TÉCNICO LISBOA

Vision based real-time obstacle avoidance for drones while following a moving target

Francisco Basílio de Aguiar Gomes

Thesis to obtain the Master of Science Degree in

Electrical and Computer Engineering

Supervisors: Prof. Rodrigo Martins de Matos Ventura
Eng. Nuno Tiago Salavessa Cardoso Hormigo Vicente

Examination Committee

Chairperson: Prof. João Fernando Cardoso Silva Sequeira
Supervisor: Prof. Rodrigo Martins de Matos Ventura
Member of the Committee: Prof. Alexandre José Malheiro Bernardino

September 2020

Dedicated to my family

Declaration

I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

First of all, I would like to thank my thesis supervisors, Prof. Rodrigo Ventura and Eng. Tiago Hormigo for the support and the possibility to work on this topic.

Furthermore, I would like to thank all my friends that, along these years, help me in and out the school. During this final period, I would like to thank the persons from the ISR: Prasad, Monica, and Kagan for the first and important push in C++ and Acado details. To the partners in the lab Rute and Formigão, thanks for all the moments shared in the long days at 6.12.

For last, and most important, I would like to thank my family for the constant support and motivation in all the aspects of my life and for giving me all the conditions to study and be happy.

Resumo

O problema deste trabalho consiste no desvio de obstáculos em tempo real em voos autónomos. Devido ao aumento da utilização de drones em múltiplas aplicações como missões de salvamento ou exploração de ambientes críticos este tópico acaba por ser muito relevante e actual. Esta tese tem como objectivo desenvolver um sistema de controlo para um drone utilizando uma câmara e um *Inertial Measurement Unit* (IMU) de forma a garantir um voo seguro enquanto segue um alvo. Algumas abordagens tradicionais focam-se na reconstrução total do espaço, no entanto, esta tese explora uma solução reactiva onde serão analisadas imagens e o Tempo para Colisão será calculado para evitar obstáculos na frente do veículo.

Numa primeira fase, foi desenvolvido um algoritmo de visão que identifica um alvo a seguir, calcula o Tempo para Colisão e identifica as zonas perigosas para o drone. De seguida, foi implementado um sistema de controlo em cascata composto por um controlador de modelo preditivo responsável por controlar a posição do drone e um controlador de atitude que calcula a velocidade dos rotores de forma a garantir um voo estável. O algoritmo de controlo preditivo usa a informação do alvo para conduzir o drone a uma posição que permita seguir o alvo em tempo real e, simultaneamente, usa os resultados do Tempo para Colisão para criar potenciais que repilam o drone levando-o a desviar-se de obstáculos.

Os resultados de vários voos em diferentes ambientes de simulação mostram que o drone consegue desviar-se de obstáculos enquanto segue um alvo. Contudo verificou-se que, em certos ambientes, a estratégia desenvolvida apresentou algumas imprecisões que levaram o sistema a estimar erradamente a distância relativa até ao obstáculo.

Palavras-chave: Drone, Controlo Preditivo, Visão por Computador, Tempo para Colisão, Desvio de Obstáculos, Seguimento de um Alvo.

Abstract

The problem of this project consists of real-time obstacle avoidance in autonomous flights. It is a relevant topic considering the increase of drone usage in multiple domains such as rescue missions or critical environments explorations and the need to obtain a collision-free system. This work focus on the development of a drone control system using as external sensors a camera and an IMU to ensure a safe flight while following a target. Whereas some traditional approaches focus on full space reconstruction, this work explores a different, reactive and less demanding computational method where it will be used image data to compute the Time-to-Collision (TTC).

In a first stage, it was designed a vision strategy where the target is identified, the TTC is computed and the zones in the image with a low TTC are determined. In a second stage, it was implemented a cascade control strategy that uses a Model Predictive Control (MPC) controller to control the drone position and an attitude controller to set up the speed of the rotors that guarantee flight stability. The MPC algorithm uses the target information from the vision method to lead the drone to a position that ensures a correct tracking in real-time and, simultaneously, use the TTC data to generate potential fields in the vehicle reference frame ensuring obstacle avoidance.

The results in the multiple flights performed in different environments show that the drone can accurately avoid obstacles while tracking a target. However, it was verified that, in certain environments, the developed strategy reveals some limitations that culminate in wrong relative distance estimates threw the obstacle.

Keywords: Drone, Model Predictive Control, Computer Vision, Time-to-Collision, Obstacle Avoidance, Target Tracking.

Contents

List of Tables

List of Figures

Nomenclature

Roman symbols

$\{B\}$	Body frame
$\{W\}$	World frame
r	Translation vector
R	Rotation matrix
${}^{W}R_{B}$	Rotation matrix from $\{W\}$ to $\{B\}$
m	Drone mass
F	Force vector
I	Inertia matrix
K_p	Proportional gain
K_i	Integrative gain
K_d	Derivative gain
$e(t)$	Error term
$u(t)$	Commands / Controls
u_0	First Control action
x^+	Sucessor state
$\mathcal{C}(x, u)$	Cost Function
$f(x, u)$	Drone model
g, h	Restrict set of states
p^T	Position of the drone
v^T	Velocity of the drone
q^T	Orientation of the drone
X	Point in space
f	Focal length
T	Translation vector
K	Intrinsic matrix
$I(x, y, t)$	Intensity at position (x, y) in the tth time step
(u, v)	Motion field

Greek symbols

Ω	Drone angular velocity
τ	Momentum / Torque vector
ω_i	rotor i angular speed
$\phi,\ \theta,\ \psi$	Roll, Pitch, Yaw respectively

Mathematical symbols

$\mathbb{R}^n$	n-Dimensional space
$\mathbb{N}$	Set of natural numbers
$\mathbb{U}$	Controls domain

Subscripts and Superscripts

B	Body frame
W	World / Inertial frame
$\times$	Skew-Symmetric matrix
$i,\ j,\ k$	Indexes
des	Desired
T	Transpose
0	Initial / First value
img	Image frame
C	Center frame

Acronyms

2D	Two-Dimensional
3D	Three-Dimensional
6D	Six-Dimensional
BRIEF	Binary Robust Independent Elementary Features
DWA	Dynamic-Window Approach
FAST	Features from Accelerated Segment Test
FOE	Focus Of Expansion
IMU	Inertial Measurement Unit
LSD-SLAM	Large-Scale Direct SLAM
MAV	Micro Aerial Vehicle
MPC	Model Predictive Control
MHE	Moving Horizon Estimation
OLOC	Open-Loop Optimal Control
OF	Optical Flow
OCP	Optimal Control Problem
ORB	Oriented FAST and Rotated BRIEF
PD	Proportional Derivative
PID	Proportional Integral Derivative
QP	Quadratic Program
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
SURF	Speeded Up Robust Features
TTC	Time-to-Collision

Chapter 1

Introduction

In recent years, the world has been evolving in a way that drones are becoming widespread. Navigation algorithms and control theory are the main tools to continuously advance this field of research [1], leading to innovations in multiple domains such as rescue missions [2], powerlines monitoring [3], or critical environments exploration [4]. Drones represent an important aid in various applications while, simultaneously, reduce the human risk possibly associated with a dangerous task.

1.1 Motivation

The employment of autonomous drones overcomes some limitations of human operators and automates repetitive or tedious tasks [5]. When discussing the operation of autonomous drones, the challenge of obstacle avoidance is one of the foremost concerns within the scientific community [6]. In particular, to ensure the safety of the drone during an autonomous flight, the demand for a fully risk-free system arises.

Recent technology is steadily improving video stabilization capabilities for airborne imaging purposes. However, the required computational power is still a limitation that must be taken into account. Besides the image analysis restraints, another challenge in this project is the course during flight [7]. The state-of-the-art shows a vast quantity of algorithms and implementation strategies that are capable of dealing with Simultaneous Localization and Mapping (SLAM) problems or enable drones to avoid obstacles while reaching the goal [8]. These methods are useful to solve navigation problems in two-dimensional (2D) (ground robots) and three-dimensional (3D) (aerial robots) scenarios, whereas the 3D case has a higher degree of complexity. The drone must identify the collision-free areas and travel through these to reach the goal destination. One possible strategy is the usage of a stereo camera that leads to space reconstruction and obstacle avoidance solutions [9]. The proposed approach, however,

does not do space reconstruction and it is based on attitude (IMU) and single image (monocular camera) data to perform collision avoidance and target tracking. The solution resides on a reactive system, capable of avoiding obstacles, that maintains the target on sight, preserving the distance to it.

The motivation for this work comes from the problem of obstacle avoidance, on the risks associated with the environment in an autonomous flight, and on the importance of a flawless system that can ensure new standards in drone usage. This thesis appears in partnership with Spin.Works where a possible application is to use a drone simultaneously with a rover in an unknown environment and in an exploratory context in which the drone can follow the rover, avoid obstacles, and at the same time provide visual information to the rover from a different point of view.

1.2 Problem formulation

The focal point of this thesis is to develop a real-time obstacle avoidance system for an aerial vehicle (drone) capable of following a target while avoiding obstacles. In order to have a collision-free flight, the time-to-collision is computed leading the drone to avoid the obstacles that emerge by adjusting its controls. With the purpose of obtaining a smooth flight (and maintain the target in sight), using predictive control, while the implemented system aims to achieve a collision-free solution, an optimization problem arises. Onboard sensors provide information on the state of the drone which, together with imagery data, allow the system to solve the Optimal Control Problem (OCP), as summarized in Figure 1.1.

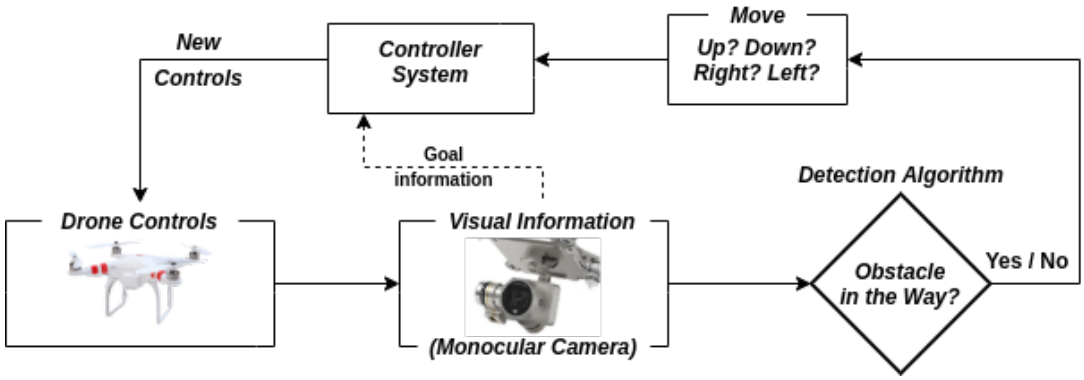


Figure 1.1: Problem overall scheme where, at each iteration, the image data is updated and analyzed providing information to the controller system.

1.3 Objectives

The goal of the present work is to develop a real-time obstacle avoidance system for an aerial vehicle. Accordingly, the following objectives are summarized:

1. Collect and analyze imagery data in real-time, identify the target, calculate the distance to it and compute the TTC.
2. Develop a control system that generates controls leading to a secure and stable flight, while following a target.
3. Evaluate the performance of the developed system in a simulation environment.

1.4 Proposed approach

The proposed approach for developing an obstacle avoidance system relies on a strategy that enables the drone to react to obstacles in a static environment. The system to be implemented will use image information from a monocular camera as a basis for decision-making combined with attitude data provided by an IMU. The inherent problem with the usage of monocular cameras is that it cannot perceive depth from the image, so, from the camera perspective, a small object near the drone moving slowly has the same representation in the image plane as a bigger object moving faster that is far from the drone. The suggested strategy pretends to overcome this problem using features extracted from the image and TTC calculation to infer where the drone must avoid going. After the time-to-collision computation, an attitude controller together with an MPC generate controls to guarantee a stable and secure path for the aircraft while following the target.

1.5 Contributions

The main contributions of this dissertation are:

- a vision algorithm capable of perceiving the environment in front of the drone and identifying the danger zones using TTC computations;
- an image reprojection implementation that enables comparisons between two consecutive frames as if they were obtained with the same roll and pitch orientation;
- an MPC algorithm capable of generating controls that can accurately track a target based on its size and position in the image;
- an MPC problem with a hyperbolic term in the cost function that generates potential fields in the drone reference frame leading to obstacle avoidance;
- a cascade control approach with an MPC controller as an outer loop to generate actuation and an attitude controller as an inner loop to guarantee flight stability;

1.6 Outline

The remainder of this document is organized as follows: Chapter 2 shows the theoretical background in quadcopter dynamics and control and information on image analysis. Chapter 3 presents the related work in the areas of obstacle avoidance, TTC calculation, and predictive control. The experimental setup, strategy and implementation are presented in Chapter 4. Verification and validation of the method are advanced in Chapter 5. The experimental results and discussion are in Chapter 6 and conclusions are in Chapter 7.

Chapter 2

Background

This chapter addresses previous knowledge required for the understanding of this dissertation. The two main areas of this thesis include the drone control and the image analysis. To control the quadrotor movement, one of the possible methods is the use of a model predictive control strategy [10]. With this approach, trajectories can be calculated based on the solution of an optimal control problem conditioned by constraints that include, for example, drone dynamics or hardware limitations. The image analysis provides information about the surroundings and the danger zones.

2.1 Drone Control

To control a drone, in an autonomous flight, it is fundamental to know and understand what are the equations that describe its movement and how the controllers actuate in the rotors speed to guarantee a stable and collision-free flight. The current section provides concepts about these two fields.

2.1.1 Drone Dynamics

Drones movement can be modeled using a rigid body model and for better understanding, it is crucial to know how to represent them in a reference frame. Let

$$\{e_1, e_2, e_3\} \tag{2.1}$$

be the three coordinate axis unit vectors, of the frame of the body ($\{B\}$) as represented in Figure 2.1, where

$$(\phi, \theta, \psi) \tag{2.2}$$

represents the rotation angles along the specific axis, also known as roll, pitch and yaw, respectively.

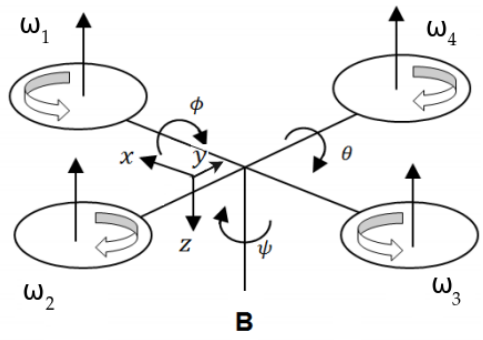


Figure 2.1: Drone coordinate system and respective rotation angles for a quadrotor configuration [11], [12].

Make $\{W\}$ an inertial frame with unit vectors

$$\{w_1, w_2, w_3\}. \tag{2.3}$$

The position and orientation of the rigid body are represented in this frame by applying a translation (r vector) and a rotation (R matrix) respectively. The rotation matrix R (or ${}^{W}R_{B}$) describes a rotation of frame $\{B\}$ in the $\{W\}$ inertial frame. Euler angles can be used to model a rotation matrix and as an example, adopting a Z-Y-X configuration will result

$$R = \begin{pmatrix} c_\psi c_\theta & c_\psi s_\theta s_\phi - s_\psi c_\phi & c_\psi s_\theta c_\phi + s_\psi s_\phi \\ s_\psi c_\theta & s_\psi s_\theta s_\phi + c_\psi c_\phi & s_\psi s_\theta c_\phi - c_\psi s_\phi \\ -s_\theta & c_\theta s_\phi & c_\theta c_\phi \end{pmatrix}, \tag{2.4}$$

where c and s are, respectively, shorthand forms for cosine and sine.

Considering m the drone mass, Ω the angular velocity of $\{B\}$ with respect to $\{W\}$ and $F, \tau \in \{B\}$ the vectors of nonconservative forces and momentums applied to the airframe, the rigid body equations of motion [13] are

$$\begin{aligned} \dot{r} &= v \\ m\ddot{r} &= mg + RF \\ \dot{R} &= R[\Omega]_\times \\ I\dot{\Omega} &= -\Omega \times I\Omega + \tau \end{aligned}, \tag{2.5}$$

where I denotes the constant inertia matrix, expressed in the body frame $\{B\}$, and $[\Omega]_\times$ the skew-

symmetric matrix

$$[\Omega]_\times = \begin{bmatrix} 0 & -\Omega_z & \Omega_y \\ \Omega_z & 0 & -\Omega_x \\ -\Omega_y & \Omega_x & 0 \end{bmatrix}. \tag{2.6}$$

2.1.2 Attitude Controller

To achieve real-time control of the drone, an attitude controller is useful to perform drone stabilization during flight [14]. One of the most common controllers is a PID (Proportional, Integrative, and Derivative) controller. This control mechanism is implemented in a feedback process with continuous error calculation to compute the commands needed to obtain the input reference, as illustrated in Figure 2.2. The equation that represents these commands is expressed as

$$u(t) = K_\mathrm{p} e(t) + K_\mathrm{i} \int_0^t e(\tau) d\tau + K_\mathrm{d} \frac{de(t)}{dt}, \tag{2.7}$$

where K_p, K_i, and K_d are the proportional, integrative, and derivative gains respectively and $e(t)$ the difference between the goal and the current values. The gains can be obtained with the Ziegler–Nichols tuning method [15] to adjust the controller response to the reference. The proportional term $P = K_\mathrm{p} e(t)$ will contribute with a direct response to the error term being more significant as the value of K_p increases. The integral term $I = K_\mathrm{i} \int_0^t e(\tau) d\tau$ responds to both magnitude and duration of the error and corrects the steady-state error over time. The derivative term $D = K_\mathrm{d} \frac{de(t)}{dt}$ is useful to improve settling time and stability of the system as the derivative component predicts the system behavior. Furthermore, besides the PID controller, the proportional, integrative and derivative components when properly combined lead to controllers such as P, PI and PD.

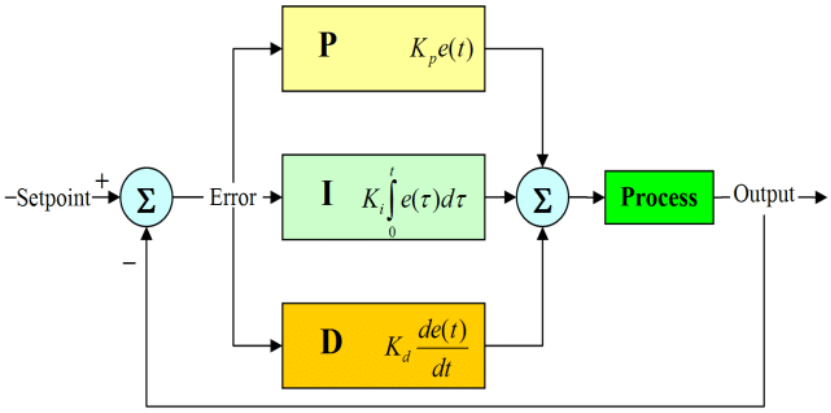


Figure 2.2: Block diagram of a PID controller.

2.1.3 Model Predictive Control

The controller design on drones can be a challenge, where the physical properties and the external forces can lead to a sub-optimal operation. Achieving a nearly perfect and efficient system is not an easy task and there are very few control methodologies that assure simultaneously the following aspects [16]:

- Physical constraint satisfaction;
- Safety;
- Optimal Performance;
- Robustness and adaptability to different operating conditions.

A promising technique that can address the previously mentioned requirements is Model Predictive Control [10]. MPC aims to optimize the system behavior and, at the same time, determine the best set of control actions in a time horizon, however, only the first actuation is used. It is an advanced technique based on a model of the mechanism that is used to control a process while satisfying a set of constraints [17]. Although classical control methods do not know physical, performance, and safety constraints, which can lead to sub-optimal plant operation, predictive control includes constraints in the design and solution optimization that leads to an efficient plant operation. In short, predictive control is a helpful tool when:

- there are constraints that condition performance;
- future knowledge can be taken into account.

In the case of this thesis, the challenge is to have an autonomous drone flight. The control problem consists in a nonlinear physical system in six-dimensions (6D, 3-position and 3-orientation) that has to hover and fly while maintaining a target in sight (on camera) and avoiding possible obstacles. In addition, the dynamics of the present apparatus can be altered by some events as shaking or the presence of external forces. The MPC controller will generate trajectories that satisfy all the necessary conditions.

MPC Formulation

MPC is a repetitive decision-making process that has the objective of aproximating the solution of an optimal control problem considering the following elements [16]:

- **Model**: in discrete-time control systems, model is represented by an equation of the form $x_k = f(x_k, u_k)$ where $x \in \mathbb{R}^n$ is the current state, $u \in \mathbb{R}^m$ is the current control and $f(\cdot, \cdot) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ the transition function that induces the successor state $x^+ \in \mathbb{R}^n$;

- **Available Information**: at any discrete time k the state x_k of the system is known whereas the control value u_k is determined;
- **Stage Constraints**: both state x_k and control u_k could be subject to constraints from, for example, physical limitations, desired operation modes and safety requirements;
- **Stage Cost Function**: performance associated to the pair state and control (x, u) is measured with stage cost function $\mathcal{C}(x_k, u_k)$.

From a theoretical point of view, there are some open-loop optimal control strategies that use an Infinite-Horizon or a Finite-Horizon. The first method, given a state $x \in \mathbb{R}^n$, selects infinite sequences of control actions $\mathbf{u}_\infty$ and controlled states $\mathbf{x}_\infty$ in order to minimize the associated stage costs

$$V_\infty(\mathbf{x}_\infty, \mathbf{u}_\infty) := \sum_{k=0}^{\infty} \mathcal{C}(x_k, u_k). \tag{2.8}$$

From the Infinite-Horizon method truncation is obtained the Finite-Horizon approach (the MPC uses a Finite-Horizon OCP to obtain an approximation of an Infinite-Horizon OCP). Equation (2.8) is rewritten as

$$V_N(\mathbf{x}_N, \mathbf{u}_{N-1}) := \sum_{k=0}^{N-1} \mathcal{C}(x_k, u_k). \tag{2.9}$$

In short, MPC problems are defined by a cost function that has to be minimized, an internal system model to predict system behavior, and constraints [18]. The following optimization problem exemplifies the situation

$$\begin{aligned} \underset{\mathbf{x},\mathbf{u}}{\text{minimize}} \quad & \sum_{k=0}^{N-1} \mathcal{C}(x_k, u_k) + \mathcal{C}(x_N) \\ \text{subject to} \quad & x_{k+1} = f_k(x_k, u_k), \quad k = 0, ..., N-1 \\ & g_k(x_k, u_k) = 0, \quad k = 0, ..., N-1 \\ & h_k(x_k, u_k) \leq 0, \quad k = 0, ..., N-1 \\ & g_N(x_N) = 0, \end{aligned} \tag{2.10}$$

where *N* is the horizon length, $x_0, ..., x_N$ and $u_0, ..., u_N$ represent the evolution of state and control inputs respectively, and g and h represent constraints that restrict the set of states for which the optimization problem is feasible. The advanced control framework shown needs to measure the current state to find the optimal input sequence, for the entire horizon, over time, as represented in Figure 2.3.

The usage of an MPC approach to control the drone trajectory will result in the overall system illustrated in Figure 2.4, where u_0 represents the first control action and the only that has to be implemented in the present time sample.

Concluding, MPC relies on a model, and in real-world situations, multiple disturbances such as wind or rain change the evolution of the system leading to sub-optimal behavior and preventing it from

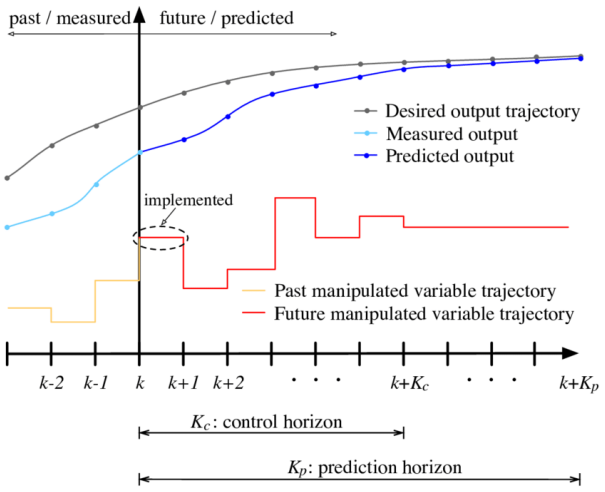


Figure 2.3: MPC horizon scheme with the computed set of controls (red), the predicted output (blue) and the desired output (black) [19].

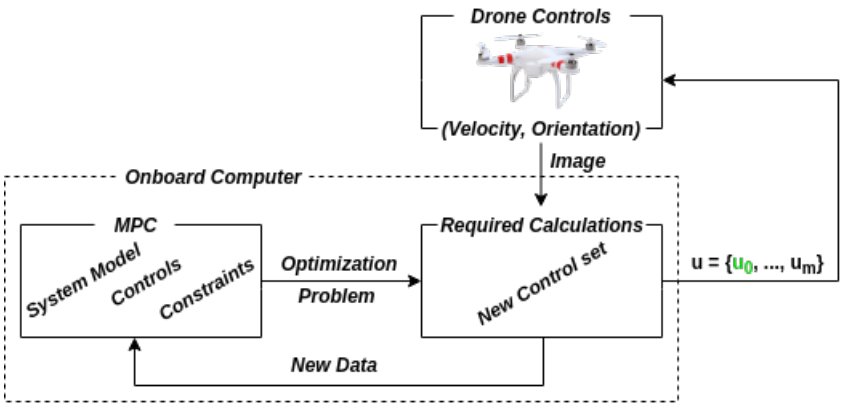


Figure 2.4: Generation of a new set of controls u while obtaining new image information from the camera and updating MPC optimization problem.

converging to the calculated solution. Even though noise models can be incorporated in the MPC formulation, solving the resulting optimal control problem is not straightforward.

ACADO Toolkit

ACADO Toolkit is a framework with multiple algorithms for automatic control and dynamic optimization [20] [21]. It is designed to implement MPC [22] and Moving Horizon Estimation (MHE) problems [23]. The toolkit is free, open-source, and has external tools that guarantee efficiency and functionality. One of the tools is ACADO Code Generation which exports C-code set for MPC problems. This tool allows to obtain implementations in C/C++ code from MATLAB.

2.2 Image Analysis

To avoid obstacles the implemented system must realize, from image data, if the path is obstructed. Accordingly, TTC calculations enable the system to infer if there are dangerous regions and where are they in front of the drone. Therefore, it is crucial to understand how the representation of the world is expressed in an image and how information can be extracted.

2.2.1 Camera model

A monocular camera will be used to obtain all the visual information. This camera, designed for small quadrotors, can be represented by a pinhole camera model as displayed in Figure 2.5, where $\mathbf{X} =$

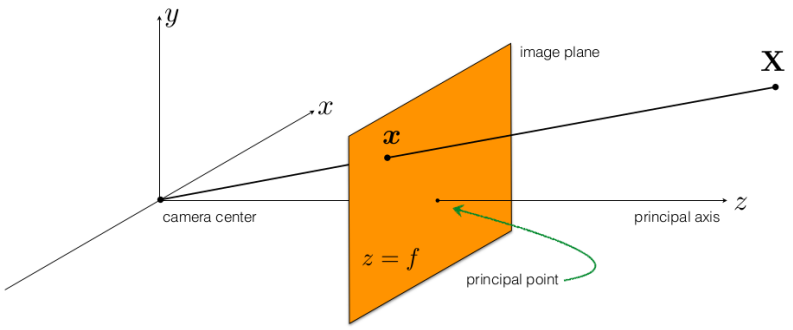


Figure 2.5: Pinhole camera model with an arbitrary point in space **X** [24].

$[X\ Y\ Z]^T$ represents a point in space, $\mathbf{x} = [x\ y]^T$ a point in the image plane and *f* the focal length. Basic equations of the model can be written as follows [24]:

$$x = f\frac{X}{Z} \quad , \qquad y = f\frac{Y}{Z} \ . \tag{2.11}$$

To estimate the distance between the camera and the object, a depth camera or image analysis (from multiple cameras) can be used to solve the problem. Mathematically speaking that corresponds to calculating the euclidean norm of $\mathbf{X}$. In a practical sense, the object image size and its variation allow the algorithm to know if the drone is getting closer or furthest from the obstacle. To extract this information, the camera model must be known and for that, it is derived with the following information [25]:

- **intrinsic parameters**: focal length *f*, scale factors (s_x,s_y) and principal point (c_x,c_y) that leads to the intrinsic matrix K;
- **extrinsic parameters**: world frame located at an arbitrary position - conversion for the camera coordinate frame (using rotation R and translation T matrices).

The intrinsic parameters constitute the internal model responsible for converting metric coordinates (x, y) to pixels (x', y'), as

$$x' = Kx \quad \Longrightarrow \quad \begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} fs_x & 0 & c_x \\ 0 & fs_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}. \tag{2.12}$$

The full camera model can finally be written as in equation (2.13):

$$x' = K \, [R_{[3\times3]}|T_{[3\times1]}] \, X. \tag{2.13}$$

Since the camera intrinsic parameters, K, and the target dimensions are known, it is possible to obtain the target position, relative to a non-inertial frame like the camera frame. An illustrative example is represented in Figure 2.6, where Y is the target real dimension.

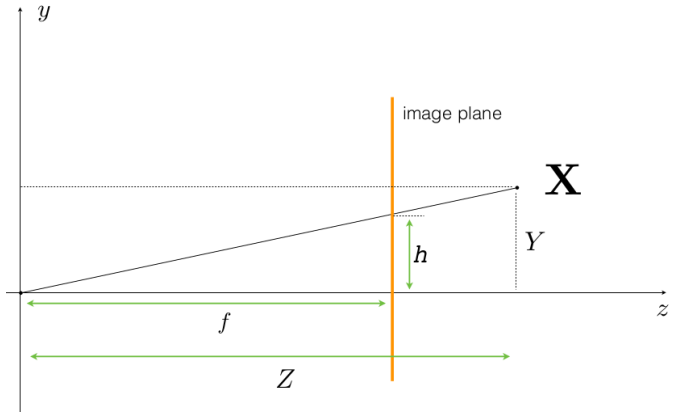


Figure 2.6: Relation of a point in the image plane to a point in space [24].

From observation is easy to conclude that the distance through the obstacle, Z, is obtained from the similarity of triangles

$$Z = \frac{Y}{h} \, f, \tag{2.14}$$

where the denominator h is obtained from the target dimension in the image plane.

2.2.2 Features Detection

After the information on the mapping from the image plane to world coordinates, it is also essential for the strategy to have in mind how to capture important points from the image. For this purpose, there are features detection strategies that extract interest points[1] from visual information [26] and facilitate their analysis.

[1] Points that highlight differences of texture in the image

In order to perform image analysis in real-time, two techniques of feature extraction are relevant from the literature due to the time of computation: ORB [27] and A-KAZE [28]. ORB features are obtained by brightness comparisons between neighbors pixels and from the analysis of a binary feature vector (generated by all the vectors that describe the FAST keypoints), whereas A-KAZE detector is a fast multiscale feature detection approach that detects features in a nonlinear scale-space [29].

2.2.3 Optical Flow and Motion Field

In image sequences, each point moves due to the object movement or due to the camera motion. The relative movement of objects generates motion on the image plane, and the apparent movement of brightness patterns between two (or more) consecutive frames is called optical flow (OF) [25]. This flow can be expressed by the equation (2.15) where $I(x, y, t)$ represents the intensity at position (x, y) in the tth time step.

$$I_x u + I_y v + I_t = 0, \quad \text{where} \quad I_x = \frac{\partial I}{\partial x},\ I_y = \frac{\partial I}{\partial y},\ I_t = \frac{\partial I}{\partial t}. \tag{2.15}$$

Besides OF, there is a similar concept to percept motion that is the motion field. It is characterized by the movement in a 3D scene projected into the image plane, usually represented as well by a two-dimensional vector (u, v). Motion field can be used for various applications and is expressed by the following equation

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \frac{1}{Z} \begin{bmatrix} -1 & 0 & x \\ 0 & -1 & y \end{bmatrix} \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} + \begin{bmatrix} xy & -(1+x^2) & y \\ (1+y^2) & -xy & -x \end{bmatrix} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}, \tag{2.16}$$

where the first term has information about the scene and corresponds to the translation part with t the linear velocity of the object and the second term only depends on the camera parameters and represents the rotation part of the movement with ω the angular velocity of the object.

2.2.4 Time-to-collision

Time-to-collision, as the name suggests, is the time left for a collision to happen. It expresses a value, associated with each pixel in the image (and consequently a point in space), that represents the time interval between the current instant and a collision with an obstacle at that pixel. However, this collision only happens if the drone is flying towards the obstacle or near them depending on the drone dimensions and the position of it. In the image, the point that represents where the drone is going in the 3D space is called Focus of Expansion (FOE), as illustrated in Figure 2.7.

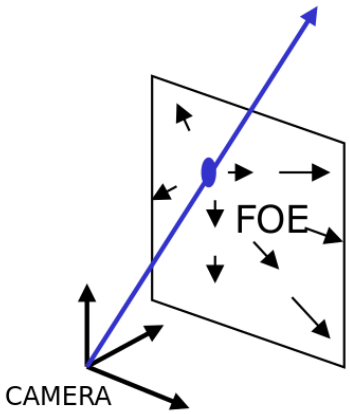


Figure 2.7: FOE illustration in the image plane, where this point is the one that is closest to all the resulting lines from the direction vectors that were originated by the feature matching [25].

In a static environment, the displacement of features between frames will increase with the distance to the estimated FOE as represented by the arrows length in Figure 2.7. With the estimation of the FOE together with the pixel movement between frames ($\|(u, v)\|$), it is possible to obtain an expression that computes TTC for a pixel k[2] [30], in a given frame, represented in equation (2.17)

$$TTC_k = \frac{\|x_k - FOE\|}{\|(u_k, v_k)\|}, \tag{2.17}$$

where x_k represents the 2D location of pixel k and $\|(u_k, v_k)\|$ gives the displacement of pixel k from the previous frame to the current frame. The numerator, $\|x_k - FOE\|$, provides information in pixels, the denominator represents the movement of pixels between frames and has information in pixels per second (time between frames) leading the TTC to have results in seconds.

[2]in [30] TTC is referred as depth information of a pixel

Chapter 3

Related Work

Autonomous flight is an emerging area that is gaining popularity over the years [31]. Infrastructure inspection, scientific data collection, field surveying, and security monitoring are some of the applications that are served nowadays by an autonomous drone [32]. Other fields that include disaster areas inspection and the creation of systems that help to manage these delicate situations are now an objective [33] and are represented in Figure 3.1. A particular point of interest, in an autonomous flight, is the obstacle avoidance problem. This issue requires both sensing and control strategies to be developed. The sensors deal with surroundings inspection, identification of objects that represent a threat in the path, and are also responsible for measuring the state of the drone. The controls of the drone must be generated accordingly with the sensor data and the finality of the situation. Moreover, controls generation often relies in solving an optimization problem [34].

(a) Supplies transportation.

(b) Mapping camp fire disasters.

Figure 3.1: Examples of drone usages.

Despite the recent technological developments, current software and hardware have some limitations [35]. The different image acquisition systems or visual sensors [36] [37] provide numerous information, nonetheless, they have some limitations such as resolution, the field-of-view, or frame rate. From the sensing devices, when an autonomous flight is performed, there is a large amount of data that has to be

processed and analyzed [38]. The drone itself has velocity and turning radius constraints that have to be taken into account when projecting an obstacle avoidance system. In order to deal with the adversities above mentioned, researchers developed several approaches to obtain an autonomous system [39].

3.1 Methods for collision avoidance

There are a few methods that can be used to perform autonomous flights, however, these differ in some aspects, such as computational power or external data needed. Approaches for solving a collision problem are based on external sensors where, for example, image information is received from a stereo camera allowing to extract depth information [9] or multiple sensor data is fused to detect obstacles [40]. Besides these classical procedures, there are learning proposals, where a controller learns the best action in each particular situation [41].

This section presents methods that deal with the obstacle avoidance problem. Brief explanations and representative ideas succinctly explain the state-of-the-art of the topic.

3.1.1 Guidance methods

The strategies usually implemented to guide an aerial vehicle during flight can be separated in the following subgroups [36]:

- Map-based or Map-building systems (ex: SLAM-based) as indoor SLAM with multiple sensors [42], [43], and monocular SLAM [44];
- Mapless systems that use imagery data and learning methods [45] or strategies that use an optical flow approach for obstacle avoidance as in [46] and [47].

An important element to perform safety navigation is the obstacle avoidance capability [48]. Although many of the obstacle avoidance methods rely on space reconstruction, there are some that do not use full space information. Figure 3.2 is representative of the two subgroups mentioned.

3.1.2 Space reconstruction methods

To integrate surroundings information, in the planning algorithm, a stereo camera can be used as in [49]. The authors suggest an approach named Stereo Large-Scale Direct SLAM (LSD-SLAM) that uses high-contrast pixels depth such as corners, edges and high texture areas to build a map. The method starts by tracking the motion of the camera to build a reference keyframe in the map. Then, the depth in

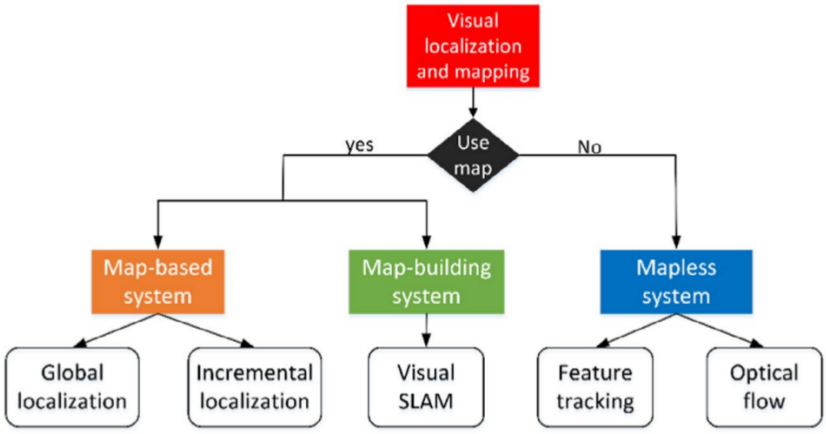


Figure 3.2: Visual localization and mapping systems [36].

the current keyframe is estimated from stereo correspondences based on the tracked motion. With this distance information, it is possible to create a map and navigate through a collision-free path.

In [50] a SLAM strategy was proposed using only a monocular camera. A map is estimated based on the obtained images and small-baseline stereo comparisons that are useful to obtain a measure of depth. The authors show that sufficient translational camera movement, as initialization, is enough to ensure convergence of the algorithm in a correct depth configuration. They also recommend an approach to solve the inherently scale problem of Monocular SLAM based on the correlation between scene depth information and tracking accuracy of the camera along time.

To recover the 3D structure of a scene[1], a set of images from different known perspectives may be used [51]. This concept relies on feature matching and precise pose estimation of the camera along time. It is often used to reconstruct some particular objects as exemplified in [52].

3.1.3 Time-to-collision

The existing vision systems in drones allow perceiving information that is in front of the vehicle. Knowing the velocity and the frame rate, due to the increase of the object scale in the image plane, the TTC can be inferred as shown in [53]. This time can be calculated based on models that rely on features and optical flow information.

A common situation is when obstacles are in front of the vehicle. The problem of detecting these is tackled in [53]. The described method uses a TTC procedure based on the relative size difference of the object in a sequence of images. Speeded Up Robust Features (SURF) features [54] are used to find features in the image that represent small areas that are getting closer. The SURF algorithm matches key points in consecutive images and compares the evolution of the surrounding regions, that is, compares region sizes between images and recognizes if an obstacle is getting closer. The SURF

[1] Structure from motion problem

matches where the respective region does not increase are discarded. The system in [53] can avoid frontal obstacles with the condition that the obstacles must have enough texture for the SURF algorithm to consider image points as key points.

A TTC approach is a procedure that deals with the problem of obstacle avoidance and combining this with OF analysis, as in [30], leads to a solution. This strategy explicitly calculates TTC as

$$TTC = \frac{d}{d'}, \tag{3.1}$$

where d represents the distance between the pixel and the FOE and d' represents the norm of the optical flow vector in the pixel location. The authors suggest using TTC and OF to surpass drone vibrations difficulties leading to the following policies:

- using a Kalman filter[2] to reduce noisy data;

- OF method improved with a balanced strategy and TTC to estimate the distance to the obstacles.

The balanced strategy starts by dividing the image into 9 regions (3x3 grid). In the four ”corners”, the OF is computed to give a reference to reduce the quadcopter vibration. In the extra five regions, the OF is computed to track movements. Results have shown that this method improved drone stability when compared with using only an OF approach.

In [56] a plan for predicting collision candidates using optical flow to interpret objects is introduced. Since calculating dense optical flow[3] results in a large computational effort, this work focus on sparse optical flow[4]. Flow vectors are obtained and point in the direction of the focus of expansion in the image. Then, the respective TTC for every object is calculated based on the distance from the camera to the object (λ), the distance covered between frames ($\Delta\lambda$) and the time to acquire two consecutive images (Δt):

$$TTC = \frac{\lambda}{\Delta\lambda}\,\Delta t. \tag{3.2}$$

Measuring TTC is useful to construct motion reflexes for collision avoidance. The authors in [57] propose a method to measure TTC denominated by Active Contour Affine Scale (ACAS). In this approach, TTC is computed based on tracking active contours and on the quotient between the scale of the object in the image (σ) and the time derivative of this scale ($d\sigma/dt$).

[2] ”Technique for filtering and prediction in linear systems” [55]
[3] Dense optical flow: the relative motion of every pixel in a single image has to be estimated
[4] This technique only processes some pixels of the image

3.1.4 Deep Learning

Besides all the existing traditional strategies, deep learning approaches are also used in drone flights. For example, methods that learn how to mimic an expert pilot's action, as shown in [58]. In this work, the only sensor needed is a camera. A Convolutional Neural Network (CNN) is used to train a classifier with a custom dataset. This classifier consistently receives visual input from the quadcopter camera and returns a flight command as if performed by a pilot. The training strategy used allows the classifier to learn the most efficient controls to minimize possible failures.

A model-based reinforcement learning approach is proposed in [59] where is used a deep neural network for determining a collision prediction model. This collision model is learned by making low-speed collisions in both of the tested vehicles (drone and RC car). The method used provides a control strategy that knows maneuvers to avoid obstacles.

Both [58] and [59] showed promising results, however, the testing environments were simple, thus behavior in more dynamic and unstable situations is yet to be tested and validated.

3.1.5 Path Planning and Trajectory Tracking

In drone navigation, one important task is path planning, i.e., the means of finding a path between a starting point and a final point. Another significant feature is trajectory tracking, that is the process of following a determined trajectory at a given time frame.

To delineate a path, depending on the situation, is whether possible to perform path planning on a global scale or locally [36]. In global path planning, it requires a global map (or static map), start location, and target location to the planner to calculate an initial path. Some heuristic methods and learning algorithms are also used in global path planning, for example, the A-star algorithm [60], or Rapidly-exploring random trees [61]. On the other hand, local planers use the environment information near the drone and its state estimation to plan a path without collisions. A traditional approach in local planning designs is the artificial potential field method. This method was explored in [62] and was referred to as a strategy that creates virtual forces in the workspace of the drone. The apparatus is attracted to the target point and repelled from obstacles leading to the goal point. One limitation stated and solved in [62] is the deadlock problem, where the vehicle has symmetrical forces acting on it. To solve this issue, a random small rotation is added to the attractive force leading to two misaligned forces and breaking the deadlock problem.

In addition, a strategy based on path planning and trajectory tracking is used in [9] and starts with the following:

- a regular 3D grid is used to represent the environment, in the form of an occupancy map, and a

roadmap graph is built by randomly sampling points (maximum one per voxel) and attaching them to the respective neighboring points (a cost is associated to each connection);

- initial and final points are added to the roadmap and the optimal path is generated through the least cost path.

As the drone moves, obstacles near the generated points may appear. In this case, the algorithm modifies the costs associated with the respective connections to those points leading the drone to change its trajectory. The costs tend to infinity to prevent the drone from going to these locations. Figure 3.3 illustrates how the roadmap and path are updated when a voxel in the occupancy map becomes occupied.

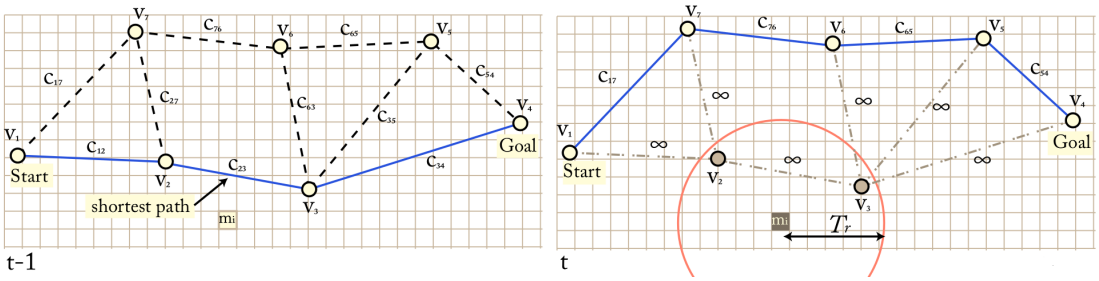


Figure 3.3: 2D representation of how the roadmap and path are updated from time step $t-1$ to t when a voxel (m_i) is occupied [9].

In the survey [63], guidance aspects such as trajectory or motion planning were analyzed and practical examples shown. One of the mentioned methods was the Dynamic-Window Approach (DWA). It is a navigation scheme developed in [64] and consists of two main components. The first is to generate a valid search space and the second is to select the optimal solution in that space. The search space was formed by circular trajectories that were free from collisions. The optimization aims to select a heading[5] and velocity that leads the robot to the goal with maximum clearance from any obstacle.

3.1.6 Gap Detection

An interesting and related topic is gap detection. In [65] was proposed a framework for a quadrotor to fly by unknown gaps only using onboard sensing and a monocular camera. The key contributions in this work are: a Temporally Stacked Spatial Parallax (TS^2P) gap detection algorithm and visual servoing[6] on a contour to follow it in the image plane enabling the quadrotor to fly safely into unknown gaps. First, a gap is detected using the TS^2P algorithm, then the other side of the gap is recognized, the background region, and also the foreground region is identified with some uncertainty associated, as illustrated in Figure 3.4. Visual information allows the drone to cross the gap because it guarantees the correct orientation when entering the hole. For the quadrotor to go through the hole safely, this approach

[5] Alignment of the robot with the target
[6] also known as vision-based robot control

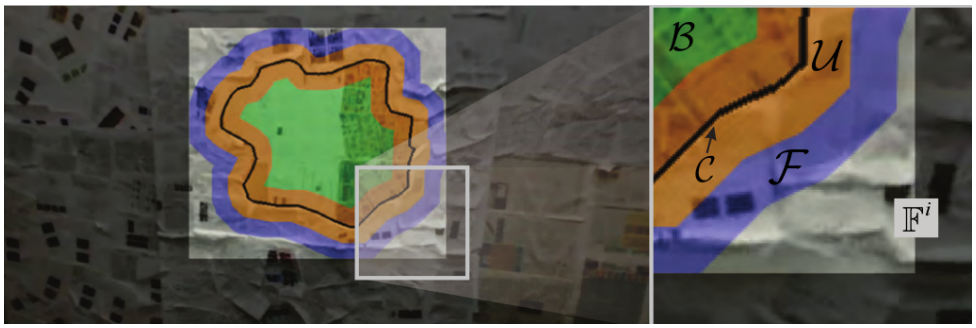


Figure 3.4: Visual representation of the different sets on a sample image (blue: foreground region, green: background region, orange: uncertainty region, black line: contour, brighter part of frame: active region, and darker part of frame: inactive region) [65]

assumes that the gap: is all on the same plane, is perpendicular to the movement of the drone, and is large enough. However, this last assumption ends up being a necessary condition because the algorithm will not be able to recognize a hole smaller than the drone. Another limitation is the background and foreground colors that cannot be equal at the expense of the gap not being identified.

3.1.7 Model Predictive Control

As seen in Section 2.1.3, Model Predictive Control is a technique that can deal with constraints and can be useful even when system dynamics are nonlinear. This approach raises two trajectory tracking controllers known as Linear Model Predictive Control (LMPC) and Nonlinear Model Predictive Control (NMPC). In the search of the most precise trajectory tracking a comparison between LMPC and NMPC was detailed in [66]. Both LMPC and NMPC solve an OCP. To implement an LMPC strategy, a Quadratic Program (QP) solver is adopted using CVX-GEN, whereas in the case of the nonlinear problem the solver used is generated using ACADO toolkit. Comparisons of the two controllers extend into:

- disturbance rejection capability;
- step response;
- tracking performance;
- computational effort.

The tests were made with and without external disturbances (wind). In the absence of external forces acting on the drone, the performance of both controllers is very comparable. However, in the presence of external wind disturbances, the NMPC showed slightly better results in the referred situations except in the tracking of an aggressive maneuver where the NMPC was significant superior.

MPC is an effective method, although, at the same time, it can be computationally demanding. In [67], it is suggested a combination of MPC with reinforcement learning, where the MPC together with

offline trajectory optimization is used to generate data at training time. The created data is used to train a deep neural network, and after the training phase, the neural network can successfully control the quadrotor. The result is a system with a low computational cost when compared with the traditional MPC.

Chapter 4

Implementation

Through this chapter, the grounds for solving autonomous flight problems are presented. In particular, image analysis leading to dangerous regions identification and drone control implementations resulting in stable flights contribute to developing a real-time obstacle avoidance system.

4.1 Implemented strategy

The objective resides in implementing an algorithm capable of avoiding obstacles while following a target. The strategy adopted in the present work consists on using a cascade control approach with an MPC controller as an outer loop to generate optimal trajectories and an attitude controller as an inner loop to guarantee flight stability and tracking of the generated trajectory. The determination of unsafe regions is performed by TTC estimation and, when TTC is relatively small, the MPC controller will generate trajectories in a way that the drone will avoid the danger zone while keeping track of the object. A scheme in Figure 4.1 illustrates the proposed architecture.

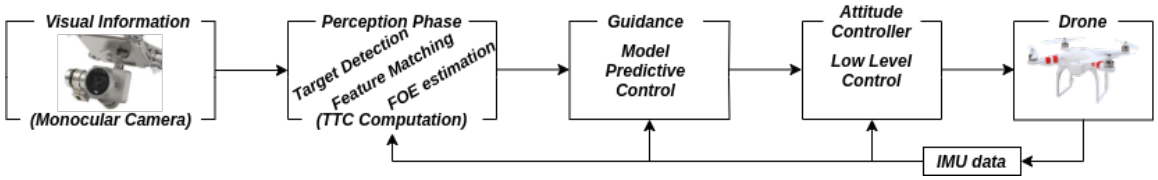


Figure 4.1: Architecture in the autonomous flight problem with multiple layers that go from analyzing images to implementing controls in the drone in order to perform a safe flight.

At the beginning of the process, the camera is responsible for collecting images to be analyzed. Then, in the perception phase, the target is identified, and TTC computed. In this stage all the imagery analysis is performed, such as feature extraction, feature matching, and Focus of Expansion (FOE) estimation. The guidance section is accomplished using an MPC controller, where the solution of an

optimization problem leads to optimal trajectories generated at each time step. Finally, the attitude controller generates low-level controls guaranteeing stability and trajectory tracking.

4.2 Experimental Setup

The system consists of a drone (*Pelican* - *AscTec*[1]) with an IMU and a camera rigidly attached to it aligned with the arm of the red propeller (x axis in '+' configuration, Figure 2.1) as in Figure 4.2.

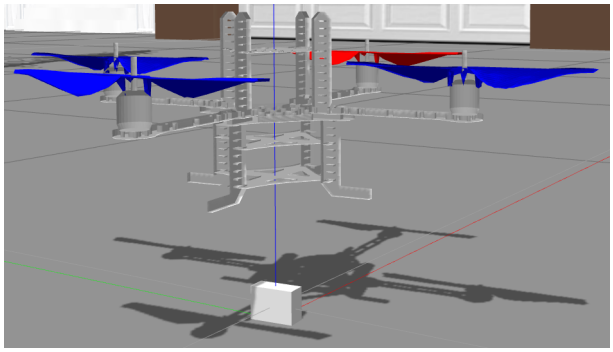


Figure 4.2: Drone (*Pelican* - *AscTec*) in Gazebo environment.

The working environment, as mentioned before, takes place in a simulation world in Gazebo using the Micro Aerial Vehicle (MAV) framework RotorS that provides the drone with all the dynamics and physical properties included.

4.3 Image Analysis

The image analysis corresponds to a crucial sector in the solution because it is the interpretation of all the information perceived from the environment. In this section are presented the strategies that go from target detection to the TTC computations.

4.3.1 Target Identification

Target identification through image analysis involves computational stages that go beyond the main scope of the present thesis. Hence, a simple way to solve this step is to impose to the target a specific color and avoid using a test environment that contains the same color (in this case red). OpenCV library has already implemented tools to extract shape, area, and center of the target. Algorithm 1 represents a pseudocode example of the steps performed to find the target in the image.

[1] http://wiki.ros.org/Robots/AscTec

Algorithm 1 Target Identification

```
1: procedure GETTARGETINFO(image)                ▷ Subscribes to image topic
2:     while (image == True) do
3:         RedFilter(image)                      ▷ Returns RedOnlyImage
4:         FindContours(FilteredImage)
5:         if (Contours ≠ 0) then
6:             CalculateArea(Contour)
7:             CalculateCenter(Contour)
8:     return Target Area, Target Center
```

4.3.2 Obstacles from image based on TTC calculations

The strategy will not identify the obstacles in front of the drone, but instead will use TTC estimations to recognize the danger zones. As indicated in Figure 4.1, for obtaining TTC in the image it is mandatory to perform feature matching and FOE estimation. For this part of the project, multiple online tools from the OpenCV library were used.

Feature detection

First, a region of interest (ROI)[2] is extracted from the image (376x240 px). Second, a feature detection algorithm from the OpenCV repository is used. In Section 2.2.2, two features detection methods were mentioned, and the ORB features were considered favorable due to the lower computational demand and by the fact that the algorithm in OpenCV (*cv::ORB*) ranks up the best features based on the Harris Score[3]. The region of the target is excluded from the image, in this phase, because the goal is to obtain the drone direction of movement, using the FOE estimation, and the target motion will alter this computation. Later, when the next frame is obtained, it is necessary to perform feature matching. However, in many occasions, the camera will not have the same orientation, in consecutive frames, and this leads to inaccurate results in the movement of features. For solving this problem a strategy that reprojects the features in a frame with a fixed roll and pitch orientation was implemented.

Features reprojection

Feature extraction provides crucial information for the vision stage of the thesis. An identified problem in the feature matching step is the orientation of the camera. The roll and pitch of the camera are the same as the drone, thus during the flight, the camera will point out in different directions, as illustrated in Figure 4.3.

[2] the ROI is centered in the image plane (752x480px)
[3] Harris algorithm rank features and distinguishes between edges from corners

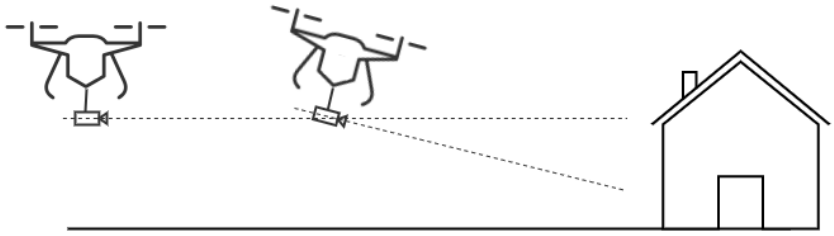
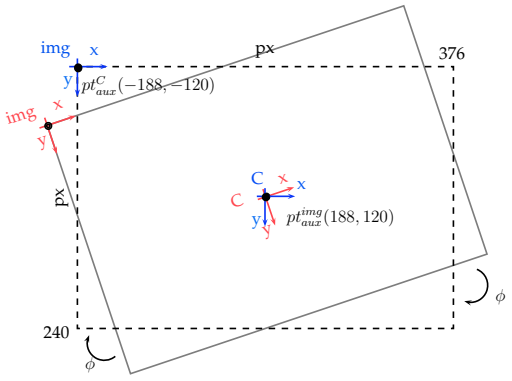


Figure 4.3: Camera attitude variation example during flight.

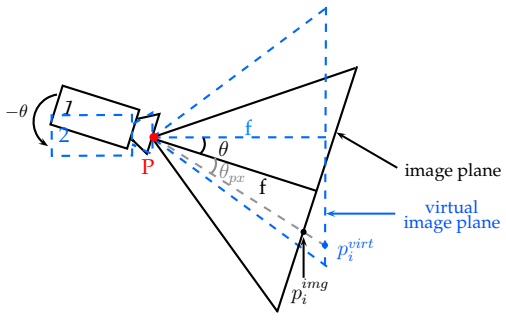
In Figure 4.3, consider that an image is obtained in each drone position. To guarantee that the differences in both images are due to translation motion only, it is necessary to compare them as if both image planes were parallel and perpendicular to the ground. For that, and in the absence of a camera stabilizer, a two-step virtual rotation in the camera is realized leading to an image, in the second situation, as if the camera heading is parallel to the ground ($c_{cam}^{\parallel}$):

$$c_{cam} = \begin{bmatrix} camera_{roll} \\ camera_{pitch} \end{bmatrix} \quad \text{and} \quad c_{cam}^{\parallel} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}. \tag{4.1}$$

The two-step rotation is based on the values of the IMU of the drone. The first rotation is along the optical axis with an angle $roll_{IMU}$ and the second rotation is along the horizontal line that crosses the camera center (point P in Figure 4.4(b)) and is parallel to the image plane with an angle $pitch_{IMU}$. Consequently, these two rotations result in a translation of the pixels x_i in the image plane and are represented in Figure 4.4.



(a) Image plane before and after a roll rotation of ϕ with frames $\{img\}$ and $\{C\}$ and auxiliary points in each frame respectively: pt_{aux}^{C} and pt_{aux}^{img}.



(b) Pitch rotation of $-\theta$ along the horizontal line in the image plane that crosses the middle point P.

Figure 4.4: Example rotation R_{roll} (a) and R_{pitch} (b) in the image plane.

With the information of Figure 4.4(a) the projected points in a virtual plane can now be expressed by

$$x_i^{proj} = R_{roll}(x_i - pt_{aux}^{img}) - pt_{aux}^{C} \quad \text{with} \quad R_\alpha = \begin{bmatrix} cos(\alpha) & sin(\alpha) \\ -sin(\alpha) & cos(\alpha) \end{bmatrix}, \tag{4.2}$$

with pt_{aux}^{img} the image center in the $\{img\}$ frame and pt_{aux}^{C} the top-left corner of the image in the $\{C\}$ frame. The camera rotation in Figure 4.4(b) will alter the y component of the pixels x_i^{proj} in equation (4.2). Therefore, the projected points in the virtual plane, from the information in Figure 4.4, are expressed as

$$x_i^{virt} = \begin{bmatrix} x_i^{proj}(1) \\ h_{img}/2 + f\ tan(\theta_{IMU} + \theta_{px}^{img}) \end{bmatrix}, \tag{4.3}$$

where $x_i^{proj}(1)$ refers to the first coordinate of the projected point x_i^{proj}, h_{img} is the image height, i denotes the feature i, f is the focal length and θ_{px}^{img} is given by

$$\theta_{px}^{img} = tan^{-1}\left(\frac{x_i^{proj}(2) - h_{img}/2}{f}\right). \tag{4.4}$$

Feature matching

After the feature detection and reprojection, it is necessary to perform feature matching. The selected ORB features, with the best Harris Score, are matched with the features selected in the previous frame by brute-force, this means that the match only occurs if the feature i_A in frame A as the best match with the feature i_B in frame B and vice-versa. The matches where the feature did not move or where the movement was larger than a certain threshold were discarded from the next steps and it was because the features that remained in the same position were probably from a point far from the drone and the features that had a considerable movement were presumably due to an incorrect match.

FOE calculation

The FOE calculation consists of a sequence of operations. After the feature matching stage (between two consecutive frames) it is necessary to compute the lines that intersect each pair of features in the image plane. First, the direction vector n that represents the translation of the feature between frames is obtained as

$$n = \frac{x_i^j - x_i^{j-1}}{\|x_i^j - x_i^{j-1}\|} \qquad \text{with} \qquad \|n\| = n^T n = 1, \tag{4.5}$$

where x_i^j represents the position in the image of the feature i in frame j. The lines appear almost immediately from the simple equation: $x_i(2) = mx_i(1) + b$ [4], where the two unknowns (m and b) can be easily obtained by replacing $x_i(1)$ and $x_i(2)$ by a couple of matched features, leading to a system of two equations and two unknowns. In vector form, the equation of a line is given by:

$$z = a + tn, \tag{4.6}$$

[4] $x_i(1)$ and $x_i(2)$ corresponds to the 1st and 2nd coordinate of x respectively

where a is the position of a point on the line. As the scalar t alters, z returns the locus[5] of the line. The distance of an arbitrary point p to this line is:

$$\text{distance } (z = a + tn, p) = \|(a - p) - ((a - p) \cdot n)n\| \tag{4.7}$$

Which can be confirmed by inspection of Figure 4.5:

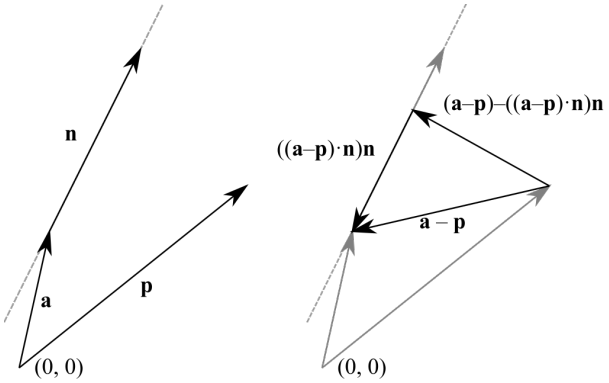


Figure 4.5: Vector formulation of the distance from a point to a line [68].

The distance of a point to a line is important to the problem at hand because the FOE will correspond to the minimum distance of a point to the multiple lines resulting from the pairs of features. For that, following [69], it is necessary to compute the square of the distance in equation (4.7):

$$\begin{aligned} D(p; a, n) &= \left\|(a - p) - \left((a - p)^T n\right) n\right\|^2 \\ &= \left\|(a - p) - nn^T(a - p)\right\|^2 \\ &= \left\|\left(I - nn^T\right)(a - p)\right\|^2 \\ &= (a - p)^T \left(I - nn^T\right)\left(I - nn^T\right)^T (a - p) \\ &= (a - p)^T \left(I - nn^T\right)(a - p) \end{aligned} \tag{4.8}$$

When considering multiple lines, the equation (4.8) is represented as

$$D(p; A, N) = \sum_{j=1}^{K} D\left(p; a_j, n_j\right) = \sum_{j=1}^{K} \left(a_j - p\right)^T \left(I - n_j n_j^T\right)\left(a_j - p\right). \tag{4.9}$$

To recover the point that is closer to all the *K* lines it is required to solve an optimization problem in the form[6]:

$$\widehat{p} = \min_{p} D(p; A, N), \tag{4.10}$$

[5] set of all points (commonly, a line or curve), whose location satisfies one or more specified conditions
[6] Remembering that minimize distance or squared distance will lead to the same global solution

which is quadratic in p. Taking the derivatives with respect to p it is easily obtained:

$$\frac{\partial D}{\partial p} = \sum_{j=1}^{K} -2\left(I - n_j n_j^T\right)(a_j - p) = 0. \tag{4.11}$$

Rearranging the previous equation leads to the system of equations:

$$\begin{matrix} Rp = q \\ R = \sum_{j=1}^{K}\left(I - n_j n_j^T\right) \quad , \quad q = \sum_{j=1}^{K}\left(I - n_j n_j^T\right) a_j \end{matrix}. \tag{4.12}$$

For obtaining point p, that corresponds to the FOE, least squares method is applied resulting in

$$FOE = \widehat{p} = (R^T R)^{-1} R^T q. \tag{4.13}$$

However, during flight, FOE estimation is relatively unstable leading to imprecise TTC calculations [30]. To accomplish stability, the moving mean method

$$FOE^j = \frac{FOE^j + FOE^{j-1} + \ldots + FOE^{j-N+1}}{N}, \tag{4.14}$$

was used to reduce high-frequency noise, where were considered the last N FOE points. Besides this strategy, a variation larger than 100 pixels from the average FOE is also filtered.

TTC calculation

Finally, TTC is calculated according to equation (4.15) [30]

$$TTC_{feature_i^j} = \frac{\|x_i^j - FOE^j\|}{\|x_i^j - x_i^{j-1}\|}\, t_r, \tag{4.15}$$

where the numerator refers to the distance between feature i and FOE (in the frame j), and the denominator represents the distance between the match of feature i in consecutive frames (as in equation (2.17)). The t_r factor is the time difference between the two consecutive frames j and $j-1$. It is indeed confirmed, from equation (4.15), that a feature moving far from the FOE represents a minimum threat (TTC large). Otherwise, features moving nearby the FOE describe a possibly dangerous situation.

Danger zones

Instead of looking for each feature individually, the proposed approach will attend to groups of features and identify the danger zones in the image. The image zone division is represented in Figure 4.6.

The algorithm identifies the zones in front of the drone that have a small TTC and drives the MPC to

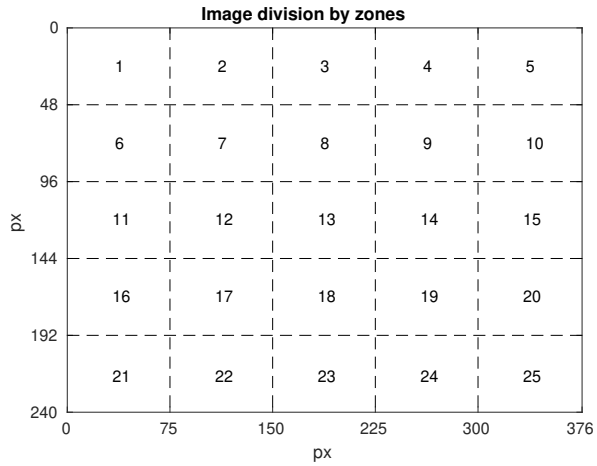


Figure 4.6: Image division in areas of 3600 px.

generate paths that avoid those specific zones. The TTC in each of the zones is given by the average TTC of the selected features in that zone

$$TTC_{\text{zone}i} = \frac{1}{k_i} \sum_{n=1}^{k_i} TTC_{feature_n}, \tag{4.16}$$

where k_i is the number of keypoints in zone i and with the condition of only the zones with 3 or more marked features have an associated TTC. Since the vision method is responsible for detecting close obstacles, the features that barely move between frames will have a higher TTC. Hence, are considered background and are not included in the average time-to-collision of each zone. Another consideration in the TTC per zone strategy is that only the zones with three or more features are considered. Finally, the algorithm considers the center zone with no features as occupied with a time-to-collision given by the average of the surroundings[7]. For example, zone 13 will have a TTC given by

$$TTC_{13} = \min \left\{ \frac{TTC_{12} + TTC_{14}}{2}, \ \frac{TTC_8 + TTC_{18}}{2} \right\}, \tag{4.17}$$

if all the surroundings have an associated TTC. For instance, if zone 18 has no features leading to no associated TTC in a time frame, the time-to-collision of the zone 13, in the respective time frame, will be given by

$$TTC_{13} = \frac{TTC_{12} + TTC_{14}}{2}. \tag{4.18}$$

This approach is relevant to identify danger zones when, for example, an obstacle has low texture and the vision algorithm only consider the zones 12 and 14 as dangerous.

[7] zones at right and left or/and zones at up and down

4.4 Drone Control

The control of the drone, as previously mentioned, resides on a combination of controllers: a PD controller for attitude control with an MPC controller for trajectory generation. As a first part of this process, the two controllers were implemented in a MATLAB environment and the ACADO Toolkit (Section 2.1.3) was used to generate C/C++ code for the MPC problem.

4.4.1 MPC Controller

The MPC controller is responsible for generating optimal controls by solving an optimization problem formulated as (2.10). To formulate the MPC problem the state vector is established as

$$\boldsymbol{x} = [x \quad y \quad z \quad v_x \quad v_y \quad v_z \quad \phi \quad \theta \quad \psi], \tag{4.19}$$

where $p^T = [x\ y\ z]$ is the position of the drone, $v^T = [v_x\ v_y\ v_z]$ the velocity of the vehicle and $q^T = [\phi\ \theta\ \psi]$ the orientation of the body frame $\{B\}$ in the world frame $\{W\}$. The control input is defined as

$$\boldsymbol{u} = [\phi_{cmd} \quad \theta_{cmd} \quad \psi_{cmd} \quad F_T], \tag{4.20}$$

where $F_T = k_F \sum_{i=1}^{4} \omega_i^2$, with k_F the thrust coefficient, is the total thrust force exerted by the four rotors. The commanded angles and force are then limited by the respective set

$$\mathbb{U} = \left\{ \boldsymbol{u} \in \mathbb{R}^4 | \begin{bmatrix} -\xi \\ -\pi \\ 0 \end{bmatrix} \leq \begin{bmatrix} \phi_{cmd}, \quad \theta_{cmd} \\ \psi_{cmd} \\ F_T \end{bmatrix} \leq \begin{bmatrix} \xi \\ \pi \\ F_{T_{max}} \end{bmatrix} \right\}, \tag{4.21}$$

where ξ corresponds to the bound desired value of the roll and pitch of the quadrotor. Drone dynamics are similar to the problem expressed by equation (2.5), however the attitude model used was based on [66] resulting in the attitude model

$$\dot{q} = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} \frac{1}{\tau_\phi} \left(k_{p\phi} \phi_{cmd} - \phi \right) \\ \frac{1}{\tau_\theta} \left(k_{p\theta} \theta_{cmd} - \theta \right) \\ \frac{1}{\tau_\psi} \left(k_{p\psi} \psi_{cmd} - \psi \right) \end{bmatrix}. \tag{4.22}$$

Finally, the complete drone model is expressed as

$$f(\boldsymbol{x}, \boldsymbol{u}) = \begin{bmatrix} v \\ -g + (RF_T + F_{ext})/m \\ \dot{q} \end{bmatrix} \in \mathbb{R}^9, \tag{4.23}$$

where $g = [0\ 0\ 9.8]\ m/s^2$ is the gravitation force, R is the rotation matrix as in equation (2.4), m is the quadrotor mass and F_{ext} corresponds to the external forces (i.e. wind) acting on the drone.

To establish a complete MPC problem it is also necessary to determine a cost function. Since the MPC controller is responsible for generating trajectories, the objective function consists in the difference between the current pose and the reference or desired pose

$$\mathcal{C}(x, u) = \|p - p_{des}\| + \|q - q_{des}\|. \tag{4.24}$$

The optimal control problem (OCP) is now defined as

$$\begin{aligned} \underset{\mathbf{x},\mathbf{u}}{\text{minimize}} \quad & \int_{t=0}^{T} \|p(t) - p_{des}(t)\| + \|q(t) - q_{des}(t)\|\ dt \\ \text{subject to} \quad & \dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}) \\ & \boldsymbol{u}(t) \in \mathbb{U} \\ & \boldsymbol{x}(0) = \boldsymbol{x}(t_0), \end{aligned} \tag{4.25}$$

where $x(0)$ is given by the state of the drone in each iteration of the control algorithm.

4.4.2 Attitude Controller

For the MATLAB simulation and verification, and before passing to the final test environment Gazebo, an attitude controller is implemented to generate the propellers angular speed ω_i (low-level control simulation) and with that verify thrust and drone behavior. Each rotor produces a force F_{Ti} and torque τ_i expressed as

$$F_{Ti} = k_F \omega_i^2 \quad \text{and} \quad \tau_i = k_\tau \omega_i^2, \tag{4.26}$$

where k_F and k_τ denote thrust and torque coefficients respectively. The relation between force and momentum with angular propeller speed is expressed as

$$\begin{bmatrix} F_T^{\{B\}} \\ \tau_x^{\{B\}} \\ \tau_y^{\{B\}} \\ \tau_z^{\{B\}} \end{bmatrix} = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & lk_F & 0 & -lk_F \\ -lk_F & 0 & lk_F & 0 \\ k_\tau & -k_\tau & k_\tau & -k_\tau \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix}, \tag{4.27}$$

where force and torque are applied in the body frame $\{B\}$ and l is the arm length of the drone. From OCP (4.25) is obtained F_T and for τ estimation it is used a PD controller that has a component proportional to the error between the observed and the desired orientation and another parcel proportional to the derivative of the error. Identically to the equation (2.7), it results

$$\tau = k_p(q_{des} - q) + k_d(\dot{q}_{des} - \dot{q}), \tag{4.28}$$

where k_p and k_d are the proportional and derivative gains respectively and are obtained using the tuning method from Ziegler-Nichols.

The MATLAB simulations verify a single movement from a point to another, so a PD controller is more than enough to verify the MPC behavior in the MATLAB environment. Following equation (4.27), the rotors speed are immediately obtained:

$$\begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{bmatrix} = \begin{bmatrix} k_F & k_F & k_F & k_F \\ 0 & lk_F & 0 & -lk_F \\ -lk_F & 0 & lk_F & 0 \\ k_\tau & -k_\tau & k_\tau & -k_\tau \end{bmatrix}^{-1} \begin{bmatrix} F_T^{\{B\}} \\ \tau_x^{\{B\}} \\ \tau_y^{\{B\}} \\ \tau_z^{\{B\}} \end{bmatrix}. \tag{4.29}$$

The attitude controller will perform drone stabilization to achieve the desired orientation. The goal is to have a null roll and pitch values ($\phi = \theta = 0$) so that the drone preserves a movement close to the hover condition. So, from the OCP in (4.25) it results

$$\begin{aligned} \underset{\mathbf{x},\mathbf{u}}{\text{minimize}} \quad & \int_{t=0}^{T} \|p(t) - p_{des}(t)\| + \|\psi(t) - \psi_{des}(t)\| \, dt \\ \text{subject to} \quad & \dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}) \\ & \boldsymbol{u}(t) \in \mathbb{U} \\ & \boldsymbol{x}(0) = \boldsymbol{x}(t_0). \end{aligned} \tag{4.30}$$

4.4.3 Image Integration in the Control Problem

Before exporting the C++ code with the ACADO Toolkit, the solution in the Gazebo environment needs to avoid obstacles and track a target during the flight.

The obstacles are estimated from the TTC calculation together with the danger zone associated as in Figure 4.6. To relate this information in the OCP and guarantee a safe flight it is necessary to have a reasonable distance between obstacle and drone. Therefore, the adopted approach, consists of using a potential field in the cost function that repulses the drone from dangerous zones. The potential term is hyperbolic making the cost function to increase when the drone is at a certain distance d to the obstacle. The objective function is now

$$\mathcal{C}(x, u) = \|p - p_{des}\| + \|\psi - \psi_{des}\| + \sum_{k=0}^{\mathcal{O}-1} \left(\frac{d}{\|p - p_{crit_k}\|} \right)^{p_w}, \tag{4.31}$$

where p_{crit_k} is the position associated to the obstacle k resulting from the center point of a zone of the image, as in Figure 4.6, with a small TTC, $\mathcal{O}$ is the number of obstacles calculated during flight and p_w is the power that makes the cost increase if the distance to the critical point $\|p - p_{crit_k}\|$ is smaller than

d. The OCP to be exported from MATLAB is

$$\begin{aligned} \underset{\mathbf{x},\mathbf{u}}{\text{minimize}} \quad & \int_{t=0}^{T} \|p(t) - p_{des}(t)\|_Q + \|\psi(t) - \psi_{des}(t)\|_P + \sum_{k=0}^{\mathcal{O}-1} \left(\frac{d}{\|p(t) - p_{crit_k}(t)\|} \right)_J^{p_w} dt \\ \text{subject to} \quad & \dot{\boldsymbol{x}} = f(\boldsymbol{x}, \boldsymbol{u}) \\ & \boldsymbol{u}(t) \in \mathbb{U} \\ & \boldsymbol{x}(0) = \boldsymbol{x}(t_0), \end{aligned} \tag{4.32}$$

where Q, P and J are weight matrices[8].

The tracking part directly influences the generated trajectory. As mentioned in Section 4.3.1, when the target is identified, the area and the center of the target in the image plane are computed. With this information, it is possible to know the drone distance and the yaw orientation, ψ, to the target. Therefore, at every position of the drone along time, the target position, the desired position and the position of the obstacles are represented in the body frame $\{B\}$. The goal is to maintain the target centered with a pre-determined area in the image plane (see Appendix A.1 and A.2) leading to the p_{des} and ψ_{des}.

4.4.4 OCP Solution and Code export

To test the MPC problem in the Robot Operating System (ROS), ACADO Toolkit [20] was used to generate the C/C++ code. For solving the OCP (4.32) a multiple shooting technique is used leading to the discretization of the system model and constraints in $N = 40$ time steps $t_0, ..., t_{39}$ with duration $dt = 0.05s$ which results in a prediction horizon of $2s$. A real-time iteration Gauss-Newton method is auto-generated and uses an algorithm for solving the quadratic programs (QPs). For last, ACADO CODE GENERATION exports multiple files containing the global values and functions together with a template to run the MPC algorithm.

[8] the weight matrices only appear in the final expression because the previous OCPs represent the previous steps to achieve the equation (4.32).

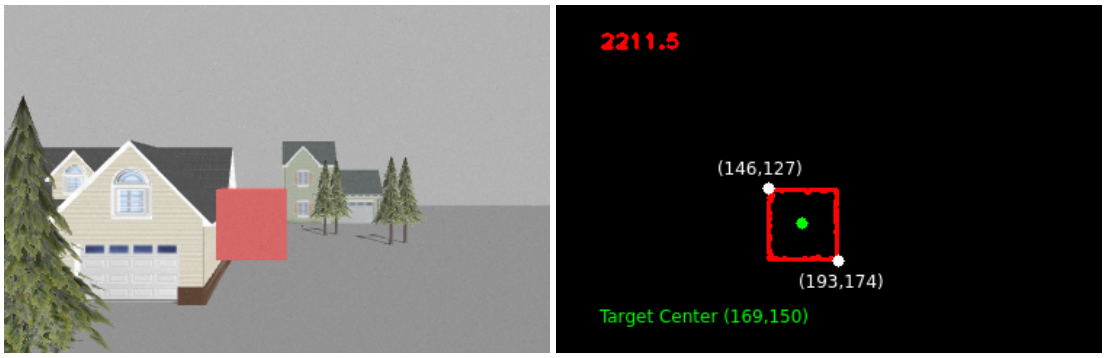
Chapter 5

Verification and Validation

In this chapter some results from multiple tests in MATLAB and Gazebo environments that validate the method described in the previous chapter are presented. In the Gazebo environment experiments from ROS implementations with the MAV simulation framework RotorS [70] are performed. In addition to simulated data, TTC computations were validated with real data.

5.1 Target Detection

To be able to track a target at a certain distance it is necessary identify it and obtain its size in the image (see Appendix A.1). From a situation where the drone hovers in front of the target, it is analyzed the image obtained from the camera, as illustrated in Figure 5.1. The Algorithm 1 is verified with the accurate computation of the target area and center represented in Figure 5.1(b).

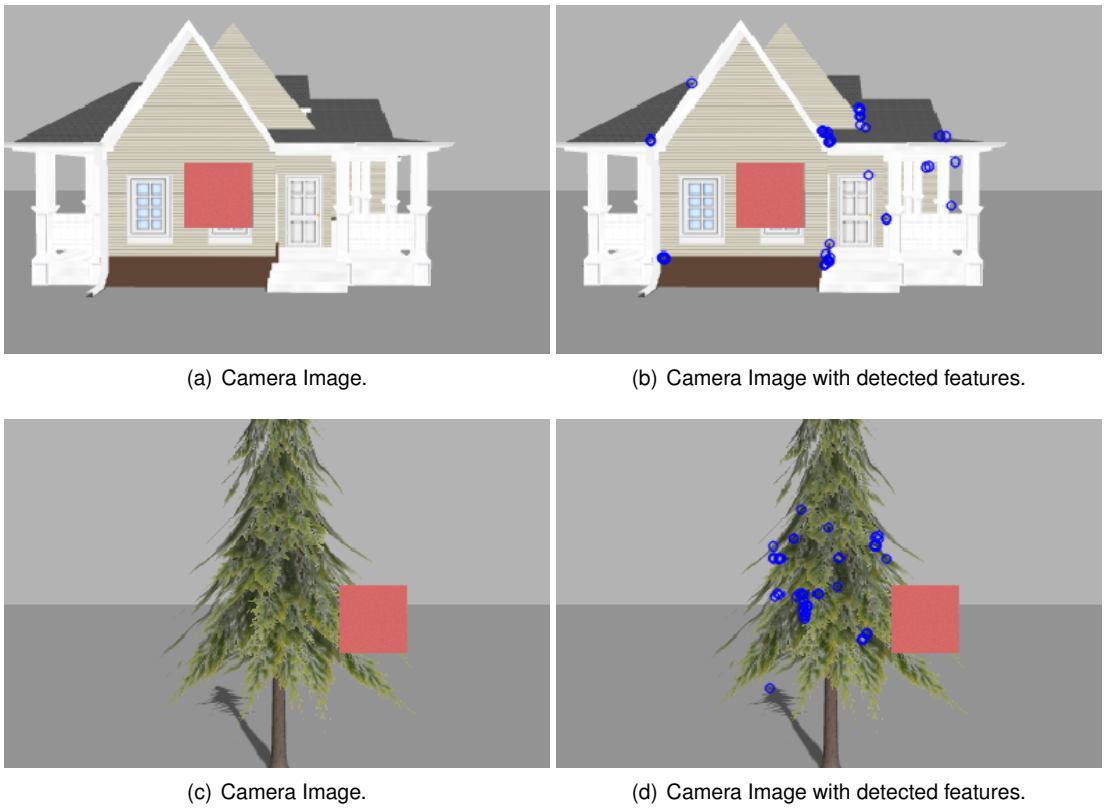


(a) Camera Image. (b) Target Detection and respective area ($2211.5\ px$).

Figure 5.1: Target detection based on color (red) and area calculation.

5.2 Feature Detection

Regarding the feature extraction from the image, the ORB detector was used. Again, one should notice that the target area is not considered in the feature extraction (see Section 4.3.2). Figure 5.2 illustrates the results of the feature detection strategy for a situation with the target in front of a house.



(a) Camera Image. (b) Camera Image with detected features.

(c) Camera Image. (d) Camera Image with detected features.

Figure 5.2: ORB Features example - detection and representation of the 50 best features in two different situations: stopped target in front of a house and moving target that starts to circumvent the tree.

The results in the figures of the right side of Figure 5.2 exhibit that the ORB detector can accurately extract interest points from the image, where the marked features correspond to high texture points. In the house situation, it is visible that the detected features correspond mostly to corner points where there are some color transitions. In the tree situation, the features are also identified where there are color modifications. These two examples are representative of the highlighted features by the detection method used during this project.

5.3 Feature Reprojection

To verify the features reprojection, it is necessary to check the roll and pitch corrections illustrated in Figure 4.4. First the roll correction is performed in the image plane (Figure 4.4(a)) followed by the pitch compensation (Figure 4.4(b)).

5.3.1 Roll correction

Figure 5.3 presents an image rotation with a marked point A which in frame $\{1\}$ has the coordinates $A^1 = (188, 0)$.

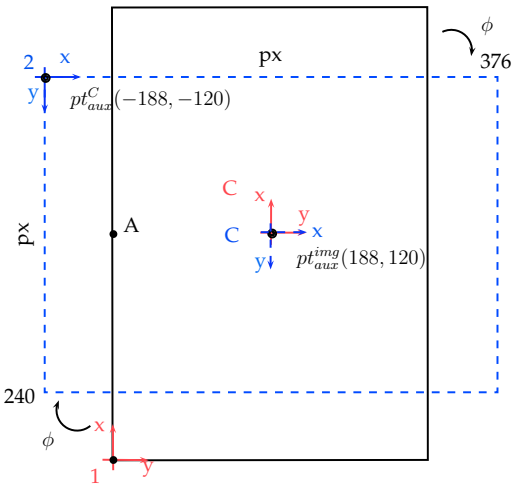


Figure 5.3: Image rotation (roll $= 90$º), with point A reprojection, from the camera orientation $\{1\}$ to the pretended orientation $\{2\}$ (blue).

From equation (4.2) and with the roll rotation presented, the point A in frame $\{2\}$ is expressed by the relation

$$A^2 = R_{90}(A^1 - pt^{img}_{aux}) - pt^C_{aux} \ . \tag{5.1}$$

As the image dimensions are 376x240 px, it is expected that $A^2 = (68, 120)$, which from equation (5.1) is immediately verified:

$$\begin{aligned} \begin{bmatrix} 68 \\ 120 \end{bmatrix} &= \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \left(\begin{bmatrix} 188 \\ 0 \end{bmatrix} - \begin{bmatrix} 188 \\ 120 \end{bmatrix} \right) - \begin{bmatrix} -188 \\ -120 \end{bmatrix} \\ &= \begin{bmatrix} 68 \\ 120 \end{bmatrix} . \end{aligned} \tag{5.2}$$

5.3.2 Pitch correction

The pitch correction verification is illustrated with an example in Figure 5.4, with a vertical field-of-view of 53º (see Appendix A.3), a pitch rotation of -26.5º and a point M representing a feature in the image plane.

Given that the image height is 240 px, in the black line case, the M point is given by $M^1 = (x, 120)$ and in the blue situation, $M^2 = (x, 240)$, where the image plane frame is the same as in Figure 4.4(a). The point M^2 in the virtual plane is obtained effortlessly and the y value in equation (4.3) is validated:

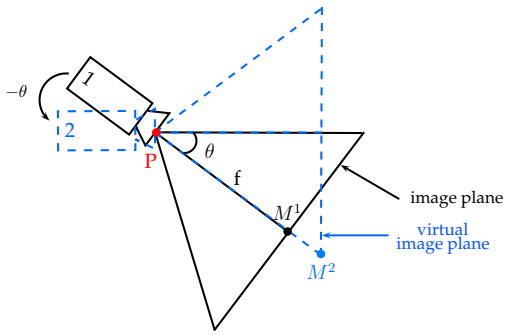


Figure 5.4: Image rotation (pitch $= -26.5$º) from the camera orientation $\{1\}$ to the pretended orientation $\{2\}$ (blue).

$$
\begin{aligned}
M^2 &= \begin{bmatrix} x \\ 120 + f\ tan(\theta_{IMU} + \theta_{px}) \end{bmatrix} \\
\begin{bmatrix} x \\ 240 \end{bmatrix} &= \begin{bmatrix} x \\ 120 + 240\ tan(26.5 + 0) \end{bmatrix} \\
\begin{bmatrix} x \\ 240 \end{bmatrix} &= \begin{bmatrix} x \\ 240 \end{bmatrix}.
\end{aligned}
\tag{5.3}
$$

As shown in equation (4.3), the y coordinate in the image plane varies according with the drone orientation and the pixel position in the image plane. In order to better illustrate the shift in the y coordinate of the pixels from the image plane to the virtual the plane, Table 5.1 represents the variation of different y pixels due to a $\theta_{IMU} = 26.5$º rotation.

Table 5.1: y pixel values in virtual plane with $\theta_{IMU} = 26.5$º as in Figure 5.4

y_{img} (px)	θ_{px} (º)	y_{virt} (px)	$\Delta y = y_{virt} - y_{img}$ (px)
120	0	240	120
90	-7.1	204	114
60	-14.0	173	113
30	-20.6	144	114
0	-26.5	120	120

5.4 Feature Matching

To validate the feature matching, between ORB features, is represented in Figure 5.5 a situation where the best features are highlighted and matched in consecutive frames.

In this situation is possible to verify that the features are not computed in the target region, as pretended for the FOE computation, and the marked matches between images are mostly correct. In this example, two wrong matches were detected, showing that the matching method is not fully precise.

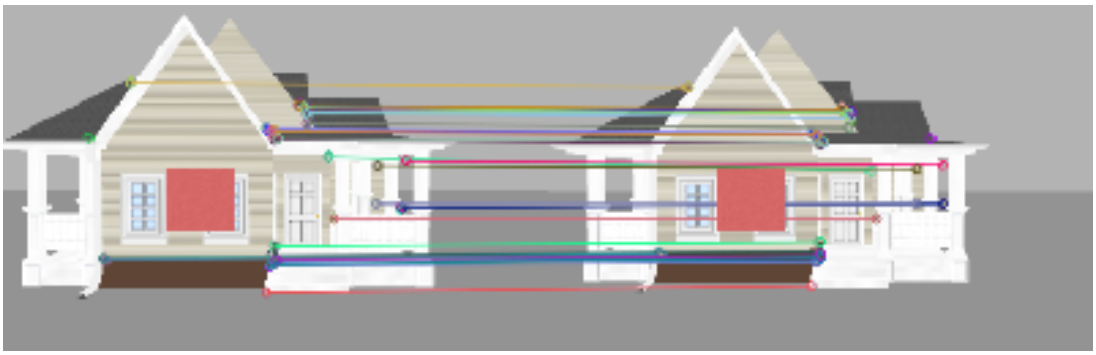


Figure 5.5: Matches example between the new frame (left) and the previous frame (right).

5.5 Target Tracking

In this section, the behavior of the drone while following a target during flight and maintaining a pre-established distance to it is verified. This distance is guaranteed due to the knowledge of the target dimensions and area in the image plane (see Appendix A.1).

5.5.1 MATLAB environment

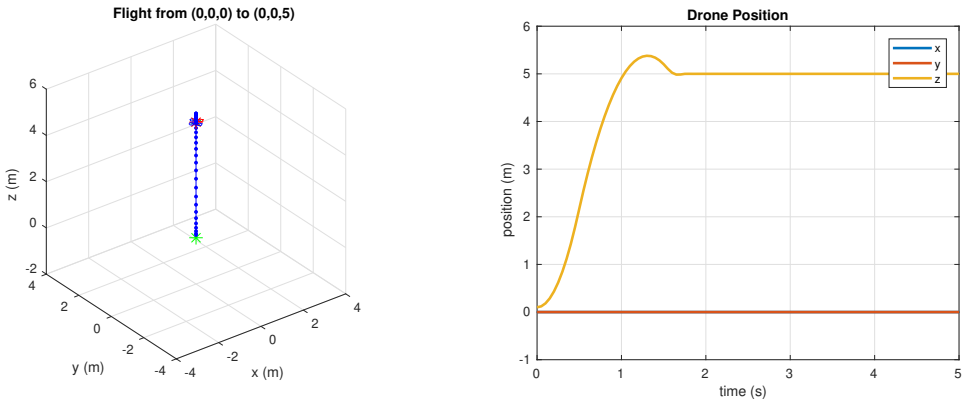
Before exporting the C++ code, the MATLAB implementation is verified. Here, the goal is to perform position tracking from a short route. Following the explanation given in Section 4.4, both MPC and PD controllers are used. The existing constraints in the RotorS framework, such as the maximum rotors speed, are also considered. In the following, performance verification in flights such as vertical take-off or waypoint tracking is presented. In this phase, obstacle avoidance and target tracking are not directly analyzed. For last, the values associated with some parameters in Chapter 4 are now indicated in Table 5.2.

Table 5.2: Parameter values used along the experiments.

Parameter	Value	Description
m	$1\ kg$	drone mass
l	$0.21\ m$	drone arm length
ξ	$\frac{\pi}{4}\ rad$	roll and pitch boundaries
$F_{T_{max}}$	$28\ N$	maximum thrust
ω_{max}	$838\ rad/s$	maximum rotor speed
k_F	9.98×10^{-6}	thrust coefficient
Q	$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 2 \end{bmatrix}$	position weight matrix
P	10	yaw weight matrix
J	identity matrix	obstacles weight matrix
$\tau_{\phi,\theta,\psi}$	$[2\ 2\ 0.5]$	attitude model parameter
$k_{p_{\phi,\theta,\psi}}$	$[2.24\ 2.24\ 2.8]$	attitude model parameter

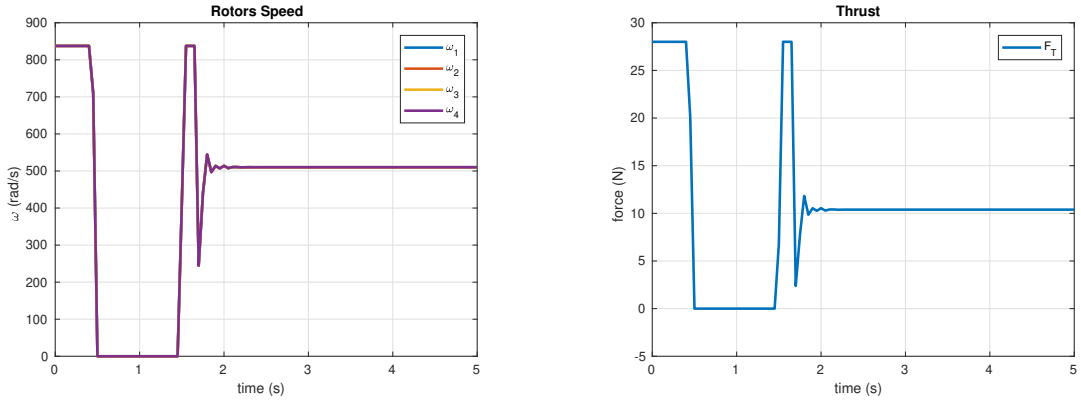
Vertical take-off

The first trajectory verified was a vertical take-off from position $(0, 0, 0)$ to $(0, 0, 5)$. The results are displayed in Figures 5.6 and 5.7.



(a) (b)

Figure 5.6: Vertical take-off (a) and drone position along time (b) from z = 0 to z = 5.



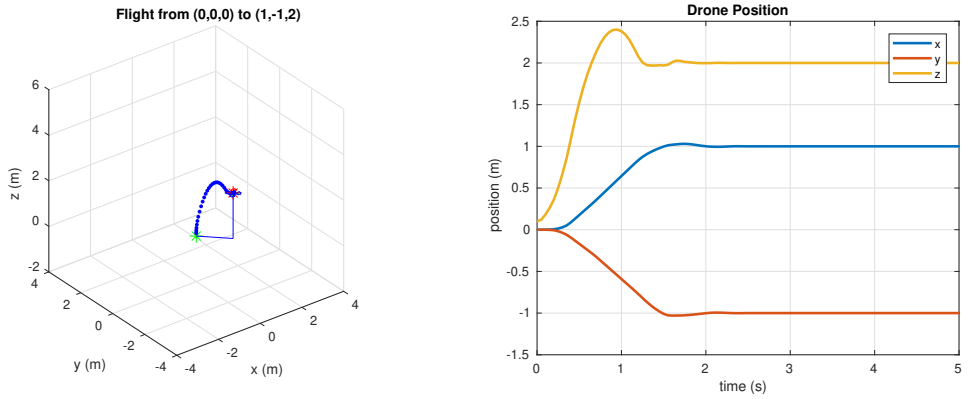
(a) (b)

Figure 5.7: Rotors speed (a) and Thrust force (b).

From inspection of the Figure 5.6 it is easily verified that the drone reaches the desired position $(0, 0, 5)$. In Figure 5.7 it is evident that the rotors speed is somewhere between the limits of the physical system (maximum $838\ rad/s$) and the resulting force, F_T, is between 0 and $28\ N$ as expected from equation (4.26).

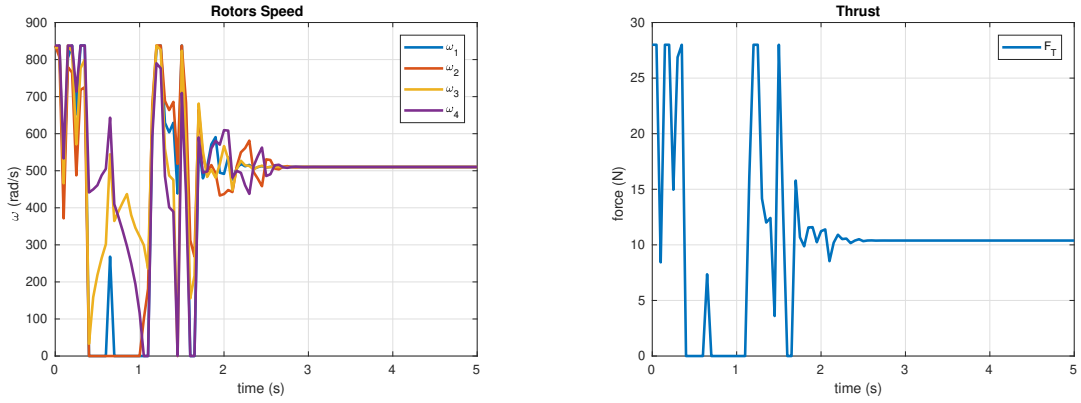
Simple path with $\psi \neq 0$

Next, it is performed an experiment where the drone performs a flight from position $(0, 0, 0)$ to position $(1, -1, 2)$ with a final yaw attitude value of $\pi/4$. The results are presented in Figure 5.8 to 5.10.



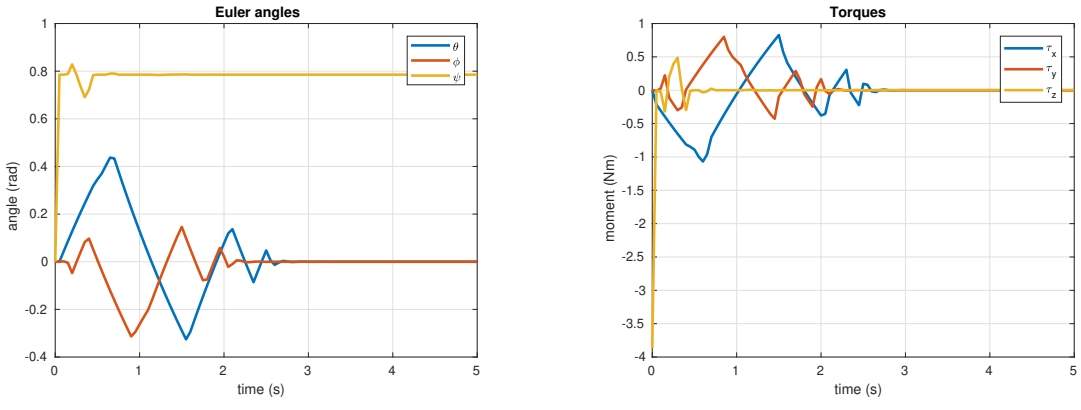
(a) (b)

Figure 5.8: Trajectory (a) and position (b) in flight from $(0, 0, 0)$ to $(1, -1, 2)$.



(a) (b)

Figure 5.9: Rotors speed (a) and Thrust force (b).



(a) (b)

Figure 5.10: Euler angles (a) and Torques (b).

From inspection of the Figure 5.8 it is verified that the drone reaches the desired position $(1, -1, 2)$. Figure 5.9 illustrates that the rotors speed and thrust are, as in the previous situation, between the physical limits (see equation (4.21)). In Figure 5.10 both Euler angles and Torques alter similarly as expected from equation (4.28).

5.5.2 Gazebo environment

For tracking verification, different flights were performed and analyzed, where the drone and target started from positions $(0, 0, 2.3)$ and $(4, 0, 2.3)$ respectively, as illustrated in Figure 5.11. The goal is to maintain the target on sight, centered in the image plane, and with an area of 900 px leading to a distance of 4 meters approximately (see Appendix A.1). It is also important to underline that, to verify the drone behavior in this environment, is used an MPC controller together with an attitude controller from the RotorS framework. The MPC controller runs at a frequency of $1\ Hz$, whereas the attitude controller is updated from IMU data with a rate of $10\ Hz$.

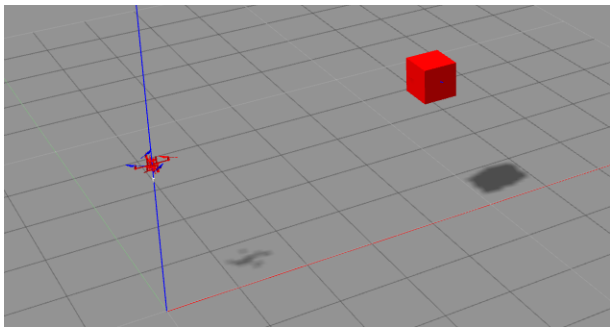


Figure 5.11: Drone and target initial positions along the tracking verification experiments.

Linear Tracking

The first flight in Gazebo environment had the goal of verifying the linear tracking. In this condition, the drone movement was restricted to the x-axis. Five trials were performed and the results are represented in Figure 5.12. The difference between the distance to the target and the objective distance is also represented in Figure 5.13.

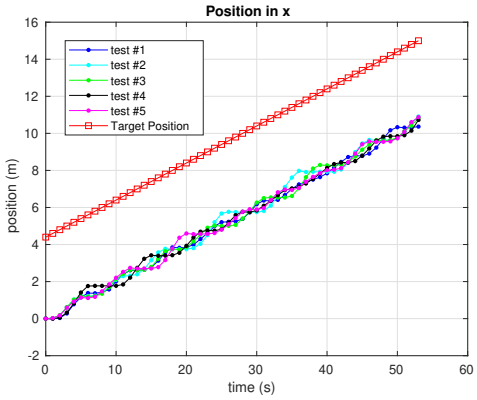
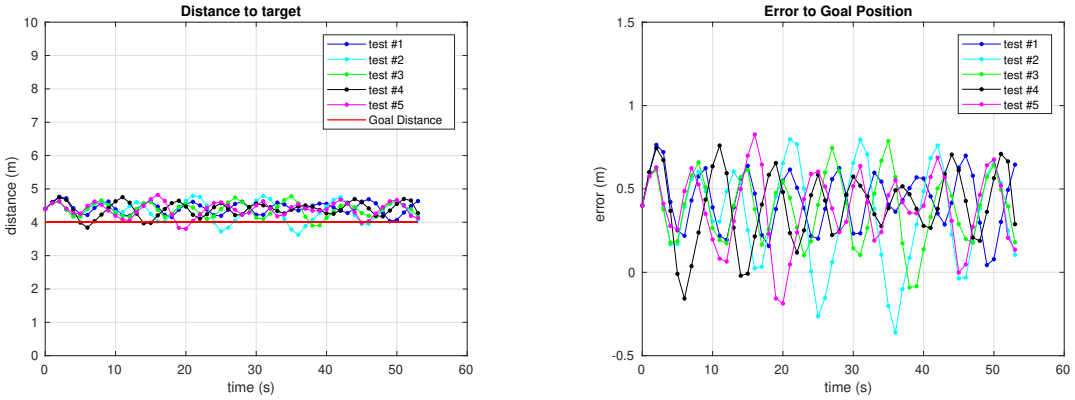


Figure 5.12: Drone and target position in x along time.



(a) Distance to the target and goal distance.

(b) Error to Goal Position.

Figure 5.13: Differences between the objective and the real position.

From Figure 5.12 is confirmed that the target starts in position $x = 4$, the drone starts in position $x = 0$, and the distance between these during the flight is close to $4\ m$, as observed in Figures 5.13(a) and 5.13(b). An analysis of Figure 5.13(b) leads to the Table 5.3 where the mean error position and standard deviation on every trial are presented.

Table 5.3: Position mean error and standard deviation during linear tracking.

Trial (#)	Mean Error (m)	Std Deviation (m)
1	0.432	0.170
2	0.342	0.293
3	0.383	0.205
4	0.400	0.214
5	0.384	0.224

The results in the Table 5.3 show a mean error of $\sim +0.4m$ and a standard deviation lower than $0.3m$. These results are consistent and support the implemented strategy. The mean error is justified by the fact that the drone reacts to the movement of the target. Moreover, it should be notice that the target is incrementally moving at a rate of $10\ Hz$ while the MPC updates the desired position at $1\ Hz$.

Circular Tracking

Further verification of the tracking algorithm is provided by an analysis with the drone operating in a horizontal circular path. Again the error between the goal distance and the real distance to the target is analyzed. Figures 5.14 and 5.15 display the results for five trials.

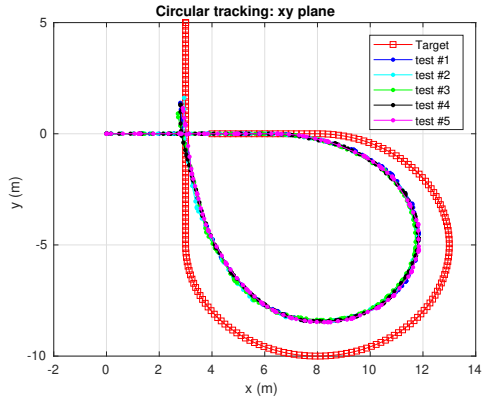
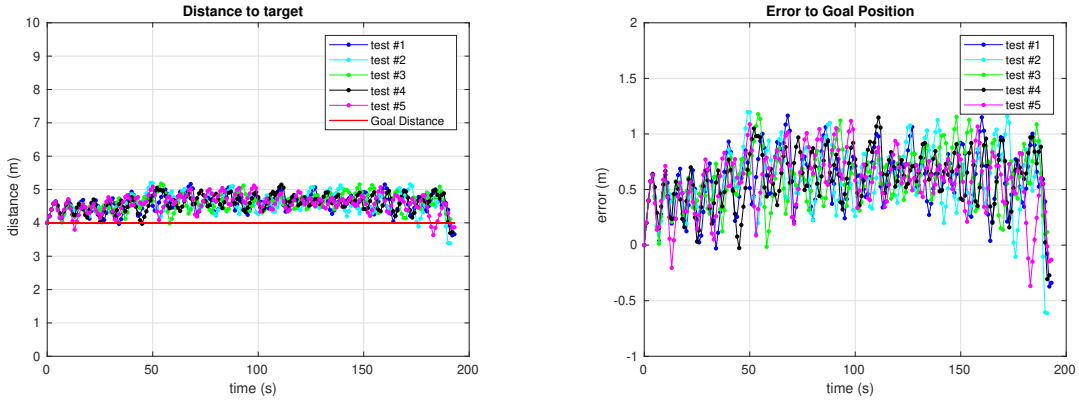


Figure 5.14: Position in the xy plane along time for the drone and target in each of the 5 trials. Drone and target starting positions are $(0, 0)$ and $(4, 0)$ respectively.



(a) Distance to the target and goal distance along time

(b) Error along time.

Figure 5.15: Differences between the objective and the real position.

Figure 5.14 shows that the circular trajectory of the drone is shorter than the target trajectory. This behavior is expected because the tracking solution consists in maintaining a certain distance to the target and adjusting yaw orientation to keep the target centered in the image plane. Thus, the drone is continuously reacting to the movement of the target. From the analysis of Figure 5.15 results the Table 5.4.

The results in Table 5.4 reveal that the solution has a mean error of $\sim +0.6m$ and a standard deviation lower than $0.3m$. As in the linear tracking case, the mean error is justified by the difference between the MPC rate and the continuous movement of the target.

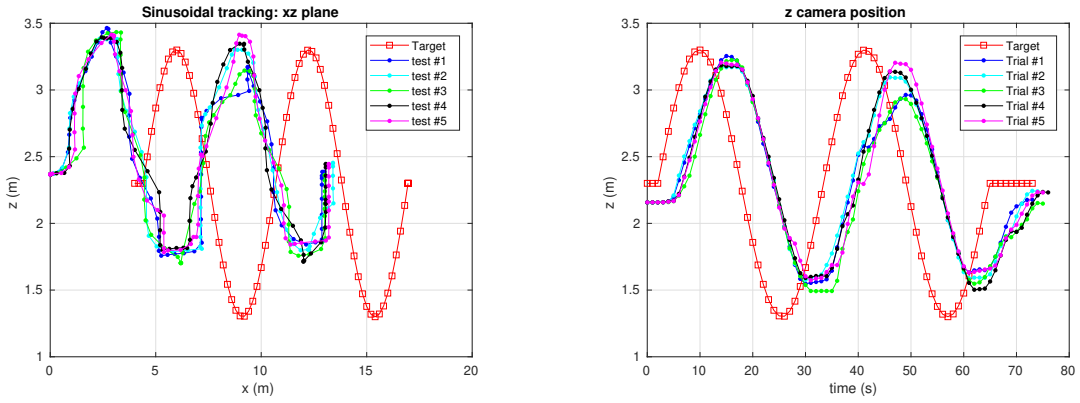
Table 5.4: Position mean error and standard deviation during circular tracking.

Trial (#)	Mean Error (m)	Std Deviation (m)
1	0.578	0.278
2	0.594	0.288
3	0.601	0.247
4	0.601	0.251
5	0.572	0.276

Comparing with the linear case, circular tracking has an error increase of around $20cm$. This was expected because the direction of the movement is not the same for the drone and the target. The standard deviation in the circular situation has also a small growth compared to the linear experiment. However, the value is lower than $30cm$, hence it is concluded that the result is consistent and verifies a correct tracking of the target in both situations.

Sinusoidal tracking

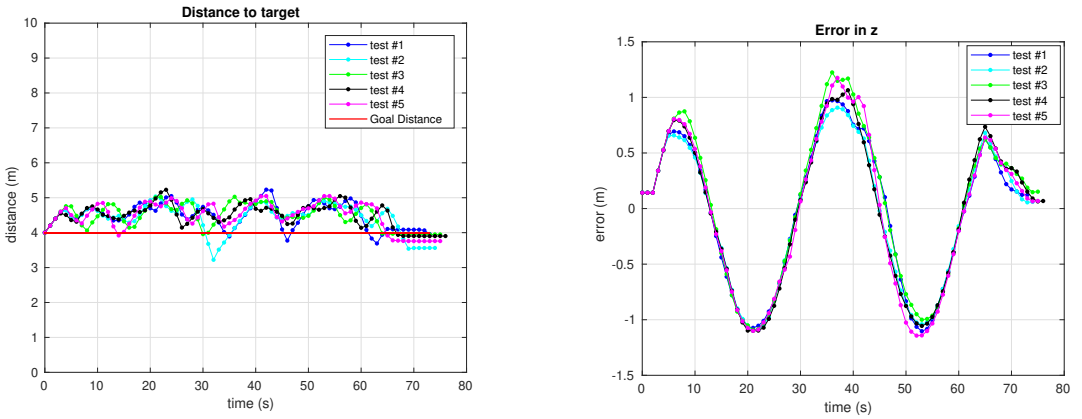
The last test environment of the tracking algorithm is performed by following a target which is moving in a sinusoidal trajectory in the xz plane. For consistency with previous results, it were also performed 5 simulations. The results are presented in Figures 5.16 and 5.17.



(a) (b)

Figure 5.16: Drone position in xz plane (a) and camera height along time (b).

In Figure 5.16(a) it is represented the movement of the target for the five flights performed, Figure 5.16(b) illustrates the camera height along time and the delay between the reaction of the drone to the movement of the target. The camera position is represented because the strategy relies on centering the target in the image plane, so the solution will drive the camera height to be the same as the target. Figure 5.17(a) shows that, in this trajectory, the drone maintains an average distance to the target within four and five meters. This result is expected, as in the two previous situations (linear and circular tracking), because the target is updating is position at a faster rate ($10\ Hz$) than the drone ($1\ Hz$). From Figure 5.17(b) is evaluated the differences in height between the camera and the target resulting in the



(a) Distance to the target along time.

(b) Difference in height between drone and target.

Figure 5.17: Differences between the goal position and the real position.

information of the Table 5.5.

Table 5.5: Mean error and standard deviation in height between camera and target during sinusoidal tracking.

Trial (#)	Mean Error (m)	Std Deviation (m)
1	-0.023	0.651
2	-0.042	0.632
3	0.043	0.689
4	-0.016	0.665
5	-0.021	0.699

Besides the small mean error, the standard deviation indicates that the camera position deviates from the average position close to $0.7m$. This outcome is justified by the fact that the target moves freely and continuously respecting to the drone, leading to a natural difference between the real position and the goal position.

5.6 TTC Verification and Validation

After the tracking part, it is necessary to verify the TTC computations. This data will be vital to avoid obstacles during flight. Flight collisions were performed to analyze the TTC evolution along time. The TTC evolution was verified, at a first instance, in the entire image and then verified in the 25 zones of the image as shown in Figure 4.6.

5.6.1 TTC verification on Gazebo environment

In the Gazebo environment, a camera that acquires 10 images per second is mounted on the drone. To reduce the computational cost of the process, only two images per second are analyzed. The present

situation consists of a drone flying towards a house. Figure 5.18 represents the beginning of the situation.

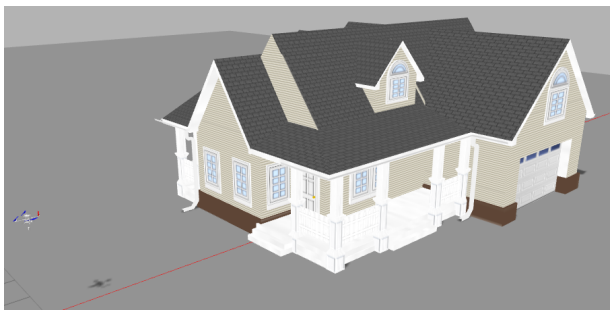
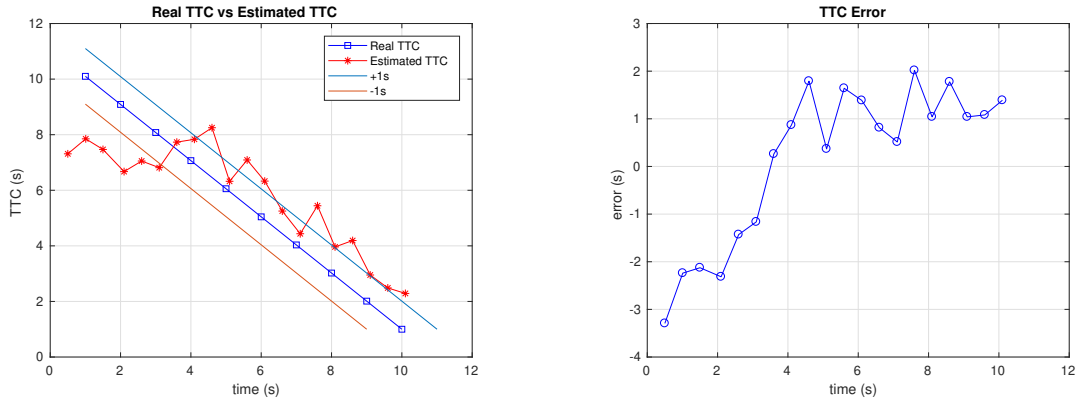


Figure 5.18: Drone flight in direction of a house in Gazebo simulation.

Whole image TTC

The estimated TTC of the image plane is displayed in Figure 5.19, together with the real TTC, obtained from the Gazebo data.



(a) Distance to the target along time.

(b) TTC error during flight.

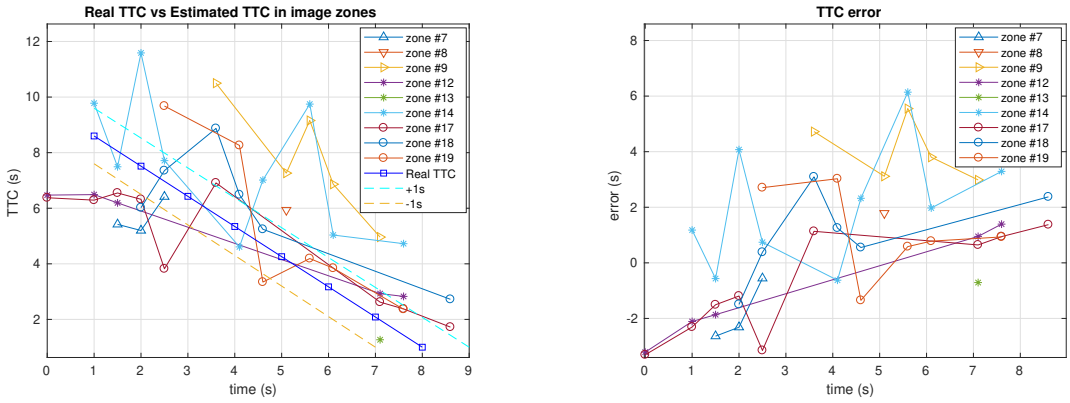
Figure 5.19: TTC results from the drone flight in Figure 5.18.

From Figure 5.19(a) it is verified that the computed TTC has similar behavior to the real TTC. However, Figure 5.19(b) shows that, after the first 4s, the estimated TTC is larger than the real TTC. The TTC computed above uses the data from all the image, therefore, feature points far from the image center that are far from the collision site can contribute to the observed increase in the TTC.

TTC by zones in house collision

In the same situation, the TTC is computed individually in all the 25 image zones, as illustrated in Figure 4.6. To avoid plotting 25 lines in the same figure, the zones were analyzed in two different groups: the center group (3 by 3 zones centered in number 13) and the border group (all 16 zones in the border).

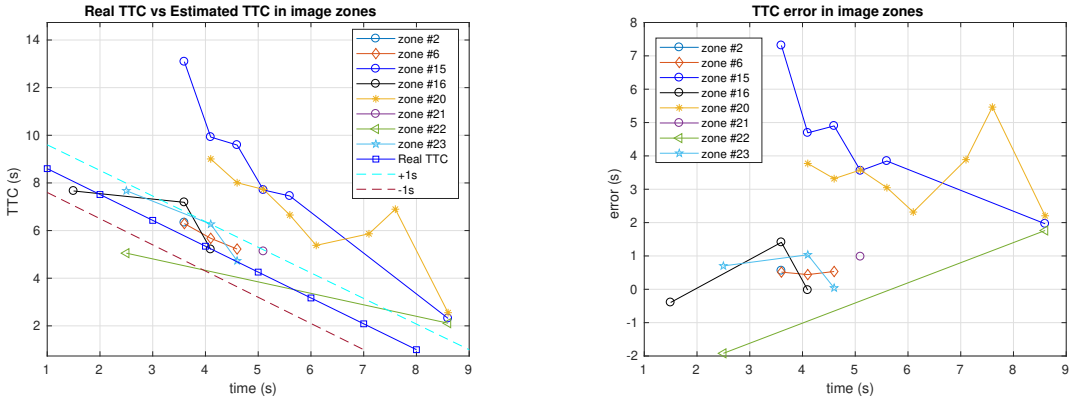
The results are presented in Figures 5.20 and 5.21.



(a) TTC along time.

(b) TTC error during flight.

Figure 5.20: TTC results from drone flight in Figure 5.18 in the zones of the center group.



(a) TTC along time.

(b) TTC error during flight.

Figure 5.21: TTC results from drone flight in Figure 5.18 in the zones of the border group.

From the figures 5.20 and 5.21, it is possible to verify that the TTC evolves, from a global perspective, according to the real time-to-collision. For a better understanding, the mean error and the standard deviation correspondent to each zone are represented in Table 5.6.

Table 5.6: TTC mean error and standard deviation in each of the zones during house collision.

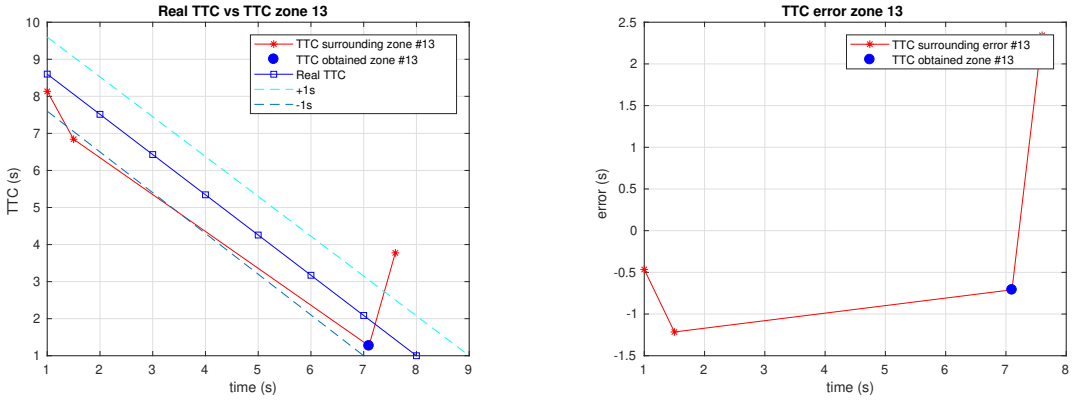
Zone	Mean Error (s)	Std Deviation (s)	Zone	Mean Error (s)	Std Deviation (s)
2	0.56	0.00	15	4.38	1.77
6	0.50	0.00	16	0.33	0.95
7	-1.84	1.12	17	-0.82	1.88
8	1.79	0.00	18	1.03	1.62
9	4.035	1.09	19	1.11	1.59
12	-0.97	2.02	20	3.45	1.02
13	-0.71	0.00	21	0.98	0.00
14	2.06	2.21	22	-0.08	2.60
			23	0.59	0.51

From inspection of table 5.6, the first thing to notice is that not all the 25 zones are represented. This

fact is due to:

- the implemented strategy only uses the 100 best ORB features accordingly to the Harris Score;
- a TTC in a zone is only assigned if it has a minimum of three key points;
- the lack of texture in the environment.

Moreover note that zone 13, the center zone, is only identified as potentially dangerous by the features one time. However, the implemented strategy predicts that this zone is also considered as occupied when both zone 12 and 14 have an associated TTC or when both zone 8 and 18 have a TTC. Following equations (4.17) and (4.18), the TTC of zone 13 is computed and expressed in Figure 5.22.



(a) Time-to-collision of zone 13 along time.

(b) TTC error of zone 13 during flight.

Figure 5.22: TTC of zone 13 completed with the surroundings data.

From Figure 5.22 it is verified that the estimated TTC is similar to the real TTC. The absence of data (only 4 data points) is explained by the fact that the strategy requires that in the same time frame, both surroundings of zone 13 have an associated TTC. The final data point discrepancy results from the influence of the high value of TTC in zone 14 at that instance.

Finally, the results in Table 5.6 show that the mean error in most of the zones is smaller than 2 seconds. The zones with high standard deviation are mostly the ones with a mean error close to zero which emphasizes the fact that the TTC values are oscillating around the true result.

TTC by zones in tree collision

Now, a TTC by zones verification is performed in a collision with a different obstacle, a tree, as represented in Figure 5.23.

In this situation the algorithm only assigned TTC to zones in the central group, leading to the results in Figure 5.24.

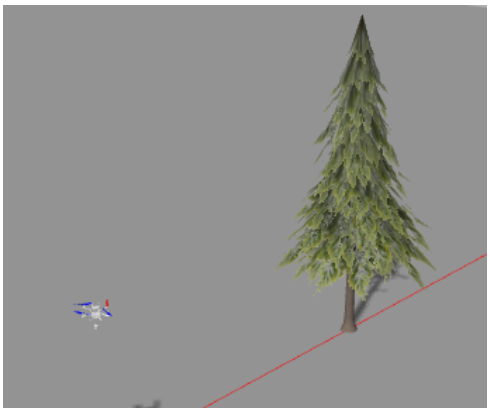
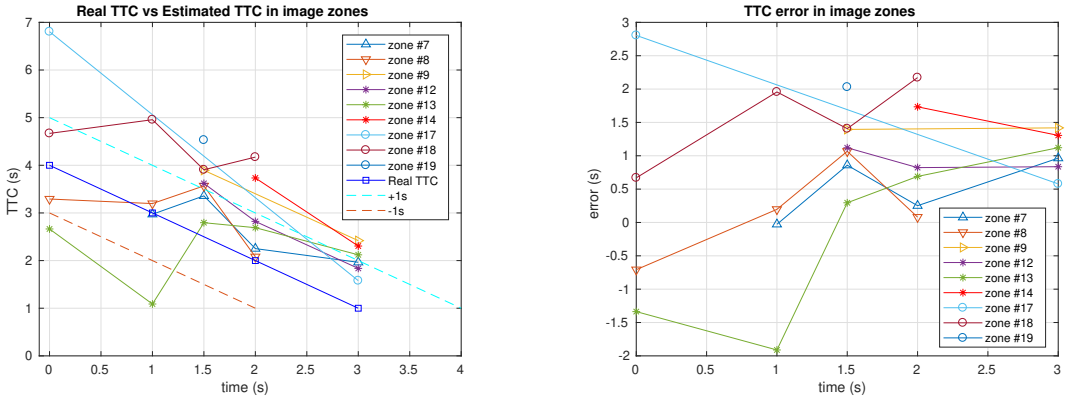


Figure 5.23: Drone flight in direction of a tree in Gazebo simulation.



(a) Time-to-collision along time.

(b) TTC error during flight.

Figure 5.24: TTC results from the drone flight in Figure 5.23 in the zones of the image.

From Figure 5.24 it is verified that the expected TTC is consistent with the real value of time-to-collision, indicating that, when there are difficult and similar features to track, the algorithm has a satisfactory performance. The data in Table 5.7 demonstrate the accuracy of the TTC.

Table 5.7: TTC mean error and standard deviation in each of the zones during tree collision.

Zone	Mean Error (s)	Std Deviation (s)
7	0.51	0.48
8	0.16	0.73
9	1.41	0.00
12	0.93	0.17
13	-0.23	1.32
14	1.52	0.30
17	1.69	1.57
18	1.55	0.67
19	2.03	0.00

The results emphasize that both the mean error and the standard deviation remain small in all zones. These outcome suggests that the algorithm deals properly with obstacles that have similar features and with obstacles that do not cover all the image, resulting in an accurate TTC computation.

5.6.2 TTC validation with real data

The TTC calculations were also validated with real video data[1]. The test environment was a situation where the video starts at the beginning of a hall and finishes colliding into an obstacle. The video was recorded on a moving platform, with a phone camera, recording at $30Hz$, with a size of 1920x1080 px. Then, a central region of interest with size 960x540 px was analyzed. Figure 5.25 illustrates the situation in three different time frames together with the marked features in each situation.

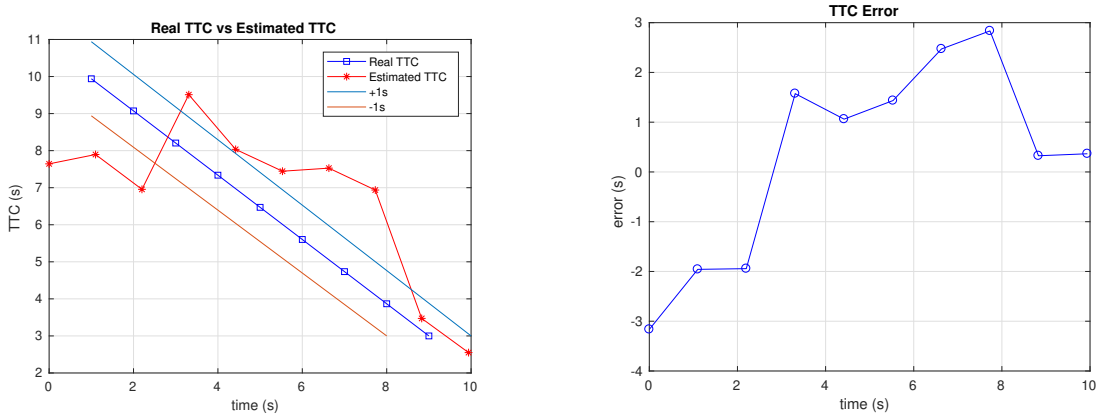
(a) Camera far from the obstacle. (b) Camera half-way to the obstacle. (c) Camera close to the obstacle.

Figure 5.25: Three different stages in a hall 'flight'. The colored dots correspond to the marked matches by the vision algorithm with the following TTC: Green points: TTC $>$ 8s. Yellow points: 4s $<$ TTC $\leq$ 8s. Red points: TTC $\leq$ 4s.

From Figure 5.25 it is possible to observe that the algorithm has the expected performance in the TTC computation. When the camera is far from the obstacle, the pixels have an associated TTC greater than 8s. When the camera gets closer to the obstacle, the TTC values are smaller as predicted.

Whole image TTC

The TTC was computed for the entire image, originating the results in Figure 5.26.



(a) Time-to-collision along time. (b) TTC error during flight.

Figure 5.26: TTC results from the entire image.

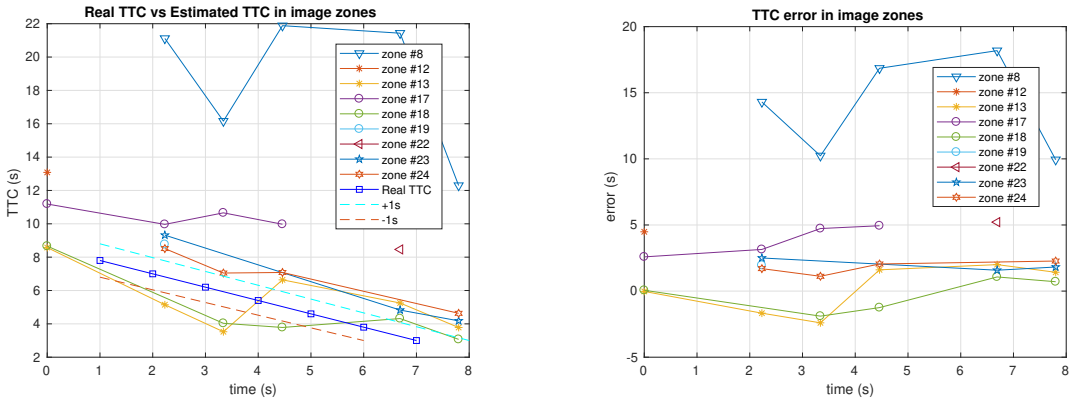
The results show that the estimated TTC has a similar behavior to the real TTC. Nevertheless, in the final seven seconds, the estimated TTC is larger than the real TTC. This event is due to the estimated

[1]available at: https://drive.google.com/file/d/14Bho8f7syhMqu2ouhmpnpfCo0kY7nDPN/view?usp=sharing

TTC being computed from all the image and some pixels have a larger associated TTC as shown in Figure 5.25(c).

TTC by zones

In the same environment, the time-to-collision was also computed in each of the 25 zones of the image plane (see Figure 4.6), producing the results in Figure 5.27.



(a) Time-to-collision along time.

(b) TTC error during flight.

Figure 5.27: TTC results from video in the hall.

From figures 5.27(a) and 5.27(b), it is visible that again the estimated TTC has similar behavior to the real TTC. The results in some zones are close to the real value, namely, the data from zones 12, 13, and 18 show identical evolution. The discrepancy in the results of zone 8 compared with the other zones can be explained by the mismatching of some features leading to an increase of the TTC.

From a global perspective, the outcome with the real data are in accordance with the one obtained in the simulation environment, nonetheless, the differences to the real value may lead the drone to misjudge the environment in front.

Computational time

Besides the time-to-collision validation, the computational time in each iteration of the algorithm was also analyzed. The results are in Figure 5.28.

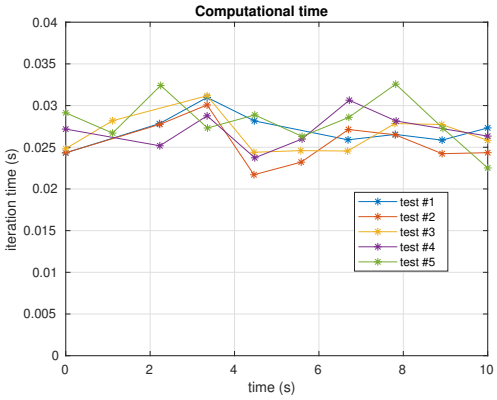


Figure 5.28: Vision algorithm iterations time along 5 tests.

The results in Figure 5.28 validate that the vision algorithm can be used in a real-time solution given that, in all the iterations, the computational cost was not higher than $35ms$. In more detail, the average iteration time and respective standard deviation are presented in Table 5.8.

Table 5.8: Average iteration time and standard deviation in the vision algorithm.

Trial (#)	Avg iter (ms)	Std Deviation (ms)
1	27.1	2.0
2	25.5	2.6
3	26.6	2.3
4	27.0	2.2
5	28.2	2.9
Average	26.9	2.4

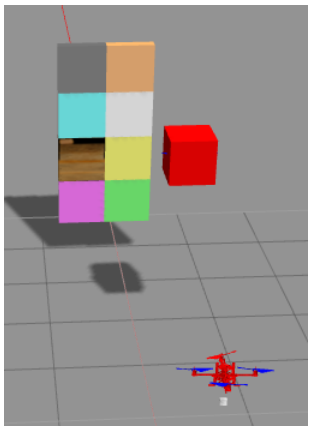
The results average, with an average iteration time of $26.9ms$ and a standard deviation of $2.4ms$, demonstrate the accuracy of the implementation and the consistency of the vision algorithm along the executed tests.

5.7 MPC potential field

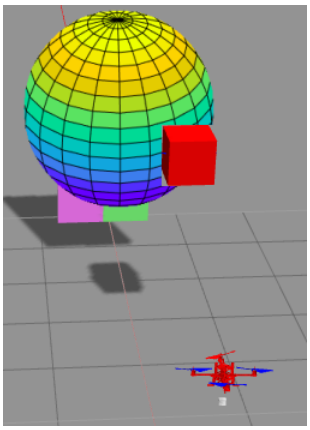
To avoid obstacles, the present solution uses spherical potential fields to repel the drone to safer zones. The potential term in the MPC optimization problem (4.32) in Section 4.4.3 is recalled:

$$\sum_{k=0}^{\mathcal{O}-1} \left(\frac{d}{\|p - p_{crit_k}\|} \right)^{p_w}. \tag{5.4}$$

The algorithm consider this field as illustrated in Figure 5.29.



(a)



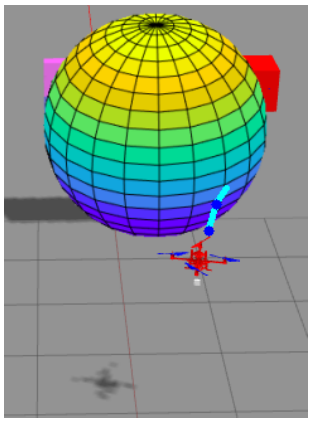
(b)

Figure 5.29: Situation before (a) and after (b) the algorithm generate a spherical potential field.

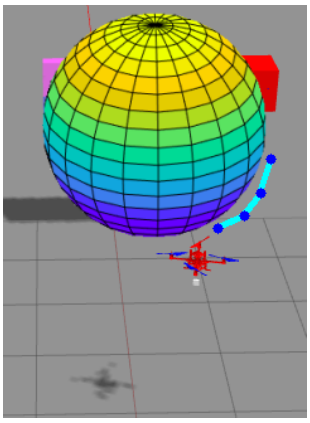
To avoid the obstacle the MPC will generate a path to evade the potential. The path is changed from the original path as a result of the cost function increase in the equation 4.32 when the drone is at a distance d or less from the obstacle, i.e., when the following condition is verified:

$$\|p - p_{crit_k}\| < d. \tag{5.5}$$

The increase of the cost function in the MPC problem drives the MPC to generate a trajectory that is far from the obstacle and as well far from the potential as represented in Figure 5.30.



(a)



(b)

Figure 5.30: Situation before (a) and after (b) the MPC computes a new trajectory that avoid the spherical potential.

To verify this behavior, a simple tracking situation was performed in the Gazebo environment, where an object was placed close to the drone.

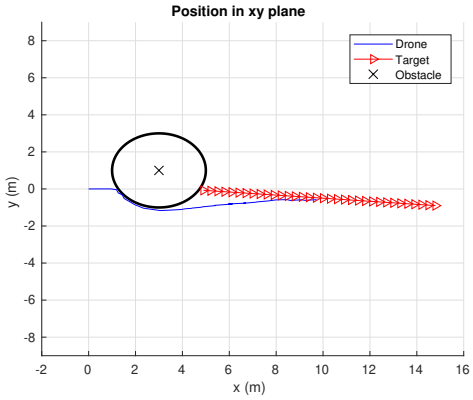


Figure 5.31: Potential field generated in the drone path during flight by the presence of an obstacle.

The results in Figure 5.31 illustrate the movement of the drone in the xy plane when the trajectory is readjusted due to an obstacle. From observation, it is verified that the drone does not penetrate the field and it is able to track the target during the flight.

Chapter 6

Results

This chapter presents results from multiple tests in the Gazebo simulation environment. The experiments used the *RotorS* framework [70] and were executed on a Lenovo computer, running Ubuntu 16.04, with an Intel Core i5-8250U CPU (4 cores @ 1.60GHz) with 8GB of RAM. Since one of the requirements in this project is to use a monocular camera and an IMU as external sensors, image analysis becomes crucial to the outcome. An accurate integration in the MPC problem is also fundamental to obtain significant results. The drone must be able to follow a target while avoiding possible collisions and flying according to the generated commands.

6.1 Testing Environments

Multiple environments were used to test the final solution. In these environments, the target is flying around one or more obstacles but it is never hidden by them. The strategy will identify potential danger regions and with this information generate safe trajectories far from the obstacles.

6.1.1 One obstacle

The first experiment is the analysis of the behavior of the drone in the presence of an obstacle. This example consists of a multi-color obstacle, as shown in Figure 6.1, with the target flying in a xy plane (see Figure 6.2(a)) maintaining a constant height. The nature of the obstacle is justified because the detection method is dependent on the existence of texture in the image. The results are displayed in Figure 6.2.

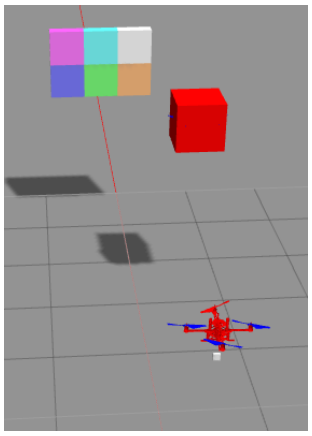
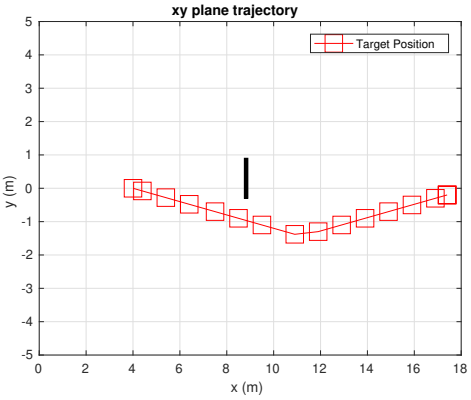
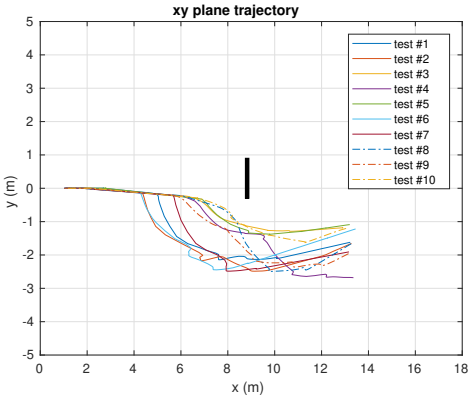


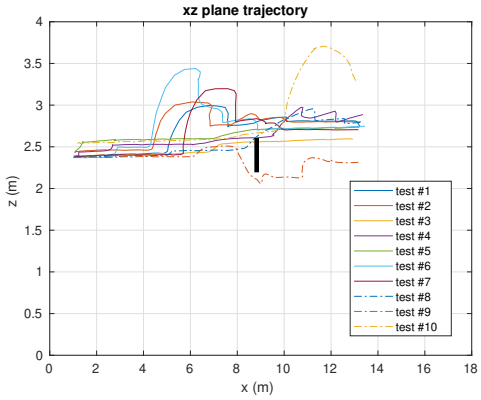
Figure 6.1: Tracking target in one obstacle environment.



(a) Target path in xy plane.



(b) Drone path in xy plane.



(c) Drone path in xz plane.

Figure 6.2: Results from tracking the target and avoiding the obstacle in 10 trials. Target initial position $(4, 0, 2.3)$ and drone initial position $(0, 0, 2.3)$.

The outcome in Figure 6.2 emphasizes that the strategy can lead the drone to avoid an obstacle while following a target. The results also show that the drone never circumvents the obstacle in the same trajectory. These different paths are due to the different TTC computations and distinctive p_{crit_k} estimations (see optimization problem (4.32)). In Figures 6.2(b) and 6.2(c), it is also visible that sometimes the drone performs an avoidance maneuver when it is distant from the obstacle or after passing by

it. This happens because the TTC and the danger zone approach are not fully precise leading to values slightly different from the real TTC. Moreover, these results show that the trajectory generated by the MPC leads the drone to avoid the spherical potential moving both in y and z directions. This indicates that the costless path is one that minimizes, at the same time, movement in y and z as expected by the cost function in the optimization problem (4.32). Figure 6.2 also shows that the target is never hidden from the drone and its tracking occurs flawlessly.

Another important metric is the distance between the obstacle and the drone when they are side-by-side. In every trial, the target position is the same, about 40 cm to the obstacle, however the drone position differs and the results of the distance of it to the obstacle are represented in Figure 6.3.

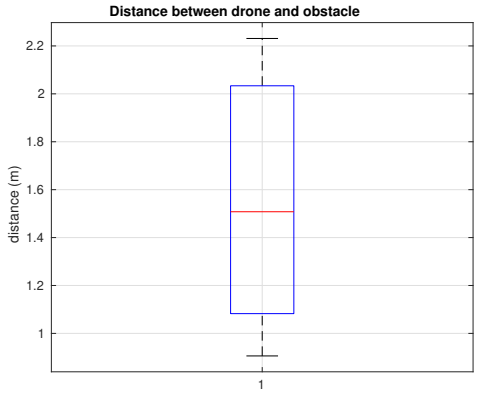
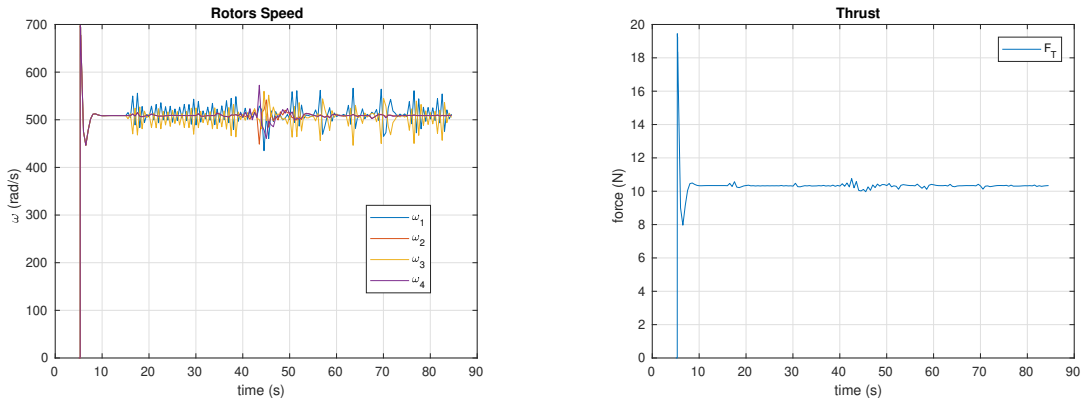


Figure 6.3: Safety distance between obstacle and drone when they are side by side in the xy plane along the 10 trials.

The results in figure 6.3 show that the median value of the distance between the drone and the obstacle, when they are side by side, is around $1.51m$, with a minimum distance of $0.91m$ and a maximum of $2.23m$. These outcomes are expected because the obstacles are estimated at every iteration and are confined to one of the 25 zones of the image plane, leading to some position errors. Nevertheless, the results are satisfactory due to the fact that the drone is able to track the target, and simultaneously, avoid the obstacle with a secure margin.

Besides the trajectory, both rotors speed and thrust along time were obtained and are displayed in Figure 6.4. For sake of simplicity, the results are only considered for the trial $\#5$ which resemble the results obtained in all simulations.

Observe that the drone starts its movement by going up and hovering in a certain position. Then, it starts the movement where all the four rotors are continuously changing the rotation rate to maintain the drone horizontal. The fact that, on most of the flight, ω_1 is larger than ω_3 verifies that the drone is moving forward as shown in figures 6.1 and 6.2.



(a) Rotors speed along time.

(b) Thrust along time.

Figure 6.4: Drone rotors speed (a) and respective thrust (b) in Trial $\#5$ with the four rotors numbered as in Figure 2.1 with rotor 1 and 3 rotating clockwise and rotors 2 and 4 counter clockwise in Gazebo simulation.

6.1.2 Two obstacles

In the next experiment, the goal is to analyze the behavior of the drone in the presence of two obstacles, as represented in Figure 6.5. The target and drone trajectories in the xy plane are displayed in figures 6.6(a) and 6.6(b) respectively. The drone xz plane trajectory is presented in Figure 6.6(c).

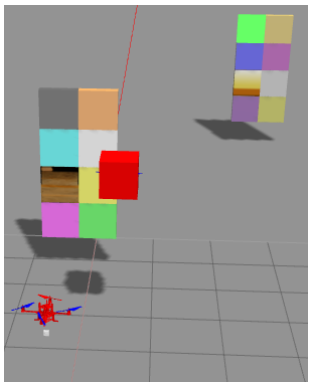
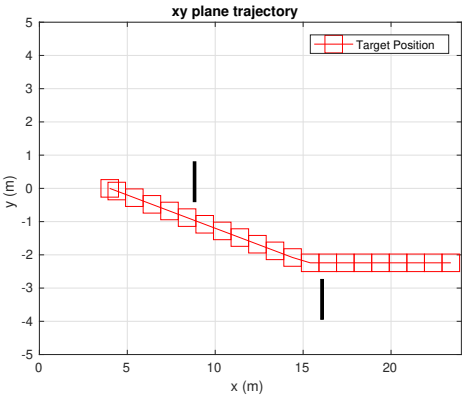
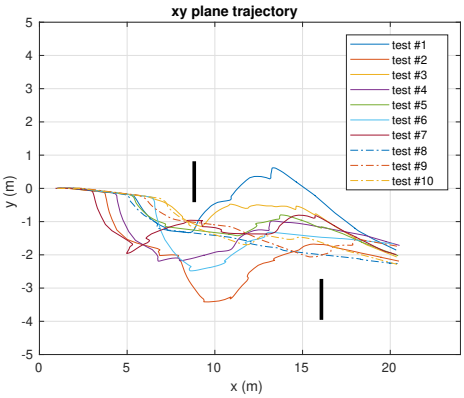


Figure 6.5: Two obstacles environment.

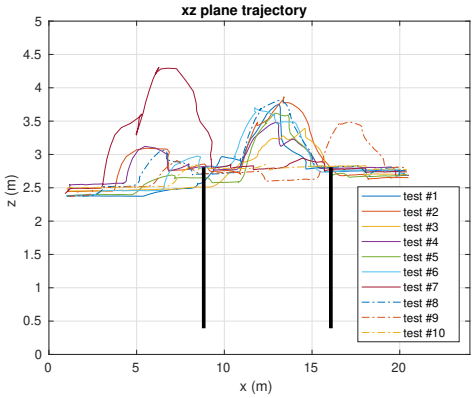
The results in Figure 6.6 exhibit that the drone can avoid multiple obstacles while maintaining a safe distance. Figure 6.6(b) shows that the drone executes the avoidance maneuver, a few meters away from the obstacle, and Figure 6.6(c) demonstrates the drone flying upwards somewhere far from the obstacles. These results are due to the imprecise TTC computations and reveal again that, for the MPC problem (4.32), it is less costly to deviate in two directions instead of one. The results in Figure 6.6 show 10 successful trials (out of 12), however, this environment revealed some restraints to the problem and the implemented strategy, such as the absence of texture in the image or target occlusion. The target occlusion problem is out of the scope of the thesis, hence, a search strategy is not covered by this dissertation. The absence of texture in the image is inherent to the current strategy and it is analyzed in more detail in the next section.



(a) Target path in xy plane.



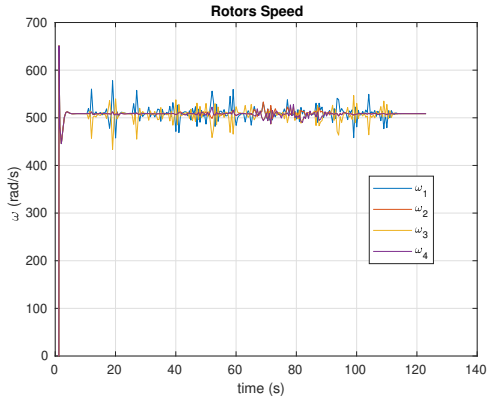
(b) Drone path in xy plane.



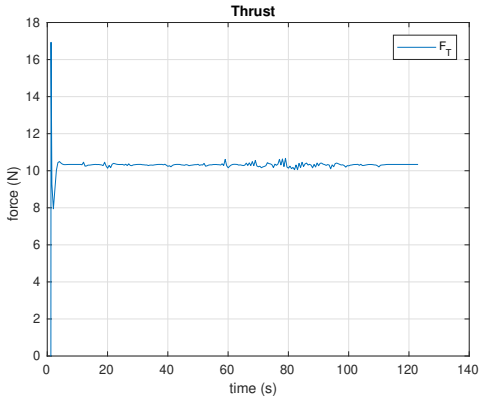
(c) Drone path in xz plane.

Figure 6.6: Target and drone movement respectively in two obstacle environments in 10 trials. Target initial position (4,0,2.3) and drone initial position (0,0,2.3).

Similarly to the previous obstacle environment simulation, the rotors speed and the thrust force were obtained and are displayed in Figure 6.7.



(a) Rotors speed along time.



(b) Thrust along time.

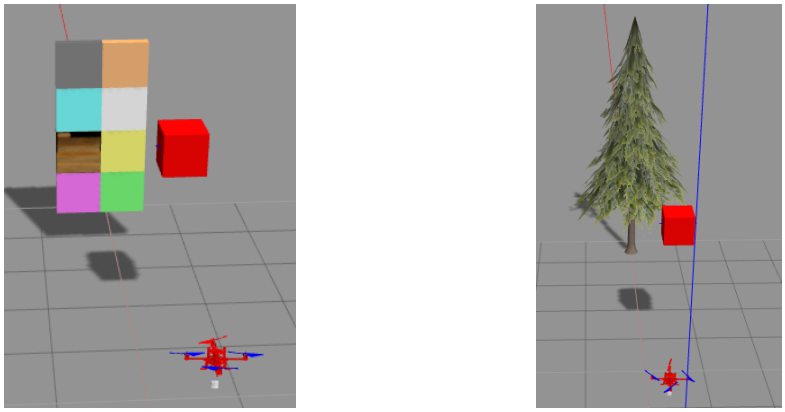
Figure 6.7: Drone rotors speed (a) and respective thrust (b) in Trial $\#5$.

The results in Figure 6.7 show that the movement of the drone is mostly forward. This is a relevant

aspect of the proposed strategy, since TTC computations are imprecise when yaw rotations or sideways movement occur.

6.1.3 Differences of the results in two environments

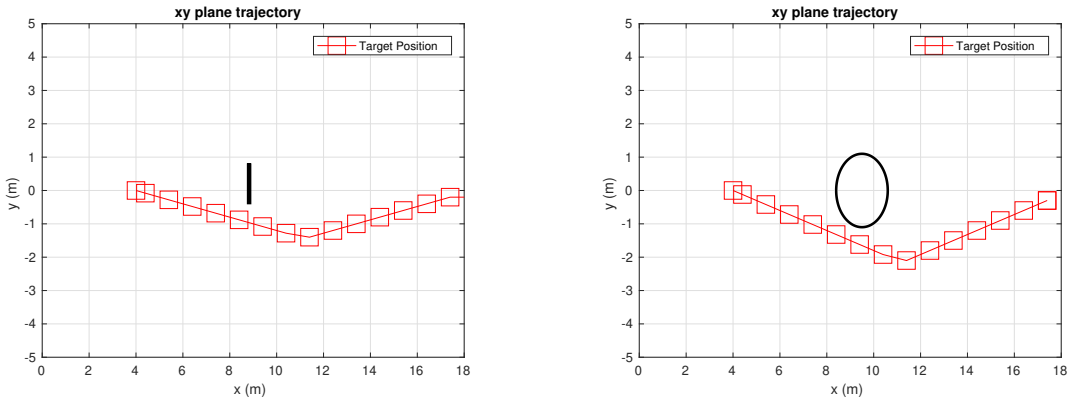
The environment used in the previous section reveals that the strategy could fail in situations where the number of features is insufficient leading to inaccurate or nonexistent TTC computations. To verify this behavior and analyze the performance of the strategy, the environments illustrated in Figure 6.8 were used.



(a) One wall obstacle. (b) Tree obstacle.

Figure 6.8: Two obstacle situations in Gazebo environment.

In both environments, the target maintains the height during the experiments. The movement in the xy plane is represented in Figure 6.9.



(a) One wall obstacle. (b) Tree obstacle.

Figure 6.9: Target movement in both situations.

In both situations, the target circumvents the obstacle with a safe distance of $30cm$. To evaluate the performance of the solution, 100 tests for both environments were executed. The safety distances for each trial are displayed in the boxplots in Figure 6.10 and in more detail in Table 6.1.

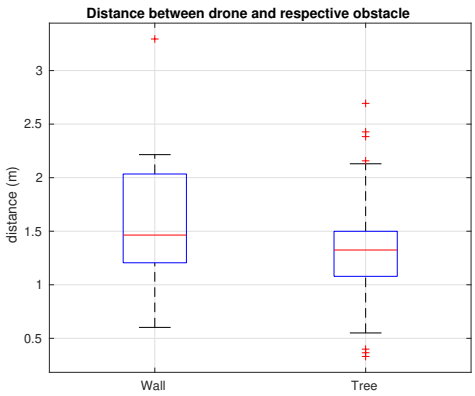


Figure 6.10: Drone safety distance in both situations along 100 tests.

Table 6.1: Drone success rate and safety distance in the two situations along 100 tests.

Environment	Success rate (%)	Distance to obstacle (m)					Outliers
		low	Q1	Median	Q2	up	
one wall	82	0.60	1.21	1.46	2.03	2.22	1
tree	99	0.55	1.08	1.32	1.50	2.13	7

As for the one wall situation, the success rate is 82%. The 18 tests that failed were due to imprecise TTC computations or/and insufficient features in the image plane, that led the drone to not identify some regions as dangerous. The information on the distance to the obstacle reveals that the avoidance maneuver is relatively consistent while avoiding the wall. The outlier can be explained by a situation where an obstacle is only on the border of a zone (see Figure 4.6 to recall image zones), leading the algorithm to consider that zone as occupied and producing an avoidance maneuver very far from the obstacle.

Regarding the tree situation, in all the 100 trials, a danger zone was identified. However, the success rate was 99% due to one trial where the considered obstacle was on the right side of the tree, creating a potential field that led the drone to execute a safety maneuver to the left side. In these circumstances, the target was occluded by the tree, leading the drone to stop moving because the target was not on sight. Nonetheless, the safety distance results indicate that the drone accurately recognizes a danger zone in the image plane. The three outliers below $0.55m$ are explained in situations where the obstacles were considered on top or bottom of the tree leading to a potential field that does not repel the drone to a safer distance. Moreover, the remaining four outliers can be explained by the fact that some matched features in the central zone, when the tree is somewhere on the left side of the image, lead the strategy to consider it as a danger zone, provoking an avoidance maneuver far from the obstacle, as the outlier in the one wall situation.

6.1.4 Larger scenario

A longer simulation was performed in an environment with three different obstacles: a tree, a wall, and a house, as represented in Figure 6.11.

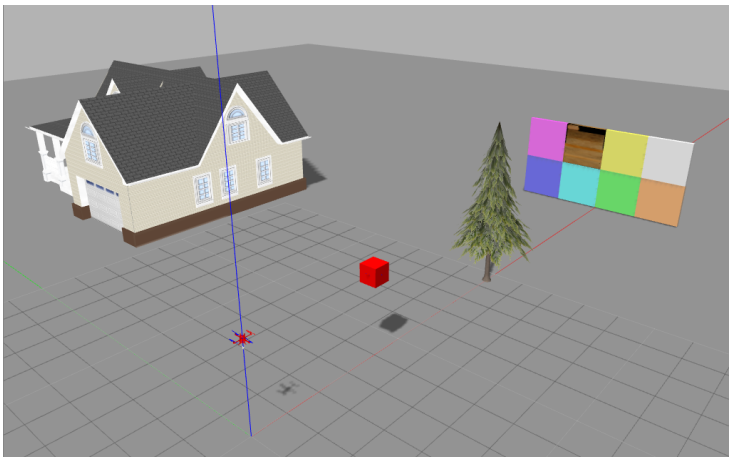


Figure 6.11: Environment with tree, wall and house as obstacles.

This simulation represents a drone flight that can accurately track a target and at the same time avoid obstacles, where the image analysis described in Chapter 4 is performed with the problem (4.32) solved in real-time. The data acquired during the test is subsequently analyzed and presented sequentially through the present section.

Trajectory

During the simulation, the target performs a trajectory that circumvents the obstacles. The drone starts its movement by hovering to the position $(0, 0, 2.3)$ while the target hovers to $(4, 0, 2.3)$. The trajectory of the target and the drone, on the xy plane, are represented in Figure 6.12. For better understanding the contours of the the house, tree and wall are added to the graph by a black solid line.

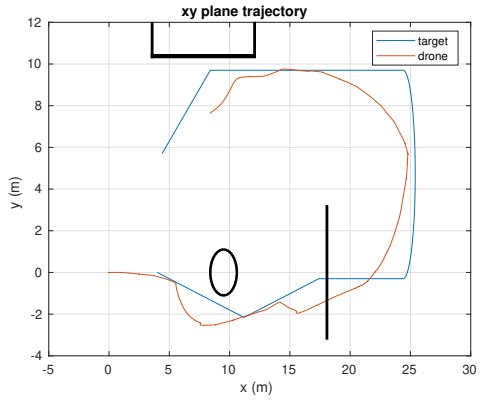


Figure 6.12: Trajectory in the xy plane of the drone and the target.

Observe that the drone trajectory is similar to the target trajectory, being visible that the drone performs an avoidance maneuver before approaching the tree and the house. Close to the house, the evasion movement is minimal because the house is far from the trajectory of the drone. In the case of the second obstacle, the wall was surpassed by flying over it as demonstrated in Figure 6.13.

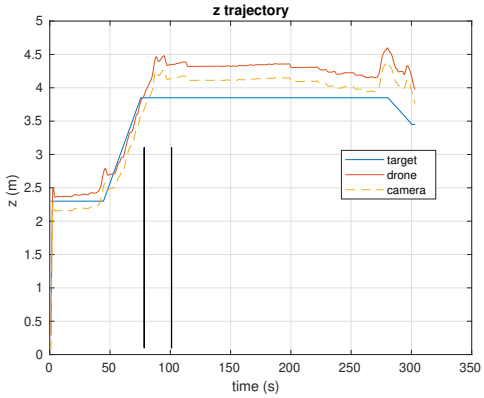


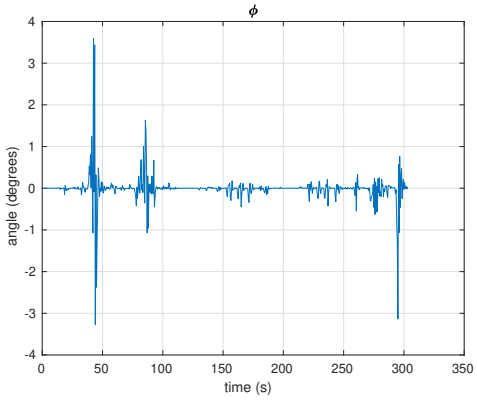
Figure 6.13: Trajectory in the z plane of the drone, target and camera.

The two walls in Figure 6.13 represent the moment in time when the target and the drone are respectively above the wall. Since the approach computes a drone trajectory to maintain the target centered in the image, a camera height similar to the target height was expected. However, and more explicitly after bypassing the wall, the camera height is bigger than the target height. This is a consequence of the target geometry (cube) that leads the algorithm to detect a center that is not in the real middle of the target but in the middle of the red shape that is detected in the image plane. Furthermore, the minimal difference between the target center and the image center leads the drone to maintain its height.

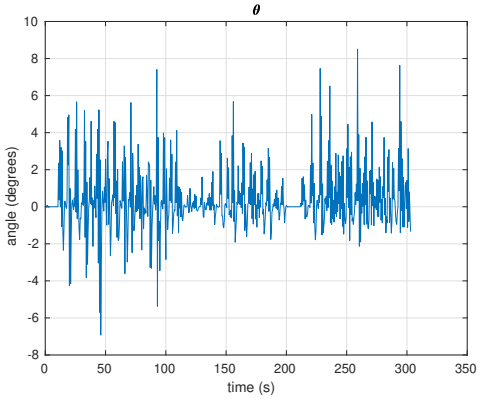
Drone attitude

Since the goal is to maintain a stable flight, close to hover condition, it is important to perform a flight without aggressive maneuvers. In order to analyze the attitude behavior, the temporal evolution of the rotation angle associated with the movements of roll, pitch and yaw are represented in Figures 6.14(a), 6.14(b) and 6.14(c), respectively.

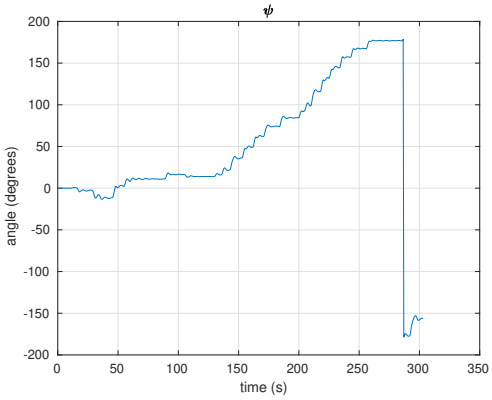
Figure 6.14(a) indicates that roll is close to zero during most of the flight. Nonetheless, it is possible to observe three significant oscillations around $t_1 = 45s$, $t_2 = 80s$ and $t_3 = 295s$. These events are related with the avoidance maneuvers. For instance, the steepest oscillation in Figure 6.14(a) is related to the maneuver performed to avoid the tree. The pitch value in Figure 6.14(b) illustrates the fact that most of the time, the drone is leaning forward, leading to forward flight. Both roll and pitch values show that the drone is close to hovering position during the flight and with a safe difference to the respective value constraints in equation (4.21). The yaw results (Figure 6.14(c)) indicate that the drone never produces aggressive oscillations around the z axis. The accentuated shift is due to the Gazebo angles domain that varies between -180 and 180 degrees.



(a) Roll.



(b) Pitch.

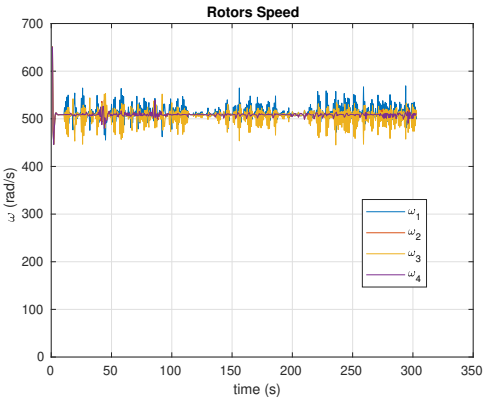


(c) Yaw.

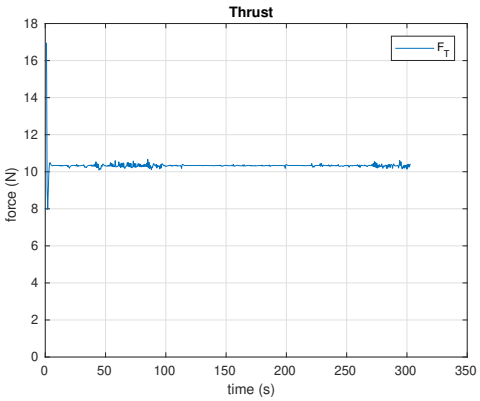
Figure 6.14: Drone attitude variation during the flight.

Rotors speed and thrust

The differences in the rotors angular velocities lead the drone to move in multiple directions with a thrust force given by the equation (4.26). In order to get into more details, the angular velocities and thrust force are represented in Figure 6.15.



(a)



(b)

Figure 6.15: Drone rotors speed (a) and thrust (b) variation during the flight.

The results show that the movement of the drone is smooth during the flight and it is mainly forward flight as $\omega_1 > \omega_3$. The instants around the $125s$ and $200s$, that reveal an almost constant rotors speed, are coincident with the close to zero pitch orientation, which demonstrates that the drone was hovering without moving forward. These events are correlated with the two sharped movements of the target between the second and the third obstacles (wall and house in Figure 6.13).

Target in the image plane

The images captured by the camera allow inferring where the target is in the drone body frame.

The target area, with a goal value of 900 px, represents indirectly the distance between the target and the drone. The target area during the flight is represented in Figure 6.16.

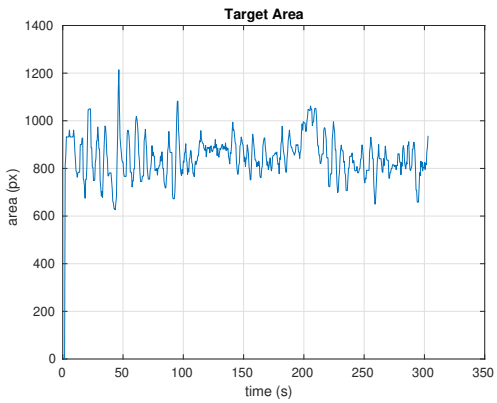
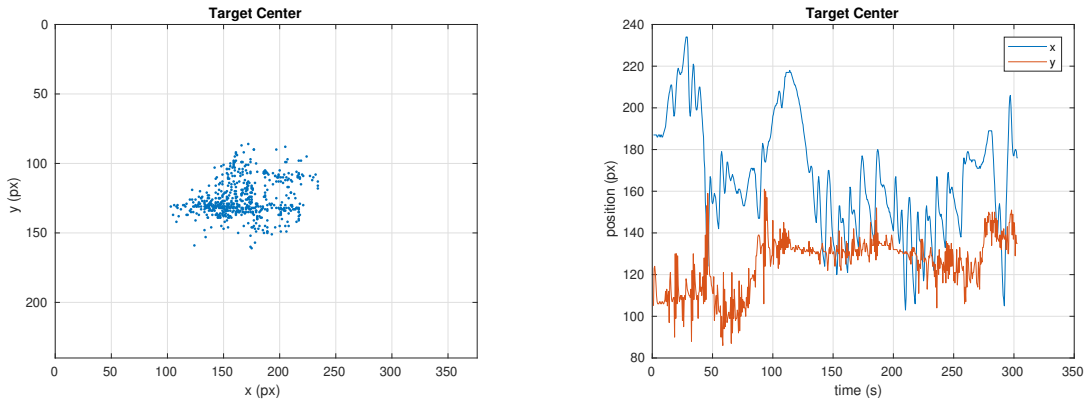


Figure 6.16: Target area variation during the flight.

The results show that the target area is somewhere around 900 px. Moreover, around $125s$ and $200s$ the target size remains relatively constant (or is larger than 900 px) which reveals that the target movement was roughly parallel to the image plane.

Furthermore, the target center information indicates the drone heading to the target during flight. The temporal evolution of this information shows how much the target and drone movements are aligned and it is displayed in Figure 6.17.



(a) Target Center in the image plane.

(b) x and y coordinates evolution.

Figure 6.17: Target Center evolution in the image during flight.

The results show that the implemented strategy can accurately track and maintain the target centered in the image plane without it approaching the boundary of the image plane.

Computational time

Finally, the last parameter to be analyzed was the computational cost. Both the vision and the MPC iteration time were extracted and the results are in Figure 6.18.

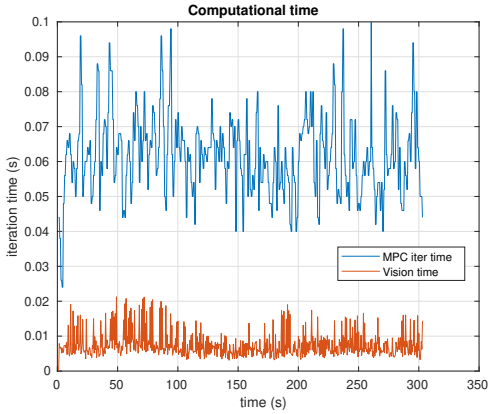


Figure 6.18: Computational cost for both MPC and Vision iterations.

From Figure 6.18, it is clear that the vision algorithm and an MPC real-time iteration are both fast with a computational time smaller than $0.1s$. The vision algorithm results are due to the usage of ORB features that contribute to a fast matching between frames. The MPC iteration result represents a solution of the problem in equation (4.32), where there is an optimization problem and constraints that have to be fulfilled. Moreover, it should be noticed that the size of the state and the number of control inputs are also important in the duration of the computational time.

One of the specific aspects of the implemented strategy is that a full space reconstruction is not needed. The strategy relies on considering obstacles, from the vision algorithm, in each iteration. This means that, in every iteration, the obstacles are derived from the TTC calculations. However, other strategies could be used, such as, space reconstruction strategies that enable the drone to perceive in full the surrounding environment. A comparison between the vision strategy used and some of the existing reconstruction strategies, in terms of average iteration time, is displayed in Table 6.2. The associated time cost of the proposed vision strategy is presented for the final experiment (denoted by sim) and for the experiment described in Section 5.6.2 (denoted by real).

Table 6.2: Iteration average computational time (iter) and standard deviation (std) in different vision strategies.

Strategy	Resolution (px)	iter (ms)	std (ms)	System
present work (sim)	752 x 480	7.6	3.7	Intel Core i5-8250U CPU 4 cores @ 1.60GHz, 8GB RAM
present work (real)	1920 x 1080	26.9	2.4	Intel Core i5-8250U CPU 4 cores @ 1.60GHz, 8GB RAM
Mono ORB-SLAM [71]	512 x 384	496	218	Intel Core i7-4700MQ CPU 4 cores @ 2.40GHz, 8GB RAM
Stereo LSD-SLAM [49]	620 x 184	70.5	-	Intel Core i7-4900MQ CPU 4 cores @ 2.40GHz

The first thing to notice is that the four strategies were tested in different conditions, with different systems and different image resolutions. It is also relevant to remember that, the strategy used in both simulation and real video, uses a region of interest of the image. Although both Monocular ORB-SLAM [71] and Stereo LSD-SLAM [49] are not used to avoid obstacles, the results show that the monocular solution is not adequate to a real-time implementation while the stereo solution is faster and can integrate a real-time application to avoid obstacles. Nonetheless, it is notable that the strategy used in this work has a smaller computational cost than the two presented strategies.

6.2 Limitations and Proposed solutions

The performed experiments reveal that the obstacle avoidance implementation has some restraints in the operating conditions. Some of the factors represent weaknesses associated with the external sensors used while others are related to the proposed method.

The most relevant limitation is the inability of dealing with the texture-less obstacles. Since the strategy relies on a monocular camera to perceive if there are obstacles near, the presence of features are crucial for the algorithm compute the TTC and generate the best commands. A possible solution to overcome this problem is to use additional hardware such as a sonar or a LIDAR to obtain depth information and fuse it with the image information. An alternative approach is to replace the monocular camera with a RGBD camera that enables registration of the 3D world by extracting depth information.

Another limitation is the accuracy in the obstacle position estimation. The relative position of the obstacle is given by the center of the danger zone and the TTC gives depth information. From this, restraints arise, such as the TTC imperfect precision that leads to consider an obstacle closest or farthest from the drone. A relevant drawback happens when the features of an obstacle barely fill a zone and are in one specific position of this, leading to consider a potential field in that zone, as in the miss occurrence in tree avoidance situation in Section 6.1.3. Figure 6.19 exemplifies the situation. It is clear that zone

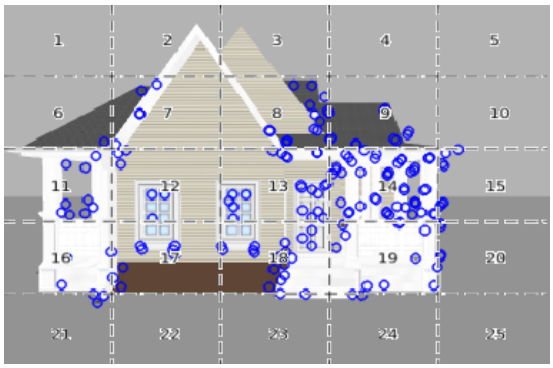


Figure 6.19: Image zones and respective features.

15 has some features, therefore, the algorithm will consider a potential field in its center, however, it is visible that it is a reliable zone. It is worth recalling that the strategy requires a minimum number of features in each zone to attribute an obstacle to them, hence, areas 23 and 24 are not considered occupied.

The computational cost is another limitation inherent to a real-time solution. In this thesis, the solution consists on analyzing a region of interest, reducing the examined pixels in each iteration. The number of MPC iterations in each new trajectory calculation is also determinant to increase or decrease the computational time. The solution to this problem consists in a balance between the computational cost and the value of the cost function to guarantee a continuous and satisfactory solution.

The aggressive maneuvers are another factor that constraints the strategy. Since the camera is rigidly attached to the drone, the aggressive maneuvers may lead the drone to lose the target and compute incorrect trajectories to solve the problem. Possible approaches consist in guaranteeing that the test environment and the target movement lead the drone to execute fluid maneuvers or in using a camera stabilizer to ensure minimal oscillations in the image.

The time-to-collision approach also has vulnerabilities due to aggressive maneuvers or steepest yaw variations. This strategy is only correctly performed with forward flights that enable correct TTC computations between images. The proposed solution consists of ignoring the TTC computations where the yaw rotation is bigger than 3 degrees because these lead to incorrect results.

Another limitation is the target detection method. The algorithm, described in Section 4.3.1, depends on the color to identify the target, so, the environment must be free of this color. A possible solution to

this is to include a QR Code in the target, enabling its tracking during the flight.

Lastly, there is the problem of the target occlusion, when an obstacle is between the drone and the target. This aspect is out of the scope of this work, nonetheless, it is indisputable that it is a limitation in the present solution. A proposed solution can be to perform a wide trajectory to see the target or using an algorithm that can predict the position of the target from its movement in the image.

Chapter 7

Conclusions

The goal of this thesis is to develop a real-time obstacle avoidance system for a drone while following a target using a monocular camera and an IMU. The project was developed with an MPC controller, implemented to generate commands according to the TTC computations, the IMU information, and the position and size of the target in the image plane. Due to the complexity of the system, the work was divided in different stages (vision method and drone control strategy) in order to properly verify and ensure both feasibility and performance.

In the first stage, since the camera do not provide depth information, it was necessary to find a solution that enables the perception of the surroundings of the drone. Strategies as structure from motion or SLAM were considered however, the interest in a solution based on image information that does not reconstruct the 3D world led to a reactive strategy based on TTC computations that identify the danger zones in the image as detailed in Section 4.3.

Next, it was necessary to develop a control strategy that generates commands leading to a flight that follows a target and avoids the obstacles. The strategy is described in Section 4.4 and relies on a cascade control approach designed with an MPC controller as an outer loop that generates controls and an attitude controller as an inner loop that computes the rotors speed. The MPC controller generates potential fields that repel the drone from the danger zones by generating safe trajectories far from the obstacles.

Lastly, the system was evaluated in the simulation environment Gazebo. The results show that the drone can accurately avoid obstacles while following a target. However, it is also indisputable that the problem and the strategy itself have some implicit limitations such as the absence of texture in the image. This condition is visible in the results presented in Section 6.1.3 that show the differences in drone behavior in two situations with a different texture.

From a global perspective, the proposed system fulfilled the main purposes of the dissertation. The

system is able to, in real-time, track a target, understand the world in front of the drone due to TTC computations, avoid obstacles, and solve an MPC problem leading to a generated safe trajectory. The scale problem, inherent to a monocular camera, was solved with the TTC approach. The TTC method is scale invariant, allows to avoid the obstacles without explicit depth information and, although the procedure is not flawless, it shows that is possible to avoid obstacles with monocular camera data and no knowledge in advance of the 3D world.

7.1 Future Work

Concerning future work, several aspects can be experimented and improved.

The first point is the usage of a monocular camera. A depth sensor can be used, and for that, some work has to be done to merge the data from both sensors and together exclude the problem of lack of texture. In the vision strategy, different approaches can be performed. The division of the image into more zones lead to more detailed information about the possible obstacles. However, this may lead to identifying one or two danger features per zone, originating a bigger uncertainty in the TTC computations. A different feature extraction method can be used, however, the computational cost has to be considered.

In terms of the implementation, an alternative to the current target detection method is using a strategy that is not color-dependent. Also, an algorithm that 'searches' the target when it is occluded, is an interest and obvious step that must be done to increase the robustness in this strategy.

Regarding the experimental results, an important goal is to test the algorithm in a real environment. First, the TTC validation has to be performed in data from a drone flight, followed by a tracking and an obstacle avoidance test. Then, if those stages were successful, an MPC can be used as a low-level controller where the generated commands will be the rotors speed that controls the movement of the drone.

Bibliography

[1] C. Goerzen, Z. Kong, and B. Mettler. A survey of motion planning algorithms from the perspective of autonomous UAV guidance. *Journal of Intelligent and Robotic Systems*, 57(1-4):65, 2010.

[2] P. Rudol and P. Doherty. Human body detection and geolocalization for UAV search and rescue missions using color and thermal imagery. In *2008 IEEE aerospace conference*, pages 1–8. IEEE, 2008.

[3] J. W. Waite, T. Gudmundsson, and D. Gargov. UAV power line position and load parameter estimation, May 19 2015. US Patent 9,037,314.

[4] G. Saggiani, F. Persiani, A. Ceruti, P. Tortora, E. Troiani, F. Giuletti, S. Amici, M. Buongiorno, G. Distefano, G. Bentini, et al. A UAV system for observing volcanoes and natural hazards. *AGUFM*, 2007:GC11B–05, 2007.

[5] M. Kulbacki, J. Segen, W. Knieć, R. Klempous, K. Kluwak, J. Nikodem, J. Kulbacka, and A. Serester. Survey of drones for agriculture automation from planting to harvest. In *2018 IEEE 22nd International Conference on Intelligent Engineering Systems (INES)*, pages 000353–000358. IEEE, 2018.

[6] B. Albaker and N. Rahim. A survey of collision avoidance approaches for unmanned aerial vehicles. In *2009 international conference for technical postgraduates (TECHPOS)*, pages 1–7. IEEE, 2009.

[7] H. Lee and H. J. Kim. Trajectory tracking control of multirotors from modelling to experiments: A survey. *International Journal of Control, Automation and Systems*, 15(1):281–292, 2017.

[8] O. Esrafilian and H. D. Taghirad. Autonomous flight and obstacle avoidance of a quadrotor by monocular slam. In *2016 4th International Conference on Robotics and Mechatronics (ICROM)*, pages 240–245. IEEE, 2016.

[9] S. Hrabar. 3d path planning and stereo-based obstacle avoidance for rotorcraft UAVs. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 807–814. IEEE, 2008.

[10] C. E. Garcia, D. M. Prett, and M. Morari. Model predictive control: theory and practice—a survey. *Automatica*, 25(3):335–348, 1989.

[11] M. Islam, M. Okasha, and M. Idres. Dynamics and control of quadcopter using linear model predictive control approach. In *IOP Conference Series: Materials Science and Engineering*, volume 270, page 012007. IOP Publishing, 2017.

[12] X. Zhang, X. Li, K. Wang, and Y. Lu. A survey of modelling and identification of quadrotor robot. In *Abstract and Applied Analysis*, volume 2014. Hindawi, 2014.

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

[14] H. Chao, Y. Cao, and Y. Chen. Autopilots for small unmanned aerial vehicles: a survey. *International Journal of Control, Automation and Systems*, 8(1):36–44, 2010.

[15] J. L. Guzmán, T. Hägglund, K. J. Åström, S. Dormido, M. Berenguel, and Y. Piguet. Understanding pid design through interactive tools. *IFAC Proceedings Volumes*, 47(3):12243–12248, 2014.

[16] U. Eren, A. Prach, B. B. Koçer, S. V. Raković, E. Kayacan, and B. Açıkmeşe. Model predictive control in aerospace systems: Current state and opportunities. *Journal of Guidance, Control, and Dynamics*, 40(7):1541–1566, 2017.

[17] C. Jones. *Lectures, Model Predictive Control*. École polytechnique fédérale de Lausanne, ME-425 Model Predictive Control, 2012.

[18] *Acado toolkit, nonlinear model predictive control and moving horizon estimation.* [Online]. Available: acado.github.io/cgt_overview.html.

[19] E. B. L. Montero. *Use of Multivariate Statistical Methods for Control of Chemical Batch Processes*. PhD thesis, The University of Manchester (United Kingdom), 2016.

[20] B. Houska, H. Ferreau, M. Vukov, and R. Quirynen. Acado toolkit user's manual, 2009–2013. URL `http://www.acadotoolkit.org`.

[21] D. Ariens, B. Houska, and H. Ferreau. Acado for matlab user's manual, 2010–2011. URL `http://www.acadotoolkit.org`.

[22] M. Vukov, A. Domahidi, H. J. Ferreau, M. Morari, and M. Diehl. Auto-generated algorithms for nonlinear model predicitive control on long and on short horizons. In *Proceedings of the 52nd Conference on Decision and Control (CDC)*, 2013.

[23] H. J. Ferreau, T. Kraus, M. Vukov, W. Saeys, and M. Diehl. High-speed moving horizon estimation based on automatic code generation. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 687–692. IEEE, 2012.

[24] K. Kitani. *Lecture slides, Camera Matrix*. Carnegie Mellon University, 16-385 Computer Vision, Spring 2017.

[25] J. S. Marques and J. Santos-Victor. *Lecture slides, Optical Flow*. Instituto Superior Técnico, Image Processing and Vision, 2019.

[26] Ş. Işık. A comparative evaluation of well-known feature detectors and descriptors. *International Journal of Applied Mathematics, Electronics and Computers*, 3(1):1–6, 2014.

[27] E. Rublee, V. Rabaud, K. Konolige, and G. R. Bradski. ORB: An efficient alternative to sift or surf. In *ICCV*, volume 11, page 2, 2011.

[28] S. A. K. Tareen and Z. Saleem. A comparative analysis of SIFT, SURF, KAZE, AKAZE, ORB, and BRISK. In *2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET)*, pages 1–10. IEEE, 2018.

[29] P. F. Alcantarilla and T. Solutions. Fast explicit diffusion for accelerated features in nonlinear scale spaces. *IEEE Trans. Patt. Anal. Mach. Intell*, 34(7):1281–1298, 2011.

[30] C. Wang, W. Liu, and M. Q.-H. Meng. Obstacle avoidance for quadrotor using improved method based on optical flow. In *2015 IEEE International Conference on Information and Automation*, pages 1674–1679. IEEE, 2015.

[31] K. Nonami. Prospect and recent research & development for civil use autonomous unmanned aircraft as UAV and MAV. *Journal of system Design and Dynamics*, 1(2):120–128, 2007.

[32] Netanel Malka. The current development and application of autonomous UAVs. URL `https://www.skyhopper.biz/autonomous-uavs/`. [Online; accessed May-2020].

[33] M. Erdelj, M. Król, and E. Natalizio. Wireless sensor networks and multi-UAV systems for natural disaster management. *Computer Networks*, 124:72–86, 2017.

[34] T. T. Mac, C. Copot, T. T. Duc, and R. De Keyser. Ar. drone UAV control parameters tuning based on particle swarm optimization algorithm. In *2016 IEEE International Conference on Automation, Quality and Testing, Robotics (AQTR)*, pages 1–6. IEEE, 2016.

[35] A. Fotouhi, M. Ding, and M. Hassan. Understanding autonomous drone maneuverability for internet of things applications. In *2017 IEEE 18th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–6. IEEE, 2017.

[36] Y. Lu, Z. Xue, G.-S. Xia, and L. Zhang. A survey on vision-based UAV navigation. *Geo-spatial information science*, 21(1):21–32, 2018.

[37] H. Eisenbeiss et al. A mini unmanned aerial vehicle (UAV): system overview and image acquisition. *International Archives of Photogrammetry. Remote Sensing and Spatial Information Sciences*, 36 (5/W1):1–7, 2004.

[38] J. Koci, J. X. Leon, B. Jarihani, R. C. Sidle, S. N. Wilkinson, and R. Bartley. Strengths and limitations of UAV and ground-based structure from motion photogrammetry in a gullied savanna catchment. 2017.

[39] G. Cai, J. Dias, and L. Seneviratne. A survey of small-scale unmanned aerial vehicles: Recent advances and future development trends. *Unmanned Systems*, 2(02):175–199, 2014.

[40] C. Giannì, M. Balsi, S. Esposito, and P. Fallavollita. Obstacle detection system involving fusion of multiple sensor technologies. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 42:127, 2017.

[41] V. Behzadan and A. Munir. Adversarial reinforcement learning framework for benchmarking collision avoidance mechanisms in autonomous vehicles. *IEEE Intelligent Transportation Systems Magazine*, 2019.

[42] S. García, M. E. López, R. Barea, L. M. Bergasa, A. Gómez, and E. J. Molinos. Indoor slam for micro aerial vehicles control using monocular camera and sensor fusion. In *2016 International Conference on Autonomous Robot Systems and Competitions (ICARSC)*, pages 205–210. IEEE, 2016.

[43] E. López, S. García, R. Barea, L. Bergasa, E. Molinos, R. Arroyo, E. Romera, and S. Pardo. A multisensorial simultaneous localization and mapping (slam) system for low-cost micro aerial vehicles in gps-denied environments. *Sensors*, 17(4):802, 2017.

[44] S. Weiss, D. Scaramuzza, and R. Siegwart. Monocular-slam–based navigation for autonomous micro helicopters in gps-denied environments. *Journal of Field Robotics*, 28(6):854–874, 2011.

[45] S. Ross, N. Melik-Barkhudarov, K. S. Shankar, A. Wendel, D. Dey, J. A. Bagnell, and M. Hebert. Learning monocular reactive UAV control in cluttered natural environments. In *2013 IEEE international conference on robotics and automation*, pages 1765–1772. IEEE, 2013.

[46] K. McGuire, G. De Croon, C. De Wagter, K. Tuyls, and H. Kappen. Efficient optical flow and stereo vision for velocity estimation and obstacle avoidance on an autonomous pocket drone. *IEEE Robotics and Automation Letters*, 2(2):1070–1076, 2017.

[47] V. Grabe, H. H. Bülthoff, and P. R. Giordano. On-board velocity estimation and closed-loop control of a quadrotor UAV based on optical flow. In *2012 IEEE International Conference on Robotics and Automation*, pages 491–497. IEEE, 2012.

[48] P. Gao, D. Zhang, Q. Fang, and S. Jin. Obstacle avoidance for micro quadrotor based on optical flow. In *2017 29th Chinese Control And Decision Conference (CCDC)*, pages 4033–4037. IEEE, 2017.

[49] J. Engel, J. Stückler, and D. Cremers. Large-scale direct slam with stereo cameras. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1935–1942. IEEE, 2015.

[50] J. Engel, T. Schöps, and D. Cremers. Lsd-slam: Large-scale direct monocular slam. In *European conference on computer vision*, pages 834–849. Springer, 2014.

[51] M. R. U. Saputra, A. Markham, and N. Trigoni. Visual slam and structure from motion in dynamic environments: A survey. *ACM Computing Surveys (CSUR)*, 51(2):37, 2018.

[52] O. Özyeşil, V. Voroninski, R. Basri, and A. Singer. A survey of structure from motion*. *Acta Numerica*, 26:305–364, 2017.

[53] T. Mori and S. Scherer. First results in detecting and avoiding frontal obstacles from a monocular camera for micro unmanned aerial vehicles. In *2013 IEEE International Conference on Robotics and Automation*, pages 1750–1757. IEEE, 2013.

[54] H. Bay, T. Tuytelaars, and L. V. Gool. *SURF: Speeded Up Robust Features*. Lecture Notes in Computer Science, vol 3951. Springer, Berlin, Heidelberg, 2006.

[55] S. Thrun, W.Burgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2^{nd} edition, 2006, pp.33-65.

[56] A. Schaub and D. Burschka. Spatio-temporal prediction of collision candidates for static and dynamic objects in monocular image sequences. In *2013 IEEE Intelligent Vehicles Symposium (IV)*, pages 1052–1058. IEEE, 2013.

[57] G. Alenyà, A. Nègre, and J. L. Crowley. *A Comparison of Three Methods for Measure of Time to Contact*. IEEE/RSJ International Conference on Intelligent Robots and Systems, 2009, pp.4565-4570.

[58] D. K. Kim and T. Chen. Deep neural network for real-time autonomous indoor navigation. *arXiv preprint arXiv:1511.04668*, 2015.

[59] G. Kahn, A. Villaflor, V. Pong, P. Abbeel, and S. Levine. Uncertainty-aware reinforcement learning for collision avoidance. *arXiv preprint arXiv:1702.01182*, 2017.

[60] F. Mao-Hui and X.Jun. *Research on Trajectory Planning Algorithm of Unmanned Aerial Vehicle Based on Improved A* Algorithm*. IEEE, International Conference on Computer Systems, Electronics and Control (ICCSEC), 2017.

[61] O. Adiyatov and H. A. Varol. Rapidly-exploring random tree based memory efficient motion planning. In *2013 IEEE international conference on mechatronics and automation*, pages 354–359. IEEE, 2013.

[62] L. Zhu, X. Cheng, and F.-G. Yuan. A 3D collision avoidance strategy for UAV with physical constraints. *Measurement*, 77:40–49, 2016.

[63] C. Goerzen, Z. Kong, and B. Mettler. A survey of motion planning algorithms from the perspective of autonomous uav guidance. *Journal of Intelligent and Robotic Systems*, 57(1-4):65, 2010.

[64] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *IEEE Robotics & Automation Magazine*, 4(1):23–33, 1997.

[65] N. J. Sanket, C. D. Singh, K. Ganguly, C. Fermüller, and Y. Aloimonos. Gapflyt: Active vision based minimalist structure-less gap detection for quadrotor flight. *IEEE Robotics and Automation Letters*, 3(4):2799–2806, 2018.

[66] M. Kamel, M. Burri, and R. Siegwart. Linear vs nonlinear MPC for trajectory tracking applied to rotary wing micro aerial vehicles. *IFAC-PapersOnLine*, 50(1):3463–3469, 2017.

[67] T. Zhang, G. Kahn, S. Levine, and P. Abbeel. Learning deep control policies for autonomous aerial vehicles with mpc-guided policy search. In *2016 IEEE international conference on robotics and automation (ICRA)*, pages 528–535. IEEE, 2016.

[68] Wikipedia contributors. Distance from a point to a line — Wikipedia, the free encyclopedia, 2019. URL `https://en.wikipedia.org/w/index.php?title=Distance_from_a_point_to_a_line&oldid=906504388`. [Online; accessed August-2019].

[69] Johannes Traa. Least-squares intersection of lines. URL `http://cal.cs.illinois.edu/~johannes/research/LS$_line$_intersect.pdf`. [Online; accessed March-2019].

[70] 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. ISBN 978-3-319-26054-9.

[71] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardos. Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, 31(5):1147–1163, 2015.

Appendix A

Image properties

A.1 Target Area and Distance to Target

The drone has to maintain a certain distance to the target while avoiding obstacles. To perceive this distance it is necessary to know the size of the target, the camera parameters, and the image dimensions. Figure 2.6 is reviewed and illustrates the situation:

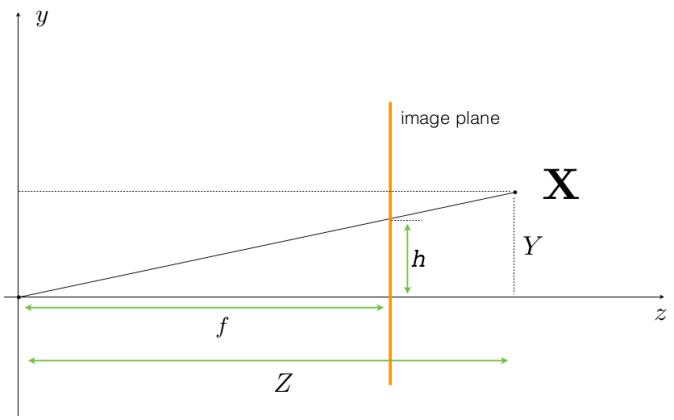


Figure A.1: Relation of a point in image plane to a point in space [24].

Consider that the target has a size Y and assume that it is parallel to the image plane. The distance to the target is given by

$$Z = \frac{Y}{h} f, \tag{A.1}$$

with $f = 240$ is the focal length. Reminding that the target is a box and that the target area is given by the area of the red square in the image plane (see Figure 5.1), the magnitude of h is given by

$$h = \sqrt{\text{TargetArea}}. \tag{A.2}$$

With the previous result, the distance from drone to the target is given by equation (A.3):

$$Z = \frac{Y}{\sqrt{\text{TargetArea}}}\, f = \frac{0.5}{\sqrt{900}}\, 240 = 4\ m, \tag{A.3}$$

where $0.5m$ is the target size Y and 900 px is the pretended target area.

A.2 Center of the target

The position of the target in the image plane is given by the target center. Since the goal is to center the target in the image, due to the distance to target Z (see Appendix A.1), the difference between the image center and the target center and the target size information, it is possible to obtain the desired yaw orientation and height of the drone. By using the relation

$$\psi = tan^{-1}\left(\frac{L}{Z}\right), \tag{A.4}$$

where L is the distance between a point at distance Z of the drone and the center of the target, it is obtained the desired ψ. The height of the drone is corrected in a similar manner, where the height difference between the drone and the center of the target is the value to be minimized.

A.3 Vertical Field-of-View (V)

ROS provides cameras to attach to the drones and to use image data it is only necessary to access the camera topic. However, not all the information is present, as the vertical field-of-view (V). Image dimensions are 752x480 px with an horizontal field-of-view (H) of $2\ rad$. Aspect Ratio is a measure expressed by the relation

$$r = \frac{\text{width}}{\text{heigth}}, \tag{A.5}$$

or by

$$r = \frac{tan(H/2)}{tan(V/2)}. \tag{A.6}$$

From equations (A.5) and (A.6), the vertical field-of-view is obtained:

$$V = 2\ tan^{-1}\left(\frac{tan(H/2)}{r}\right) \simeq 1.56\ rad \simeq 90^{\text{o}}. \tag{A.7}$$

Since it is used a Region of the Interest in the image plane, it is necessary to compute the new V. The region of interest consists in an image with 240 px of height as shown in figure A.2.

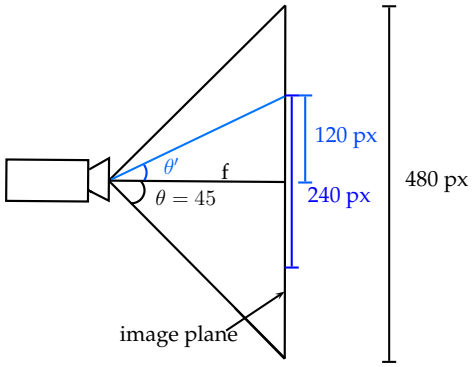


Figure A.2: New vertical FOV calculation.

From inspection of the Figure A.2, with $f = 240$ the new V is now

$$V = 2\,\theta' = 2\,tan\left(\frac{120}{f}\right) \simeq 53^{\text{o}}. \tag{A.8}$$