.3). To simplify the problem, one can approximate the system model with a linear one. Linear models are divided into linear time-variant and linear time-invariant models depending on if the evolution of the model itself evolves over time.

A linear time-variant model is described as follows:

$$\frac{d\mathbf{x}(t)}{dt} = \mathbf{A}(t)\mathbf{x}(t) + \mathbf{B}(t)\mathbf{u}(t) \tag{4.4}$$

$$\mathbf{y}(t) = \mathbf{C}(t)\mathbf{x}(t) + \mathbf{D}(t)\mathbf{u}(t) \tag{4.5}$$

$$\mathbf{x}(t_0) = \mathbf{x}_0 \tag{4.6}$$

where $\mathbf{A}(t) \in \mathbb{R}^{n \times n}$ is called *state transition matrix*, $\mathbf{B}(t) \in \mathbb{R}^{n \times m}$ is the *input matrix*, $\mathbf{C}(t) \in \mathbb{R}^{p \times n}$ is the *output matrix* and $\mathbf{D}(t) \in \mathbb{R}^{p \times m}$ represent the coupling between input signal $\mathbf{u}$ and output $\mathbf{y}$. If the matrices $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$ and $\mathbf{D}$ are time-invariant, the new model looks as follows:

$$\frac{d\mathbf{x}}{dt} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u} \tag{4.7}$$

$$\mathbf{y} = \mathbf{C}\mathbf{x} + \mathbf{D}\mathbf{u} \tag{4.8}$$

$$\mathbf{x}(t_0) = \mathbf{x}_0. \tag{4.9}$$

MPC is a powerful technique because it allows engineers to impose constraints on inputs and state variables during the computation of the control law. We require states and inputs to satisfy constraints

$$\mathbf{x}(t) \in \mathbf{X} \subseteq \mathbb{R}^{n_x}, \forall t \in [0, T[\cup]T, \infty[, \tag{4.10}$$

$$\mathbf{x}(t) \in \mathbf{X}_f \subseteq \mathbb{R}^{n_x}, t = T, \tag{4.11}$$

$$\mathbf{u}(t) \in \mathbf{U} \subseteq \mathbb{R}^{n_u}, \forall t \in [0, \infty[\tag{4.12}$$

with initial condition for the system given by $\mathbf{x}(t) = \mathbf{x}_0$ and horizon T. Constraints as defined below

$$\mathbf{X} = \{\mathbf{x} \in \mathbb{R}^{n_x} : x_{min} \leq \mathbf{x} \leq x_{max}\}, \tag{4.13}$$

$$\mathbf{U} = \{\mathbf{u} \in \mathbb{R}^{n_u} : u_{min} \leq \mathbf{u} \leq u_{max}\} \tag{4.14}$$

are called *hard constraints* because they cannot be broken. They are usually used for hardware limitations or to prevent safety critical situations to occur. On the other hand, *soft constraints* are constraints that can be broken if necessary. It is possible to *softern* the constraints by adding a slack variable s, as follows

$$\mathbf{X} = \{\mathbf{x} \in \mathbb{R}^{n_x} : x_{min} - s \leq \mathbf{x} \leq x_{max} + s\}. \tag{4.15}$$

The slack variable s needs to be positive definite. Moreover, to make sure that the constraints are broken just when necessary, a slack cost needs to be added to the cost function. It is usually referred to as $\sigma(s)$ and it is positive definite.

MPC has its basis in linear-quadratic optimal control. For this reason we will be considering how to derive an optimal control law for linear systems with a quadratic cost function. Consider T time steps, the input sequence is $\mathbf{u} = (u(0), u(1), ..., u(T-1))$. The constraints discussed previously constitute the main difference between a standard linear quadratic control and model predictive control. We define the cost function to measure the deviation of the trajectory of $\mathbf{x}(k), \mathbf{u}(k)$ from the reference. This deviation is computed as follows

$$V(x(0), \mathbf{u}) = \tag{4.16}$$

$$\sum_{t=0}^{T-1} \Delta\mathbf{x}[t]^T \mathbf{Q} \Delta\mathbf{x}[t] + \mathbf{u}[t]^T \mathbf{R} \mathbf{u}[t] + \Delta\mathbf{x}[T]^T \mathbf{Q}_{final} \Delta\mathbf{x}[T] \tag{4.17}$$

where $\Delta\mathbf{x}[t] = \mathbf{x}[t] - \mathbf{x}_{ref}$, $\Delta\mathbf{x}[T] = \mathbf{x}[T] - \mathbf{x}_{ref}$ and $\mathbf{Q} \geq 0$, $\mathbf{R} > 0$ and $\mathbf{Q}_{final} \geq 0$.

The intuition behind a cost function is to penalize the system while diverging from the reference trajectory and using a lot of energy. Depending on the requirements of the specific problem, one can tune the weights $\mathbf{Q}$ and $\mathbf{R}$ accordingly. $\mathbf{Q}_{final}$ is the terminal cost and it needs to approximate the infinite horizon cost. In linear MPC it can be computed by solving the Riccati equation or, for a more conservative solution, one can use a value higher than the ARE solution.

4.1 Problem Formulation

The system we are considering is a linear time-invariant discrete system that can be described by the generic formulation below

$$\mathbf{x}(k+1) = \mathbf{A}_d \mathbf{k}(t) + \mathbf{B}_d \mathbf{u}(k), \tag{4.18}$$

where $\mathbf{A}_d$ and $\mathbf{B}_d$ are the dynamics and transfer matrices of the linearized and discretized state space model, as described in the section 2.6. Let n_x be the number of states $\mathbf{x}$ of the system and n_u be the number of inputs $\mathbf{u}$. At each time instant t, we optimize on an horizon of T steps in the future. $\mathbf{x}(t+i|t)$ and $\mathbf{u}(t+i|t)$ are the $t+i$ state and input predicted at time t. We denote the set of predicted states and inputs as:

$$\mathbf{x} = \{\mathbf{x}(t+1|t), \mathbf{x}(t+2|t), ..., \mathbf{x}(t+T|t)\}, \tag{4.19}$$

$$\mathbf{u} = \{\mathbf{u}(t|t), \mathbf{u}(t+1|t), ..., \mathbf{u}(t+T-1|t)\}. \tag{4.20}$$

We consider the following cost function:

$$V(x(0), \mathbf{u}) = \tag{4.21}$$

$$\sum_{t=0}^{T-1} \Delta\mathbf{x}[t]^T \mathbf{Q} \Delta\mathbf{x}[t] + \mathbf{u}[t]^T \mathbf{R} \mathbf{u}[t] + \Delta\mathbf{x}[T]^T \mathbf{Q}_{final} \Delta\mathbf{x}[T] \tag{4.22}$$

where $\Delta\mathbf{x}[t] = \mathbf{x}[t] - \mathbf{x}_{ref}$, $\Delta\mathbf{x}[T] = \mathbf{x}[T] - \mathbf{x}_{ref}$ and $\mathbf{Q} \geq 0$, $\mathbf{R} > 0$ and $\mathbf{Q}_{final} \geq 0$.

Finally, we can write the formulation for the optimization problem in the following form:

$$\begin{aligned}
&minimize\ (V(x_0, \mathbf{u})) && \qquad (4.23)\\
&subject\ to\ \mathbf{x}(k+t+1) = \mathbf{A}_d\mathbf{x}(k+t) + \mathbf{B}_d\mathbf{u}(k+t), && \forall k \in [0, T-1]\\
&\qquad \mathbf{x}(k+t) \in \mathbf{X}, && \forall k \in [0, T-1]\\
&\qquad \mathbf{u}(k+t) \in \mathbf{U}, && \forall k \in [0, T-1]\\
&\qquad \mathbf{x}(T+t) \in \mathbf{X}_{final}.
\end{aligned}$$

4.1.1 States and Input Constraints

Model predictive control is very powerful because allow the designer to include constraints on the states and on input. In our specific problem formulation, we are including constraints on $\mathbf{\Omega}_\theta$, $\mathbf{\Omega}_\phi$ and $\mathbf{\Omega}_\psi$ to keep the system close to the equilibrium point we are linearizing around. Moreover, input signals (τ_θ, τ_ϕ, τ_ψ and $thrust$) to the system need to be constrained as well to protect the mechanical components.

4.1.2 Solvers

Model predictive control's problem can be translated to a quadratic programming (QP) problem if the following conditions are true:

- $\mathbf{Q}_{final} \geq 0$,
- $\mathbf{X}$, $\mathbf{U}$ and $\mathbf{X}_{final}$ are polyhedral sets.

A polyhedral set is a set in $\mathbb{R}^n$ is said to be polyhedral if it is the intersection of a finite number of closed half spaces, therefore it is described by linear inequalities. Model predictive control's problem can be also directly solved using an algebraic modelling environment such as YALMIP, [23], with Gurobi solver, [13], or a code generator and solver for convex optimization like CVXGEN, [26].

Chapter 5

Experiments

In this chapter, we are going to describe the software implementation. Moreover, we will describe the different simulation setups used and the experiments performed. Preliminary studies have been conducted in Matlab and in Python, after that Simulations in Gazebo have been performed and the model of the UAV used is a Storm SRD-370. Finally, experiments with a real UAV have been conducted. In particular, we used a Foxtech Hover 1 Quadcopter. We considered a simple experimental set up with a drone that needs to perform a specific motion with respect to reference features placed on the floor. Four features points of different colours have been used. Even in its simplicity, this use case has many practical applications in particular in any situation in which a drone is required to approach or lend in a specific spot without any human intervention.

5.1 Control System Architecture

The control loop implemented is represented in fig. 5.1. Reference signal is given by the goal configuration of the features. Here we used four points as features:

$$\mathbf{s} = \begin{bmatrix} x_1 & x_2 & x_3 & x_4 \\ y_1 & y_2 & y_3 & y_4 \\ Z_1 & Z_2 & Z_3 & Z_4 \end{bmatrix}. \tag{5.1}$$

This choice has been motivated in chapter 3. Then the error between the desired feature's position and the detected feature's position in the image frame is computed. This error signal is fed into the IBVS control block. In this block, the velocity commands are computed as described in chapter 3, equation (3.6).

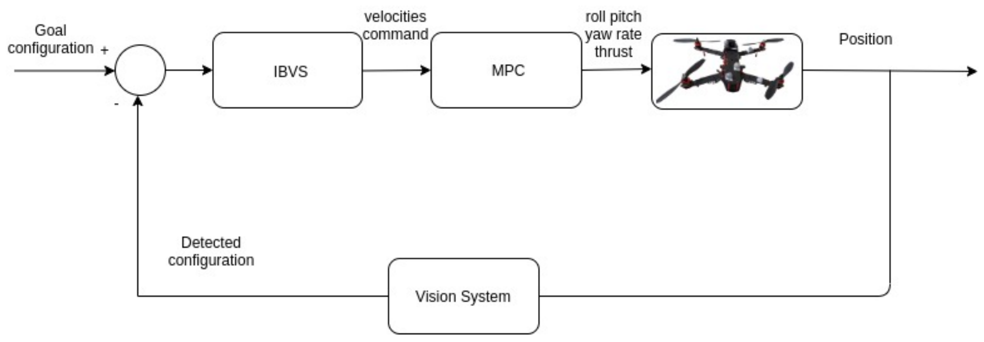

Figure 5.1: Model Predictive Image based control scheme.

These velocity commands are fed into the MPC control block that takes into account the under-actuated dynamics of the real system and it outputs a new command for the UAV in terms of roll, pitch, yaw and thrust. The feedback in the system is provided by the visual system that detects the new position of the features in the image plane.

5.2 Matlab Simulation

A preliminary implementation has been prepared in MATLAB to simulate the visual servo control loop, the blue block in fig. 5.2. For the sake of this preliminary study, we are considering as inputs the starting and goal position of the camera in the world frame. Then, given the fixed position of the four point features in the world frame, we compute the positions of each feature in the image frame by projection. Then, with that, we compute the visual-servo control law such that the error on the features positions and velocities on the image plane goes to zero. When that happens it means that that camera has reached the location in the real world in the given goal position. Here, to simulate the UAV behaviour, we used the linear dynamical model described in chapter 2, equations (2.16) and (2.17). We tested a translation of 0.5 along x, y and z. In fig. 5.3, 5.5 and 5.7 the movement of the features on the image plane is shown. Moreover, in 5.4, 5.6 and 5.8 one can see the camera position and velocity converging towards the desired one and the error going to zero. Respectively the three translations are as follows:

- from position $(0, 0, 2)$ to position $(0.5, 0, 2)$,
- from position $(0, 0, 2)$ to position $(0, 0.5, 2)$,
- from position $(0, 0, 2)$ to position $(0, 0, 2.5)$.

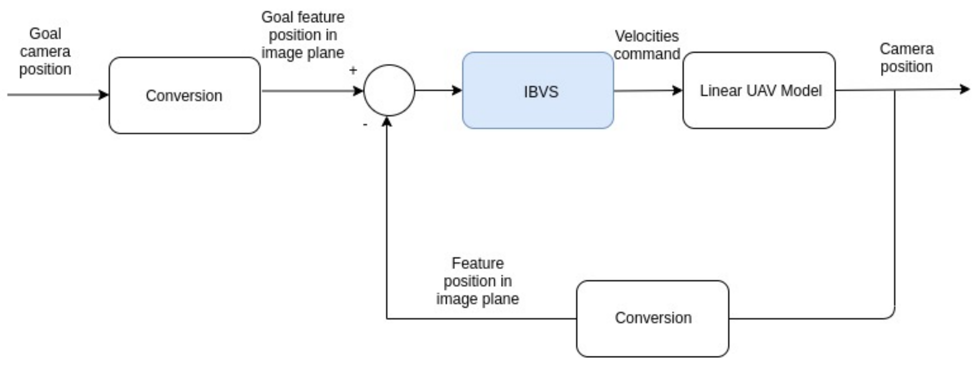

Figure 5.2: Control diagram for the preliminary simulation study on IBVS controller.

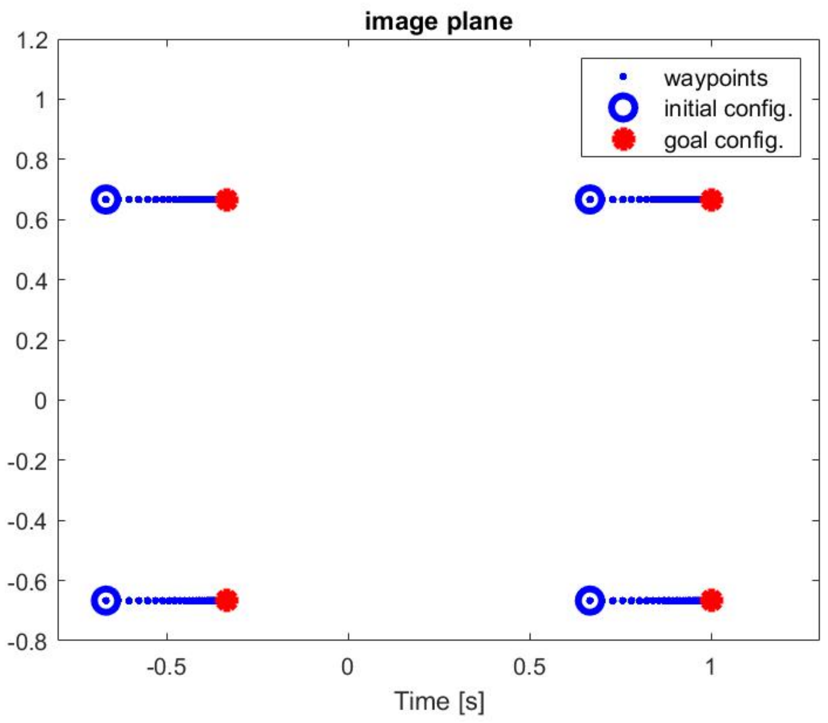

Figure 5.3: Motion of the feature on the image plane during a x translation of the camera from position $(0, 0, 2)$ to position $(0.5, 0, 2)$.

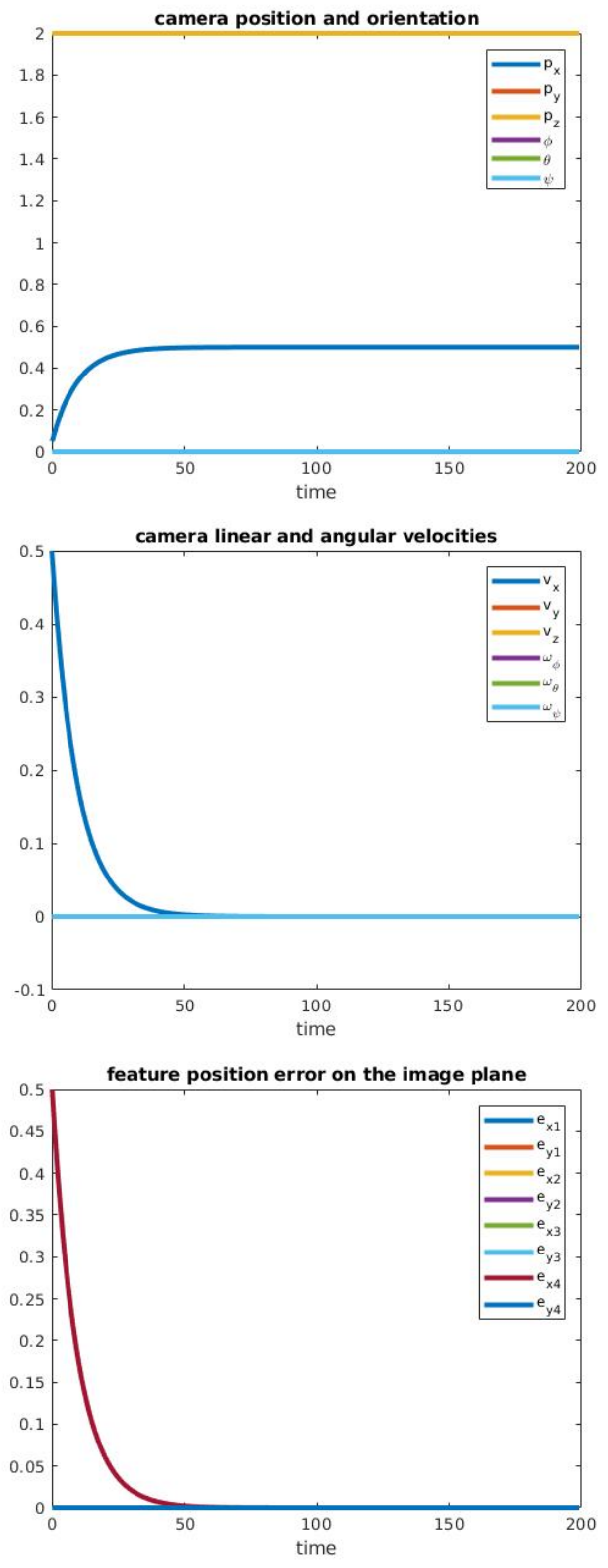

Figure 5.4: Plots of camera positions, velocities and error in position during a translation x translation of the camera from position $(0, 0, 2)$ to position $(0.5, 0, 2)$.

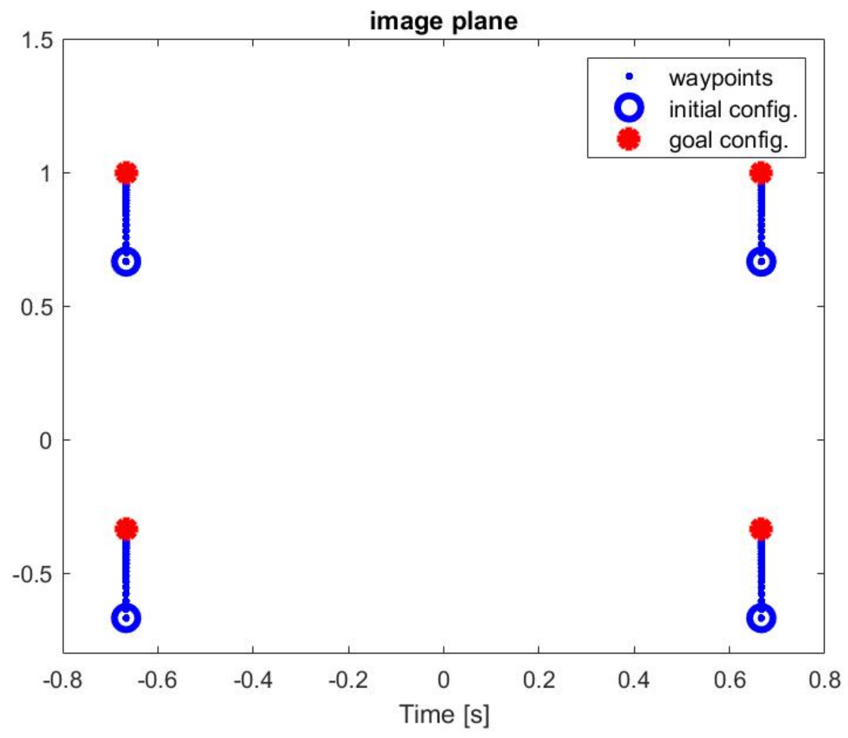

Figure 5.5: Motion of the feature on the image plane during a y translation of the camera from position $(0, 0, 2)$ to position $(0, 0.5, 2)$.

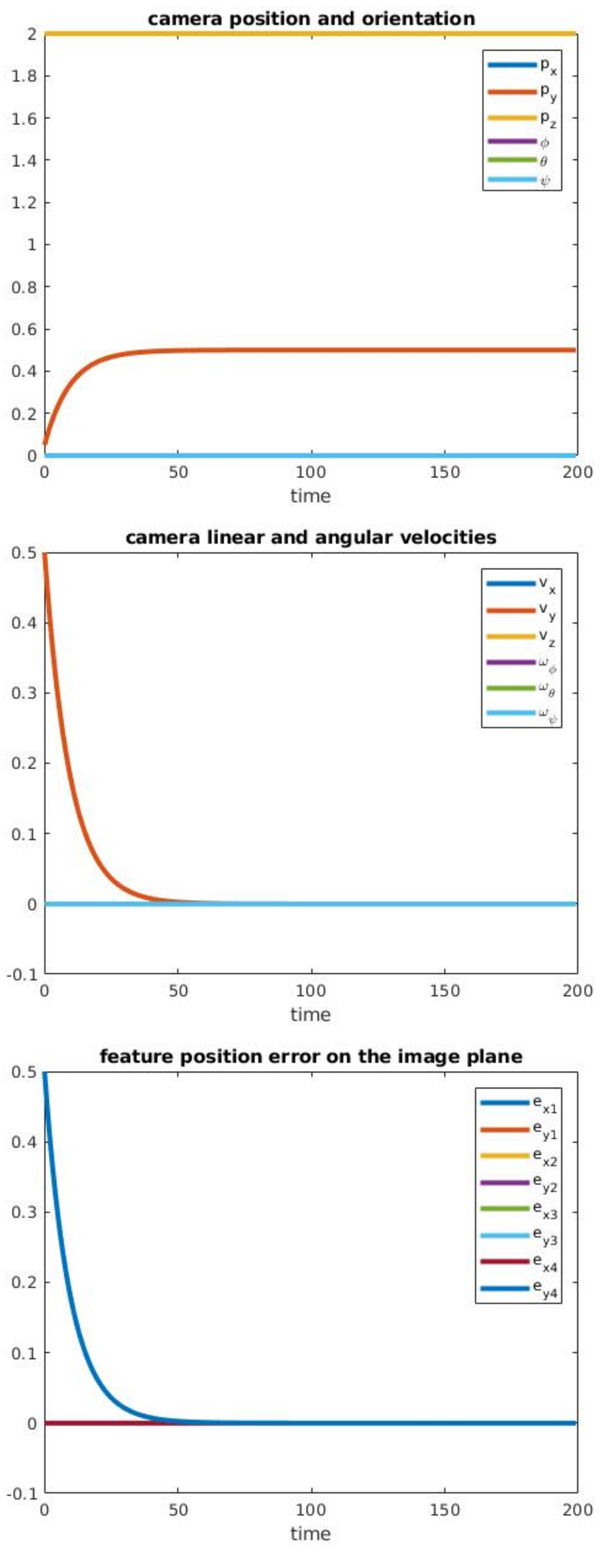

Figure 5.6: Plots of camera positions, velocities and error in position during a translation y translation of the camera from position $(0, 0, 2)$ to position $(0, 0.5, 2)$.

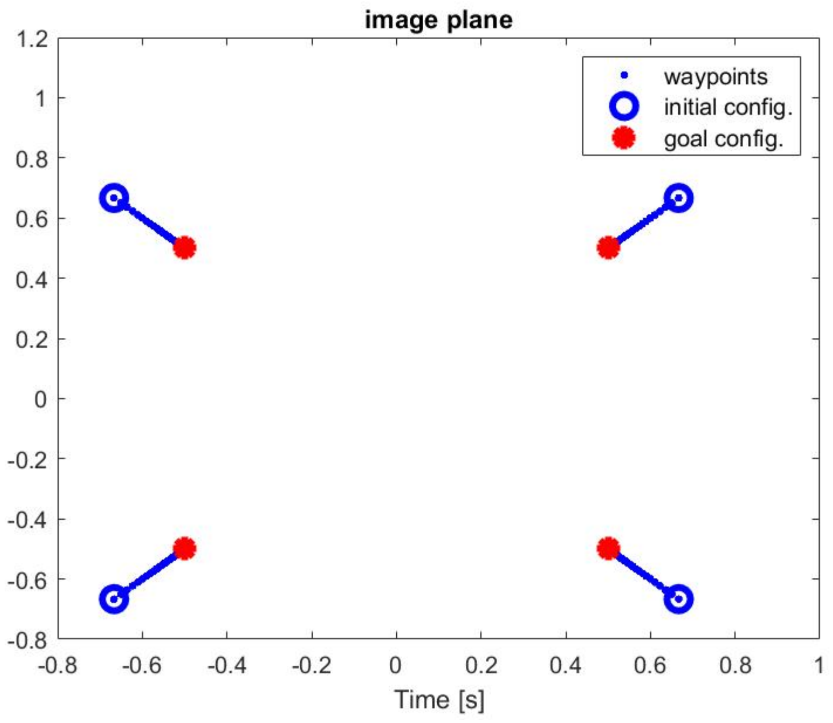

Figure 5.7: Motion of the feature on the image plane during a z translation of the camera from position $(0, 0, 2)$ to position $(0, 0, 2.5)$.

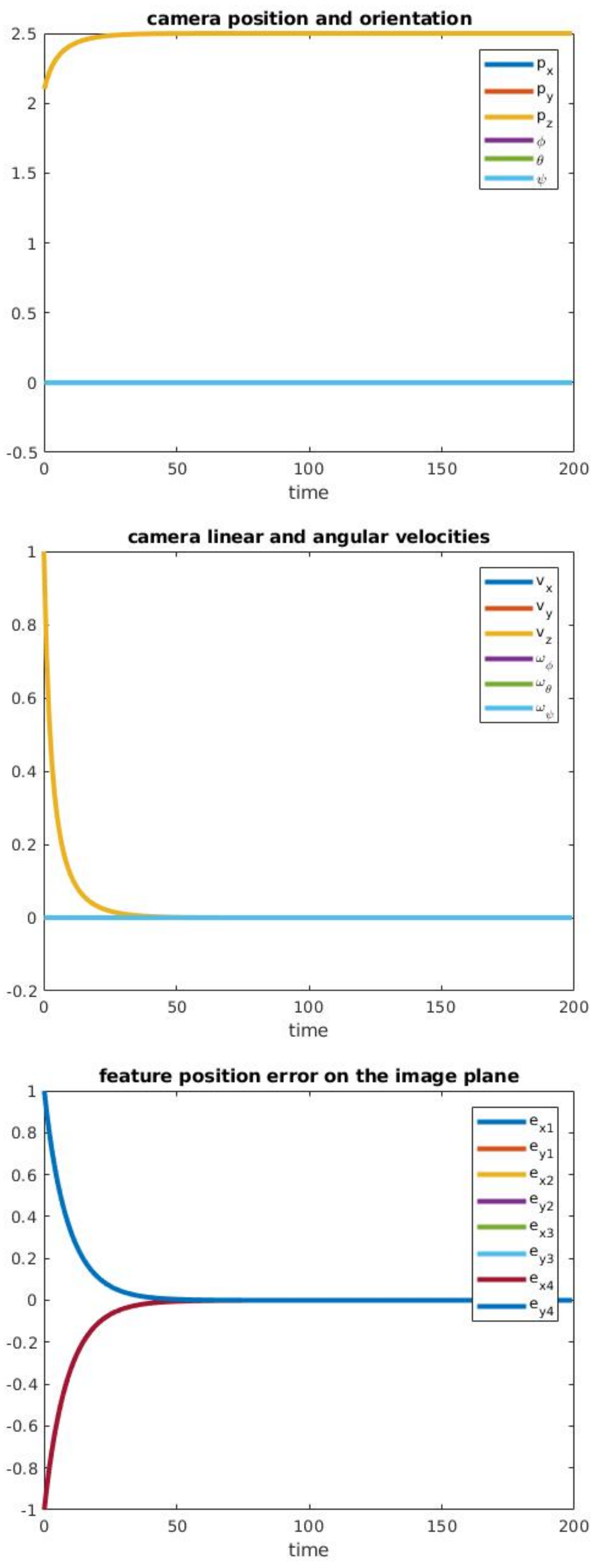

Figure 5.8: Plots of camera positions, velocities and error in position during a translation z translation of the camera from position $(0, 0, 2)$ to position $(0, 0, 2.5)$.

5.3 ROS implementation

After the preliminary analysis in MATLAB, the entire software architecture has been developed using ROS. It has been tested using the simulation environment Gazebo [19]. We have structured the code into three ROS nodes: one for detection, one for the visual servo control and one for the model predictive controller. In this section, we are going to describe in detail the functioning of each one of them and discuss the result obtained in simulation.

5.3.1 Perception node

This module is used to extract the image from the camera and to detect the features on the image. The features consist of four coloured points (green, blue, black and red). The implementation consists of a ROS node in C++ that uses OpenCV functions to detect the four blobs of the features and then publishes their positions to a specific topic such that then the subscriber in the IBVS node can access them. This node is just a prototype and it has been used just in the simulations. For the experiments, we did not have a camera mounted on the drone and for time constraints we decided to use the information of the UAV position from the Motion Capture System (MoCap) from the laboratory to compute the feedback information on the position of the features on the image plane by software. The focus of this thesis is to investigate how to overcome the problem of under-actuation when using visual servo control with UAV by adding a lower level MPC controller. For this reason, we did not go in-depth for the perception.

5.3.2 Visual Servo Control

The Visual Servo Control node, or IBVS node, is a ROS node in Python. It gets the position of the features in the image plane from the perception node and then it uses an image-based visual servo control algorithm, as described in chapter 3. It compute the required velocities that allow the UAV to reach its target goal position. This target positon for the robot is the one where the features on the image plane are in the desired configuration. This node is implemented using classes as illustrated in the scheme below:

- IBVS: here the desired velocities are computed using the theory explained in chapter 3,
- FEATURES CLASS: this class is the one that subscribes to the vision

node in order to access and update the position of each feature on the image plane,

- UAV CLASS: This class contains the model of the system used by the IBVS to compute the required velocities.

5.3.3 Model Predictive Control

Next is the lower-level controller which is an MPC, we are using this to convert the control input from velocities to attitude control. The MPC takes into account the fact that the UAV is under-actuated, by dynamical constraints and state constraints in the optimization problem. The optimization problem we are trying to solve is formulated in equation 4.23. We expressed it using the solver CVXGEN [26]. To guarantee that the node was fast enough, we used a C++ ROS node for this task. The model is a linear and discrete model of a quadrotor, computed off-line using MATLAB as described in Chapter 2, by linearizing around hovering configuration and discretizing using zero-order hold with a sample time of $T_s = 0.05$. The state-space model consists of nine states and four inputs and it's described as follows:

$$\mathbf{x}(t+1) = \mathbf{A}_d\mathbf{x}(t) + \mathbf{B}_d\mathbf{u}(t) \tag{5.2}$$

$$\mathbf{y}(t+1) = \mathbf{C}_d\mathbf{x}(t) + \mathbf{D}_d\mathbf{u}(t) \tag{5.3}$$

$$\tag{5.4}$$

where

$$\mathbf{A}_d = \begin{bmatrix} \mathbf{I}^{3\times3} & 0.4905[\mathbf{i}]_\times & 0.0122625[\mathbf{i}]_\times \\ \mathbf{0}^{3\times3} & \mathbf{I}^{3\times3} & 0.5\mathbf{I}^{3\times3} \\ \mathbf{0}^{3\times3} & \mathbf{0}^{3\times3} & \mathbf{I}^{3\times3} \end{bmatrix}, [\mathbf{i}]_\times = \begin{bmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{5.5}$$

$$\mathbf{B}_d = \begin{bmatrix} 0 & \frac{-0.000204375}{I_y} & 0 & 0 \\ \frac{-0.000204375}{I_x} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{0.05}{m} \\ \frac{0.00125}{I_x} & 0 & 0 & 0 \\ 0 & \frac{0.00125}{I_y} & 0 & 0 \\ 0 & 0 & \frac{0.00125}{I_z} & 0 \\ \frac{0.05}{I_x} & 0 & 0 & 0 \\ 0 & \frac{0.05}{I_y} & 0 & 0 \\ 0 & 0 & \frac{0.05}{I_z} & 0 \end{bmatrix}, \tag{5.6}$$

$$\mathbf{C}_d = \mathbf{I}^{9\times9}, \tag{5.7}$$

$$\mathbf{D}_d = \mathbf{0}^{9\times4}. \tag{5.8}$$

	min value	max value
τ_ϕ	-0.1	0.1
τ_θ	-0.1	0.1
τ_ψ	-0.5	0.5
f	0	30
ϕ	-0.26	0.26
θ	-0.26	0.26
ψ	-0.26	0.26

Table 5.1: Constraints on the states from the MPC optimization problem

The time horizon used is $T = 10s$ and we added some constraints to the input and states to make sure the system remains in close to the equilibrium point around which we linearized the model, in particular the values are contained in table 5.1. These values have been chosen empirically to ensure that the system remains close enough to the equilibrium point around which the model has been linearized.

Moreover, we used the RotorS [9] that converts from roll-pitch-yawrate-thrust to the actuators input command. Here we are sending as output roll, pitch, yaw rate and thrust instead of torques and thrust. To do so we are directly feeding in the predicted states for roll, pitch and yaw rate and the computed control input for the thrust. Note that this method works for translations only, not for rotations as allowing small rotations around the z-axis would change the kind of optimization problem.

We consider the following cost function:

$$V(x(0), \mathbf{u}) = \tag{5.9}$$

$$\sum_{t=0}^{T-1} \Delta\mathbf{x}[t]^T \mathbf{Q} \Delta\mathbf{x}[t] + \mathbf{u}[t]^T \mathbf{R} \mathbf{u}[t] + \Delta\mathbf{x}[T+1]^T \mathbf{Q}_{final} \Delta\mathbf{x}[T+1] \tag{5.10}$$

where $\Delta\mathbf{x}[t] = \mathbf{x}[t] - \mathbf{x}_{ref}$, $\Delta\mathbf{x}[T+1] = \mathbf{x}[T+1] - \mathbf{x}_{ref}$ and $\mathbf{Q} \geq 0, \mathbf{R} > 0$ and $\mathbf{Q}_{final} \geq 0$. State, final state and input costs $\mathbf{Q}$ and $\mathbf{R}$ are tuned to improve performance, the final value selected are the following:

$$\mathbf{Q} = \begin{bmatrix} \mathbf{0}^{3\times3} & 100\mathbf{I}^{3\times3}\mathbf{a} & \mathbf{0}^{3\times3} & \mathbf{0}^{3\times3} \\ \mathbf{0}^{3\times3} & \mathbf{0}^{3\times3} & \mathbf{I}^{3\times3} & \mathbf{0}^{3\times3} \\ \mathbf{0}^{3\times3} & \mathbf{0}^{3\times3} & \mathbf{0}^{3\times3} & \mathbf{I}^{3\times3} \end{bmatrix}, \mathbf{a} = \begin{bmatrix} 1 \\ 1 \\ 3 \end{bmatrix} \tag{5.11}$$

$$\mathbf{Q}_{final} = \begin{bmatrix} \mathbf{0}^{3\times3} & 500\mathbf{I}^{3\times3}\mathbf{a} & \mathbf{0}^{3\times3} & \mathbf{0}^{3\times3} \\ \mathbf{0}^{3\times3} & \mathbf{0}^{3\times3} & \mathbf{I}^{3\times3} & \mathbf{0}^{3\times3} \\ \mathbf{0}^{3\times3} & \mathbf{0}^{3\times3} & \mathbf{0}^{3\times3} & 100\mathbf{I}^{3\times3} \end{bmatrix}, \mathbf{a} = \begin{bmatrix} 1 \\ 1 \\ 2 \end{bmatrix} \tag{5.12}$$

$$\mathbf{R} = \begin{bmatrix} 1000 & 0 & 0 & \\ 0 & 200 & 0 & 0 \\ 0 & 0 & 200 & 0 \\ 0 & 0 & 0 & 100 \end{bmatrix}. \tag{5.13}$$

The weights have been tuned with a trial and error procedure in order to find the best configurations. We wanted to penalize the deviation between the actual and the desired linear and angular velocities and input.

5.4 Gazebo Simulation

Preliminary tests of the full software architecture have been performed in Gazebo using the model of a UAV SRD370. We approximated its mass and inertia as follows: $m = 1.45kg$, $I_x = 0.04$, $I_y = 0.04$ and $I_z = 0.1$. In simulation, the MPC solver took a minimum of $0.72ms$, an average of $5.8ms$ and a maximum of $1.1ms$. From Fig. 5.10, 5.12 and 5.14 we observe how the system converges to the desired set-points with no steady state error. Moreover, in Fig. 5.9,5.11 and 5.13 we can observe the trajectory followed by the features on the image plane. The tests included translations along x, y and z axis. Respectively the three translations are as follows:

- from position $(-0.2, 0.2, 1.2)$ to position $(0.2, 0.2, 1.2)$,
- from position $(-0.2, -0.2, 1.2)$ to position $(-0.2, 0.2, 1.2)$,
- from position $(0.2, 0.2, 1.2)$ to position $(0.2, 0.2, 0.8)$.

5.5 Experiments

All the experiments have been performed in the Smart Mobility Lab (SML), that is a hub for the development and experimentation of intelligent transportation solutions inside the Integrated System Research Lab (ITRL) KTH.

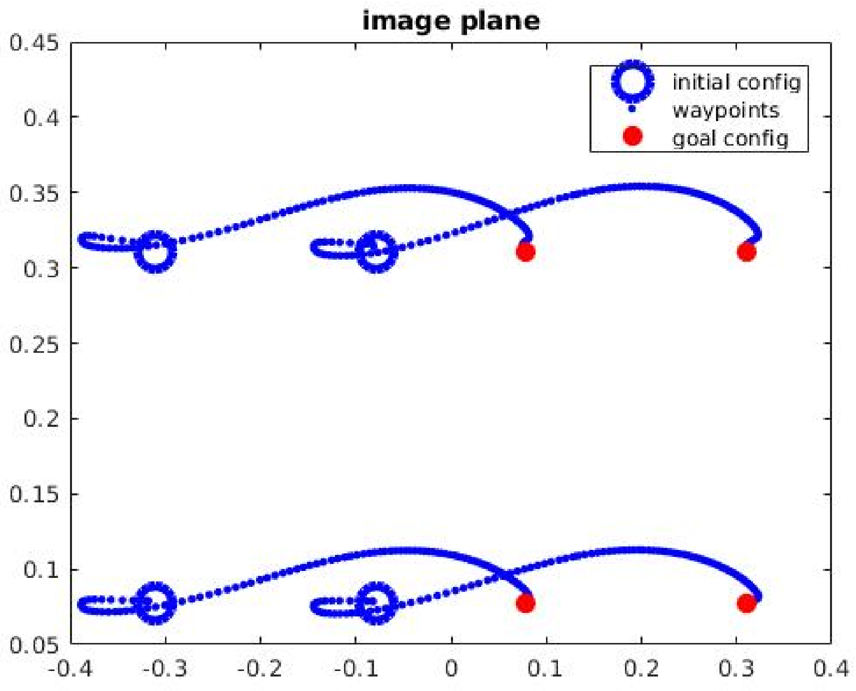

Figure 5.9: Motion of the feature on the image plane during a x translation of the camera from position $(-0.2, 0.2, 1.2)$ to position $(0.2, 0.2, 1.2)$.

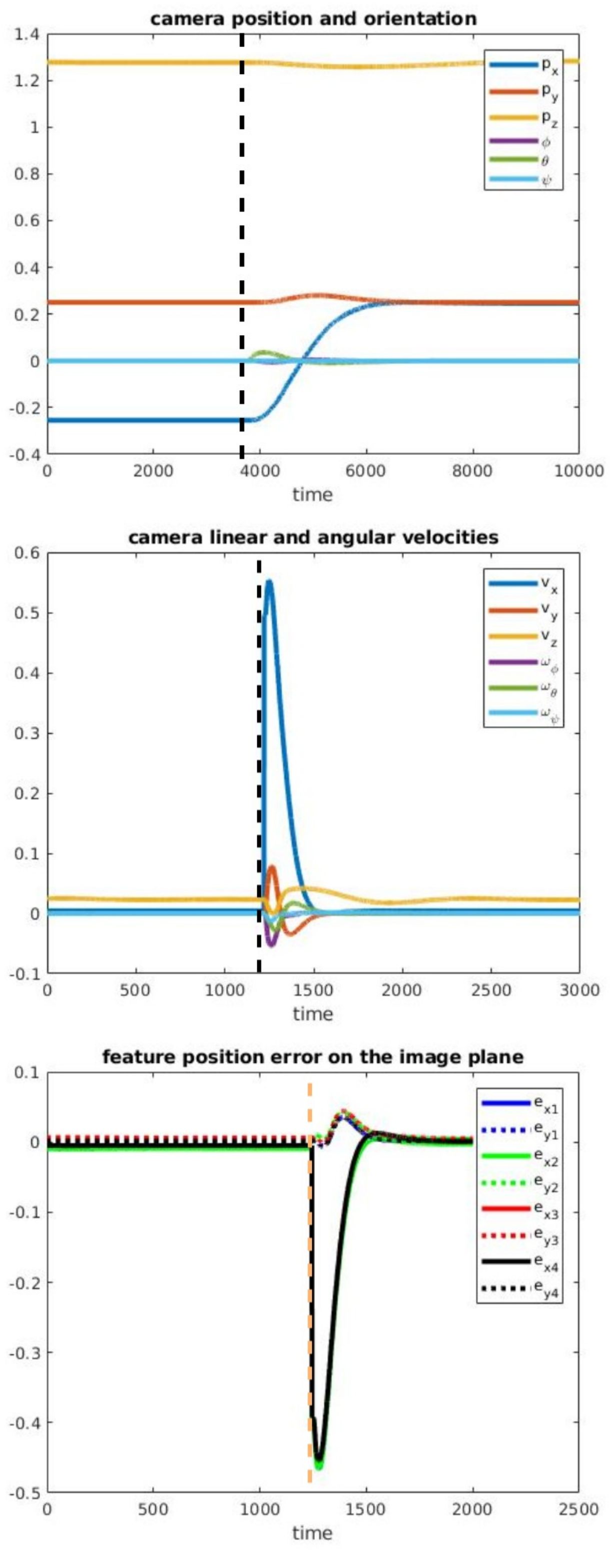

Figure 5.10: Plots of camera positions, velocities and feature position error in the image plane during a translation x translation of the camera from position $(-0.2, 0.2, 1.2)$ to position $(0.2, 0.2, 1.2)$. The vertical dashed line indicates the time instant when the new goal position is set.

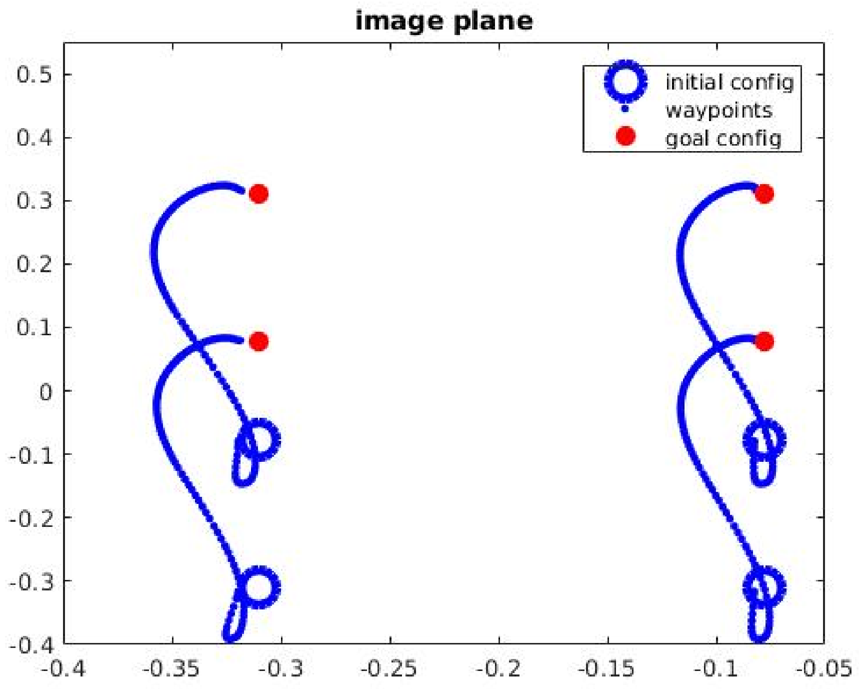

Figure 5.11: Motion of the feature on the image plane during a y translation of the camera from position $(-0.2, -0.2, 1.2)$ to position $(-0.2, 0.2, 1.2)$.

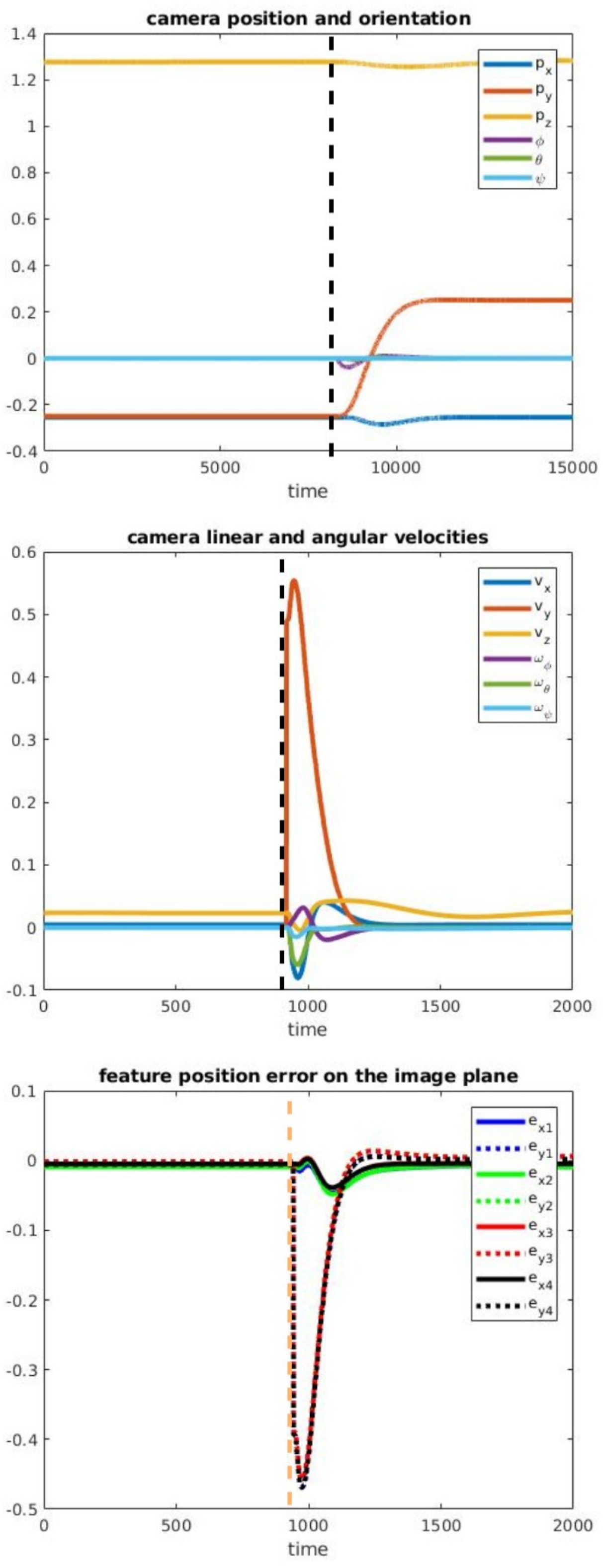

Figure 5.12: Plots of camera positions, velocities and feature position error in the image plane during a translation x translation of the camera from position $(-0.2, -0.2, 1.2)$ to position $(-0.2, 0.2, 1.2)$. The vertical dashed line indicates the time instant when the new goal position is set.

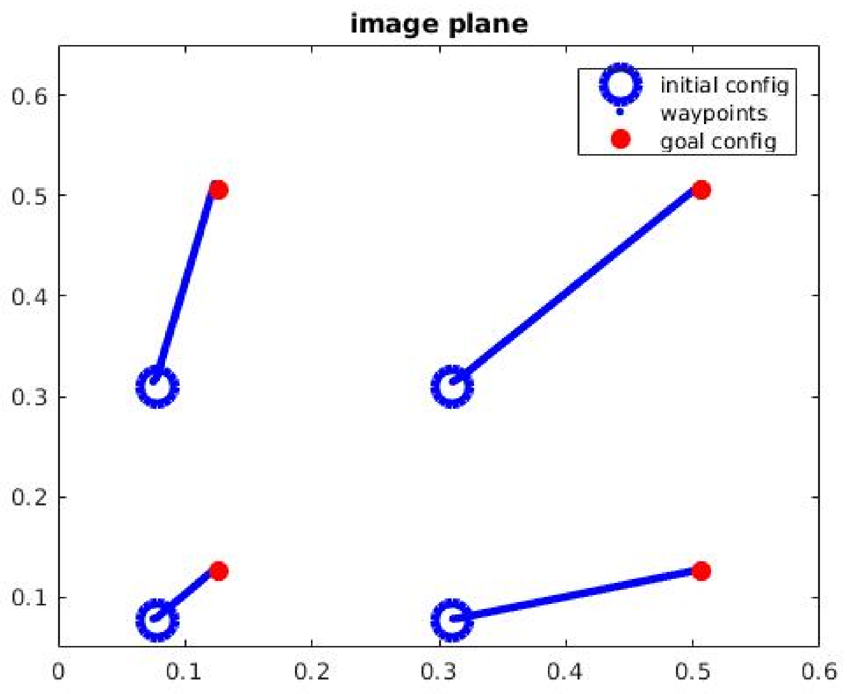

Figure 5.13: Motion of the feature on the image plane during a z translation of the camera from position $(0.2, 0.2, 1.2)$ to position $(0.2, 0.2, 0.8)$.

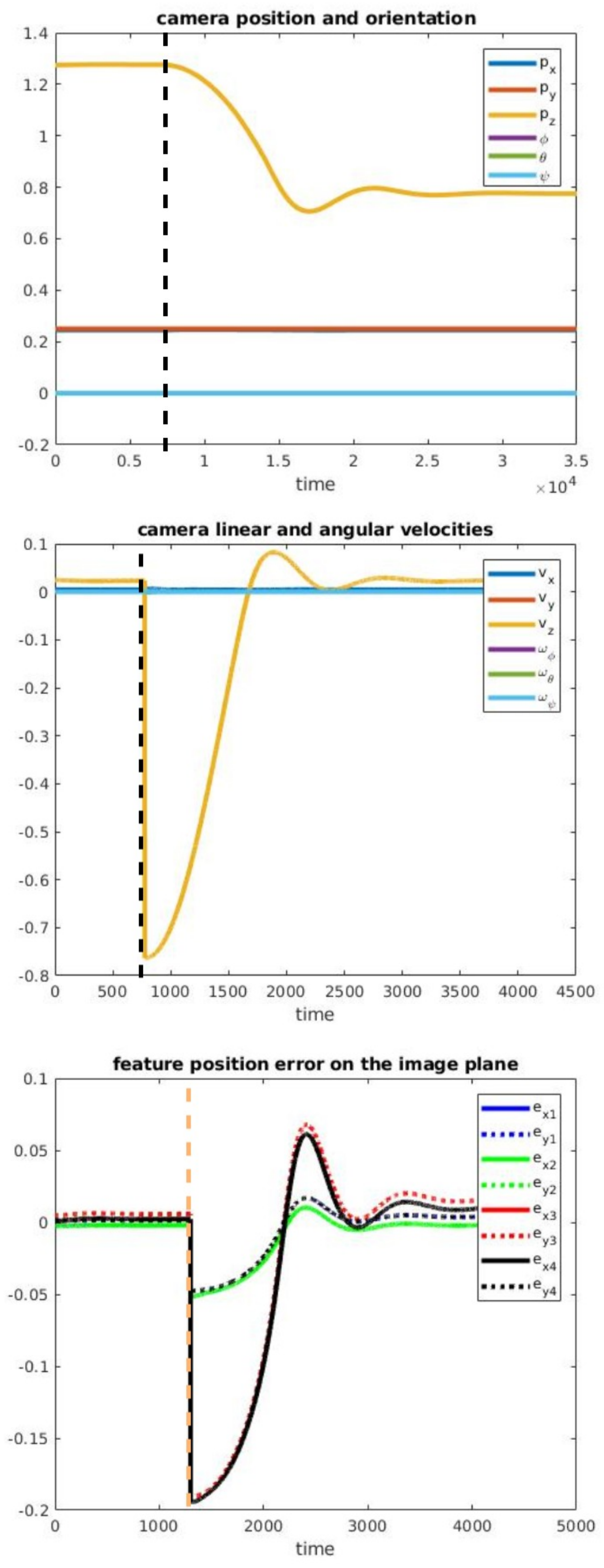

Figure 5.14: Plots of camera positions, velocities and feature position error in the image plane during a translation x translation of the camera from position $(0.2, 0.2, 1.2)$ to position $(0.2, 0.2, 0.8)$. The vertical dashed line indicates the time instant when the new goal position is set.

In SML there is a flying arena for UAVs equipped with motion capture system (MoCap). The test-bed used for the experiments is based on a Foxtech Hover 1 Quadrotor, with an Nvidia Jetson TX2 and a PX4 flight controller. We approximated its mass and inertia as follows: $m = 1.73kg$, $I_x = 0.04$, $I_y = 0.04$ and $I_z = 0.1$. The experiment has been performed with a series of translations of $0.5m$ along x, y and z axis. To guarantee safety during the experiment, a safety switch has been developed using a ROS service. The switch has been used to transition from our controller to a safe PID during the tuning of the weights of the MPC. From Fig. 5.15, 5.16 and 5.17 we observe how the system converges to the desired set-points, although a small steady state error. Comparing these results with the one obtained in simulation, we can notice a small decrease in tracking performances that is caused by the uncertainty o the inertial parameters and an inaccurate force-to-trust mapping that depend on motor properties and propellers performances. During the experiments, the MPC solver took a minimum of $7.9ms$, an average of $8.4ms$ and a maximum of $10.8ms$. The tests included translations along x, y and z axis. Respectively the three translations are as follows:

- from position $(-0.2, 0.2, 1.2)$ to position $(0.2, 0.2, 1.2)$,
- from position $(-0.2, -0.2, 1.2)$ to position $(-0.2, 0.2, 1.2)$,
- from position $(0.2, 0.2, 1.2)$ to position $(0.2, 0.2, 0.8)$.

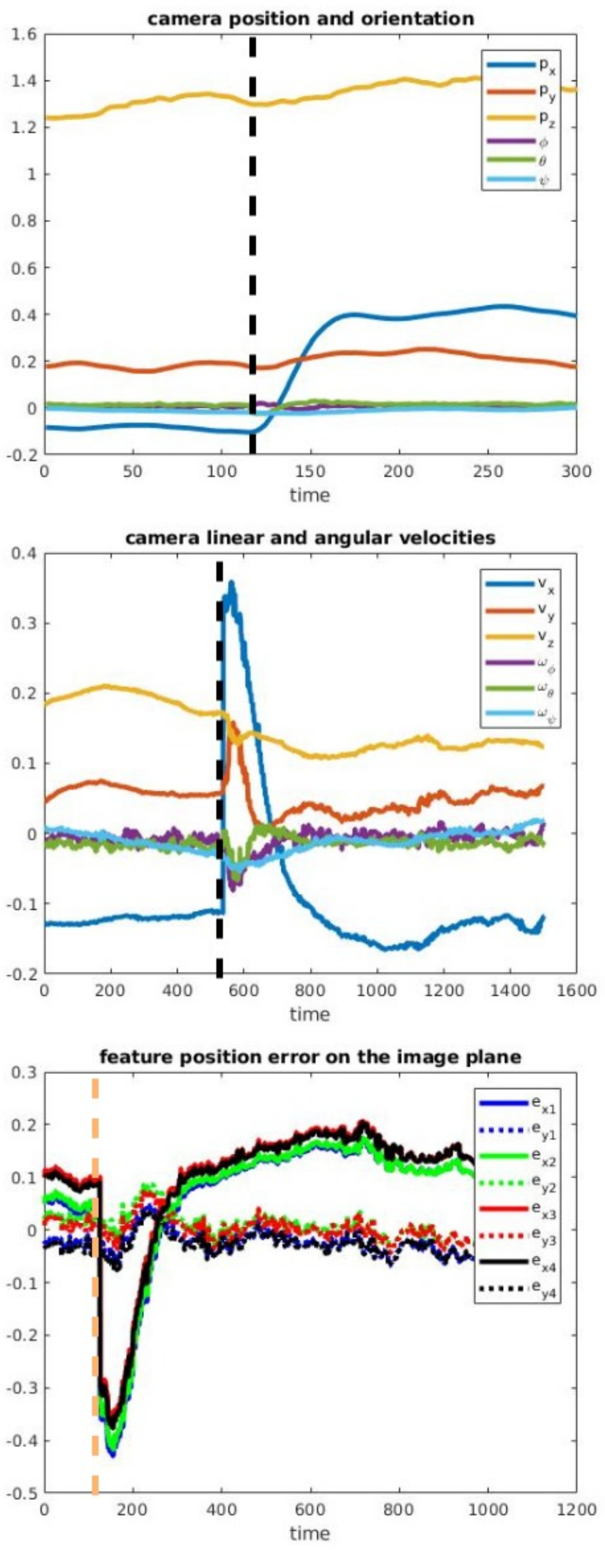

Figure 5.15: Plots of camera positions, velocities and feature position error in the image plane during a translation x translation of the camera from position $(-0.2, 0.2, 1.2)$ to position $(0.2, 0.2, 1.2)$. The vertical dashed line indicates the time instant when the new goal position is set.

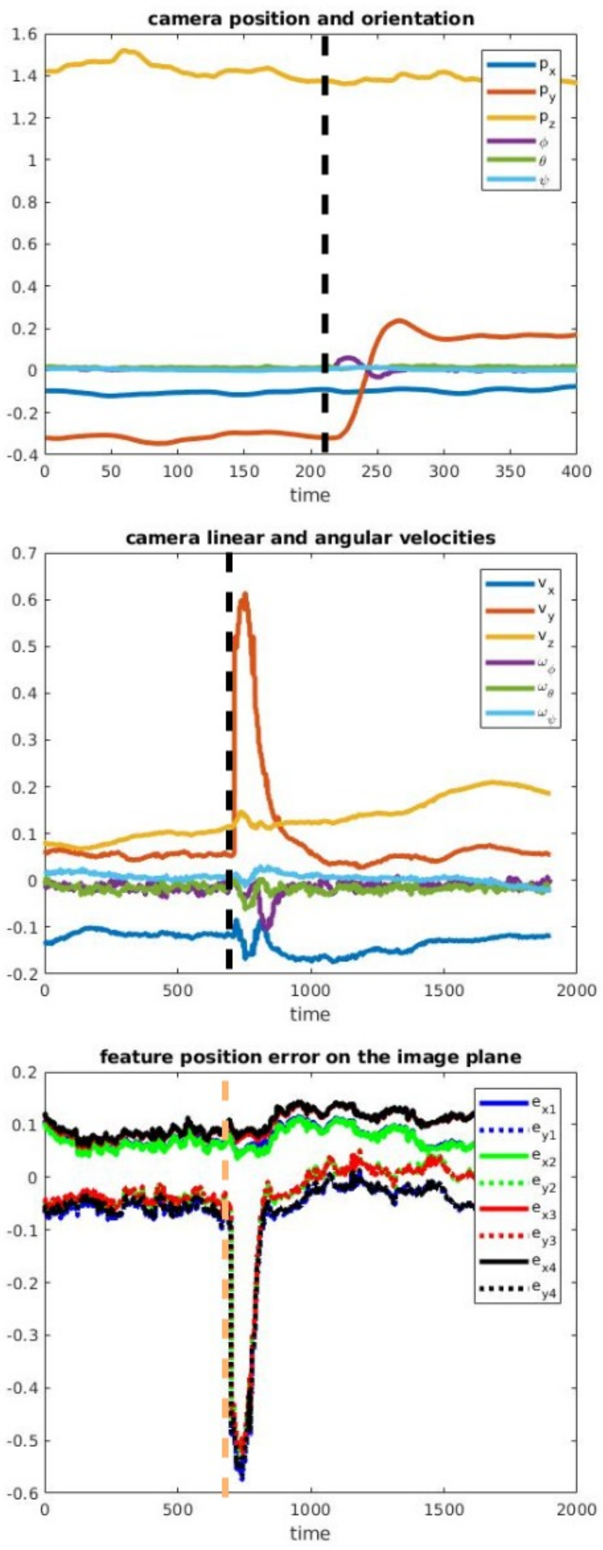

Figure 5.16: Plots of camera positions, velocities and feature position error in the image plane during a translation y translation of the camera from position $(-0.2, -0.2, 1.2)$ to position $(-0.2, 0.2, 1.2)$.

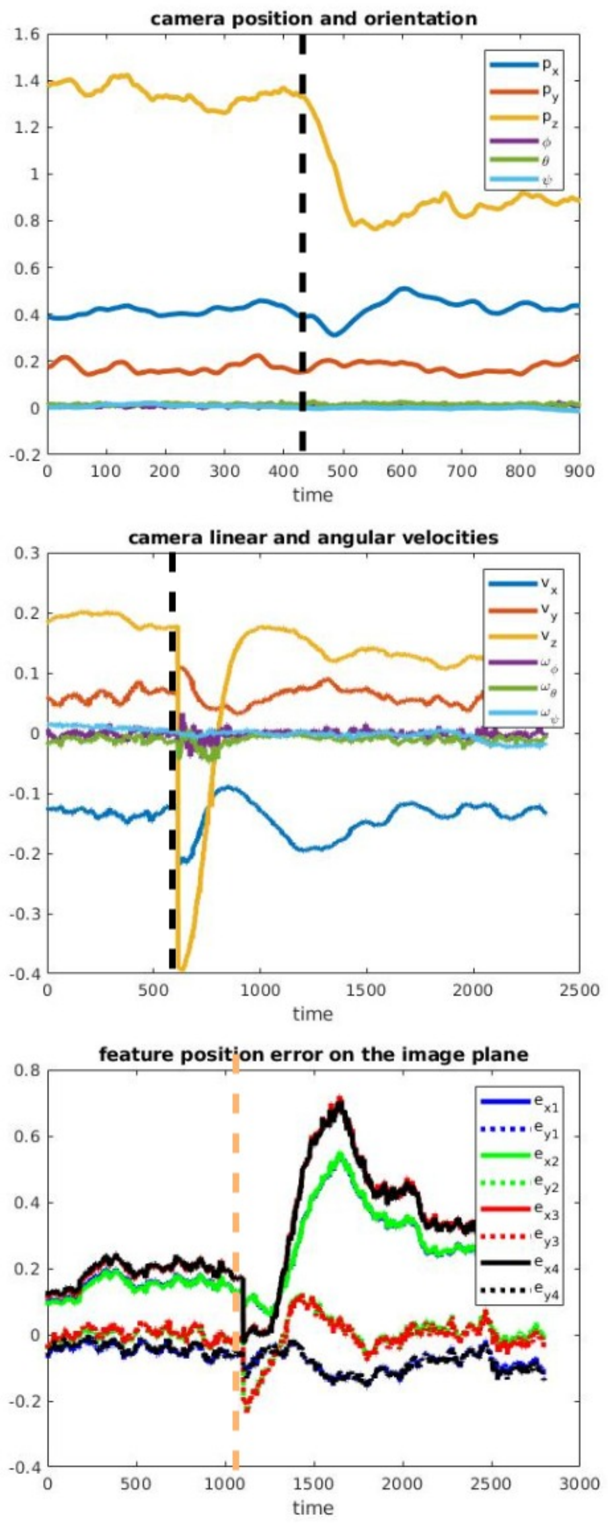

Figure 5.17: Plots of camera positions, velocities and feature position error in the image plane during a translation z translation of the camera from position $(0.2, 0.2, 1.2)$ to position $(0.2, 0.2, 0.8)$. The vertical dashed line indicates the time instant when the new goal position is set.

Chapter 6

Conclusions

6.1 Contributions

With this work, we presented a novel method to tackle the under-actuation issue in the control of UAVs using image-based visual servoing. We implemented a linear model predictive control loop to solve the issue. We tested the proposed algorithm both with synthetic data in the Gazebo environment and in experimental scenarios with UAVs, to prove that the method is fast enough to run at high-frequency on-board the vehicle.

In particular, we focused our study on visual servoing that is a well-known control strategy that uses visual information as feedback to control the motion of a robot with respect to a given reference without the need of knowing its global position. Despite the many advantages that this method provides, applying it for the control of drones is not trivial. In particular the problem of the system being under-actuated needs to be faced: visual servoing algorithms output a required velocity to be tracked by the vehicle for each of the six degrees of freedom, but the UAV is actuated just along four of them. For this reason, it is essential that the tracking takes into account the actual dynamics of the system and its physical limitations that are not taken into account by the visual servoing. We investigate the use of linear model predictive control (MPC) to accomplish this task. MPC is an effective advanced control method that uses the dynamic model of the system to control it while satisfying a set of constraints.

6.2 Results

A preliminary implementation has been prepared in MATLAB to simulate the visual servo control loop. For the sake of this preliminary study, we are considering as inputs the starting and goal position of the camera in the world frame. Then, given the fixed position of the four point features in the world frame, we compute the positions of each feature in the image frame by projection. Then, we compute the visual-servo control law such that the error on the features positions and velocities on the image plane goes to zero. When that happens it means that that camera has reached the location in the real world in the given goal position. Here, to simulate the UAV behaviour, we used the linear dynamical model described in chapter 2. Then we developed a new software architecture based on ROS, with visual servo control node written in python and a model predictive control node written in C++. The solver for the model predictive control optimization problem has been generated with CVXGEN. The first tests of the full software architecture have been performed in Gazebo using the model of a UAV SRD370. Furthermore, experiments have been performed in the flying arena of the Smart Mobility Lab (SML), that is equipped with a motion capture system (MoCap). The test-bed used for the experiments is based on a Foxtech Hover 1 Quadrotor, with an Nvidia Jetson TX2 and a PX4 flight controller. Both in simulation and in the experiments we tested translations along x, y and z axes. From the plots reported in chapter 5, one can observe how the system converges to the desired state. Althrough there is some steady state error in the experiments with the real vehicle due to uncertainties in the model, we achieved good performances.

Bibliography

[1] G. Allibert, E. Courtial, and F. Chaumette. Predictive control for constrained image-based visual servoing. *IEEE Transactions on Robotics*, 26(5):933–939, Oct 2010.

[2] M. Bangura and R. Mahony. Real-time model predictive control for quadrotors. *IFAC Proceedings Volumes*, 47(3):11773 – 11780, 2014. 19th IFAC World Congress.

[3] D. Cabecinhas, S. Brás, R. Cunha, C. Silvestre, and P. Oliveira. Integrated visual servoing solution to quadrotor stabilization and attitude estimation using a pan and tilt camera. *IEEE Transactions on Control Systems Technology*, 27(1):14–29, Jan 2019.

[4] E. Camacho and C. Bordons. *Model Predictive Control*, volume 13. 01 2004.

[5] F. Chaumette. Potential problems of stability and convergence in image-based and position-based visual servoing. In Kriegman, D., Hager, G. ., Morse, and A.S., editors, *The Confluence of Vision and Control*, pages 66–78. LNCIS Series, No 237, Springer-Verlag, 1998.

[6] F. Chaumette and S. Hutchinson. Visual servo control, Part I: Basic approaches. *IEEE Robotics and Automation Magazine*, 13(4):82–90, 2006.

[7] F. Chaumette and S. Hutchinson. Visual servo control, Part II: Advanced approaches. *IEEE Robotics and Automation Magazine*, 14(1):109–118, 2007.

[8] B. Espiau, F. Chaumette, and P. Rives. A new approach to visual servoing in robotics. *IEEE Transactions on Robotics and Automation*, 8(3):313–326, June 1992.

[9] F. Furrer, M. Burri, M. Achtelik, and R. Siegwart. *Robot Operating System (ROS): The Complete Reference (Volume 1)*, chapter RotorS—A Modular Gazebo MAV Simulator Framework, pages 595–625. Springer International Publishing, Cham, 2016.

[10] J. Gao, G. Zhang, P. Wu, X. Zhao, T. Wang, and W. Yan. Model predictive visual servoing of fully-actuated underwater vehicles with a sliding mode disturbance observer. *IEEE Access*, 7:25516–25526, 2019.

[11] N. Guenard, T. Hamel, and R. Mahony. A practical visual servo control for an unmanned aerial vehicle. *IEEE Transactions on Robotics*, 24(2):331–340, April 2008.

[12] N. Guenard, T. Hamel, and R. E. Mahony. A practical visual servo control for an unmanned aerial vehicle. *IEEE Transactions on Robotics*, 24:331–340, 2007.

[13] L. Gurobi Optimization. Gurobi optimizer reference manual, 2020.

[14] T. Hamel and R. Mahony. Visual servoing of an under-actuated dynamic rigid-body system: an image-based approach. *IEEE Transactions on Robotics and Automation*, 18(2):187–198, April 2002.

[15] T. Hamel and R. Mahony. Image based visual servo-control for a class of aerial robotic systems. *Automatica*, 43:1975 – 1983, May 2007.

[16] R. Kalman. Contribution to the theory of optimal control. *Bull. Soc. Math*, Mex.5:102–119, 1960.

[17] R. Kalman. A new approach to linear filtering and prediction problems. *Trans. ASME, J. Basic Engineering*, pages 35–45, 1960.

[18] M. Kamel, M. Burri, and R. Siegwart. Linear vs nonlinear MPC for trajectory tracking applied to rotary wing micro aerial vehicles. *CoRR*, abs/1611.09240, 2016.

[19] N. Koenig and A. Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS) (IEEE Cat. No.04CH37566)*, volume 3, pages 2149–2154 vol.3, Sep. 2004.

[20] D. Kragic and M. Vincze. Vision for robotics. *Foundations and Trends® in Robotics*, 1(1):1–78, 2009.

[21] C. Lazar and A. Burlacu. Visual servoing of robot manipulators using model-based predictive control. In *2009 7th IEEE International Conference on Industrial Informatics*, pages 690–695, June 2009.

[22] C. Liu, H. Lu, and W. Chen. An explicit mpc for quadrotor trajectory tracking. In *2015 34th Chinese Control Conference (CCC)*, pages 4055–4060, July 2015.

[23] J. Löfberg. Yalmip : A toolbox for modeling and optimization in matlab. In *In Proceedings of the CACSD Conference*, Taipei, Taiwan, 2004.

[24] R. Mahony, V. Kumar, and P. Corke. Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *IEEE Robotics Automation Magazine*, 19(3):20–32, Sep. 2012.

[25] R. E. Mahony, V. S. A. Kumar, and P. I. Corke. Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *IEEE Robotics and Automation Magazine*, 19:20–32, 2012.

[26] J. Mattingley and S. Boyd. CVXGEN: A code generator for embedded convex optimization. *Optimization and Engineering*, 12(1):1–27, 2012.

[27] L. Mejias, P. Campoy, S. Saripalli, and G. S. Sukhatme. A visual servoing approach for tracking features in urban areas using an autonomous helicopter. In *Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006.*, pages 2503–2508, May 2006.

[28] N. Metni, T. Hamel, and F. Derkx. A uav for bridge's inspection: visual servoing control law with orientation limits. *IFAC Proceedings Volumes*, 37(8):454 – 459, 2004. IFAC/EURON Symposium on Intelligent Autonomous Vehicles, Lisbon, Portugal, 5-7 July 2004.

[29] S. J. Qin and T. A. Badgwell. A survey of industrial model predictive control technology. 2003.

[30] A. Raemaekers. *Design of a model predictive controller to control UAVs*. DCT rapporten. Technische Universiteit Eindhoven, 2007. DCT 2007.141 Stageverslag.

[31] J. Rawlings, D. Mayne, and M. Diehl. *Model Predictive Control: Theory, Computation, and Design*. 01 2017.

[32] Ruchika and Neha Raghu. Model predictive control: History and development. *International Journal of Engineering Trends and Technology (IJETT)*, 4(6):2600–2602, Jun 2013.

[33] Y. Shirai and H. Inoue. Guiding a robot by visual feedback in assembling tasks. *Pattern Recognition*, 5:99–108, 1973.

[34] Y. Shirai and H. Inoue. Guiding a robot by visual feedback in assembling tasks. *Pattern Recognition*, 5:99–106, 1973.

[35] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo. *Visual Servoing*, pages 407–467. Springer London, London, 2009.

[36] T. Templeton, D. H. Shim, C. Geyer, and S. S. Sastry. Autonomous vision-based landing and terrain mapping using an mpc-controlled unmanned rotorcraft. In *Proceedings 2007 IEEE International Conference on Robotics and Automation*, pages 1349–1356, April 2007.