UNIVERSITAT POLITÈCNICA DE CATALUNYA

Programa de Doctorat:

AUTOMÀTICA, ROBÒTICA I VISIÓ

Tesi Doctoral

GUIDANCE, NAVIGATION AND CONTROL OF MULTIROTORS

Bartomeu Rubí Perelló

Directors de la tesi:
Bernardo Morcego Seix i Ramon Pérez Magrané

Octubre de 2020

To my parents
and to my significant other

Abstract

This thesis presents contributions to the Guidance, Navigation and Control (GNC) systems for multirotor vehicles by applying and developing diverse control techniques and machine learning theory with innovative results. The aim of the thesis is to obtain a GNC system able to make the vehicle follow predefined paths while avoiding obstacles in the vehicle's route. The system must be adaptable to different paths, situations and missions, reducing the tuning effort and parametrisation of the proposed approaches.

The multirotor platform, formed by the Asctec Hummingbird quadrotor vehicle, is studied and described in detail. A complete mathematical model is obtained and a freely available and open simulation platform is built. Furthermore, an autopilot controller is designed and implemented in the real platform.

The control part is focused on the path following problem. That is, following a predefined path in space without any time constraint. Diverse control-oriented and geometrical algorithms are studied, implemented and compared. Then, the geometrical algorithms are improved by obtaining adaptive approaches that do not need any parameter tuning. The adaptive geometrical approaches are developed by means of Neural Networks. To end up, a deep reinforcement learning approach is developed to solve the path following problem. This approach implements the Deep Deterministic Policy Gradient algorithm. The resulting approach is trained in a realistic multirotor simulator and tested in real experiments with success. The proposed approach is able to accurately follow a path while adapting the vehicle's velocity depending on the path's shape.

In the navigation part, an obstacle detection system based on the use of a LIDAR sensor is implemented. A model of the sensor is derived and included in the simulator. Moreover, an approach for treating the sensor data to eliminate the possible ground detections is developed.

The guidance part is focused on the reactive path planning problem. That is, a path planning algorithm that is able to re-plan the trajectory online if an unexpected event, such as detecting an obstacle in the vehicle's route, occurs. A deep reinforcement learning approach for the reactive obstacle avoidance problem is developed. This approach implements the Deep Deterministic Policy Gradient algorithm. The developed deep reinforcement learning agent is trained and tested in the realistic simulation platform. This agent is combined with the path following agent and the rest of the elements developed in the thesis obtaining a GNC system that is able to follow different types of paths while avoiding obstacle in the vehicle's route.

Resumen

Esta tesis doctoral presenta varias contribuciones relacionas con los sistemas de Guiado, Navegación y Control (GNC) para vehículos multirotor, aplicando y desarrollando diversas técnicas de control y de machine learning con resultados innovadores. El objetivo principal de la tesis es obtener un sistema de GNC capaz de dirigir el vehículo para que siga una trayectoria predefinida mientras evita los obstáculos que puedan aparecer en el recorrido del vehículo. El sistema debe ser adaptable a diferentes trayectorias, situaciones y misiones, reduciendo el esfuerzo realizado en el ajuste y la parametrización de los métodos propuestos.

La plataforma experimental, formada por el cuadricóptero Asctec Hummingbird, se estudia y describe en detalle. Se obtiene un modelo matemático completo de la plataforma y se desarrolla una herramienta de simulación, la cual es de código libre. Además, se diseña un controlador autopilot, el cual es implementado en la plataforma real.

La parte de control está enfocada en el problema de path following. En este problema, el vehículo debe seguir una trayectoria predefinida en el espacio tridimensional sin ninguna restricción temporal. Se estudian, implementan y comparan varios algoritmos de control y geométricos de path following. Luego, se mejoran los algoritmos geométricos usando redes neuronales para convertirlos en algoritmos adaptativos. Para finalizar, se desarrolla un método de path following basado en técnicas de aprendizaje por refuerzo profundo (deep reinforcement learning). Este método implementa el algoritmo Deep Deterministic Policy Gradient. El agente inteligente resultante es entrenado en un simulador realista de multirotores y validado en la plataforma experimental real con éxito. Los resultados muestran que el agente es capaz de seguir de forma precisa la trayectoria de referencia adaptando la velocidad del vehículo según la curvatura del recorrido.

En la parte de navegación se implementa un sistema de detección de obstáculos basado en el uso de un sensor LIDAR. Se deriva un modelo del sensor y este se incluye en el simulador. Además, se desarrolla un método para tratar las medidas del sensor para eliminar las posibles detecciones del suelo.

En cuanto a la parte de guiado, está focalizada en el problema de reactive path planning. Es decir, un algoritmo de planificación de trayectoria que es capaz de re-planear el recorrido del vehículo al instante si ocurre algún evento inesperado, como lo es la detección de un obstáculo en el recorrido del vehículo. Se desarrolla un método basado en aprendizaje por refuerzo profundo para la evasión de obstáculos. Este implementa el algoritmo Deep Deterministic Policy Gradient. El agente de aprendizaje por refuerzo se entrena y valida en un simulador de multirotors realista.

Este agente se combina con el agente de path following y el resto de elementos desarrollados en la tesis para obtener un sistema GNC capaz de seguir diferentes tipos de trayectorias evadiendo los obstáculos que estén en el recorrido del vehículo.

Resum

Aquesta tesi doctoral presenta diverses contribucions relaciones amb els sistemes de Guiatge, Navegació i Control (GNC) per a vehicles multirrotor, aplicant i desenvolupant diverses tècniques de control i de machine learning amb resultats innovadors. L'objectiu principal de la tesi és obtenir un sistema de GNC capaç de dirigir el vehicle perquè segueixi una trajectòria predefinida mentre evita els obstacles que puguin aparèixer en el recorregut del vehicle. El sistema ha de ser adaptable a diferents trajectòries, situacions i missions, reduint l'esforç realitzat en l'ajust i la parametrització dels mètodes proposats.

La plataforma experimental, formada pel cuadricòpter Asctec Hummingbird, s'estudia i es descriu en detall. S'obté un model matemàtic complet de la plataforma i es desenvolupa una eina de simulació, la qual és de codi lliure. A més, es dissenya un controlador autopilot i s'implementa en la plataforma real.

La part de control està enfocada al problema de path following. En aquest problema, el vehicle ha de seguir una trajectòria predefinida en l'espai sense cap tipus de restricció temporal. S'estudien, s'implementen i es comparen diversos algoritmes de control i geomètrics de path following. Després, es milloren els algoritmes geomètrics usant xarxes neuronals per convertirlos en algoritmes adaptatius. Per finalitzar, es desenvolupa un mètode de path following basat en tècniques d'aprenentatge per reforç profund (deep Reinforcement learning). Aquest mètode implementa l'algoritme Deep Deterministic Policy Gradient. L'agent intel·ligent resultant és entrenat en un simulador realista de multirotors i validat en la plataforma experimental real amb èxit. Els resultats mostren que l'agent és capaç de seguir de forma precisa la trajectòria de referència adaptant la velocitat del vehicle segons la curvatura del recorregut.

A la part de navegació, s'implementa un sistema de detecció d'obstacles basat en l'ús d'un sensor LIDAR. Es deriva un model del sensor i aquest s'inclou en el simulador. A més, es desenvolupa un mètode per tractar les mesures del sensor per eliminar les possibles deteccions del terra.

Pel que fa a la part de guiatge, aquesta està focalitzada en el problema de reactive path planning. És a dir, un algoritme de planificació de trajectòria que és capaç de re-planejar el recorregut del vehicle a l'instant si algun esdeveniment inesperat ocorre, com ho és la detecció d'un obstacle en el recorregut del vehicle. Es desenvolupa un mètode basat en aprenentatge per reforç profund per l'evasió d'obstacles. Aquest mètode implementa l'algoritme Deep Deterministic Policy Gradient. L'agent d'aprenentatge per reforç s'entrena i valida en un simulador de multirotors realista. Aquest agent es combina amb l'agent de path following i la resta d'elements desenvolupats en

la tesi per obtenir un sistema GNC capaç de seguir diferents tipus de trajectòries, evadint els obstacles que estiguin en el recorregut del vehicle.

Acknowledgements

First of all, I would like to thank my supervisors, Prof. Bernardo Morcego and Prof. Ramon Pérez, for their valuable advice, dedication, corrections and support throughout the work of my thesis. For believing in me, and for giving me suggestions and ideas whenever I needed them. Without their contribution this thesis wouldn't have been possible.

A special thanks to my parents for their constant support. They have always provided me with the motivation, the ambition and the resources to make this thesis possible, paving the way towards this goal since I was a little kid.

Thanks to my significant other, Kimberly, for her constant support, not only during the work on this thesis, but also in every other part of my life. She transmitted me the strength in the toughest moments of this PhD. Thank you.

I would like to thank to the research group CS2AC to bring me the opportunity to conduct this thesis. They showed that, more than a research group, they seem a family. In this group I had the opportunity to meet many wonderful people who have become my friends: Damiano, Helem, Jean Carlo, Carlos, Juli, and many more. In particular, I want to thank Adrian and Jaime for their dedication and patience in all the flight experiments that we performed together, and Julen for his constant help, recommendations and support during the thesis.

I want to express my gratitude to all of my friends who showed interest and admiration for my thesis. Especially, I would like to thank a group of friends which always have been there and who provided me with a phrase that I will never forget; "where your thoughts go, your steps go" (non gogoa, han zagoa), which perfectly represents the soul of this thesis.

Bartomeu Rubí
Terrassa, 20th October of 2020

Contents

List of Figures

List of Tables

Acronyms

ADRC	Active Disturbance Rejection Control
APF	Artificial Potential Field
BN	Batch Normalisation
BS	Backstepping
CC	Carrot-Chasing
CCN	Convolutional Neural Networks
DAPF	Dynamic Artificial Potential Field
DDPG	Deep Deterministic Policy Gradient
DF	Differential Flatness
DQN	Deep Q-Network
DPG	Deterministic Policy Gradient
DRL	Deep Reinforcement Learning
EKF	Extended Kalman Filter
ESC	Electronic Speed Controller
FL	Feedback Linearisation
FOV	Field-Of-View
GNC	Guidance, Navigation and Control
GPS	Global Positioning System
HOSM	High-Order Sliding Mode
HSA	Heuristic Search Algorithm
IGC	Integrated Guidance and Control
IMU	Inertial Measurement Unit

LIDAR	Light Detection And Ranging
LOS	Line-of-sight
LQG	Linear Quadratic Gaussian
LQR	Linear Quadratic Regulator
LSTM	Long Short-Term Memory
MAE	Mean Absolute Error
MILP	Mixed Integer Linear Programming
MIMO	Multiple-input Multiple-output
MPC	Model Predictive Control
MRAC	Model Reference Adaptive Control
NLGL	Nonlinear Guidance Law
NN	Neural Network
OA	Obstacle Avoidance
ORCA	Optimal Reciprocal Collision Avoidance
PF	Path Following
PID	Proportional-Integral-Derivative
PLOS	Pure Pursuit and Line-of-sight
PP	Pure Pursuit
PPN	Proportional Navigation Guidance
PRM	Probabilistic Road Maps
RADAR	Radio Detection And Ranging
R/C	Radio Control
RDPG	Recurrent Deterministic Policy Gradient
ReLU	Rectified Linear Unit
RHC	Receding Horizon Control
RL	Reinforcement Learning
RNN	Recurrent Neural Network
ROS	Robot Operating System
RRT	Rapidly Exploring Random Trees

SDK	Software Development Kit
SGC	Separated Guidance and Control
SLAD	Safe Landing Area Detection
SLAM	Simultaneous Location And Mapping
SMAP	Simultaneous Mapping And Planning
SMC	Sliding Mode Control
SOSM	Second-Order Sliding Mode
TD	Temporal-Difference error
TS	Trajectory Shaping
UAS	Unmanned Aerial System
UAV	Unmanned Aerial Vehicle
UGV	Unmanned Ground Vehicle
URDF	Unified Robot Description Format
VF	Vector Field
VTP	Virtual Target Point
VTV	Virtual Target Vehicle

Symbols

c_D	Drag coefficient
c_T	Thrust coefficient
d_m	Distance from a motor to the c.o.g
d_{min}	Distance from the vehicle to the path
$\boldsymbol{F}$	Thrust forces in the body frame
$\boldsymbol{F}_d$	Drag forces in the body frame
$\boldsymbol{F}_w$	Wind disturbance forces in the body frame
${}^w\boldsymbol{g}$	Gravity expressed in the body frame
$\mathbf{H}$	Matrix that relates body angular velocity with Euler angle's rate of change
$\mathbf{J}$	Mass moment of inertia matrix
J_m	Inertia of each motor's rotating components
k	Curvature of the path
k_Q	Torque constant
k_T	Thrust constant
L	Parameter of *NLGL* algorithm
L_{opt}	Optimal value of the parameter of *NLGL* algorithm
m	Mass of the quadrotor
p	Angular velocity, x axis
$\boldsymbol{p}$	Position vector
$\boldsymbol{p}_d$	Desired path vector
q	Angular velocity, y axis
r	Angular velocity, z axis
$\mathbf{R}$	Rotational matrix from body to the world frame (zyx sequence)
$\mathbf{R}_d$	Desired rotational matrix from body to the world frame (zyx sequence)
t	Time

T	Total thrust force
T_d	Desired total thrust force
t_f	Final time
T_i	Thrust force generated by the ith motor in the z-body axis
u	Linear velocity, x body axis
${}^w u$	Linear velocity, x world axis
u_{cmd}	Linear velocity command, x body axis
u_z	Thurst input normalised from 0 to 200
u_ϕ	Roll input normalised from 0 to 200
u_θ	Pitch input normalised from 0 to 200
u_ψ	Yaw input normalised from 0 to 200
$\boldsymbol{v}$	Velocity vector in body frame
${}^w\dot{\boldsymbol{v}}$	Velocity vector in world frame
v	Linear velocity, y axis
${}^w v$	Linear velocity, y world axis
v_{cmd}	Linear velocity command, y body axis
V_{ref}	Velocity reference (module)
$\boldsymbol{v}_w$	Wind velocity vector
w	Linear velocity, z axis
${}^w w$	Linear velocity, z world axis
$\boldsymbol{x}$	State vector
x	World position in x axis
x_d	Desired path, x axis
y	World position in y axis
y_d	Desired path, y axis
x	World position in z axis
z_cmd	Command of position in z axis
z_d	Desired path, z axis
γ	Virtual arc of the path
$\gamma_{d_{min}}$	Value of virtual arc at the closest point to the vehicle
γ_f	Final value of virtual arc of the path
γ_{VTP}	Value of virtual arc at the virtual target point (VTP)
δ	Parameter of CC algorithm

δ_{opt}	Optimal value of the parameter of CC algorithm
θ	Pitch Euler angle
θ_{cmd}	Pitch angle command
ρ	Air mass density
$\boldsymbol{\tau}$	Moments generated on the body frame
τ_m	Motor time constant
$\boldsymbol{\tau}_{gyr}$	Moments generated by gyroscopic effects in body frame
$\tau_{gyr,x}$	Moment generated by gyroscopic effects in body frame, x axis
$\tau_{gyr,y}$	Moment generated by gyroscopic effects in body frame, y axis
$\tau_{gyr,z}$	Moment generated by gyroscopic effects in body frame, z axis
τ_θ	Torque applied on y body axis
τ_ϕ	Torque applied on x body axis
τ_ψ	Torque applied on z body axis
ϕ	Roll Euler angle
ϕ_{cmd}	Roll angle command
$\boldsymbol{\Phi}$	Euler angles vector
ψ	Yaw Euler angle
ψ_{cmd}	Yaw angle command
$\boldsymbol{\omega}$	Angular velocity vector
ω_{mi}	Rotational velocity of motor i
$\omega_{n,ri}$	Rotational velocity reference of motor i, normalised from 0 to 200
ω_{r1}	Rotational velocity reference of motor i

Chapter 1

Introduction

1.1 Motivation

In recent years, the growing interest to develop fully autonomous aerial vehicles, known as Unmanned Aerial Vehicles (UAVs), has seen a civil market demand increase compared to its military applications. The increasing number of tasks and applications that UAVs can perform is well known. The interest in automating these applications drives a continuous progress both in the artificial intelligence and control areas.

Out of all UAVs, multirotors stand out for their good manoeuvrability and mechanical simplicity. Throughout the education of the thesis' author, focused on electronic engineering and automation, the research on this type of vehicles has become one of his main interests. This platform permits to study, develop and implement approaches on diverse areas such as control, machine learning, artificial vision or state estimation, which are aligned with the author's background. While studying physics, automatic control or artificial intelligence subjects, the UAV has always been a baseline system to study and apply the author's acquired knowledge. His final master's project was focused on the modelling and control of a coaxial helicopter UAV. Furthermore, the radio controlled aerial vehicles have for long been one of the major author's passions, having expertise on piloting multirotors and small helicopters, as well as a wide knowledge on their mechanical and electronic parts.

The Research Center for Supervision, Safety and Automatic Control (RS2AC), the research group were this thesis was carried on, has a large expertise on the research on UAVs and other type of autonomous vehicles. Furthermore, the group has different indoor and outdoor UAV experimental platforms that become a great opportunity for implementing and testing the work developed in this thesis. Moreover, the group is involved in projects in which the research focus is put on security and control of autonomous vehicles.

The motivation of this thesis emerges from both the interest of the group on the UAV research and its practical implementation as well as the author's interests on this type of vehicles and in the research and implementation of novel control and machine learning algorithms.

1.2 Thesis objectives

The principal objective of this PhD thesis is to study, develop and apply innovative control techniques and machine learning algorithms to implement a Guidance, Navigation and Control system for a multirotor vehicle. This system must be able to make the vehicle to follow predefined paths while avoiding obstacles in the vehicle's route. The aim of the thesis is to obtain a highly autonomous system that can be adapted to different paths, situations and missions, reducing the tuning effort and parametrisation of the proposed approaches.

Secondary objectives involved in the generation of the Guidance, Navigation and Control system are:

- Control Structure: Define a structure of the Guidance, Navigation and Control system that includes the required modules and their information flows for addressing the challenges derived from the defined problem.
- Path Following: Study the path following problem of a multirotor vehicle. Implement and compare state-of-the-art path following algorithms for multirotor vehicles and/or adapt algorithms applied to other vehicles. Improve the algorithms or develop an approach that will be proposed as solution. The proposed approach must be straightforwardly adaptable to different reference paths without requiring any additional tuning of the algorithm parameters. Moreover, it must dynamically adapt the velocity of the vehicle to the shape of the path, anticipating the curves to come.
- Path Planning & Obstacle Avoidance: Study the problem of planning a route online to follow a predefined path while avoiding possible obstacles and propose a reactive path planning approach. The proposed approach must integrate the information of the state estimator, the perception systems and the given reference path to generate the reference route. Also, it must be capable of replanning the route online when an unexpected event, such as the detection of an obstacle, occurs. This approach must be integrated with the path following algorithm.
- Obstacle Detection: Implement a perception system capable to detect obstacles in the vehicle's route. This system must be installed in the real platform and the perceived information must be treated and adapted to send it to the reactive path planning algorithm.
- Real implementation: Implement and test in the real experimental platform the Guidance, Navigation and Control System that integrates the work and contributions undertaken on each area. This includes the setup of the experimental platform, the implementation of inner controllers and the treatment of the sensor measurements to estimate the required states.

1.3 Thesis Outline

This thesis is organised as follows:

- **Chapter 2: Guidance, Navigation and Control System**

  This chapter proposes the control structure for the Guidance, Navigation and Control system that is developed in this thesis. The elements of this structure and their communication flow are described in detail. Next, a thorough literature review of the control, navigation and control fields for multirotor vehicles is presented. This literature review is focused on the problems that are studied in this thesis. These are the path following problem, the obstacle detection problem and the reactive path planning problem.

- **Chapter 3: Multirotor Environment**

  This chapter is focused on the multirotor platform. The Asctec Hummingbird quadrotor and the rest of the elements that form the experimental platform are carefully described. Next, a complete and realistic mathematical model of the real multirotor platform is derived. An autopilot controller is designed and implemented in the actual platform. Path-Flyer, a freely available and open simulation benchmark for testing path following algorithms on a quadrotor vehicle, is developed in this chapter. Experimental tests are compared with the simulation results of Path-Flyer to prove its validity. To end up, the RotorS/Gazebo platform is introduced and the modifications made to this platform to emulate the real experimental one are explained in detail.

- **Chapter 4: Control-oriented and Geometric Path Following**

  In this chapter, four of the most relevant path folowing algorithms reviewed in Chap. (2) are implemented with the quadrotor model described in Chap. (3). These are, *Backstepping* and *Feedback Linearisation* control-oriented algorithms and *Nonlinear Guidance Law* and *Carrot-Chasing* geometric algorithms. The geometric algorithms are adapted to the three-dimensional space. A complete comparison and discussion of these four path following algorithms is derived from the simulation results. Finally, a comparison including qualitative and quantitative indicators is provided.

- **Chapter 5: Adaptive Geometric Path Following**

  The geometrical algorithms implemented in Chap. (4) (*Nonlinear Guidance Law* and *Carrot-Chasing*) are simple, effective and have only one tunning parameter. However, their control parameter depends on various factors, such as the velocity of the vehicle, the shape of the reference path and the dynamics of the vehicle. This chapter analyses the effect of the control parameter of these algorithms on their performance. Next, an adaptive version of both algorithms based on the use of Neural Networks is proposed. The proposed approaches include a velocity reduction term. Stability proofs are also given. Simulation results show that the proposed approaches improve the performance of the standard geometric algorithms. Furthermore, they have no parameters to tune.

- **Chapter 6: Path Following with Deep Reinforcement Learning**

  This chapter proposes a solution for the path following problem of a quadrotor vehicle based on deep reinforcement learning theory. Three different approaches implementing the *Deep Deterministic Policy Gradient* algorithm are presented. Each approach emerges as an improved version of the preceding one. The first approach uses only instantaneous

information of the path for solving the problem. The second approach includes a structure that allows the agent to anticipate to the curves. The third agent is capable to compute the optimal velocity according to the path's shape.

A training framework that combines the tensorflow-python environment with Gazebo-ROS using the RotorS simulator is built. The three agents are tested in RotorS and experimentally with the Asctec Hummingbird quadrotor. Experimental results prove the validity of the agents, which are able to achieve a generalized solution for the path following problem.

- **Chapter 7: Obstacle Detection**

  In this chapter, a study of obstacle detection solutions based on different sensors, such as cameras, LIDARs, RADARs or ultrasound sensors, is carried out. From the reviewed solutions, a LIDAR-based system is chosen to be implemented in the experimental platform. A model of the LIDAR sensor is developed in the RotorS environment. Then, an algorithm for processing the sensor measures to eliminate the possible ground detections is derived.

- **Chapter 8: Obstacle Avoidance with Deep Reinforcement Learning**

  A deep reinforcement learning approach for solving the obstacle avoidance problem is proposed in this chapter. This approach implements the *Deep Deterministic Policy Gradient* algorithm. It uses the LIDAR processed information to detect obstacles around the vehicle. If an obstacle is detected in the vehicle's route, the agent modifies the reference path to avoid it. The developed agent communicates with the path following agent by sending it the modified reference path. A detailed description of the process of defining the state vector, the reward function and the action of the agent is given. Different solutions are obtained and compared. The agents are trained and tested in the RotorS/gazebo platform. Simulations results prove the validity of the proposed approach.

- **Chapter 9: Conclusions**

  This chapter summarizes the contributions of the thesis, gives conclusions and describes the next steps.

Part I

Background

Chapter 2

Guidance, Navigation and Control System

The main topic of this PhD thesis is the multirotor Guidance, Navigation and Control (GNC) systems. Hence, before presenting the methodologies, strategies and contributions related to these systems, it is important to understand what is a Guidance, Navigation and Control system and which are its main components. Moreover, it is necessary to elaborate an exhaustive literature review on this area. These concerns are covered in this chapter.

2.1 Control Structure

One of the most important issues when designing or implementing a Guidance, Navigation and Control system is to define a proper control structure. The control structure defines the main elements of the system, and how they are connected with each other. This structure will depend on the type of vehicle that we are using and also in the problem that is going to be solved. The structure of the Guidance, Navigation and Control system proposed here is presented in Fig. 2.1. This is a general and typical structure for a GNC system [90][88]. However, it includes various blocks that are specific for the problem considered in this PhD thesis, as is the case of the Obstacle Detection block. The introduction of this structure will be helpful to understand the rest of the literature review presented in this chapter.

The main element of the structure of Fig. 2.1 is the Unmanned Aerial System (UAS). The UAS is composed of the Unmanned Aerial Vehicle (UAV), the actuators of the vehicle and the set of sensors. The description of the UAV platform employed in this thesis (i.e. the multirotor vehicle, sensors, actuators, on-board PCs, software, etc.) as well as the mathematical model of the system is found in Chap. (3).

The Control block is responsible for the stabilization of the vehicle (Autopilot) and for making the vehicle follow a desired given trajectory (Path Following). These two elements receive the

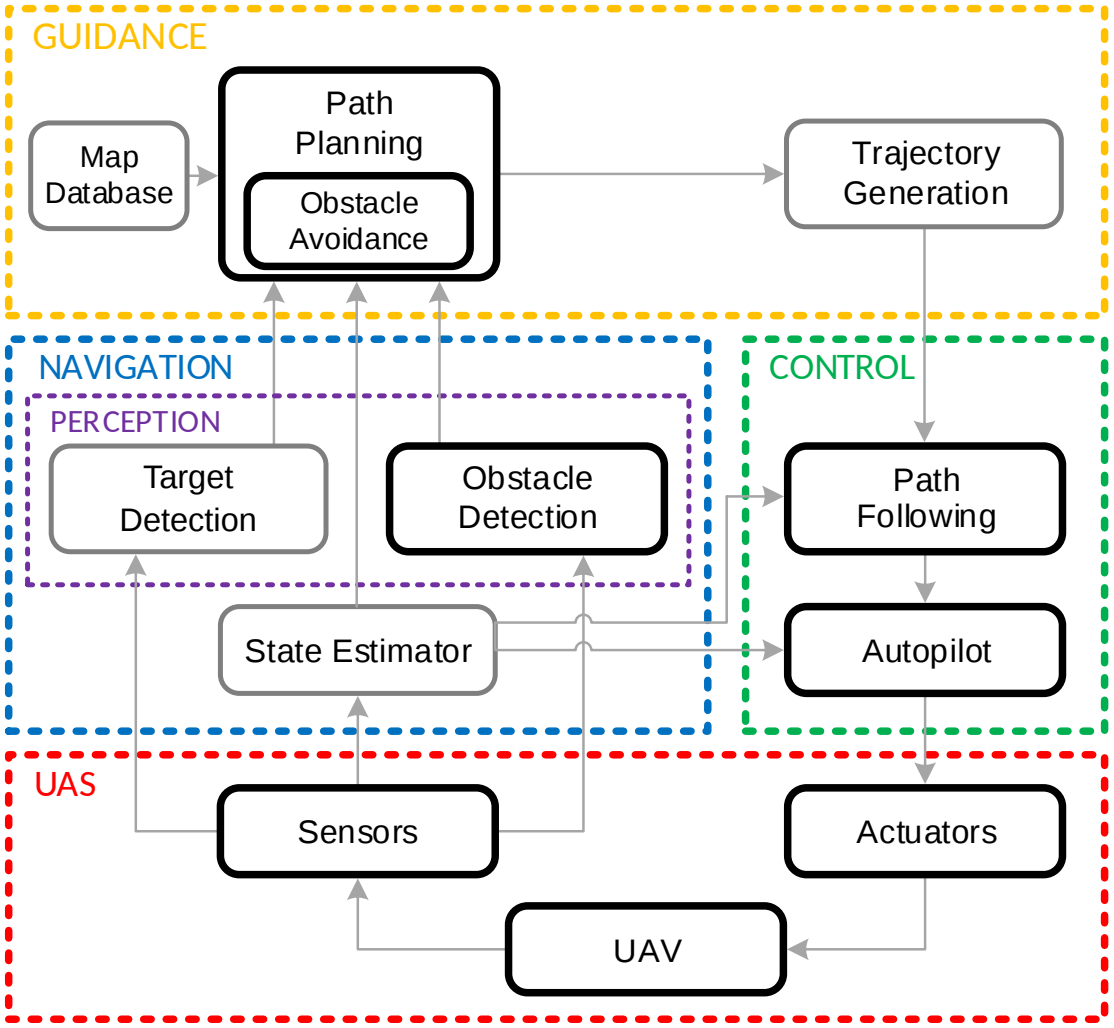

FIGURE 2.1: Guidance, Navigation and Control structure.

state information from the state estimation block and they end up controlling the UAV by sending commands to the actuators of the vehicle. It is important to mention that the autopilot is not always necessary, since some path following approaches deal with the stabilization of the vehicle too.

The Navigation block includes tasks that process the information from the sensors. Those tasks depend on the specific navigation problem being solved. It can be divided in two parts; the one dedicated to obtain the state of the vehicle (State Estimator) and the perception block that acquires information of the environment. In the perception block we include a task dedicated to the localization or detection the target (Target Detection) and a task dedicated to detect any obstacle in the vehicle's route (Obstacle Detection).

The Guidance block is the responsible of determining the route that the vehicle has to complete to follow a path, to follow a target or to get to a desired position. That is, to generate the reference trajectory that receives the Control block. In this particular case, the Path Planning task takes into account the location of the path and the state of the vehicle, both received from

the Navigation block, to design the best route to follow the path. Furthermore, it includes an Obstacle Avoidance task, that re-plans a new route when an obstacle in the vehicle's trajectory is detected. The Map Database gives a priori information of the path and environment, such as path point coordinates or localization of obstacles, to the path planner. The Trajectory Generation task is an optional block that is sometimes needed to adapt the reference trajectory generated by the path planner to the control block requirements. That is, to modify the trajectory, such as decreasing the vehicle's velocity or smoothing the path curves [82][35], to fulfil the control or vehicle constrains. This block is also responsible of generating additional trajectory parameters, if they are required by the trajectory controller.

The blocks marked in black in Fig. 2.1 denote the elements that are studied in the PhD thesis. They include tasks where contributions are produced (i.e. Path Following and Path Planning & Obstacle Avoidance) and other tasks where state-of-the-art solutions are implemented (i.e. Autopilot and Obstacle Detection). The UAS is also included in this list since it was necessary to perform the modelling, identification, simulation and experimental set-up of the system. The literature of the three main blocks of the GNC system is reviewed in detail in the rest of this chapter.

2.2 Control

The control block includes the stabilization of the system and the trajectory control. The autopilot is in charge of stabilizing the attitude of the multirotor, the fastest and most influential dynamics. The stabilization control problem for the particular case of a quadrotor has been solved using different techniques such as *Backstepping*, *Feedback Linearisation*, *Sliding Mode Control*, *PID*, optimal control, robust control, learning-based control, etc [13][131][101]. Since the stabilization control has already been widely studied, it is not studied in this thesis. However, a simple and effective solution for the autopilot is implemented in Section 3.3.

The trajectory control problem, defined as making a vehicle follow a pre-established path in space, can be solved mainly by two different approaches: using a trajectory tracking controller or with a path following controller. For the trajectory tracking problem a reference specified in time is tracked, where the references of the path are given by a temporal evolution of each space coordinate. Whereas path following (PF) handles the problem of following a path with no preassigned timing information, thus any time dependence of the problem is removed.

In [2] the authors demonstrate that following a geometric path is less demanding than tracking a timed reference signal. They argue that, although it is possible to perfectly track any reference with minimum phase stable systems, the tracking error increases in non-linear systems with presence of unstable zero dynamics as the signal frequencies approach those of the unstable zeros. PF controllers offer a number of advantages over trajectory tracking controllers, not only these are easier to design [84] but also result in smoother convergence to the path and less demand on the control effort [27], a smaller transient error and a stronger robustness [171] and the control signals are less likely to be saturated [39].

The PF problem [1][27][84] is defined as:

Definition 2.2.1. Path Following Problem: Let the desired path be described by a curve in the three-dimensional space $\boldsymbol{p}_d(\gamma) := [x_d(\gamma), y_d(\gamma), z_d(\gamma)]^T$, parametrized by the virtual arc $\gamma \in [0; \gamma_f]$, where γ_f is the total virtual arc length. The control objective is to ensure the convergence of the vehicle's position $\boldsymbol{p}(t)$ to the path $\boldsymbol{p}_d(\gamma)$ and $\boldsymbol{p}(t_f) = \boldsymbol{p}_d(\gamma_f)$ for a finite time t_f.

It is important to mention that some control-oriented algorithms may need a timing law for the virtual arc parameter, $\gamma(t)$, since some of this techniques consist in an adaptation of a trajectory tracking algorithm, such as *Backstepping*.

There are several approaches to define the desired path. Dubins defines the path as a combination of circle's arcs and lines tangent to them [47][10]. In the waypoint-based approach the sequence of points is commonly connected by straight-lines [35][127] or splines [190][43]. The most generic approach is a continuous function parametrized by the virtual arc length [27][4][84].

The guidance and control system's design to solve the PF problem is done by applying two different methodologies: the Separated Guidance and Control (SGC) approach and the Integrated Guidance and Control (IGC) approach [37]. The SGC approach (Fig. 2.2) is based on a separation between translational dynamics and rigid-body rotational dynamics. It consists on an outer-loop guidance law for generation of the movement commands ($\boldsymbol{\Phi}_{cmd}/\boldsymbol{v}_{cmd}$) and an inner-loop controller to track those commands, sometimes known as the autopilot. While the IGC approach (Fig. 2.3) combines both controllers in the same control loop.

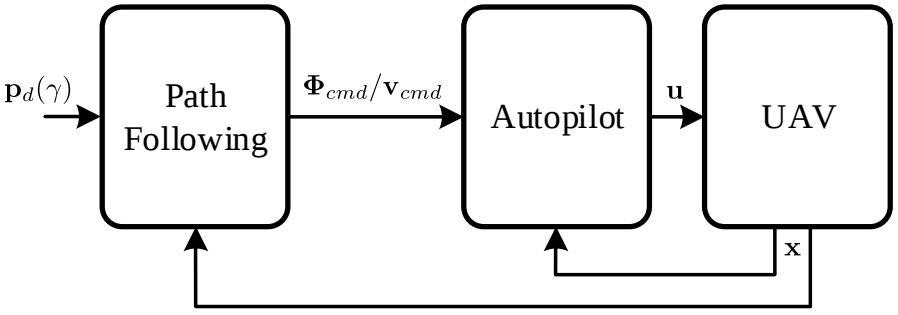

FIGURE 2.2: Separated Guidance and Control structure.

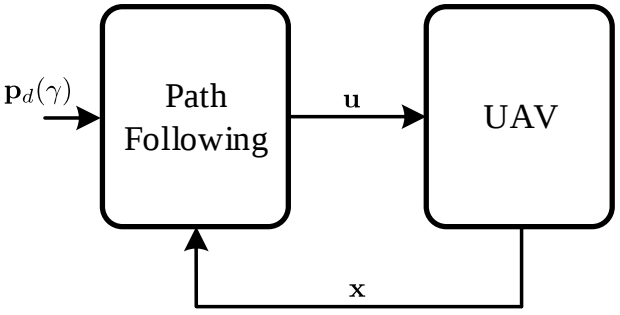

FIGURE 2.3: Integrated Guidance and Control structure.

The next subsections provide a literature review of the path following problem, the problem that originated some of the main contributions of this PhD thesis. A thorough review of literature related to PF control applied to quadrotors has been carried out. The algorithms are organized in subsections and a qualitative comparison is given. It is important to mention that some papers use the term path following when implementing controllers that track timed trajectory references. As these papers do not follow Definition 2.2.1, they are not included in the bibliographic review. The trajectory control problem, as opposed to the stabilization problem, is less dependent on the plant. Therefore, PF techniques for other type of UAVs are also provided.

2.2.1 Backstepping

Backstepping (*BS*) is a renowned technique widely used for control of non-linear systems [167][93]. This technique is based on the Lyapunov theory. Its control objective is to force the convergence of a set of predefined errors to zero. For this purpose, a Lyapunov function is stated for each error in such a way that if the time derivative of those functions is negative definite, the stability of the system, and therefore the convergence of the error to zero, is assured. Then, the control actions of the system are obtained as the ones that make negative definite the time derivative of all the stated Lyapunov functions.

This control strategy is used in most of the trajectory tracking control literature, as stated in [141]. Good performance in the reference tracking is achieved, as mentioned in [136], since *Backstepping* is able to provide larger regions of attraction than other types of controllers. However, as quadrotors present an underactuated nature, the control laws that have been developed do not assure global tracking. The solution to this problem is to eliminate the time dependence of the reference path and thus transform the trajectory tracking problem into a path following problem, in which it is possible to obtain a globally convergent *Backstepping* controller while keeping a large control capability [28].

In several publications *Backstepping* technique was used to solve both path following and stabilization problems. See [27] where a *BS* technique is applied using the IGC structure for the 3D control of a quadrotor. The proposed solution consists of a non-linear state feedback controller for thrust and torque control actions and a timing law that maintains the PF control law well-defined. This controller guarantees global asymptotic convergence of the path following error to zero for a wide class of desired paths and ensures that the actuation does not grow unbounded. The performance of the proposed controller is validated with simulation results. In [28] the authors improve their controller to deal with the presence of constant wind disturbance. This controller includes an estimate of the external disturbance to mitigate its effect. Experimental results demonstrate the robustness of the controller.

In [91] the problem of time cooperative PF for multirotors with a suspended payload is addressed. A team of multirotors transporting a suspended payload. A robust PF controller is developed for each vehicle in the system formed by an autopilot that controls the attitude and a *BS* controller that ensures each vehicle follows the desired path along a given speed profile. Numerical results verify path following accuracy and low coordination errors.

2.2.2 Lyapunov-based

Similarly to *Backstepping*, these algorithms are based on the Lyapunov theory. They are developed by assuring the Lyapunov stability condition, and thus, the convergence of the controller. The generated guidance control laws are usually simple and effective [33].

In [39] the authors propose a 3D path following control law based on Lyapunov theory by using the rotation matrix, that belongs to the 3D Special Orthogonal group (SO(3)), for attitude representation. This results in a singularity-free solution and allows speed profile's independent adjustment. The control law generates angular rates and thrust reference commands. Experimental results for the mission of following a desired path and for the time coordination problem [40] are given to illustrate the efficacy of the proposed control law.

In [118] a Lyapunov-based PF controller is proposed and it is complemented with a velocity observer and a constant disturbance estimator based on the immersion and invariance technique. Experimental indoor results for 3D paths show that the proposed approach fulfils the geometric specifications with an error that, according to the authors, is acceptable.

Some of the Lyapunov-based path following algorithms perform well against wind disturbances. As in [121], where an adaptive non-linear path following method is applied on a fixed-wing experimental platform. In this paper, the Lyapunov function is constructed based on the error equations and desired path function applying the vector field theory. Experimental results show good performance under wind disturbances. [32] presents another path following application for a fixed-wing vehicle that also takes into account wind disturbances by including an *Active Disturbance Rejection Control* (ADRC) for the attitude inner-loop combined with the *Lyapunov-based* control for the outer-loop. Experimental flight tests verify the effectiveness of this method.

2.2.3 Feedback Linearisation

Feedback Linearisation, along with *Backstepping*, is one of the most commonly used techniques for the control of quadrotors as discussed in [4]. The aim of this control technique is to linearise a system in a certain region of the state space by applying a non-linear inversion of the plant, so that non-linearities in the plant are cancelled and linear control theory can be applied [26]. Some of the advantages of this method are the simplicity in the control structure, the facility of implementation [166] and formal profs of error convergence when appropriate conditions are met. When applied to the PF problem, this method achieves the property of path invariance. That is, to ensure that once the system reaches the path it will stay on it for all future time [3][4].

In [141] the 3D path following problem for a quadrotor is solved by applying input dynamic extension and input-output *Feedback Linearisation* (FL). The designed controller allows to specify the speed on the path and the *yaw* angle of the vehicle as a function of the displacement along the path. Simulation results for a constant cruise speed along a circular path show that the quadrotor converges to the path as well as the velocity and *yaw* angle converge to the desired values. In [3] the authors propose an improvement of the previous stated controller by

implementing a transverse *Feedback Linearisation* plus input dynamic extension. This enhanced controller, which fully linearises the system, allows the quadrotor to move along the path in any desired direction, and both closed and non-closed curves can be used for the path definition. The capabilities of the controller are demonstrated with simulation results.

In [4], the authors of [3] adapted its PF controller to operate in the context of fault tolerant control, in the particular case when one of the four motors is completely disabled due to a failure. With the faulty system, only partial *FL* can be achieved. That is because the three rotor system is not differentially flat and presents uncontrolled internal dynamics. However, the uncontrolled non-linear dynamics are proved to be bounded, and thus it is shown that the system can be made to stay precisely on the path while the quadrotor is running only on three rotors.

In [55], a 3D PF implementation of a helicopter UAV is described. They apply the Feedback Linearisation technique basing it on the kinematic model of a helicopter. Six *PID* controllers are employed to control the attitude angles and the velocities. The main contribution of such approach is the consideration of the desired speed of the vehicle as a function of the assigned path. Simulations compare the designed controllers with variable and fixed velocity profile. Better performance is achieved whith variable speed profile.

2.2.4 Geometric

Geometric techniques were initially described in the missile guidance and control literature. Some of them were adapted to other type of vehicles, such as UGVs or UAVs. Examples of these techniques are: *Carrot-Chasing* algorithm, *Non-Linear Guidance Law*, *Pure Pursuit*, *Line-of-sight* and *Trajectory Shaping* guidance law. *Carrot-Chasing* (*CC*) [122] is a simple geometric strategy that consists on steering the UAV toward a Virtual Target Point (VTP) located on the path. The VTP is periodically updated and obtained by adding a constant distance along the path to the vehicle's closest point. *Non-Linear Guidance Law* (*NLGL*) [133] is also based on the VTP concept. In this case the point is calculated by creating a circumference around the vehicle with a constant radius and taking the point in which the circumference intersects the path. The obtained VTP is at a distance from the vehicle equal to the radius of the circumference. Next, the acceleration commands that steer the vehicle to the VTP are calculated. The *Pure Pursuit* (*PP*) [11][130] algorithm tries to guide the vehicle straight to a target point on the path. *Line-of-sight* (*LOS*) [151][10] seeks to steer the vehicle directly towards the closest point on the path. *Pure Pursuit and Line-of-sight* (*PLOS*) [94] is the combination of *PP* and *LOS*. In the *Trajectory Shaping* (TS) [137] guidance law, the commanded lateral acceleration is generated as function of the vehicle heading angle, target heading angle and line-of-sight angle.

Two algorithms based on missile guidance laws, *Pure Pursuit* and *Trajectory Shaping*, have been implemented on a quadrotor vehicle in [113]. The concept of VTP navigation has been utilized to generate the required curved trajectories. A guidance algorithm based on the notion of Differential Flatness (DF) was implemented as a baseline control-oriented algorithm. Simulation and experimental results comparing these algorithms show that *TS* generates smaller position errors and requires lower control effort than *PP* and *DF* approaches. Furthermore, it is proved that missile guidance laws can be successfully utilized by a quadrotor.

In [57] various geometric algorithms, such as *PP*, *LOS* and *Proportional Navigation Guidance* (PPN) law, are applied to the problem of autonomous landing of a quadrotor UAV. Simulation results suggest that the *PPN* algorithm performs the best in terms of time and control effort, where *LOS* presents the worst performance.

In [172] a survey of some of the most common PF algorithms applied to fixed-wing UAVs is presented. In addition to the control-oriented algorithms, *Carrot-Chasing*, *NLGL* and *PLOS* are described. Simulation results comparing those algorithms show that the *NLGL* is the geometric approach that achieves the best performance in terms of path distance error.

2.2.5 Model Predictive Control

Model Predictive Control (MPC) is a well-known technique [30][56][115] that transforms the control problem into an optimization problem. At any sampling time instant, a sequence of future control values is computed by solving a finite horizon optimal control problem. Only the first element of the computed control sequence is used and the overall process is repeated at the next sampling time. The most important drawback of this technique is that resources needed for computation and memory grow rapidly with the time horizon.

In contrast to the most common path following approaches, such as the geometric algorithms or the *Backstepping* technique, the *MPC* approach is able to handle constraints on states and inputs, non-linear MIMO dynamics, and non-linear reference paths [51].

In [129] a cascaded control structure applying the *Non-linear MPC* technique for the PF outer-loop control is presented. The inner-loop control for the acceleration tracking is based on non-linear dynamic inversion. This controller allows multirotor vehicles to follow a path whose geometry is defined as a spline in 3D space. Simulation results demonstrate the good performance of this approach. Furthermore, the obtained runtimes, well below the sample rate, suggest that it could be implemented in on-board embedded systems. An improvement of this approach is presented in [5], where an adaptive augmentation scheme for the inner-loop controller is designed. The adaptive augmentation is based on a non-linear design plant and uses *Model Reference Adaptive Control* (MRAC). Simulation results show a stronger robustness of the adaptive augmentation in relation to the baseline controller.

A trajectory optimization strategy for UAVs based on non-linear optimal control techniques is presented in [148]. The approach is based on a Virtual Targed Vehicle (VTV) perspective where a virtual target is introduced. Numerical computation results show that it is able to compute aggressive manoeuvres. In [149] the authors extend and adapt the proposed strategy for path following in presence of time varying wind disturbances by means of a sample-data *MPC* architecture. Simulation results with a fixed-wing vehicle show the effectiveness of the predictive PF approach.

In [63] a non-linear receding horizon guidance law is developed to solve the PF problem on a fixed-wing vehicle. An extended Kalman filter is used to estimate wind velocities. The proposed

law achieves efficient PF by making input constraints active. Its effectiveness is demonstrated by flight tests.

In addition to path following approaches, several trajectory tracking implementations on quadrotor vehicles are found applying the *Model Predictive Control* technique. Some of them have reported experimental results showing a good performance against disturbances [8][18].

2.2.6 Vector Field

In the *Vector Field* (*VF*) based control, a set of vectors is virtually placed around the path in such a way that if the vehicle follows the direction of those vectors it will converge into the path. Those vectors are used to generate desired course inputs to the inner-loop attitude controllers.

The first application of a path following approach using this method for a UAV is found in [127]. In this paper *Vector Field* path following control laws are developed for straight-line paths and circular arcs and orbits. Asymptotic decay of path following errors in the presence of constant wind disturbances is demonstrated with Lyapunov theory. Experimental tests are carried out with a fixed-wing UAV to show the effectiveness of this method.

In [200] the 3D path following problem is solved by implementing a velocity vector field following controller based on the differential flatness notion. This approach is designed with an inner-loop controller that makes the vehicle follow a specified velocity vector field. The validity of the method is demonstrated by numerical simulations and experiments with a quadrotor for three different vector fields.

A *VF* path following guidance for 2D and 3D twice differentiable curves is stated in [102]. The proposed approach is based on the Helmholtz theorem, which states that an arbitrary vector field can be decomposed into two parts, conservative part (irrotational) and solenoidal part (rotational). This approach combines both parts by using the conservative part for long distances to the desired path and using the solenoidal part when the vehicle is along the path. UAV input constraints and constant wind disturbances are assumed to be present. The method's performance is validated by simulation results.

In [199] an adaptive control scheme for UAVs path following under wind disturbances is proposed. This control strategy integrates the *Vector Field* PF law with an adaptive term to deal with the effect of unknown wind disturbances. Simulation results with wind conditions show that the proposed method compensates for the lack of knowledge of the wind vector and, according to the authors, it attains a smaller path following error than the state-of-art vector field method.

Refer to [183][58][44][83][184] for path following *Vector Field* implementations applied on different types of aerial vehicles. In [172], simulation results comparing this algorithm to other geometric and control-oriented algorithms show that the *VF-based* algorithm is the one with least cross-track error. However, final remarks conclude that it can be difficult to implement.

2.2.7 Learning-based

Learning-based algorithms constitute an emerging field that, due to the significant progress made in recent years, has become a wise solution for different types of problems. In particular, it is being introduced in the trajectory control problem on different types of vehicles, including UAVs. This field includes supervised and unsupervised techniques, reinforcement learning algorithms, data-driven approaches and other deep machine learning methods.

In [169] a *MPC* controller is used as a supervisor to train a neural network control policy to solve the path following and obstacle avoidance problem on a quadrotor vehicle. With this algorithm the computational efficiency problem of the *MPC* technique is solved while maintaining a similar performance, as proved by experimental results. An Iterative Learning Control approach is proposed in [197] to solve the PF problem on a quadrotor. The control scheme is composed by a PD controller and an feed-forward controller. This approach focuses on repetitive flight to learn from experience. Simulation tests and off-line experimental results are presented to prove the effectiveness of the controller. In [195] a data-driven control approach is proposed for the adaptive path following control of a fixed-wing vehicle. The reliability of the approach is demonstrated through simulation results and flight tests. Other learning-based approaches have been used to solve the path following problem on different types of vehicles, such as vessels [62][114][160] or airships [128].

Various learning-based approaches are found in the literature solving the trajectory tracking problem for a quadrotor vehicle [178][38]. Some of these approaches consider wind disturbances in its design, as in [104] where an adaptive trajectory tracking control based on a reinforcement learning algorithm is presented.

2.2.8 Optimal Control

The Optimal Control theory aims to operate a dynamic system at a minimum cost. That is, to follow a path with a minimum error and control effort. The most well-known control techniques to solve this problem are the *Linear Quadratic Regulator* (LQR) and the *Linear Quadratic Gaussian* (LQG).

In [179] a general solution for the PF problem is presented. The proposed approach is developed as a fixed end-time optimal control problem and it relies on a geometric formulation based on the notion of differential flatness. The resulting controller is applied to a quadrotor simulation system with the mission of performing aggressive manoeuvres and it is demonstrated that the proposed problem formulation is solved efficiently.

A UAV guidance law using an adaptive *LQR* formulation is addressed in [95]. The *LQR* is optimized using a genetic algorithm for tighter control of UAV errors in high disturbances. Simulations for straight line and loiter paths under various wind conditions prove the effectiveness of the approach.

Experimental results applying optimal control theory to solve the trajectory tracking problem on a quadrotor vehicle can be found in [69]. In this paper, the authors propose a solution based on

the definition of a path-dependent error space to express the dynamic model of the vehicle. The controller is designed using *LQR* state space feedback and adopts the *D-methodology* integrated with the anti-windup technique in order to achieve zero static error for the integral states and avoid actuator saturation.

2.2.9 Sliding Mode Control

Sliding Mode Control (SMC) is a non-linear control method that attains the control objectives by constraining the system dynamics to a pre-defined surface by means of a discontinuous control law. This control technique is considered to be effective and robust [48]. However, it can present implementation issues due to the chattering effect.

The bibliographic search revealed no *SMC* application solving the path following problem for a quadrotor vehicle. However, it has been applied to solve the PF problem on other UAVs, such as fixed-wing vehicles. In [159] a lateral guidance law for cross-track control based on the *SMC* technique is developed. This guidance law includes a feed-forward component related to the rate of change of the desired heading angle, which permits to improve the performance and achieve accurate tracking while following curved paths. The proposed guidance scheme is evaluated based on experimental flight results. According to the authors, it presents a good performance in the presence of wind and parametric uncertainty. In [9] the controller presented in [159] is improved by implementing a Partially-IGC strategy that includes a sliding mode control on the inner and the outer loops using a non-linear sliding surface based on *Second Order Sliding Mode* (*SOSM*) control theory. Experimental tests compare the conventional SGC approach and the proposed Partially-IGC approach to show that the second one presents a faster convergence of the cross-track error toward zero.

In [190] the Second Order Sliding structure is used to develop a PF application. The proposed controller provides smooth bank and turn coupled motions. To estimate the uncertain sliding surfaces a *High-Order Sliding Mode* (*HOSM*) differentiator is applied. The sliding surface structure is based on the Pure Pursuit algorithm through a set of intermediate control variables and also introducing a virtual target point in the path. This approach eliminates time-consuming and intensive computation. According to the authors, simulations show that it provides an excellent performance even under wind turbulence conditions.

Regarding the trajectory tracking problem, numerous *SMC* quadrotor implementations are found [196][20]. Some of these approaches are able to deal with wind disturbances [49][182].

2.2.10 Comparison

Refer to Table 2.1 for a comparison of the reviewed PF algorithms. The characteristics of these algorithms are evaluated only in the context of the path following problem applied to UAVs and are based on the reviewed literature. The columns refer respectively to: the control structure (i.e. Integrated Guidance and Control or Separated Guidance and control); the type of results (experimental or simulation); the application to quadrotors; good experimental results against

external disturbances; including a Fault Tolerant Control (FTC) strategy; and implementing an adaptive approach.

TABLE 2.1: Comparison of the PF techniques.

	Structure	**Results**	**Quadrotor**	**Wind dist.**	**FTC**	**Adaptive**
Backstepping	IGC	Experimental	Yes	Yes	No	No
Lyapunov-based	SGC and IGC	Experimental	Yes	Yes	No	Yes
Feedback Linearization	SGC and IGC	Simulation	Yes	No	Yes	No
Geometric	SGC	Experimental	Yes	No	No	No
Model Predictive Control	SGC	Experimental	Yes	Yes	No	Yes
Vector Field	SGC	Experimental	Yes	Yes	No	Yes
Learning-based	SGC	Experimental	Yes	Yes	No	Yes
Optimal Control	SGC and IGC	Simulation	No	Yes	No	Yes
Sliding Mode Control	SGC and IGC	Experimental	No	Yes	No	No

2.3 Navigation

Navigation can be defined as the process of data acquisition, data analysis and extraction of information about the vehicle's state and its surrounding environment with the objective of accomplishing assigned missions successfully and safely [88]. This information is obtained by means of the sensors of the Unmanned Aerial System. The most common sensors for UAS are the accelerometers, gyroscopes, magnetometers, pressure sensors, GPS, cameras and LIDARs. The raw measurements provided by these sensors are used to perform the state estimation and for the perception algorithms.

In this thesis the navigation is mainly focused on the detection of an obstacle. There exist different forms of performing this task in addition to the pure obstacle avoidance approaches, as for instance with a *Simultaneous Mapping And Planning* method. The literature of these perception algorithms are reviewed in this section. Furthermore, state estimation methodologies and other important perception tasks are reviewed.

2.3.1 State Estimation

The state estimation concerns mainly the processing of raw sensor measurements to estimate variables that are related to the vehicle's state, such as attitude, position and velocity [88]. The state estimation algorithms usually fuse information from many sensors to estimate this state.

The most common approach for fusing sensor data to estimate the state is the extended Kalman filter (EKF) [155][72]. That is, using one filter with all the state variables that need to be estimated. Sometimes, the state estimator is composed by two cascaded EKFs: one for attitude

estimation using Inertial Measurement Unit (IMU) raw data, and the other for position and velocity estimation using GPS raw measurements and translational accelerations.

Another widely used method that provides good results with the estimation of the attitude on UAVs is the complementary filter [111][110][52]. This method is only applicable to the measurements of accelerometers, gyroscopes and magnetometers of an IMU sensor. The measurements of these sensors are combined to obtain an estimation of the orientation of the vehicle represented in quaternions.

The GPS systems depend on the access to the signals from satellites. Since satellites are not always available, as in the case of indoor and urban environments, other approaches to solve the state estimation problem are needed. The ranging systems, like infrared [23] or ultrasonic [162] sensors, are used to aid an state estimator for the stabilization of a vehicle relative to the walls in an indoor environment. As the outdoor UAS generally fly in relatively large open spaces, there is not sufficient environmental structure for the relative position estimation using this ranging systems.

In outdoor environments where the GPS signal are not available, the vision-based state estimators become a great solution. In the on-ground vision, cameras are placed externally to the vehicle and are used to track it and to estimate its attitude and/or position. For example, [74] and [123] use the VICON system in their research on flight control and for cooperative path following. Visual odometry [12] consists on an incremental method that analyses a sequence of images to estimate changes in position and/or orientation over time. In the target relative navigation [78] [107] the position of the vehicle relative to a specific detected target is estimated. The terrain relative navigation [41] estimates the vehicle's position, and sometimes the velocity, by comparing terrain measurements from on-board cameras with a terrain map.

2.3.2 Perception

Perception is the ability to use measurements from sensors to build an internal model of the environment and to generate events of situations perceived in the environment [88]. The recognition process involves comparing what is observed with the UAS a priori knowledge [80]. The information is usually obtained by vision systems (i.e. with cameras) and by LIDAR sensors. Typically, the information obtained by the perception algorithms is not directly used as measurements in the flight controller, but as inputs to higher-level guidance systems. The perception block can serve to several functions such as target detection, obstacle detection or mapping. These functions are covered in next subsections, and a literature review on both vision-based and LIDAR-based approaches is given.

Target Detection

Most of these perception algorithms are based on vision systems. These algorithms are similar to the target relative navigation mentioned in the state estimation section. The main difference is that here the information is used for guidance and the trajectory control relies on another

state estimator algorithm, while in the target relative navigation the visual estimates are used to control the vehicle.

One of the applications of target detection is the automatic landing. For example, in [156] a vision algorithm is used to detect and recognize from a monocular camera a helipad for landing a helicopter. In [71] a similar approach is presented to land a UAV on a stationary target using vision and GPS.

The horizontal target approach is also a typical application of the target detection. That is, to approximate to a frontal target and to hover at some distance from it. A vision-based velocity controller for frontal target tracking is presented in [117]. The vision algorithm detects and tracks building windows. It detects the target by using segmentation and square finding. The tracking is made with template-matching and a Kalman filter.

Another application is the mobile ground target detection and tracking. In [180] a pursuit-evasion game with a helicopter and UGVs is implemented by means of a vision-based approach. The implemented vision system can actively track coloured objects. An indoor quadrotor application for cooperative vision-based tracking of ground vehicles is presented in [22]. An optimization technique and a Kalman filter are used by the vision tracking algorithm.

Obstacle Detection

This subsection introduces perception approaches for obstacle detection that do not perform a mapping of the environment. The approaches of obstacle detection that include maps are presented in the next subsection. Both vision-based and LIDAR-based systems are commonly used to detect obstacles.

Computer vision can be applied to solve this problem by using optical flow, stereo vision systems or monocular cameras. The optical flow estimates the motion of the elements of an image. The stereo vision systems are commonly used to obtain the depth information. Different techniques can be used to detect the obstacle with monocular vision. These techniques include estimating the relative size or clarity of the obstacle, using a texture gradient, by means of interposition or by motion parallax [17]. Also, the obstacle can be detected from the known characteristics of an object (e.g. color or shape).

A single camera obstacle detection algorithm based on the estimation of an optical flow probability distribution is presented in [98]. The resulting algorithm provides distance to the obstacles surrounding the UAV. This methodology was tested experimentally. In [79] the obstacle detection for the navigation of a UAV through urban canyons is solved by the use of an optic flow (from a pair of sideways-looking fish-eye cameras) and a stereo vision. The environment is represented with a 3D point cloud map, and obstacles are detected by using a distance threshold. A multi-obstacle detection algorithm based on stereo vision is presented in [189]. First, a depth map of the scene is obtained, and then, another algorithm is used to find out five dangerous objects and give them bounding boxes. The results show that the algorithm can detect at most five obstacles in 15m. In [15] the obstacle global position is estimated by tracking some coloured

flags on the obstacle and using a GPS and IMU for knowing the global position of the vehicle. Other interesting application with monocular vision systems that estimate the size expansion [6] or the relative direction of an obstacle [108], or the relative distance to an obstacle [152] can be found in the literature.

Using LIDAR sensors for obstacle detection and avoidance is an interesting solution because these approaches are less computationally expensive than vision-based approaches, providing a fast reactive system that can prevent collisions in the last minute. In [119] the authors propose a system that uses immediate LIDAR measurements to detect the ground and frontal obstacles and computes a reactive action based on the current context. A collision avoidance approach for an hexacopter UAV with a mounted LIDAR is presented in [132]. A Kalman filter is used to estimate the position, velocity, and acceleration of the obstacle by using the data of the LIDAR as the associated measurement. The estimation of the obstacle state is used to predict the future trajectory of the moving obstacle. In [170] a simple obstacle detection and avoidance approach for a quadrotor UAV is developed by using the information of a rotating LIDAR sensor. The aim is to support the pilot during the manual flights avoiding possible collisions. The approach is assessed by experimental results. A 2D-LIDAR based obstacle detection method for a UAV is implemented in [198]. In this approach a velocity estimation method is used to estimate the position of the LIDAR as it scans each point and then corrects the twisted point cloud. The effectiveness of the methodology is proved by simulation and experimental results.

RADAR sensors are also used to detect obstacles in UAV systems [96]. However, this sensors are very heavy, which limits their application to small multirotor vehicles. Applications using acoustic sensors (ultrasounds) can also be found in the literature [138]. The problem of this sensors is that they have a short operational range, which restricts their use to only indoor usage and low-speed objects. Other multi-sensor applications [138][54] use information from several sources to detect an obstacle.

Mapping

Mapping the environment consists of building some internal representation of the scene [88]. Mapping-based approaches allow the use of more sophisticated path planning and obstacle avoidance algorithms. The mapping perception systems can be classified in three categories: the simultaneous location and mapping (SLAM), the simultaneous mapping and planning (SMAP) and the safe landing area detection (SLAD).

Simultaneous Localization And Mapping: Consists on building a map of an unknown environment and localizing the vehicle on the map at the same time. This map is usually represented by a set of features or a point cloud. Generally, SLAM approaches are not applied to the detection and avoidance of obstacles.

In [16] the implementation of visual SLAM techniques to outdoors images taken from UAVs is presented and tested experimentally. An optic flow-based vision system for autonomous localization and scene mapping for a small and micro-UAVs is presented in [89].

SLAM LIDAR-based approaches can be also found in the literature. They have been generally developed for indoor navigation of small vehicles [61] [19]. However, in some cases LIDAR has also been used on bigger UAVs for outdoor mapping, such as [85], where a 3D terrain mapping from LIDAR on-board a helicopter is demonstrated.

Simultaneous Mapping And Planning: The maps built with this approaches are typically used for obstacle avoidance and path planning. SMAP focuses more in efficiency and robustness rather than accuracy. State estimates are generally available from the GPS-IMU.

Some interesting work on applying vision-based SMAP for UAVs can be found in the literature. Such as in [14] where SMAP is performed using a stereo vision system, or in [25] where another perception system with stereo cameras combining depth image information with image segmentation is presented.

However, the most successful approaches on SMAP for UAS have been done using LIDARs. [161] proposes an obstacle detection and avoidance scheme based on building local obstacle maps and designing object-free trajectories using MPC. In [176] a LIDAR system is used to develop a effective 3D outdoor navigation system for an autonomous UAV. Safe autonomous flight with 3D obstacle avoidance capability is demonstrated in [157]. This method combines global planning and reactive obstacle avoidance. [66] developed a navigation approach for a quadrotor to explore and map unknown indoor environments.

Safe Landing Area Detection: These algorithms are needed when UAVs are commanded to land on unknown terrains to accomplish their mission or to achieve an emergency landing.

[116] presents a stereo vision-based system for a UAV that combines terrain mapping with SLAD. In [175] a stereo range map of the terrain is created and then a SLAD algorithm finds all safe landing regions by applying a set of landing point constraints (slope, roughness, distance to obstacles) to the map.

Typically, LIDAR sensors are used in situations with complex terrains or poor light conditions. In [154], two different SLAD algorithms are compared; one using a monocular vision and the other a hemispherical LIDAR. Although the algorithms reported similar results, the LIDAR-based algorithm, unlike the vision-based, was able to run on-board the UAV due to its lower computational cost. Moreover, the LIDAR-based approach proved its effectiveness in night flights. In [185], another SLAD algorithm was applied to the 3D point cloud obtained from a hemispherical 3D LIDAR to determine the safe landing regions. [158] investigated the effects of factors such as smoke and dust on a LIDAR-based SLAD approach.

2.4 Guidance

Guidance is the part of the system that is in charge of carrying out the planning and decision-making functions to achieve assigned missions or goals [88]. It takes inputs from the navigation system and uses mission information to generate reference trajectories for the control system. This block allows to replace the cognitive process of a human pilot or operator. Guidance can

include different tasks such as path planning, mission planning, decision-making or trajectory generation. This thesis focuses on the path planning and obstacle avoidance task. Thus, only path planning literature is reviewed.

Path planning is defined as the process of using accumulated navigation data and a priori information to allow the UAS to find the best and safest way to accomplish a specific mission [88]. In most of the cases, the path planning is also responsible for the obstacle avoidance task [31][34].

In this section the most relevant and practical path planning techniques for UAS are presented. Each technique is employed for different situations and problems. For example, reactive path planing is commonly applied to perform real-time obstacle avoidance to avoid last instant collisions.

2.4.1 Road Maps

A road map is generally represented as a connectivity graph. The nodes correspond to positions of the vehicle and the edges represent obstacle-free paths between these positions. In these algorithms the connectivity graph is first constructed and then the best path according to a defined criteria is searched [163]. The most usual path planning methods using Road maps are briefly described hereafter:

Visibility Graph: In this method obstacles are approximated by polygons and the edges of these polygons are connected by straight line segments. It is a complete method but it only works in two dimensions. In [73] visibility graphs are applied for path generation on a quadrotor system.

Voronoi Diagrams: The road map is formed by a set of Voronoi edges. These edges are equidistant from all the obstacles in the region. Voronoi paths are, by definition, as far as possible from the obstacles. In [75] the authors developed an obstacle field route planner that is based on Voronoi road maps.

Probabilistic Road Maps (PRM): It takes random samples from a set of discretized positions from the vehicle's space and it connects them with obstacle-free segments. This method is adequate for large spaces, however, due to its slow searching rate it is inefficient for dynamic obstacle avoidance. A path planner based on the PRM method for a UAV is presented in [76].

Rapidly Exploring Random Trees (RRT): This approach is a variant of the PRM method. Instead of taking random samples, the planner starts at the initial vehicle's position and randomly expands a tree. That is, nodes are added successively to the tree, connected via edges, until a termination condition is reached [163]. This method improves the efficiency of PRM since it is able to rapidly search in large spaces. Furthermore, it is appropriate for unknown environments, as demonstrated in [187], where the authors compare the performance of the PRM and RTT planners with a helicopter UAS.

2.4.2 Potential Fields

In this method the vehicle is considered under the influence of force fields generated by the goals (attractive forces) and obstacles (repulsive forces) in the space. It has low computational cost and is easy to implement. The principal limitation is that a local minima can appear depending on the obstacle shape and size [163]. In [157], a variant of the Potential Field based path planning algorithm implemented on a UAV is presented. The system relies only on LIDAR-based sensing and perception.

2.4.3 Heuristic Search Algorithms

As they name denote they are based on heuristic rules. These rules are used for guessing which path moves the vehicle closer to the defined objective. Heuristic Search Algorithms (HSA) are able to provide reasonably good performance with low computational cost. $A*$ and $D*$ algorithms are the most common HSA. These algorithms are based on a tree-search with nodes that represent the possible solutions (i.e. positions of the vehicle). $A*$ algorithm evaluates the goodness of each node by estimating the distance to the goal. $D*$ is an incremental version of the $A*$ algorithm that is able to re-plan the path in real time when changes in the environment occur. The term incremental refers to the ability of reusing previous search effort in subsequent search iterations. NASA researchers [176][185] have developed two 3D path planners for a multirotor combining heuristic planning concepts and the $A*$ search algorithm.

2.4.4 Optimization Methods

In these methods the path planning problem is considered as a numerical optimization problem. Constraints such as obstacles or vehicle's kinematic and dynamic limitations are represented with mathematical relationships [60]. The main advantages are that, theoretically, they produce optimal solutions and they consider the limitations of the vehicle. Nevertheless, they are computationally expensive. The most investigated optimization methods for UAS are Mixed Integer Linear Programming (MILP), Receding Horizon Control (RHC) and Motion Primitive (MP). MILP and RHC have been successfully applied for path planning of an indoor quadrotor [42][120].

2.4.5 Planning Under Uncertainties

When finding the best path for a specific mission, uncertainties such as position, environment knowledge or limited precision in tracking commands can become a serious problem [60]. To deal with these uncertainties, the most common approach is to consider the worst case by introducing a conservative safety margin. However, there are some works that consider uncertainties directly in the planning algorithm, such as in [67], where this problem was addressed for autonomous

indoor exploration using a quadrotor vehicle. NASA researchers [59] developed a 3D path planning algorithm based on risk minimization that allows the UAS to operate with reliability and safety under uncertainties.

2.4.6 Reactive Path Planning

Most of the path planners previously presented are based on a global representation of the environment and they are generally computationally expensive. Instead, reactive planning or reactive obstacle avoidance algorithms run very fast and are useful for preventing collisions in the last instant. The term reactive planning refers in general to a class of algorithms that use only local knowledge of the obstacle field to plan the trajectory [60].

There exist different methodologies to implement a reactive path planning approach such as with potential fields, by implementing optimization methods, machine learning approaches or other methods based on geometric relations. Note that reactive path planning includes techniques that are also used in global path planning (i.e. potential fields or optimization methods). However, in the reactive approaches these methods only use local information of the environment. Some of these techniques and other methodologies implemented on multirotors and other UAVs are reviewed next.

In [45] a reactive obstacle avoidance approach based on the potential fluid flow theory is implemented to a fixed-wing UAV. The algorithm computes the instantaneous local potential velocity vector, which is used as a command of the inner controller. Manoeuvring constraints of the UAV are included in the control design. Obstacles are approximated by bounding rectangles. The efficacy of the proposed approach is demonstrated through numerical simulations. Another UAV approach based in potential fields is presented in [46]. The proposed solution is based on the dynamic artificial potential field (DAPF) algorithm, which generates real-time reactive collision-free paths according to the threat level of moving obstacles. The effectiveness of the proposed path planning method is assessed through simulations results.

A 3D reactive motion planner based on optimal control theory for a fixed wing UAV in a dynamic workspace is presented in [21]. A virtual space representation is used to formulate the problem and generate the locally optimal trajectories in real time. These trajectories are defined by the vehicle's speed, the flight path angle (pitch) and the heading angle. The dynamic and kinematic constraints are taken into account. The effectiveness of the proposed methods are shown by simulation results. [188] combines a reactive collision avoidance algorithm with global path planning techniques for UAVs operating in unknown environments. The implemented global path planning techniques are the Probabilistic Roadmaps and the Rapidly-Exploring Random Trees. Additionally, the system includes a reactive controller based on Optimal Reciprocal Collision Avoidance (ORCA) for achieving a fast sense-and-avoid behaviour. The proposed system is evaluated in simulation and experimentally. The experimental platform is formed by a quadrotor vehicle equipped with a structured-light depth sensor that is used to obtain information about the environment in form of occupancy grid map. In [7] the OCRA algorithm is used to develop a decentralized method for reactive avoidance with multiple aerial vehicles in a

industrial environment. This algorithm computes an optimal solution for the near future. Both static and moving obstacles are considered in a 3D environment. The approach was tested in real indoor experiments with four quadrotors. The UAVs were able to fly in the presence of static obstacles while avoiding potential collisions in real-time. In [105] a quadrotor LIDAR-based collision avoidance algorithm is developed by combining an RRT path planner with a Signed Distance Field (SDF) based collision checking algorithm. In this methodology the trajectory is optimized by a short cut and Optimal Polynomial Trajectory algorithms. The proposed solution is assessed in several simulating scenarios using RotorS Gazebo simulator in the presence of static and dynamic obstacles. In [150] a 3D ocupancy grid map is build with the information of a monocular camera and depth camera. Then, a receding horizon planning architecture is used to obtain the optimal trajectory in the local horizon. The authors demonstrated the planner capabilities with autonomous quadrotor flights in an urban environment.

Other approaches use geometrical notions to define the manoeuvre to avoid an obstacle. In [132] the collision cone approach is used to avoid potential collisions with a moving obstacle detected by a frontal LIDAR. A Kalman filter is used to estimate the state of the obstacle (position, velocity, and acceleration), which allows to predict the future positions of the moving obstacle. Numerical simulations with a multirotor vehicle are conducted to verify the performance of the collision avoidance algorithm. In [77] a 3D occupancy map is defined in a cylindrical volume around the UAV. Then, an expanding elliptical search is performed to find a waypoint that offers a collision free route towards a goal waypoint. The proposed approach is validated in experimental flights with an autonomous helicopter equipped with a stereo camera and LIDAR sensor. An optical flow based obstacle avoidance planner is presented in [168]. The proposed approach predicts a cylinder of free space into the image flow representation of the environment and steers the vehicle by manoeuvring this cylinder through the upcoming environment. The approach is validated with real experiments with a quadrotor flying in a forest environment. A reactive method for static obstacle avoidance of a UAV is presented in [68]. The field-of-view of the obstacle detection sensor is converted to a spherical occupancy grid map. The proposed method calculates collision-free paths within the field-of-view of the sensor by applying approximated reachable sets and the wavefront algorithm. The task of the wavefront algorithm is to find a short path from the initial cell of the UAV to a goal cell. The approach is tested in a simulation environment with a LIDAR sensor and with a stereo vision system.

Recent years showed an increase on the application of learning-based solutions to different problems, including reactive obstacle avoidance. In [153] an obstacle avoidance planner based on deep reinforcement learning theory is presented. In this approach the implemented agent receives the information of a LIDAR sensor placed on a quadrotor vehicle. The reward function of the agent is defined as a artificial potential field. The agent is trained in the RotorS Gazebo simulator. The proposed system is evaluated in simulated and real indoor scenarios in the presence of static and dynamic obstacles with success. A UAV ground target tracking approach under obstacle environments using a deep reinforcement learning algorithm is found in [99]. The reward function is constructed based on the line of sight and an artificial potential field. Long Short-Term Memory (LSTM) networks are used to approximate the state of the environment that is fed as the state of the deep reinforcement learning agent. This networks improve the approximation accuracy and the efficiency of data utilization. The proposed method is validated

with simulation results. An imitation learning technique is used in [140] for solving the reactive path planning problem. The system consists on a small quadrotor with a monocular camera that flies at low altitude through natural forest environments. Given a small set of human pilot demonstrations, the authors use imitation learning techniques to train a controller that can avoid trees by adapting the yaw angle of the UAV. The performance of the proposed system is evaluated in indoor experiments and in real natural forest environments outdoors. [193] presents a quadrotor reactive obstacle avoidance system which employs an online adaptive Convolutional Neural Network (CNN) that progressively improves depth estimation from a monocular camera in unknown environments. The depth map computed from the CNN is transformed into Ego Dynamic Space (EDS). Then, traversable waypoints are automatically computed. The presented methodology takes into account the dynamic constraints of the quadrotor. This methodology is validated through experimental indoor results.

Chapter 3

Multirotor Environment

The methodologies and algorithms developed in this thesis, as well as its simulation and experimental results, are based on a multirotor vehicle. This chapter is focused on the multirotor environment employed in this thesis, the Asctec Hummingbird quadrotor. The experimental platform is described and a mathematical model of the plant is derived. Furthermore, this chapter describes the quadrotor simulation benchmark produced within this thesis for an easier algorithm development and test. Details of the designing process and implementation of the autopilot are also given.

3.1 Experimental Platform

The experimental platform is formed by the Asctec Hummingbird vehicle (Fig. 3.1), a quadrotor with a mass of 0.698 kg and a maximum airspeed of 15 $^{m}/_{s}$. Fig. 3.2 presents a scheme of the main elements of this platform an how they are connected. The original Hummingbird platform has two on-board processors that are connected to the sensor suit, the motor controllers (ESC) and the radio control (R/C) receiver. In our platform a new on-board computer (Odroid-XU4) is included. This PC is equipped with Robot Operating System (ROS) and can communicate with the ground station through wifi. The most important components of the platform are explained in this section.

3.1.1 Sensors

Sensors are the most important elements in order to perform a correct estimation of the vehicle's state and a proper perception of the vehicle's environment. The Hummingbird vehicle has five different data sources: the accelerometers, the gyroscopes, the magnetometers, the pressure sensor and the GPS. These sensors are described next.

The Inertial Measurement Unit (IMU) of the Asctec platform, which includes the accelerometers and gyroscopes, provides measures at fast rates, up to $1000hz$. Three axial accelerometers

FIGURE 3.1: Asctec Hummingbird Quadrotor with the Odroid XU4Q on-board PC (center bottom of the UAV).

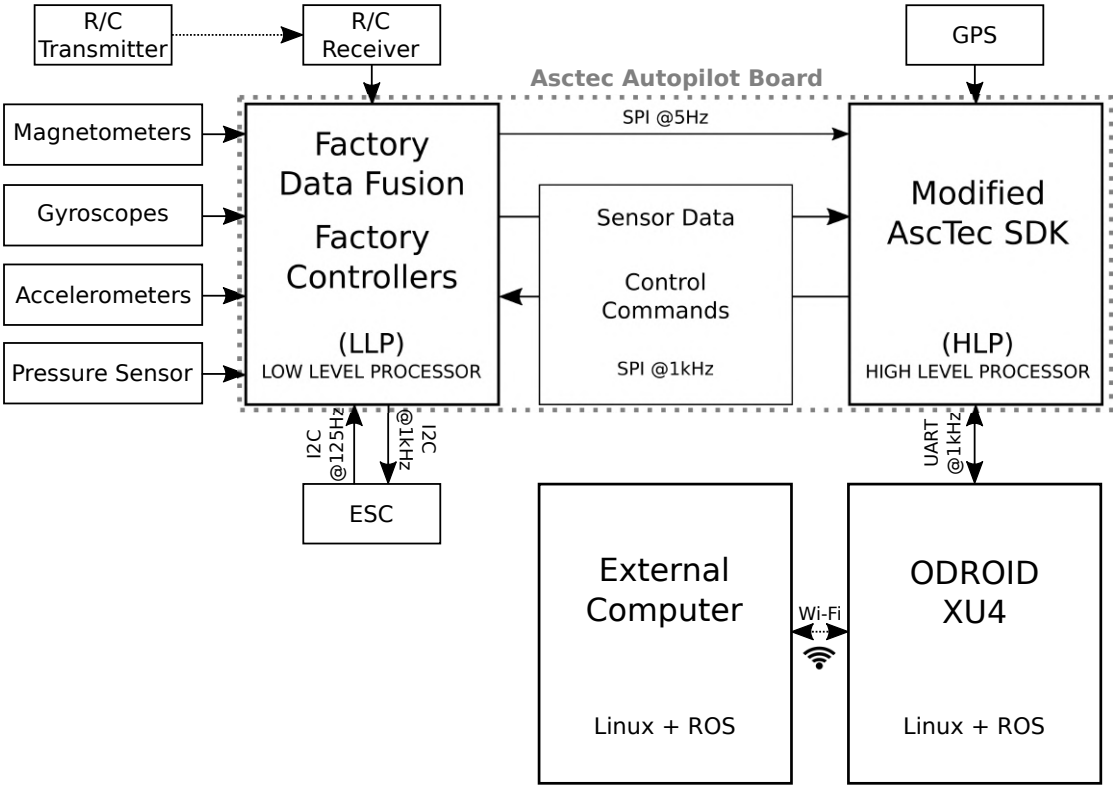

FIGURE 3.2: Scheme of the elements of the experimental platform (based on an image from wiki.asctec.de).

measure the acceleration of the forces acting on the vehicle on each axis of the body frame of reference. These forces are the generated by the rotors and the external forces. The angular velocity on each axis of the body frame is measured by the gyroscopes. The angular velocity measures provided by the IMU are treated in such a way the bias of the sensors are eliminated. That is, a software eliminates any constant or low-frequency angular velocity of the raw measurement.

The magnetometers measure the strength of the local electromagnetic field. This measurements

are easily disrupted by nearby ferric objects. Therefore, the quality of the measurements is degraded in indoor environments. From the measured magnetic field vector and the acceleration vector, the Asctec's software computes an estimation of the attitude of the angle (i.e. the Euler angles).

The pressure sensor provides an estimation of the altitude and velocity in the z axis of the world frame of reference. This sensor is considerably affected by wind gusts, which may alter the pressure in the sensor's membrane.

Finally, GPS sensor serves measures of the position with respect to the earth frame at a rate of $10hz$. This measurements are given by the latitude and longitude. The main problem of this sensor is that it depends from the access to the signals from satellites. Furthermore, satellites are not always available, such is the case of indoor or urban environments.

3.1.2 Actuators

The quadrotor is actuated by four rotors. These rotors are moved by X-BL-52S motors. Each motor is controlled by a electronic speed controller (ESC). The ESCs receive digital signals from the low-level processor. The dynamics of the controlled motors are very fast compared to the rest of the quadrotor's dynamics. Table 3.1 presents some parameters of the Asctec Hummingbird motors extracted from the motor's datasheet.

TABLE 3.1: Parameters of the X-BL-52S motor.

Description	**Value**
Weight.	0.028 kg
Radius.	0.0139 m
Motor torque constant.	0.01638 $^{Nm}/_{A}$
Maximum motor speed.	10880 rpm
Armature resistance.	1.0467 Ω
Polar moment of inertia of armature.	$1.0322 \cdot 10^{-4}$ $^{Nm}/_{repm}$
Viscous damping in motor.	0.01832 $^{Nms}/_{rpm}$
Mechanical time constant.	$5.634 \cdot 10^{-3}$ s
Motor constant.	0.854 $^{Krpm}/_{V}$
Time constant.	$5.957 \cdot 10^{-3}$ s

3.1.3 On-board Computers

The original Hummingbird platform has a flight control unit named AutoPilot Board. In addition to this unit, a new computer was included to the platform, the Odroid XU4. Both units are described in this section.

Asctec AutoPilot Board

The Autopilot Board has two processors (Fig. 3.2): the low-level processor and the high-level processor. The low-level processor receives signals from the sensors (except the GPS) and it is in charge of sending commands to the motor controllers. This unit is provided with a sensor fusion software that treats the measures received by sensors and also gives an estimation of the attitude angles. Furthermore, the low-level processor has an autopilot controller (attitude, altitude and position controllers). This processor is not accessible for the user. However, the user can control some of its functionalities through the high-level processor.

The high-level processor is connected to the GPS sensor. Also, it is connected to the low-level processor by using a SPI protocol. This processor is user-programmable through the Asctec software development kit (SKD). Nevertheless, in the experimental platform presented in this thesis, this processor is only used as a bridge to communicate the low-level processor with the the Odroid XU4, a more powerful computer that was included to the platform.

Odroid-XU4

The Odroid-XU4 (Fig. 3.3) is a small-size computer (83x58x20mm, weight: 60g) that, despite its low cost, provides very good performance. This odroid has 2GB of RAM and the Samsung Exynos5422 Cortex-A15 and Cortex-A7 CPUs. This computer was included on-board the quadrotor to enhance its capabilities, as well as to permit a better monitoring of the plant. The computer runs with linux Ubuntu 16.04LTS and it is equipped with Robot Operating System (ROS). ROS is a standard framework in robotics and aerial vehicles, supported by a large community. One of the main advantages of ROS is that permits the user to build complex programs without having a deep knowledge on the hardware system. The *asctec_mav_framework* ROS package allows the communications between the Asctec's high-level processor and the Odroid computer. That is, it works like a driver to receive and send information to the Asctec AutoPilot Board. To permit this communication, the high-level processor needs to be flashed with a specific firmware (*asctec_hl_firmware*). This package provides the information about the sensors, actuators, battery and other information that helps in the monitoring of the plant.

FIGURE 3.3: Odroid-XU4 computer (original from hardkernel.com).

3.1.4 Ground Station

The ground station of this experimental platform is equipped by a laptop PC and a R/C transmitter (Futaba FF7). The laptop runs with linux Ubuntu with the i7-8550U intel processor and 16GB RAM. The PC is also equipped with the ROS framework and it connects to the ROS core in Odroid via a wifi communication. That is, ROS nodes can be launched from this PC and topics with information of the quadrotor can be monitored as well. The R/C transmitter can be used to pilot the aircraft manually. Typically, it is used in the take off and landing manoeuvrings or in case of emergency.

Fig. 3.4 shows an image of the experimental platform. As it can be observed, it is a an outdoors platform. The picture includes the vehicle platform detailed in this section and the ground station R/C transmitter with the pilot prepared to perform manual manoeuvrings in eventual case of emergency.

FIGURE 3.4: Outdoors experimental platform.

3.2 Mathematical Model

In this section, the mathematical model of the AscTec Hummingbird quadrotor is presented. The coordinate systems as well as the axes labels and rotational conventions defined in this model are shown in Fig. 3.5. Two coordinate systems can be found: The body frame of reference $\{B\}$, which is attached to the body of the quadrotor, and the world reference frame $\{W\}$, considered inertial. Details of this model are given in next subsections.

3.2.1 Dynamic Equations

The quadrotor dynamic model is based on the standard Newton-Euler nonlinear model described in detail in [112][24]. Additionally, the equations presented here include other effects such as gyroscopic moments, drag forces and wind disturbance forces. The dynamic model has twelve

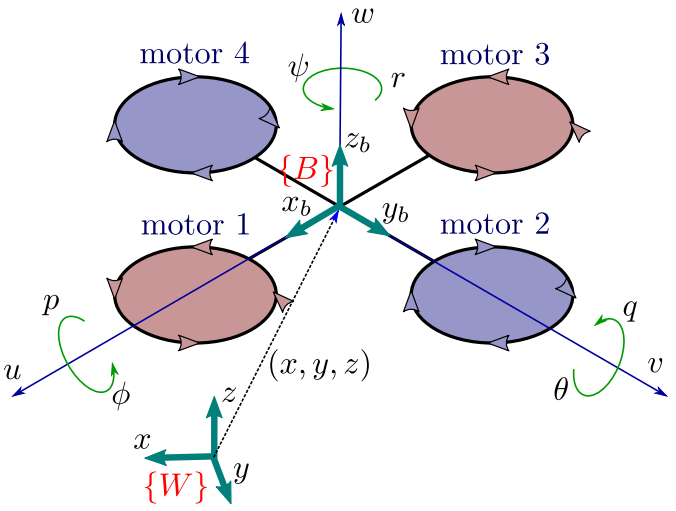

FIGURE 3.5: States conventions and frames of references of the quadrotor model.

states that are the world position (x, y and z), the Euler angles (ϕ-*roll*, θ-*pitch* and ψ-*yaw*), the body velocities (u, v and w) and the body angular velocities (p, q and r). Eqs. (3.1)-(3.4) define the state equations of the dynamic model of the quadrotor.

$$\dot{\boldsymbol{p}} = \mathbf{R}\boldsymbol{v} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^T \tag{3.1}$$

$$\dot{\boldsymbol{\Phi}} = \mathbf{H}\,\boldsymbol{\omega} = \begin{bmatrix} \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}^T \tag{3.2}$$

$$\dot{\boldsymbol{v}} = \frac{1}{m}\left(\boldsymbol{F} - \boldsymbol{F_d} - \boldsymbol{F_w}\right) + {}^{B}\boldsymbol{g} - \boldsymbol{\omega} \times \boldsymbol{v} = \begin{bmatrix} \dot{u} & \dot{v} & \dot{w} \end{bmatrix}^T \tag{3.3}$$

$$\dot{\boldsymbol{\omega}} = (\mathbf{J})^{-1}\left[\boldsymbol{\tau} - \boldsymbol{\omega} \times \mathbf{J}\boldsymbol{\omega}\right] = \begin{bmatrix} \dot{p} & \dot{q} & \dot{r} \end{bmatrix}^T \tag{3.4}$$

The position state vector (x, y and z) is calculated in Eq. (3.1). The term $\mathbf{R}$ is a rotation matrix to transform from the body to the world frame using the rotational sequence *zyx*. Eq. (3.5) shows how the construction of this matrix is made and used to transform the motion of the vehicle from the body frame to the inertial frame of reference. The resultant rotational matrix $\mathbf{R}$ is shown in Eq. (3.6).

$${}^{W}\boldsymbol{v} = \begin{bmatrix} \cos(\phi) & \sin(\phi) & 0 \\ -\sin(\phi) & \cos(\phi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\psi) & \sin(\psi) \\ 0 & -\sin(\psi) & \cos(\psi) \end{bmatrix} {}^{B}\boldsymbol{v} \tag{3.5}$$

$$\mathbf{R} = \begin{bmatrix} \cos(\psi)\cos(\theta) & \cos(\psi)\sin(\theta)\sin(\psi) - \sin(\psi)\cos(\phi) & \cos(\psi)\sin(\theta)\cos(\phi) + \sin(\psi)\sin(\phi) \\ \sin(\psi)\cos(\theta) & \sin(\psi)\sin(\theta)\sin(\psi) + \cos(\psi)\cos(\phi) & \sin(\psi)\sin(\theta)\cos(\phi) - \cos(\psi)\sin(\phi) \\ -\sin(\psi) & \cos(\theta)\sin(\phi) & \cos(\theta) \end{bmatrix} \tag{3.6}$$

Eq. (3.2) is the Euler kinematic equation that determines the rate of change of the Euler angles (ϕ-*roll*, θ-*pitch* and ψ-*yaw*) in the world frame. The basic idea of the Euler angles is to represent the orientation by decomposition the rotation in three consecutive simpler rotations about known axes. Matrix $\mathbf{H}$ relates the angular velocity of the vehicle in the body frame with the Euler angle's rate of change. This rotation matrix (Eq. 3.8) is obtained by using sequential rotation matrices with the *roll-pitch-yaw* sequence, as shown in Eq. (3.7) where angular velocity vector is obtained from the Euler angles.

$$\boldsymbol{\omega} = \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} + \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & \sin(\phi) \\ 0 & -\sin(\phi) & \cos(\phi) \end{bmatrix} \left(\begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) \\ 0 & 1 & 0 \\ \sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} \right) = \mathbf{H}^{-1} \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} \tag{3.7}$$

$$\mathbf{H} = \begin{bmatrix} 1 & \tan(\theta)\sin(\phi) & \tan(\theta)\cos(\phi) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & {}^{\sin(\phi)}/_{\cos(\theta)} & {}^{\cos(\phi)}/_{\cos(\theta)} \end{bmatrix} \tag{3.8}$$

Eq. (3.3) is the linear velocity state (u, v and w) equation. They are the linear velocities that correspond to the x, y and z axis of $\{B\}$, respectively. $\boldsymbol{F}$ represents the thrust forces. $\boldsymbol{F}_d$ and $\boldsymbol{F}_w$ are the drag and wind disturbance forces, expressed in the body axis. These forces are explained in Section 3.2.3. The term ${}^{w}\boldsymbol{g}$ represents the gravity acceleration constant expressed in the body axis. Eq. (3.4) defines the angular velocity state update. p, q and r are the rate change of *roll*, *pitch* and *yaw*, respectively, represented in the body frame.

Eq. (3.9) shows the thrust force generated by the ith motor. ω_{mi} is the rotational velocity of motor i, c_T is the thrust coefficient, ρ is the air mass density and A is the area of the rotor. k_T is the thrust constant (${}^{1}/_{2}c_T\rho A$), which relates the square rotational velocity of the motors in *rpm* with the generated thrust force. The thurst force generated by the four motors of the quadrotor is shown in Eq. (3.10). This force is always parallel to the z body axis. Eq. (3.11) defines the generated aerodynamic, gyroscopic and thrust moments on each body axis. The four motor rotational velocities are considered the inputs of the dynamic model.

$$T_i = \frac{1}{2} c_T \rho A (\omega_i)^2 = k_T (\omega_i)^2 \tag{3.9}$$

$$\boldsymbol{F} = \begin{bmatrix} 0 & 0 & k_T \left(\omega_{m1}^2 + \omega_{m2}^2 + \omega_{m3}^2 + \omega_{m4}^2 \right) \end{bmatrix} \tag{3.10}$$

$$\boldsymbol{\tau} = \begin{bmatrix} d_m \, k_T \, \omega_{m2}^2 - d_m \, k_T \, \omega_{m4}^2 + J_m \, q \, (\pi/30) \left(-\omega_{m1} + \omega_{m2} - \omega_{m3} + \omega_{m4} \right) \\ -d_m \, k_T \, \omega_{m1}^2 + d_m \, k_T \, \omega_{m3}^2 + J_m \, p \, (\pi/30) \left(\omega_{m1} - \omega_{m2} + \omega_{m3} - \omega_{m4} \right) \\ -k_Q \, \omega_{m1}^2 + k_Q \, \omega_{m2}^2 - k_Q \, \omega_{m3}^2 + k_Q \, \omega_{m4}^2 \end{bmatrix} \tag{3.11}$$

The parameters of the model are defined in Table 3.2 and introduced in Section 3.2.4, where the parameter identification process is described.

3.2.2 Motor Model and Control Mixing

The inputs of the dynamic model are the velocities of the motors (ω_{mi}). However, in the real AscTec Hummingbird platform, the inputs are four digital signals limited from 0 to 200 that are associated to thrust (u_z) and to roll (u_ϕ), pitch (u_θ) and yaw (u_ψ) rates. These inputs are related to the square of the rotational velocity reference of the motors, $\omega_{n,ri}$, by Eq. (3.12).

$$\begin{bmatrix} {\omega_{n,r1}}^2 \\ {\omega_{n,r2}}^2 \\ {\omega_{n,r3}}^2 \\ {\omega_{n,r4}}^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 & -1 \\ 1 & 1 & 0 & 1 \\ 1 & 0 & 1 & -1 \\ 1 & -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} u_z \\ u_\phi - 100 \\ u_\theta - 100 \\ u_\psi - 100 \end{bmatrix} \tag{3.12}$$

The motor's rotational velocity references ($\omega_{n,ri}$) obtained from Eq. (3.12) are normalised from 0 to 200. Thus, they are still digital signals. According to the Asctec's manual, the velocity reference of the motors in *rpm* (ω_{r1}) can be obtained from the linear expression of Eq. (3.13), being 8600 *rpm* the maximum velocity and 1075 *rpm* the minimum. The process of calculating these motor velocity references from the quadrotor digital inputs is performed by a software block known as the control mixing.

$$\omega_{ri} = 37.625 \omega_{n,ri} + 1075 \quad \text{rpm} \tag{3.13}$$

The motors of the quadrotor are controlled to follow the references computed by the control mixing block. The dynamics of the motors are modelled by a first order differential equation. Real velocities of the motors, ω_{mi}, are obtained by Eq. (3.14).

$$\omega_{mi} = \frac{\omega_{ri}}{\tau_m s + 1} \tag{3.14}$$

3.2.3 Other Effects

In addition to the basic dynamic behaviour, other effects are included in the model. For instance, the gyroscopic effects ($\boldsymbol{\tau}_{gyr}$), apparent in the moment vector (Eq. 3.11) with the terms shown in Eq. (3.15).

$$\begin{aligned} \tau_{gyr,x} &= J_m\, q\, (\pi/30)\, (-\omega_{m1} + \omega_{m2} - \omega_{m3} + \omega_{m4}) \\ \tau_{gyr,y} &= J_m\, p\, (\pi/30)\, (\omega_{m1} - \omega_{m2} + \omega_{m3} - \omega_{m4}) \end{aligned} \tag{3.15}$$

The drag and wind disturbance forces are obtained by a simple aerodynamic equation (Eq. 3.16) which calculates the drag experienced by the quadrotor due to movement through the air. This force is calculated from the flow velocity relative to the vehicle, where ρ is the air mass density, S is vehicle's surface on the xz plane, c_D is the drag coefficient and $\boldsymbol{v}_w$ is the wind velocity vector.

$$\boldsymbol{F}_d + \boldsymbol{F}_w = \frac{1}{2}\rho S c_D (\boldsymbol{v} - \boldsymbol{v}_w)^2 \tag{3.16}$$

3.2.4 Parameter Identification

The parameters of the model (Table 3.2) are obtained from the Hummingbird vehicle. The mass of the vehicle and distances are directly measured. Motor parameters are acquired from the datasheet of the motor (see Section 3.1.2).

The thurst constant (k_T) is computed from the data obtained with different experimental tests in Hover (i.e. experiments around the equilibrium point). Knowing the mass of the vehicle and the average rotational velocity of the motors during these experiments, k_T can be obtained as shown in Eq. (3.17). Fig. 3.6 shows the normalized speed of each motor during a hover experiment of 1 minute. In this experiment the average normalized speed is 98.2094, which is equivalent to 4760.3 *rpm* (Eq. 3.13), and applying Eq. (3.17) we obtain a k_T of $7.5543 \cdot 10^{-8}$. Taking into account more Hover experiments in different conditions permits to obtain a more accurate value of $k_T = 7.1103 \cdot 10^{-8}$. On the other hand, torque constant (k_Q) is obtained from [87], where aerodynamic parameters of Hummimgbird quadrotor are estimated through wind tunnel tests.

$$k_T = \frac{mg}{\overline{\omega}_{m1}^2 + \overline{\omega}_{m2}^2 + \overline{\omega}_{m3}^2 + \overline{\omega}_{m4}^2} \tag{3.17}$$

The mass moment inertia matrix has been calculated applying the Huygens–Steiner theorem assuming body of the quadrotor as solid cylinders, the arms as cylindrical rods, the ESCs as flat plates and the motors as solid cylinders. Masses and dimensions of each component are measured directly on the vehicle. The obtained mass moment inertia matrix is shown in Eq. (3.18).

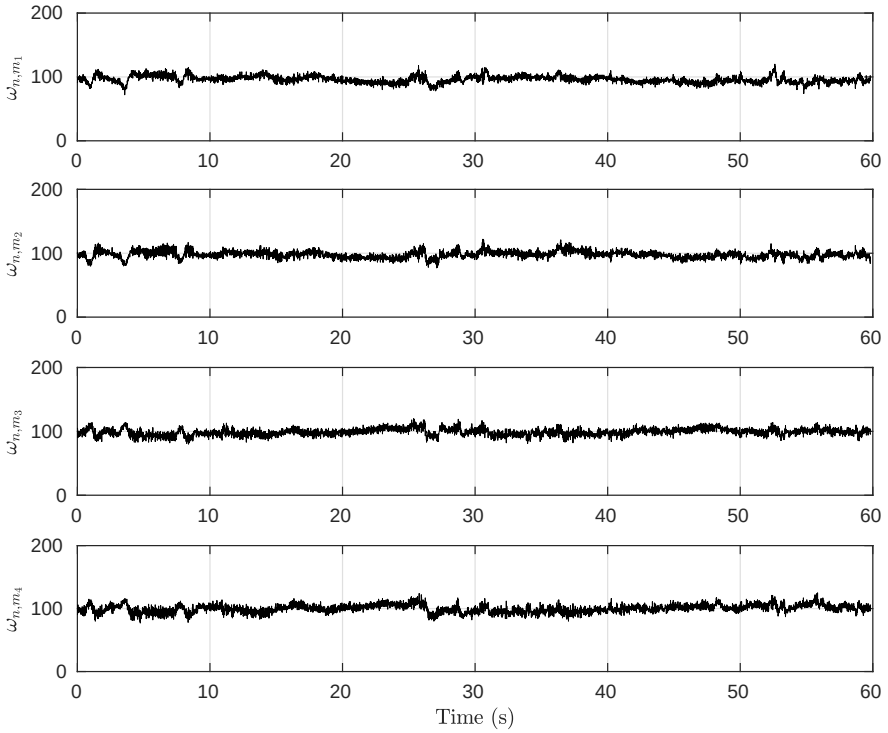

FIGURE 3.6: Normalized motor speed in a hover experiment.

$$\mathbf{J} = 10^{-3} \begin{bmatrix} 3.4313 & 0 & 0 \\ 0 & 3.4313 & 0 \\ 0 & 0 & 6.002 \end{bmatrix} kg\,m^2 \tag{3.18}$$

TABLE 3.2: Parameters of the model.

Symbol	Description	Value
m	Mass of the quadrotor.	0.698 kg
$\mathbf{J}$	Mass moment of inertia matrix.	Eq. (3.18)
d_m	Distance from a motor to the c.o.g.	0.171 m
J_m	Inertia of each motor's rotating components.	$1.302 \cdot 10^{-6}$ $kg\,m^2$
τ_m	Motor time constant	$5.634 \cdot 10^{-3}$ s
k_T	Thrust constant.	$7.1103 \cdot 10^{-8}$
k_Q	Torque constant.	$1.0088 \cdot 10^{-9}$

3.3 Autopilot

This thesis is not focused in the study of the stabilization problem. However, it is necessary to implement an autopilot controller if separated guidance and control PF strategies are going to

be studied. An autopilot based on a set of *Proportional-Integral-Derivative* (PID) controllers is developed. Fig. 3.7 shows the control structure of the autopilot with the path following controller. The autopilot is formed by a velocity controller and an altitude & attitude controller. The controllers have been tuned by pole-placement technique and re-adjusted experimentally in the real platform. Table 3.3 presents the final parameters of these *PID*s. Altitude controller includes a feed-forward gravity offset term, calculated in Eq. (3.19), that is added to the PID output. This autopilot with the parameters of Table 3.3 is used in the simulation and experimental results presented in this thesis.

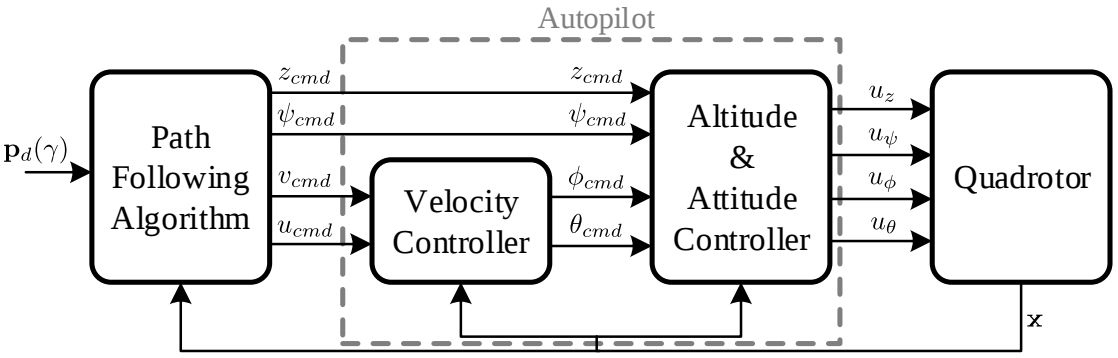

FIGURE 3.7: SGC structure for path following algorithms.

$$g_{offset} = \frac{\sqrt{mg/4k_T} - 1075}{37.625} \tag{3.19}$$

TABLE 3.3: Constants of the *PID* controllers of the autopilot.

	Kp	**Ki**	**Kd**
$\boldsymbol{u}$	0.32	-	0.1
$\boldsymbol{v}$	0.32	-	0.1
$\boldsymbol{\phi}$	4	2.2	2.4
$\boldsymbol{\theta}$	4	2.2	2.4
$\boldsymbol{\psi}$	8	1	7
$\boldsymbol{z}$	4	2.2	6.6

This autopilot is programmed as a ROS package. Four ROS nodes are implemented: attitude controller at 100Hz, velocity controller at 50Hz, altitude controller at 20Hz and path following controller at 5Hz. PID controllers of the autopilot are discretized using the Tusting approximation. This package has a modular structure in the sense that each of the control blocks can be easily activated/deactivated or substituted by another controller. Also, new controllers can be incorporated to the structure straightforwardly.

3.4 Path-Flyer: A Benchmark of Quadrotor Path Following Algorithms

Before implementing and testing a control approach experimentally, its performance and functionality should be studied from numerical simulations. Moreover, nowadays the solutions proposed for problems as the obstacle or collision avoidance, given the impact and costs of an experimental failure, are only validated in simulation. For these reasons, it becomes extremely important to have a complete, realistic and validated simulation platform of the multirotor and its environment.

There exist various real-time simulators implemented in ROS, as for instance the RotorS Gazebo UAV Simulator [53]. ROS is a programming framework for robots and autonomous vehicles, but is not focused on simulation tasks. Therefore, programming in ROS can become tedious for those users that only require a simulator. Apart from ROS-based platforms, there are few simulation models described in the literature but, most of them are not available on-line, they are little user-friendly, little customizable and very general purpose, in the sense of not being focused to any particular control problem. Nevertheless, there can still be found some interesting simulation platforms dedicated to different problems such, for instance, visual tracking [126] or path planning [174]. However, to the best of my knowledge, there is no simulation platform aimed to the path following problem, which is one of the main research topic of this thesis.

A simulation benchmark, named Path-Flyer [146], is developed with the double objective of procuring a platform to perform simulations presented in this thesis and provide to the community a user-friendly quadrotor simulation tool. Path-Flyer, is a simulation benchmark for testing path following algorithms on a quadrotor vehicle. This benchmark allows to compare multiple path following algorithms in the same scenarios with flexibility. The model of the vehicle is complete and realistic. An identification process has been carried out to obtain its parameters. Also, the model has been validated experimentally. Path-Flyer is modular and programmable, meaning that new algorithms, blocks and/or simulation scenarios can be easily incorporated. For all these reasons, I consider this platform to be of interest for research as well as for education. What is more, it has been successfully used as a support tool for lab sessions in various subjects at the Automatic Control Department in UPC.

Path-Flyer accomplishes the next key features:

1. The mathematical model of the quadrotor and its environment is complete, realistic and experimentally validated.
2. The benchmark is designed to allow the incorporation of new path following algorithms and reference paths with ease.
3. The benchmark facilitates to change simulation scenarios and it helps the user to explore and analyse relevant information of simulation results.

Path-Flyer is built upon the Quad-Sim platform, a MatLab simulation tool developed in Drexdel University for simulation and control of quadrotors [65]. To this end, the objective is to improve

and modify this tool to have a realistic mathematical and dynamic model of the quadrotor and its environment, to implement a proper structure to solve the path following problem and include various PF algorithms, and to incorporate new functionalities suited for this benchmark.

3.4.1 Quadrotor Model

The mathematical model implemented in this platform is exactly the same as the one described in Section 3.2. That is, the same dynamic and motor equations, as well as the same control mixing and parameters are used. The magnitude and direction of the wind velocity vector, $\boldsymbol{v}_w$ apparent in Eq. (3.16), has been modelled as a random walk by adding a constant value to the integral of a band-limited white noise. An example of the resultant wind disturbance signal is shown in Fig. 3.8. This random walk is fully customizable. Furthermore, the user can decide whether to perform simulations with wind disturbance or without. User can also program their own wind profile.

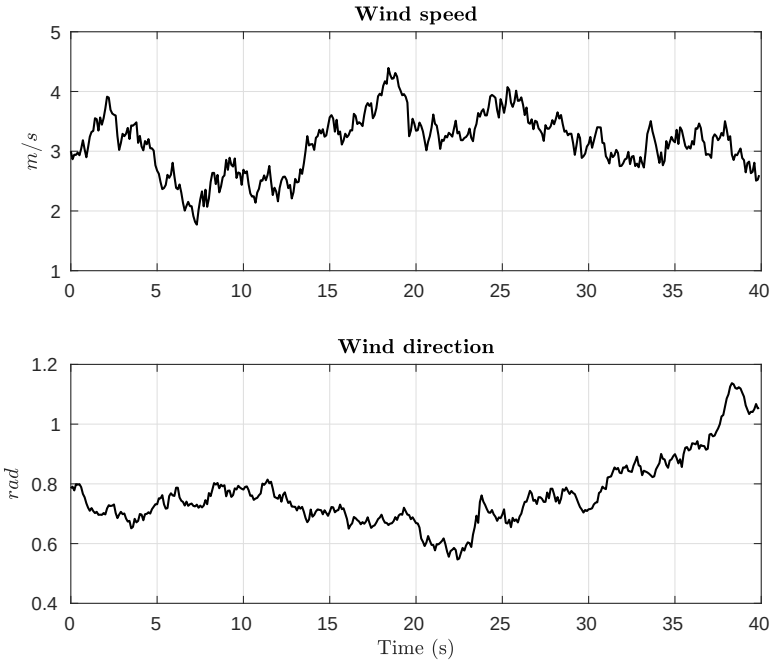

FIGURE 3.8: Wind disturbance generated with a random walk.

On the other hand, a band-limited white noise signal is added to each of the twelve measured states. Those signals have been adjusted to resemble the noise of the sensors of the real quadrotor platform. Moreover, Path-Flyer allows the user to customize each noise signal and to enable each one separately.

3.4.2 Implemented Algorithms

Four path following algorithms are implemented in this benchmark: *Non-Linear Guidance Law* (*NLGL*), *Carrot-Chasing* and an adaptive version of both algorithms. These are geometric

algorithms, which means that they compute the control actions from geometric relations. More information about these two algorithms and the adaptive approaches is found in Sections 4 and 5.

Since the implemented algorithms present a separated guidance and control structure, Path-Flyer uses the same structure of Fig. 3.7. Furthermore, the same Autopilot described in Section 3.3 is implemented. The quadrotor state vector, $\mathbf{x}$ $(x, y, z, u, v, w, \phi, \theta, \psi, p, q, r)$, is assumed to be available for every control block.

Non-Linear Guidance Law

This algorithm is based on the Virtual Target Point (VTP) concept. That means that the vehicle is moved towards a point on the path (the VTP) which is constantly being updated. In *NLGL* [133] the VTP is calculated as the point on the path that is at a distance L from the vehicle (assuring that γ at $VTP > \gamma$ at $\boldsymbol{p}_{d_{min}}$), as shown in Fig. 3.9. Summarizing, the vehicle is required to travel at a reference speed, V_{ref}, on the x-body axis and the vehicle's direction is controlled with the *yaw* angle, which is always controlled to face the vehicle to the VTP. The altitude reference (z_{cmd}) is set as the altitude of the point on the path with minimum distance to the vehicle (z at d_{min}).

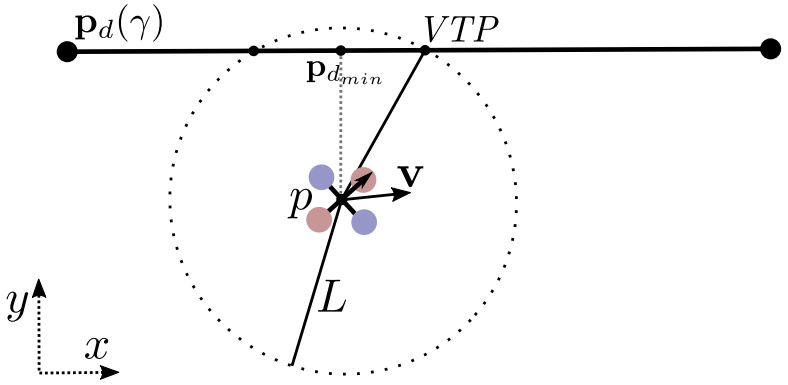

FIGURE 3.9: NLGL: Obtaining the VTP for a straight line path.

Carrot-Chasing

Carrot-Chasing algorithm [122] is similar to *NLGL*. The main difference is that the target point is calculated differently, as shown in Fig. 3.10. That is, the VTP is obtained as the point that is at a distance δ along the path from the point on the path at a minimum distance to the vehicle ($\boldsymbol{p}_{d_{min}}$). This slight difference results in a variation of the path following performance and the convergence time. The rest of the algorithm works the same way as *NLGL*.

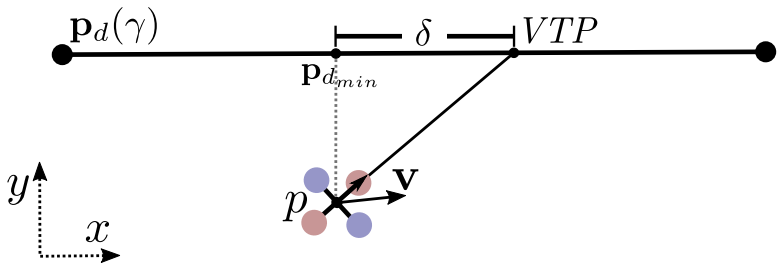

FIGURE 3.10: Carrot-Chasing: Obtaining the VTP for a straight line path.

Adaptive-NLGL and Adaptive-CC

Both geometric algorithms have one parameter to tune (L and δ). These parameters define the performance of the path following algorithm and their optimal selection depends on variables such as the velocity of the vehicle or the path's shape [172][70]. In the present thesis, an adaptive approach of the *NLGL* algorithm is developed (see Chap.5). The *Adaptive NLGL* uses a Neural Network to calculate the parameter L from the two mentioned external variables. This approach is also included in the Path-Flyer, as well a similar approach for the *Carrot-Chasing* algorithm.

3.4.3 Functionalities

Path-Flyer is formed by several Simulink blocks (Fig. 3.11). These are, the blocks which implement quadrotor dynamics, control mixing and altitude & attitude controller, that were present in the original simulator but have been modified in this benchmark, and velocity controller and path following controller blocks, which have been aggregated to the platform. Each of these blocks is implemented as explained in previous sections.

Also, there are additional blocks included in the original Quad-Sim platform that implement different functionalities. Among the original functionalities the Quad-Sim platform permits to change model parameters or create an initial conditions vector. It presents plots for the evolution of all the states and motor velocities and it includes a simple 3D flight animation that displays the evolution the orientation and trajectory of the quadrotor along the last simulation.

As already mentioned, the functionalities that have been incorporated in Path-Flyer are the noise on the measurements and the wind disturbance, both customizable features. These two features are present in the quadrotor dynamics block. Furthermore, the platform is provided with a user interface that gives the user the possibility to set diverse simulation parameters and analyse significant information of simulation results.

User Interface

The user interface, shown in Fig. 3.12, is divided in two parts: the Path Following Control (left) and the Path Following Evaluation (right). In the first one the user can select, from a list, the path following algorithm and the reference path to be simulated. Four reference paths are

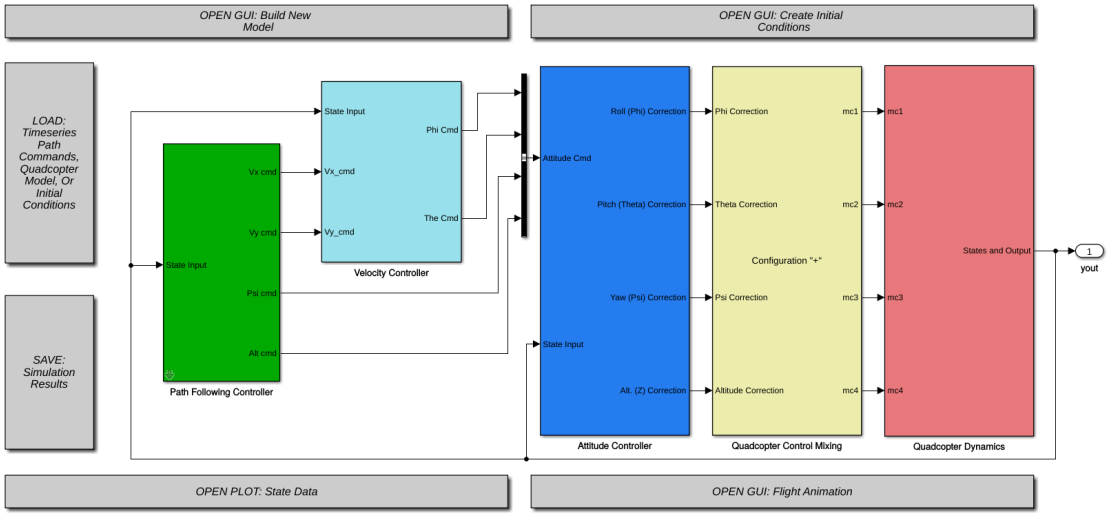

FIGURE 3.11: Benchmark simulink blocks.

available: circle, eight-shaped, spiral and a line plus circle path. Also, it is possible to modify the path or the path amplitude and length.

The Path Following Evaluation permits the user to run an evaluation after a simulation. That is, the user can select which plots to be generated, for instance the three-dimensional trajectory followed by the vehicle, the path distance error or the control effort. Also, it gives values of relevant information of simulation results, such as the total time that the vehicle took to complete the path or the average path distance error.

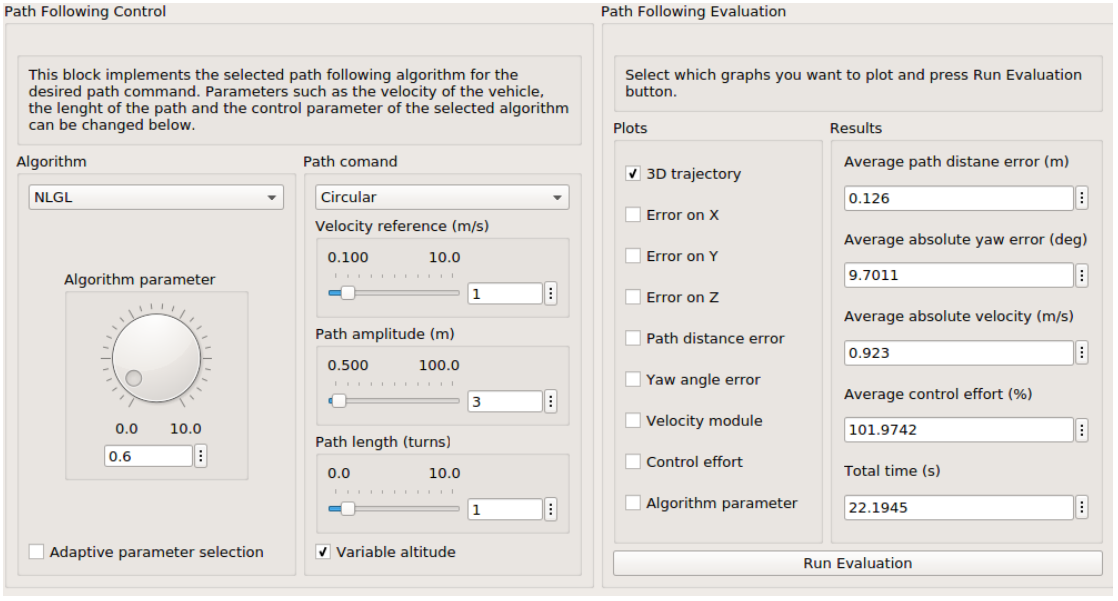

FIGURE 3.12: User interface.

Program your own Algorithm and Path

One of the most important characteristics of the Path-Flyer benchmark is that it has been prepared to add new algorithms in the platform straightforwardly. That is, users only need to program their algorithms on a pre-structured matlab function which already has the set of inputs and outputs defined. Next, this function needs to be called in another matlab file and the new algorithm will be then available.

Similarly, Path-Flyer also allows to program new reference paths. The user needs to define the reference path (inside a *case* condition) on a matlab function where all paths are defined.

Repository and Documentation

Path-Flyer is available online [142]. It has been uploaded followed by a set of documentation that, among other things, explains the user how to add new algorithms and new reference paths, how to use the user interface and how to modify parameters such as initial conditions, wind disturbance profile and measurements noise.

3.4.4 Validation of the Platform with Real Experiments

To validate the benchmark, real experiments have been replicated with the Path-Flyer platform, and both simulation and experimental results have been compared. Experiments consist on following a Lemniscate path (Eq. 3.20) of 4 meters of amplitude (A) at a velocity reference of 1 $^{m}/_{s}$ with the NLGL algorithm ($L = 2m$).

$$\begin{aligned} x_d &= 2A\cos(\gamma) \\ y_d &= A\sin(2\gamma) \end{aligned} \tag{3.20}$$

The experiments are performed with the experimental platform described in Section 3.1. Attitude measures are estimated by the IMU sensor, position and velocity measures on the xy plane are obtained by the GPS sensor and altitude measure is obtained by the pressure sensor. The autopilot presented in Section 3.3 is used in both real and simulation tests. It is important to mention that the response obtained by the inner-loop controllers in the experiments is very similar, in terms of settling time, overshoot and steady-state error, to the one obtained in the Path-Flyer. This is shown in Fig. 3.13 and Fig. 3.14, where the performance of *pitch* and *yaw* is tested with step references.

Fig. 3.15 shows the trajectory of the vehicle in the real experiment (red), in the Path-Flyer platform (blue) and the path reference (black). Note that real experiments present wind disturbances, for this reason simulations have been carried out including a wind disturbance signal according to the meteorological data at the day and hour of the experiment (wind magnitude: $3^{m}/_{s}$, direction: North-East -xy-). Magnitude and direction have been set in the benchmark generating a random realistic wind signal from these parameters.

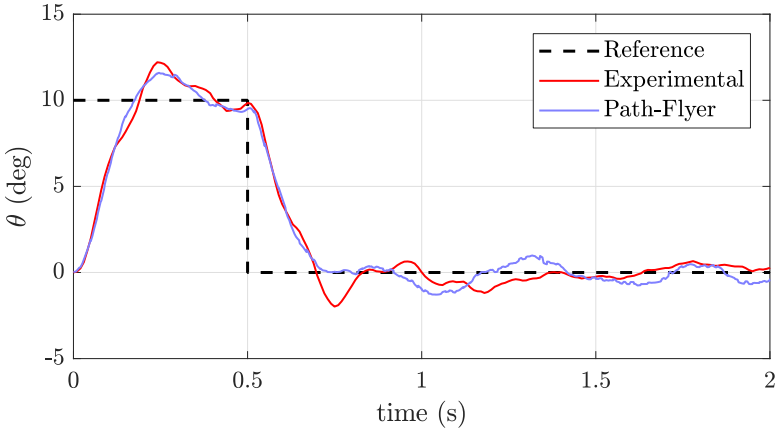

FIGURE 3.13: Results performing a 10 degrees step during 0.5 seconds of *pitch*: Experimental (red) and simulation (blue).

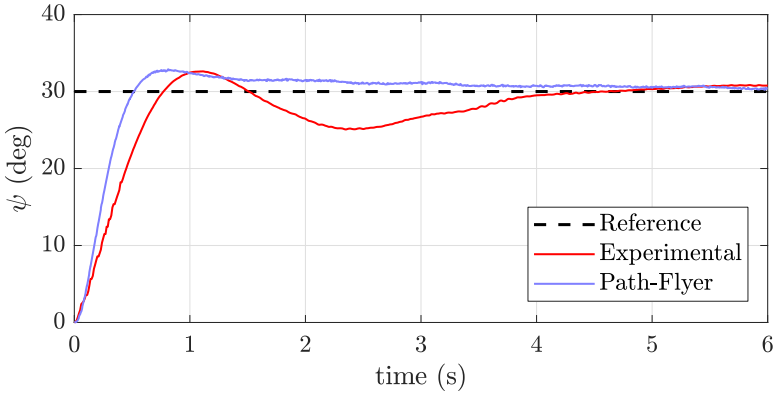

FIGURE 3.14: Results performing a 30 degrees step of *yaw*: Experimental (red) and simulation (blue).

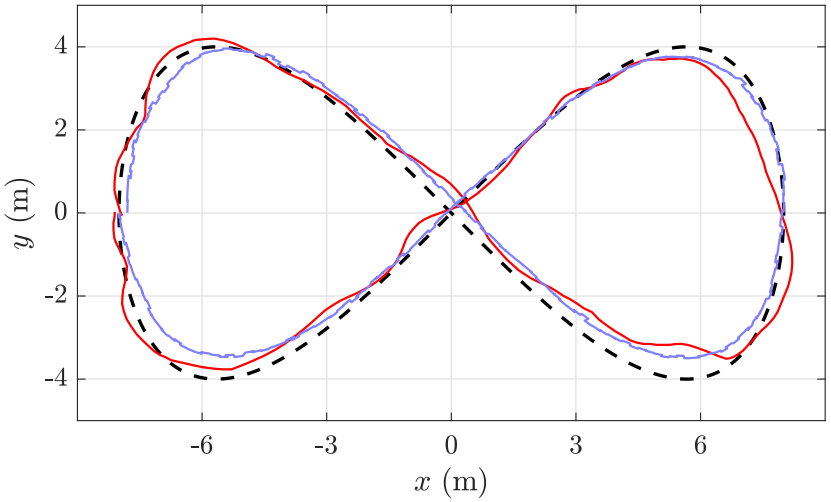

FIGURE 3.15: Trajectory on the xy plane of *NLGL* algorithm: Benchmark simulation with wind (blue) and experimental (red) (Eight-shape path reference (black), $V_{ref} = 1^{m}/_{s}$).

Table 3.4 shows relevant parameters of Fig. 3.15 tests. These parameters are: the total time needed to perform a lap to the path, the mean distance to the path error ($\overline{d}$), the mean *yaw* error ($\overline{|e_\psi|}$) and the mean velocity of the vehicle ($\overline{\|\boldsymbol{v}\|}$).

TABLE 3.4: Results for one lap of the Lemniscate with $NLGL$: Experimental and Simulation.

	time (s)	$\overline{\mathbf{d}}$ **(m)**	$\overline{\mathbf{\lvert e_\psi \rvert}}$ **(deg)**	$\overline{\mathbf{\lVert v \rVert}}$ ($^m/_s$)
Experimental	57.9986	0.2216	3.8503	0.8493
Path-Flyer	50.8911	0.2047	1.923	0.9631
Path-Flyer (no wind)	47.4449	0.1727	1.983	0.97481

As observed in Table 3.4, the values obtained by the Path-Flyer simulation are quite similar to the ones exhibited in the real experimental results. Particularly, the distance to the path error parameter, which is the main indicator of the path following performance. Considering that the uncertainty of the sensors of the plant is not taken into account in the model, but is present in real experiments (especially in the GPS sensor), the authors consider that the presented experiments prove the validity of the Path-Flyer benchmark.

3.5 RotorS

The simulation results presented in Chaps. (4) and (5) are performed with the Path-Flyer benchmark, described in Section 3.4. However, in Chaps. (6) and (8) another multirotor simulator is used to perform the simulation and training of the developed deep reinforcement learning approaches. This is RotorS, a simulator built in the Gazebo/ROS platform. Since the proposed approaches are programmed in the tensorflow/python environment, integrating them into the Gazebo/ROS framework rather than in MatLab presents some advantages, as discussed in Chap. (6).

RotorS [53] is a realistic multirotor simulator. It includes, within the available vehicle models, a model of the Asctec Hummingbird, the quadrotor employed in this thesis. This simulator has a graphic interface, built in Gazebo, and permits the incorporation of complex sensors, such as cameras or LIDARs, as well as the generation of different objects, obstacles and urban or indoor environments. It provides both ground truth measurements (ideal sensors) and the measurements of the modelled sensors. This sensor models can be modified or new sensors can be included.

The autopilot ROS package developed in Section 3.3 is also implemented in this platform, reporting a similar performance as in the real experimental results. Since the names and structure of the topics (messages in ROS) of RotorS do not correspond to the ones of the real platform, in order to use the same exact ROS code of the Autopilot, a ROS node that acts as a driver between RotorS and the Autopilot package was programmed. This node generates the topics with the same structure, name and format that the *asctec_mav_framework* package (Section 3.1.3) generates. This node can either provide ground truth or sensor measurements, depending on the value of an argument of the node's function. Furthermore, another node that emulates the

interaction of the R/C transmitter present in the real platform is also included in the RotorS environment.

A scheme of the structure of the nodes and topics programmed in the RotorS environment is presented in Fig. 3.16. This scheme shows each of the nodes implemented in this simulation environment represented as a single block. The set of topics generated by each of these nodes is also included inside every block. The RotorS platform is included in the scheme as well. The RotorS block represents the UAS of the control structure in Fig. 2.1. RotorS is formed by several nodes, however, in this scheme it is represented as a unique block. Furthermore, RotorS generates a large number of topics but only the topics that are used in this thesis are included. This topics are, essentially, the topic of the LIDAR sensor, the topics of the other modelled sensors, the topics that provide the ground truth measurements and the topic that receives the commands of the quadrotor's motors speeds.

The LIDAR is the sensor added in the platform to detect obstacles in the vehicle's route. It is used to perform the perception task. A realistic model of the LIDAR sensor that is installed on the real quadrotor is employed to generate the LIDAR topic. More information about the selection, installation and modelling of this sensor is found in Section 7.

The *translate_topics* node communicates the RotorS platform with the guidance and control nodes. That is, this node generates the topics with the same structure, name and format of those in the real platform. The topics that this node publishes as well as the set of topics of RotorS that it is subscribed to are denoted in light green in the scheme of Fig. 3.16. Note that two groups of topics of RotorS are highlighted with this color. These correspond to the set of topics that provide the measurements of the modelled sensors and the set of topics that provide the ground truth measurements. The scheme of Fig. 3.16 presents the structure when ground truth measurements are used. Nevertheless, it is important to mention that, when sensor measurements are used, another node is required. This node computes the position of the vehicle in the world frame (x and y coordinates) from the latitude and longitude measurements received by the gps sensor. The *emulate_rcdata* node is used to emulate the data of a R/C transmitter. The topics generated by both the *translate_topics* and *emulate_rcdata* nodes are sent to the rest of the nodes that form the GNC structure; *translate_topics* provides the measurements and *emulate_rcdata* is used to manage the mission control. That is, mission control in the sense that the R/C data is used to define when to perform trajectory control, hover control or manual control (controlled manually by a pilot), just as in the real experiments. The R/C data can be modified by the user by using different ROS services.

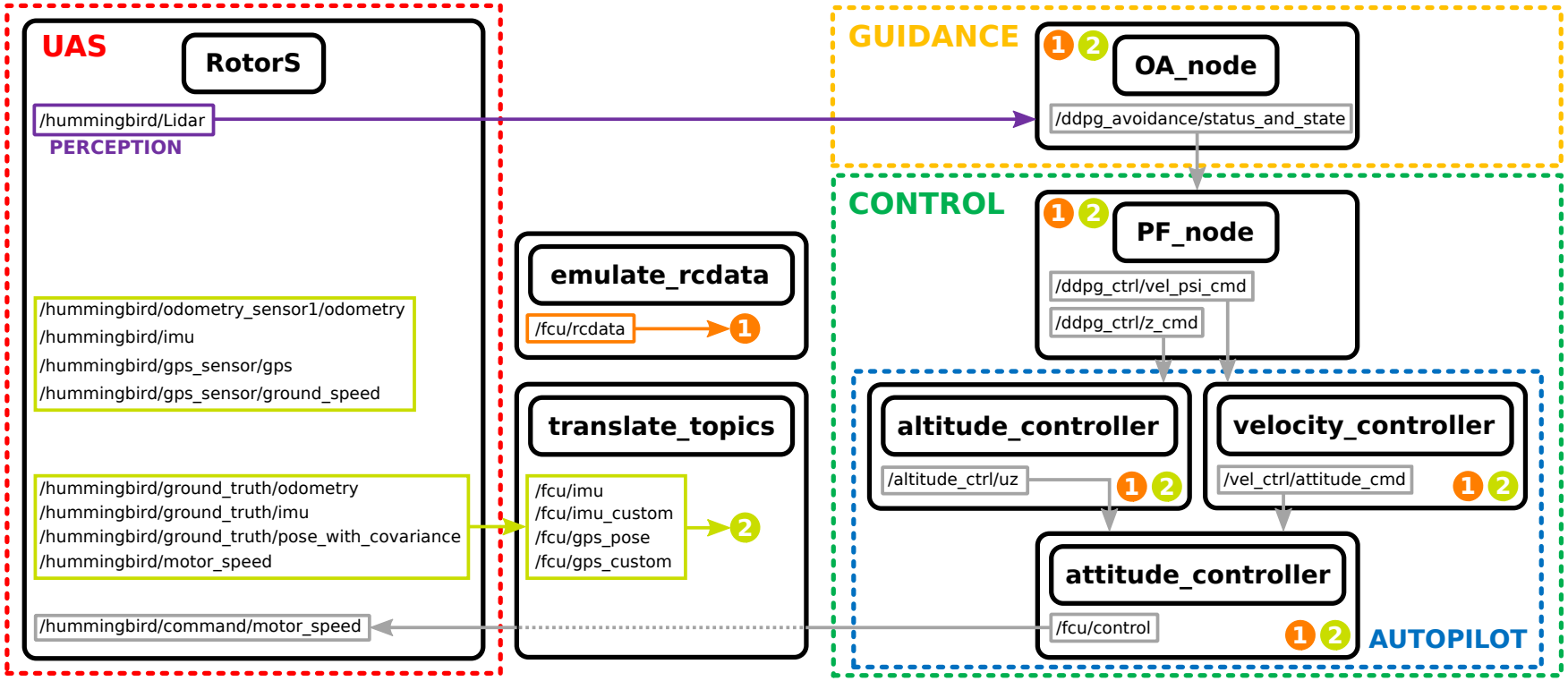

FIGURE 3.16: Scheme with nodes and topics programmed in the RotorS environment.

The rest of the nodes form the Guidance, Navigation and Control structure (Fig. 2.1). The control block is formed by the PF algorithm (*PF_node*) and the autopilot. The path following block sends commands of velocities, *yaw* angle and altitude to the velocity controller and altitude controller of the autopilot. These two controllers provide the angle commands and the altitude action (u_z) to the attitude controller, respectively. Then, the attitude controller generates the digital inputs of the Asctec Hummingbird vehicle. These inputs are treated in the *translate_topics* node, which computes the commands of the motor velocities that are sent to the RotorS block. The guidance block is formed by the obstacle avoidance node (*OA_node*). This node sends a topic to the path following block that contains the path following state, used for knowing the path to follow, and the status, that tells whether to perform path following control or hover control.

More information about the design, training, implementation and results of the developed deep reinforcement learning approaches for path following and for obstacle avoidance is found in Chaps. (6) and (8), respectively.

Part II

Control

Chapter 4

Control-oriented and Geometric Path Following

The path following problem consists on following a trajectory defined in space without any temporal constraint. Path following can be solved by implementing a separated guidance and control structure (SGC) or with a integrated guidance and control structure (IGC). The SGC control is divided in two elements: the path following and the autopilot. The autopilot is the inner controller that is in charge of tracking the commands generated by the path following controller. In this thesis, the autopilot is solved by a set of PID-based controllers (Section 3.3). The main contributions of this thesis in the control part are made in the path following problem. More information about the control structure and the path following problem are found in Chap. (2) where an exhaustive literature review of UAV path following algorithms is presented.

In this chapter, four path following algorithms have been chosen from the literature review made in Section 2.2 to be implemented and compared. They were selected for their popularity and performance. These algorithms are described in detail, harmonizing the nomenclature of different approaches, and then, they are applied to the realistic quadrotor simulation model. A comprehensive comparison of the PF algorithms on the same scenario in equal conditions is provided. This chapter is divided in two sections; first, the implementation of the algorithms, and then, the comparison with simulation results.

4.1 Implementation of Path Following Algorithms on a Quadrotor

In this section, four path following algorithms are implemented on the quadrotor vehicle. First, *Backstepping* and *Feedback Linearization*, which are the two most referenced algorithms applied to quadrotors, as reported in Section 2.2. Next, *Non-Linear Guidance Law* (*NLGL*) and *Carrot-Chasing* geometric algorithms.

In the reviewed literature, no application to quadrotors has been reported of the *NLGL* and *Carrot-Chasing* algorithms. However, they are commonly used on fixed-wing vehicles [172][133], as well as on Unmanned Ground Vehicles (UGV) [122][130], and sometimes on other types of unmanned vehicles. They are simple and they typically provide feasible solutions with good path following performance. These geometric algorithms are very similar. They differ in how the VTP is calculated. In this chapter such simple and fruitful algorithms are implemented to evaluate their performance.

Developing my own implementation of these four algorithms will permit to evaluate and compare them under the same conditions. This is shown in the Section 4.2, where the simulation results are presented.

These algorithms are designed and tuned only considering the dynamic equations of the body (Eqs. 3.1-3.4). Thus, the equations of the motors, control mixing and effects such as gyroscopic effects or wind disturbance have been omitted to perform the calculations to obtain the control laws. However, to validate the algorithms properly, they are taken into account in the simulation model. The inputs of this model are the total thrust (T) and the torques applied on each rotational axis ($\tau_\phi, \tau_\theta, \tau_\psi$). Eq. (4.1) presents the relation between the rotation speed of the motors (ω_{m1-4}) and these inputs.

$$\begin{bmatrix} T \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} k_T & k_T & k_T & k_T \\ 0 & d\,k_T & 0 & -d\,k_T \\ -d\,k_T & 0 & d\,k_T & 0 \\ -k_Q & k_Q & -k_Q & k_Q \end{bmatrix} \begin{bmatrix} \omega_{m1}^2 \\ \omega_{m2}^2 \\ \omega_{m3}^2 \\ \omega_{m4}^2 \end{bmatrix} \tag{4.1}$$

4.1.1 Backstepping

The *Backstepping* controller developed in this chapter is an adaptation of the one used in [27]. Note that the UAV model used in [27] is slightly different from the one defined in Section 3.2. The frames of reference considered in each model are different and [27] uses a rotation matrix to represent the attitude.

In this algorithm, $\mathbf{S}()$ represents the skew symmetric matrix that verifies $\mathbf{S}(\boldsymbol{x})\boldsymbol{y} = \boldsymbol{x} \times \boldsymbol{y}$. And $\boldsymbol{p}_d'$ is the partial derivative of $\boldsymbol{p}_d$ with respect to γ, $\frac{\partial \boldsymbol{p}_d}{\partial \gamma}$. It is important to recall that γ is the scalar parameter of the virtual arc that parametrizes the path (Definition 2.2.1).

To solve the path following problem, four backstepping error vectors are defined, each one of three components. The first error vector, Eq. (4.2), is the position error. The second error vector, Eq. (4.3), includes a term of position error and a term of velocity error in the world frame. The sigmoidal function $\boldsymbol{\sigma}(\cdot)$, Eq. (4.4), limits the growth of $\boldsymbol{e}_2$ when there are large position errors, where p_{max} is the allowed limit. This guarantees that the actuation does not grow unbounded.

$$\boldsymbol{e}_1 = \boldsymbol{p} - \boldsymbol{p}_d(\gamma) \tag{4.2}$$

$$\boldsymbol{e}_2 = \boldsymbol{\sigma}(\boldsymbol{e}_1) + \frac{1}{k_1}\dot{\boldsymbol{e}}_1 \tag{4.3}$$

$$\sigma(x) = p_{max}\frac{x}{1+\|x\|} \tag{4.4}$$

The convergence of $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ to zero is assured by defining two control Lyapunov functions, whose derivatives are negative definite if the thrust force, T, follows T_d, Eq. (4.5), with the direction $\boldsymbol{r}_{3d}$, Eq. (4.6), where $\boldsymbol{u}_3 = \begin{bmatrix}0 & 0 & 1\end{bmatrix}^T$.

$$T_d = m\left\|k_1^2 k_2 \boldsymbol{e}_2 + k_1\dot{\boldsymbol{\sigma}}(\boldsymbol{e}_1) + g\boldsymbol{u}_3 - \ddot{\boldsymbol{p}}_d\right\| \tag{4.5}$$

$$\boldsymbol{r}_{3d} = \frac{k_1^2 k_2 \boldsymbol{e}_2 + k_1\dot{\boldsymbol{\sigma}}(\boldsymbol{e}_1) + g\boldsymbol{u}_3 - \ddot{\boldsymbol{p}}_d}{\|k_1^2 k_2 \boldsymbol{e}_2 + k_1\dot{\boldsymbol{\sigma}}(\boldsymbol{e}_1) + g\boldsymbol{u}_3 - \ddot{\boldsymbol{p}}_d\|} \tag{4.6}$$

The control law expression for the thrust force is calculated as shown in Eq. (4.7). $\boldsymbol{r}_3$ is the third column of $\mathbf{R}$, which represents the direction of the z body axis of the vehicle. Thus, if $\boldsymbol{r}_3$ is equal to the desired thrust direction $\boldsymbol{r}_{3d}$, the thrust force will be the same as the desired thrust force T_d. For this reason, the third error vector is defined as the error of the thrust direction, as stated in Eq. (4.8).

$$T = \boldsymbol{r}_{3d}^T \boldsymbol{r}_3 T_d \tag{4.7}$$

$$\boldsymbol{e}_3 = \boldsymbol{r}_3 - \boldsymbol{r}_{3d} \tag{4.8}$$

A third control Lyapunov function, which includes the first and second Lyapunov functions as well as a term for the third backstepping error, is defined. With the aim of making the first three errors tend to zero, the expression of the fourth error vector, Eq. (4.9), is defined in such a way that if this error becomes zero, the time derivative of the third Lyapunov function remains negative definite. In Eq. (4.9) $\mathbf{R}_d$ represents the desired orientation matrix and $\boldsymbol{\omega}_d$ stands for the desired angular speed of the vehicle.

$$\begin{aligned}\boldsymbol{e}_4 = &- k_3\mathbf{S}(\boldsymbol{u}_3)^2\mathbf{R}^T\boldsymbol{r}_{3d} + \mathbf{S}(\boldsymbol{u}_3)(\boldsymbol{\omega} - \mathbf{R}^T\mathbf{R}_d\boldsymbol{\omega}_d)\\ &- \tfrac{T_d}{mk_1}\mathbf{S}(\boldsymbol{u}_3)^2\mathbf{R}^T\boldsymbol{e}_2\end{aligned} \tag{4.9}$$

Finally, a fourth Lyapunov function is defined for the fourth error. To assure the stability of the system with this Lyapunov function, the expression of the angular acceleration of the vehicle is calculated as stated in Eq. (4.10). In this expression $\dot{\boldsymbol{\omega}}_c$ stands for the calculated angular acceleration and $\dot{\omega}_{3c}(t)$ is an arbitrary function that defines the dynamics of the *yaw*

angle. Then, the control law of the torque can be calculated from this expression by means of Eq. (4.11).

$$\begin{aligned}\dot{\boldsymbol{\omega}}_c = &-\mathbf{S}(\boldsymbol{u}_3)\Big(-k_4\boldsymbol{e}_4-\mathbf{R}^T\boldsymbol{r}_{3d}\\ &+k_3\mathbf{S}(\boldsymbol{u}_3)^2\big(\dot{\mathbf{R}}^T\boldsymbol{r}_{3d}+\mathbf{R}^T\dot{\boldsymbol{r}}_{3d}\big)\\ &+\tfrac{1}{mk_1}\mathbf{S}(\boldsymbol{u}_3)^2\big(\dot{T}_d\mathbf{R}^T\boldsymbol{e}_2+T_d\dot{\mathbf{R}}^T\boldsymbol{e}_2+T_d\mathbf{R}^T\dot{\boldsymbol{e}}_2\big)\Big)\\ &+\dot{\mathbf{R}}^T\mathbf{R}_d\boldsymbol{\omega}_d+\mathbf{R}^T\mathbf{R}_d\dot{\boldsymbol{\omega}}_d+\begin{bmatrix}0 & 0 & \dot{\omega}_{3c}(t)\end{bmatrix}^T\end{aligned} \tag{4.10}$$

$$\boldsymbol{\tau} = \mathbf{J}\dot{\boldsymbol{\omega}}_c+\mathbf{S}(\boldsymbol{\omega})\mathbf{J}\boldsymbol{\omega} \tag{4.11}$$

It can be noticed that constants k_{1-4} appear in the calculated control laws. These constants define the dynamics of each backstepping error.

More details about the derivation of these control laws and about their stability proofs are given in [27].

Notice that defining the error vectors this way sets only the third column of $\mathbf{R}_d$. This is evident by checking the definitions of error $\boldsymbol{e}_3$ (in Eq. 4.8) or function $\dot{\omega}_{3c}(t)$, that appears in the expression of the calculated angular velocity (Eq. 4.10). Thus, the direction of the x and y axis can be assigned arbitrarily, as long as the orthogonality of the frame of reference is satisfied. That is to say, a degree of freedom appears for the evolution of *yaw*. Exploiting this degree of freedom, the desired rotation matrix $\mathbf{R}_d$ has been defined as the one that makes the velocity vector of the vehicle tangent to the path, as shown in Eq. (4.12). To assure this control specification, a PD-like controller is designed as a control law for function $\dot{\omega}_{3c}$, as shown in Eq. (4.13), where $\boldsymbol{r}_1$ is the first column of the rotation matrix $\mathbf{R}$ and $\boldsymbol{r}_{2d}$ is the second column of the desired rotation matrix $\mathbf{R}_d$.

$$\mathbf{R}_d = \begin{bmatrix}-\frac{\mathbf{S}(\boldsymbol{r}_{3d})^2\boldsymbol{p}'_d}{\|\mathbf{S}(\boldsymbol{r}_{3d})^2\boldsymbol{p}'_d\|} & \frac{\mathbf{S}(\boldsymbol{r}_{3d})\boldsymbol{p}'_d}{\|\mathbf{S}(\boldsymbol{r}_{3d})\boldsymbol{p}'_d\|} & \boldsymbol{r}_{3d}\end{bmatrix} \tag{4.12}$$

$$\dot{\omega}_{3c} = -l_2\left(\omega_3-\omega_{3d}+l_1\boldsymbol{r}_{2d}^T\boldsymbol{r}_1\right)+\dot{\omega}_{3d}-l_1\tfrac{d}{dt}\left(\boldsymbol{r}_{2d}^T\boldsymbol{r}_1\right) \tag{4.13}$$

Parameter γ is present in the definition of $\boldsymbol{e}_1$, Eq. (4.2), and implicit in the rest of the controller expressions. This parameter defines the desired trajectory position, velocity and acceleration at a certain time instant. Therefore, the controller algorithm needs to have this parameter properly scheduled to work correctly. This is known as the timing law. The timing law stated in this algorithm is given by the second time derivative of γ, defined in Eq. (4.14). Matrix ${}^W_T\mathbf{R}^T(\gamma)$ represents the rotation matrix from a frame $\{T\}$ tangent to the path to the world frame, obtained as shown in Eq. (4.15), where $\xi(\gamma)$ is a function that changes sign in the inflection points of the path. This timing law has been defined to make the time derivative of γ converge to its desired value $\dot{\gamma}_d$, and also to preserve the stability of the system as long as the condition stated in

Eq. (4.16) is satisfied. That condition assures the ascending behaviour of the path. $\dot{\gamma}_d$ is a control specification that can be related to the path's shape. Examples of the definition of this variable are given in Section 4.2.

$$\ddot{\gamma} = -k_\gamma \sigma(\dot{\gamma} - \dot{\gamma}_d) + \boldsymbol{u}_1^T {}_T^W\mathbf{R}^T \left(k_1^2 k_2 \boldsymbol{e}_2 + k_1 \dot{\boldsymbol{\sigma}}(\boldsymbol{e}_1) - \dot{\gamma}\boldsymbol{p}_d''\right) \tag{4.14}$$

$${}_T^W\mathbf{R}^T(\gamma) = \left[\frac{\boldsymbol{p}_d'(\gamma)}{\|\boldsymbol{p}_d'(\gamma)\|} \quad \xi(\gamma)\frac{\mathbf{S}(\boldsymbol{p}_d'(\gamma))^2\boldsymbol{p}_d''(\gamma)}{\|\mathbf{S}(\boldsymbol{p}_d'(\gamma))^2\boldsymbol{p}_d''(\gamma)\|} \quad \xi(\gamma)\frac{\mathbf{S}(\boldsymbol{p}_d'(\gamma))\boldsymbol{p}_d''(\gamma)}{\|\mathbf{S}(\boldsymbol{p}_d'(\gamma))\boldsymbol{p}_d''(\gamma)\|}\right] \tag{4.15}$$

$$|\boldsymbol{p}_d'(\gamma)^T \boldsymbol{u}_3| > \alpha \tag{4.16}$$

Some of the mathematical expressions stated for the *Backstepping* algorithm require derivatives that have to be estimated in order to avoid their arduous analytical calculations. In this chapter, a first order filter was used to obtain a time derivative estimation of a known variable. The scheme of this filter is shown in Fig. 4.1, where x is the known variable, $\hat{x}$ is the filtered variable, $\dot{\hat{x}}$ is the time derivative estimation of x and x_0 is its initial value.

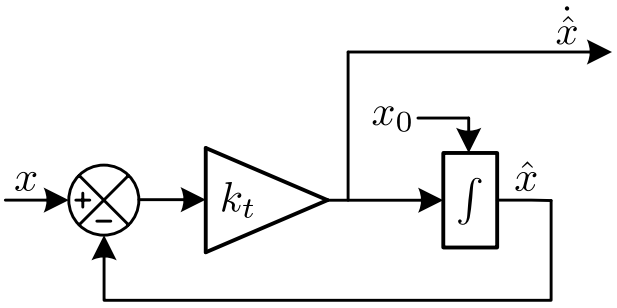

FIGURE 4.1: General derivative estimation diagram

The filter was used to estimate the time derivative of the thrust ($\dot{T}_d$) and the three components of the direction vector of this force ($\dot{\boldsymbol{r}}_{3d}$). With $\dot{\boldsymbol{r}}_{3d}$ it is possible to calculate $\dot{\mathbf{R}}_d$ as shown in Eq. (4.17). The angular speed vector $\boldsymbol{\omega}_d$ can be obtained from its skew symmetric matrix, calculated as shown in Eq. (4.18). The time derivative of the desired angular speed $\dot{\boldsymbol{\omega}}_d$ can be estimated using an estimation filter for each component of vector $\boldsymbol{\omega}_d$.

$$\dot{\mathbf{R}}_d = \frac{\partial \mathbf{R}_d}{\partial \boldsymbol{r}_{3d}}\dot{\boldsymbol{r}}_{3d} + \frac{\partial \mathbf{R}_d}{\partial \boldsymbol{p}_d'}\boldsymbol{p}_d''\dot{\gamma} \tag{4.17}$$

$$\mathbf{S}(\boldsymbol{\omega}_d) = \mathbf{R}_d^T \dot{\mathbf{R}}_d \tag{4.18}$$

The control structure of the *BS* controller is shown in Fig. 4.2. It is an IGC structure, as it is implemented without any inner loop controller. The second derivative of γ, calculated by the timing law, is integrated twice to obtain $\dot{\gamma}$ and γ. The derivative estimation filters have been omitted to clarify the scheme.

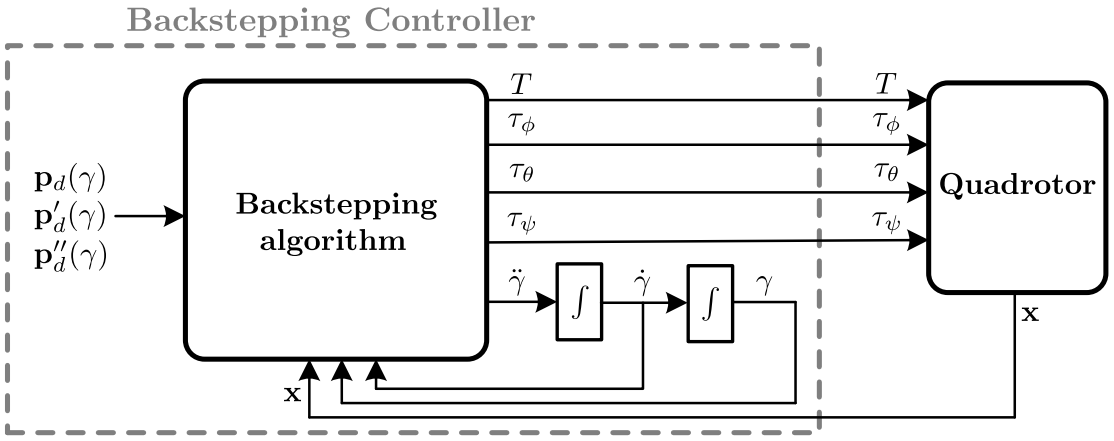

FIGURE 4.2: Backstepping control structure.

The implementation of the *Backstepping* controller is given in Algorithm 1. This algorithm requires the state vector of the system ($\boldsymbol{x}$), the estimated derivative parameters ($\hat{\dot{T}}_d, \hat{\dot{\boldsymbol{r}}}_{3d}, \hat{\dot{\boldsymbol{\omega}}}_d$), the values of γ, its derivative $\dot{\gamma}$ and its desired derivative $\dot{\gamma}_d$, the control design parameters ($k_{1\text{-}4}, p_{max}, l_{1\text{-}2}, k_\gamma$), the desired path ($\boldsymbol{p}_d(\gamma)$) and the first and second partial derivatives of the desired path with respect to γ ($\boldsymbol{p}'_d(\gamma), \boldsymbol{p}''_d(\gamma)$). The algorithm returns the control signals of thrust and torques ($T, \tau_\phi, \tau_\theta$ and τ_ψ). The first step of this algorithm obtains the position ($\boldsymbol{p}$), velocity ($\boldsymbol{v}$) and angular velocity ($\boldsymbol{\omega}$) as well as the rotational matrix ($\boldsymbol{R}$) from the system's states. Next, the set of equations of the algorithm are applied in a specific sequence aiming to obtain the control actions of the thrust and torque forces.

Notice that the components of the y and z axis of each state are made negative to obtain the vectors and rotational matrix and so are these components of the torque vector once the control actions are obtained. This is equivalent to a 180° rotation of the reference frame around the x axis. As explained at the beginning of this section, both models consider different frames of reference. In order to use the equations as stated in [27] this rotation is applied to the states before and after applying the algorithm.

4.1.2 Feedback Linearisation

The *Feedback Linearisation* path following algorithm implemented in this chapter is based on the development found in [3], which is an improvement of the one in [141]. The employed mathematical model is very similar to one defined in Section 3.2. Thus, the same control design process can be used. Nevertheless, note that in [3] the velocities of the vehicle are defined with respect to the world frame. Therefore, Eq. (4.19) is now used to define the evolution of these states, where the W on the superscript indicates that it is referenced to the world frame. And thus, the evolution of the world position is equivalent to the world velocities, as shown in Eq. (4.20). Also, note that in this thesis a *zyx*-sequence is used to define the rotation matrix (section Section 3.2), while [3] uses a *zxy*-sequence. However, the selection of this matrix does not affect the behaviour of the system as long as the matrix is well-defined.

Algorithm 1 Backstepping

Require: $\boldsymbol{x} \coloneqq \{x, y, z, u, v, w, p, q, r, \phi, \theta, \psi\}$,
$\hat{T}_d$, $\hat{\boldsymbol{r}}_{3d}$, $\hat{\boldsymbol{\omega}}_d$, γ, $\dot{\gamma}$, $\dot{\gamma}_d$, $k_{1\text{–}4}$, p_{max}, $l_{1\text{–}2}$, k_γ,
$\boldsymbol{p}_d(\gamma) \coloneqq [x_d(\gamma), y_d(\gamma), z_d(\gamma)]^T$, $\boldsymbol{p}'_d(\gamma)$, $\boldsymbol{p}''_d(\gamma)$

Create the vectors and rotation matrix from the states:
$\boldsymbol{p} = [x, -y, -z]^T$, $\boldsymbol{v} = [u, -v, -w]^T$,
$\boldsymbol{\omega} = [p, -q, -r]$, $\mathbf{R}(\phi, -\theta, -\psi)$

Calculate $\dot{\boldsymbol{p}}_d$
$\dot{\boldsymbol{p}}_d = \boldsymbol{p}'_d\dot{\gamma}$

Calculate $\boldsymbol{e}_1$, $\dot{\boldsymbol{e}}_1$, $\boldsymbol{\sigma}(\boldsymbol{e}_1)$, $\dot{\boldsymbol{\sigma}}(\boldsymbol{e}_1)$ and $\boldsymbol{e}_2$
$\boldsymbol{e}_1 = \boldsymbol{p} - \boldsymbol{p}_d$, $\dot{\boldsymbol{e}}_1 = \mathbf{R}\boldsymbol{v} - \dot{\boldsymbol{p}}_d$
$\boldsymbol{\sigma}(\boldsymbol{e}_1) = p_{max}\frac{\boldsymbol{e}_1}{1+\sqrt{e_{11}^2+e_{12}^2+e_{13}^2}}$
$\dot{\boldsymbol{\sigma}}(\boldsymbol{e}_1) = \frac{\partial\boldsymbol{\sigma}(\boldsymbol{e}_1)}{\partial\boldsymbol{p}}\mathbf{R}\boldsymbol{v} + \frac{\partial\boldsymbol{\sigma}(\boldsymbol{e}_1)}{\partial\boldsymbol{p}_d}\dot{\boldsymbol{p}}_d$
$\boldsymbol{e}_2 = \boldsymbol{\sigma}(\boldsymbol{e}_1) + \frac{1}{k_1}\dot{\boldsymbol{e}}_1$

Calculate ${}^W_T\mathbf{R}^T$, $\ddot{\gamma}$ and $\ddot{\boldsymbol{p}}_d$
${}^W_T\mathbf{R}^T$ *from Eq. (4.15)*
$\ddot{\gamma} = -k_\gamma\sigma(\dot{\gamma} - \dot{\gamma}_d) + \boldsymbol{u}_1^T{}^W_T\mathbf{R}^T\left(k_1^2k_2\boldsymbol{e}_2 + k_1\dot{\boldsymbol{\sigma}}(\boldsymbol{e}_1) - \dot{\gamma}\boldsymbol{p}''_d\right)$
$\ddot{\boldsymbol{p}}_d = \boldsymbol{p}''_d\dot{\gamma}^2 + \boldsymbol{p}'_d\ddot{\gamma}$

Calculate T_d, $\boldsymbol{r}_{3d}$ and T
$T_d = m\left\|k_1^2k_2\boldsymbol{e}_2 + k_1\dot{\boldsymbol{\sigma}}(\boldsymbol{e}_1) + g\boldsymbol{u}_3 - \ddot{\boldsymbol{p}}_d\right\|$
$\boldsymbol{r}_{3d} = \frac{k_1^2k_2\boldsymbol{e}_2+k_1\dot{\boldsymbol{\sigma}}(\boldsymbol{e}_1)+g\boldsymbol{u}_3-\ddot{\boldsymbol{p}}_d}{\left\|k_1^2k_2\boldsymbol{e}_2+k_1\dot{\boldsymbol{\sigma}}(\boldsymbol{e}_1)+g\boldsymbol{u}_3-\ddot{\boldsymbol{p}}_d\right\|}$
$T = \boldsymbol{r}_{3d}^T\boldsymbol{r}_3T_d$

Calculate $\ddot{\boldsymbol{e}}_1$ and $\dot{\boldsymbol{e}}_2$
$\ddot{\boldsymbol{e}}_1 = -\frac{T}{m}\mathbf{R}\boldsymbol{u}_3 + g\boldsymbol{u}_3 - \ddot{\boldsymbol{p}}_d$
$\dot{\boldsymbol{e}}_2 = \dot{\boldsymbol{\sigma}}(\boldsymbol{e}_1) + \frac{1}{k_1}\ddot{\boldsymbol{e}}_1$

Calculate $\mathbf{R}_d$, $\dot{\mathbf{R}}_d$ and $\boldsymbol{\omega}_d$
$\mathbf{R}_d = \begin{bmatrix} -\frac{\mathbf{S}(\boldsymbol{r}_{3d})^2\boldsymbol{p}'_d}{\left\|\mathbf{S}(\boldsymbol{r}_{3d})^2\boldsymbol{p}'_d\right\|} & \frac{\mathbf{S}(\boldsymbol{r}_{3d})\boldsymbol{p}'_d}{\left\|\mathbf{S}(\boldsymbol{r}_{3d})\boldsymbol{p}'_d\right\|} & \boldsymbol{r}_{3d} \end{bmatrix}$
$\dot{\mathbf{R}}_d = \frac{\partial R_d}{\partial\boldsymbol{r}_{3d}}\dot{\boldsymbol{r}}_{3d} + \frac{\partial\mathbf{R}_d}{\partial\boldsymbol{p}'_d}\boldsymbol{p}''_d\dot{\gamma}$
$\mathbf{S}(\boldsymbol{\omega}_d) = \mathbf{R}_d^T\dot{\mathbf{R}}_d \rightarrow \boldsymbol{\omega}_d = [\mathbf{S}(\boldsymbol{\omega}_d)_{23}, \mathbf{S}(\boldsymbol{\omega}_d)_{31}, \mathbf{S}(\boldsymbol{\omega}_d)_{12}]$

Calculate $\dot{\omega}_{3c}$, $\boldsymbol{e}_4$, $\dot{\boldsymbol{\omega}}_c$ and $\boldsymbol{\tau}$
$\dot{\omega}_{3c} = -l_2\left(\omega_3 - \omega_{3d} + l_1\boldsymbol{r}_{2d}^T\boldsymbol{r}_1\right) + \dot{\omega}_{3d} - l_1\frac{d}{dt}\left(\boldsymbol{r}_{2d}^T\boldsymbol{r}_1\right)$
$\boldsymbol{e}_4$ *from Eq. (4.9)*
$\dot{\boldsymbol{\omega}}_c$ *from Eq. (4.10)*
$\boldsymbol{\tau} = \mathbf{J}\dot{\boldsymbol{\omega}}_c + \mathbf{S}(\boldsymbol{\omega})\mathbf{J}\boldsymbol{\omega}$

Obtain real torque inputs
$\tau_\phi = \tau_1$, $\tau_\theta = -\tau_2$, $\tau_\psi = -\tau_3$

return $T, \tau_\phi, \tau_\theta, \tau_\psi$

$$ {}^{w}\dot{\boldsymbol{v}} = \left(\frac{1}{m}\right)\mathbf{R}\,\boldsymbol{F} - \begin{bmatrix} 0 & 0 & g \end{bmatrix}^{T} = \begin{bmatrix} {}^{w}\dot{u} & {}^{w}\dot{v} & {}^{w}\dot{w} \end{bmatrix}^{T} \tag{4.19} $$

$$ \dot{\boldsymbol{p}} = {}^{w}\boldsymbol{v} = \begin{bmatrix} \dot{x} & \dot{y} & \dot{z} \end{bmatrix}^{T} \tag{4.20} $$

Eq. (4.21) defines the system that will be used to apply the *FL* control, where J_1, J_2 and J_3 are inertial terms defined in Eqs. (4.22 - 4.26); R_1, R_2, R_3 and R_4 are auxiliary terms, Eqs. (4.27 - 4.30); and C_x, C_y, C_z are the inverse of the inertia in each coordinate axis, Eq. (4.31).

$$ \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ {}^{w}\dot{u} \\ {}^{w}\dot{v} \\ {}^{w}\dot{w} \\ \dot{p} \\ \dot{q} \\ \dot{r} \\ \dot{\psi} \\ \dot{\theta} \\ \dot{\phi} \end{bmatrix} = \begin{bmatrix} {}^{w}u \\ {}^{w}v \\ {}^{w}w \\ 0 \\ 0 \\ -g \\ q\,r\,J_1 \\ p\,r\,J_2 \\ p\,q\,J_3 \\ p + \tan(\theta)\,R_4 \\ q\,\cos(\phi) + r\,\sin(\phi) \\ \sec(\theta)\,R_4 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ R_1 & 0 & 0 & 0 \\ R_2 & 0 & 0 & 0 \\ R_3 & 0 & 0 & 0 \\ 0 & C_x & 0 & 0 \\ 0 & 0 & C_y & 0 \\ 0 & 0 & 0 & C_z \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} T \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} \tag{4.21} $$

$$ J_1 = \left(\frac{J_{yy} - J_{zz}}{J_{xx}}\right) \tag{4.22} $$

$$ \tag{4.23} $$

$$ J_2 = \left(\frac{J_{zz} - J_{xx}}{J_{yy}}\right) \tag{4.24} $$

$$ \tag{4.25} $$

$$ J_3 = \left(\frac{J_{xx} - J_{yy}}{J_{zz}}\right) \tag{4.26} $$

$$ R_1 = \frac{\sin(\phi)\sin(\psi) + \cos(\phi)\sin(\theta)\cos(\psi)}{m} \tag{4.27} $$

$$ R_2 = \frac{-\sin(\phi)\cos(\psi) + \cos(\phi)\sin(\theta)\sin(\psi)}{m} \tag{4.28} $$

$$ R_3 = \frac{\cos(\phi)\cos(\theta)}{m} \tag{4.29} $$

$$R_4 = \sin(\phi)\, q + \cos(\phi)\, r \tag{4.30}$$

$$C_x = \frac{1}{J_{xx}}, \quad C_y = \frac{1}{J_{yy}}, \quad C_z = \frac{1}{J_{zz}} \tag{4.31}$$

Besides the dynamics of the system, the outputs of interest need to be defined. In this case, a set of four virtual outputs, Eq. (4.32), are stated. This algorithm depends explicitly on the path projection on the xy plane. According to the simulations planed in Section 4.2, this projection must be a circumference.

$$\boldsymbol{h}(\boldsymbol{x}) = \begin{bmatrix} h_1(\boldsymbol{x}) \\ h_2(\boldsymbol{x}) \\ h_3(\boldsymbol{x}) \\ h_4(\boldsymbol{x}) \end{bmatrix} = \begin{bmatrix} x^2 + y^2 - A^2 \\ z - f_z(\boldsymbol{x}) \\ A \arctan\left({}^{y}\!/_{x}\right) \\ \psi \end{bmatrix} \tag{4.32}$$

The virtual output h_1 tracks the position error of the vehicle in the xy plane, where A is the radius of the circumference. h_2 tracks the position error in the z coordinate, where $f_z(\boldsymbol{x})$ is a function that defines the altitude reference. The third virtual output returns a scalar, $h_3 \in [0, 2\pi A]$ in meters, that is the position on the path at a minimum distance in the xy projection. Finally, h_4 is the *yaw* angle.

Defining the outputs this way, the position error can be eliminated by making h_1 and h_2 converge to zero. The point on the path to be tracked by the vehicle can be controlled with h_3, as well as the velocity of this point if the time derivative of h_3 is regulated too. And clearly, the *yaw* angle can be regulated by controlling h_4.

The vector relative degree of the system given by the dynamics in Eq. (4.21) and the outputs in Eq. (4.32) is equal to $\{2, 2, 2, 2\}$. This vector represents, for each output, the number of time derivatives that need to be done to h to get an explicit relation with one of the inputs. However, as demonstrated in [141] and [3], the vector relative degree is not well-defined. This is because the so called decoupling matrix becomes always singular. That means that the linearised system is not fully controllable. Two new states (ζ_1 and ζ_2) are added to the system to solve this problem. These states satisfy Eq. (4.33). The total thrust, T, is now a state of the system and the new input u_1 is equivalent to the second derivative of this state.

$$\begin{aligned} T &= \zeta_1, \quad \dot{\zeta}_1 = \zeta_2, \quad \dot{\zeta}_2 = u_1, \\ \tau_\phi &= u_2, \quad \tau_\theta = u_3, \quad \tau_\psi = u_4 \end{aligned} \tag{4.33}$$

Adding these states to the system, the vector relative degree becomes $\{4, 4, 4, 2\}$, which derives in a well-defined decoupling matrix. Since the total number of states of the system is 14 (12 states of the original system and 2 new states) and the vector relative degree adds up to 14,

it is known that this system is fully linearisable with this set of outputs. Thus, the Full State Feedback Linearisation problem can be solved now for the system defined in eqs. (4.21), (4.32) and (4.33).

To solve the Full State *FL* problem, the vector function $\boldsymbol{\Psi}(\boldsymbol{x}_e)$ that transforms the states of the system to a set of linear states is obtained as

$$\begin{bmatrix} \boldsymbol{\xi}_{1i} \\ \boldsymbol{\xi}_{2i} \\ \boldsymbol{\eta}_{1i} \\ \boldsymbol{\eta}_{2k} \end{bmatrix} = \boldsymbol{\Psi}(\boldsymbol{x}_e) = \begin{bmatrix} L_f^{i-1} h_1(\boldsymbol{x}_e) \\ L_f^{i-1} h_2(\boldsymbol{x}_e) \\ L_f^{i-1} h_3(\boldsymbol{x}_e) \\ L_f^{k-1} h_4(\boldsymbol{x}_e) \end{bmatrix}, \tag{4.34}$$

for $i \in \{1, 2, 3, 4\}$ and $k \in \{1, 2\}$. Where x_e is the set of 14 states of the extended system, $\boldsymbol{\xi}_{1i}$ and $\boldsymbol{\xi}_{2i}$ are the linear states that are required to converge to zero (h_1 and h_2 subsystems), $\boldsymbol{\eta}_{1i}$ and $\boldsymbol{\eta}_{2k}$ are the linear states that have to be regulated to control the position or velocity on the path (h_3 subsystem) and the *yaw* angle (h_4 subsystem). Moreover, $L_f^m h_n(\boldsymbol{x_e})$ refers to the mth Lie derivative of $h_n(\boldsymbol{x_e})$ with respect to f.

It is important to mention that $\boldsymbol{\Psi}(\boldsymbol{x}_e)$ is a diffeomorphism. That is, a derivable function defined in a certain region whose inverse exists and is derivable as well.

Once the transformation of Eq. (4.34) is applied to the extended system, the resulting system is

$$\begin{aligned}
\dot{\xi}_{11} &= \xi_{12} \\
\dot{\xi}_{12} &= \xi_{13} \\
\dot{\xi}_{13} &= \xi_{14} \\
\dot{\xi}_{14} &= \alpha_1(\boldsymbol{\xi}, \boldsymbol{\eta}) + \beta_1(\boldsymbol{\xi}, \boldsymbol{\eta})\, \boldsymbol{u} \\
\dot{\xi}_{21} &= \xi_{22} \\
\dot{\xi}_{22} &= \xi_{23} \\
\dot{\xi}_{23} &= \xi_{24} \\
\dot{\xi}_{24} &= \alpha_2(\boldsymbol{\xi}, \boldsymbol{\eta}) + \beta_2(\boldsymbol{\xi}, \boldsymbol{\eta})\, \boldsymbol{u} \\
\dot{\eta}_{11} &= \eta_{12} \\
\dot{\eta}_{12} &= \eta_{13} \\
\dot{\eta}_{13} &= \eta_{14} \\
\dot{\eta}_{14} &= \alpha_3(\boldsymbol{\xi}, \boldsymbol{\eta}) + \beta_3(\boldsymbol{\xi}, \boldsymbol{\eta})\, \boldsymbol{u} \\
\dot{\eta}_{21} &= \eta_{22} \\
\dot{\eta}_{22} &= \alpha_4(\boldsymbol{\xi}, \boldsymbol{\eta}) + \beta_4(\boldsymbol{\xi}, \boldsymbol{\eta})\, \boldsymbol{u}
\end{aligned} \tag{4.35}$$

where the vector formed by each $\boldsymbol{\alpha}$ function is calculated as shown in Eq. (4.36) and the $\boldsymbol{\beta}$ functions conform the (4x4) decoupling matrix of the system, which is obtained in Eq. (4.37).

$$\boldsymbol{\alpha}(\boldsymbol{\xi},\boldsymbol{\eta}) = \begin{bmatrix} \alpha_1(\boldsymbol{\xi},\boldsymbol{\eta}) \\ \alpha_2(\boldsymbol{\xi},\boldsymbol{\eta}) \\ \alpha_3(\boldsymbol{\xi},\boldsymbol{\eta}) \\ \alpha_4(\boldsymbol{\xi},\boldsymbol{\eta}) \end{bmatrix} = \begin{bmatrix} L_f^4 h_1 \\ L_f^4 h_2 \\ L_f^4 h_3 \\ L_f^2 h_4 \end{bmatrix} \tag{4.36}$$

$$\boldsymbol{\beta}(\boldsymbol{\xi},\boldsymbol{\eta}) = \begin{bmatrix} \beta_1(\boldsymbol{\xi},\boldsymbol{\eta}) \\ \beta_2(\boldsymbol{\xi},\boldsymbol{\eta}) \\ \beta_3(\boldsymbol{\xi},\boldsymbol{\eta}) \\ \beta_4(\boldsymbol{\xi},\boldsymbol{\eta}) \end{bmatrix} = \begin{bmatrix} L_g L_f^3 h_1 \\ L_g L_f^3 h_2 \\ L_g L_f^3 h_3 \\ L_g L_f h_4 \end{bmatrix} \tag{4.37}$$

It can be observed that the resulting system of Eq. (4.35) is composed by four canonical controllable subsystems. Thus, if the inputs of the system (u_{1-4}) are chosen to be

$$\boldsymbol{u} = \begin{bmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \end{bmatrix} = \begin{bmatrix} \beta_1(\boldsymbol{\xi},\boldsymbol{\eta})^{-1}\left[v_1 - \alpha_1(\boldsymbol{\xi},\boldsymbol{\eta})\right] \\ \beta_2(\boldsymbol{\xi},\boldsymbol{\eta})^{-1}\left[v_2 - \alpha_2(\boldsymbol{\xi},\boldsymbol{\eta})\right] \\ \beta_3(\boldsymbol{\xi},\boldsymbol{\eta})^{-1}\left[v_3 - \alpha_3(\boldsymbol{\xi},\boldsymbol{\eta})\right] \\ \beta_4(\boldsymbol{\xi},\boldsymbol{\eta})^{-1}\left[v_4 - \alpha_4(\boldsymbol{\xi},\boldsymbol{\eta})\right] \end{bmatrix}, \tag{4.38}$$

the system results in a pure multiple integrator of r_i-order for each virtual output h_i, where r_i is the relative degree. Then, the virtual input, v_i, is obtained by applying a state feedback control with the respective linearised states as shown in Eq. (4.39). The feedback gains K_{ij} can be calculated using the pole-placement method.

$$\begin{aligned} v_1 &= \xi_{11} K_{11} + \xi_{12} K_{12} + \xi_{13} K_{13} + \xi_{14} K_{14} \\ v_2 &= \xi_{21} K_{21} + \xi_{22} K_{22} + \xi_{23} K_{23} + \xi_{24} K_{24} \\ v_3 &= (\eta_{11} - \eta_{11}^{ref}) K_{31} + (\eta_{12} - \eta_{12}^{ref}) K_{32} \\ &\quad + \eta_{13} K_{33} + \eta_{14} K_{34} \\ v_4 &= (\eta_{21} - \eta_{21}^{ref}) K_{41} + \eta_{22} K_{42} \end{aligned} \tag{4.39}$$

The control structure of the *FL* controller is shown in Fig. 4.3. It is an IGC structure as the *Backstepping* algorithm. As shown, there is a block that transforms the states of the extended system $(\boldsymbol{x}, \zeta_1, \zeta_2)$ into the set of linear states $(\boldsymbol{\xi},\boldsymbol{\eta})$. Then, another block calculates the virtual inputs $(\boldsymbol{v})$ from these linear states. Later, the virtual inputs are transformed into the inputs of the extended system $(\boldsymbol{u})$. And finally, the first input (u_1) is integrated twice to obtain the total thrust applied on the quadrotor (T), which is the actual input of the system. Note that this scheme does not include the path as an external reference, since it must be embedded in the algorithm calculations.

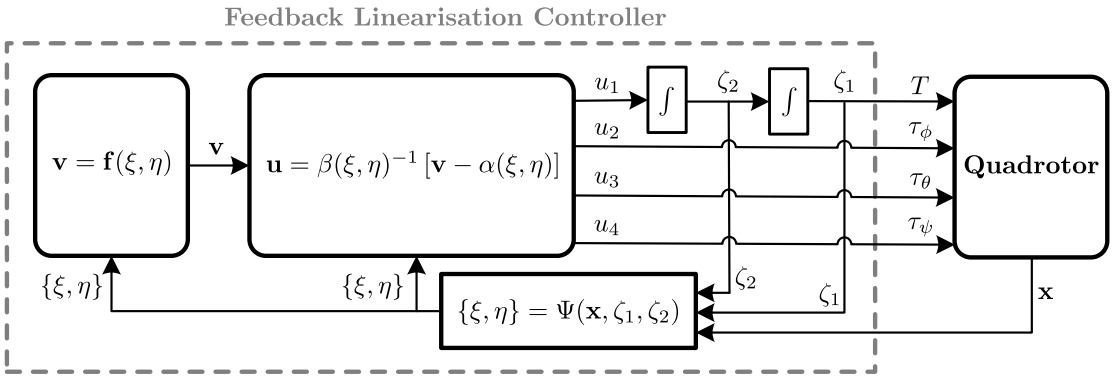

FIGURE 4.3: Feedback Linearization control structure.

The implementation of the *Feedback Linearisation* controller is stated in Algorithm 2. This algorithm requires the full state of the extended system ($\boldsymbol{x}_e$), the radius of the circumference (A), the desired velocity of the vehicle (V_{ref}) and the feedback gains of the linear controller (K_{ij}). It returns the four inputs of the quadrotor. The algorithm starts by an axis transformation of the velocities to the world frame. Later, the outputs of the system are obtained by means of Eq. (4.32). Note that the arctan is implemented using the *atan2* function. Next, the references of the linear states (η_{11}^{ref}, η_{12}^{ref} and η_{21}^{ref}) are calculated. These define the experiment references and are detailed in theSection 4.2. Note that it is ensured that η_{21}^{ref}, the *yaw* reference, satisfies the condition $\mid \eta_{21}^{ref} - \psi \mid \leqslant \pi$, since it is expected that the vehicle moves the smallest angle towards its reference *yaw* angle. Final steps are the calculation of the linear states ($\boldsymbol{\xi}$, $\boldsymbol{\eta}$), $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ (Eqs. 4.34, 4.36 and 4.37), to obtain the virtual inputs (Eq. 4.39), and to calculate the inputs used to control the system (Eq. 4.38). It is important to note that the input of the total thrust of the quadrotor is obtained by double integrating the u_1 input of the extended system.

4.1.3 Three-dimensional NLGL

NLGL, as mentioned in Section 2.2.4, is a geometric PF algorithm based on the Virtual Target Point (VTP) concept. The VTP is a target reference point placed on the desired path, which is updated periodically. The *NLGL* algorithm obtains the VTP as the point on the path at a predefined distance L from the vehicle (Fig. 3.9), where L is a scalar constant parameter that affects the performance of the *NLGL* controller. Once the VTP is obtained, the algorithm calculates the needed *yaw* angle to face the vehicle towards the VTP on the path. Since the vehicle is required to move at a constant predefined speed, it will end up moving to the path.

From the algorithm description it can be seen that it is designed to work only in the two-dimensional space given by the xy plane. For this reason, as the two previous algorithms operate in three dimensions and the objective of this chapter is to compare the algorithms in the same conditions, *NLGL* algorithm presented in [172] has been adapted to operate in 3D as well. To this end, the *3D-NLGL* algorithm has been stated in Algorithm 3.

The Algorithm 3 requires the current position $\boldsymbol{p}$ of the vehicle and its *yaw* angle (ψ), as well as the desired velocity of the vehicle (V_{ref}) and the scalar parameter of the guidance law (L).

Algorithm 2 Feedback Linearization

Require: $vecx_e \coloneqq \{x, y, z, u, v, w, p, q, r, \phi, \theta, \psi, \zeta_1, \zeta_2\}$,
A, V_{ref}, K_{ij}

Transform velocities axis:
1: $[{}^{W}u \ {}^{W}v \ {}^{W}w]^T = \mathbf{R}\,[u \ v \ w]^T$

Calculate outputs:
2: $h_1 = x^2 + y^2 - A^2$
3: $h_2 = Z - f_z(\boldsymbol{x}_e)$
4: $h_3 = A\,\text{atan2}(y, x)$
5: $h_4 = \psi$

Calculate linear state references:
6: $\eta_{11}^{ref} = 0 \ \ (K_{31} = 0)$
7: $\eta_{12}^{ref} = V_{ref}$
8: $\eta_{21}^{ref} = {}^{h_3}\!/_{A} + {}^{\pi}\!/_{2}$
→ ensure that $\left|\eta_{21}^{ref} - \psi\right| \leqslant \pi$

Calculate linear states ($\boldsymbol{\xi}$, $\boldsymbol{\eta}$), $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$
9: $\{\boldsymbol{\xi}_{1i}, \boldsymbol{\xi}_{2i}, \boldsymbol{\eta}_{1i}, \boldsymbol{\eta}_{2k}\} \coloneqq \boldsymbol{\Psi}(\boldsymbol{x}_e)$
10: $\alpha_i = L_f^m h_i$
11: $\beta_i = L_g L_f^{m-1} h_i$
→ $i \in \{1, 2, 3, 4\}, k \in \{1, 2\}$

Calculate virtual inputs
12: $\{v_1, v_2, v_3, v_4\}$ *from Eq. (4.39)*

Calculate inputs of the extended system
13: $u_i = \beta_i^{-1}\,[v_i - \alpha_i]$
→ $i \in \{1, 2, 3, 4\}$

Obtain real inputs
14: $\dot{\zeta}_2 = u_1$, $\dot{\zeta}_1 = \zeta_2$, $T = \zeta_1$,
15: $\tau_\phi = u_2$, $\tau_\theta = u_3$, $\tau_\psi = u_4$

return $T, \tau_\phi, \tau_\theta, \tau_\psi$

It returns the commands for the velocities in the x and y axis, the altitude and the *yaw* angle of the quadrotor. In this algorithm, γ_{prev} represents the value of $\gamma_{d_{min}}$ saved from the previous execution of the algorithm, where $\gamma_{d_{min}}$ is the point on the path, given by the value of γ, that is at a minimum distance to the location of the vehicle. The value of γ_{prev} is initialized to 0 in the first execution of this algorithm.

As seen in the algorithm, the first step after the initialization of the variables is to calculate the point on the path that is at a minimum distance from the vehicle, given by $\gamma_{d_{min}}$, and the value of this distance. Then, if the distance to $\gamma_{d_{min}}$ is larger than L, the VTP is defined as $\gamma_{d_{min}}$, making the vehicle move directly towards the path. When the vehicle is closer to the path, the algorithm will find the first point on the path, starting from $\gamma_{d_{min}}$, that is at an L distance from the vehicle, and will set it as the VTP. The fact that it starts to search the VTP from the point

Algorithm 3 Three-dimensional NLGL

Require: $\boldsymbol{p} : (x, y, z)$, ψ, L, V_{ref},
$\boldsymbol{p}_d(\gamma) \coloneqq [x_d(\gamma), y_d(\gamma), z_d(\gamma)]^T$, γ_f

Initialize (First execution):
1: $\gamma_{prev} = 0$

Calculate $d_{min}, \gamma_{d_{min}}$:
2: $d_{min} \coloneqq \min_{\gamma} (\|\boldsymbol{p} - \boldsymbol{p}_d(\gamma)\|) \mid \gamma \in [\gamma_{prev}; \gamma_f]$
3: $\gamma_{d_{min}} \coloneqq \operatorname{argmin}_{\gamma} (\|\boldsymbol{p} - \boldsymbol{p}_d(\gamma)\|) \mid \gamma \in [\gamma_{prev}; \gamma_f]$

Calculate γ_{VTP}:
4: **if** $d_{min} > L$ **then**
5: $\quad \gamma_{VTP} \coloneqq \gamma_{d_{min}}$
6: **else**
7: $\quad \gamma_{VTP} \coloneqq \operatorname{argmin}_{\gamma} (|\|\boldsymbol{p} - \boldsymbol{p}_d(\gamma)\| - L|) \mid \gamma \in [\gamma_{d_{min}}; \gamma_f]$

Calculate commands:
8: $\psi_{cmd} \coloneqq \text{atan2}(y_d(\gamma_{VTP}) - y, x_d(\gamma_{VTP}) - x)$
$\rightarrow$ ensure that $|\psi_{cmd} - \psi| \leqslant \pi$
9: $z_{cmd} \coloneqq z_d(\gamma_{d_{min}})$
10: $v_{cmd} \coloneqq 0$
11: $u_{cmd} \coloneqq \frac{V_{ref} d_{VTP,xy}}{L}$

return $\psi_{cmd}, z_{cmd}, v_{cmd}, u_{cmd}$

at a minimum distance ($\gamma_{d_{min}}$) going forwards, ensures that the vehicle moves always forward on the path.

The next step of this algorithm is to calculate the commands, which are the outputs of the controller. This is important to ensure the correct adaptation of the algorithm from 2D to 3D. First of all, similarly to the *2D-NLGL* algorithm, the *yaw* command angle is calculated as the angle that would face the vehicle towards the VTP, projected in the xy plane of the vehicle. Then, as in the *Feedback Linearisation* controller, it is ensured that ψ_{cmd} satisfies the condition $|\psi_{cmd} - \psi| \leqslant \pi$. The altitude command z_{cmd} is obtained as the desired Z in the point of minimum distance $\gamma_{d_{min}}$.

As in the two-dimensional algorithm, the velocity command v_{cmd} in the y body coordinate is set to zero, as no movement is required on this coordinate. In the *2D-NLGL* algorithm the velocity command u_{cmd} in the x body coordinate has a constant value (V_{ref}). Considering the vertical distance to the VTP, in this 3D algorithm the value of u_{cmd} is set proportional to $d_{VTP,xy}$, that is the projection on the xy plane of the distance L to the VTP. Line 11 of the algorithm calculates this command. It can be seen that when the VTP is on the xy plane $u_{cmd} = V_{ref}$.

The outputs of this controller are the u_{cmd} and v_{cmd} velocity command, the *yaw* command (ψ_{cmd}) and the altitude command (z_{cmd}). That is, this algorithm uses a SGC structure (Fig. 3.7) for solving the path following problem. Thus, the autopilot of Section 3.3 is implemented. It is important to mention that, with an SGC structure, the path following controller's behaviour

Algorithm 4 Three-dimensional Carrot-Chasing

Require: $\boldsymbol{p} : (x, y, z)$, ψ, δ, V_{ref},
$\boldsymbol{p}_d(\gamma) := [x_d(\gamma), y_d(\gamma), z_d(\gamma)]^T$, γ_f

Initialize (First execution):

$\gamma_{prev} := 0$

Calculate $d_{min}, \gamma_{d_{min}}$:

$d_{min} := \min_{\gamma} \left(\|\boldsymbol{p} - \boldsymbol{p}_d(\gamma)\| \right) \mid \gamma \in [\gamma_{prev}; \gamma_f]$

$\gamma_{d_{min}} := \operatorname{argmin}_{\gamma} \left(\|\boldsymbol{p} - \boldsymbol{p}_d(\gamma)\| \right) \mid \gamma \in [\gamma_{prev}; \gamma_f]$

Calculate γ_{VTP}:

$\gamma_{VTP} := \operatorname{argmin}_{\gamma} \left(\left| \left(\int_{\gamma_{d_{min}}}^{\gamma} \|\boldsymbol{p}'_d(\gamma)\| \, d\gamma \right) - L \right| \right) \mid \gamma \in [\gamma_{d_{min}}; \gamma_f]$

Calculate commands:

$\psi_{cmd} := \text{atan2}(y_d(\gamma_{VTP}) - y, \, x_d(\gamma_{VTP}) - x)$
$\rightarrow$ ensure that $|\psi_{cmd} - \psi| \leqslant \pi$

$z_{cmd} := z_d(\gamma_{d_{min}})$

$v_{cmd} := 0$

$u_{cmd} := \frac{V_{ref} d_{VTP,xy}}{d_{VTP}}$

return $\psi_{cmd}, z_{cmd}, v_{cmd}, u_{cmd}$

will always be affected by the performance of the inner-loop controller. Other techniques, such as *LQR*, could be applied to design the autopilot controller with different results. However, the *PID* controllers, which do not require the full state estimation, have been chosen by their simplicity which is aligned with the simplicity of the geometric algorithms. This allows to make a comparison of the control algorithms (*Backstepping* and *Feedback linearisation*) with the simplest PF algorithms.

4.1.4 Three-dimensional Carrot-Chasing

Carrot-Chasing is also a geometric algorithm based on the VTP concept. In the *CC* approach, the VTP is called carrot and is obtained as the point at constant length δ along the path from the point at a minimum distance to the vehicle (Fig. 3.10). That is, δ is computed along the path arc length, and therefore it is not equivalent to the Euclidean distance computed in the *NLGL* algorithm. The vehicle, which is required to move at a constant speed, is faced to the VTP by changing its *yaw* angle, and thus it is said to chase the carrot.

Again, it can be noted that the algorithm is designed to operate in the two-dimensional space. Thus, in this chapter it has been modified and adapted to work in the three-dimensional space. The resulting developed 3D approach is found in Algorithm 4. This algorithm is very similar to the *3D-NLGL* algorithm, since the structure and the operations to calculate the commands are identical. However, in this case the VTP is obtained as the point on the path that is at a δ length from the $\gamma_{d_{min}}$ point, where this length is computed along the path arc. Another difference between *CC* and *NLGL* algorithms is that in *CC* the VTP is always calculated the same way, regardless of the distance from the UAV to the path.

The control structure used to apply the *3D-CC* is the same as the *3D-NLGL*, represented in the Fig. 3.7. Thus, the same autopilot has been used.

4.2 Algorithm Comparison Based on Simulation Results

In this section a set of simulation results comparing the four algorithms described previously is presented. The simulations have been performed on the simulation model presented in Chap. (3). Thus, the dynamics of the motors, the control mixing and aerodynamic effects, such as gyroscopic effects and drag forces, are included. Noise on the measurements and wind disturbances are not included unless otherwise specified. The desired path is a helix defined by

$$\boldsymbol{p}_d(\gamma) = \begin{bmatrix} A\cos(\gamma) \\ A\sin(\gamma) \\ \gamma + 3 \end{bmatrix} \tag{4.40}$$

where A is the radius of the helix, which is 3 meters. Note that with this definition the path starts at 3 meters of altitude. Additionally, the vehicle is required to travel at a constant velocity V_{ref} on the path and with a *yaw* angle tangent to the path.

To meet the velocity specification with the *Backstepping* controller, the desired evolution of γ has been set as defined in Eq. (4.41). The *yaw* requirement is assured by defining the rotational matrix as in Eq. (4.12) and including the PD-like controller for the angular acceleration on the z axis as defined in Eq. (4.13).

$$\dot{\gamma}_d = \frac{V_{ref}}{A} \tag{4.41}$$

Assuming that the vehicle's orientation is tangent to the path, γ can be obtained subtracting $\pi/2$ to the *yaw* command (ψ_{cmd}). Thus, function $f_z(\boldsymbol{x})$ (Eq. 4.32 of *Feedback Linearisation* algorithm) becomes

$$f_z(\boldsymbol{x}) = \gamma + 3 = \psi_{cmd} - \frac{\pi}{2} + 3 \tag{4.42}$$

In the *FL* controller the desired velocity requirement is met by making the reference of the first derivative of h_3 (η_{12}^{ref}) equal to V_{ref}. The feedback gain K_{31}, which regulates the position of the vehicle along the path, is set to zero. Hence, h_3 is only in charge of controlling the velocity of the vehicle along the path. On the other hand, to follow the *yaw* specification, the reference for h_4 is calculated as the path tangent angle on the closest point of the path to the vehicle, as seen on line 8 of the Algorithm 2.

For the *3D-NLGL* and *3D-CC* algorithms, the velocity and *yaw* specifications are accomplished by construction.

The control parameters for each of the four algorithms have been tuned as those that achieve the least mean absolute error (MAE) in terms of distance to the path error. This MAE has

been evaluated in the simulation platform. These parameters are presented in Table 4.1. For the geometric algorithms, as they only have one parameter to tune, a parameter sweep was performed in order to find the value which procures the minimum distance MAE for the defined path. It is important to remark that those parameters depend on the velocity of the vehicle and the path shape. Regarding the FL algorithm, the constants that control each of the four linear subsystems (xy-position, altitude, path velocity and yaw) were tuned separately minimizing the error of each variable. It was done by means of an iterative redesign through the pole placement technique, assuring stability on each iteration. Once the constants of each subsystem were obtained, it was verified that the complete system still behaved with minimum path following error and that it was stable. In the BS algorithm, the stability of the controlled system is very sensitive to changes in the control parameters (k_{1-4}). As they were difficult to tune, they were empirically set to make the system stable. The rest of the parameters of this algorithm were obtained by means of a parameter sweep search.

TABLE 4.1: Control parameters for each algorithm.

Algorithm	**Parameters**
Backstepping	$k_1 = 2, k_2 = 1, k_3 = 50, k_4 = 2, k_\gamma = 50,$ $l_1 = 10, l_2 = 2, p_{max} = 1, k_f = 10$
Feedback Linearization	$k_{11} = -18.75, k_{12} = -40, k_{13} = -32.25, k_{14} = -10,$ $k_{21} = -1\,250, k_{22} = -1\,650, k_{23} = -435, k_{24} = -36,$ $k_{31} = 0, k_{32} = -104, k_{33} = -124, k_{34} = -21,$ $k_{41} = -29, k_{42} = -10$
3D-NLGL	$L = 0.59$
3D Carrot-Chasing	$\delta = 0.59$

Three types of results are reported in this section. First, on the steady state regime, when the vehicle has converged to the path. Next, on the transient regime, changing the initial state conditions. Last, with time-varying wind disturbances.

4.2.1 Steady State Regime

This section shows results about the steady state regime of the vehicle's response with each controller. Steady state refers to the controller's error being constant. The performance of the four algorithms for one full lap of the helix is compared in Table 4.2. The columns relate to: the time to travel one lap, the mean of the distance to the path ($\overline{d}$), the mean absolute yaw error, the mean velocity of the quadrotor and the computational effort of the algorithm. The path distance is the minimum distance between the path and the vehicle, the mean absolute yaw error is calculated from the error between the vehicle's yaw angle and the path tangent angle and the computational effort ($Comp_{eff}$) is a dimensionless parameter that represents the normalized computation time of each algorithm.

The computational effort was calculated as follows: A discrete time simulation with a time step of 1ms was performed for each of the four algorithms and the execution time of each block of the simulink model was monitored. The total runtime dedicated to execute the calculations of the control ($t_{control}$) part were divided by the total runtime needed for the calculations of the dynamics of the system (t_{model}) for each algorithm (Eq. 4.43). Assuming that the time dedicated to compute the dynamics of the quadrotor is similar on each execution, the obtained quotient is considered a representative value to evaluate the computational effort. Finally, result was normalized, setting to 1 the lowest one which corresponds to the *NLGL* algorithm. The implementation of the algorithms was not optimized in terms of computation time. Nevertheless, none of them includes especially intensive calculations (such as solving optimization problems or matrix operations). Thus, they are subject to slight changes on the computation effort term.

$$Comp_{eff} = \frac{t_{control}}{t_{model}} \tag{4.43}$$

TABLE 4.2: Results for one lap in steady state regime.

	time (s)	$\overline{\mathbf{d}}$ **(m)**	$\overline{\lvert \mathbf{e}_\psi \rvert}$ **(deg)**	$\overline{\lVert \mathbf{v} \rVert}$ ($^m/_s$)	$\mathbf{Comp_{eff}}$
Backstepping	44.6306	0.0016	1.6094	0.4451	57.3895
Feedback Linearization	40.7400	0.0078	3.0522	0.4874	1.4877
3D-NLGL	40.3800	0.0198	2.6953	0.4897	1
3D-Carrot-Chasing	40.3800	0.0194	2.6949	0.4897	1.0147

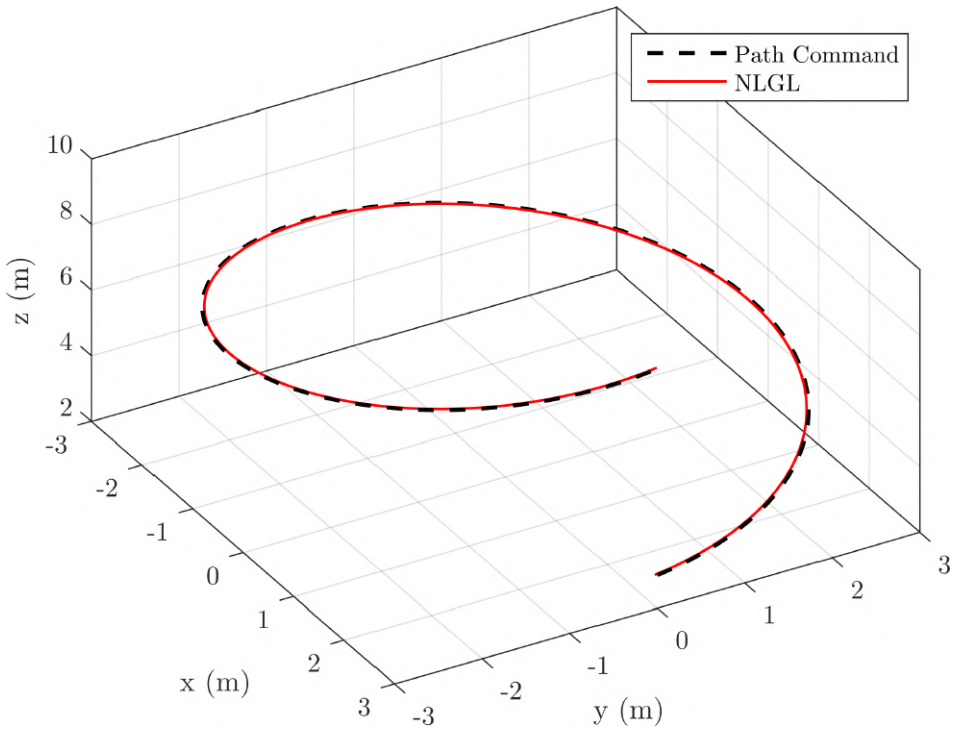

FIGURE 4.4: 3D trajectory for one lap of the helix in steady state regime: *3D-NLGL* algorithm.

The three-dimensional trajectory for one lap of the helix in steady state regime using the *3D-NLGL* algorithm is shown in Fig. 4.4. The performance of this algorithm and the *3D-CC* algorithm is poor compared to the control-oriented algorithms where the vehicle stabilized closer to the path. Nevertheless, it can still be considered that both algorithms present an accurate behaviour as evidenced by Fig. 4.4.

4.2.2 Transient Regime

In these simulations the initial state conditions are modified to observe the transient behaviour of the algorithms. In particular, the effects of changing the initial x-coordinate of the vehicle and the effects of varying its initial *yaw* are analyzed.

In the simulations where the initial x-coordinate position changes, the quadrotor starts on the position given by $x = \{0.5k \mid k = 1..12\}$, $y = 0$ and $z = 3$. The initial *yaw* angle is 90 degrees in the case of the *BS* and *FL* controllers whereas in the geometric algorithms the vehicle faces the VTP. This orientations result in a minimum initial *yaw* error. The initial vehicle's linear velocities, angular velocities, accelerations, roll and pitch angles are zero.

Fig. 4.5 shows four performance indicator graphs, for each algorithm and for each initial position. First, the time the quadrotor takes to converge to the path. Convergence condition is defined as the vehicle remaining at path distance less than 10 times the stable state regime error. Second plot shows the accumulated distance to the path during this convergence period. Note that the periods are different for each algorithm, since their stabilization time are different. The third

graph, shows a parameter for the control effort, i.e. the mean of the absolute value of the control action derivative. Note that the control action of motors (u_{mi}) is given in percentage. Finally, the convergence position on the path given by γ is represented, where the magnitude of γ is given in number of radius along the path (A). The mathematical definition of these indicators is stated in Eqs. (4.44)-(4.47), where t_{conv} is the stabilization time, d is the distance to the path, d_{ss} is the distance error on the steady state regime, d_{int} is the integral of the path distance on the convergence time, c_{eff} is the control effort term, u_{mi} is the control action of the ith motor and γ_{conv} is the convergence position given by the virtual arc parameter.

$$t_{conv} : d < 10d_{ss} \ \forall \ t > t_{conv} \tag{4.44}$$

$$d_{int} = \int_{t_{init}}^{t_{conv}} d(t)dt \tag{4.45}$$

$$c_{eff} = \frac{\left(\overline{\left|\frac{du_{m1}}{dt}\right|} + \overline{\left|\frac{du_{m2}}{dt}\right|} + \overline{\left|\frac{du_{m3}}{dt}\right|} + \overline{\left|\frac{du_{m4}}{dt}\right|}\right)}{4} \tag{4.46}$$

$$\gamma_{conv} = \gamma(t_{conv}) \tag{4.47}$$

Fig. 4.6 and Fig. 4.7 show the three-dimensional trajectory when the initial x-coordinate is 0.5m and 6m, respectively, comparing the response of the four algorithms for one lap. In both figures, circles indicate the convergence point of each algorithm. A solid line represents the convergence trail, and a dotted line where the vehicle has converged.

The simulation results changing the initial *yaw* angle are shown in Fig. 4.8. In these simulations the vehicle starts at the initial point of the path ($x = 3, y = 0, z = 3$) and the *yaw* orientation varies from 0 to 180 degrees in steps of 22.5° in each simulation. The rest of the initial state conditions are identical to the previous simulations. Furthermore, the plots represented in Fig. 4.8 are equivalent to the ones found in Fig. 4.5. Since the convergence criterion only takes into account the distance to the path, in some cases it is considered that the quadrotor has converged to the path while there is still significant *yaw* error.

Once more, the three-dimensional trajectory plots comparing the behaviour of each algorithm are presented in figures 4.9 and 4.10, for the extreme cases of the initial *yaw* angle. The convergence point, represented with a circle, divide the convergence trail (solid line) and the rest of the trajectory (dotted line).

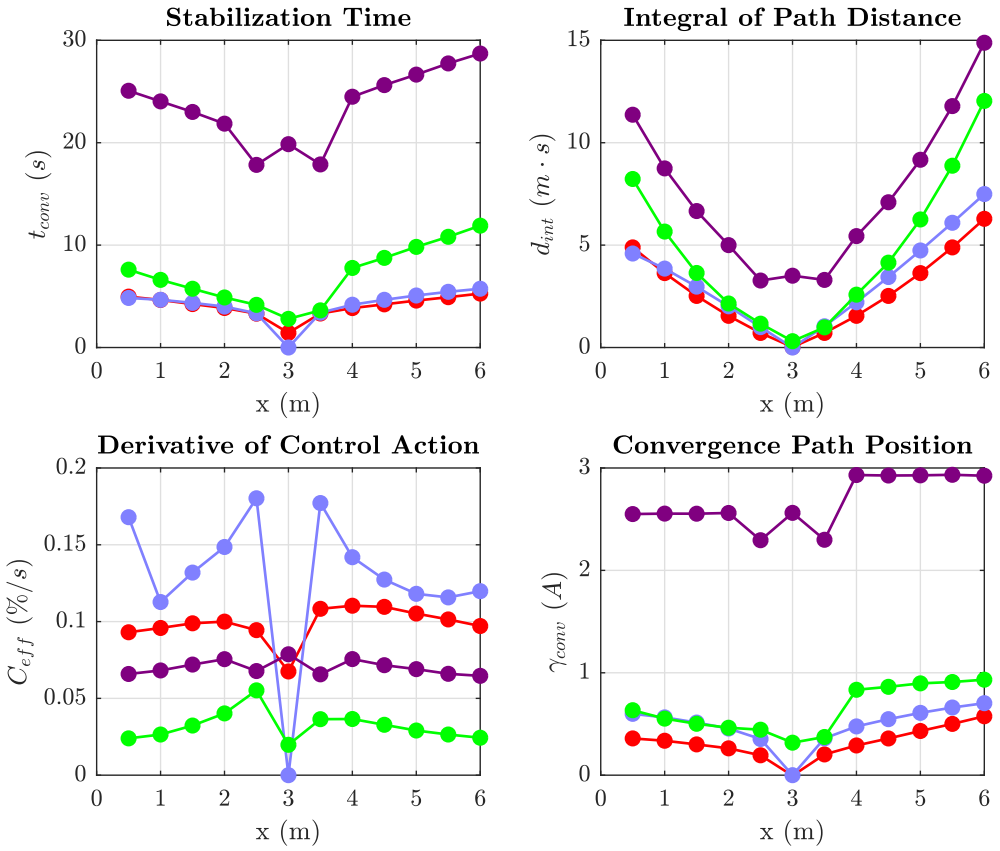

FIGURE 4.5: Simulation results varying the initial x position from 0.5 to 6 meters: *Backstepping* (red), *Feedback Linearisation* (blue), *3D-NLGL* (purple) and *3D Carrot-Chasing* (green).

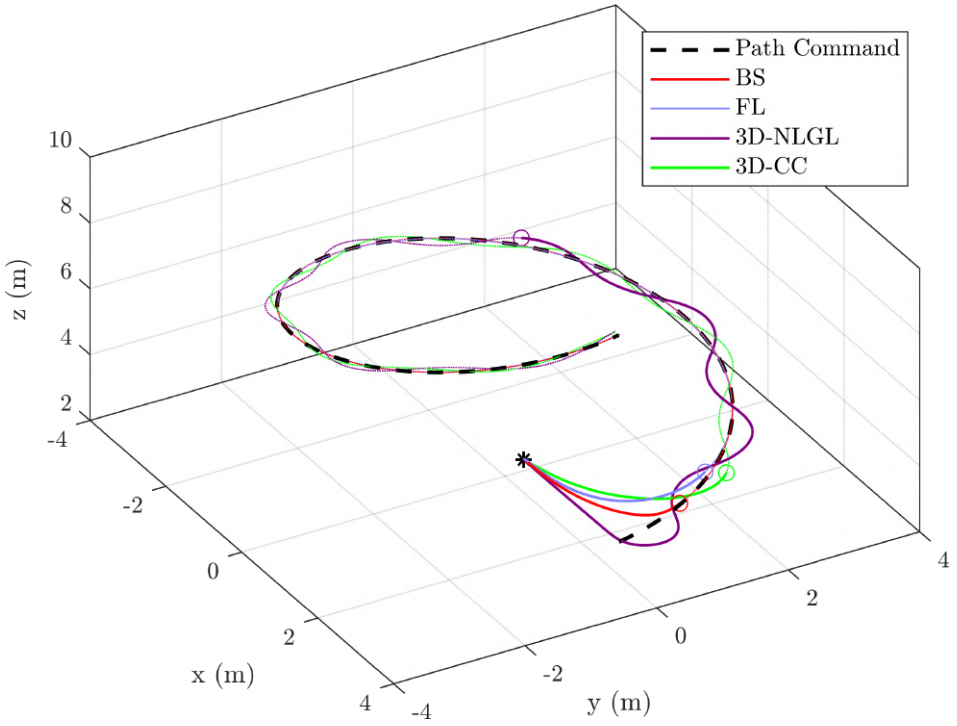

FIGURE 4.6: 3D trajectory comparing the four algorithms for one lap of the helix. Initial position: $x = 0.5, y = 0, z = 3$.

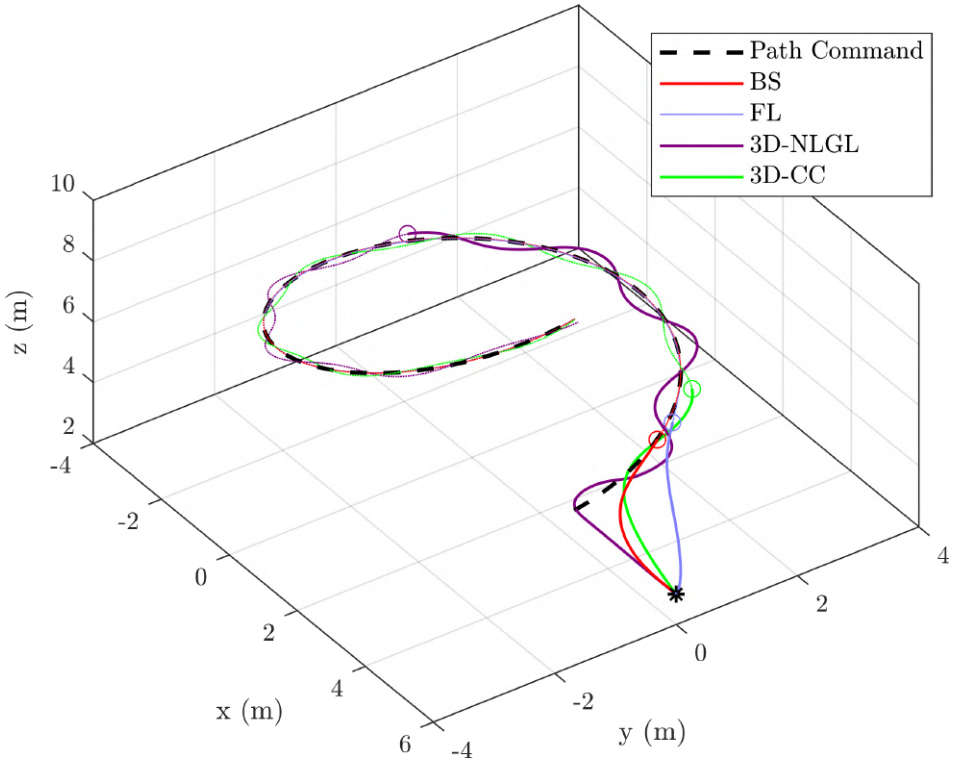

FIGURE 4.7: 3D trajectory evolution comparing the four algorithms for one lap of the helix. Initial position: $x = 6, y = 0, z = 3$.

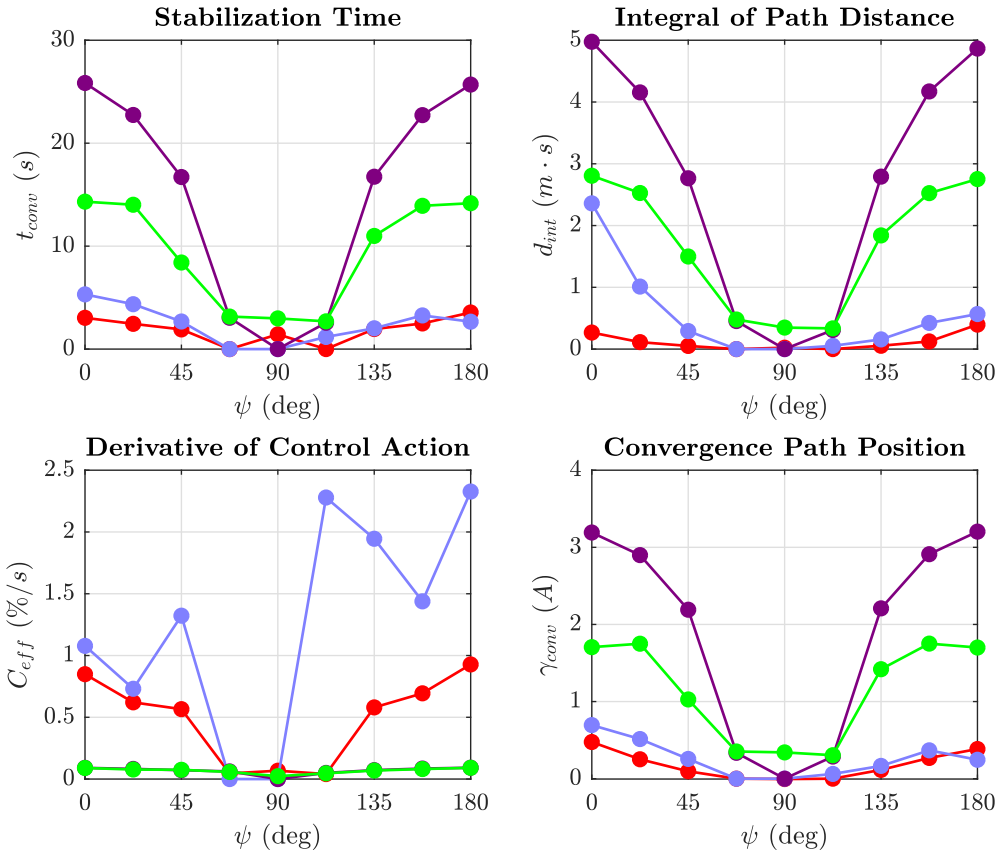

FIGURE 4.8: Simulation results varying the initial *yaw* angle from 0 to 180 degrees: *Backstepping* (red), *Feedback Linearisation* (blue), *3D-NLGL* (purple) and *3D Carrot-Chasing* (green).

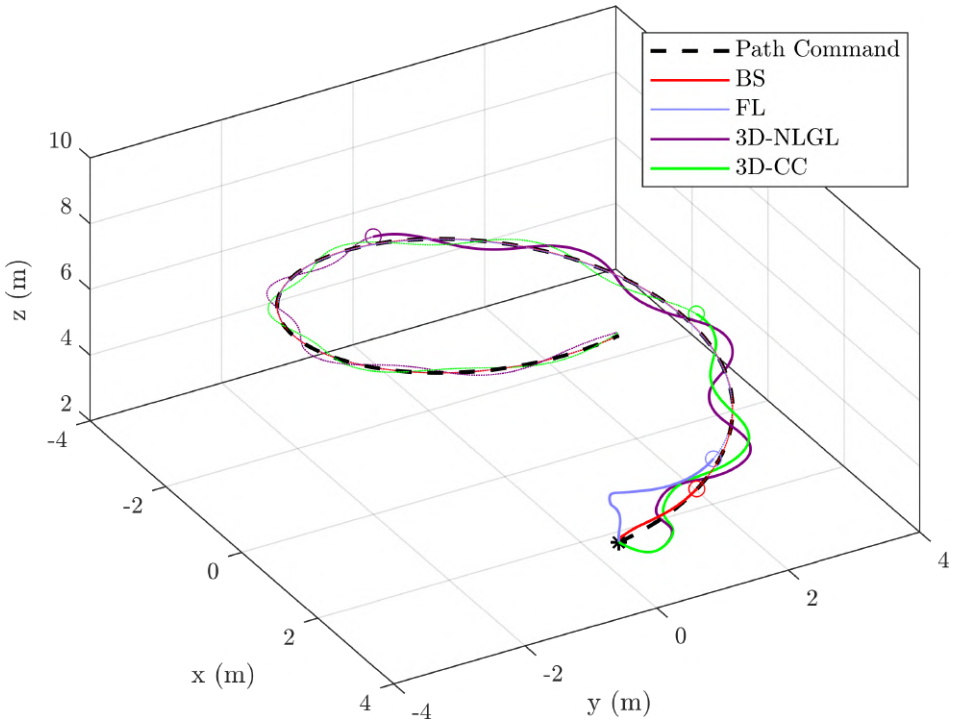

FIGURE 4.9: 3D trajectory evolution comparing the four algorithms for one lap of the helix. Initial position: $x = 3, y = 0, z = 3$. Initial yaw $= 0°$.

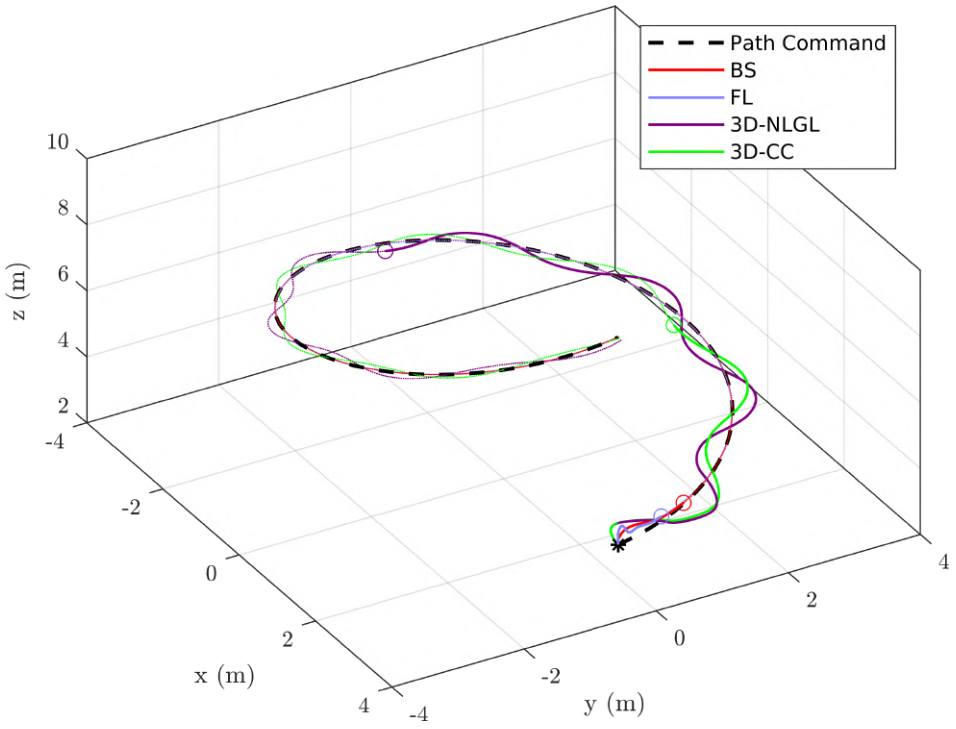

FIGURE 4.10: 3D trajectory evolution comparing the four algorithms for one lap of the helix. Initial position: $x = 3, y = 0, z = 3$. Initial yaw $= 180°$.

4.2.3 Wind Disturbance

In the real world, UAVs deal with diverse environmental influences, such as wind disturbances. Thus, it is of utmost importance that path following algorithms show robustness to these external disturbances. The performance of the implemented algorithms in the presence of time-varying wind is analysed in this section.

To carry out these simulations a wind force has been created. Both its magnitude and direction changes randomly, ranging from 0 to 1 Newton and from $\pi/8$ to $3\pi/8$ radians (i.e. north-east direction), respectively. The performance of each of the four algorithms following the helix path with this wind disturbance force is shown in Fig. 4.11. In these simulations the vehicle starts from the initial point of the path at hover conditions (i.e. zero linear and angular velocities), and it is requested to follow the path for 100 seconds.

The first and second plot of Fig. 4.11 show the wind magnitude and direction, respectively, changing every 10 seconds. The third plot shows the evolution of the path distance error for each algorithm during the 100 seconds experiment.

The results of the simulation experiment of Fig. 4.11 are summarized in Table 4.3: the average path distance error, the average *yaw* error, the average velocity and the path travelled (given by the virtual arc γ), for each algorithm.

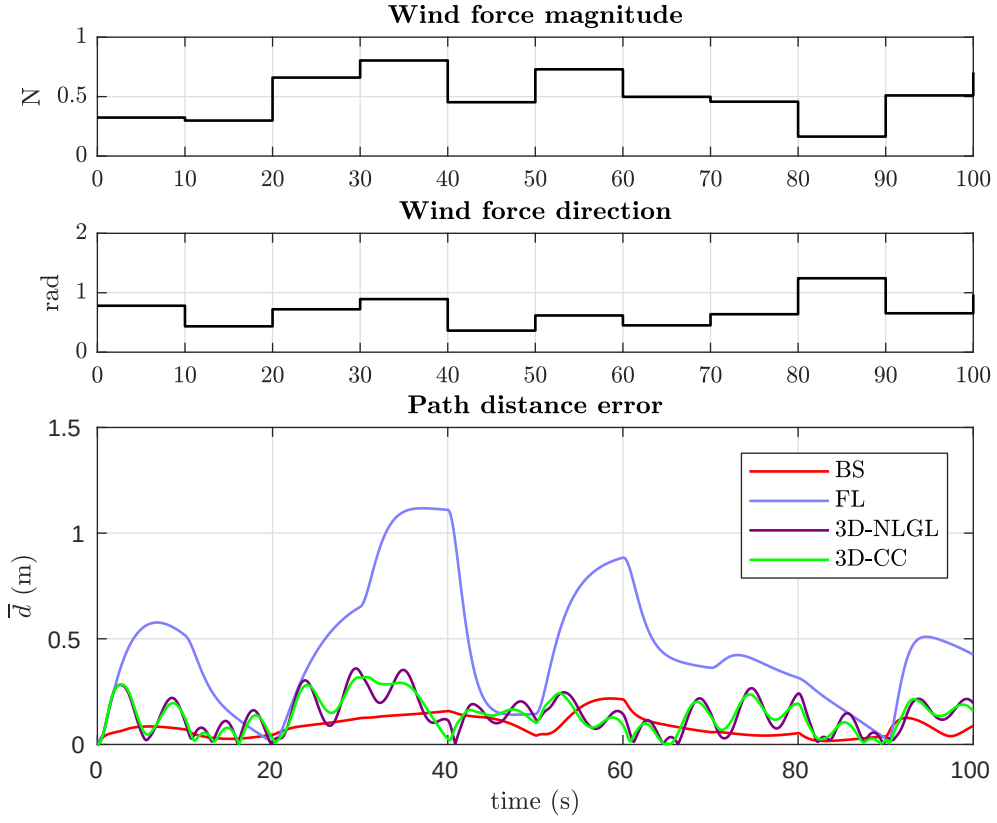

FIGURE 4.11: Path distance error against wind disturbances.

TABLE 4.3: Results for 100 seconds in time-varying wind conditions.

	$\overline{\mathbf{d}}$ (m)	$\overline{\lvert \mathbf{e}_\psi \rvert}$ (deg)	$\overline{\lVert \mathbf{v} \rVert}$ ($^m/_s$)	Path traveled (γ)
Backstepping	0.0859	2.5096	0.3935	12.3731
Feedback Linearisation	0.4491	0.8709	0.1761	2.7191
3D-NLGL	0.1396	14.9657	0.4381	13.6861
3D Carrot-Chasing	0.1377	14.0547	0.4385	13.8209

4.2.4 Realistic Flight Conditions

In this section, *Backstepping* and *Carrot-Chasing* algorithms are tested on a simulation set under realistic flight conditions. That is, with a different and more complex path, with higher velocity references, including noise in the measured states and with a more realistic wind disturbance. These two algorithms have been chosen as they are the control-oriented algorithm and the geometric algorithm that provide the best performance, as seen in previous results.

The desired path in this simulations is a 3D Lemniscate (eight-shaped path) defined by

$$\mathbf{p}_d(\gamma) = \begin{bmatrix} 2A\cos(\gamma) \\ A\sin(2\gamma) \\ \gamma + 3 \end{bmatrix} \tag{4.48}$$

where A is the amplitude of the path, and takes a value of 5 meters. As before, it starts at a height of 3 meters and constantly increases the altitude. Again, the vehicle is required to follow the path at a speed of V_{ref} and with a *yaw* angle tangent to the path.

To fulfil the velocity requirement in the *BS* algorithm, the desired derivative of γ is defined as the velocity reference divided by the module of the partial derivative of the path with respect to the scalar parameter ($\partial \mathbf{p}_d(\gamma)/\partial\gamma$), as shown in Eq. (4.49).

$$\dot{\gamma}_d = \frac{V_{ref}}{\sqrt{(2A\sin(\gamma))^2 + (2A\cos(2\gamma))^2 + 1^2}} \tag{4.49}$$

A noise signal has been added on the measured states to assess the robustness of the controllers. The measured states are the position and the orientation angles (typical measured outputs on an indoor vision-based platform). The linear and angular velocities are assumed to be estimated states, and thus, the noise is filtered. In the present section, the noise signal of the position has a variance of 3cm while the variance on the angles measurement noise is of 2 degrees. In both cases, the signals are sampled at 100Hz.

Regarding the wind disturbance, instead of having a stair-like wind as in section 4.2.3, here the wind magnitude and its direction is generated by integrating a white noise signal. That is, the

wind speed is obtained by integrating a white noise of 10Hz and 0.2 $^m/_s$ of amplitude, and adding it to a constant value of 3 $^m/_s$. And its direction is generated by integrating a white noise signal of the same frequency and 0.0025 *rad* magnitude added to a $^\pi/_4$ constant angle (i.e. north-east direction). Fig. 4.12 shows the obtained wind speed and direction during 40 seconds. Note that the wind is computed as a wind velocity, and not as a force as in section 4.2.3. Therefore, the disturbance force is obtained by the vehicle's equations given a wind profile.

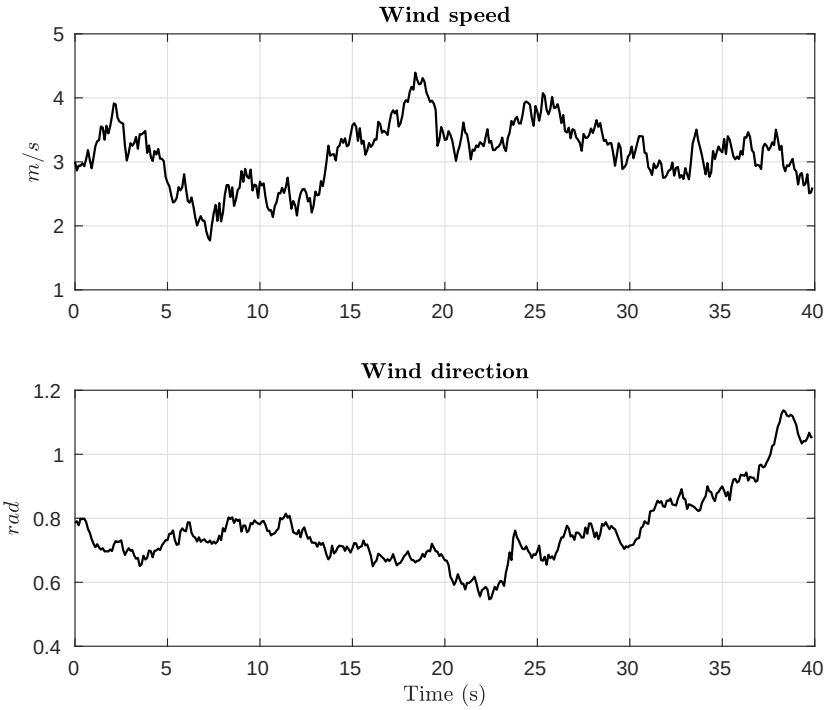

FIGURE 4.12: Realistic wind disturbances.

The three-dimensional evolution of the vehicle following the Lemniscate path, with the *Backstepping* and with the *3D-Carrot-Chasing* algorithms, is shown in Fig. 4.13. The vehicle starts at hover conditions on the initial point of the path and it is required to perform a lap at a velocity of 2 $^m/_s$ under the noise and wind conditions explained above. The results comparing both algorithms are shown in Table 4.4. That is, the total time necessary to accomplish a lap, the average path distance error, the mean absolute *yaw* error and the average velocity of the vehicle.

TABLE 4.4: Results for one lap under realistic flight conditions (2 $^m/_s$).

	time (s)	$\overline{\mathbf{d}}$ **(m)**	$\overline{\lvert\mathbf{e}_\psi\rvert}$ **(deg)**	$\overline{\lVert\mathbf{v}\rVert}$ ($^m/_s$)
Backstepping	35.93	0.2081	3.7344	1.6997
3D-Carrot-Chasing	39.88	0.4437	10.5275	1.6310

To end up, the same aspects shown in Table 4.4 are evaluated for both algorithms and for different path velocity references in Fig. 4.14. The velocity references range from 1 to 4 $^m/_s$, in 0.5 $^m/_s$

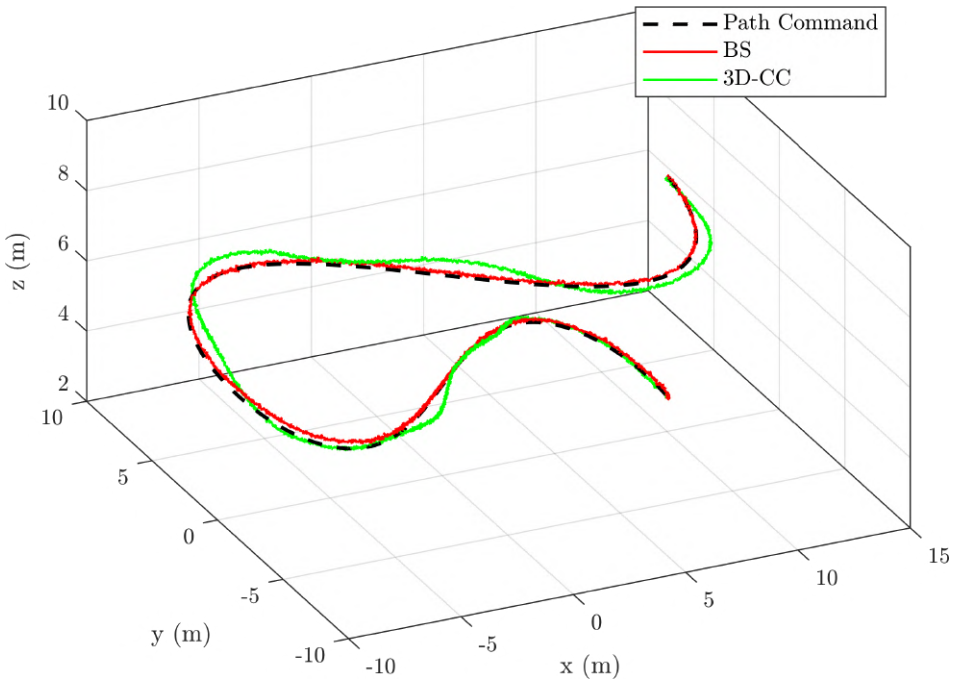

FIGURE 4.13: 3D trajectory evolution comparing *Backstepping* and *3D-Carrot-Chasing* for one lap of the Lemniscate under realistic flight conditions (2 $^{m}/_{s}$).

steps. Each point on the graphs corresponds to a simulation of a full lap of the Lemniscate as in Fig. 4.13.

4.2.5 Discussion

From the steady state regime simulation results, in Table 4.2, it can be observed that the *Backstepping* algorithm is the one that achieves the best performance in terms of distance to the path, which is usually the most important parameter. *BS* presents a little more than 1.5mm of error, compared to the 8mm of the *Feedback Linearisation* algorithm or the almost 2cm of the geometric algorithms. However, the *BS* algorithm presents a very large computational effort in comparison to the other algorithms, which obtain similar values of this indicator. Another drawback of the *BS* algorithm is that it reduces the cruise velocity in order to achieve this great precision. This is observed in the higher time that it takes to accomplish one lap of the helix. Regarding the *yaw* error, again the *Backstesping* algorithm has the best response with a mean *yaw* error of 1.6°. The geometric algorithms have a larger yaw error than *BS* because these algorithms are not designed to keep the vehicle tangent to the path but to face it to the VTP. *FL* algorithm shows a *yaw* error even larger than the geometric algorithms, although it is designed to be tangent to the path.

Regarding the transient regime results, in the simulations where the initial x-coordinate changes, the *3D-NLGL* algorithm presents the worst transient behaviour. It has wide oscillations along the path that increase its convergence time. The *3D Carrot-Chasing* algorithm also presents an oscillating performance. However, the oscillations are smaller, as can be noticed from the 3D

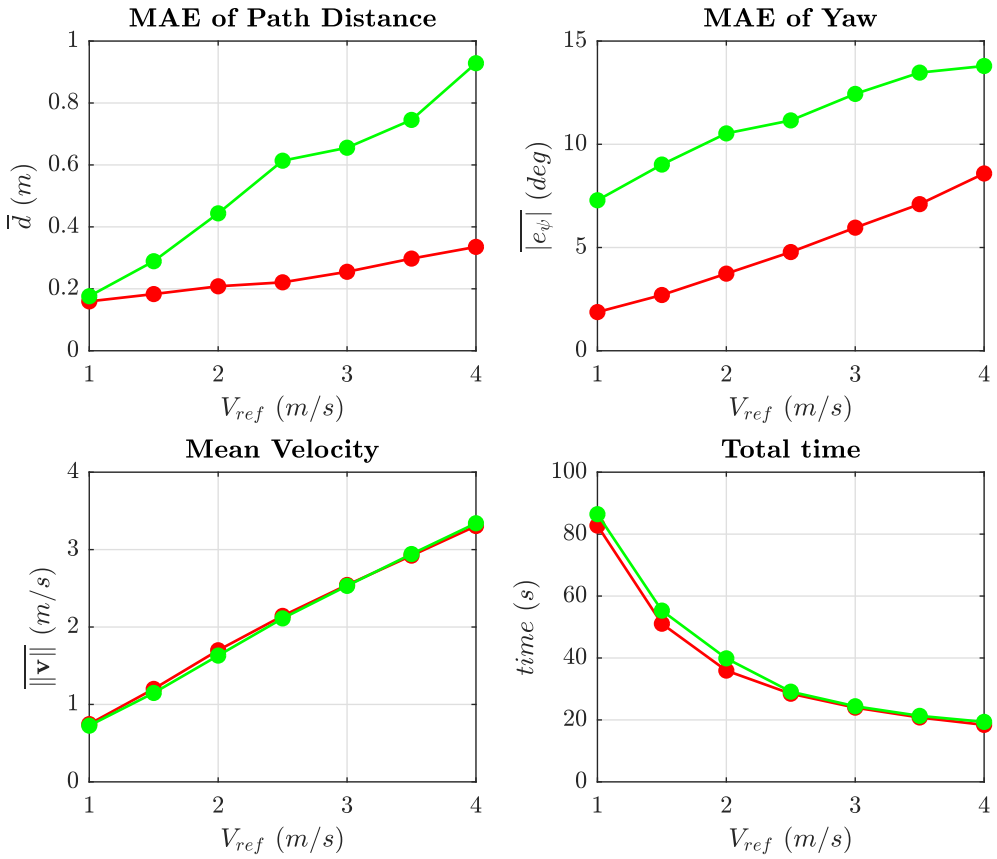

FIGURE 4.14: Simulation results varying the velocity reference from 1 to 4 $^m/_s$ under realistic flight conditions: *Backstepping* (red) and *3D Carrot-Chasing* (green).

plots of Fig. 4.6 and Fig. 4.7. This performance distinction between the two geometric algorithms is due to the way they approach to the path. The *3D-NLGL* algorithm moves directly to the minimum distance point on the path ($\gamma_{d_{min}}$), while the *3D-CC* moves always to a δ distance from this point. This slight difference becomes significant on the final performance. From our experience, these oscillations on the geometric algorithms, and thus the convergence time, can be reduced by increasing the geometric control parameters (L and δ). However, this results in an increment of the path distance error too.

The modified initial x-coordinate simulations also reflect that the control-oriented algorithms (*BS* and *FL*) obtain very similar stabilization times. Neverthelles, *FL* results in higher path distance errors because it converges to a further point on the path, as evidenced by the convergence path position plot (Fig. 4.5).

The control-oriented algorithms, especially the *FL* algorithm, make a higher control effort to remain on the path (Fig. 4.5) than the geometric algorithms. Control effort results at $x = 3$ should not be considered as some algorithms converged with as few as 2 or 3 time steps.

Analysing the results in which the initial *yaw* orientation is varied, it can be seen again that the control-oriented algorithms obtain better performance than the geometric ones. *3D-NLGL* and *3D-CC* perform similarly in terms of convergence time and path position. That is because,

when the vehicle is close to the path, the distinct behaviour between these two algorithms on the approach to the path takes no remarkable effect. Also, it is important to note that the control-oriented algorithms make a significantly larger control effort to correct the *yaw* angle than the geometric algorithms. That is due to the aggressive *yaw* control that the *BS* and *FL* algorithms produce.

When the quadrotor deals with the effect of time-varying wind disturbances, represented in Fig. 4.11 and Table 4.3, *FL* algorithm performs worst. *BS* is the algorithm that handles best the external forces in terms of path distance error. Note that, apparently, the *FL* does not have sufficient strength to cope with wind, evidenced by its low average speed and the short distance along the path (γ = 2.72) that it is able to cover, compared with the other algorithms. The yaw error of geometric algorithms is larger than the one of *BS*, while the path distance error is similar.

From the results of the simulations with realistic flight conditions (Fig. 4.14), it can be seen that with a velocity reference of 1 $^{m}/_{s}$ the path distance error of the *BS* and *3D-CC* algorithms is quite similar, but when the velocity reference increases, the gap between the algorithms increases and *3D-CC* starts behaving worse. Regarding the *yaw* error, it behaves similarly in both algorithms, although it is always larger in *3D-CC*. It is also observed that the average velocity of the vehicle is almost identical for both algorithms, having always nearly the same relative error. However, it is possible to notice that the time spent by the *BS* algorithm is slightly shorter due to its smaller path distance error.

TABLE 4.5: Qualitative comparison of the path following algorithms.

	BS	***FL***	***NLGL***	***CC***
Path Distance Control	1	2	4	4
***Yaw* Control**	1	4	2	2
Velocity Control	4	3	1	1
Convergence to the Path	1	2	4	3
Computational Effort	4	3	1	1
Wind Disturbance	1	4	2	2
Control Effort	3	4	2	1
Design & Tuning Effort	3	4	1	1
Path Adaptability	3	4	1	1
State Information Requirements	3	3	1	1
Model-Based	3	3	1	1
Domain of Attraction	1	4	2	2

Table 4.5 presents a comparison between the four algorithms summarizing the simulation result

analysis and other qualitative indicators. This table sorts from best (1) to worst (4) the four algorithms on each qualitative aspect.

Regarding the design & tuning effort, the geometric algorithms (*3D-NLGL* and *3D-CC*) are easier to implement and they require only one parameter to tune. In contrast, the implementation of the control-oriented algorithms (*BS* and *FL*) is tedious since both present several complex derivatives and they have more parameters to tune. Furthermore, in the *FL* algorithm it is necessary to define a mathematical function (h_3) that calculates the minimum distance point on the path ($\gamma_{d_{min}}$) given the position of the vehicle. This function depends exclusively on the geometry of the path and no solution is guaranteed to exist for every path.

The path adaptability defines the capability of each algorithm to adapt to different types of paths. The requirement of finding h_3 makes *FL* the worst for adapting to new path shapes. The *BS* algorithm is rated third since the defined path ($\boldsymbol{p}_d(\gamma)$) needs to be continuous and derivable. The geometric algorithms are again the best rated because they can be adapted to any path by changing only their specific control parameter.

The control-oriented algorithms require the full state information, while the geometric ones (along with the *PID*s of the autopilot) only require the position, attitude and the velocities on the x and y axis. Furthermore, the design of the control-oriented algorithms is model-based, while it is model-free in the geometric algorithms. Because of this, they can be applied to different kinds of vehicles. Both qualitative aspects are reflected in Table 4.5.

To end up, the domain of attraction of each algorithm is evaluated. The behaviour of the algorithms with the vehicle away from the path and in different initial conditions is analysed. *Backstepping* is global since it is based on the non-linear model and it includes a saturation on the position error term that results also in a saturation on the control actions. The *BS* algorithm is rated first on this characteristic since when the vehicle is far from the path and independently of the initial condition it is always able to converge to the path. *Feedback Linearisation* should be a global algorithm as well, since it is based on a full-state non-linear dynamic inversion. Nevertheless, the simulation results show that it is not really global, since it becomes unstable in specific initial conditions. That is, opposite to the *BS* case, the approach velocity of the *FL* algorithm grows unbounded as the distance to the path increases. Moreover, it gets unstable when the vehicle is in position $x = 0, y = 0$ since the third output (h_3) is indeterminate. This is the reason why the transient experiments start on $x = 0.5m$ and not on $x = 0$. For these reasons, the *FL* algorithm is rated worst in the domain of attraction aspect. Regarding the geometric algorithms, the *NLGL* has a domain of attraction defined by the distance L. When the vehicle is further away a special procedure must be performed, which may not assure stability. The *CC* is considered global, as the convergence to the path is assured for any position of the vehicle. Note that the analysis of the domain of attraction for the geometric algorithms depends exclusively on the position states, since the rest of states (velocity, orientation and angular velocity) are controlled by the autopilot.

Chapter 5

Adaptive Geometric Path Following

In Chap. (2) several control-oriented, geometric and learning-based algorithms for UAV path following are reviewed and compared. The most prominent and popular are implemented in a realistic quadrotor model in Chap. (4). Conclusions reveal that, in spite of its slightly worse performance in comparison with the control-oriented algorithms, geometrical algorithms are easier to implement, require less state information and result in a lower computational and control effort. Therefore, they become a wise solution for the path following problem.

Geometric guidance algorithms were originally described in the missile guidance literature, but most of them were adapted to other type of vehicles, such as UGVs [122][70] or UAVs. Examples of geometric algorithms that are applied to UAVs are: *NonLinear Guidance Law* (*NLGL*) [172], *Trajectory Shaping* [137], *Vector-field* based [127] and *Pure Pursuit* [57]. Some of those geometric guidance algorithms have very few (1 or 2) parameters to tune. These parameters define the performance and the stability of the controller [172][70]. The choice of the proper values of those parameters depends on factors such as the velocity of the vehicle or the reference path shape, and needs to be done manually each time the experiments conditions vary. In [134] the authors adjust these parameters for different path following algorithms by means of an optimization procedure based on *Genetic Algorithms*. However, this optimization is performed off-line and needs to be redone when path conditions vary.

The present chapter is focused on the study of the parameter selection for the *NonLinear Guidance Law* and *Carrot-Chasing* algorithms, and presents an adaptive approach based on the use of neural networks. That is, the parameters of the geometric algorithm are automatically generated (on-line) depending on the path shape and the vehicle's velocity. Stability proofs of the proposed approach are given. The performance of the adaptive algorithms are assessed by numerical simulations.

All the results and graphs reported in this chapter were obtained with the Path-Flyer quadrotor simulator, which implements a Separated, Guidance and Control structure (Fig. 3.7) along with

the standard nonlinear dynamic equations. It also includes the dynamics of the motors and other aerodynamic effects, such as drag forces. In this simulator the autopilot (Section 3.3) is implemented by a set of PID controllers. More information of the Path-Flyer platform in Section 3.4.

5.1 Adaptive NLGL

The aim of this section is to develop an adaptive strategy for the *NonLinear Guidance Law* (*NLGL*) applied to a quadrotor vehicle to solve the path following problem. The algorithm corresponds to the three-dimensional version of the *NLGL* developed in Section 4.1.3.

5.1.1 Performance of *NLGL* Depending on the Parameter Selection

This subsection analyses the performance of *NLGL* as a function of parameter L. The mean absolute error (MAE) in terms of distance to the path, noted $\overline{d}$ hereafter, is used to evaluate the performance of the algorithm. The dynamics of the autopilot and the vehicle are not modified throughout the chapter. That is, it is not analyzed how having different inner dynamics affects on the optimal selection of L.

Path distance MAE as a function of parameter L

The simulation scenario is a circular path of radius R and a constant velocity reference of 1 $^m/_s$. The vehicle starts in a point on the path at hover condition (i.e. zero linear and angular velocities) and it is requested to perform a full lap of the path. The average distance of the vehicle to the path ($\overline{d}$) is measured in each simulation.

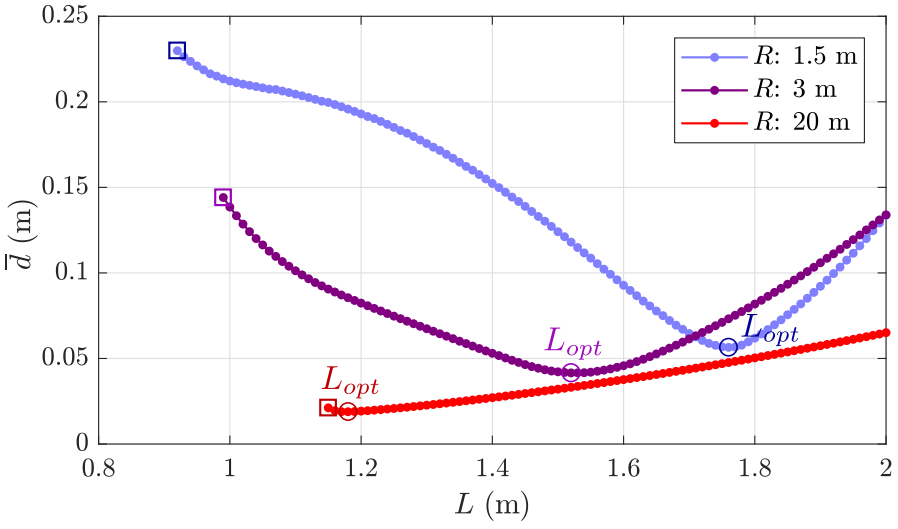

FIGURE 5.1: MAE of d on a full lap of a circular path varying L. Constant speed: $1^m/_s$. Different radius: 1.5m, 3m and 20m.

Fig. 5.1 shows the results for three different path radius (1.5m, 3m and 20m) when parameter L is varied in each simulation by steps of 0.01m. Each point on the graph corresponds to a full lap simulation. The initial value of L on each of the three cases, denoted with a square on the plot, is the first one that does not make the system unstable. That is, smaller values of L make the vehicle's trajectory become unstable. Note that for each radius, there always exists a value of L that achieves the minimum error. In this section, this value is represented by L_{opt}, which stands for optimal L.

Optimal L value in function of vehicle's velocity and path radius

The results showing how L_{opt} changes with the vehicle's reference velocity (V_{ref}) and the path radius (R) are presented in Fig. 5.2. The reference paths are circumferences again. Each point on the graph shows the value of L_{opt} for a given velocity reference and path radius. L_{opt} was obtained with an exhaustive discrete search (step of 0.01m). The behaviour of L_{opt} in function of V_{ref} and R can be divided in three zones, represented by A, B and C in Fig. 5.2. Zone C shows constant L_{opt} with regard to the path radius. The speed of the vehicle here is slow enough to assume the path as a straight line. Thus, the value of L_{opt} corresponds to the value of the optimal L for a straight path. Zone B corresponds to the regular behaviour of L_{opt} for a circular path. Finally, zone A is the one where the vehicle moves too fast for the given path radius and it is not able to follow the path correctly. There is no value of L that makes the vehicle converge to the path and follow it, due to the kinematic constrains of the plant.

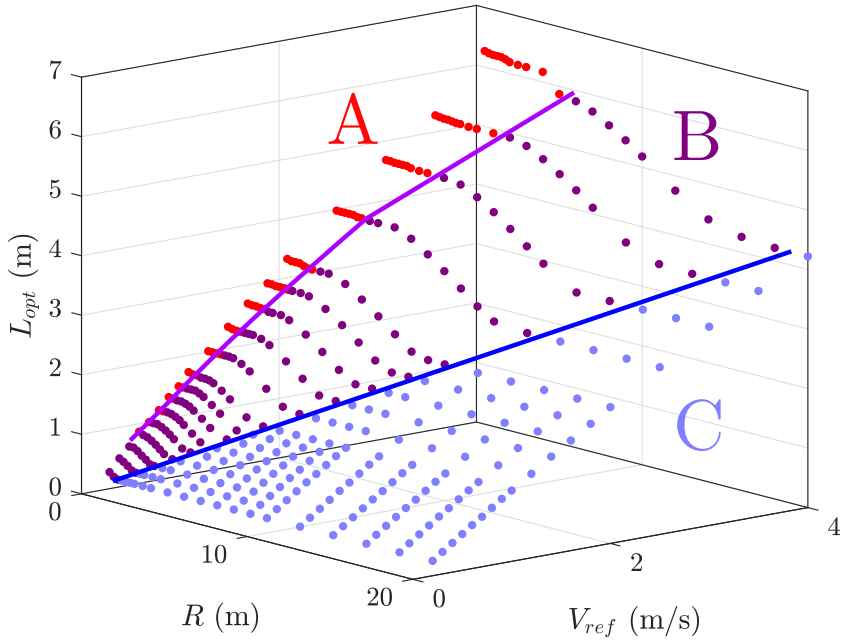

FIGURE 5.2: L_{opt} in function of the path radius and vehicle's velocity.

The average of the path distance error ($\overline{d}$) achieved with L_{opt} when the vehicle is requested to perform a full lap starting from hover conditions is reported in Fig. 5.3. To make it clearer, a surface was chosen to represent this error, however, this plot is obtained with the same values

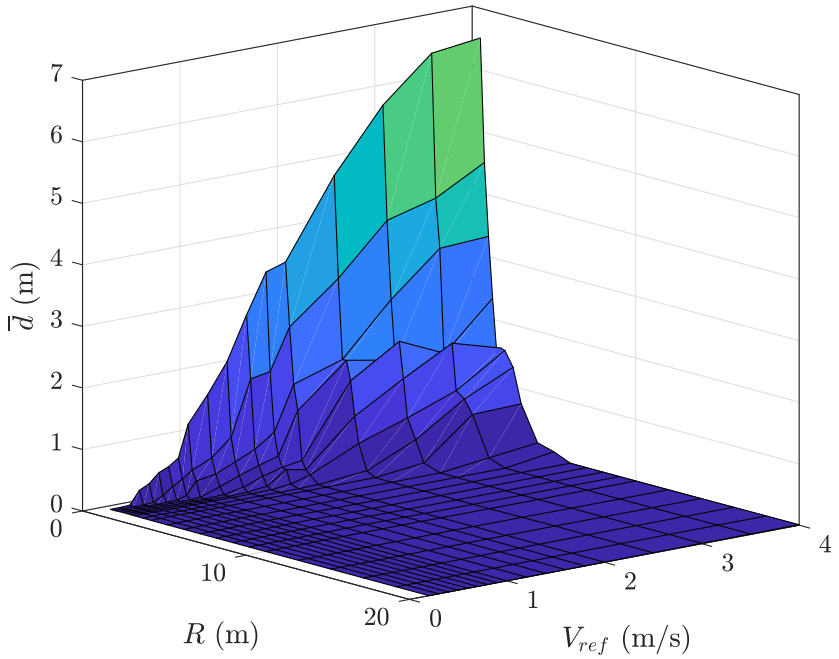

FIGURE 5.3: MAE of d with L_{opt} in function of the path radius and vehicle's velocity.

of R and V_{ref} of Fig. 5.2. From this plot, it is clear that the vehicle is unable to follow the path in Zone A, as evidenced by the very high path distance error exhibit on this zone.

5.1.2 Adaptive Parameter Selection

As seen in the previous section, the selection of parameter L depends on the velocity of the vehicle and the radius of the path. To develop an adaptive approach for the *NonLinear Guidance Law* it is necessary to define a function (or algorithm) that computes the value of L_{opt} given the vehicle's velocity and path radius. The output of this function has to be as similar as possible to the set of points obtained in Fig. 5.2.

In this chapter a neural network (NN) is employed to fit a surface to the given set of points. This problem can also be solved using other types of approximations (polynomial, wavelets, etc.), however, since the shape of the surface to be fitted is similar to a sigmoidal function the NN seems to be the best option. The details of this NN are explained below.

Neural network

The network architecture consists in three hidden layers full connected in feedforward sequence. The inputs of the NN are the radius of the path and the velocity of the vehicle. The output is parameter L_{opt}. The hidden layers have 3, 6 and 3 neurons, respectively. The activation function of the neurons is sigmoidal, chosen mainly because of its resemblance with the surface required to fit. This architecture was found as the one that fits better the surface minimizing the overfitting problem. The training algorithm used is the Levenberg-Marquardt method with a regularization

term. The regularization term is included to reduce the average generalization error, in other words, to avoid the overfitting problem. Fig. 5.4 shows the surface function obtained by the neural network with the training points denoted in red. These points correspond to the points of Zone B and Zone C of Fig. 5.2. That is, points of Zone A were removed, since this zone presents inaccurate performance for any value of L.

Note that the real velocity (u) was used as input to the NN, instead of the reference velocity plotted in Fig. 5.2. That is because, despite these two velocities are quite similar in the analysed simulation scenarios, they can differ significantly in other circumstances, e.g., when the vehicle is required to move on the z direction, due to a bad performance of the inner velocity controller or because of external disturbances.

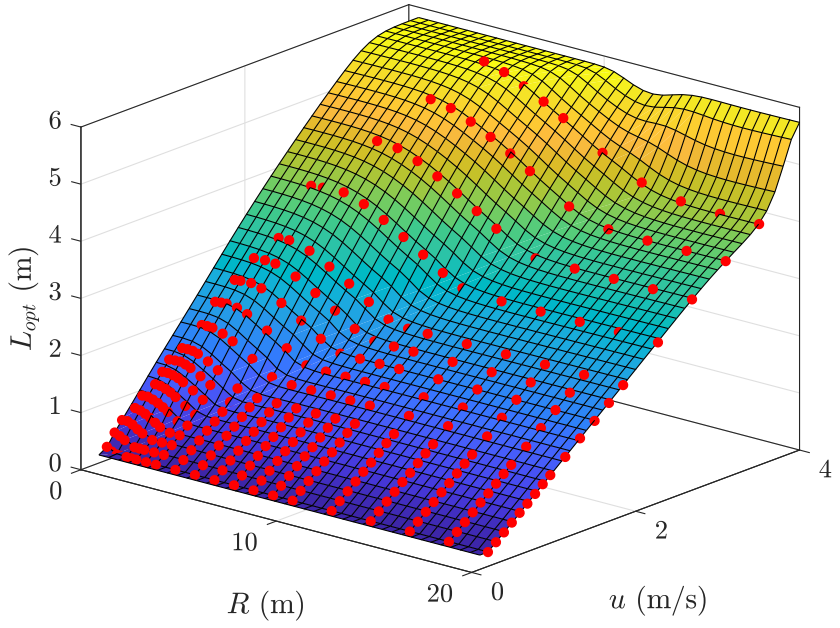

FIGURE 5.4: Neural Network surface and its training points (NLGL).

Obtaining the path radius

With the developed NN it is possible to obtain the optimal value of L, given the current x-body velocity of the vehicle and the radius of the path. The radius of a continuous parametrized path ($\boldsymbol{p}_d$, Definition 2.2.1), can be calculated with the inverse of the path curvature (k), given by

$$k(\gamma) = \frac{1}{R(\gamma)} = \frac{\left\| \frac{d\boldsymbol{p}_d(\gamma)}{d\gamma} \times \frac{d^2\boldsymbol{p}_d(\gamma)}{d\gamma} \right\|}{\left\| \frac{d\boldsymbol{p}_d(\gamma)}{d\gamma} \right\|^3} \tag{5.1}$$

The value of γ used to calculate k (or equivalently, R) must be chosen carefully. One could be tempted to use $\gamma_{d_{min}}$ (i.e. γ at minimum distance to the vehicle), but that is not a sensible choice. It would be equivalent, when driving a car, to steer it when you are already on the

curve. For that reason, it is necessary to have an anticipation distance to have the capability of adapting to the curves that come.

A new simulation scenario was designed to determine the optimal anticipation distance for each curve. This scenario is represented in Fig. 5.5. It consists on a straight path of 10 meters long followed by a semicircle of radius R_c. The vehicle is required to follow the path at constant speed. In each simulation case, the reference velocity and the radius of the semicircle are changed. An anticipation distance (d_a), computed along the path, is used to obtain R, which is fed to the NN. The instantaneous value of parameter L, determined by the neural network, is used afterwards to obtain the VTP.

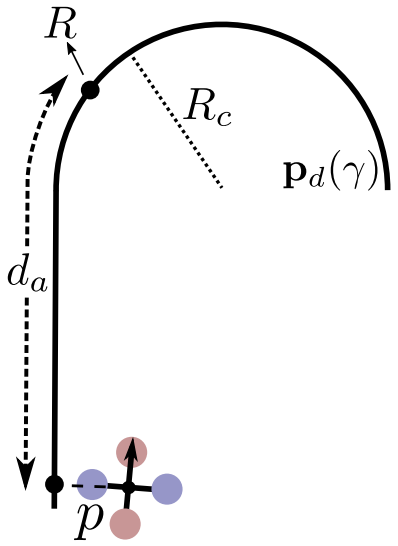

FIGURE 5.5: Simulation scenario to obtain the optimal anticipation distance ($d_a{}^*$).

From the described simulations, an optimal anticipation distance for each case given by the vehicle's velocity and semicircle radius is obtained. The optimal anticipation distance is the one that achieves the best performance in terms of path distance error. The results of this set of simulations are presented in Fig. 5.6. This plot represents the relation of the optimal distance of anticipation, noted by $d_a{}^*$, with the average of the x-body velocity ($\overline{u}$) and the radius of the semicircle (R_c). It is important to mention that only the Zone B (Fig. 5.2) was analysed in these simulations, since in the straight line zone (Zone C) the same value of L is obtained with any anticipation distance. That is because the line and the semicircle will always have the same value of L_{opt} in this zone.

The set of optimal anticipation distances ($d_a{}^*$) is approximated by a linear function of speed (u) and radius (R), Eq. (5.2), which is also represented in Fig. 5.6. The regression mean absolute error of the approximation is $0.1316m$. Note that, the anticipation distance increases as the vehicle's velocity increases or the radius of the curve decreases.

$$d_a{}^* = 1.5544u - 0.1787R + 0.2053 \tag{5.2}$$

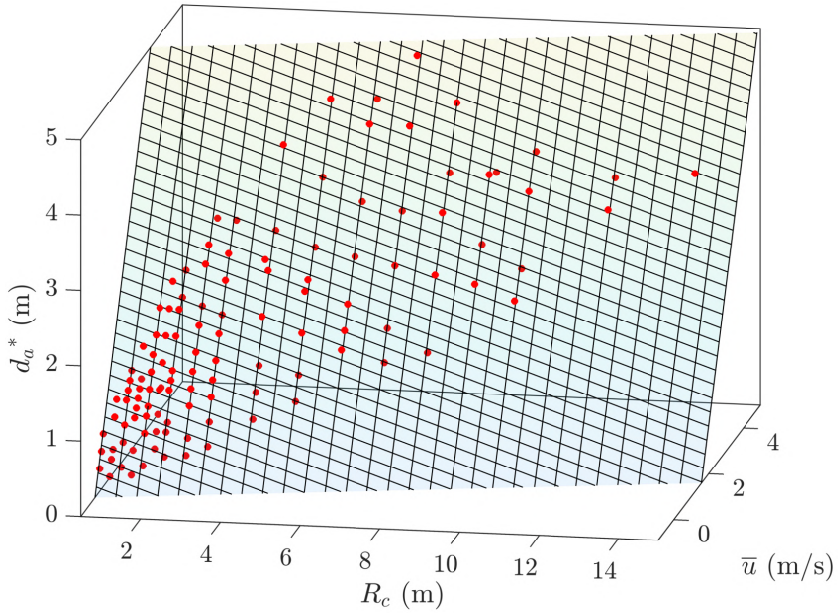

FIGURE 5.6: Optimal anticipation distance in function of the vehicle's velocity and radius of the curve.

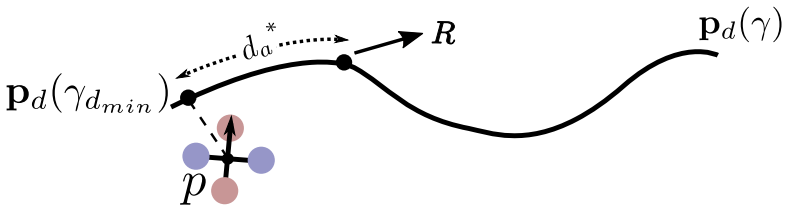

FIGURE 5.7: Optimal anticipation distance can not be used to obtain the path radius (input of the NN).

Eq. (5.2) computes the optimal anticipation distance. This distance determines the point of the path to be evaluated to obtain the R (with Eq. 5.1) that is fed to the NN (Fig. 5.7). Paradoxically, it is not possible to calculate the optimal anticipation distance without knowing R in advance ($d_a{}^*$ depends on the radius R). Therefore, using $d_a{}^*$ to obtain the point to evaluate R is not possible. Instead, the solution undertaken in this work consists on using Eq. (5.2) to generate an anticipation distance window along the path, and then, implement an algorithm to find the most restrictive curve (depending on its radius and the distance to it) inside this window. The anticipation window for a given velocity u is created by considering the extreme values of R. That is, the maximum anticipation distance (d_A) of this window is given when the radius is 0 and the minimum anticipation distance (d_a) is obtained with the maximum possible radius, noted R_{max}. This maximum radius is equivalent to the maximum radius of Zone B of Fig. 5.2, since for all $R > R_{max}$, the L_{opt} remains constant. The value of this radius is obtained by approximating the edge between B and C zones by a linear function. Eqs. 5.3 and 5.4 show how the minimum and maximum distances are computed, respectively. With these distances the anticipation distance window is defined (Fig. 5.8).

$$d_a = 1.5544u - 0.1787R_{max} + 0.2053 \mid R_{max} = 4.991u + 0.0333 \tag{5.3}$$

$$d_A = 1.5544u - 0.1787R_{min} + 0.2053 \mid R_{min} = 0 \tag{5.4}$$

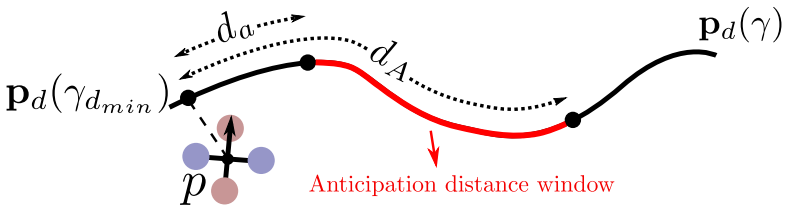

FIGURE 5.8: Anticipation distance window to obtain the path radius (input of the NN).

The anticipation window might contain several curves, but there is always one that is most urgent to consider, depending on how far it is and how closed it is. This curve determines R to be used as an input of the neural network. Curves are considered more urgent when their radius is smaller and also when they are closer to the vehicle. To determine the most urgent curve, it is necessary to consider both parameters (radius and distance). That is, if the optimal anticipation distance (Eq. 5.2) computed for a given curve radius is shorter than the actual distance to that curve, it means that this curve is not urgent, i.e. there is still time to tackle this curve. However, if the optimal anticipation distance is larger or equal than the actual distance to that curve, it means that it is necessary to immediately consider this curve. With this reasoning, the radius of the most urgent curve is the minimum possible radius on the anticipation window which satisfies that the optimal anticipation distance is greater or equal than the distance to the curve. Therefore, the criterion employed to obtain the radius of the most urgent curve, R_u, is stated as

$$R_u := \min_{\gamma} \left(R(\gamma) \mid {d_a}^* \geq d_\gamma \right) \tag{5.5}$$

where d_γ is the distance along the path from point $\gamma_{d_{min}}$ to the evaluated point γ inside the anticipation window.

The process of searching for the most urgent radius on the anticipation distance window is detailed in Algorithm 5. First, the distances of anticipation that define the window are calculated (d_a and d_A). Next, these distances along the path are converted to points on the path given by the scalar parameter γ. These two points can be calculated knowing the point at a minimum distance to the vehicle ($\gamma_{d_{min}}$), the shape of the path and d_a and d_A. Then, the most urgent radius, inside the window given by γ_a and γ_A, is searched by means of a loop. To this end, on each ith step of the loop, the distance from $\gamma_{d_{min}}$ to a given point on the path inside the window, γ_i, is calculated. And so the optimal anticipation distance for the path radius ($R(\gamma_i)$) of the same point is calculated. Then, if the anticipation distance, ${d_a}^*(\gamma_i)$, is larger than the distance to the point, $d(\gamma_i)$, it is stated that γ_i is a conflictive point. That is, a candidate to be the point with the most urgent radius. Finally, an *if* condition is defined to find the most urgent radius on the window, R_u, which corresponds to the smallest radius within the set of conflictive points.

Algorithm 5 Most urgent radius on the anticipation distance window

Require: u, $\boldsymbol{p}_d(\gamma) \coloneqq [x_d(\gamma), y_d(\gamma), z_d(\gamma)]^T$, $\gamma_{d_{min}}$

Obtain the limits of the window
$R_{max} = 4.991u + 0.0333$
$d_a = 1.5544u - 0.1787R_{max} + 0.2053$
$d_A = 1.5544u - 0.1787(0) + 0.2053$

Convert the limits to points on the path (γ)
$\gamma_a \leftarrow f(\boldsymbol{p}_d, \gamma_{d_{min}}, d_a)$
$\gamma_A \leftarrow f(\boldsymbol{p}_d, \gamma_{d_{min}}, d_A)$

Search for the most urgent radius
$R_u \coloneqq R(\gamma_a) \leftarrow$ `init. variable`
for $\gamma_i \coloneqq \gamma_a$ `to` γ_A **do**
  Calculate the distance to γ
    $d(\gamma_i) = \int_{\gamma_{d_{min}}}^{\gamma_i} \|\boldsymbol{p}'_d(\gamma)\| \, d\gamma$

  Calculate the optimal anticipation distance for $R(\gamma_i)$
    $R(\gamma_i) \leftarrow$ `Eq. (5.1)`
    $R_{sat} = \min(R(\gamma_i), R_{max})$
    $d_{a_{opt}}(\gamma_i) = 1.5544u - 0.1787R_{sat} + 0.2053$

  Verify if it is a conflictive point
  if ${d_a}^*(\gamma_i) \geq d(\gamma_i)$ **then**
    Check if it is the most urgent
      if $R(\gamma_i) < R_u$ **then**
        $R_u \coloneqq R(\gamma_i)$

return R_u

Velocity reduction

To assure that the vehicle does not enter zone A of Fig. 5.2, in which it is not able to follow the path, a maximum velocity reference has been defined, as shown in Eq. (5.6). This velocity depends on the path radius, and it is computed from the edge function between the A and B zones shown in Fig. 5.2, which can be approximated by an inverse tangential function. This expression has been obtained by means of the MatLab curve fitting tool.

$$V_{path_{max}} = 1.886 \arctan(0.6197R - 1.037) + 1.783 \tag{5.6}$$

5.1.3 Stability Analysis

This subsection gives conditions for the stability of the adaptive *NLGL*. In [70] the stability of the *NLGL* algorithm, implemented on a UGV, is analysed and a stability condition for parameter L is given.

The UGV and the quadrotor present quite different dynamics. Nevertheless, *NLGL* is implemented using a SGC structure (i.e. with an autopilot). As the autopilot is in charge of controlling the inner dynamics of the vehicle, the *NLGL* algorithm only takes into account the kinematic model of the vehicle. If it is imposed that the quadrotor can only be driven on the x-body direction (no lateral movement), as it is already assured with the presented *NLGL* algorithm, it results that the kinematic models of the UGV and the quadrotor are very similar. The most relevant difference is that the UGV moves in the 2D space, while the quadrotor moves in 3D. However, as discussed in Section 3.3 the autopilot controls the altitude, therefore *NLGL* is only in charge of controlling the movement in the xy plane. In summary, both kinematic models can be considered equivalent. Thus, the results from [70] are taken to prove the stability of the proposed approach.

The stability condition derived in [70] is stated in Eq. (5.7), where u is the velocity of the vehicle in the x body axis, τ is the time constant of the yaw angle dynamics and $k_{\boldsymbol{p}_d}$ is the curvature (Eq. 5.1) of a point on the path.

$$L > u\,\tau\sqrt{\frac{2}{1+k_{\boldsymbol{p}_d}^2}+\frac{2}{k_{\boldsymbol{p}_d}^2\left(1+k_{\boldsymbol{p}_d}^2\right)}+\frac{2}{k_{\boldsymbol{p}_d}^2\sqrt{1+k_{\boldsymbol{p}_d}^2}}} \tag{5.7}$$

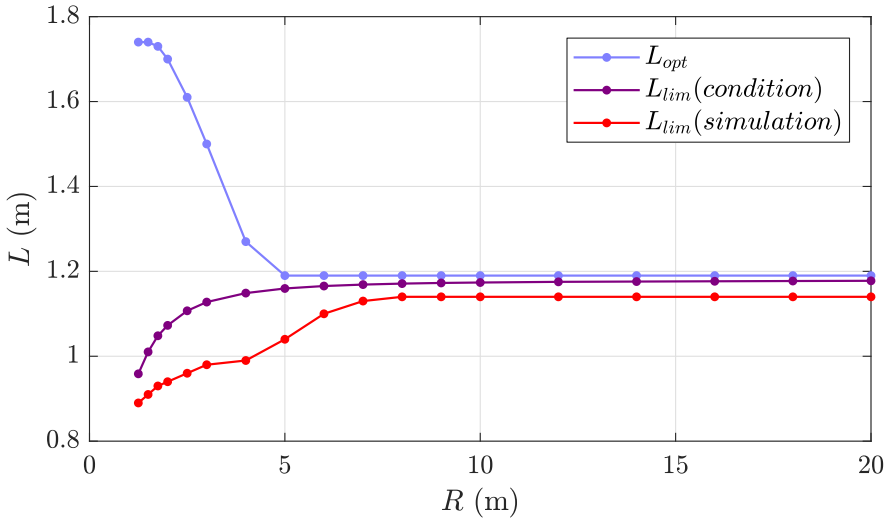

FIGURE 5.9: Optimal L (blue), stability condition limit for L (purple) and stability limit in simulation (red) in function of the path radius ($u = 1^m/_s$).

Fig. 5.9 shows the limit of the stability condition of Eq. (5.7) (in purple) and the limit of stability presented in the simulation model (in red) in function of the radius for a vehicle's speed of $u = 1^m/_s$. The stability limit of the simulation model is obtained empirically by gradually reducing L until the response of the system becomes unstable. The last value that maintains the system stable is considered the limit of stability. Furthermore, Fig. 5.9 also presents the optimal L value fitted by the NN (blue color), which fulfills the stability condition. It is important to mention that this plot only corresponds to a vehicle's velocity of $1^m/_s$, however, it has been also verified that the NN output fulfills the stability condition for all its surface (velocity from $0.2^m/_s$ to $4^m/_s$, and radius from 0.5m to 20m).

As seen in Fig. 5.9, both the theoretical condition and the empirical stability limits are similar, specially for large radius, being the stability condition more restrictive. This is because the estimation made to obtain the parameter τ, that determines the dynamics of the system, was conservative. On the other hand, it can be observed that in the zone C of Fig. 5.2, where the algorithms undertakes the path as if it was a straight line, the value of L_{opt} approximates to the limit of stability condition, while in zone C the optimal value and the limit condition diverge as the radius decreases.

5.1.4 Results

Simulation results compare the performance of the proposed *Adaptive NLGL* algorithm with the regular *NLGL*. Two path references are analyzed: an lemniscate 2D path and a spiral 3D path.

Lemniscate path

The lemniscate path is defined by Eq. (5.8). The altitude is fixed at 3m. And the amplitude is $A = 2.5$m.

$$\boldsymbol{p}_d(\gamma) = \begin{bmatrix} 2A\cos(\gamma) \\ A\sin(2\gamma) \\ 3 \end{bmatrix} \tag{5.8}$$

Fig. 5.10 shows the evolution on the xy plane of the UAV controlled by the *Adaptive NLGL* with the velocity reduction term (red) and the original *NLGL* (blue). The velocity reference is $1^m/_s$. The vehicle starts at $(x, y) = (5, 0)$ in hover conditions. Parameter L of the original *NLGL* was chosen minimizing path distance error for this particular path.

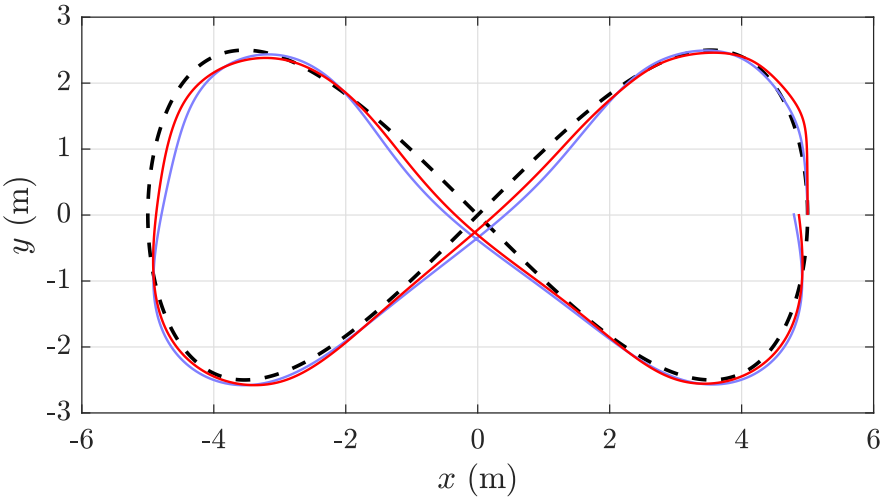

FIGURE 5.10: Trajectory on the xy plane of *NLGL* (blue) and *Adaptive NLGL* with velocity reduction (red) (Lemniscate path, $V_{ref} = 1^m/_s$).

Table 5.1 shows MAE of the path distance, the total time and the average velocity obtained by three variants of the *NLGL* algorithm performing one lap on the lemniscate path. These variants are: the original *NLGL*, the proposed *Adaptive NLGL* and the *Adaptive NLGL* with the velocity reduction term. As shown in the results, the *Adaptive NLGL* with the velocity reduction is the one that presents the smallest distance error, but a slightly lower average velocity. It is important to highlight that the *Adaptive NLGL* presents a better performance than the regular *NLGL*. Furthermore, the *Adaptive NLGL*, as opposed to the standard *NLGL*, does not need any tuning of its parameters when the reference path is changed, which becomes the main advantage of this approach.

TABLE 5.1: Results for one lap of the lemniscate path (V_{ref} = 1 m/s).

	$\overline{\mathbf{d}}$ **(m)**	**time (s)**	$\overline{\|\mathbf{v}\|}$ **(m/s)**
NLGL	0.1282	32.8375	0.9339
Adaptive NLGL	0.1054	32.9414	0.9338
Adaptive NLGL + Vel. red.	0.0947	33.4749	0.9146

Fig. 5.11 shows the evolution in time of parameter L computed by the *Adaptive NLGL* (green) and *Adaptive NLGL* with velocity reduction (red) when performing a full lap of the reference path. The velocity reduction term reduces the velocity in the curves which, at the same time, makes the algorithm reduce parameter L.

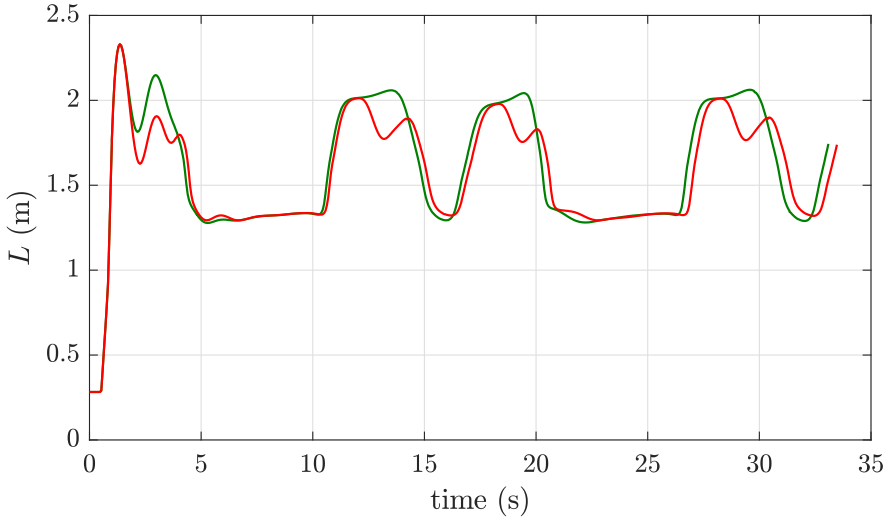

FIGURE 5.11: Evolution of L with the *Adaptive NLGL* (green) and *Adaptive NLGL* with velocity reduction (red) (Lemniscate path, V_{ref} = 1 m/s).

Spiral path

The spiral path is defined by Eq. (5.9).

$$\boldsymbol{p}_d(\gamma) = \begin{bmatrix} \gamma/2\cos(\gamma) \\ \gamma/2\sin(\gamma) \\ \gamma + 3 \end{bmatrix} \tag{5.9}$$

Fig. 5.12 shows the evolution in space of the regular *NLGL* (blue) and *Adaptive NLGL* with velocity reduction (red) following the spiral path with a velocity reference of $2^m/_s$. Again, the L of the original *NLGL* was tunned to minimize path distance error for the given path and velocity.

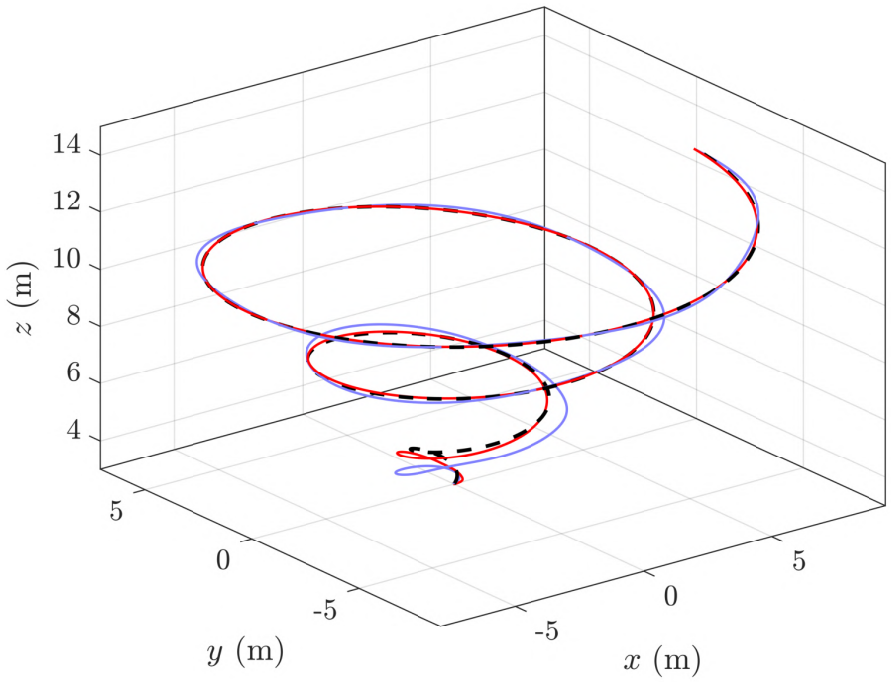

FIGURE 5.12: Trajectory of *NLGL* (blue) and *Adaptive NLGL* with velocity reduction (red) (Spiral path, $V_{ref} = 2^m/_s$).

A comparison of the obtained results with the three variants of the *NLGL* algorithm when following the spiral path is found in Table 5.2. This table shows the mean absolute error, the total time and the average velocity exhibited by the three variants when traveling from 0 rad to 6π rad of the spiral path. Again, the proposed *Adaptive NLGL* with the velocity reduction term provides the best performance and the simple *Adaptive NLGL* shows a better performance than the *NLGL*.

TABLE 5.2: Results for one lap on the spiral path ($V_{ref} = 2^m/_s$).

	$\overline{\mathbf{d}}$ **(m)**	**time (s)**	$\overline{\|\mathbf{v}\|}$ $(^m/_s)$
NLGL	0.2205	53.2537	1.7899
Adaptive NLGL	0.1787	52.2592	1.7866
Adaptive NLGL + Vel. red.	0.0730	55.3999	1.6563

The evolution of parameter L computed by the *Adaptive NLGL* (green) and by the *Adaptive NLGL* with velocity reduction term (red) is shown in Fig. 5.13. The effect of the velocity reduction is mainly observed in the first 10 seconds, when the path radius is smaller.

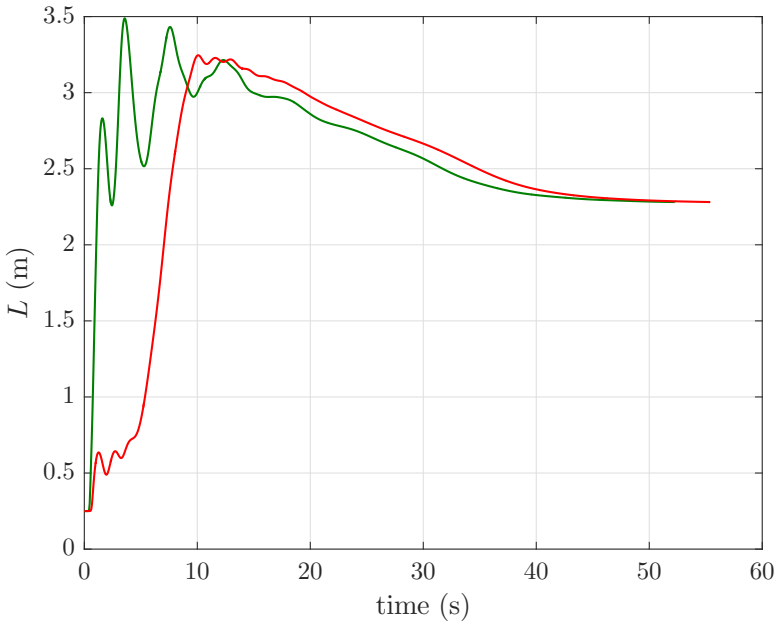

FIGURE 5.13: Evolution of L with the *Adaptive NLGL* (green) and *Adaptive NLGL* with velocity reduction (red) (Spiral path, $V_{ref} = 2^{m}/_{s}$).

5.2 Adaptive Carrot-Chasing

This section presents an adaptive version of the *Carrot-Chasing* algorithm that follows the same structure and methodology of the *Adaptive NLGL* approach introduced in the previous section. This approach is based on the three-dimensional version of this algorithm developed in Section 4.1.4. It is necessary to recall that this algorithm, just as the *NLGL*, only has one parameter to tune, that is the δ distance.

5.2.1 Adaptive Parameter Selection

The process of obtaining the *Adaptive CC* approach is similar to the one described in Section 5.1. First, the optimal parameter of the Carrot-Chasing algorithm, δ_{opt}, in function of the radius of the path and the vehicle's velocity is obtained by performing a set of simulations with a circumference path. The value of δ_{opt} in function of V_{ref} and R is presented in Fig. 5.14. This set of points is divided in three zones as in Fig. 5.14. These correspond to the Zone A, where the vehicle cannot follow the path correctly due to the kinematic constrains, the Zone B, where parameter δ varies in function of the path radius, and the Zone C, where the path radius is sufficiently large for a given high velocity so the algorithm behaves as if it were following a

straight line path. Fig. 5.15 represents the average distance error obtained with the optimal values of parameter δ.

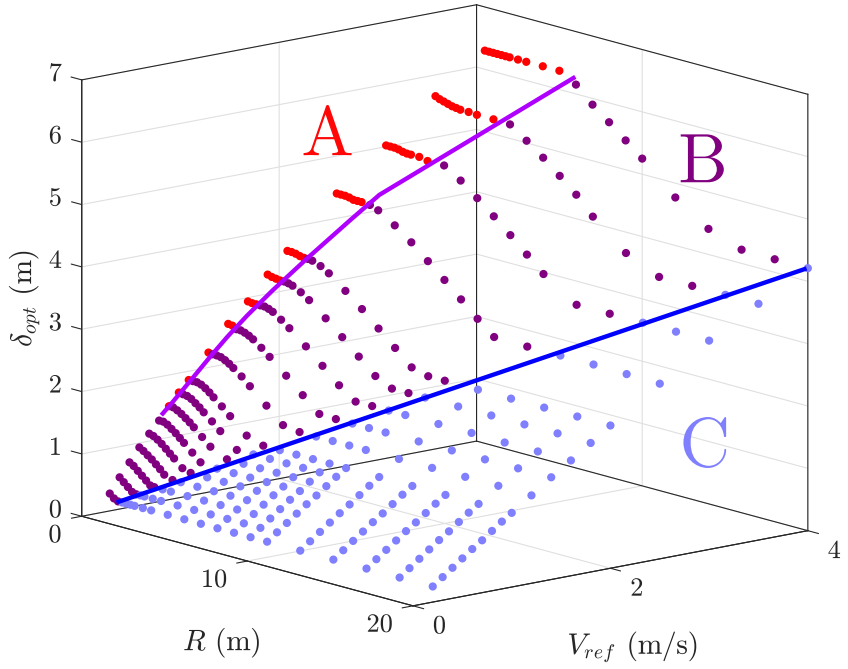

FIGURE 5.14: δ_{opt} in function of the path radius and vehicle's velocity.

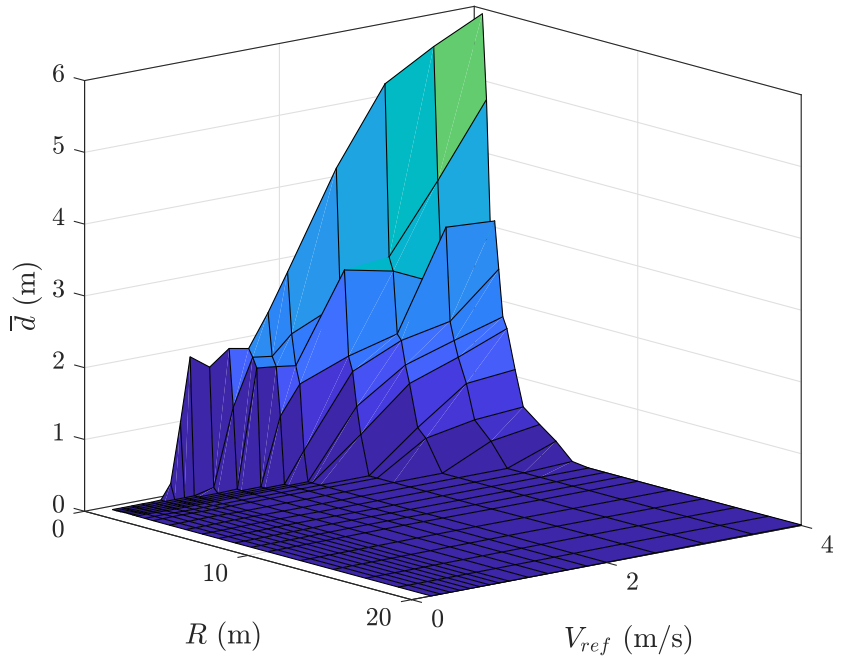

FIGURE 5.15: MAE of d with δ_{opt} in function of the path radius and vehicle's velocity.

The values of δ_{opt} are approximated by a neural network of 3 feed-forward hidden layers of 3, 7 and 3 neurons, respectively. The inputs of the net are the velocity of the vehicle and the path radius. As the NN of the *Adaptive NLGL* approach, it uses a sigmoidal function as activation function, and it is trained with the Levenberg-Marquardt method with a regularization term.

Fig. 5.16 shows the output surface of the NN with the training points highlighted in red. These points correspond to the points of zones B and C of Fig. 5.14.

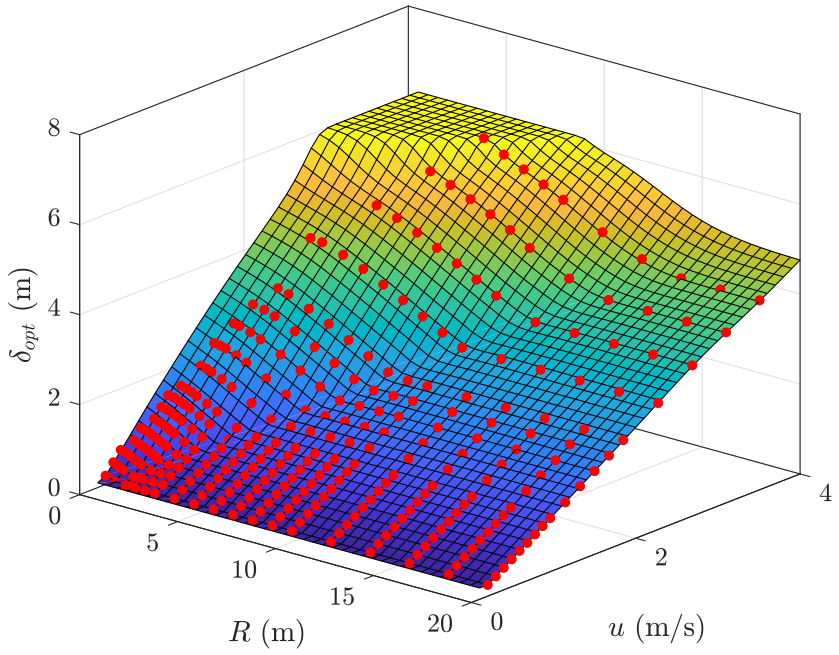

FIGURE 5.16: Neural Network surface and its training points (Carrot-Chasing).

Once again, an anticipation distance is used to evaluate the radius that is fed to the neural network. The optimal anticipation distance is obtained by performing several simulations with the path of Fig. 5.5 as described in Section 5.1.2. This set of optimal anticipation distances is approximated by a linear function of the path radius and the vehicle's velocity (Eq. 5.10). This formula is used to generate an anticipation distance window with the maximum and minimum possible radius. The Algorithm 5 is used to obtain the most restrictive radius of the path inside this anticipation distance window, which then is fed to the NN.

$$d_a{}^* = 1.8099u - 0.1535R + 0.2034 \tag{5.10}$$

A velocity reduction term is included to avoid the algorithm to enter in the Zone A of Fig. 5.14. Eq. (5.11) shows the maximum permitted velocity of the vehicle in function of the path radius. This function was obtained with the MatLab curve fitting tool approximating the edge between zones A and B of Fig. 5.14.

$$V_{path_{max}} = 5.529 \arctan(0.2779R - 0.4759) + 2.017 \tag{5.11}$$

5.2.2 Results

The results presented in this section compare the *Adaptive CC* algorithm with the *Adaptive NLGL* approach in realistic flight conditions simulations. Noise on the measurements are included. This noise is adjusted so that it emulates the observed in the real sensors. Furthermore, a realistic wind profile is generated. Similarly to the wind profile of Fig. 4.12, the magnitude of the wind disturbance is generated as a random walk of $10^{m}/_{s}$ in average and the direction is also a random walk with an average of $\pi/4$ rad (i.e. $+x$,$+y$ direction). Again, results are obtained with the Path-Flyer benchmark. First, both adaptive approaches are tested with the lemniscate path of Eq. (5.8) at a velocity of $1^{m}/_{s}$. The trajectories obtained by the proposed adaptive approaches following the stated path are shown in Fig. 5.17. Both approaches include the velocity reduction term. The results of these simulations are summarized in Table 5.3, where the cross-track error, the total time and the average velocity are evaluated. The two algorithms are able to follow the path correctly, presenting a similar performance. Nevertheless, the *Adaptive NLGL* is capable of achieving a slightly lower average distance error.

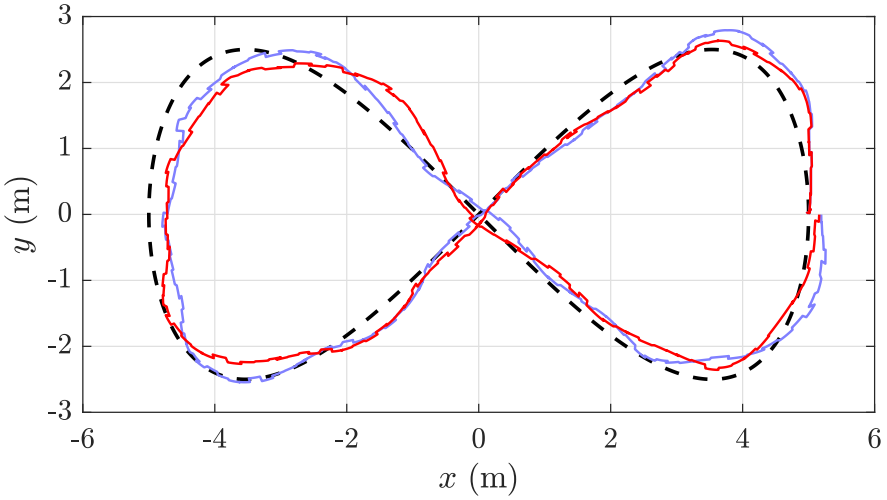

FIGURE 5.17: Trajectory on the xy plane of *Adaptive NLGL* (blue) and *Adaptive CC* (red) in realistic flight conditions (Lemniscate path, $V_{ref} = 1^{m}/_{s}$).

TABLE 5.3: Results for one lap on the lemniscate path in realistic flight conditions ($V_{ref} = 1^{m}/_{s}$).

	$\overline{\mathbf{d}}$ **(m)**	**time (s)**	$\overline{\|\mathbf{v}\|}$ $(^{m}/_{\mathbf{s}})$
Adaptive NLGL	0.1607	32.7287	0.9388
Adaptive CC	0.1738	31.3084	0.9557

Next, the adaptive algorithms are tested with a spiral path of constant height, Eq. (5.12), at a velocity reference of $2^{m}/_{s}$. The trajectory performed by the adaptive approaches while following the spiral path is presented in Fig. 5.18. Table 5.4 compares the path distance error, the time to travel the path and the average velocity of the vehicle of the two algorithms. Again, the *Adaptive NLGL* approach achieves better performance than the *Adaptive CC* in terms of path following

error. In this path, both algorithms present larger cross-track errors than in the lemniscate, especially in the first part of the path that has a very sharp curve. Nevertheless, in this part of the path both algorithms reduce their reference velocity to procure a better performance.

$$\boldsymbol{p}_d(\gamma) = \begin{bmatrix} 1.25\gamma\cos(\gamma) \\ 1.25\gamma\sin(\gamma) \\ 3 \end{bmatrix} \tag{5.12}$$

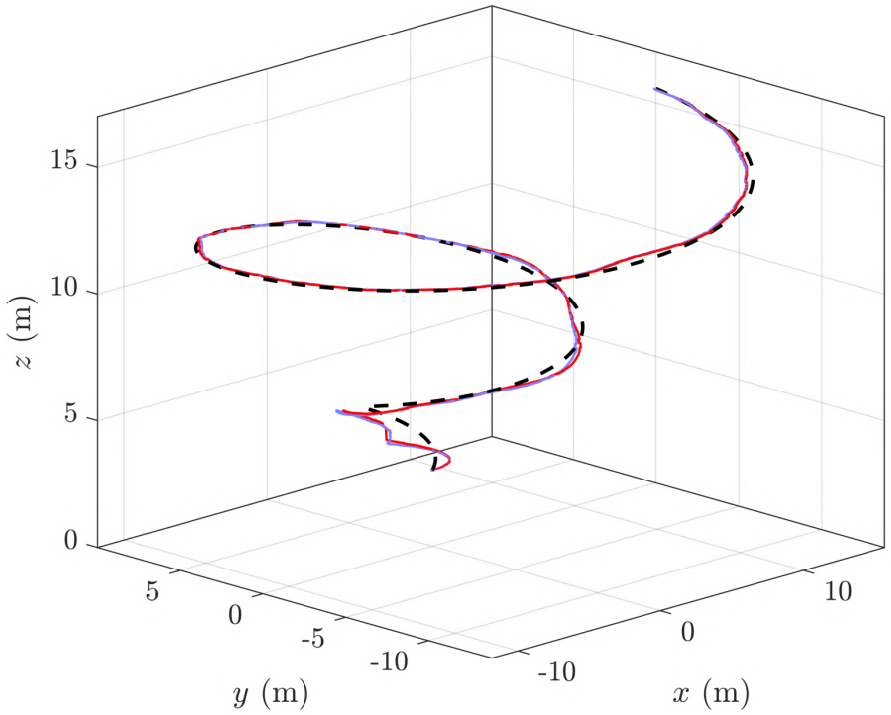

FIGURE 5.18: Trajectory on the xy plane of *Adaptive NLGL* (blue) and *Adaptive CC* (red) in realistic flight conditions (Spiral path, $V_{ref} = 2^{m}/_{s}$).

TABLE 5.4: Results for one lap on the spiral path in realistic flight conditions ($V_{ref} = 2^{m}/_{s}$).

	$\overline{\mathbf{d}}$ **(m)**	**time (s)**	$\overline{\|\mathbf{v}\|}$ $(^{m}/_{s})$
Adaptive NLGL	0.3449	51.0562	1.8954
Adaptive CC	0.3753	51.1005	1.8946

More results comparing both algorithms and their standard versions can be straightforwardly obtained with the freely available and open Path-Flyer platform [142].

Chapter 6

Path Following with Deep Reinforcement Learning

An adaptive version of the *NonLinear Guidance Law* (NLGL) was developed in Chap. (5). In that work neural networks were used to approximate the relation between the optimal control parameters of *NLGL*, the vehicle's velocity and the path's shape. Results showed that it outperforms the standard *NLGL* in different experimental conditions without requiring any retuning of its parameters. The main drawback of this approach is that it is an ad-hoc solution since it relies on approximations made from the collected data of the mathematical model. Therefore, if the vehicle's model or the dynamics of the attitude controller change, it is necessary to carry out again a large number of simulations, extract the information of interest and adjust the data, which may become tedious.

The limitations on the adaptability of the approach to different experimental conditions as well as the portability to different multirotor vehicles motivated the subsequent work. It was important, nonetheless, to preserve the control structure and the advantages of the geometrical algorithms. The emerging deep reinforcement learning theory appeared as a promising option to accomplish those objectives. In recent years, a significant progress has been made in the fields of reinforcement learning (RL) and deep learning. Thus, now RL is no longer constrained to discrete and small environments. *Deep Q-Network* (DQN) [125] and *Deep Deterministic Policy Gradient* (DDPG) [103] are two of the most popular deep RL algorithms. In *DQN* the inputs of the agent are images, while *DDPG* is especially designed for continuous state-action spaces. Both algorithms have been used to solve diverse computer science and engineering problems [29][165][194][177][100]. *DDPG* has been also implemented on a quadrotor vehicle to solve the landing problem [139] with successful results. Other quadrotor applications of deep reinforcement learning can also be found in the literature [86][92][97][124][135].

In this chapter three different approaches implementing the *DDPG* algorithm to solve the path following problem in a quadrotor are presented. The approaches implement the same structure and concept of the geometrical algorithms. That is, they use a separated control and guidance structure with an autopilot tracking the attitude and velocity commands. Each approach

emerges as an improved version of the preceding one. The first approach uses only instantaneous information of the path for solving the problem. The second approach includes a structure that allows the agent to anticipate to the curves. The third agent is capable to compute the optimal velocity according to the path's shape. The agents are implemented in the tensorflow-python framework an trained in Gazebo-ROS using the RotorS simulator, a realistic multirotor simulator (Section 3.5). The agents are trained to deal with noisy sensor measurements and to perform well when the vehicle is far from the reference path. The resulting agents are implemented and validated in the Asctec Hummingbird experimental platform.

6.1 Problem Statement

The aim of this chapter is to develop a deep reinforcement learning agent capable of solving the path following problem for a quadrotor vehicle. The agent must be capable of learning online from real experimental tests and must simplify the training process of the work presented in Chap. (5). This approach will follow the control structure of geometric algorithms, a Separated Guidance and Control (SGC) structure (Fig. 3.7). The agent must be able to work with continuous state-action spaces and also be portable to other multirotor vehicles. Moreover, this agent must compute the proper velocity of the vehicle which, according to the defined reward, best adapts to the shape of the given path. This agent will be implemented with the *Deep Deterministic Policy Gradient* algorithm. It will be trained in a simulated environment and tested experimentally.

Fig. 6.1 shows the typical reinforcement learning structure adapted to the control structure of the geometric algorithms. In this structure the agent is the path following algorithm and the environment includes the autopilot controller, the path reference and the quadrotor's environment. The reinforcement learning agent receives the current state and reward and computes the action that is sent to the environment.

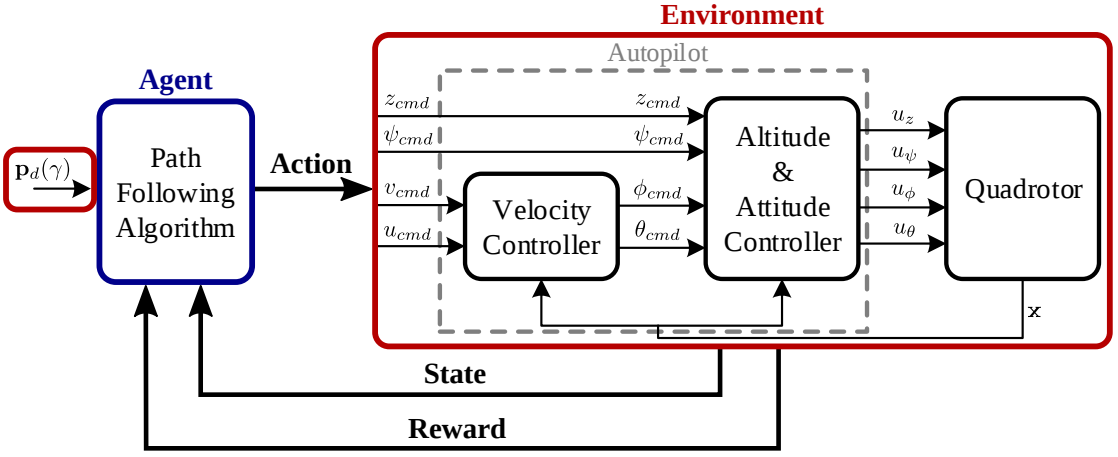

FIGURE 6.1: Reinforcement learning structure adapted to the SGC structure.

6.2 Deep Deterministic Policy Gradient

The deep reinforcement learning algorithm implemented in this work is the *Deep Deterministic Policy Gradient*. This algorithm is an improvement of the standard *Deterministic Policy Gradient* [164] algorithm including new concepts of deep learning theory. One of its major advantages is that it is able to provide good performance in large and continuous state-action space environments, which motivated its selection.

Deep Deterministic Policy Gradient [103] is an actor-critic RL algorithm. It is off-policy since the policy that is being improved is different from the policy that is used to generate the action to compute the loss function. And it is model-free because it makes no effort to learn the dynamics of the environment. Instead, it estimates directly the optimal policy and value function.

Fig. 6.2 shows a common structure of an actor-critic agent, where the policy (actor) is represented independently from the value function (critic). According to the learned policy function ($\mu(s)$), the actor computes the optimal action depending on a state of the environment. The critic estimates the value function ($Q(s,a)$) given the state and the action. The value function gives us information of the expected cumulated future reward for this state-action pair. The critic is also in charge of calculating the temporal-difference error (TD) (i.e. the loss function) that is used on the learning process for both the critic and the actor. In deep reinforcement learning the policy function and the value function, actor and critic, are approximated by neural networks.

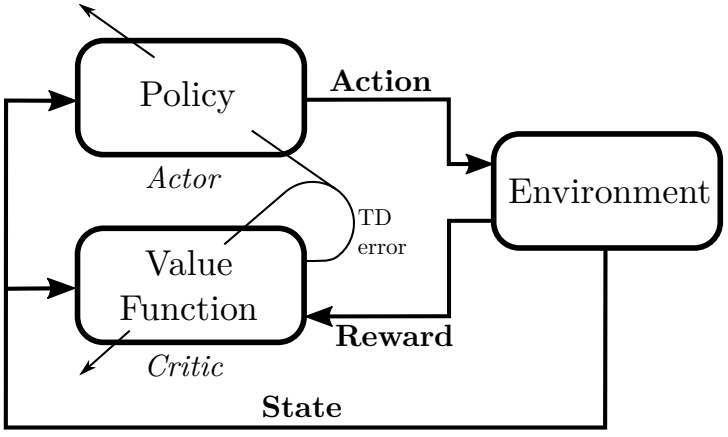

FIGURE 6.2: Actor-Critic agent structure.

DDPG uses two characteristic elements of Deep-Q-Network [125]; the replay buffer and the target networks, which are used to stabilize the learning of the Q-function. A replay buffer is a finite sized memory that stores the transition tuple at each step. Fig. 6.3 shows the main elements of the transition tuple. This tuple is formed by the current state (s_i), the action (a_i), the obtained reward (r_i), the next state (s_{i+1}) and a boolean variable that indicates if the next state is terminal or not (t_i). A terminal state is understood as a state where the experiment ends. At each timestep the critic and the actor are trained from a minibatch obtained by sampling random tuples of the replay buffer. This way of training reduces time correlation between learning samples and facilitates convergence in the learning process.

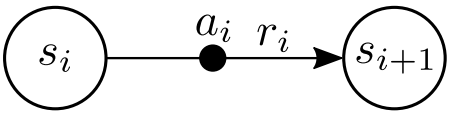

FIGURE 6.3: Elements of the transition tuple.

On the other hand, a target network is a network used during the training phase. This network is equivalent to the original network being trained and it provides the target values used to compute the loss function. Once the original network is trained with the set of tuples of the minibatch, the trained network is copied to the target network. Nevertheless, in *DDPG* the target network is modified using a soft update, rather than directly copying the network weights. This means that the target weights are constrained to change slowly. The use of target networks with soft update allows to give consistent targets during the temporal-difference backups and makes the learning process remain stable. Note that *DDPG* requires four neural networks; the actor and the critic and their respective target networks.

When the agent states or actions have different physical units it can be difficult for the neural networks to learn properly and to generalize the solution of the problem. The batch normalization technique [81] is included in the *DDPG* algorithm to avoid this issue. This technique is widely used in deep learning and consists, essentially, on normalizing each dimension of the samples in a minibatch to have zero mean and unit variance.

Eqs. (6.1 - 6.2) show the gradient functions used to update the weights of the critic and actor, respectively. ϕ are the set of weights of the critic network and θ the weights of the actor, η_ϕ and η_θ are the learning rates of the critic and actor, B represents the minibatch of transition tuples and N its size. Target networks are represented with the prime symbol. y_k (Eq. 6.3) are the target Q-values (Not to be confused with target networks) and are used to compute the loss function. The weights of the critic are updated to minimize this loss function. The discount factor, a value between 0 and 1 that tunes the importance of future rewards to the current state, is represented by γ_{rl} (it is usually represented by γ in the literature, however, in this thesis this symbol is assigned to the virtual arc parameter of the PF problem -Definition 2.2.1-). Note that the target Q-Values (Eq. 6.3) are obtained from the outputs of the actor and critic target networks, following the target network concept.

$$\Delta\phi = \eta_\phi \nabla_\phi \left(\frac{1}{N} \sum_{i \epsilon B} \left(Q(s_i, a_i \mid \phi^{Q'}) - y_i \right)^2 \right) \tag{6.1}$$

$$\Delta\theta = \eta_\theta \nabla_\theta \left(\frac{1}{N} \sum_{i \epsilon B} Q(s_i, \mu(s_i \mid \theta^\mu) \mid \phi^Q) \right) \tag{6.2}$$

$$y_i = r_i + \gamma_{rl} Q'(s_{i+1}, \mu'(s_{i+1} \mid \theta^{\mu'}) \mid \phi^{Q'}) \tag{6.3}$$

Eqs. (6.4 - 6.5) show the update of the weights of the target networks from the trained networks. Parameter τ indicates how fast this update is carried on. This soft update is made each step

after training the main networks.

$$\phi^{Q'} \leftarrow \tau\phi^{Q} + (1-\tau)\phi^{Q'} \tag{6.4}$$

$$\theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \tag{6.5}$$

6.3 Agent Environment

The environment of the agent includes the robot together with the robot's environment [173]. In this work the robot is the Asctec Hummingbird quadrotor vehicle (Fig. 3.1). Chap. (3) describes the multirotor experimental platform and the mathematical model of this vehicle. This section gives details of the simulation environment wherein the agent is trained.

6.3.1 Training Environment

First training steps of the agent are unpredictable and can become unsafe for the real platform. That is why having a simulated environment is very important in order to maintain the integrity of the experimental platform. Training the agent in a realistic and complete simulated environment will strengthen its effectiveness on real experiments.

In this work a simulation environment was built in the Gazebo-ROS (Robot Operating System) platform, making use of the RotorS simulator [53]. RotorS, as explained in Section 3.5, is a multirotor simulator integrated in Gazebo-ROS which, among the available multirotor models, has a model of the Asctec Hummingbird quadrotor (Fig. 6.4), the vehicle studied in this work. Since certain modifications were made in the real Hummingbird vehicle, some parameters of the simulation model were updated too (some sensors, pc and other items where placed on the vehicle in such a way that inertias and mass changed). Furthermore, a model of the sensors was included and adjusted to resemble the sensors of the actual quadrotor platform. Nevertheless, simulations assuming ground truth measurements (i.e. ideal sensors) can still be made.

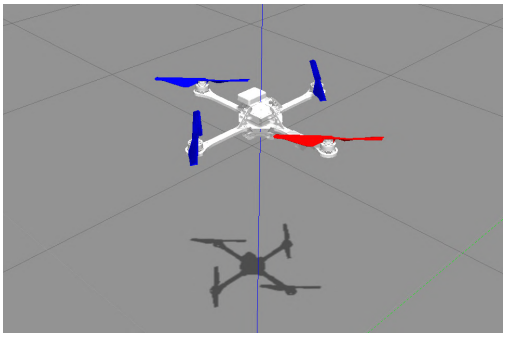

FIGURE 6.4: GUI of RotorS simulator with Hummingbird model.

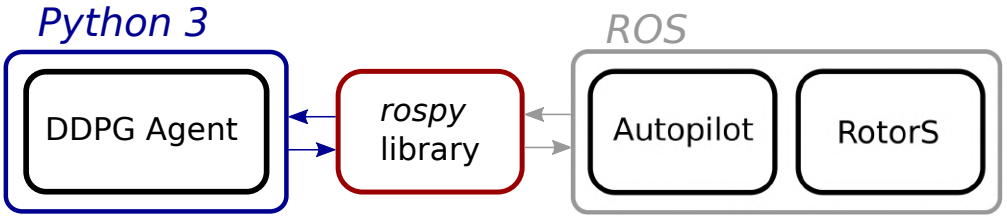

FIGURE 6.5: Scheme of the training environment.

The autopilot of Section 3.3 was implemented as a package in ROS. This autopilot was already tested in real experiments with success as shown in Section 3.4 and in [146], and it presents a similar response on the RotorS simulated environment, thus proving the validity of the model. However, it is important to mention that the mathematical model (and its parameters) of the Asctec Hummingbird quadrotor of the RotorS platform is not exactly the same as the one presented in Section 3.2. Thus, the behaviour of the vehicle in RotorS can be slightly different than in the Path-Flyer benchmark, which presents a very accurate model with respect to the real vehicle (see Section 3.4).

The *DDPG* agent was programmed in python 3.5, using the tensorflow and tflearn libraries to generate and train the neural networks. These libraries permit to save (and restore) the trained nets in order to perform tests or retrain them. Since ROS is only prepared for version 2 of python, the agent was implemented as a regular python script and it communicates with ROS (to subscribe and publish topics) by means of the *rospy* library.

A simplified scheme of the simulated environment is shown in Fig. 6.5. The complete scheme with the set of nodes and topics programmed in the RotorS environement is shown in Fig. 3.16 (This scheme also includes the obstacle avoidance node developed in Chap. 8). The main advantage of building this environment in ROS is that, since the real platform also runs under ROS, the same code of the autopilot and the *DDPG* agent can be transferred to the real quadrotor platform. Thus, the real environment is equivalent to the one presented in Fig. 6.5, except that RotorS simulator is substituted by the real vehicle and sensors.

6.4 DDPG for Path Following

This section presents the main characteristics of the DRL agents that are developed in this chapter. That is, states, actions and rewards are defined. Other details regarding the structure of the networks or the type of noise added to the actions are introduced as well. Three different approaches, implemented using the *Deep Deterministic Policy Gradient* algorithm, are presented. Each approach emerges as an improved version of the preceding one.

6.4.1 First Approach: Two States

According to the structure of Fig. 3.7, the path following algorithm must compute four control commands (z_{cmd}, ψ_{cmd}, u_{cmd} and v_{cmd}). Nevertheless, in this first approach the deep reinforcement learning agent is only in charge of computing the reference of the yaw angle (ψ_{cmd}). Actually, the action (a) produced by this agent is not directly the yaw command but a desired correction ($\psi_{corr,k}$ given in $^{rad}/_{s}$) over the current yaw angle. Eq. (6.6) shows how the yaw reference at step k is produced, where Δt is the time step. The reason to use the angle correction and not the angle itself as the agent action is to avoid undesired fast angle changes. Moreover, note that the correction is made over the current value of yaw and not over the last yaw reference, which would lead to an incremental control action. Having an incremental control action is equivalent to adding a new integral to the plant, which in this case results in an unstable behaviour. Hence, the selected action achieves a smooth movement while keeping the stability of the system.

$$\psi_{cmd,k} = a_k \, \Delta t + \psi_k \;\; | \;\; a_k = \psi_{corr,k} \tag{6.6}$$

The other commands defined by the path following controller will depend on the path specifications; z_{cmd} is given by the altitude of the path at the closest point to the vehicle (hereafter named p_{ct}, for cross track error point) and u_{cmd} is set to the desired path's velocity. Velocity on y axis (v_{cmd}), as in most of the geometrical algorithms, is fixed to 0 $^{m}/_{s}$.

The basic structure of the *DDPG* algorithm determines that, given a state of the environment, the agent will always choose the best action according to the learned policy. This may not lead the a proper exploration of the action space while training the agent. To enhance the exploration of the agent an Ornstein–Uhlenbeck noise (Eq. 6.7) is added to the action at training time. n_k is the value of the noise at the kth iteration, θ_n is a parameter that defines the speed rate of mean reversion, μ_n is the drift term which affects the asymptotic mean, Δt is the time of a step and dW_t is the standard Wiener process scaled by volatility σ_n.

Yaw command (ψ_{cmd}) including the noise signal is computed as shown in Eq. (6.8). The exploration rate decreases continuously with the number of training episodes (j) in such a way that a smooth transition between exploration and exploitation is achieved while the agent keeps learning. Parameter λ indicates the speed of this transition.

$$n_k = n_{k-1} + \theta_n \left(\mu_n - n_{k-1}\right) \Delta t + \sigma_n dW_t \tag{6.7}$$

$$\psi_{cmd,k} = \left(a_k + \frac{n_k}{j/\lambda + 1} \right) \Delta t + \psi_k \tag{6.8}$$

In this first approach, the state vector (s) is formed by two states (Eq. 6.9); the distance error (e_d) and the angle error (e_ψ), both with respect to $\boldsymbol{p}_{ct}$ (Fig. 6.6). Subscript T is referred to the

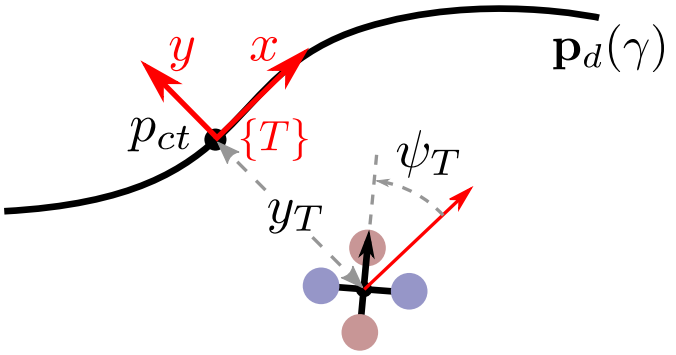

FIGURE 6.6: States of the agent are with respect of the tangential frame $\{T\}$.

tangential frame of reference $\{T\}$ that is placed on $\boldsymbol{p}_{ct}$ with x pointing to the path's tangential direction, z pointing up and y pointing to the resultant direction of $x \times z$.

$$\mathbf{s} = \{e_d,\, e_\psi\} \;\mid\; e_d = y_T,\; e_\psi = \psi_T \tag{6.9}$$

The reward defined for this agent is shown in Eq. (6.10). This is the reward function that achieves the best performance and fastest convergence among the numerous types of rewards that were evaluated (i.e. continuous or discrete, penalizing bad behaviour or rewarding good path following performance, and mixed strategies). The term $-k_1|e_d|$ penalizes the cross-track error (e_d). The term $k_2 v_T$ gives positive reward when the vehicle is moving forward on the path and negative otherwise, where v_T is the velocity of the vehicle projected in the x axis of the tangential frame of reference. k_1 and k_2 are constants that define the importance of each of the two terms. In this approach these constants take the values of 20 and 10, respectively. Being those the best values amongst several that were evaluated.

$$r = -k_1|e_d| + k_2 v_T \tag{6.10}$$

The structures of the actor (Fig. 6.7) and critic (Fig. 6.8) neural networks consist on four layered feed forward networks with 400 neurons in the first hidden layer and 300 neurons in the second one. However, while in the actor's network both the state and the action vectors are connected to the first hidden layer, in the critic networks the action vector is connected directly to the second hidden layer, since skipping the first layer proved to be beneficial. The neurons of both networks are rectified linear units (ReLU). Batch normalisation technique is used in the two layers of the actor nets, while it is only used in the state input layer in the critic networks.

Table 6.1 presents the relevant parameters and their values of this first proposed *DDPG* agent.

The agent proposed in this subsection can solve the path following problem properly (see Section 6.6). In fact, it is the best agent setup in terms of PF performance that was obtained among numerous and diverse agent setups that were tested with only two states. However, notice that these two states of the agent (Eq. 6.9) only provide instantaneous information about the path. In other words, states are computed only from the point $\boldsymbol{p}_{ct}$ in the path and they provide no

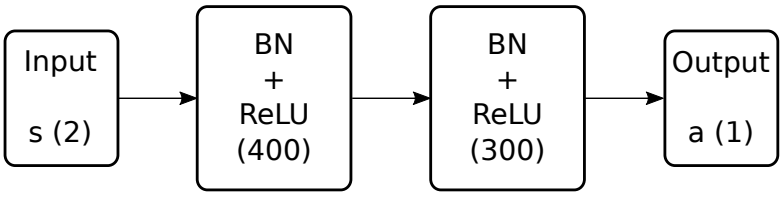

FIGURE 6.7: Actor NN structure: 2 feed-forward hidden layers.

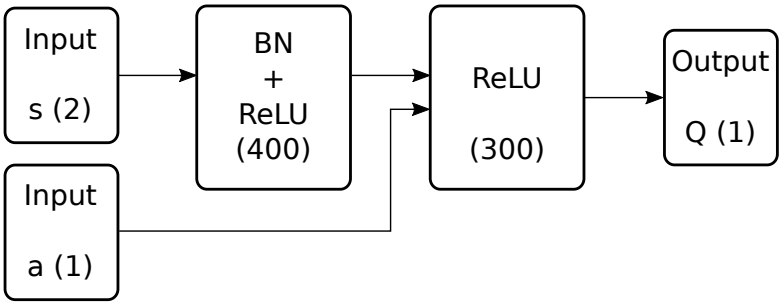

FIGURE 6.8: Critic NN structure: 2 feed-forward hidden layers.

TABLE 6.1: Parameters of the *DDPG* agent.

Symbol	**Description**	**Value**
η_θ	Learning rate of actor network.	0.0001
η_ϕ	Learning rate of critic network.	0.001
τ	Soft target update parameter.	0.001
γ_{rl}	Discount factor for critic updates.	0.99
-	Replay buffer size.	1,000,000
N	Minibatch size.	64
-	Maximum steps of one episode.	300
Δt	Agent time step.	0.1 s
θ_n	Mean reversion rate of noise function.	0.0075
σ_n	Volatility of noise function.	0.15
λ	Ratio of exploration-exploitation transition.	200

information about the path shape to come. Therefore, it is not possible for the agent to anticipate the curves of the path. Next subsection presents an improvement over this approach that handles this issue.

6.4.2 Second Approach: Anticipation State

To deal with the anticipation, the issue mentioned in the previous subsection, a new form of the state vector is proposed. The rest of the parameters and structure of the agent of the first approach are maintained. In addition to the two states defined in Eq. (6.9), another state is

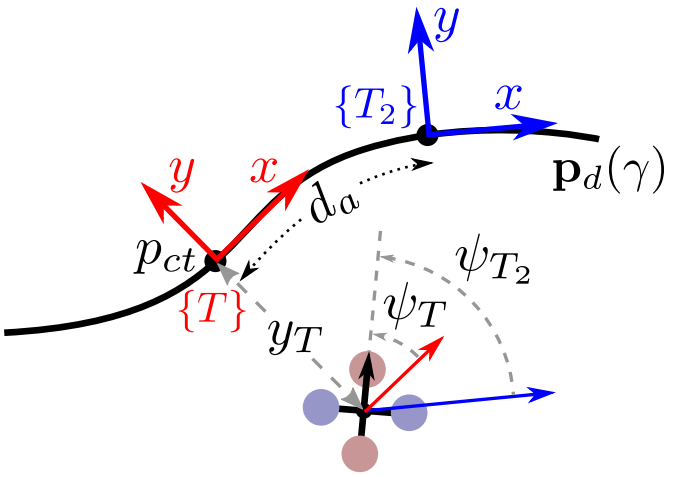

FIGURE 6.9: States of the second approach; angle error with respect forward tangential frame $\{T_2\}$.

included (ψ_{T_2}). This state is an angle error between the vehicle's yaw angle and the path's tangential angle. However, in this case the angle error is not computed from the point $\boldsymbol{p}_{ct}$ but in a point that is forward on the path, as represented in Fig. 6.6. This new state gives information about future orientation of the path with respect to the vehicle and makes it possible for the agent to anticipate the curves to come, improving substantially its path following performance (see Section 6.6). The state vector of this approach is presented in Eq. (6.11), where T_2 subscript indicates that the state is computed from the tangential frame on a point that is forward on the path.

$$\mathbf{s} = \{y_T,\, \psi_T,\, \psi_{T_2}\} \tag{6.11}$$

The distance at which the second tangential frame, $\{T_2\}$, is placed on the path is named anticipation distance and it is represented by d_a. To obtain the best possible performance of the agent, it is necessary to choose a proper anticipation distance. From different tests, it was proven that d_a depends on the velocity of the vehicle on the path. That is, with higher velocities it is necessary to have a larger anticipation distance. For instance, the optimal anticipation distance (according to the obtained PF performance) at a velocity of 1 $^m/_s$ is $0.6m$.

6.4.3 Third Approach: Adaptive Velocity

The main drawback of the previous approaches is that, with the defined structure, the agent can only learn to solve the problem at one specific velocity. That is, if during the training process the velocity of the vehicle is changed every episode, convergence cannot be achieved. In other words, the policy depends on the vehicle's velocity. This subsection presents an improvement that permits the agent to work at different velocities and also makes it capable of computing at each step the velocity of the vehicle that best adapts to the shape of the path according to the defined reward.

In order to have an agent that is resilient to different velocities and path's shapes, the first step is to include the velocity of the vehicle ($\|\mathbf{v}\|$) as a state of the agent. Nevertheless, this modification is not sufficient to accomplish our goal. In *DDPG* it is necessary to define a state vector that fulfils the deterministic property. This means that, knowing the current state vector and action, the next state can be estimated. Therefore, since the velocity of the vehicle is an exogenous variable of the system (defined by the user) it is not possible to predict its value, and thus, it is not a deterministic state. To make it deterministic, the action vector must act on the velocity state.

In this third approach, in addition to the yaw correction action defined in Eq. (6.6), a new action that computes a velocity correction ($u_{corr,k}$) over the current velocity of the vehicle is included. Eq. (6.12) shows how the velocity command on the x axis is produced from this action (including exploration noise of Eq. 6.7, only used during the training phase). Again, with the aim of avoiding fast changes on the velocity and to assure the stability of the system, a correction action has been used rather than a velocity action or an incremental action of the command.

$$u_{cmd,k} = \left(u_{corr,k} + \frac{n_k}{j/\lambda + 1} \right) \Delta t + u_k \tag{6.12}$$

Introducing this new state ($\|\mathbf{v}\|$) and new action ($u_{corr,k}$) to the agent may seem to be enough to solve the problem. However, as mentioned in Section 6.4.2, notice that the path's position where the future angle error state (ψ_{T_2}) is computed depends on the velocity of the vehicle. Therefore, having only this state computed with a fixed anticipation distance (d_a) does not provide enough information to solve the path following problem at different velocities. To deal with this problem, two solutions were considered: Adding more future angle error states at different anticipation distances or modifying at each step the anticipation distance at which the angle error is computed in function of the vehicle's velocity.

Including several future angle states at different distances resulted disadvantageous for two reasons: first, having more states makes the training process much slower; second, since at a given velocity only the information of 1 or 2 future angle states is exploited, the remaining states become irrelevant. Having many states that do not provide significant information to solve the problem leads the agent to lose effectiveness. For this reason, in this approach the mentioned issue is solved by having only one future angle state (ψ_{T_2}), which is computed with an anticipation distance adapted according to the vehicle's velocity.

Several tests at different velocities were performed in order to find the relation between the velocity of the vehicle and the optimal anticipation distance ($d_{a,opt}$). Optimal in the sense of being the distance that provides more information, and thus, results in a higher performance of the agent. The results obtained from these tests were approximated by the linear piecewise function shown in Eq. (6.13). This function computes the optimal anticipation distance as a function of the current velocity of the vehicle.

$$d_{a,opt} = \begin{cases} 0.6\|\mathbf{v}\| + 0.1 & \text{if} \quad \|\mathbf{v}\| < 1 \\ \|\mathbf{v}\| - 0.3 & \text{else} \end{cases} \tag{6.13}$$

Summarizing, in this approach the velocity of the vehicle is added as part of the state vector and the future angle state is computed with an adaptive anticipation distance ($d_{a,opt}$). A velocity correction is included in the action vector. Eqs. 6.14 and 6.15 present the state and action vectors, respectively. The agent computes the velocity command on the x axis in such a way that it adapts to the path's shape. Velocity on the y axis is still fixed to 0. The reward function of the first approach (Eq. 6.10) is also maintained. Weights of the reward (k_1 and k_2) acquire a significant role in this approach, since they define the priority of the trade-off between having small path distance error or travelling at high velocities. All parameters of Table 6.1 are preserved except for the ratio of the exploration-exploitation transition (λ), which is set to 1000. This is because the training process of this approach is slower.

$$\mathbf{s} = \{y_T,\, \psi_T,\, \psi_{T_2}(d_{a,opt}),\, \|\mathbf{v}\|\} \tag{6.14}$$

$$\mathbf{a} = \{\psi_{corr,k},\, u_{corr,k}\} \tag{6.15}$$

The ingredients that make the agent capable of following a trajectory in space with adaptive velocity have been defined. Nevertheless, it is of utmost importance to design a rich training environment that allows the agent to converge to an efficient and robust solution. Details of this training process are given in Section 6.5.

6.5 Training Process

The training process of the agents has been performed in the training environment detailed in Section 6.3.1. This training environment is integrated in a linux Xubuntu virtual machine with a dedication of 8GB RAM and four 1.80GHz processors (i7-8550U CPU). The training process is performed in real time.

6.5.1 Training of 1^{st} and 2^{nd} approaches

The first and second approaches followed the same structure in the training phase. That is, the vehicle is required to follow a half lemniscate (8-shaped) path at a constant velocity of 1 $^{m}/_{s}$ in the x body axis (u_{cmd}). This path is defined in Eq. (6.16), where A is the radius of one of the circumferences of the path, fixed to $4m$, and γ is the virtual arc, which ranges from 0 to 2π rad. The path is discretized with a precision of 0.01m between each path point.

$$\begin{aligned} x_d(\gamma) &= 2A\cos(\gamma) \\ y_d(\gamma) &= A\sin(2\gamma) \end{aligned} \tag{6.16}$$

Both agents were trained following the specified path in ideal conditions, meaning that the system uses ground truth measurements and the vehicle starts each episode at the initial position of the path with the *yaw* angle oriented tangentially to it. As denoted in Table 6.1, each episode has 300 steps of 0.1 seconds. The learning evolution of the first and the second approaches are shown in Figs. 6.10 and 6.11, respectively. These figures show, for each episode, the average path distance error ($\overline{|d|}$) and the accumulated reward ($\sum r$) in all the steps of the episode. As the agents keep training the average error decreases and the accumulated reward grows until training converges.

It is important to mention that, as the training process is stochastic, even if the same parameters and structure are maintained, the performance of the trained agents can vary. The agents presented in this chapter are the ones that achieve the best performance, in terms of path distance error, among a set of different trained agents that were obtained. In this particular case, the 1^{st} approach converged around the 120^{th} episode while the 2^{nd} approach did it approximately at episode 90.

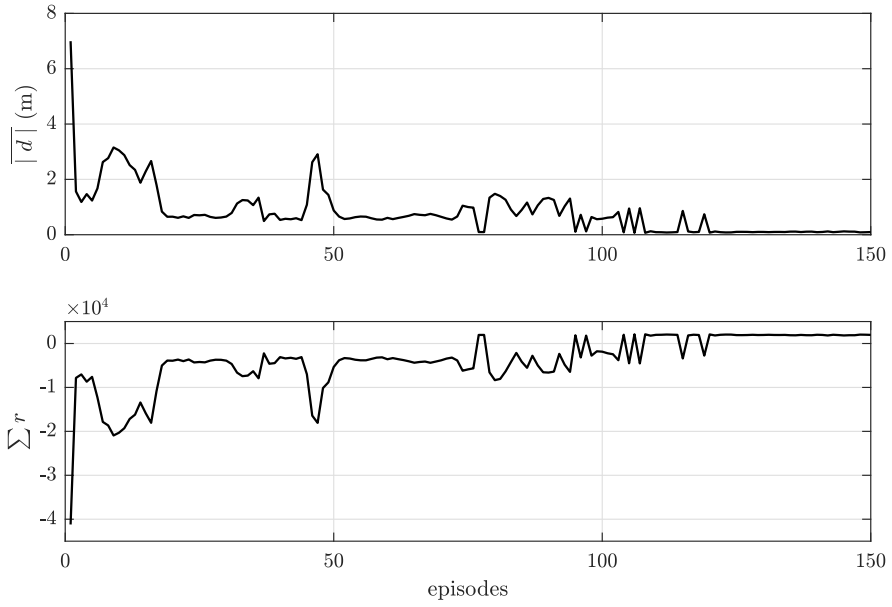

FIGURE 6.10: Average distance error and accumulated reward on each episode during training phase of 1^{st} approach agent (2 states).

The resulting agents were tested in the RotorS simulation platform (see Section 6.6.1). They proved to perform well with ground truth measurements. However, if a model of the sensors is added, the agents present some difficulties to follow the path properly. Particularly, when the vehicle moves far from the path (due to drift or jumps on sensor measurements) and needs to converge back, the vehicle can start loitering around the path without being able to converge to it.

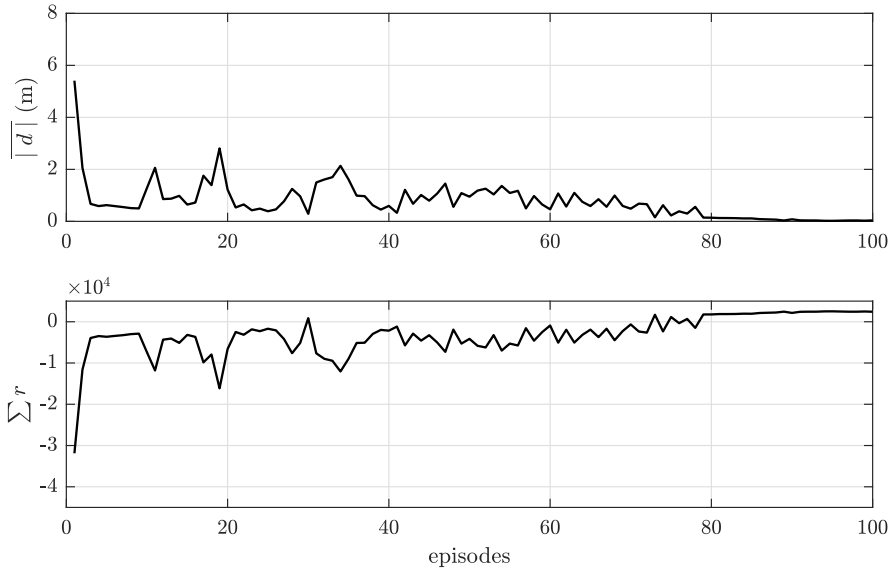

FIGURE 6.11: Average distance error and accumulated reward on each episode during training phase of 2^{nd} approach agent (3 states).

The solution to the mentioned problem could be to train the agent with the model with sensors. However, to capture the dynamics of the system with noisy measurements becomes challenging for the agent and, sometimes, training does not converge in these conditions. Alternatively, this issue is tackled by retraining the agents to learn the policy when the vehicle is far from the path. To do so, the agents are first trained as explained before, and then, they are retrained following the same path but starting at random positions and orientations different from the initial point of the path. In this way, the agents learn how to behave out of the path. Thus, if the vehicle occasionally moves out of the path because of the noisy sensor measurements, the agent will be able to drive the vehicle back to the path.

Both agents (1^{st} and 2^{nd} approaches) were trained 100 more episodes following the specified lemniscate path (Eq. 6.16) with random initial conditions. That is, in each episode of this training phase the starting position of the vehicle is set at a distance of $-2m$ to $2m$ from the initial position of the path, and the initial orientation is incremented an angle between $-\pi/2$ to $\pi/2$ radians from the initial path tangential angle. The initial position and angle are selected randomly with a uniform probability distribution in the defined intervals.

Fig. 6.12 shows the learning evolution of the 2^{nd} approach with the 100 new training episodes. Since the initial position and orientation change randomly in each episode, the obtained average distance error and accumulated reward also vary arbitrarily. For this reason, to show better the progression of this learning phase, a 20-values moving average is presented in both plots. That is, episode values are represented with gray dashed lines, while the moving average is represented with solid black lines in Fig. 6.12. The learning results show how this training phase permits the agent to learn to perform better in diverse initial conditions. This acquired knowledge will notably improve the performance in real experiments, as revealed in Section 6.6.

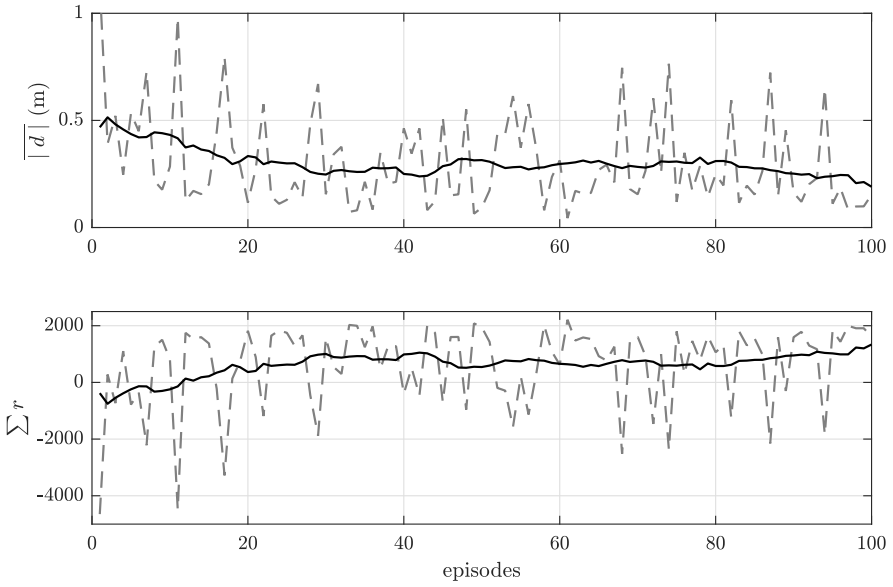

FIGURE 6.12: Average distance error and accumulated reward on each episode during training phase with non-ideal initial conditions of 2^{nd} approach agent; gray dashed lines are real values and black lines are a 20-episodes moving average.

6.5.2 Training of 3rd approach

The 3^{rd} *DDPG* approach developed in this work requires training in a richer environment than the previous versions. That is because the agent needs to train with different curves in order to learn the optimal vehicle's velocity and the *yaw* angle's policy according to the path radius.

In the training process of this agent the vehicle will be required to follow an asymmetrical half lemniscate path. This is an 8-shaped path where each circle has a different radius. This path is defined in Eq. (6.17), where A_1 and A_2 are the radius of each circumference of the path, respectively. The value of this radius is changed every episode, taking a random value between $0.5m$ and $10m$ with a uniform probability distribution. Again, the virtual arc parameter (γ) ranges from 0 to $\pi/2$ rads, and the path is discretized with a precision of $0.01m$.

$$
\begin{aligned}
x_d(\gamma) &= \begin{cases} 2A_1\cos(\gamma) & \text{if} \quad 0 \le \gamma \le \pi/4 \\ 2A_2\cos(\gamma) & \text{if} \quad \pi/4 < \gamma \le \pi/2 \end{cases} \\
y_d(\gamma) &= \begin{cases} A_1\sin(2\gamma) & \text{if} \quad 0 \le \gamma \le \pi/4 \\ A_2\sin(2\gamma) & \text{if} \quad \pi/4 < \gamma \le \pi/2 \end{cases}
\end{aligned}
\tag{6.17}
$$

The first training attempts of the agent with the stated environment resulted to be quite unfruitful. Concretely, after hundreds of episodes, the agent just learned that the best way of maximizing the reward (reward function in Section 6.4) was to keep the vehicle static. The reason for this strange behaviour can be explained as follows: since turning around arbitrarily is not penalized when, due to the lack of exploration the policy is not defined yet, whenever the agent starts moving the vehicle forward, as it is rotating, it ends up moving in the opposite

direction of the path, receiving a penalty for that policy; therefore the best action is to keep $u_{cmd} = 0$.

A simple but effective solution for such issue is proposed in this work. It consists on forcing the vehicle to move constantly by establishing a minimum velocity of 0.1 $^{m}/_{s}$. Even if this condition initially produces negative rewards, it ends up promoting the agent to learn the policy of the *yaw* action. At the same time, as soon as the velocity vector of the vehicle starts to be parallel to the path, the agent can start learning that higher velocities lead to greater rewards. Hence, a successful learning process is achieved.

The training results of this agent are shown in Fig. 6.13. This figure shows the average distance error ($\overline{|d|}$), the average velocity on the x axis ($\overline{u}$) and the accumulated reward ($\sum r$) on each episode. A 50-episodes moving average is applied to episode values to help the interpretation of each of the three plots. Again, gray dashed lines represent episode values while black lines show the moving average.

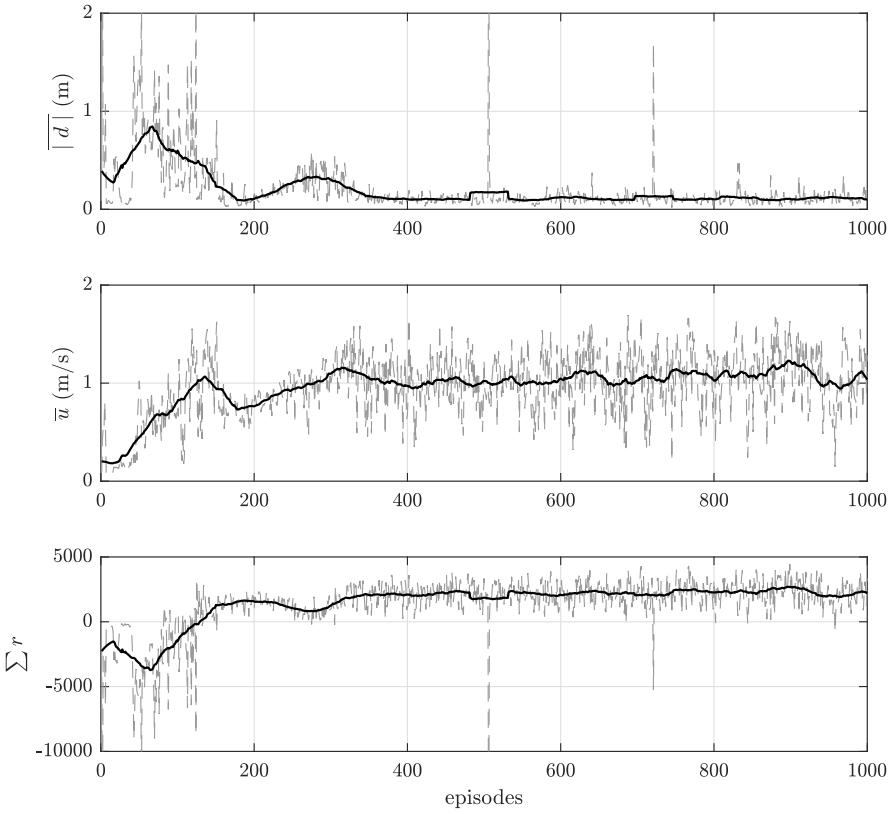

FIGURE 6.13: Average distance error, average velocity and accumulated reward on each episode during training phase of 3^{rd} approach agent; gray dashed lines are real values and black lines are a 50-episodes moving average.

It may seem that training converged around episode 400. However, even the average error or reward appear to be constant, evaluating the trained agents with simulation tests showed that they kept learning and improving their performance until around episode 1000. The reason for that is because training more episodes permits to learn the policy on unusual states.

Such long and complex training process allows the agent to learn the policy out of the path. Thus, unlike the 1^{st} and 2^{nd} approaches, this approach does not need any additional training with diverse initial conditions to improve its performance on experimental results.

6.6 Results

This section presents the results obtained with the three trained agents while following a path in different conditions. The agents were tested in simulation and experimentally with the Asctec Hummingbird platform.

6.6.1 Simulation

The simulations presented in this section were performed in the same framework where the agents were trained. That is, the RotorS simulator integrated in the ROS-Gazebo platform.

First, the three approaches were tested following a lemniscate path (Eq. 6.16), the same path used in the training phase. Again, the radius of the path is $A = 4m$. However, this time the vehicle was required to follow a full lemniscate, with the virtual arc parameter, γ, ranging from 0 to 4π rads. The vehicle started at the initial point on the path with the *yaw* angle oriented tangentially to it.

Table 6.2 shows the results obtained while following this path with ground truth measurements. That is, in the same conditions used for training. This table shows the average cross-track error ($\bar{d}$), the average velocity ($\overline{\|\mathbf{v}\|}$) and the total time taken to perform a full lap of the path by each agent. Also, to evaluate the 3^{rd} approach agent in the same conditions of the two other agents, another simulation was made with this agent limiting its maximum velocity to 1 $^{m}/_{s}$. Note that 1^{st} approach is denoted as *Agent 1* in the table, 2^{nd} approach is *Agent 2* and so on. This nomenclature is maintained hereafter in this section.

TABLE 6.2: Results for one lap of the lemniscate path, simulations with ground truth measurements.

	$\bar{\mathbf{d}}$ **(m)**	**time (s)**	$\overline{\|\mathbf{v}\|}$ ($^{m}/_{s}$)
Agent 1	0.1041	67.10	0.8707
Agent 2	0.0398	54.79	0.8780
Agent 3	0.0671	39.81	1.2276
Agent 3 ($v_{max} = 1$)	0.0669	56.10	0.8696

The results performing a full lap of the lemniscate path while using the sensor measurements instead of ground truth values, are shown in Table 6.3. Same parameters and agents of Table 6.2 are evaluated. The trajectory on the xy plane followed by these agents is shown in Fig. 6.14.

TABLE 6.3: Results for one lap on the lemniscate path, simulations with sensor models.

	$\overline{\mathbf{d}}$ (m)	time (s)	$\overline{\|\mathbf{v}\|}$ (m/s)
Agent 1	0.1123	54.79	0.9476
Agent 2	0.0895	51.70	0.9484
Agent 3	0.0968	40.00	1.2338
Agent 3 ($v_{max} = 1$)	0.0816	54.41	0.9111

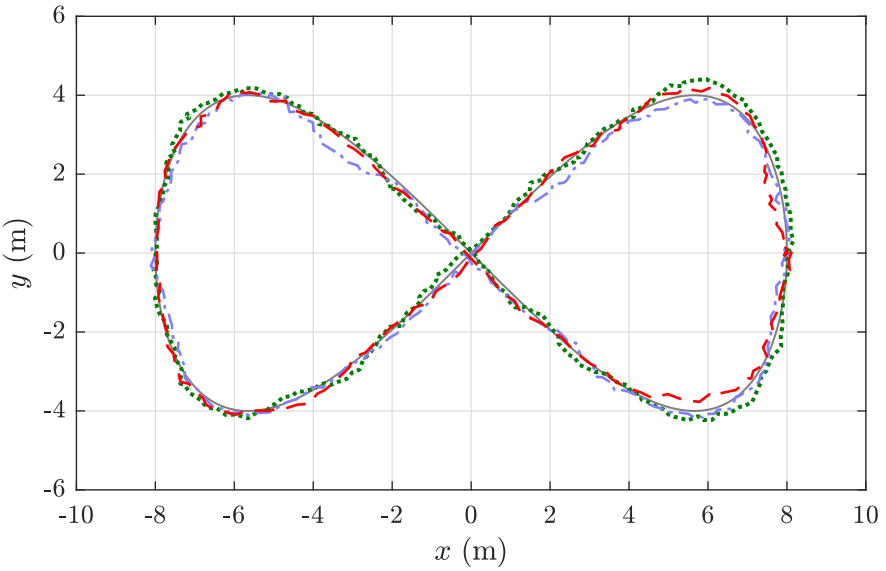

FIGURE 6.14: Trajectories on xy of lemniscate path, simulation with sensor models: *Agent 1* in green (dotted line), *Agent 2* in blue (dash-dotted line) and *Agent 3* (dashed line) in red.

Fig. 6.15 shows the references of *yaw* angle (ψ_{cmd}) and velocity in the x axis (u_{cmd}) computed by the *Agent 3* and the values of the angle ψ and the velocity u in the same simulation.

As observed in the simulation results following the lemniscate path, the *Agent 2* appears to be the one that obtains the best results in terms of cross-track error. However, it is important to recall that this agent was only trained to perform well at the particular velocity of 1 m/s. On the other hand, *Agent 3* achieved a similar performance while reducing considerably the time taken to perform a full lap of the lemniscate. That is, this agent computes the optimal velocity at each part of the path, which allows the vehicle to accelerate in the straight lines, arriving at a maximum velocity of 1.82 m/s. Thus, it was able to increase the average velocity while maintaining almost the same error.

To analyse the performance of the agents while following a different path from the one that was used to train them, a new path was defined. This new path is a spiral, stated in Eq. (6.18). This time, parameter A determined the rate at which the radius of the spiral grows and takes a value of 1.25. The virtual arc (γ) ranges from 0 to 2π. Table 6.4 shows the simulation results obtained while following the spiral path with ground truth measures, while Table 6.5 presents the results of the agents following the same path with sensors measurements.

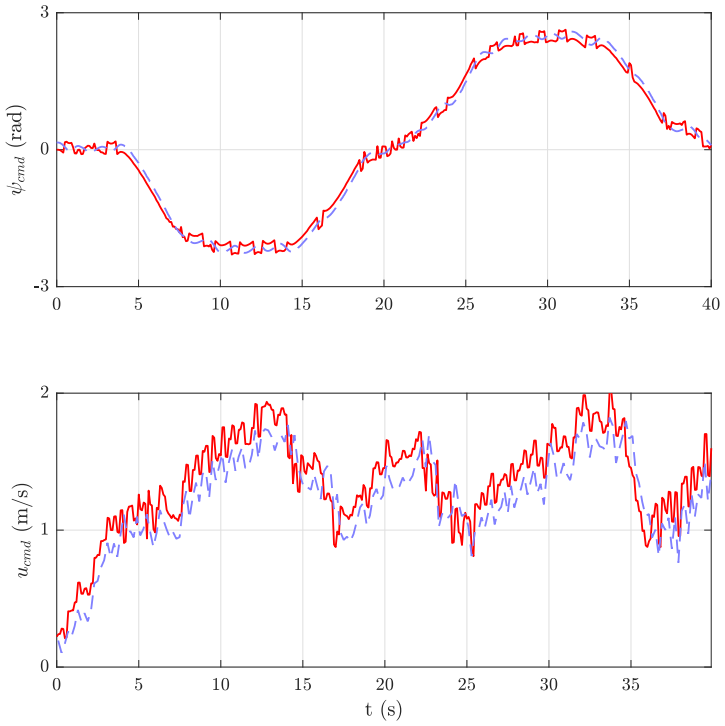

FIGURE 6.15: Actions of *Agent 3* following a lemniscate path, simulations with sensor models: references computed by agent (angle and velocity) in red and real values in blue (dashed line).

$$\begin{aligned} x_d &= -A\gamma\cos(\gamma) \\ y_d &= A\gamma\sin(\gamma) \end{aligned} \tag{6.18}$$

TABLE 6.4: Results for one lap of the spiral path, simulations with ground truth measurements.

	$\overline{\mathbf{d}}$ **(m)**	**time (s)**	$\overline{\|\mathbf{v}\|}$ (m/s)
Agent 1	0.2907	34.30	0.8860
Agent 2	0.1840	32.43	0.8872
Agent 3	0.1418	23.52	1.2119
Agent 3 ($v_{max} = 1$)	0.0759	32.10	0.8530

TABLE 6.5: Results for one lap on the spiral path, simulations with sensor models.

	$\overline{\mathbf{d}}$ **(m)**	**time (s)**	$\overline{\|\mathbf{v}\|}$ (m/s)
Agent 1	0.3035	32.86	0.9448
Agent 2	0.2540	31.18	0.9366
Agent 3	0.1677	22.62	1.2262
Agent 3 ($v_{max} = 1$)	0.0987	30.59	0.8830

The trajectories in the xy plane of the three agents following the spiral path with sensor measures are shown in Fig. 6.16. These results correspond to the simulations presented in Table 6.5. Fig. 6.17 shows the angle and velocity references obtained by the *Agent 3* and their respective real values during that simulation. In that case the vehicle reached a maxim velocity of 1.71 $^{m}/_{s}$.

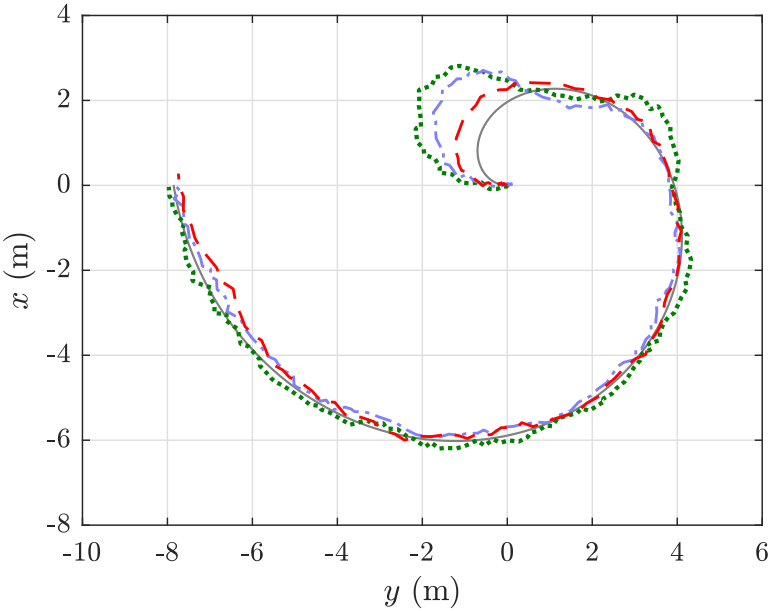

FIGURE 6.16: Trajectories on xy of spiral path, simulations with sensor models: *Agent 1* in green (dotted line), *Agent 2* in blue (dash-dotted line) and *Agent 3* (dashed line) in red.

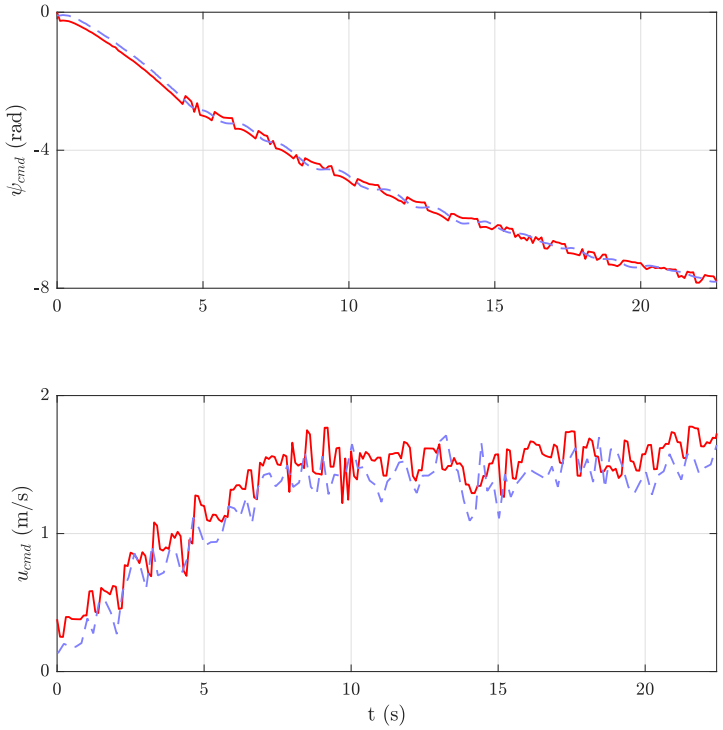

FIGURE 6.17: Actions of *Agent 3* following a spiral path, simulations with sensor models: references computed by agent (angle and velocity) in red and real values in blue (dashed line).

In the simulation results following the spiral path, the performance obtained by each agent varies more than in the results following the lemniscate path. With these results it is clear that, at least in simulation, the *Agent 3* is able to outperform the other agents, reducing the average cross-track error while travelling at higher velocities.

The simulations results show how the agents, even though having been trained with ground truth measurements following a lemniscate path, can also solve the path following problem with sensor measurements and follow other paths such as the stated spiral path. Next, the agents are tested in the real experimental platform.

6.6.2 Experimental

The experimental platform is the one described in Section 3.1, formed by the Asctec Hummingbird vehicle with a supplementary on-board PC (Odroid-XU4) with ROS platform installed. The vehicle is equipped with an IMU sensor which, among other values, provides an estimation of the orientation of the vehicle, with a pressure sensor that estimates the altitude and with a GPS that provides the position and an estimation of the vehicle's velocity on the xy plane. In this work, the states are obtained directly from the measurements provided by the sensors, without the use of any additional filter.

Since tensorflow library is required to operate at 64 bits and Odroid-XU4 works at 32 bits, the *DDPG* python agent cannot run in this PC. Instead, this program is executed in the ground station laptop that communicates through Wi-Fi with the ROS master in the on-board PC. The laptop runs with linux Ubuntu with the i7-8550U intel processor and 16GB RAM. The *DDPG* python3 program runs at 10Hz.

The three agents were tested with real experiments following the same paths as in Section 6.6.1. However, although the three agents were able to solve the path following problem correctly, the results were not as good as expected. That is, the trajectory of the vehicle was slightly oscillating around the path. After various tests the authors concluded that this behaviour was due to a slight discrepancy between the RotorS simulation model and the real dynamics of the vehicle. Namely, the rotational dynamics around the z axis were a little slower in the real vehicle.

In order to improve the performance of the agents two solutions were considered: the first one consists in training the agents in the experimental platform; the second one is to adjust the parameters of the agents to modify their dynamics. Training the agents during real flights can be harmful for the plant due to the unexpected behaviour of the vehicle. Furthermore, it has been observed that training with noisy measurements reduces the learning effectiveness. On the other hand, apparently, it does not exist any methodology for modifying the dynamics of the agents by changing some of their parameters. Indeed, out of the set of design and training parameters, involved only in the training phase, the *DDPG* algorithm does not have any other parameter to tune. However, in this chapter we propose a form of modifying the control dynamics of the agent by adding a new parameter that will scale the output of the agent. That is, since the outputs of the agent are corrections (angle and velocity corrections), this parameter will regulate the

speed at which the correction is made, and thus, it ends up regulating the dynamics of the angle and/or velocity reference too.

Since the discrepancy between the two models affects in the *yaw* dynamics, only the angle action was scaled with the mentioned parameter. This new parameter, known as the angle correction constant (k_a), is apparent in Eq. (6.19) and it is set experimentally, $k_a = 2$; the value that provided the best performance from the different values that were tested. This correction constant was included in the three agents that were used to obtain all the experimental results that are presented in this chapter. It is important to remember that this constant was just used in the experimental phase to improve the performance of the agents.

$$\psi_{cmd,k} = k_a a_k \Delta t + \psi_k \tag{6.19}$$

Next, the agents were tested with the same lemniscate of Section 6.6.1 (Eq. 6.16) with $A = 4$ and γ ranging from 0 to 4π. The results of the three agents performing a full lap of this path are shown in Table 6.6. The results of the *Agent 3* with the maximum velocity limited to 1 $^m/_s$ are also included. Again, the table shows the average cross-track error ($\overline{d}$), the total time and the average velocity of the vehicle ($\overline{\|\mathbf{v}\|}$). Fig. 6.18 presents the trajectory on the xy plane of the agents while following this path. Furthermore, Fig. 6.19 shows the angle and velocity references computed by the *Agent 3* while following this lemniscate path, where references are shown in red and real values in blue dashed lines.

TABLE 6.6: Experimental results for one lap on the lemniscate path.

	$\overline{\mathbf{d}}$ (m)	time (s)	$\overline{\|\mathbf{v}\|}$ ($^m/_s$)
Agent 1	0.1739	55.90	0.8859
Agent 2	0.1140	55.01	0.9141
Agent 3	0.1682	39.39	1.6311
Agent 3 ($v_{max} = 1$)	0.2275	59.50	0.8829

The experimental results following the lemniscate reveal a behaviour that is very similar to the simulation results. The *Agent 2* displays again the best performance in terms of cross-track error, but the *Agent 3* achieves a similar distance error while increasing the average velocity, arriving at a maximum velocity of 2.49 $^m/_s$.

Another important remark from these experimental results is found in the last curve of the trajectory performed by the *Agent 3* (bottom-right curve in Fig. 6.18), a curve that the agent should clearly undertake better. This behaviour in the fast counter-clockwise curves appeared in all the experiments that we performed with this agent, and was even more evident in the counter-clockwise spiral paths. The cause of that strange pattern was found in the designed training framework. Although the agent was trained with asymmetrical lemniscates with diverse radius, this training framework resulted to be incomplete. The reason is that the agent was trained with a half lemniscate beginning with a counter-clockwise curve and ending in a clockwise curve. This

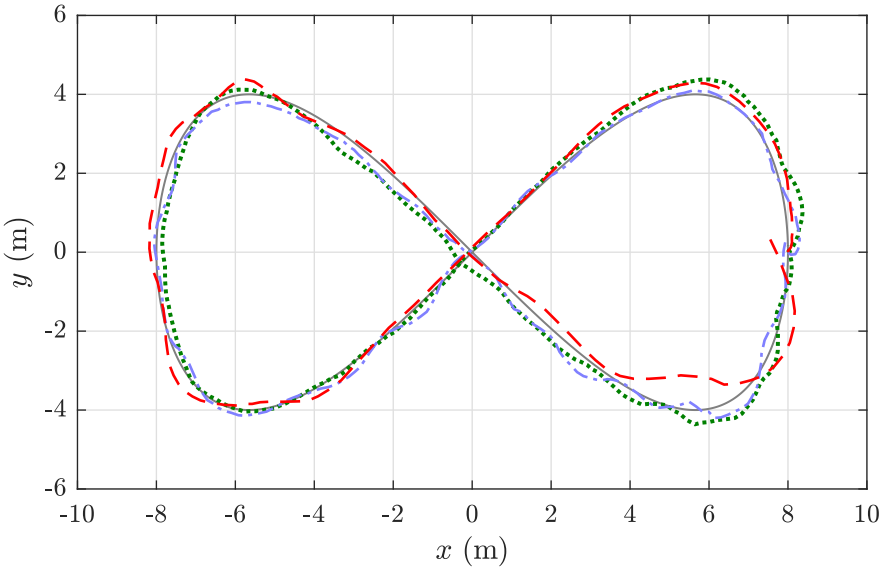

FIGURE 6.18: Trajectories on xy of lemniscate path, experimental results: *Agent 1* in green (dotted line), *Agent 2* in blue (dash-dotted line) and *Agent 3* (dashed line) in red.

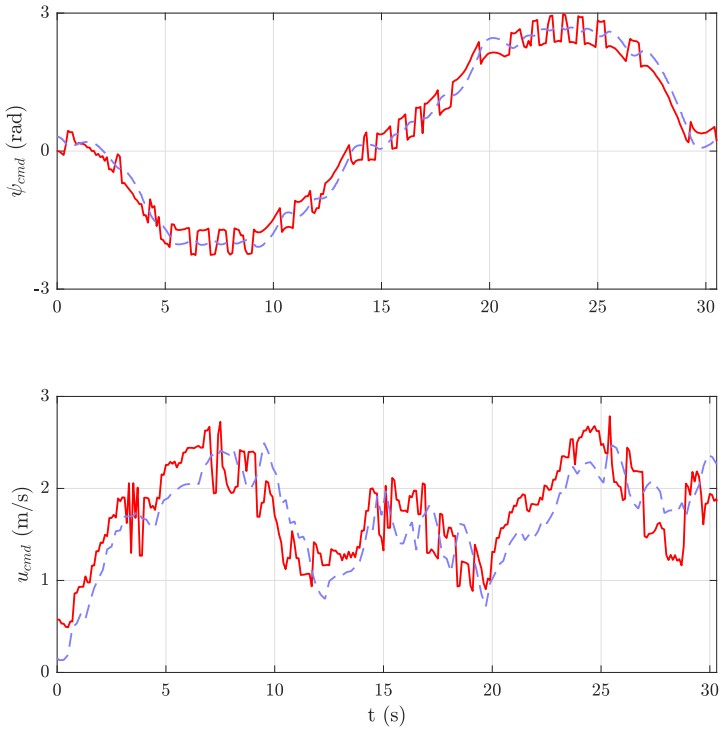

FIGURE 6.19: Actions of *Agent 3* following a lemniscate path, experimental results: references computed by agent (angle and velocity) in red and real values in blue (dashed line).

way, the first curve is always slower than the second one and the agent learned to perform fast clockwise curves and slow counter-clockwise curves. Consequently, it resulted in a bad behaviour while following counter-clockwise curves at high velocities. The solution to address this problem consists in changing the training framework of this 3^{rd} agent to have both types of curves at slow and fast velocities. It could be done, for instance, with full asymmetrical lemniscates.

Finally, the agents were tested with the spiral path defined in Eq. (6.18), with $A = 2.5$ and γ ranging from 0 to 4π. Table 6.7 presents the results of the three agents plus the *Agent 3* with limited velocity, just as in Table 6.6. The trajectories of the agents following this spiral path are shown in Fig. 6.20, and Fig. 6.21 shows the angle and velocity references computed from the actions of the *Agent 3*.

TABLE 6.7: Experimental results for one lap on the spiral path.

	$\overline{\mathbf{d}}$ (m)	time (s)	$\overline{\|\mathbf{v}\|}$ ($^m/_s$)
Agent 1	0.2848	28.81	1.0503
Agent 2	0.2503	27.79	1.0460
Agent 3	0.2257	18.59	1.5844
Agent 3 ($v_{max} = 1$)	0.2342	33.10	0.8826

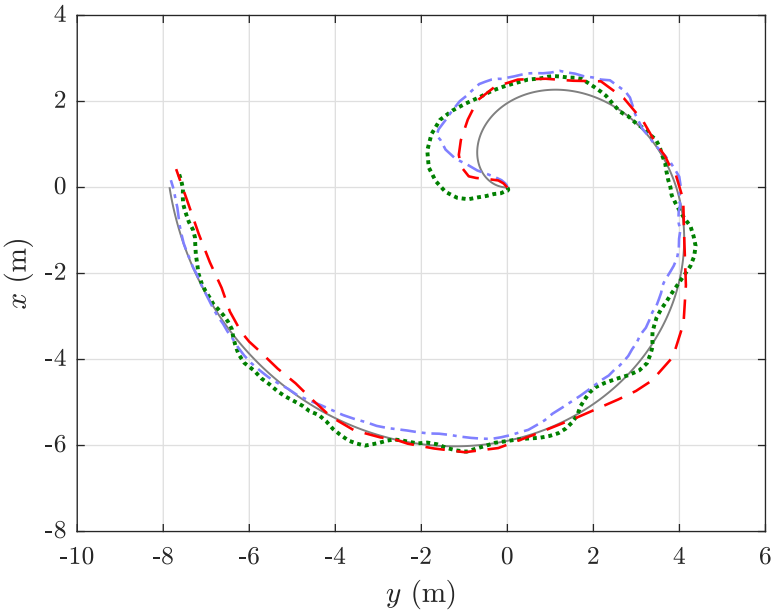

FIGURE 6.20: Trajectories on xy of spiral path, experimental results: *DDPG v1* in green (dotted line), *DDPG v2* in blue (dash-dotted line) and *Agent 3* (dashed line) in red.

In the experimental results following the spiral path, the *Agent 3* outperforms the other agents by exhibiting a lower cross-track error with higher velocity. Specifically, this agent arrives at a maximum velocity of 2.52 $^m/_s$.

The initial trajectory of the agents when following the spiral path (Fig. 6.20) evidences a difference of behaviour between each of the three approaches presented in this chapter. That is, the *Agent 1* is required to travel at a constant speed of 1 $^m/_s$ and only has information of the instantaneous distance and orientation error. Thus, due to the lack of anticipation, in the initial part of the path it starts going forward, moving out of the path. The *Agent 2* moves also at a constant velocity. However, this agent has information about the upcoming orientation of the path, which allows it to anticipate the curve. Hence, in the initial part of the path, this

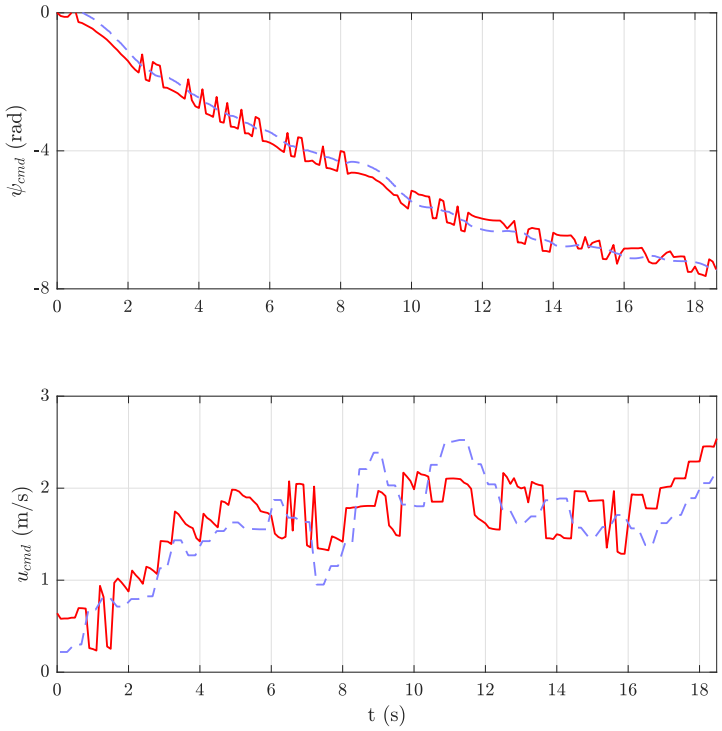

FIGURE 6.21: Actions of *Agent 3* following a spiral path, experimental results: references computed by agent (angle and velocity) in red and real values in blue (dashed line).

agent starts moving towards the curve. Finally, the *Agent 3* knows the evolution of the path's curvature in advance and it is able to modify the longitudinal speed. That allows it to command slower speeds at the beginning of the path and turn towards the curve to follow the path as accurately as possible and then, when it is correctly oriented, start increasing the velocity.

Part III

Navigation and Guidance

Chapter 7

Obstacle Detection

To perform fully autonomous flights in unknown or uncontrolled environments it is necessary to develop an anti-collision system, specially in low-altitude, rapid manoeuvring or indoor flights. A knowledge of the obstacles' localization (relative or absolute) is required to carry out this task. The obstacle avoidance algorithm developed in this thesis implements a reactive avoidance approach. Hence, the obstacle detection system must detect obstacles in the environment in real-time during the flight.

This chapter presents the solution adopted in this thesis to solve the obstacle detection problem. The selection of the perception sensor is discussed. The main characteristics of this sensor are introduced. The sensor is modelled in the RotorS/ROS framework. Finally, an approach to treat the sensor measures to eliminate ground detections is developed.

It is important to recall that the objective of this chapter is not to develop an obstacle detection algorithm itself, but to provide a simple and efficient solution to supply the obstacle's information to the obstacle avoidance algorithm. The obstacle avoidance problem is solved by implementing a deep reinforcement learning approach (Chap. 8). Therefore, the treatment of the sensor measurements to localize the obstacle is mainly conducted by the DRL algorithm.

7.1 Selection of the Sensor

The obstacle avoidance system relies on the on-board sensors to perceive the environment. Thus, it is important to select the proper sensor (or set of sensors) according to the problem requirements. First, it is necessary to define if the information of the environment must be relative or absolute. Relative information means that the localization of the obstacle is made relative to the UAV (i.e. distance from the vehicle to the obstacle). In the global or absolute localization approaches, the position of obstacles is given with respect to the world frame. These approaches are mainly used in SLAM solutions, where the obstacle avoidance task is made by mapping the environment, such as with an occupancy grid map. In this work, the objective of the obstacle

avoidance task is to perform a fast reactive manoeuvre to avoid the obstacles. Thus, it is necessary to find an obstacle detection solution that provides relative information of the environment at fast rates.

The obstacle avoidance sensor selection must consider several important aspects, such as the real-time measurement capability, the operational environment and the payload constraints [96]. The operation environment includes the scan capabilities, the field-of-view (FOV) and the range of the sensor. The payload constrains for small UAV are the weight, volume and power of the sensor.

In Section 2.3.2, a review of UAV perception focused on the obstacle detection problem is presented. From that literature review it is possible to observe that cameras and LIDARs are the most common sensors for the obstacle detection, since they achieve best results. Even so, there are other approaches using different sensors such as RADARs or acoustic sensors. However, RADAR-based solutions are not feasible for the Asctec Hummingbird platform due to their heavy weight. On the other hand, acoustic sensors are constrained to low distances and slow velocities, thus, they do not accomplish the requirements.

Different obstacle detection solutions based on the use of cameras (i.e. monocular vision, stereo vision, optical flow) and LIDARs (i.e. 3D, 2D, frontal) are explored in Section 2.3.2. Although images contain rich information about the environment, they suffer from many issues related to high computational requirements and high sensitivity to the environment, such as light and textures [88]. On the other hand, LIDARs can provide accurate information about the environment with lower computational requirements. However, its weight and power consumption can limit their use to small UAVs.

Presumably, the on-board PC of our experimental platform (Section 3.1) does not have enough capacity to deal with the high computational requirements of vision-based systems, especially, taking into account the requirement of providing information at fast rates to the reactive avoidance algorithm. Thus, it would be necessary to incorporate a new on-board PC to the system, which would increase the total weight of the UAV. Therefore, the advantage of low-weight that cameras present, wouldn't be an important factor in this case. Furthermore, considering that the objective of this chapter is to provide a simple and efficient solution, vision-based algorithms are not particularly simple. For these reasons, the LIDAR sensors were considered to be the best option for the specific problem presented here. That is, LIDAR-based obstacle detection approaches provide simple solutions with good results.

There exist diverse types of LIDAR sensors in the marked. Among them, the 3D LIDARs stand out for their capacity of providing a detailed information of the environment, which can facilitate the task of obstacle avoidance. Also, they are commonly used for mapping. Due to the large field-of-view, these sensors are able to generate a point cloud map of the environment in the three-dimensional space. A well-known example of this type of sensor is the Velodyne Puck VLP-16 (Fig. 7.1a). The main problem of these sensors are their weight and their power consumption, which make it impossible to include them in our experimental platform. Specifically, Velodyne VLP-16 weights 830g, which is more than the total weight of our Hummingbird platform.

(A) Velodyne Puck VLP-16

(B) RPLIDAR A3.

(C) Scanse Sweep LIDAR.

(D) LIDAR-Lite V3.

(E) LeddarOne.

(F) Leddar M16.

FIGURE 7.1: Examples of LIDAR sensors for UAVs (images obtained from the company's website of each sensor).

Other LIDAR solutions are the 2D sensors, which are commonly used in unmanned ground vehicles. These sensors usually provide a 360° of vision in a 2D plane. This field-of-view is ideal for the UGV. However, it can be incomplete for UAV, since the FOV relies on the orientation of the vehicle (pitch and roll angles), which is affected by its velocity. Thus, if the sensor is placed horizontally on the vehicle, its capability to detect obstacles would decrease at high velocities. Moreover, these type of sensors usually have a rotational mechanism that can create disturbances to the sensor measurements (e.g. the accelerometers and gyroscopes) due to the generated vibrations. What is more, the vibrations generated by the UAV can also affect the rotational mechanism of the LIDAR, and thus, it may compromise the quality their measurements. Examples of this sensors are the RPLIDAR A3 (Fig. 7.1b) or the Scanse Sweep (Fig. 7.1c).

Frontal LIDARs are much lighter than 3D models and do not suffer from the vibration problems of the 2D ones because they do not have rotational components. For this reason, this type of LIDARs become a great solution for our problem. In the market there exist several and diverse frontal LIDARs, such as the LIDAR-Lite v3 (Fig. 7.1d), the LeddarOne (Fig. 7.1e), the Leddar VU8 or the Leddar M16 (Fig. 7.1f). Among these sensors, the Leddar VU8 was selected to be implemented as the obstacle detection sensor. This is mainly because it accomplishes the weight, dimensions and power supply requirements, and because it has an appropriate field-of-view (48°) and range (85m). Moreover, one of its main advantages is that it can be easily integrated in ROS with the package provided by the manufacturer.

Table 7.1 summarizes the characteristics, according to the manufacturers, of the most common LIDAR sensors for UAVs in the market.

TABLE 7.1: Comparison of the most common LIDAR sensors for UAVs.

	Range	**Horitzontal FOV**	**Vertical FOV**	**Weight**
3D LIDARs				
Velodyne VLP-16	100 m	360°	30°	830 g
Velodyne HDL-32E	100 m	360°	40°	1.4 kg
Ouster OS1-32	120 m	360°	45°	455 g
2D LIDARs				
RPLIDAR A3	25 m	360°	-	190 g
Scanse Sweep	40 m	360°	-	120 g
Hokuyo UST-20LX	20 m	270°	-	130 g
Frontal LIDARs				
LeddarOne	40 m	3°	3°	14 g
Leddar VU8	85 m	48°	3°	107 g
Leddar M16R-75J0011	91 m	48°	5.5°	162 g
Leddar M16D-75B0005	55 m	48°	6°	175 g

It is important to mention that sensors with larger field-of-view than the Leddar VU8 would facilitate the task of the reactive avoidance DRL agent. That is because, with the Leddar VU8, when the UAV sees an obstacle and it starts avoiding it, at some point it will lose the obstacle from its field-of-view. This may lead the agent to assume, mistakenly, that the UAV has already avoided the obstacle and drive it back to the path provoking a collision. This problem is discussed in Chap. (8) and a solution is proposed. Nonetheless, it was impossible to find a LIDAR with larger field-of-view that accomplished the weight, dimensions, power supply, range and precision requirements (Leddar VU8 has a version with 99° of FOV, but only procures a range of 6m with grey targets).

7.2 Leddar VU8 LIDAR

The sensor used to perform the obstacle detection task is the Leddar VU8 LIDAR (Fig. 7.2). There exist several models of this sensor with various field-of-view configurations that provide different range capabilities. The model selected for our approach is the configuration that provides medium FOV with a medium/high detection range. The characteristics of this sensor are detailed in Table 7.2. It needs a power supply of 12V, which corresponds to the voltage of the batteries used on-board the vehicle. As can be observed, the detection range of the sensor depends on the type of obstacle that it is detecting.

The Leddar VU8 divides its field-of-view in 8 horizontal segments, as it is shown in Fig. 7.3. That is, this sensor provides a vector of 8 values representing the minimum distance to an obstacle in that segment. Each segment has a FOV of 6°x3°. Distances are given with a precision of 1cm.

FIGURE 7.2: Leddar VU8 (original image from leddartech.com).

TABLE 7.2: Characteristics of the Leddar VU8 Medium FOV.

Description	**Value**
Weight.	107.6 g
Dimensions.	70.0 x 35.9 x 49.6 mm
Power supply.	12 V
Horizontal FOV.	48°
Vertical FOV.	3°
Range (Retro-reflector).	85 m
Range (white target).	19 m
Range (grey target).	13 m
Accuracy.	5 cm
Distance precision.	1 cm
Refresh rate.	100 Hz

It may seem that 3° is a poor horizontal FOV, being similar to the scan capability of a 2D LIDAR. However, note that 3° means that, at a distance of $10m$, it covers an arc of $52cm$ on the horizontal axis.

This sensor can be straightforwardly installed and includes a graphic interface that helps to configure the sensor and communication parameters as well as the visualization of the LIDAR data. This interface is shown in Fig. 7.4. Furthermore, the manufacturer provides a ROS package that transforms the data received by the serial port to a ROS topic. Fig. 7.5 shows the data generated by this ROS topic represented in RVIZ, a 3D visualization tool of the ROS framework. This figure shows the points detected by each of the beams of the sensor while detecting a cylindrical obstacle. The obstacle and the quadrotor vehicle were added to the figure to help its interpretation.

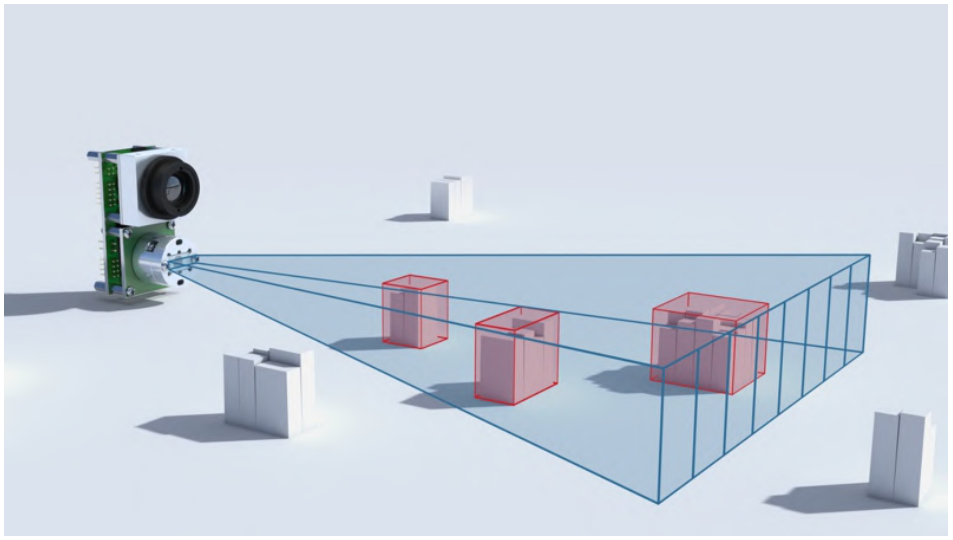

FIGURE 7.3: Leddar VU8 detects obstacles in 8 segments (original image from leddartech.com).

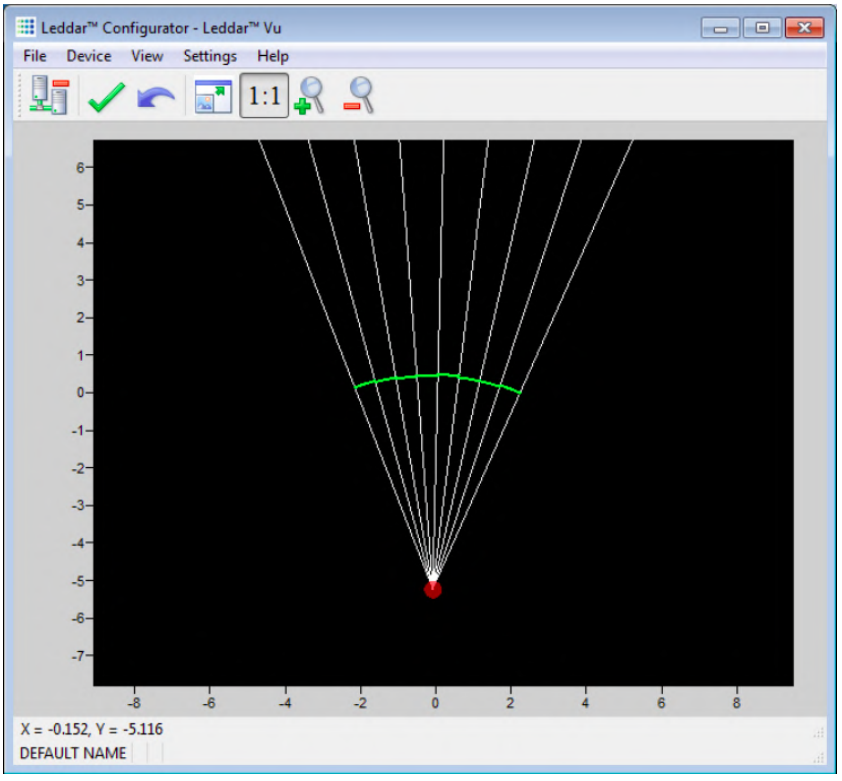

FIGURE 7.4: Leddar VU8 data represented in RVIZ.

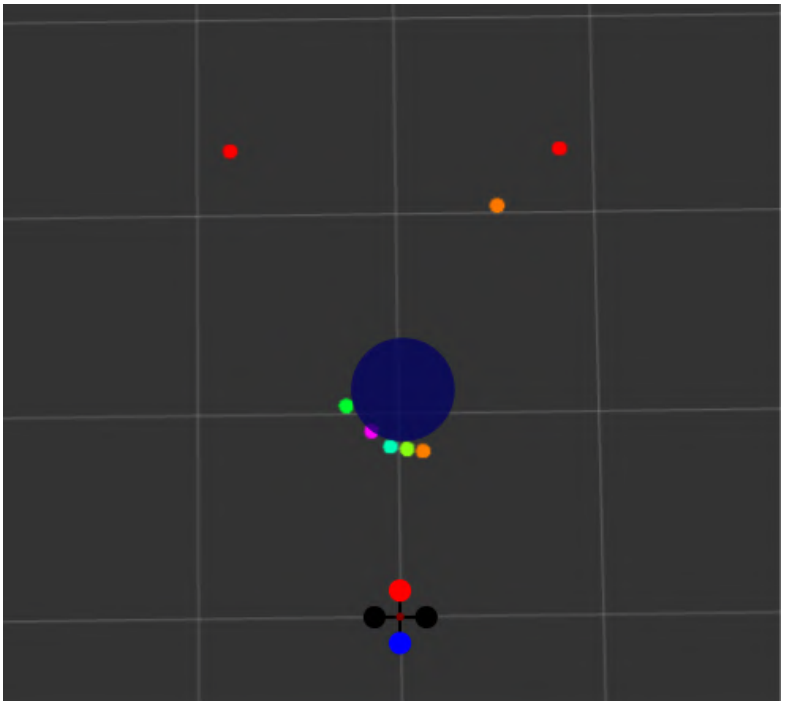

FIGURE 7.5: Leddar VU8 data represented in RVIZ.

7.3 Modelling the Sensor in Gazebo/ROS

In order to obtain a robust obstacle avoidance approach with a good experimental performance, it is necessary to train the deep reinforcement learning agent in a realistic and complete model of the environment. This includes modelling the vehicle and its sensors. To this end, the Leddar VU8 sensor was modelled in the RotorS platform (Section 3.5), the realistic UAV simulation platform where the DRL agents are trained.

The process of modelling a sensor in RotorS/Gazebo is divided in two parts; an URDF model that creates the visual part and mechanical constants of the sensor (i.e. masses, dimensions, inertias) and a ROS plugin that generates the ROS topic of the sensor.

URDF (Unified Robot Description Format) is a standardized format for representing a robot model. These models are defined in XML scripts. Since the visual part does not play a key role in the functionality of the sensor, the LeddarVU8 was modelled as an object of simple two-cylinder shape, similarly to a 3D LIDAR sensor. The sensor is attached to the top of the vehicle, centred, and aligned with the x axis of the body frame. The real mass of the sensor and an estimation of its inertia on each axis were introduced to the model. Then, the eight beams of the sensor, corresponding to each of the LIDAR's segments, were included into the model. Each beam points to the center of the respective segment. These beams are used to calculate the distance from the sensor to an obstacle in each segment direction. These distances are given with a precision of $1cm$ and are limited to a range of $13m$, corresponding to the maximum distance range with grey targets (Table 7.2). A gaussian noise with a standard deviation of 0.01 was included in each of the obtained distances to resemble the values provided by the real sensor.

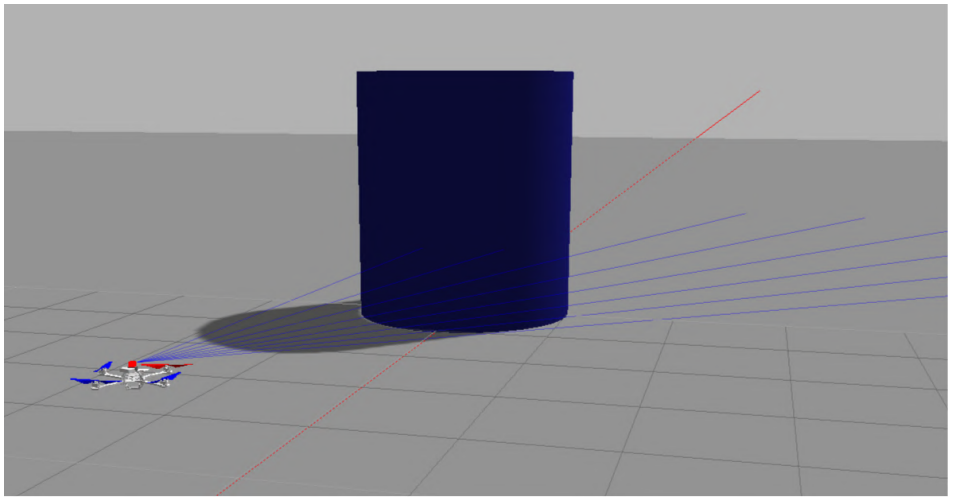

FIGURE 7.6: Capture of Gazebo including the Hummingbird quadrotor with the LIDAR sensor and an obstacle.

Fig. 7.6 shows a capture of the Gazebo framework including the Asctec Hummingbird vehicle with a cylindrical obstacle. The simulated Leddar VU8 sensor estimates the distance of the obstacle in each of the 8 beams directions. These beams are presented in blue. The red line of Fig. 7.6 represents the x world axis.

The ROS plugin extracts from the Gazebo framework the set of distances computed by the modelled sensor. With these distances it generates a ROS topic with the same structure and type of message of the real Leddar VU8 sensor. This includes a time stamp, the vector with the eight distances, and other information such as the maximum range or the minimum and maximum scan angles.

7.4 Eliminating Ground Detections

The LIDAR sensor is used to detect obstacles, but it can also detect the ground. Eliminating ground detections from the measured distances is very helpful for the Obstacle Avoidance algorithm in order to interpret the data in a correct way.

To eliminate ground detections, first it is necessary to estimate the theoretical distance from the LIDAR to the ground in each beam direction. Then, the estimated ground distances are used to define a threshold for each beam. Finally, any distance provided by the sensor being larger than the calculated threshold is eliminated.

Two approaches to eliminate ground detections are provided in this section. First, an approach that only considers the ground detections caused by the pitch angle. Second, an approach that considers both the pitch and roll angles to estimate the ground detections on each LIDAR beam.

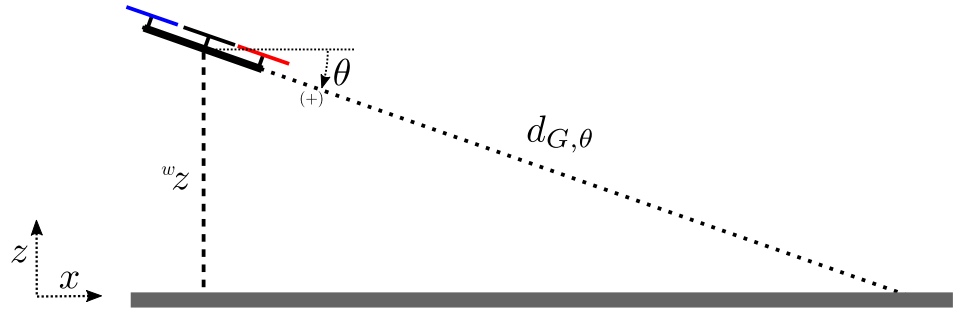

FIGURE 7.7: Distance from LIDAR to ground depending on pitch angle.

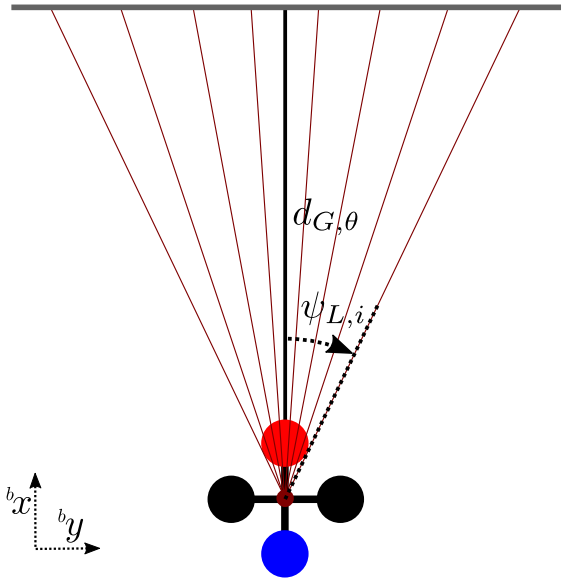

FIGURE 7.8: LIDAR ground detections generated by pitch angle: xy-body plane.

Detections Caused by Pitch Angle

This approach considers that the roll angle is around zero and pitch angle is positive. Fig. 7.7 shows how the pitch angle (θ) can cause the ground detections, where $d_{G,\theta}$ is the distance from the LIDAR to the ground in the x-body axis. This distance is calculated in Eq. (7.1). Since our sensor is a frontal LIDAR, the ground detections are only possible with positive values of θ.

$$d_{G,\theta} = \frac{^{w}z}{\sin(\theta)} \tag{7.1}$$

Fig. 7.8 shows the xy-body plane of the scenario presented in Fig. 7.7, where $\psi_{L,i}$ is the angle of the i-th beam of the LIDAR with respect to the x-body axis. Leddar VU8 beams are represented in dark red and the detected ground with a grey line. The estimated ground distance measured by the i-th beam by only considering the pitch angle ($d_{G,\theta,i}$) is calculated in Eq. (7.2).

$$d_{G,\theta,i} = d_{G,\theta}\frac{1}{\cos(\psi_{L,i})} = \frac{^{w}z}{\sin(\theta)}\frac{1}{\cos(\psi_{L,i})} \tag{7.2}$$

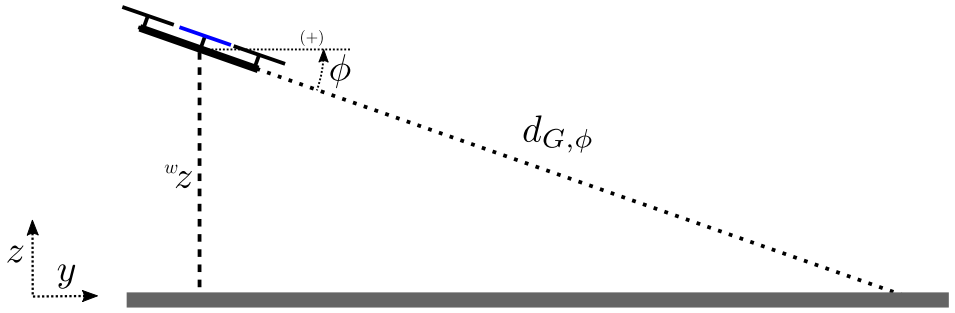

FIGURE 7.9: Distance from LIDAR to ground depending on roll angle.

The condition used to eliminate the ground detections from the data measurements is shown in Eq. (7.3). That is, if the measured distance of the LIDAR beam ($d_{L,i}$) is equal or larger than the estimated distance to the ground, the measure is set to infinity (i.e. no obstacle detected). Due to noise or inaccuracies of the sensor, the measured distance to the ground can be slightly different to the estimated one. For this reason, a margin m, set to 0.2, is applied to the estimated ground distances.

$$\text{if} \quad d_{L,i} \geq d_{G,\theta,i}(1-m) \quad \text{then} \quad d_{L,i} = \infty \tag{7.3}$$

Detections Caused by the Combination of Pitch and Roll Angles

This approach considers both pitch and roll angles to estimate the distance to the ground measured by each LIDAR beam. First, let us calculate the distance to the ground in the y-body axis, $d_{G,\phi}$, when roll is different to zero (Fig. 7.9), which can be computed with Eq. (7.4). With this distance and the distance to the ground on the x-body axis when pitch is positive ($d_{G,\theta}$) it is possible to define the ground line detected by the LIDAR. This line is represented in grey in Fig. 7.11, which shows the xy-body plane of the vehicle when pitch and roll angles are different from zero. Again, LIDAR beams are represented in dark red.

$$d_{G,\phi} = \frac{^{w}z}{\sin(\phi)} \tag{7.4}$$

The problem of obtaining the distance from the LIDAR to the ground measured by each beam can be addressed as a trigonometrical problem, as it is represented in Fig. 7.11, which is a simplification of Fig. 7.10 with only one beam. α angle can be calculated by Eq. (7.5), β angle is obtained by Eq. (7.6) and the height of the triangle h is computed by Eq. (7.7). The LIDAR-ground distance of the i-th beam ($d_{G,i}$), obtained by trigonometrical relations, is calculated with Eq. (7.8). This equation is only valid if $|\beta| < \pi/2$, since otherwise it means that the beam does not detect the ground.

$$\alpha = \arctan\left(\frac{d_{G,\phi}}{d_{G,\theta}}\right) = \arctan\left(\frac{\sin(\theta)}{\sin(\phi)}\right) \tag{7.5}$$

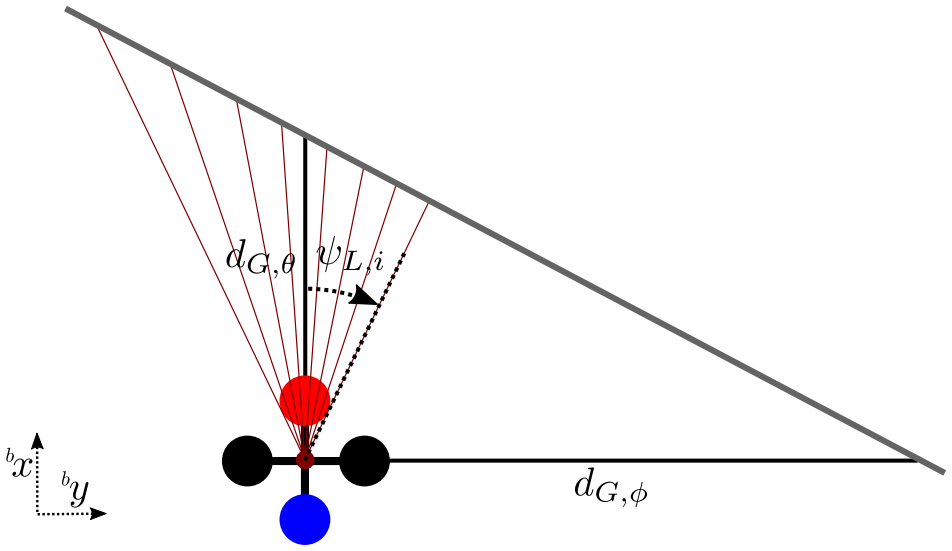

FIGURE 7.10: LIDAR ground detections generated by pitch and roll angles: xy-body plane.

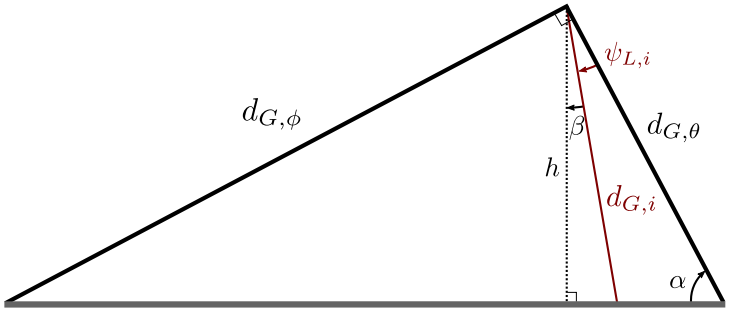

FIGURE 7.11: Trigonometrical problem to obtain ground detections ($d_{G,i}$).

$$\beta = \pi/2 - \alpha - \psi_{L,i} \tag{7.6}$$

$$h = d_{G,\theta} \sin(\alpha) \tag{7.7}$$

$$d_{G,i} = \frac{h}{\cos(\beta)} = \frac{{}^{w}z \sin\left(\arctan\left(\frac{\sin(\theta)}{\sin(\phi)}\right)\right)}{\cos(\theta)\cos\left(\pi/2 - \arctan\left(\frac{\sin(\theta)}{\sin(\phi)}\right) - \psi_{L,i}\right)} \tag{7.8}$$

Once obtained the estimated distance to the ground, the condition to eliminate ground detections when pitch is positive and roll is different to zero is represented in Eq. (7.9). Once again, due to sensor noise and inaccuracies, a margin m is used and set to 0.2.

$$\text{if} \quad d_{L,i} \geq d_{G,i}(1-m) \quad \text{then} \quad d_{L,i} = \infty \tag{7.9}$$

Depending on the values of pitch and roll angles, the elimination of ground detections is made by using condition Eq. (7.3) (when $\theta > 0$ and $\phi \simeq 0$) or by condition Eq. (7.9) (when $\theta > 0$ and $\phi \neq 0$). If pitch is not positive, ground detection algorithm is not applied.

Chapter 8

Reactive Obstacle Avoidance with Deep Reinforcement Learning

Recent years are revealing an exponential growth on the research and applications on the deep reinforcement learning field. DRL has been applied to a large number of different computer science, engineering and control problems with outstanding results. In Chap. (6) a DRL algorithm was implemented to solve the path following problem with an adaptive velocity selection approach obtaining successful experimental results. In this chapter, the capabilities of this research field are exploited by implementing a deep reinforcement learning approach for solving the reactive obstacle avoidance problem.

Most of the UAV applications used to solve the obstacle avoidance (OA) problem by means of machine learning theory are based on the use of *Convolutional Neural Networks* (CNN). That is, a type of deep neural network where the inputs may be images. In [109] *CNN*s are used to extract the information of an obstacle from the images of a monocular vision system. Then, this information is used as the state input of an actor-critic RL algorithm that designs the trajectory to avoid the obstacle. In other cases, the obstacle avoidance algorithms based on DRL implement the *Deep Q-Network* (DQN) algorithm [50][192] or the *Double DQN* algorithm [186][191][64]. These algorithms are variants of *Q-Learning* that use *CNN*s. They receive raw images of a camera (or cameras) as the reinforcement learning state. Applications of the *DQN* algorithm for the OA problem are also found in other systems such as marine vessels [36] and mobile robots [106].

Other UAV approaches implementing the *Deep Deterministic Policy Gradient* (DDPG) algorithm are also found in the literature. In [99] a *DDPG*-based approach for a fixed-wing UAV that performs ground target tracking while avoiding obstacles is presented. This approach uses *Long Short-Term Memory* (LSTM) neural networks to approximate the state of the environment from the obstacle detection distance measures. The *LSTM* networks are a type of recurrent neural networks (RNN) that use a sequence as a data input. That is, the training results at each step are determined by both the current training data and the historical training data. The obstacle detection sensor used in this paper has a field-of-view of 180°. The proposed approach

is trained and validated in a simulated environment. In [181] a modification of the *Recurrent Deterministic Policy Gradient* (RDPG) algorithm, a *DDPG*-like algorithm specially designed for partially observable Markov decision processes, is used to solve the UAV obstacle avoidance problem. The *DDPG* algorithm updates the weights of the networks step-by-step making use of the newest experience. Rather than that, the *RDPG* only updates the weights when an episode ends, using the entire episode experience. Thus, unlike *DDPG* algorithm, *RDPG* is not sample-efficient. In thas paper a Fast-*RDPG* algorithm is designed in such a way that it permits learning online by updating its weights in terms of history trajectories instead of the entire episode. The proposed algorithm uses the distance measures of a range sensor with a 180° of field-of-view. The proposed framework is validated through simulation results. A LIDAR-based approach for multirotor navigation based on the *DDPG* algorithm is presented in [153]. The objective of this work is to develop an agent capable of driving the vehicle to a goal position avoiding obstacles in the vehicle's route. The proposed approach uses the measures of a LIDAR with a FOV of 270° as part of the environment state. The reward is designed using an Artificial Potential Field. The agent is trained in RotorS and is validated with real experimental results.

In this chapter the *DDPG* algorithm is implemented to solve the reactive obstacle avoidance problem by using the LIDAR measures as states of the agent (Section 7.2). Since the obstacle detection sensor has a narrow field-of-view, an approach for providing the agent a certain memory of the previously seen obstacles is developed. An OA training environment is developed in the RotorS/Gazebo framework (Section 7.3). The resulting agents are trained and tested in the RotorS simulated environment following a path while dealing with static obstacles.

8.1 Problem Statement

The aim of this chapter is to develop a reactive obstacle avoidance approach by means of deep reinforcement learning theory. The resultant system must be able to follow a predefined generic path with an adaptive velocity selection approach. That is, maintaining the same functionality of the DRL path following agent (*Agent 3*) presented in Chap. (6). Furthermore, the system must be able to avoid static obstacles that may appear in the vehicle's route. To do so, the agent must use only the local information of the environment provided by the obstacle detection sensor. The obstacle detection sensor is the frontal LIDAR described in detail in Chap. (7). The measures of the LIDAR are treated to eliminate ground detections, as explained in Section 7.4.

Reactive obstacle avoidance is understood as a path planning algorithm that only considers local information of the vehicle's environment to design a reactive trajectory to avoid obstacles in the vehicle's route. That is, it plans online a trajectory that prevents collisions in the last minute.

The selected algorithm to solve the reactive OA problem is the *Deep Deterministic Policy Gradient*, the same algorithm that was implemented in Chap. (6) to solve the path following problem. Details of this deep reinforcement learning algorithm are given in Section 6.2.

The proposed structure to solve the described problem relies on maintaining the PF agent developed in Chap. (6) and create a new agent that is in charge of providing the reference path to the

PF controller in such a way that the main functionalities of the PF agent are maintained while being able to avoid static obstacles. The path following and obstacle avoidance problem could be also solved by a unique agent. However, considering the amount of training episodes needed to learn to follow a path selecting the optimal vehicle's velocity depending on the path radius, and considering the difficulty of the reactive obstacle avoidance problem itself, the number of training episodes and the size of the network structures could increase exponentially. This is the main reason to divide the solution into two agents in this work.

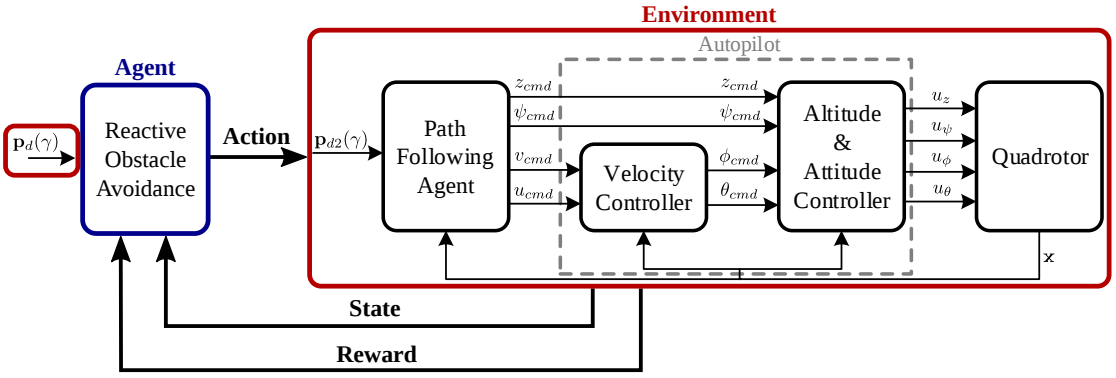

FIGURE 8.1: Reinforcement learning structure employed in this work.

The main elements of the reinforcement learning structure employed in this work are shown in Fig. 8.1. In this case, the agent is the OA algorithm, and the environment includes the rest of elements that interact with the agent. That is, the path following agent, the autopilot, the quadrotor, the reference path and the quadrotor's environment. Note that the OA agent receives the original reference path and is in charge of computing the reference path that is commanded to the path following agent, $\boldsymbol{p}_{d2}(\gamma)$. This path contains the needed trajectories to avoid any obstacle on the vehicle's route, and must be equal to the original reference path when there are no obstacles.

8.2 Agent Environment

The quadrotor of the agent's environment is the Asctec Hummingbird (see Chap. (3)). However, just as in Chap. (6), the agent is trained in a simulated environment with the purpose of maintaining the integrity of the experimental platform. Details of the training environment of the reactive obstacle avoidance agent are given next.

8.2.1 Training Environment

RotorS simulator is used again as the training environment for the agent. However, to perform obstacle avoidance experiments it is necessary to include new elements in this simulator. The LIDAR sensor of the real platform, the Leddar VU8 (Section 7.2), was modelled in the RotorS

simulation environment. This model includes the visual part, the dynamics part and the generated ROS topic. Gaussian noise on the LIDAR measures was included. The modelling process of this sensor is described in Section 7.3.

To perform OA experiments, a key element is necessary; the obstacle/s. In this work, obstacles are modelled as cylindrical objects. Just as the LIDAR sensor, modelling the obstacle involves the generation of an URDF model that defines the visual part and the dynamics of the object. The generated obstacle has a radius of 1m and an altitude of 2m. Fig. 7.6 shows a screen capture of the gazebo framework with the generated obstacle and the modelled LIDAR sensor.

The reactive obstacle avoidance agent was programmed in Python 3.5, and it communicates with the ROS framework by means of the *rospy* library. Tensorflow and tflearn libraries were used to generate the neural networks and to train them. Obstacles can be spawned and deleted using the *spawn_sdf_model* and *delete_model* gazebo services. These services permit to define the initial position of the obstacle and permit to create multiple instances of the same object. In the defined training framework, obstacles are generated by calling those services in the Python script that implements the OA agent.

A scheme with all the elements programmed in ROS to create the training environment of the reactive obstacle avoidance agent is shown in Fig. 3.16. It includes the RotorS framework, the nodes used to simulate the RC transmitter and the topics of the real platrofm, the nodes of the autopilot, the path following agent and the reactive avoidance agent. This scheme is explained in detail in Section 3.5.

8.3 DDPG for Reactive Obstacle Avoidance

This section gives details of each of the elements that form the *DDPG* approach to solve the reactive obstacle avoidance problem that was developed in this work. That includes the employed control structure, the action, state and reward of the agent, the structures of the neural networks and other parameters related to the *DDPG* algorithm.

8.3.1 Action

Fig. 8.1 shows the reinforcement learning structure. In this structure, the obstacle avoidance agent receives the reference path ($\boldsymbol{p}_d(\gamma)$) and sends it, possibly modified ($\boldsymbol{p}_{d2}(\gamma)$), to the path following agent. Just as in Chap. (6), the reference path is given by two vectors, one for each component (x_d and y_d), discretized by parameter γ with a precision of 0.01. There are different ways of sending the modified path to the PF agent. For instance, the full path could be sent at each step, however, this would be computationally expensive and inefficient. Otherwise, a small section of the path, corresponding to the part that is being currently followed, could be sent, but it could still become too demanding for the communication channel. Furthermore, making the agent to compute the full sequence of path points would become a challenging problem. Instead, in this work a simple and efficient way of sending the path to the PF agent was considered;

sending only the RL state vector of the path following agent. That is, the real environment's state vector of the PF agent is calculated in the obstacle avoidance script, and the OA agent computes certain variations or increments over this state. In this way, the PF agent is blind about the reference path and only receives the modified RL state vector, and the OA agent can generate the state increments to modify the state/path in such a way that any obstacle is avoided.

At this point, it is important to recall that the state vector of the PF agent is formed by four elements (see Section 6.4.3); the cross-track error, the angle error, the angle error in a point forward on the path and the velocity of the vehicle. The vehicle's velocity is an intrinsic state of the vehicle, but the rest of the states depend on the path that is being followed. Thus, modifying these states is equivalent to changing the reference path. In the present work, only the cross-track error state will be modified, leaving the other three states unchanged.

The defined action of the OA agent will act only in the cross-track error state, that computes the distance from the path to the vehicle. This action is equivalent to having an offset to the reference path, as it is represented in Fig. 8.2. That is, this action can be interpreted as the original path displaced in the y axis of the path tangential frame, $\{T\}$. Controlling the offset to the path, the agent will generate the corresponding trajectories to avoid the appearing obstacles. It is important to mention that, as the angle error state is computed as the angle between two vectors (x-body axis and x-tangential axis), it remains constant even if the path is translated.

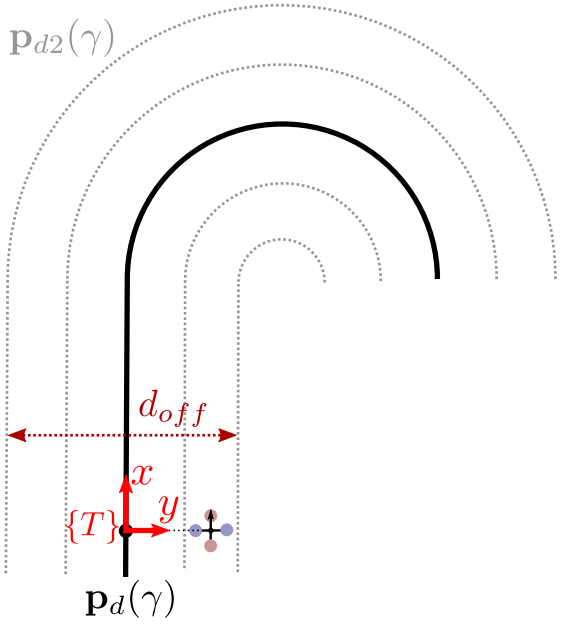

FIGURE 8.2: Action of the agent modifies the path offseet d_{off}.

The distance offset to the path is named d_{off}. To maintain a smooth profile of this path offset, the action of the OA agent is set to be an increment over it. That is, the action is set to be the derivative of the path offset. This is represented in Eq. (8.1). This equation shows, at time step k, how the modified cross-track error state ($e_{d2,k}$) is computed from the distance offset ($d_{off,k}$), and how the distance offset is obtained from the action (a_k) of the OA agent, where Δt is the

time step of the agent. The derivative of the path offset is limited to $3^{m}/_{s}$, which was considered to be a sufficiently high speed to execute a reactive manoeuvre.

$$e_{d2,k} = e_{d,k} + d_{off,k} \;\mid\; d_{off,k} = d_{off,k-1} + a_k\,\Delta t \tag{8.1}$$

During the training process, an Ornstein–Uhlenbeck noise function (Eq. 6.7) is added to the action of the agent for exploration purposes. The modified PF state including this noise function is computed in Eq. (8.2). The influence of this noise function is decreased continuously with the number of training episodes (j) in such a way that a transition between exploration and exploitation of the policy is achieved during the learning process. Parameter λ regulates the velocity of this transition.

$$e_{d2,k} = e_{d,k} + d_{off,k-1} + \left(a_k + \frac{n_k}{j/\lambda + 1}\right)\Delta t \tag{8.2}$$

8.3.2 State

Regarding the state vector of this agent, the first state that must be included is the path offset, d_{off}. The action computed by the agent is the derivative of this parameter, thus, if the agent does not know the real value of the path offset, it will become impossible for it to perform the correct variations over d_{off} to avoid obstacles.

Next, LIDAR measures are included in the state vector to provide the agent information about the environment and possible obstacles. The LIDAR measurements consist on 8 range distances measured by eight beams distributed in a horizontal field-of-vision of 48°(more information about the LIDAR sensor in Section 7.2). These distance measurements are treated to eliminate ground detections (Section 7.4). Eliminating ground detections permitted a much faster convergence of the trained agents. Actually, in most of the cases the agents were not able to converge without this treatment.

The cross-track error (i.e. the first state of the PF agent, e_d) is also included in the state vector. That is, knowing the distance from the vehicle to the original reference path permits the agent to localize the vehicle in space, which helps the interpretation of the LIDAR data.

With the introduced set of states, the system is capable of following the path in a correct way and, when the LIDAR detects an obstacle in the vehicle's route, it is able to perform a reactive action to start avoiding it. However, since the LIDAR has a FOV of 48°, as soon as the vehicle starts evading the obstacle, it disappears from the LIDAR's field-of-view. When this happens, the agent considers that the vehicle has already avoided the obstacle and moves back to the path provoking a collision. This situation is represented in Fig. 8.3, where the trajectory of the vehicle is represented with a grey line and the collision instant is marked with a red star. That behaviour occurs because, in the *DDPG* algorithm, the action computed according to the agent's policy only considers the current information provided by the state vector, without taking into

account the historical data. Therefore, if the agent notices that the vehicle is out of the path and no obstacles are detected, it will always try to converge back to the reference path.

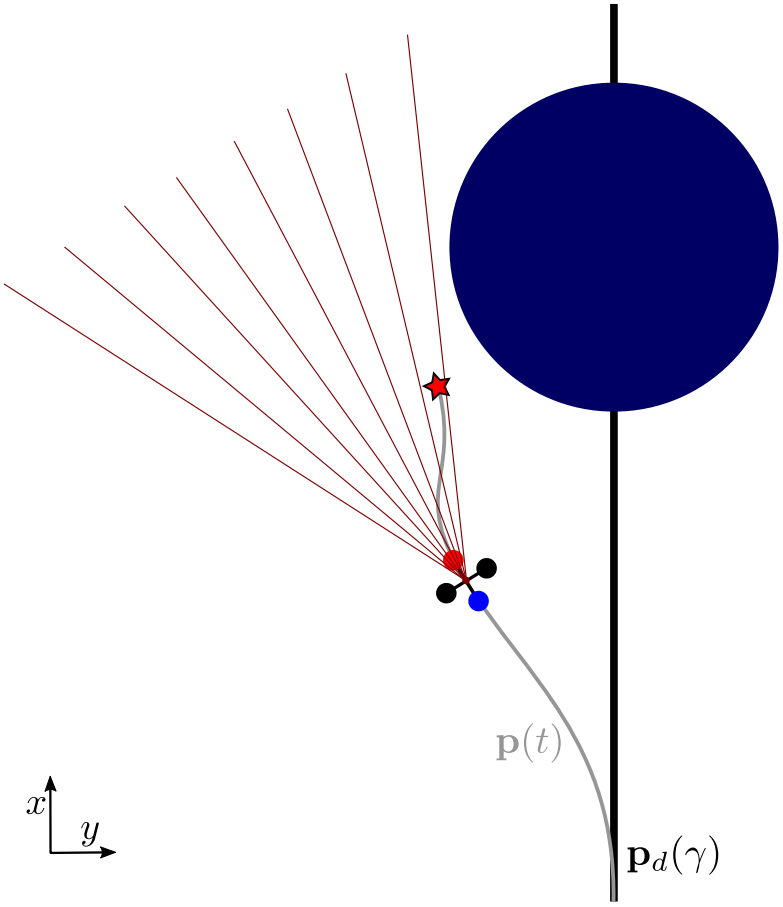

FIGURE 8.3: Vehicle crashes if only instantaneous information of LIDAR is used.

To deal with that, it is necessary to supply historical information of the obstacles to the agent. Other works provide the historical information by using recurrent neural networks, such as [99] where *LSTM*s networks are integrated in the *DDPG* algorithm. The issue of RNNs is that convergence is not always achieved and the training process becomes slower. In this work a simpler and functional solution has been chosen. The proposed approach consists in defining a state that increases when the LIDAR is detecting an obstacle and decreases otherwise. To compute this state, if an obstacle is detected, the inverse of the distance measured by each LIDAR beam is integrated. And when there are no near obstacles in the LIDAR's field-of-view, the state is decreased with a predefined rate. That is, this state is equivalent to an integral of the LIDAR detections. The agent can learn that positive values of this state mean that some obstacle has been recently detected. Thus, the vehicle will not move back to the path until this state reaches low values.

The formal definition of the LIDAR integral state, n_L, is shown in Eq. (8.3), where $d_{L,i}$ is the distance detected by the ith beam of the LIDAR. To determine whether a detection is occurring or not, a distance threshold, d_T, is used. The values of k_c and k_d constants (charging and discharging rates of the state, respectively) determine the dynamics of this integral state. Note

that n_L is constrained to avoid negative values and to have a maximum value of $n_{L,max}$. The maximum value, $n_{L,max}$, permits to regulate the time that the vehicle must converge back to the path after the avoiding manoeuvre has started.

$$\begin{aligned} &\text{if} \quad \forall i \in [1, 2 \cdots 8] \;\; \exists! \; d_{L,i} < d_T \quad \text{then} \quad n_L = \max(0, \; n_L - k_d) \\ &\text{otherwise} \quad n_L = \min\left(n_{L,max} \,, \; n_L + \sum_{i=1}^{8} \frac{k_c}{d_{L,i}} \right) \end{aligned} \tag{8.3}$$

Fig. 8.4 shows the evolution of the LIDAR integral state, n_L, while performing an avoidance manoeuvre on a straight line path. n_L is shown in the bottom graph and the avoidance manoeuvre, represented by the path distance error, d_{min}, is shown in the top graph. This graphs are obtained with a simulation in the RotorS environment with the trained OA agent. In this simulation, n_L is obtained with charging and discharging constants of $k_c = 1$ and $k_d = 0.1$, respectively. The maximum value of the state, $n_{L,max}$, is set to 6. The values of these parameters are kept constant in the rest of the chapter.

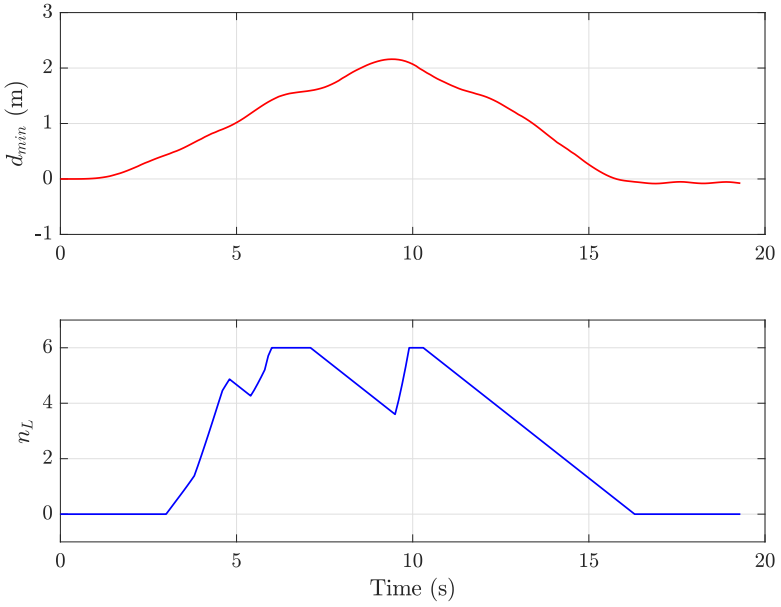

FIGURE 8.4: Integral LIDAR state, n_L, while performing an avoidance manoeuvre ($k_c = 1$, $k_d = 0.1$ and $n_{L,max} = 6$.

Including n_L in the state vector allows the agent to know when it is performing an avoidance manoeuvre even if the obstacle is not in the LIDAR's field-of-view. The agents trained with this state vector set-up are capable of avoiding obstacles centred on a straight line path. However, some tests still resulted in a collision. The cause of that problem is that the velocity of the vehicle was not considered when performing the avoidance manoeuvre (vehicle's velocity continues increasing until the cruise velocity is reached). That is, the evasion trajectory not only depends on the vehicle and obstacle's position, but also on the vehicle's velocity. Once the velocity of the vehicle in the x-body axis was included into the state vector, the agent could learn evasion trajectories at different velocities.

The set of states introduced before contain enough information to build a reactive obstacle avoidance approach for a straight line path. However, when translated to a different path with curves, such as a Lemniscate path, it becomes very difficult for the trained agents to avoid obstacles while following a curve. That is because the optimal avoiding trajectory depends on the shape of the path. To deal with this issue, the curvature of the path is set as a new parameter of the state vector. Specifically, this state is computed as the average of the path's curvature in a section of 5 meters around γ_{min} (path closest point to the vehicle given by γ). This new state, $\overline{k}(\gamma)$, is computed in Eq. (8.4), where $\gamma_{min,i}$ and $\gamma_{min,f}$ are the initial and final points of the average window, and n is the number of points used to calculate the average.

$$\overline{k}(\gamma_{min}) = \left(\sum_{\gamma=\gamma_{min,i}}^{\gamma_{min,f}} \frac{\left\| \frac{d\boldsymbol{p}_d(\gamma)}{d\gamma} \times \frac{d^2\boldsymbol{p}_d(\gamma)}{d\gamma} \right\|}{\left\| \frac{d\boldsymbol{p}_d(\gamma)}{d\gamma} \right\|^3} \right) \Big/ n \tag{8.4}$$

Adding the path curvature to the set of states results in a much better performance, avoiding most of the obstacles with different path shapes. Every agent trained with this state vector always converges to a solution in which it constantly performs the avoidance manoeuvre by the same side of the obstacle (either left or right). Nevertheless, depending on the vehicle's position, the path's shape and the obstacle's position, there is always a side of the obstacle that is more convenient to go to. Thus, this behaviour is harmful both for the path following performance and the obstacle avoidance capabilities.

To provide the agent with the ability of avoiding obstacles by both sides, richer information of the environment must be included in the state vector. In this work, the LIDAR integral state, n_L, has been split into two states: one state for the detections of the left-side beams of the LIDAR ($n_{L,l}$) and another state for the detections on the right-side beams ($n_{L,r}$). In this way, the agent not only knows that an obstacle has been recently detected even if it is not in the LIDAR's field-of-view, but it also can determine whether it was detected on the left or on the right side of the vehicle. With this approach, the trained agents acquired the ability of deciding when to avoid the obstacles by the left-side and when by the right-side, depending on the vehicle, path and obstacle's situation.

The formal definition of $n_{L,l}$ and $n_{L,r}$ states is shown in Eq. (8.5) and Eq. (8.6), respectively. Note that, since n_L state is split into two states, the charging and discharging constants are also divided by 2.

$$\begin{aligned} &\text{if} \quad \forall i \in [1, 2 \cdots 4] \;\; \exists! \; d_{L,i} < d_T \quad \text{then} \quad n_{L,l} = \max\left(0, \; n_{L,l} - {}^{k_d}\!/_2\right) \\ &\text{otherwise} \quad n_{L,l} = \min\left(n_{L,max}, \; n_{L,l} + \sum_{i=1}^{4} \frac{{}^{k_c}\!/_2}{d_{L,i}} \right) \end{aligned} \tag{8.5}$$

$$\begin{aligned} &\text{if} \quad \forall i \in [5, 6 \cdots 8] \;\; \exists! \; d_{L,i} < d_T \quad \text{then} \quad n_{L,r} = \max\left(0, \; n_{L,r} - {}^{k_d}\!/_2\right) \\ &\text{otherwise} \quad n_{L,r} = \min\left(n_{L,max}, \; n_{L,l} + \sum_{i=5}^{8} \frac{{}^{k_c}\!/_2}{d_{L,i}} \right) \end{aligned} \tag{8.6}$$

Summarizing, the state vector defined in this work includes the path distance offset (d_{off}), the LIDAR measurements treated to eliminate ground detections (L_i), the cross-track error (e_d), the vehicle's velocity on the x-body axis (u), the average path curvature around the closest path point to the vehicle ($\overline{k}(\gamma_{min})$) and the integral (with saturation) of the LIDAR detections of the left beams ($n_{L,l}$) and of the right beams ($n_{L,r}$). This vector, with a total of 14 states, is represented in Eq. (8.7).

$$\boldsymbol{s} = \left\{d_{off},\, L_1 \cdots L_8,\, e_d,\, u,\, \overline{k}(\gamma_{min}),\, n_{L,l},\, n_{L,r}\right\} \tag{8.7}$$

8.3.3 Reward

To define the reward function, two virtual zones around the obstacle were created: the banned zone and the safety zone. These zones are represented in Fig. 8.5, which shows the obstacle in blue surrounded by two circles that represent the two zones. The banned zone is created with a circumference around the obstacle at a distance of 0.5m from it. The safety zone is created with a circumference at a distance of 0.25m of the banned zone edge. That is, at a distance of 0.75m to the obstacle. In this figure the vehicle is represented proportionally to the obstacle and to the defined zones.

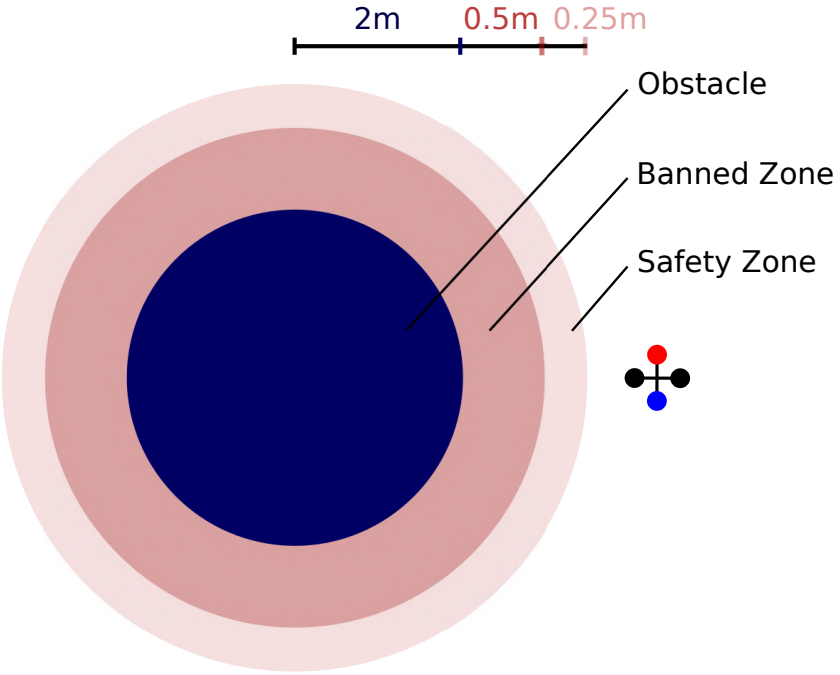

FIGURE 8.5: Obstacle surrounded by the prohibited zone and the safety zone.

If the vehicle enters the banned zone, it is considered that a collision has occurred, and the simulation is stopped. Thus, the vehicle is not allowed to enter this zone. If the vehicle enters the safety zone while training, the agent will receive a negative reward, but the training episode will continue. Therefore, the trained agent will learn to avoid entering this zone to maximise

the received reward. The safety zone is only present while training, so, if the vehicle enters the safety zone while performing a test, nothing will happen.

Defining a banned zone is always necessary to keep a proper distance to the obstacle when avoiding it. However, when only the banned zone is defined, the trained agents will always learn to avoid the obstacle travelling around the edge of the banned zone. Therefore, it will become very easy that, due to sensor noise or external disturbances, the vehicle enters the banned zone and the experiment is stopped. To avoid this issue, the mentioned safety zone is defined.

The first reward function defined in this work is shown in Eq. (8.8). This function has three terms. The first term, $-10|d_{off}|$, penalizes the path offset distance. That is, the agent will try to send the real cross-track error state (with no path offset) to the PF agent if no obstacles are present. The second term, $-50SZ$, penalizes the vehicle when it enters the safety zone, where SZ is a boolean parameter that is 1 when the vehicle is in the safety zone and 0 otherwise. And the third term, $-2000BZ$, penalizes the vehicle when it enters into the banned zone, where BZ is another boolean parameter that is 1 when the vehicle enters the banned zone and 0 otherwise. Note that this third term can only be activated once in an episode, since when the vehicle enters the banned zone it is considered that a collision has occurred and the episode is terminated. The reward function also includes a positive reward of 10 units given at each step. This positive reward is used only for helping the interpretation of the training data, since with it the accumulated episode reward converges to a positive value during the training process of the agent.

$$r = -10|d_{off}| - 50SZ - 2000BZ + 10 \tag{8.8}$$

It is important to mention that this is the reward that provided the best results in terms path following error and obstacle avoidance capability. Several other reward functions were also tested with poorer results. For instance, instead of the defined constant penalty when entering the safety zone ($-50SZ$), a penalty proportional to the inverse of the obstacle's distance (similar to an artificial potential field) was tested. However, even having a proportional reward instead of a constant one may intuitively seem a better option, the agents trained with this reward presented problems to converge to a stable solution.

Since the reward function of Eq. (8.8) is simple and does not include any immediate reward, the training process requires more time to converge. Having no immediate rewards means that only when the vehicle enters the vicinity of the obstacle (Fig. 8.5) the reward changes, but it remains unaltered when the vehicle is moving towards the obstacle. On the other hand, this simple reward function allows the agent to freely search for the best policy to avoid the obstacles without any additional constraints.

Other reward functions that include immediate terms were also tested. With these reward functions, the trained agents may not present an optimal policy, but the immediate terms help to achieve a faster convergence reducing its training time. Among the explored rewards including the mentioned immediate terms, there is one that showed considerably good results with short training time.

This reward function contains the same terms as the ones stated in Eq. (8.8) incorporating a LIDAR-based reward term. Basically, this new term gives a penalty when the LIDAR sensor is detecting an obstacle, taking into account the distance at which the obstacle is detected, provided by each LIDAR beam. Including this term, the reward progressively becomes more negative when the vehicle is moving towards an obstacle and the LIDAR is detecting it. Thus, the agent can easily learn that detecting an obstacle near to the vehicle is hazardous and that an avoidance manoeuvre must be carried out.

A straightforward manner of generating this new term would be to sum the inverse of the distances measured by each LIDAR beam, similarly on how the integral state (n_L) is updated when the LIDAR is detecting an obstacle. This form of calculating the LIDAR-based reward term, r_{im}, is shown in Eq. (8.9). However, note that this term would still compute negative rewards even at long distances to the obstacle, which is not convenient. Furthermore, the rewards generated, for instance, at 1m and 0.75m to the obstacle, would be notably different, when actually there is difference. That is, an obstacle is equally urgent to consider if it is at 1m or if it is at 0.75m to the vehicle.

$$r_{im} = -\sum_{i=1}^{8} \frac{1}{d_{L,i}} \tag{8.9}$$

In this work a sigmoidal function is used for representing how urgent is to avoid an obstacle depending on its distance to the vehicle. The LIDAR-based reward term is computed with a sigmoidal function, as shown in Eq. (8.10), where $d_{T,r}$ is the distance threshold of the sigmoidal reward function (i.e. the distance at which the reward starts to decrease).

$$r_{im} = -\sum_{i=1}^{8} \frac{1}{1 + \exp\left(2(d_{L,i} - d_{T,r})\right)} \tag{8.10}$$

Fig. 8.6 compares the two ways of calculating the LIDAR-based reward term: with an inverse function of the distance and with a sigmoidal function. The graphs are computed in function of the distance measured by a LIDAR beam (d_L). The distance threshold of the sigmoidal reward function, $d_{T,r}$, is fixed to 4, which means that the reward function is activated around 4m to the obstacle.

With the defined sigmoidal function, the agent will start receiving a penalty when the LIDAR beams detect an obstacle at a distance of $d_L \lessapprox d_{T,r} + 2$. And so, the agent will try to avoid that penalty by moving the vehicle to another direction. Therefore, $d_{T,r}$ will determine the distance at which the avoidance manoeuvre is started. Note that defining this distance threshold to a fixed value for all the LIDAR beams, let's say 4 meters, is equivalent to considering equally hazardous an obstacle at a given distance in front or to the left of the vehicle. With a constant threshold, the hazardous zone is defined as a circumference around the vehicle. Since the vehicle is constrained to move only on the x-body axis, defining a circular hazardous zone would be inappropriate. Instead, in this work the distance $d_{T,r}$ is defined as a function of the LIDAR beam angle to create an elliptical hazardous zone around the vehicle. The comparison between having a circular or elliptical distance threshold is represented in Fig. 8.7, where the circular

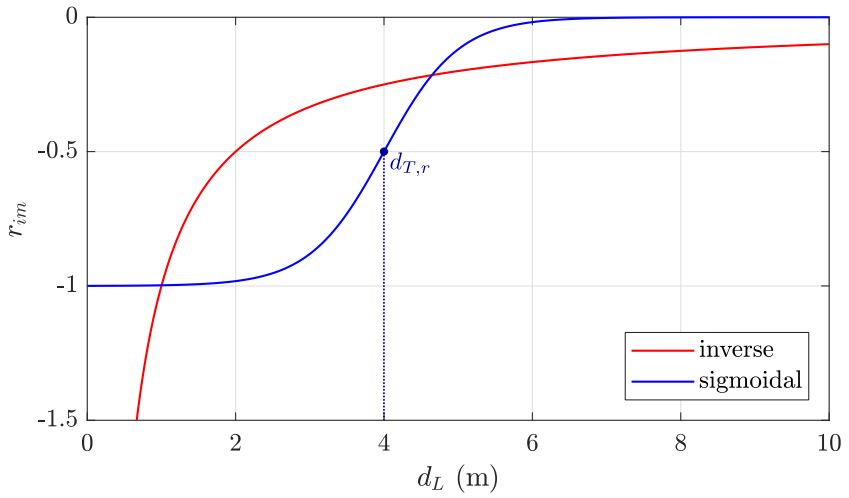

FIGURE 8.6: Comparing two forms of calculating the LIDAR-based reward term: inverse function and sigmoidal function.

distance threshold is set to 4m and the main axes of the ellipse are 4m and 1m. Note that with the elliptical distance threshold, when an obstacle appears in front of the vehicle, the penalty starts decreasing as soon as the vehicle starts turning.

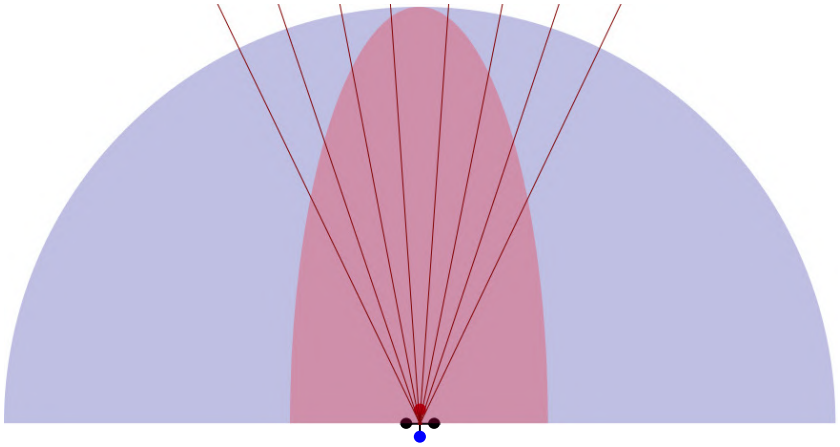

FIGURE 8.7: Circular distance threshold or elliptical threshold ($d_{T,r}$).

Eq. (8.11) shows how the elliptical distance threshold is computed in function of the angle of each LIDAR beam ($\psi_{L,i}$), where r_a and r_b are the long and the short axes of the ellipse, respectively. These axes will define the shape of the hazardous zone, and thus, they will constrain how the avoidance manoeuvre is accomplished.

$$d_{T,r}(\psi_{L,i}) = r_a r_b \sqrt{\frac{1 - \left(\cos(|\psi_{L,i}|)/\sin(|\psi_{L,i}|)\right)^2}{r_b^2 - r_a^2 \left(\cos(|\psi_{L,i}|)/\sin(|\psi_{L,i}|)\right)^2}} \tag{8.11}$$

Finally, the reward function which includes the LIDAR-based term, is shown in Eq. (8.12), where $d_{T,r}(\psi_{L,i})$ is computed as in Eq. (8.11). The LIDAR-based reward term is scaled by 10 units.

$$r = -10|d_{off}| - 10\sum_{i=1}^{8}\frac{1}{1+\exp\left(2(d_{L,i}-d_{T,r}(\psi_L,i))\right)} - 50SZ - 2000BZ + 10 \tag{8.12}$$

8.3.4 Structure of the *DDPG* agent

The structures of the Neural Networks of the actor and critic are the same as the ones employed in the path following agent developed in Chap. (6). They are shown in Fig. 6.7 and Fig. 6.8, respectively. Moreover, most of the parameters of the PF agent, defined in Table 6.1, are also maintained in the agent presented in this chapter. Thus, it is shown that, once obtained an appropriate and functional structure and set of parameters of a *DDPG* agent, the agent can be straightforwardly translated to solve other similar problems or problems with similar environments.

The parameters that are modified in the reactive obstacle avoidance agent are the ones related to the noise exploration function and the maximum steps of one training episode. The values of the modified parameters are presented in Table 8.1. These parameters denote that the exploration ratio is much larger in the obstacle avoidance agent than in the path following agent. Having a large exploration ratio permits the OA agent to learn the best form of avoiding each obstacle. For instance, it helps to determine whether it is better to avoid an obstacle by the left side or by the right side. Furthermore, the maximum number of steps of one episode is considerably increased. The reason for this is explained in Section 8.5.

TABLE 8.1: Modified noise parameters of the *OA* agent.

Symbol	Description	Value
θ_n	Mean reversion rate of noise function.	0.15
σ_n	Volatility of noise function.	3
λ	Ratio of exploration-exploitation transition.	500
-	Maximum steps of one episode.	3000

8.4 Implementation of the PF and Reactive OA Approach

Two python 3.5 scripts are programmed to implement the path following and reactive obstacle avoidance approach. In Fig. 8.8 the flowchart of these two scripts is presented, where communication elements are represented in grey color. It can be observed that the PF script includes two control modes: the path following mode and the hover mode. In the path following mode, the *DDPG* PF agent is in charge of following the reference path that is received from the OA script in the form of subsequent PF state vectors. In the hover mode a PID controller is used to hover the vehicle around a reference point in space. In both control modes, the path following commands (ψ_{cmd}, z_{cmd}, u_{cmd} and v_{cmd}) are sent to the autopilot controller. The PF script starts at hover mode and waits for the OA script to proceed with the path following task.

The path to be followed and the transition between the two control modes is supervised by the OA script. In the OA script, first, the definition of the reference path is made, and then, the *DDPG* OA agent is launched. The *DDPG* OA agent computes the path offset action, which is used to obtain the PF state that is sent to the PF script. Once the path following task is finished, the OA script sends the PF script back to hover mode. There are two ways of finishing the PF task: by reaching the end of the path or when a collision is produced. In the simulated framework, it is considered that a collision is produced when the vehicle enters the banned zone (Fig. 8.5). When this happens, the obstacle is deleted from the gazebo framework and the hover mode is activated. In the simulated framework the OA script is also in charge of generating obstacles and place them randomly along the path and around it. This procedure is made after the reference path is defined.

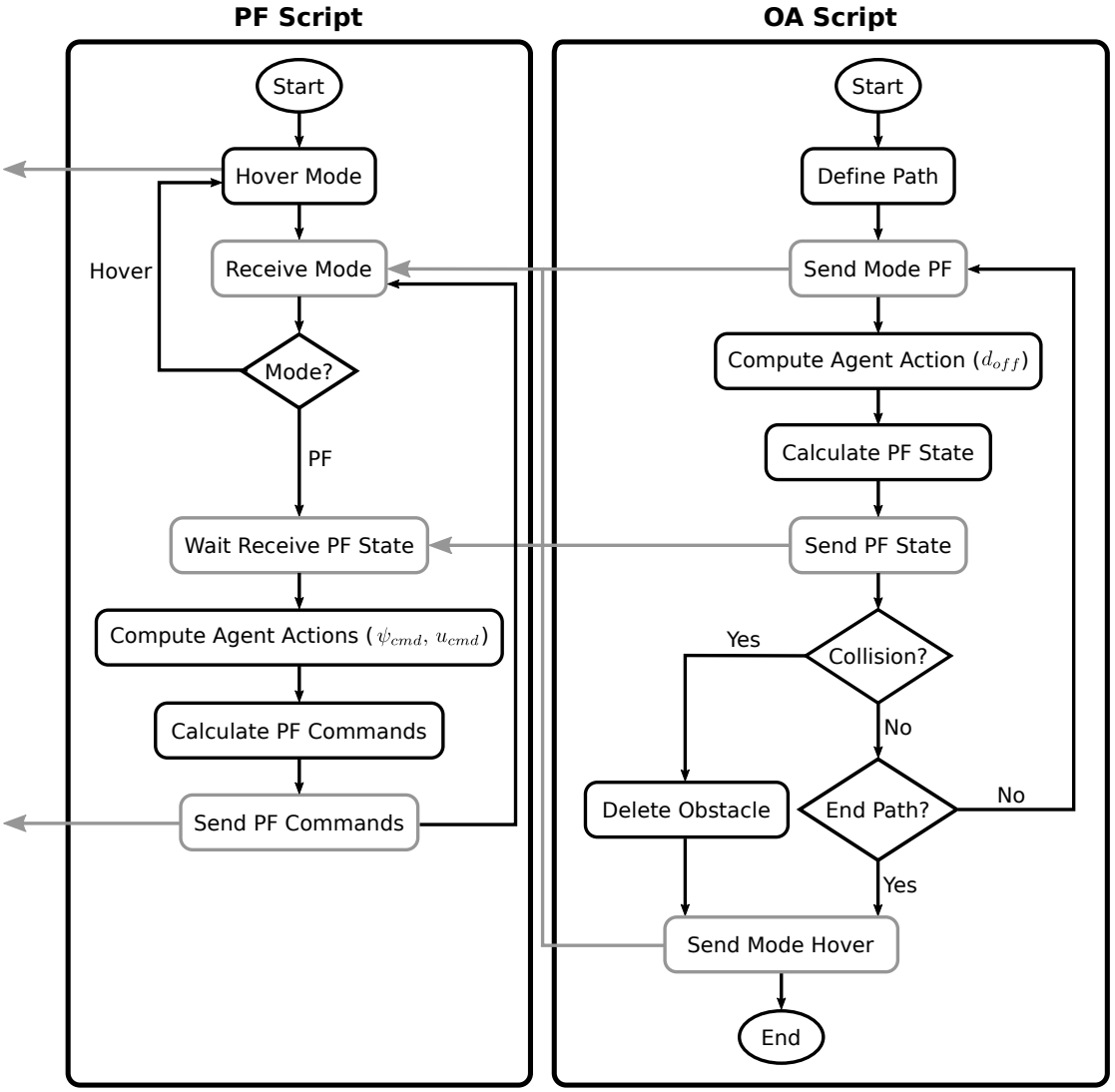

FIGURE 8.8: Flowchart of the two scripts that implement the PF and Reactive OA approach.

8.5 Training Process

The training process of the agents was made in the environment detailed in Section 8.2.1. Just as in Chap. (6), the training environment is integrated in a linux Xubuntu virtual machine with a dedication of 8GB RAM and four 1.80GHz processors (i7-8550U CPU), and the training process is performed in real time.

The path following and obstacle avoidance problem can become very complex because many different situations can occur. That is, obstacles can appear at different locations while following diverse path shapes. To obtain a complete and functional agent to deal with the stated problem, a very rich training framework must be generated.

The generated training framework consists on making the vehicle follow a straight line of 10m followed by a half asymmetrical lemniscate path, defined in Eq. (6.17), where the value of its two radius change every episode, taking a random value between $3m$ and $10m$ with a uniform probability distribution. That is, a straight line path plus the same path that was used to train the PF agent developed in Chap. (6). Following paths with different curves and with straight lines will permit the agent to learn the avoiding manoeuvres at different situations.

The straight line at the beginning of the path is used to prevent from having an obstacle just in front of the vehicle when starting an episode. Therefore, no obstacles are placed in this line. This is coherent with real experiments, since the human supervisor would not launch the vehicle with an obstacle very close to it. Furthermore, this starting line on the path permits to place obstacles in any point along the Lemniscate path, which allows the agent to learn avoiding manoeuvres with different path's shapes. Also, the straight line path permits to reach the cruise velocity before starting the lemniscate path, preventing the avoidance manoeuvres at the transient phase.

Obstacles are generated in each episode with a probability of 75%. That is, in average, 1 out of 4 episodes will be performed without any obstacle. That permits the agent to learn the policy when no obstacles are present. The obstacles are placed at random locations along the lemniscate path, with a uniform probability distribution, in a range of ±2m of distance to the path. That allows the agent to learn to avoid obstacles that are centred on the path as well as obstacles that are close to it, deciding which side is better to avoid them, depending on the location of the obstacle, the vehicle and the path shape. Furthermore, it is trained with obstacles that are out of the vehicle's route in such a way that the LIDAR detects them but no avoidance manoeuvre is needed.

The agents are trained in ideal conditions. That is, the system uses ground truth measurements and the vehicle starts each episode at the initial position of the path with the yaw angle oriented tangentially to it. It is important to recall that the orientation and velocity of the vehicle are controlled by the autopilot and that the path following problem is solved by the path following agent (*Agent 3*) developed in Chap. (6). Therefore, these two control blocks are also included while training the obstacle avoidance agent.

It is interesting to mention that, without the treatment of LIDAR measurements to eliminate ground detections, most of the trained agents had problems to interpret the distance measurements provided by this sensor. The agents trained in these conditions converged to a solution where the vehicle always follows the path a certain distance to it. That is, the vehicle remains at a large enough distance to the path to elude all the obstacles that may appear. That is because obstacles are generated only in a range of ±2m of distance to the path. Farther than this distance, there are no obstacles. When the LIDAR measurements are treated, this peculiar behaviour is solved.

Another particular issue is that, if the length of an episode is set to 300 steps (30 seconds), just as the training episodes of the PF agent of Chap. (6), the agents converge to a wrong but tricky policy. That is, the resulting policy makes constant and abrupt changes of its action (which affects d_{off}) in such a way that the velocity of the vehicle is severely reduced due to the constant changes on the reference path. Since the vehicle moves very slowly, most of the episodes end before the obstacle has appeared in the vehicle's route, thus, avoiding the collision. This issue is solved by extending the length of the episodes. In this work, the number of steps is set to 3000 (300 seconds). However, even the number of episodes must be considerably extended to solve this problem, after the first training episodes, the agent can improve its policy, and the rest of episodes end in around 30-50 seconds (the average time needed to perform a lap on the path at the optimal velocity while avoiding the obstacle).

The results of the training process of the agent with the state defined in Eq. (8.7) and the reward function stated in Eq. (8.8) are shown in Fig. 8.9. This figure shows the average path distance error, the average velocity, a boolean parameter that indicates if a collision occurred or not (1-collision, 0-no collision) and the accumulated reward on each episode during the training process of the agent. The real data is shown with gray dashed lines and black lines represent a 1000-episode moving average. This agent was trained during 10500 episodes. As can be observed, the average path distance and velocity increase over the episodes, reaching a stable value of around $0.31m$ and $1.24{}^{m}/_{s}$, respectively. The accumulated reward stabilizes around 2053. The moving average of the boolean parameter that represents whether a collision in the episode occurred or not, stabilizes to a value of 0.048. This value can be considered an indicator of the probability of having a collision. This indicator remains close to 0 at the first episodes, since the agent can not properly follow the path yet, and starts increasing when the agent improves the path following performance because obstacles are placed near to the path. It arrives to a maximum value of around 0.25, meaning that 1 out of 4 episodes end in a collision. However, the stabilized value of 0.048 can be considered of having a collision every 20 episodes. It is important to mention that the noise function can increase the probability of having a collision.

The training process of the agent that includes the LIDAR-based reward term (Eq. 8.12) is shown in Fig. 8.10. This figure evaluates the same parameters of Fig. 8.9. Again the real values are represented with grey dashed lines and the black solid lines represent a 1000-episode moving average. It can be observed that the agent including the LIDAR-based reward term converges a little faster that the previous one. The training process of this agent was stopped at episode 8900. Training more than this resulted in a worse performance. The evolution of the parameters evaluated in Fig. 8.10 is similar to the one shown before (Fig. 8.9). The average

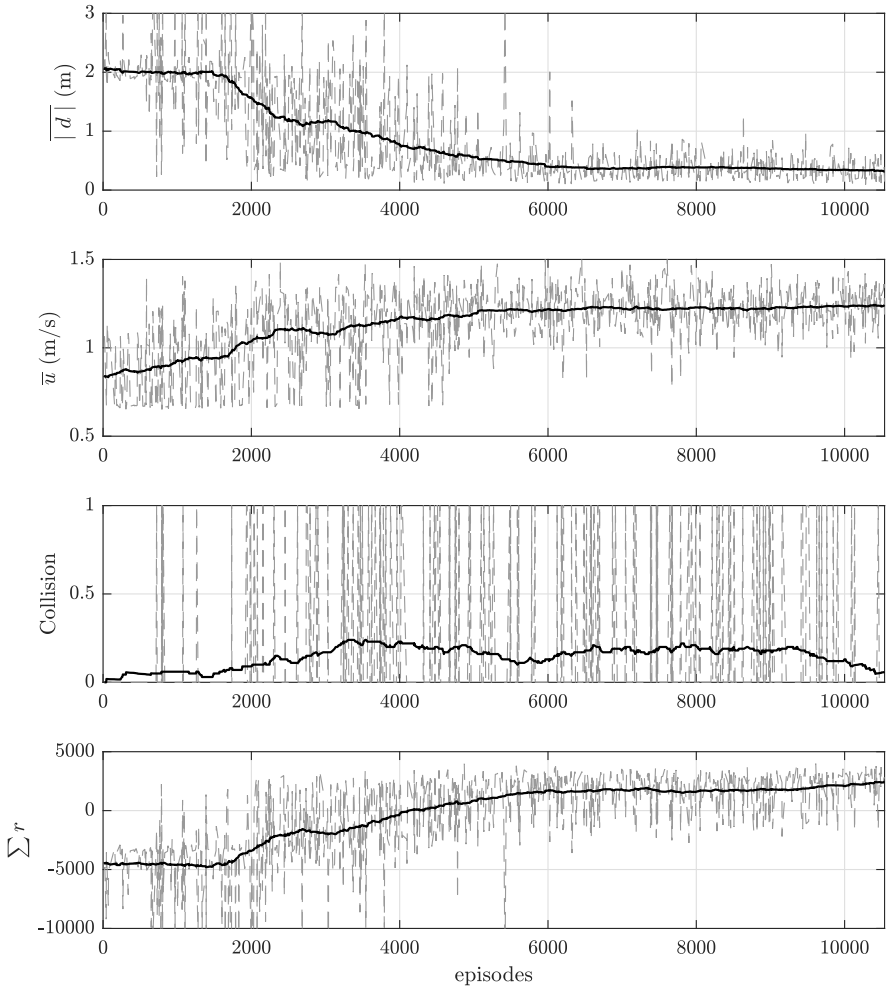

FIGURE 8.9: Average distance error, average velocity, probability of collision and accumulated reward on each episode during training phase of the OA agent with the standard reward; gray dashed lines are real values and black lines are a 100-episodes moving average.

distance error converges around $0.37m$, the velocity at $1.25^{m}/_{s}$ and the accumulated reward stabilized about 1879. Regarding the average collision, it converges around 0.098. That means that the probability of having a collision with this agent is around the double of the one obtained with the other agent. Therefore, even this agent is able to converge in a slightly shorter number of episodes, the final performance is decreased as well.

8.6 Results

This section presents the results obtained with the agent that achieved the best performance in terms of path following error and obstacle avoidance capabilities among the different agents that were trained. This agent was trained with the reward function defined in Eq. (8.8) and its training process is presented in Fig. 8.9. The *Agent 3* developed in Chap. (6) is used as the path following algorithm. The results are obtained with the RotorS simulation platform, the

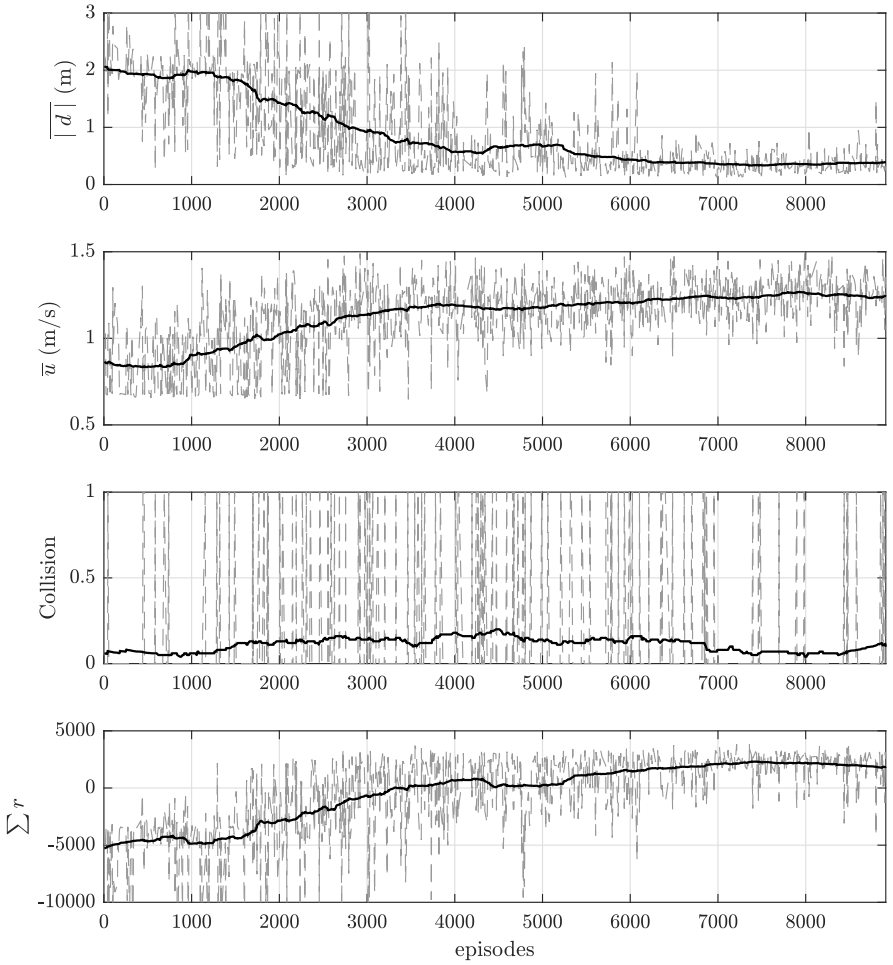

FIGURE 8.10: Average distance error, average velocity, probability of collision and accumulated reward on each episode during training phase of the OA agent with the LIDAR-based reward; gray dashed lines are real values and black lines are a 100-episodes moving average.

simulation framework where both the PF agent and the OA agent were trained. Two paths are used to test the approach; a lemniscate path and a spiral path. These simulation results are shown in next subsections.

In addition to the agent tested in this section, several obstacle avoidance agents with different states, reward functions and RL parameters were trained and tested during the process of obtaining the final approach presented in this chapter. Table 8.2 introduces a summary of the main characteristics of some of these agents and a description of their performance handicaps.

Fig. 8.11 shows a comparison of the trajectory obtained with the agent of the standard reward function, represented with a red dashed line, and the agent that includes the LIDAR-based reward, represented with a dash-dotted green line, while following a lemniscate path with a cylindrical obstacle, represented in blue. The starting point of the vehicle is shown with a red arrow pointing to the initial x-body orientation of the vehicle. In most of the tested experiments, these two agents exhibited very similar trajectories to avoid the obstacle, having a nearly identical path following performance. However, due to the constraints that introduces the LIDAR-based

TABLE 8.2: Behaviour of some of the trained and tested agents.

	Agent Conditions	Behaviour Description
1	$\boldsymbol{s} = \{d_{off}, L_1 \cdots L_8, e_d, u\}$ Reward: Eq. (8.8)	The agent has not enough information of the obstacle. As soon as the obstacle disappears from the LIDAR field-of-view, the agent moves the vehicle back to the path provoking a collision.
2	$\boldsymbol{s} = \{d_{off}, L_1 \cdots L_8, e_d, u, n_L\}$ Reward: Eq. (8.8)	The agent knows if an obstacle has been recently seen. However, it does not have enough information of the path. The agent cannot learn the manoeuvres for different path's shape. The vehicle often collides.
3	$\boldsymbol{s} = \{d_{off}, L_1 \cdots L_8, e_d, u, n_L, \overline{k}(\gamma_{min})\}$ Reward: Eq. (8.8)	The agent has information of the path's shape and can learn different avoidance manoeuvres. From state n_L, the agent only knows if an obstacle has been recently seen, but ignores whether it is at the right or at the left of the vehicle. The agent always learns to avoid by the same side of the obstacle.
4	State: Eq. (8.7) $n_{L,l}$ and $n_{L,r}$ without saturation Reward: Eq. (8.8)	If n_L is split in two states, the agent has enough information to learn to choose from which side is better to avoid an obstacle. However, if no saturation is used to these states, the vehicle can take a lot of time to converge back to the path when avoiding an obstacle (integral states affect directly the convergence time).
5	State: Eq. (8.7) Reward: Eq. (8.8) Noise: $\theta_n = 0.0075$, $\sigma_n = 0.15$ and $\lambda = 200$	If the same noise power of PF agent is used, the OA agent does not achieve a proper exploration during training phase. The lack of exploration affects the avoidance capabilities, resulting in more collisions and worse avoiding manoeuvres in terms of PF error.
6	State: Eq. (8.7) Reward: Eq. (8.12)	The agent with the LIDAR-based term in the reward function converges faster. Nevertheless, due to the introduced constraints, the agent can present difficulties to avoid some obstacles. Thus, more collisions occur.

reward term, in some cases, the agent trained with this reward can execute unusual trajectories to satisfy the defined reward function. This is the case of the example shown in Fig. 8.11, where this agent (dash-dotted green line) performs a very aggressive manoeuvre to avoid the obstacle and, then, it takes more time to converge back to the path. Moreover, some of these aggressive manoeuvres performed by the agent with the LIDAR-based reward end in a collision. Therefore, though the LIDAR-based reward provides more information to the agent, leading to a slightly faster convergence, it derives in a worse avoidance capability.

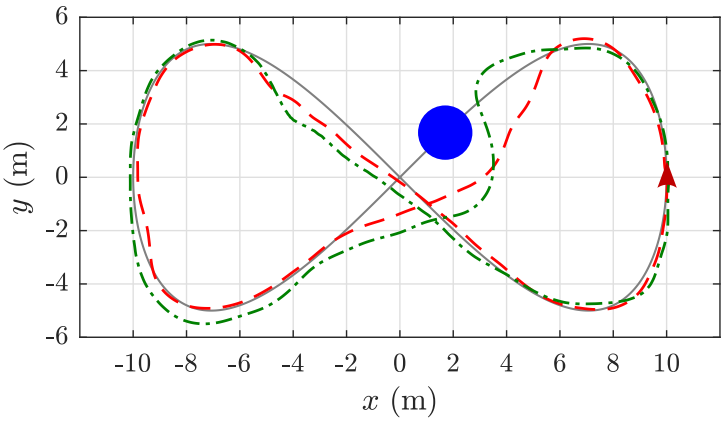

FIGURE 8.11: Trajectory on the xy plane of the agent with the standard reward (red dashed line) and the agent including the LIDAR-based reward (dash-dotted green line) while following a lemniscate path with an obstacle (blue circular object).

8.6.1 Lemniscate Path

This section evaluates with a lemniscate path the agent that is proposed in this work as a solution to the obstacle avoidance problem. The agent corresponds to the one trained with the reward function with no LIDAR-based term. The lemniscate path is defined in Eq. (6.16), where the amplitude, A, is set to 5m, and γ ranges from 0 to 2π, corresponding to a full lap on the path. On each of the presented tests, the vehicle starts in hover conditions at the initial point of the path with the *yaw* oriented tangentially to the path.

Table 8.3 compares the results obtained by the PF agent developed in Chap. (6) with the ones obtained by the PF+OA approach presented in this chapter. That is, the approach that combines the two DRL agents to solve the path following and obstacle avoidance problem. This results evaluate the performance of both approaches performing one lap of the lemniscate path without any obstacle. The table shows the average cross-track error, the total time to perform the lap on the path and the average velocity of the vehicle. The PF agent achieves a slightly better performance in terms of path distance error. Therefore, it is shown that having the OA agent can affect the PF performance somehow.

TABLE 8.3: Simulation results for one lap on the lemniscate path with no obstacles.

	$\overline{\mathbf{d}}$ **(m)**	**time (s)**	$\overline{\|\mathbf{v}\|}$ $(^m/_s)$
PF Agent	0.0765	44.6	1.3798
OA Agent + PF agent	0.1191	46.2	1.3223

Next, the proposed approach is tested performing a full lap of the stated lemniscate path with an obstacle at different positions centred on the path. The results of these simulations are presented in Table 8.4. This table shows the path position of the obstacle, given by γ_{obs}, and the distance from the center of the obstacle to the path, d_{obs}. It is important to recall that, as mentioned in Section 8.2.1, the obstacle is a cylindrical object with a radius of 1m. Note that obstacles are

located at $\gamma_{obs} \geq 1.2$. That is because the OA agent was not trained to avoid obstacles in the transient phase. Furthermore, this table presents the same parameters evaluated in Table 8.3, that is, the average cross-track error, the total time and the average velocity. In all the cases presented in the table, the vehicle is able to properly avoid the obstacle, having an average distance error of about 0.3-0.4m in most of the simulations. Other simulations present more error because the obstacle was placed near the center of the lemniscate so the vehicle needs to avoid it two times when performing a full lap.

TABLE 8.4: Simulation results for one lap on the lemniscate path with obstacle centred on the path.

γ_{obs}	$\mathbf{d_{obs}}$	$\overline{\mathbf{d}}$ (m)	time (s)	$\overline{\|\mathbf{v}\|}$ ($^m/_s$)
1.2	0	0.3839	55.4	1.1265
1.4	0	0.3904	54.6	1.1292
1.6	0	0.7718	59.2	1.0767
1.8	0	0.3027	50.6	1.2046
2	0	0.3587	50.3	1.2040
2.2	0	0.3113	50.6	1.1824
2.4	0	0.3846	54.5	1.1913
2.6	0	0.3319	52	1.2326
2.8	0	0.2902	50.6	1.2514
3	0	0.3309	55.3	1.1550
3.2	0	0.3714	55.4	1.1429
3.4	0	0.3233	51	1.1780
3.6	0	0.2993	50.9	1.1821
3.8	0	0.3347	54.7	1.1760
4	0	0.3190	57.3	1.1193
4.2	0	0.3070	52.7	1.1940
4.4	0	0.3036	53.2	1.1778
4.6	0	0.6016	54.4	1.1511
4.8	0	0.4739	51.4	1.2318
5	0	0.4249	48.7	1.3078
5.2	0	0.3679	50.6	1.2585
5.4	0	0.3389	55.2	1.1631
5.6	0	0.3631	52.9	1.2257
5.8	0	0.4198	51.5	1.2473

Fig. 8.12 shows the trajectory in the xy plane performed by the proposed approach in some of the simulations of Table 8.4. In this figure and the rest of the figures of this chapter, the initial position of the vehicle is marked with a red arrow pointing to the initial x-body orientation. From left to right and top to bottom, these figures correspond to the simulations with the obstacle placed at γ_{obs} = 1.2, γ_{obs} = 1.6, γ_{obs} = 1.8, γ_{obs} = 2.6, γ_{obs} = 3.2, γ_{obs} = 3.4, γ_{obs} = 3.8 and γ_{obs} = 5.4, respectively. These results show diverse examples of obstacle avoidance manoeuvres; at the long straight line, at the short straight line, at left-sided curves and at right-sided curves.

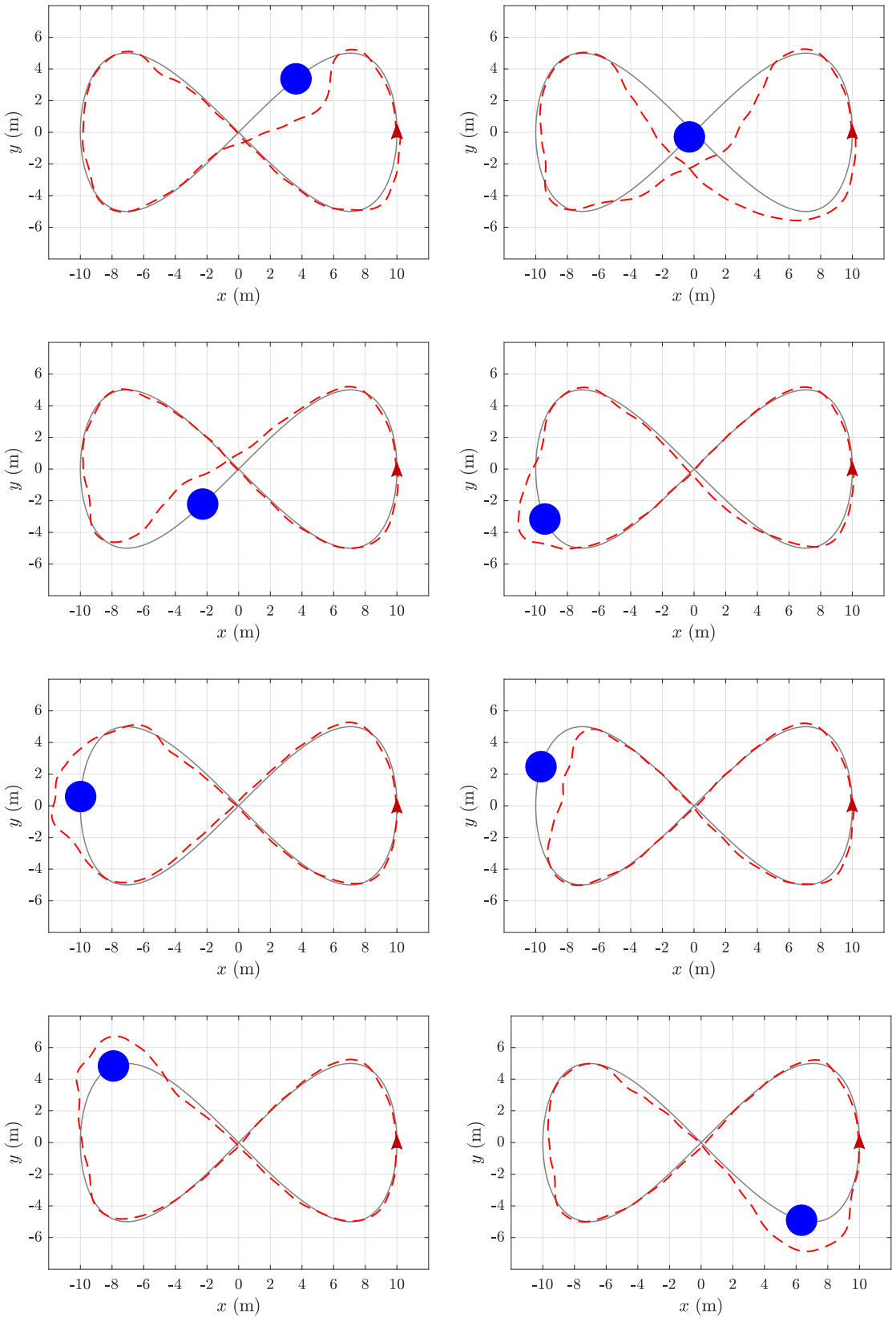

FIGURE 8.12: Simulation results following the lemniscate path with obstacles centred on the path.

Table 8.5 presents the results obtained while following the lemniscate path with obstacles that are not centred on the path. This means, there is a preferred side to avoid these obstacles and sometimes, if the obstacle is too far from the path, no avoidance manoeuvring is needed. This table shows the same parameters of Table 8.4. The average path distance error is lower than the simulations of Table 8.4 since the avoidance manoeuvres are less demanding.

TABLE 8.5: Simulation results for one lap on the lemniscate path with obstacles not centred on the path.

γ_{obs}	$\mathbf{d_{obs}}$	$\overline{\mathbf{d}}$ (m)	time (s)	$\overline{\|\mathbf{v}\|}$ ($^m/_s$)
1.2	2	0.1905	48.8	1.2511
1.4	1.5	0.2216	53.2	1.1528
1.6	1	0.3969	56.5	1.0923
1.8	0.5	0.3106	57	1.0769
2	-0.5	0.2834	58	1.0976
2.2	-1	0.1968	46.9	1.3265
2.4	-1.5	0.1936	48.2	1.2884
2.6	-2	0.1999	50.4	1.2290
2.8	-1.5	0.1929	52.7	1.1871
3	-1	0.1815	51.2	1.2135
3.2	-0.5	0.3120	55.4	1.1410
3.4	0.5	0.3126	50.2	1.2032
3.6	1	0.2720	53	1.1431
3.8	1.5	0.2014	56.1	1.1087
4	2	0.1699	52	1.1908
4.2	1.5	0.5201	54	1.1981
4.4	1	0.2124	55.3	1.1235
4.6	0.5	0.5707	57.5	1.0987
4.8	-0.5	0.3685	57.6	1.0758
5	-1	0.3215	56.3	1.0930
5.2	-1.5	0.2253	53.9	1.1476
5.4	-2	0.1513	49.7	1.2247
5.6	-1.5	0.2128	51.5	1.1769
5.8	-1	0.2836	50.6	1.2094

The trajectory followed by the vehicle in some of the simulations of Table 8.5 is shown in Fig. 8.13. These simulations correspond to the ones with the obstacle placed at γ_{obs} = 1.4, γ_{obs} = 1.6, γ_{obs} = 1.8, γ_{obs} = 2.2, γ_{obs} = 2.6, γ_{obs} = 3.8, γ_{obs} = 4.4 and γ_{obs} = 5.6, respectively. In all these simulations the agent chooses the correct side of the obstacle to perform the avoidance manoeuvre. Moreover, if the obstacle is out of the vehicle's route, as is the case in the simulation with γ_{obs} = 2.6 (third-left in Fig. 8.13), the vehicle remains in the path. It is important to remind that, due to the zones defined in Fig. 8.5, ideally, the vehicle is required to remain at least at 1.75m away from the center of the obstacle, and less than 1.5m is considered a collision. Thus, even with the obstacles placed at 1.5m of the path, as it is the case of the simulations with γ_{obs} = 1.4 (first-left), γ_{obs} = 3.8 (third-right) and γ_{obs} = 5.6 (fourth-right), the agent needs to perform an avoiding manoeuvre to prevent the collision.

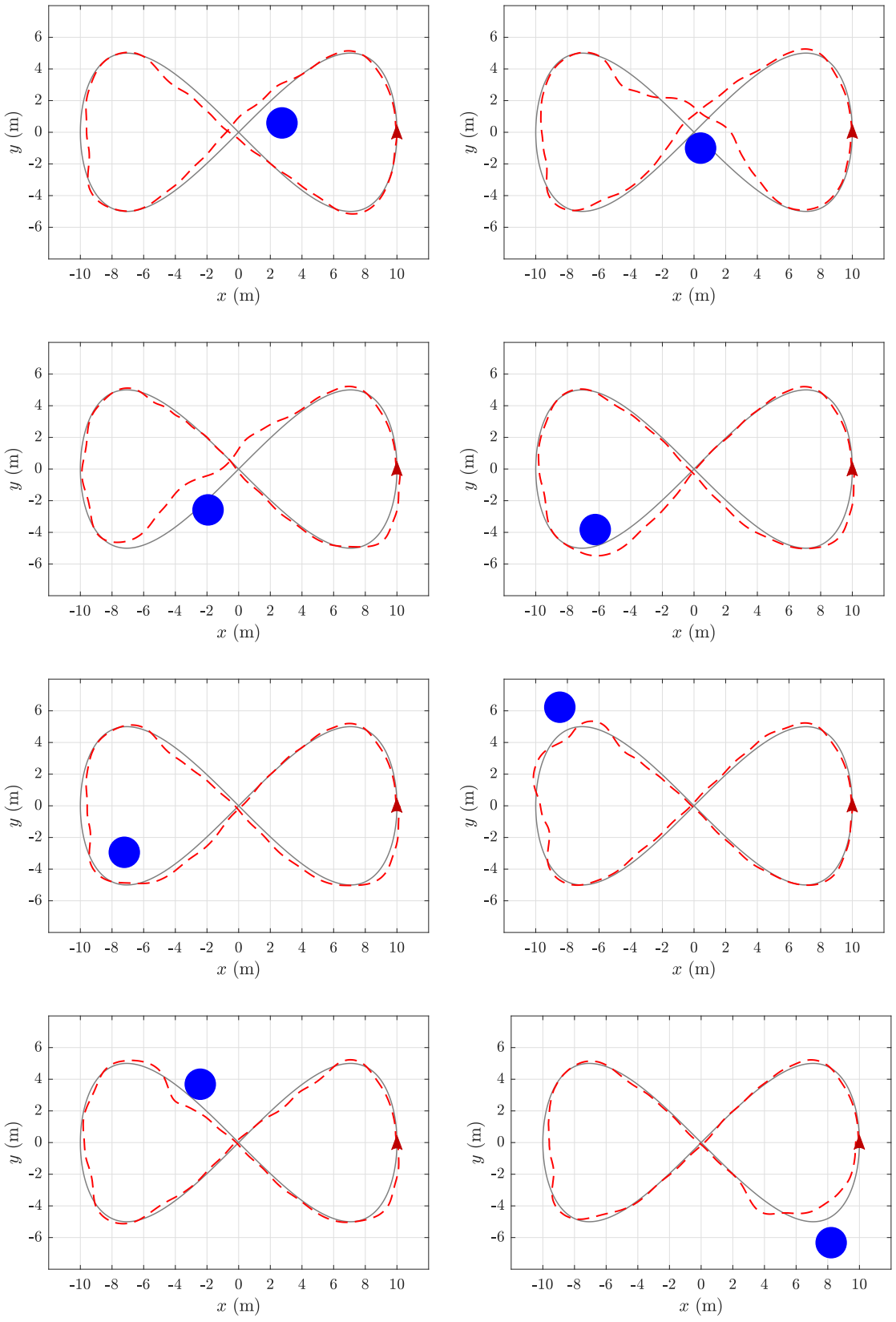

FIGURE 8.13: Simulation results following the lemniscate path with obstacles not centred on the path.

In the simulation results presented in Figs. 8.12 and 8.13, the agent always choose the most convenient side of the obstacle to avoid it. Nevertheless, occasionally, the agent may also choose the wrong side of the obstacle, as it is evident in the simulation of Fig. 8.14. In this simulation, the obstacle (center) is located on the left of the path at 1.5m, however, the agent unnecessarily avoids it taking the longest route. This behaviour may be caused by the fact that when the vehicle sees the obstacle for the first time, it is at its right. Therefore, as the agent cannot predict the upcoming curve, it starts the avoidance manoeuvre by the left side of it. Nevertheless, even in the scenario of Fig. 8.14, this behaviour only happens in very few occasions. On the other hand, the agent can also drive the vehicle to a collision, as shown in Fig. 8.15. Colliding is really a very rare event and, it is more likely to happen on the starting part of the path where the vehicle is still accelerating.

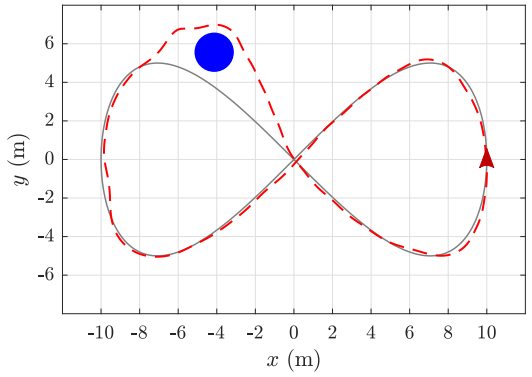

FIGURE 8.14: Trajectory on the xy plane of a simulation where the agent avoids the obstacle by the wrong side.

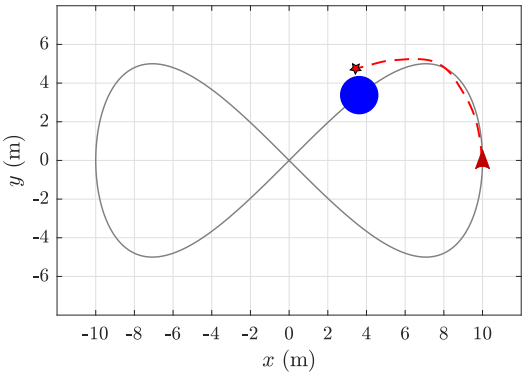

FIGURE 8.15: Trajectory on the xy plane of a simulation that ended in a collision.

8.6.2 Spiral Path

In this section the PF+OA approach is tested with a spiral path, a different path from the used to train both the PF agent and the agent developed in this chapter. The spiral path is defined in Eq. (6.18), where A is set to 1.25 and γ ranges from 0 to 3π. Again, the vehicle starts at the initial point on the path, oriented tangentially to it.

Table 8.6 compares the path following performance of the PF agent with the performance of the PF+OA approach following the spiral path with no obstacles. Just as the results of the lemniscate path, the PF agent achieves slightly better results. Nevertheless, the PF+OA approach still achieves remarkable results.

TABLE 8.6: Simulation results for one lap on the spiral path with no obstacles.

	$\overline{\mathbf{d}}$ (m)	time (s)	$\overline{\|\mathbf{v}\|}$ ($^m/_s$)
PF Agent	0.1149	47.4	1.2457
OA Agent + PF agent	0.1304	46	1.2444

The results obtained with the proposed approach when following the stated spiral path with different obstacle positions is shown in Table 8.7. The first 7 simulations correspond to simulations with obstacles centred on the path. In these simulations the approach achieves an average cross-track error of around 0.3-0.4m, very similar to the results obtained with the lemniscate path.

TABLE 8.7: Simulation results for one lap on the spiral path with different obstacle positions.

γ_{obs}	$\mathbf{d_{obs}}$	$\overline{\mathbf{d}}$ (m)	time (s)	$\overline{\|\mathbf{v}\|}$ ($^m/_s$)
4	0	0.3699	52.4	1.1412
4.8	0	0.3901	51.2	1.1575
5.6	0	0.3768	49.3	1.1954
6.4	0	0.3741	51.2	1.1521
7.2	0	0.4036	51	1.1585
8	0	0.3216	52.1	1.1347
8.8	0	0.3606	50.2	1.1632
4	2	0.1835	48.8	1.1696
4.8	1.5	0.2348	46	1.2293
5.6	1	0.3529	43.6	1.2823
6.4	0.5	0.3860	44.4	1.2560
7.2	-0.5	0.3578	48.7	1.2037
8	-1	0.2496	49.2	1.1803
8.8	-1.5	0.1779	48.8	1.1951

Some of the trajectories in the xy plane of the simulations presented in Table 8.7 are shown in Fig. 8.16. From left to right and top to bottom, these simulations correspond to $\gamma_{obs} = 4, d_{obs} = 0$; $\gamma_{obs} = 5.6, d_{obs} = 0$; $\gamma_{obs} = 8, d_{obs} = 0$; $\gamma_{obs} = 4, d_{obs} = 2$; $\gamma_{obs} = 6.44, d_{obs} = 0.5$ and $\gamma_{obs} = 8.8, d_{obs} = -1.5$, respectively. Note that in a spiral path the preferred side to avoid the obstacle is the outside of the curve. However, if the obstacle is placed shifted to the left of the path from the vehicle's point of view, the agent will avoid the obstacle from the inside of the curve, as is the case of the simulation with $\gamma_{obs} = 6.44 - d_{obs} = 0.5$ (third-left in Fig. 8.16).

With the presented results it is shown how the proposed approach for path following and obstacle avoidance is able to follow a path different from the trained one while being able to avoid obstacles at different positions of the path. Thus, it is shown how a generalized solution of the problem for different path shapes is achieved.

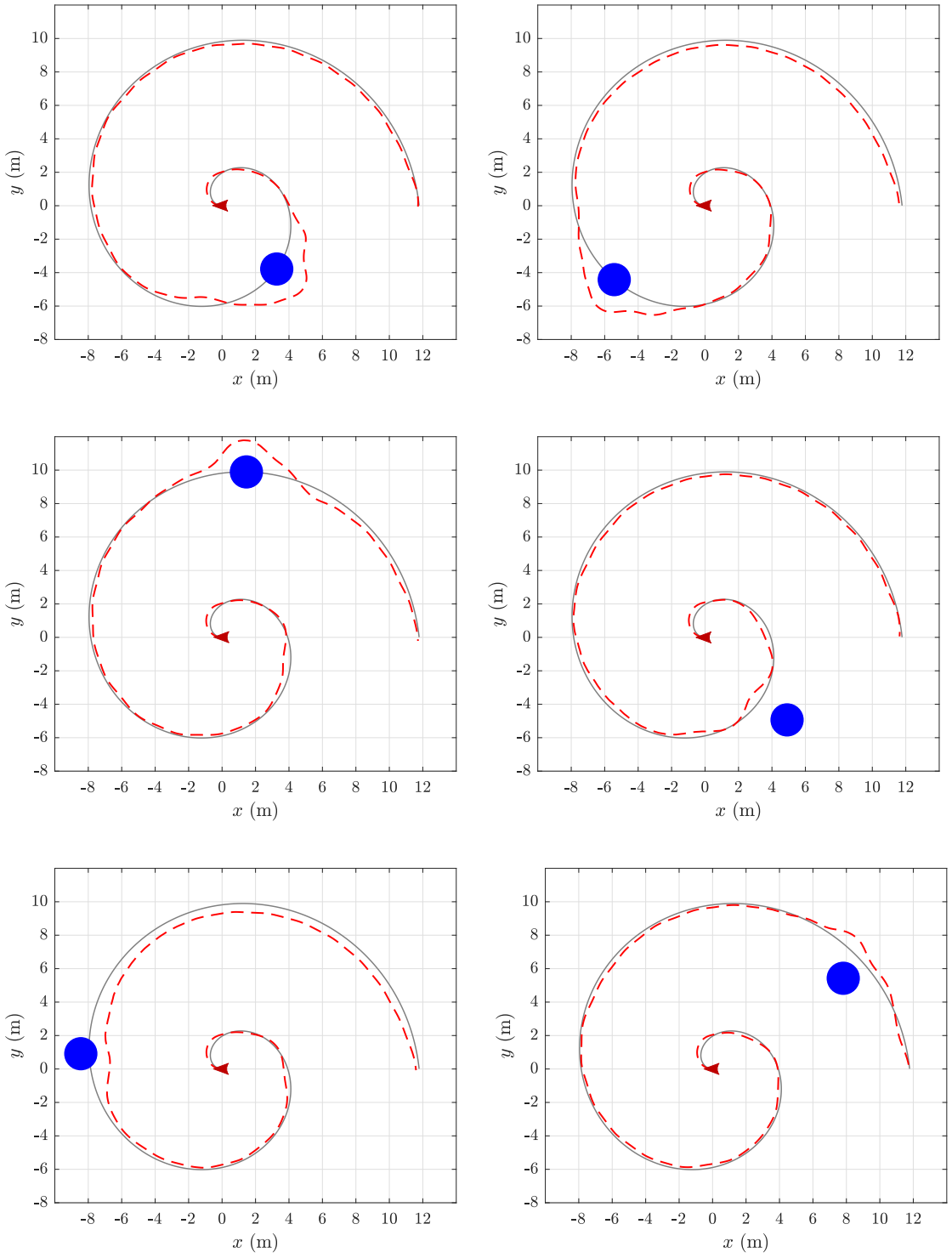

FIGURE 8.16: Simulation results following the spiral path with different obstacle positions.

Part IV

Concluding Remaks

Chapter 9

Conclusions

In this PhD thesis a Guidance, Navigation and Control system for a quadrotor vehicle was implemented. This dissertation has proposed contributions to different problems that appear when implementing a GNC system, studying and developing different approaches combining both control theory and machine learning theory.

A path following and obstacle avoidance approach based on two deep reinforcement learning agents was proposed as a solution to the main objective addressed in the thesis. The resulting approach is capable of accurately following a predefined path by adapting the velocity of the vehicle to the shape of the path. Furthermore, it is capable of avoiding obstacles that may appear in the vehicle's route, by replanning the route online. The proposed approach is adaptable to different paths and conditions without requiring any additional tuning of its parameters. The proposed path following approach was implemented in the real experimental platform with successful results.

To fulfil the main objective, several tasks were accomplished, including the literature review, the development of a realistic simulation benchmark, the setup of the experimental platform, the implementation of an obstacle detection system and the study, implementation and improvement of state-of-the-art path following algorithms, among others. Next sections summarize the work and contributions produced in this dissertation, as well as the future research lines and perspectives.

9.1 Summary and Contributions

Chapter 2:

This chapter proposes the structure of the Guidance, Navigation and Control system that is implemented in this thesis. The elements of this structure and their communication flow are described in detail. Next, a comprehensive literature review on the Control, Navigation and Guidance fields focused on multirotor vehicles is carried on.

The concept of path following is defined, which according to the reviewed bibliography, results in a smoother convergence to the path, less demand of control effort, a stronger robustness and other advantages over traditional trajectory tracking. Several control-oriented, geometrical and learning-based techniques and approaches for solving the path following problem are reviewed. A qualitative comparison is given. The state estimation problem is described and the most common solutions for this type of UAV are referred. The main perception problems for multirotor vehicles are reviewed with special emphasis to the obstacle detection problem. To end up, the most common and proficient path planning techniques are described and several state-of-the-art approaches are detailed, focussing on the reactive path planning methods for UAVs.

Related publications:

Part of this chapter was published in the following journal paper [144]:
Rubí, B., Pérez, R. & Morcego, B. A Survey of Path Following Control Strategies for UAVs Focused on Quadrotors. *Journal of Intelligent & Robotic Systems* 98, 241–265 (2019).

Chapter 3:

This chapter is focused on the multirotor platform employed in this thesis. The actual experimental platform, including the Asctec Humming vehicle, the on-board PCs and the ground station, is described in detail. A complete mathematical model of the multirotor is derived. This model includes the body dynamics of the vehicle, the dynamics of the rotors and the motors, the control mixing block and other effects such as gyroscopic effects, drag forces and forces generated by wind disturbances. Moreover, a parameter identification with the real vehicle is carried on. A PID-based autopilot controller is designed, tuned and implemented in the actual platform to control the inner dynamics of the vehicle. The RotorS platform, a multirotor simulation environment build in Gazebo/ROS, is modified to add to the Asctec Hummingbird vehicle a model of the sensors that are present in the actual platform as well as an emulator of the R/C transmitter of the ground station.

Path-Flyer, a freely available and open benchmark for the simulation and comparison of path following algorithms on a quadrotor environment is developed in this thesis. It has a complete and validated model of the Asctec Hummingbird Quadrotor. Wind disturbance and noise on the measured states are modelled and can be customized. Two path following algorithms, Non-linear Guidance Law and Carrot-Chasing, and their adaptive versions are implemented. Path following algorithms can be tested in diverse simulation scenarios. A user interface helps the user to modify test conditions and to explore simulation results. It is modular and programmable, meaning that new path following algorithms and/or reference paths can be incorporated with ease. Also, a comparison of the Path-Flyer results with real experimental results is provided to prove the validity of the benchmark.

Related publications:

Part of this chapter was published in the following conference paper [146]:
Rubí, B., Ruiz, A., Pérez, R. & Morcego, B. Path-Flyer: A Benchmark of Quadrotor Path Following Algorithms, in *2019 15th IEEE International Conference on Control and Automation (ICCA)* (2019).

Chapter 4:

Two control-oriented algorithms (*Backstepping* and *Feedback Linearisation*) and two geometric algorithms (*Nonlinear Guidance Law* and *Carrot-Chasing*) are implemented in this chapter to solve the path following problem on a quadrotor vehicle. Additionally, the geometric algorithms are adapted to cope with three-dimensional paths. A detailed description of the derivation of the algorithms, their control structure and their algorithmic implementation is given. Finally, a thorough comparison of these four path following algorithms is carried out. These algorithms are evaluated in simulation in different conditions such as in steady state regime, in transient regime with different initial conditions, with wind disturbances and in realistic conditions with sensor noise and a realistic wind profile. From these simulation results, different parameters and indicators are obtained.

Simulation results reveal that the *Backstepping* algorithm achieves the best performance in terms of path distance and yaw error as well as the best behaviour out of the path and at high velocities. However, it results in a very high computational effort and a significant control effort. Meanwhile, the *3D Carrot-Chasing* algorithm, in spite of its worse path distance performance, turns out to be easier to implement on any type of path, requires less state information and it results in a lower computational and control effort. Therefore, the author considers that, among the state-of-the-art algorithms implemented in this chapter, these two are the best to solve the PF problem. The choice between them would depend on the problem requirements.

Related publications:

Part of this chapter was published in the following journal paper [144]:
Rubí, B., Pérez, R. & Morcego, B. A Survey of Path Following Control Strategies for UAVs Focused on Quadrotors. *Journal of Intelligent & Robotic Systems* 98, 241–265 (2019).

Chapter 5:

Geometrical algorithms provide a considerably good path following performance with low computational effort and they are easy to implement. *NLGL* and *Carrot-Chasing* algorithms only have one parameter to tune. However, their control parameter depend on external parameters such as the velocity of the vehicle or the path's shape.

In this chapter, the effect of the scalar parameter L on the performance of the *NLGL* algorithm when it is applied to a quadrotor vehicle is analysed. The mean absolute path distance error is used as a performance indicator. The optimal value of L is analysed and it is shown to change depending on the vehicle's velocity and the path radius. From these analysis, an adaptive law by means of a neural network is developed. This NN computes the optimal value of L from the vehicle's velocity and the value of the path radius. An algorithm is developed to find the most restrictive radius on the path on a defined anticipation distance window. The anticipation distance window depends on the vehicle's velocity. Then, a velocity reduction term is added to the *Adaptive NLGL* to decrease the speed when the radius of a curve is too small to be followed by the vehicle at the current velocity. An stability analysis of the proposed approach is given.

Simulation results compare the performance of the regular *NLGL*, the proposed *Adaptive NLGL* and the *Adaptive NLGL* with the velocity reduction term with two different reference

paths: a lemniscate path and a spiral 3D path. The results show that the *Adaptive NLGL* achieves better performance than the *NLGL*. Also, adding the velocity reduction term, makes the vehicle follow with higher accuracy the smaller radius curves.

Next, the same process is followed to develop an *Adaptive Carrot-Chasing* approach with the use of a NN. This NN computes the optimal value of δ from the vehicle's velocity and the value of the path radius. The algorithm to find the most restrictive radius on the path on the defined anticipation window is used, and a velocity reduction term is added in the *Adaptive Carrot-Chasing* approach.

To end up, simulation results compare the performance of the *Adaptive NLGL* and the *Adaptive Carrot-Chasing* in realistic conditions (including sensor noise and a realistic wind disturbance) while following a lemniscate path and a spiral 3D path. Both approaches presented similar performance, improving the performance of their standard versions. Furthermore, it is important to mention that the main advantage of the presented approaches is that, as opposed to their standard versions, it is not necessary to tune their parameters when the reference path or the cruise velocity is changed.

Related publications:

Part of this chapter was published in the following conference paper [143]:
Rubí, B., Morcego, B & Pérez, R. Adaptive Nonlinear Guidance Law Using Neural Networks Applied to a Quadrotor, in *2019 15th IEEE International Conference on Control and Automation (ICCA)* (2019).

Chapter 6:

In this chapter, a deep reinforcement learning algorithm, the *Deep Deterministic Policy Gradient*, is proposed to solve the path following problem in a quadrotor vehicle. The path following control computes the references for the velocity, the altitude and the angle in the z axis that are then tracked by the autopilot controller. Three different *DDPG* approaches with different behaviours are presented. The first approach solves the PF problem only with information about the instantaneous position and angle errors. The second approach adds information about the upcoming path. Both approaches work at constant velocity. The third approach permits the agent to compute the optimal vehicle's velocity that adapts better to the shape of the path, according to the defined agent reward.

Each of the proposed agents arises as an improved version of the previous one, that is one of the main strengths of the methodology used in this work. The main structure, common in the three approaches, permits the incorporation of new functionalities (such as having anticipation to curves or adapting the vehicle's velocity) without changing the core of the agent. This is very promising since it means that new functionalities (e.g. wind disturbance rejection) could be straightforwardly integrated to the agent without altering the rest of the functionalities.

The agents were programmed in python using the tensorflow library. The designed training framework integrates the python script with Gazebo-ROS and uses RotorS, a the realistic multirotor simulator. This simulator includes a model of the Asctec Hummingbird, the quadrotor used in the experimental platform. Models of the real sensors of our platform were included in the simulator. The first and second agents were trained with lemniscate

paths of fixed radius. They were also trained with different initial conditions to improve their performance in the experimental results. In order to learn the policy of the velocity action with different path's radius, the third agent was trained with asymmetrical lemniscates and changing the radius on each episode. The three agents were trained assuming ground truth measurements. The main advantage of training the agents in ROS is that it facilitates the transition from the simulator to the real plant. Furthermore, since ROS is a standard platform in the robotics field, it is supported by a large community, which can be very useful. The only concern to consider when training in ROS is that, since simulations are made real-time, it may become a time-consuming process.

The three agents were tested in simulations in the RotorS environment with realistic models of the sensors. They were evaluated with a lemniscate path and with a spiral path. Then, the agents were also tested in real experiments with the Asctec Hummingbird quadrotor following the same paths as in simulation. Even thought the agents were able to follow the pre-established paths correctly in the first experiments that were carried out, they performed worse than expected. The authors concluded that this behaviour was due to the small errors of the simulated model, which affected mainly to the yaw dynamics. To improve the performance of the agents a new parameter (angle correction constant, k_a) was included. This parameter scales the *yaw* action of the agent. And permits to modify the dynamics of the agent to cope with the model's discrepancy. Training the agents experimentally was dismissed since it can be harmful for the plant due to the unexpected behaviour of the vehicle. Furthermore, it was observed that training with noisy signals was unfruitful.

The agents were tested experimentally including the angle correction constant, which improved significantly their performance. In the lemniscate path, the *Agent 2* achieved the best performance in terms of average cross-track error ($0.114m$), but the *Agent 3* exhibited a similar distance error ($0.168m$) while being able to significantly increase the vehicle's velocity. In the spiral path, the *Agent 3* stands out over the other approaches by achieving the lowest average cross-track error ($0.223m$) while travelling at higher velocities. In conclusion, the experimental results show that the agents are able to successfully follow the spiral path, a different path from the one that they were trained with. And thus, it is proved that the proposed approach is able to find a generalized solution for the path following problem with adaptive velocity.

Related publications:

Part of this chapter was published in the following conference paper [147]:
Rubí, B., Morcego, B & Pérez, R. A Deep Reinforcement Learning Approach for Path Following on a Quadrotor, in *2020 European Control Conference (ECC)* (2020).

Part of this chapter was published in the following journal paper [145]:
Rubí, B., Morcego, B. & Pérez, R. Deep Reinforcement Learning for Quadrotor Path Following with Adaptive Velocity. *Autonomous Robots*, doi: 10.1007/s10514-020-09951-8, (2020)

Part of this chapter was sent for being published as part of a chapter of the *Deep Reinforcement for Autonomous Systems* Springer book:

Rubí, B., Morcego, B. & Pérez, R. Deep Reinforcement Learning for Quadrotor Path Following and Obstacle Avoidance.

Chapter 7:

This chapter proposes a solution for the obstacle detection problem. The best sensor system to detect obstacles is studied for the particular addressed in this thesis. The selected obstacle detection and localization system consists on a frontal LIDAR sensor. Details of the Leddar VU8 LIDAR sensor are given. The sensor is installed in the real platform. A realistic model is programmed in the RotorS/Gazebo environment. This model includes a graphical part and a mechanical part of the LIDAR as well as a script that generates the measurements of the sensor including noise. To end up, an algorithm for eliminating the possible ground detections is developed. The algorithm takes into account the orientation and altitude of the vehicle and the orientation of each LIDAR beam to estimate the distance to the ground.

Chapter 8:

The aim of this chapter is to develop a deep reinforcement learning solution for the path following and obstacle avoidance problem. To this end, a reactive obstacle avoidance approach based on the application of the *Deep Deterministic Policy Gradient* algorithm is proposed. This approach communicates with the PF agent developed in Chap. (6). It receives the data of the obstacle detection system, which consists on a LIDAR sensor treated to eliminate ground detections. If an obstacle on the vehicle's route is detected the OA agent generates a modification over the reference path to avoid a possible collision. The modification of the path is made in the form of a distance offset of the original reference path. Specifically, the OA agent modifies path distance error state of the PF agent. The information of the modified reference path is transmitted to the path following agent, whose mission is to follow this new path.

This chapter explores different solutions to solve the problem and a detailed explanation is given on how the action, the state vector and reward functions are obtained. The main issue addressed in this work is caused by the fact that the LIDAR sensor has a limited field-of-view of 48° which compromises the detection of an obstacle. That is, as soon as the vehicle starts avoiding an obstacle it will disappear from the LIDAR's field-of-view. Since the agent only uses the current state to compute the action, when this happens, the agent considers that it has already avoided the obstacle and moves the vehicle back to the path provoking a collision. This chapter proposes a solution to this problem that consists on including information of the historical data of the LIDAR sensor in the state vector. This historical information is generated by integrating the inverse of the range data provided by each LIDAR beam. If no obstacles are near, this state is progressively decreased until it reaches zero. In this way, if the integral LIDAR state is positive, it means than an obstacle has been recently seen. The closer the obstacle is to the vehicle, the larger this state will be. In the proposed final approach, this state is divided in two; one for each side of the LIDAR sensor, providing more information to the agent to know in what side of the vehicle the obstacle is.

The OA agent is trained in the RotorS/Gazebo environment with the model of the Asctec Hummingbird vehicle. The agent is programmed in pyhton by using the tensorflow library to generate and train the neural networks. A cylindrical obstacle of 1m radius is used in both the training phase and the tests. The agent is trained by following a half asymmetrical lemniscate path with random radius and with obstacles placed randomly on the path in a range of $\pm 2m$ of distance to the path.

The path following and obstacle avoidance approach is tested in the RotorS environment by following a lemniscate path and a spiral path with obstacles at different positions. The simulations results show how it successfully avoids obstacles while following a path. The OA agent is able to choose the side to better perform the avoidance manoeuvre depending on the obstacle and vehicle positions and the path's shape. The agent takes into account the vehicle's velocity and the path's shape to generate this manoeuvre. The path following performance without obstacles is very similar to the one obtained by the PF agent. In conclusion, a generalized solution for the path following and obstacle avoidance problem for different path's shapes and obstacle positions is achieved.

Related publications:

Part of this chapter was submitted to the Journal of Intelligent & Robotics Systems:
Rubí, B., Morcego, B. & Pérez, R. Quadrotor Path Following and Reactive Obstacle Avoidance with Deep Reinforcement Learning. *Journal of Intelligent & Robotic Systems.*

Part of this chapter was sent for being published as part of a chapter of the *Deep Reinforcement for Autonomous Systems (Studies in Computational Intelligence series)* 2021 Springer book:
Rubí, B., Morcego, B. & Pérez, R. Deep Reinforcement Learning for Quadrotor Path Following and Obstacle Avoidance.

9.2 Perspectives and Future Work

Path Following with Deep Reinforcement Learning:

In Chap. (6) a deep reinforcement learning approach to solve the path following problem for a quadrotor vehicle is proposed. This approach is validated experimentally with successful results. The DRL agent is able to accurately follow the reference path by adapting the vehicle's velocity. Nevertheless, a strange pattern was observed in the *Agent 3* while performing counter-clockwise curves at high velocities (Fig. 6.18). This behaviour was attributed to the design of the training environment. That is, since the agent was trained with half asymmetrical lemniscates, the first curve of the path (counter-clockwise) is followed in the transient part of the experiment (velocity is still increasing). Therefore, the agent ended up training the clockwise curves at faster velocities than the counter-clockwise ones. This fact highlights the importance of having not only a proper structure and parametrization of the agent, but also a rich, complete and adequate training framework. The solution to this issue would be to train with a different path that permits the agent to learn both curves at different velocities. Future work is to study the effect of the trained path in the

performance of the agent and to find the best training environment to exploit the benefits of the agent.

The proposed approach is designed to solve the path following problem with adaptive velocity. However, due to the nature of the deep reinforcement learning applications, this approach presents an initial framework that can be upgraded to solve more challenging problems such as wind disturbance rejection, adaptability to different models, resilient control, etc. For instance, an estimation of the wind disturbance forces could be introduced as new states of the model. Considering the importance of the disturbance generated by the wind in outdoors environments, this could substantially improve the performance of the approach. Another common issue when flying in experimental environments are caused by the disruptions of the magnetic field measured by the magnetometer sensor, which are generated by nearby ferric objects. These disruptions degrade the estimation of the *yaw* angle. A deep learning approach could be used to detect possible disruptions on the magnetic field, and this information could be sent to the PF agent to enhance the performance. Therefore, future work is to improve the presented agent to make it capable of solving other challenging problems enhancing the resilience of the approach.

Obstacle Avoidance with Deep Reinforcement Learning:

The obstacle avoidance approach developed in Chap. (8) was trained and tested in RotorS/Gazebo, a realistic multirotor simulation environment with a model of the Asctec Hummingbird vehicle. Future work is to implement the resulting approach in the real experimental platform. Both the PF and OA agents were already integrated in ROS and the LIDAR system was already installed in the real platform. Therefore, next step would be to build a safe experimental framework containing obstacles, as well as creating a safety protocol to avoid possible collisions.

The path following and obstacle avoidance approach proposed in this thesis is based on the combination of the path following agent developed in Chap. (6) and the reactive obstacle avoidance agent developed in Chap. (8). The main advantage of dividing the problem in two deep reinforcement learning agents is that it permits to reduce the complexity of the problem and, therefore, the total time needed to train the DRL system is reduced as well. Nevertheless, reducing the complexity of the problem may result in a degradation of the performance and potential of the approach. Future work would be to explore a solution for the path following and obstacle avoidance problem based on one unique DRL agent. To this end, the state information and reward functions of both the developed PF and OA agents could be combined to generate the new PF and OA agent. This approach should be compared with the 2-agents approach presented in this thesis, evaluating the advantages and disadvantages of each one.

The main issue addressed in the process of developing the OA agent (Chap. (8)) was derived for both the narrow field-of-view of the employed obstacle detection sensor and the inability of the *DDPG* algorithm of using historical state information. This issue was solved by introducing two states that integrate the inverse of the LIDAR detections providing some kind of historical information of the LIDAR data. Future work could be to explore other solutions to provide the agent with information of the obstacle. This

solutions may include estimating the position of the obstacle, generating occupancy grid maps or defining other states that supply detailed obstacle's information to the agent. Another solution that could be explored consists on modifying the structure of the *DDPG* algorithm to give the algorithm the ability of using historical data. This can be done by using recurrent neural networks, for instance *LSTM* networks, which use a sequence as a data input.

The proposed obstacle avoidance approach was trained and tested with a predefined type of obstacles. That is, cylindrical objects with a fixed size and shape. Future work would be to obtain an approach that achieves a generalized solution for different obstacle shapes. That may include providing more information of the obstacle shape to the agent as well as training it with different types of obstacles. Moreover, three-dimensional avoiding manoeuvres could be explored too.

Improving the Deep Deterministic Policy Gradient Algorithm:

In this thesis the Deep Deterministic Policy Gradient was implemented for solving both the path following problem and the obstacle avoidance problem. Future work is to propose modifications to this deep reinforcement learning algorithm to improve its performance and functionality for the particular problems addressed here. This improvements may include implementing an approach for a wise selection of the experience replay that is used for training. That is, DDPG is trained with random samples of transitions that are saved in an experience replay buffer. Instead of that, the training transitions could be selected according to some criteria, exploiting the ones that provide more information. Another improvement that may be explored, consists on training an agent with different vehicles flying at the same time, allowing for a faster training convergence. That is, the transition tuples of each flying vehicle could be saved in the experience replay buffer, and then, the agent would be trained from this set of transitions. Moreover, another example of a potential improvement to be studied is the incorporation of LSTM neural networks to the algorithm, allowing for interpretation of sequential data.

Path-Flyer Benchmark:

Path-Flyer (Section 3.4), an open-source simulation tool of quadrotor path following algorithms was developed in this thesis. This benchmark permits the incorporation of new path following algorithms with ease. Nevertheless, these algorithms must follow a Separated, Guidance and Control (SGC) structure. That is, algorithms that require an autopilot to control the inner dynamics, as it is the case of the geometrical PF algorithms that are already implemented in this platform. Future work is to modify the benchmark to facilitate the possibility to incorporate path following algorithms that present an Integrated Guidance and Control (IGC) structure. In other words, PF algorithms that are also in charge of controlling the inner dynamics of the vehicle, such as the *Backstepping* and *Feedback Linearisation* approaches studied in Chap. (4). In this way, the benchmark would be prepared to include any type of path following algorithm, improving the robustness and utility of the tool. Furthermore, since the last version of MatLab already allows for the implementation of deep reinforcement learning agents, other future work is to include the DRL path following approach developed in this thesis in the Path-Flyer benchmark.

References

[1] A. P. Aguiar and J. P. Hespanha. Trajectory-tracking and path-following of underactuated autonomous vehicles with parametric modeling uncertainty. *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, 52(8):1362–1379, Aug 2007. doi: 10.1109/TAC.2007.902731.

[2] A. Pedro Aguiar, Joao P. Hespanha, and Petar V. Kokotovic. Performance limitations in reference tracking and path following for nonlinear systems. *AUTOMATICA*, 44(3): 598–610, MAR 2008. ISSN 0005-1098. doi: {10.1016/j.automatica.2007.06.030}.

[3] Adeel Akhtar, Steven L. Waslander, and Christopher Nielsen. Path following for a quadrotor using dynamic extension and transverse feedback linearization. In *2012 IEEE 51ST ANNUAL CONFERENCE ON DECISION AND CONTROL (CDC)*, IEEE Conference on Decision and Control, pages 3551–3556, 2012.

[4] Adeel Akhtar, Steven L. Waslander, and Christopher Nielsen. Fault Tolerant Path Following for a Quadrotor. In *2013 IEEE 52ND ANNUAL CONFERENCE ON DECISION AND CONTROL (CDC)*, IEEE Conference on Decision and Control, pages 847–852, 2013.

[5] V. S. Akkinapalli, P. Niermeyer, B. Lohmann, and F. Holzapfel. Adaptive nonlinear design plant uncertainty cancellation for a multirotor. In *2016 INTERNATIONAL CONFERENCE ON UNMANNED AIRCRAFT SYSTEMS (ICUAS)*, pages 1102–1110, June 2016. doi: 10.1109/ICUAS.2016.7502555.

[6] Abdulla Al-Kaff, Fernando García, David Martín, Arturo de la Escalera, and Jose M. Armingol. Obstacle detection and avoidance system based on monocular camera and size expansion algorithm for uavs. *SENSORS (Basel, Switzerland)*, 17, 2017. doi: 10.3390/s17051061.

[7] D. Alejo, J. A. Cobano, G. Heredia, and A. Ollero. A reactive method for collision avoidance in industrial environments. *JOURNAL OF INTELLIGENT & ROBOTIC SYSTEMS*, 84: 745 – 758, 2016. doi: 10.1007/s10846-016-0359-7.

[8] Kostas Alexis, Christos Papachristos, Roland Siegwart, and Anthony Tzes. Robust model predictive flight control of unmanned rotorcrafts. *JOURNAL OF INTELLIGENT & ROBOTIC SYSTEMS*, 81(3):443–469, Mar 2016. ISSN 1573-0409. doi: 10.1007/s10846-015-0238-7.

[9] Syed Ussama Ali, Raza Samar, M. Zamurad Shah, Aamer I. Bhatti, Khalid Munawar, and Ubaid M. Al-Sggaf. Lateral guidance and control of UAVs using second-order sliding modes. *AEROSPACE SCIENCE AND TECHNOLOGY*, 49:88–100, FEB 2016. ISSN 1270-9638. doi: {10.1016/j.ast.2015.11.033}.

[10] G. Ambrosino, M. Ariola, U. Ciniglio, F. Corraro, E. De Lellis, and A. Pironti. Path Generation and Tracking in 3-D for UAVs. *IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY*, 17(4):980–988, JUL 2009. ISSN 1063-6536. doi: {10.1109/TCST.2009.2014359}.

[11] Omead Amidi and Charles Thorpe. Integrated mobile robot control. In *MOBILE ROBOTS V*, volume 1388, 1991. doi: 10.1117/12.25494.

[12] Omead Amidi, Takeo Kanade, and Keisuke Fujita. A visual odometer for autonomous helicopter flight. *ROBOTICS AND AUTONOMOUS SYSTEMS*, 28(2):185 – 193, 1999. ISSN 0921-8890. doi: 10.1016/S0921-8890(99)00016-0. Intelligent Autonomous Systems (IAS-5).

[13] Roohul Amin, Li Aijun, and Shahaboddin Shamshirband. A review of quadrotor UAV: control methodologies and performance evaluation. *INTERNATIONAL JOURNAL OF AUTOMATION AND CONTROL*, 10(2):87–103, 2016. ISSN 1740-7516. doi: {10.1504/IJAAC.2016.076453}.

[14] Franz Andert and Florian Adolf. Online world modeling and path planning for an unmanned helicopter. *AUTONOMOUS ROBOTS*, 27(3):147–164, 2009. ISSN 1573-7527. doi: 10.1007/s10514-009-9134-y.

[15] Franz Andert, Florian-M. Adolf, Lukas Goormann, and Jörg S. Dittrich. Autonomous vision-based helicopter flights through obstacle gates. *JOURNAL OF INTELLIGENT & ROBOTIC SYSTEMS*, 57(1-4):259–280, January 2010. ISSN 0921-0296. doi: 10.1007/s10846-009-9357-3.

[16] Jorge Artieda, José M. Sebastian, Pascual Campoy, Juan F. Correa, Iván F. Mondragón, Carol Martínez, and Miguel Olivares. Visual 3-d slam from uavs. *JOURNAL OF INTELLIGENT & ROBOTIC SYSTEMS*, 55(4):299, 2009. ISSN 1573-0409. doi: 10.1007/s10846-008-9304-8.

[17] N. Aswini, E. Krishna Kumar, and S.V. Uma. Uav and obstacle sensing techniques – a perspective. *INTERNATIONAL JOURNAL OF INTELLIGENT UNMANNED SYSTEMS*, 6(1):32–46, 2018. doi: 10.1108/IJIUS-11-2017-0013.

[18] T. Baca, G. Loianno, and M. Saska. Embedded model predictive control of unmanned micro aerial vehicles. In *2016 21ST INTERNATIONAL CONFERENCE ON METHODS AND MODELS IN AUTOMATION AND ROBOTICS (MMAR)*, pages 992–997, Aug 2016. doi: 10.1109/MMAR.2016.7575273.

[19] A. Bachrach, R. He, and N. Roy. Autonomous flight in unknown indoor environments. *INTERNATIONAL JOURNAL OF MICRO AIR VEHICLES*, 1(4):217–228, 2009.

[20] Abdullah Başçi, Kaan Can, Kamil Orman, and Adnan Derdiyok. Trajectory tracking control of a four rotor unmanned aerial vehicle based on continuous sliding mode controller. *ELEKTRONIKA IR ELEKTROTECHNIKA*, 23(3):12–19, 06 2017. ISSN 1392-1215.

[21] Fethi Belkhouche. Reactive optimal uav motion planning in a dynamic world. *ROBOTICS AND AUTONOMOUS SYSTEMS*, 96:114 – 123, 2017. ISSN 0921-8890. doi: 10.1016/j.robot.2017.07.006.

[22] Brett Bethke, Mario Valenti, and Jonathan How. *Cooperative Vision Based Estimation and Tracking Using Multiple UAVs*, pages 179–189. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007. ISBN 978-3-540-74356-9. doi: 10.1007/978-3-540-74356-9_11.

[23] S. Bouabdallah and R. Siegwart. *Design and Control of a Miniature Quadrotor*, pages 171–210. Springer Netherlands, Dordrecht, 2007. ISBN 978-1-4020-6114-1. doi: 10.1007/978-1-4020-6114-1_6.

[24] S. Bouabdallah, P. Murrieri, and R. Siegwart. Design and control of an indoor micro quadrotor. In *2004 IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA)*, volume 5, pages 4393–4398 Vol.5, April 2004. doi: 10.1109/ROBOT.2004.1302409.

[25] J. Byrne, M. Cosgrove, and R. Mehra. Stereo based obstacle detection for an unmanned air vehicle. In *PROCEEDINGS 2006 IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA)*, pages 2830–2835, May 2006. doi: 10.1109/ROBOT.2006.1642130.

[26] CI Byrnes and A Isidori. Asymptotic Stabilization of Minimum Phase Nonlinear-Systems. *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, 36(10):1122–1137, OCT 1991. ISSN 0018-9286. doi: {10.1109/9.90226}.

[27] D. Cabecinhas, R. Cunha, and C. Silvestre. Rotorcraft path following control for extended flight envelope coverage. In *PROCEEDINGS OF THE 48TH IEEE CONFERENCE ON DECISION AND CONTROL, 2009 HELD JOINTLY WITH THE 2009 28TH CHINESE CONTROL CONFERENCE (CDC/CCC 2009)*, IEEE Conference on Decision and Control, pages 3460–3465, 2009. ISBN 978-1-4244-3872-3. doi: {10.1109/CDC.2009.5400665}.

[28] David Cabecinhas, Rita Cunha, and Carlos Silvestre. A Globally Stabilizing Path Following Controller for Rotorcraft With Wind Disturbance Rejection. *IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY*, 23(2):708–714, MAR 2015. ISSN 1063-6536. doi: {10.1109/TCST.2014.2326820}.

[29] J. C. Caicedo and S. Lazebnik. Active object localization with deep reinforcement learning. In *2015 IEEE INTERNATIONAL CONFERENCE ON COMPUTER VISION (ICCV)*, pages 2488–2496, Dec 2015. doi: 10.1109/ICCV.2015.286.

[30] E.F. Camacho and C. Bordons. *Model Predictive Control*. Advanced Textbooks in Control and Signal Processing. Springer London, 2004. ISBN 9781852336943.

[31] Hongda Chen, Kuochu Chang, and Craig S. Agate. UAV Path Planning with Tangent-plus-Lyapunov Vector Field Guidance and Obstacle Avoidance. *IEEE TRANSACTIONS ON AEROSPACE AND ELECTRONIC SYSTEMS*, 49(2):840–856, APR 2013. ISSN 0018-9251.

[32] Yang Chen, Jianhong Liang, Chaolei Wang, and Yicheng Zhang. Combined of lyapunov-stable and active disturbance rejection control for the path following of a small unmanned aerial vehicle. *INTERNATIONAL JOURNAL OF ADVANDES ROBOTIC SYSTEMS*, 14(2), 2017. doi: 10.1177/1729881417699150.

[33] Yang Chen, Jianhong Liang, Chaolei Wang, Yicheng Zhang, Tianmiao Wang, and Chenghao Xue. Planar Smooth Path Guidance Law for a Small Unmanned Aerial Vehicle with Parameter Tuned by Fuzzy Logic. *JOURNAL OF CONTROL SCIENCE AND ENGINEERING*, 2017:11, 2017. doi: {10.1155/2017/6712602}.

[34] YongBo Chen, JianQiao Yu, XiaoLong Su, and GuanChen Luo. Path Planning for Multi-UAV Formation. *JOURNAL OF INTELLIGENT & ROBOTIC SYSTEMS*, 77(1, SI): 229–246, JAN 2015. ISSN 0921-0296. doi: {10.1007/s10846-014-0077-y}.

[35] Yongbo Chen, Jianqiao Yu, Yuesong Mei, Yafei Wang, and Xiaolong Su. Modified central force optimization (MCFO) algorithm for 3D UAV path planning. *NEUROCOMPUTING*, 171:878–888, JAN 1 2016. ISSN 0925-2312. doi: {10.1016/j.neucom.2015.07.044}.

[36] Yin Cheng and Weidong Zhang. Concise deep reinforcement learning obstacle avoidance for underactuated unmanned marine vessels. *Neurocomputing*, 272:63 – 73, 2018. ISSN 0925-2312. doi: 10.1016/j.neucom.2017.06.066.

[37] Namhoon Cho, Youdan Kim, and Sanghyuk Park. Three-Dimensional Nonlinear Differential Geometric Path-Following Guidance Law. *JOURNAL OF GUIDANCE CONTROL AND DYNAMICS*, 38(12):2366–2385, DEC 2015. ISSN 0731-5090. doi: {10.2514/1.G001060}.

[38] Seungwon Choi, Suseong Kim, and H. Jin Kim. Inverse reinforcement learning control for trajectory tracking of a multirotor uav. *INTERNATIONAL JOURNAL OF CONTROL, AUTOMATION AND SYSTEMS*, 15(4):1826–1834, Aug 2017. doi: 10.1007/s12555-015-0483-3.

[39] Venanzio Cichella, Ronald Choe, S. Bilal Mehdi, Enric Xargay, Naira Hovakimyan, Isaac Kaminer, and Vladimir Dobrokhodov. A 3d path-following approach for a multirotor uav on so(3). In *IFAC PROCEEDINGS VOLUMES*, volume 46, pages 13 – 18, 2013. doi: 10.3182/20131120-3-FR-4045.00039.

[40] Venanzio Cichella, Isaac Kaminer, Vladimir Dobrokhodov, Enric Xargay, Ronald Choe, Naira Hovakimyan, A. Pedro Aguiar, and Antonio M. Pascoal. Cooperative Path Following of Multiple Multirotors Over Time-Varying Networks. *IEEE TRANSACTIONS ON AUTOMATION SCIENCE AND ENGINEERING*, 12(3):945–957, JUL 2015. ISSN 1545-5955. doi: {10.1109/TASE.2015.2406758}.

[41] Jonathan Courbon, Youcef Mezouar, Nicolas Guénard, and Philippe Martinet. Vision-based navigation of unmanned aerial vehicles. *CONTROL ENFINEERING PRACTICE*, 18(7):789 – 799, 2010. ISSN 0967-0661. doi: 10.1016/j.conengprac.2010.03.004. Special Issue on Aerial Robotics.

[42] K. Culligan, M. Valenti, Y. Kuwata, and J. P. How. Three-dimensional flight experiments using on-line mixed-integer linear programming trajectory optimization. In *2007 AMERICAN CONTROL CONFERENCE (ACC)*, pages 5322–5327, July 2007. doi: 10.1109/ACC.2007.4283101.

[43] Navid Dadkhah and Berenice Mettler. Control system design and evaluation for robust autonomous rotorcraft guidance. *CONTROL ENGINEERING PRACTICE*, 21(11):1488–1506, NOV 2013. ISSN 0967-0661. doi: {10.1016/j.conengprac.2013.04.011}.

[44] Héctor García de Marina, Yuri A. Kapitanyuk, Murat Bronz, Gautier Hattenberger, and Ming Cao. Guidance algorithm for smooth trajectory tracking of a fixed wing UAV flying in wind flows. In *2017 IEEE INTERNATIONAL CONFERENCE ON TOBOTICS AND AUTOMATION (ICRA)*, pages 5740–5745, 2017.

[45] A. H. J. de Ruiter and S. Owlia. Autonomous obstacle avoidance for fixed-wing unmanned aerial vehicles. *THE AERONAUTICAL JOURNAL*, 119(1221):1415–1436, 2015. doi: 10.1017/S0001924000011325.

[46] Y. Du, X. Zhang, and Z. Nie. A real-time collision avoidance strategy in dynamic airspace based on dynamic artificial potential field algorithm. *IEEE ACCESS*, 7:169469–169479, 2019.

[47] L.E. Dubins. On Curves of Minimal Length with a Constraint on Average Curvature, and with Prescribed Initial and Terminal Positions and Tangents. *AMERICAN JOURNAL OF MATHEMATICS*, 79(3):497–516, 1957. ISSN 0002-9327. doi: {10.2307/2372560}.

[48] C. Edwards and S. Spurgeon. *Sliding Mode Control: Theory And Applications*. Series in Systems and Control. Taylor & Francis, 1998. ISBN 9780748406012.

[49] J. Escareño, S. Salazar, H. Romero, and R. Lozano. Trajectory control of a quadrotor subject to 2d wind disturbances. *JOURNAL OF INTELLIGENT ROBOTIC & SYSTEMS*, 70(1):51–63, Apr 2013. ISSN 1573-0409. doi: 10.1007/s10846-012-9734-1.

[50] H. Eslamiat, Y. Li, N. Wang, A. K. Sanyal, and Q. Qiu. Autonomous waypoint planning, optimal trajectory generation and nonlinear tracking control for multi-rotor uavs. In *2019 18th European Control Conference (ECC)*, pages 2695–2700, 2019.

[51] T. Faulwasser and R. Findeisen. Nonlinear model predictive control for constrained output path following. *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, 61(4):1026–1039, April 2016. ISSN 0018-9286. doi: 10.1109/TAC.2015.2466911.

[52] Hassen Fourati and Djamel E. C. Belkhiat. *Multisensor Attitude Estimation : Fundamental Concepts and Applications*. Devices, Circuits, and Systems. CRC Press, 2016. ISBN 9781498745710.

[53] Fadri Furrer, Michael Burri, Markus Achtelik, and Roland Siegwart. *RotorS—A Modular Gazebo MAV Simulator Framework*, pages 595–625. Springer International Publishing, Cham, 2016. ISBN 978-3-319-26054-9. doi: 10.1007/978-3-319-26054-9_23.

[54] N. Gageik, P. Benz, and S. Montenegro. Obstacle detection and collision avoidance for a uav with complementary low-cost sensors. *IEEE ACCESS*, 3:599–609, 2015. doi: 10.1109/ACCESS.2015.2432455.

[55] Daniel C. Gandolfo, Lucio R. Salinas, Juan M. Toibero, and Alexandre Brandao. Path Following for Unmanned Helicopter: An Approach on Energy Autonomy Improvement. *INFORMATION TECHNOLOGY AND CONTROL*, 45(1):86–98, 2016. ISSN 1392-124X. doi: {10.5755/j01.itc.45.1.12413}.

[56] Carlos E. García, David M. Prett, and Manfred Morari. Model predictive control: Theory and practice-a survey. *AUTOMATICA*, 25(3):335 – 348, 1989. ISSN 0005-1098. doi: 10.1016/0005-1098(89)90002-2.

[57] A. Gautam, P. B. Sujit, and S. Saripalli. Application of guidance laws to quadrotor landing. In *2015 INTERNATIONAL CONFERENCE ON UNMANNED AIRCRAFT SYSTEMS (ICUAS)*, pages 372–379, June 2015.

[58] A. R. Gerlach, D. Kingston, and B. K. Walker. Uav navigation using predictive vector field control. In *2014 AMERICAN CONTROL CONFERENCE*, pages 4907–4912, June 2014. doi: 10.1109/ACC.2014.6859082.

[59] C. Goerzen and M. Whalley. Minimal risk motion planning: A new planner for autonomous uavs in uncertain environments. In *PROCEEDINGS OF THE AHS INTERNATIONAL SPECIALISTS MEETING ON UNMANED ROTORCRAFT*, pages 1–21, 2011. Tempe, AZ.

[60] C. Goerzen, Z. Kong, and B. Mettler. A survey of motion planning algorithms from the perspective of autonomous uav guidance. *JOURNAL OF INTELLIGENT & ROBOTIC SYSTEMS*, 57(1):65, 2009. ISSN 1573-0409. doi: 10.1007/s10846-009-9383-1.

[61] Slawomir Grzonka, Giorgio Grisetti, and Wolfram Burgard. Towards a navigation system for autonomous indoor flying slawomir grzonka giorgio grisetti wolfram burgard. In *PROCEEDINGS - IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION*, pages 2878–2883, 2009. ISBN 9781424427895. doi: 10.1109/ROBOT.2009.5152446.

[62] N. Gu, D. Wang, L. Liu, B. Zhang, and Z. Peng. Adaptive line-of-sight guidance law for synchronized path-following of under-actuated unmanned surface vehicles based on low-frequency learning. In *2017 36TH CHINESE CONTROL CONFERENCE (CCC)*, pages 6632–6637, July 2017. doi: 10.23919/ChiCC.2017.8028408.

[63] Yoshiro Hamada, Taro Tsukamoto, and Shinji Ishimoto. Receding horizon guidance of a small unmanned aerial vehicle for planar reference path following. *AEROSPACE SCIENCE AND TECHNOLOGY*, 77:129 – 137, 2018. ISSN 1270-9638. doi: 10.1016/j.ast.2018.02.039.

[64] X. Han, J. Wang, J. Xue, and Q. Zhang. Intelligent decision-making for 3-dimensional dynamic obstacle avoidance of uav based on deep reinforcement learning. In *2019 11th International Conference on Wireless Communications and Signal Processing (WCSP)*, pages 1–6, 2019.

[65] D. Hartman, K. Landis, M. Mehrer, S. Moreno, and J. Kim. Quadcopter dynamic modeling and simulation (Quad-Sim), 2014. URL https://github.com/dch33/Quad-Sim.

[66] R. He, A. Bachrach, and N. Roy. Efficient planning under uncertainty for a target-tracking micro-aerial vehicle. In *2010 IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AN AUTOMATION*, pages 1–8, May 2010. doi: 10.1109/ROBOT.2010.5509548.

[67] Ruijie He, S. Prentice, and N. Roy. Planning in information space for a quadrotor helicopter in a gps-denied environment. In *2008 IEEE INERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION*, pages 1814–1820, May 2008. doi: 10.1109/ROBOT.2008.4543471.

[68] Tanja Hebecker, Robert Buchholz, and Frank Ortmeier. Model-based local path planning for uavs. *JOURNAL OF INTELLIGENT & ROBOTIC SYSTEMS*, 78:127 – 142, 2015. doi: 10.1007/s10846-014-0097-7.

[69] Xie Heng, D. Cabecinhas, R. Cunha, C. Silvestre, and Xu Qingsong. A trajectory tracking lqr controller for a quadrotor: Design and experimental evaluation. In *TENCON 2015 - 2015 IEEE Region 10 Conference*, pages 1–7, Nov 2015. doi: 10.1109/TENCON.2015.7372729.

[70] Guillermo Heredia and Anibal Ollero. Stability of autonomous vehicle path tracking with pure delays in the control loop. *ADVANCED ROBOTICS*, 21(1-2):23–50, 2007. doi: 10.1163/156855307779293715.

[71] J. Hermansson, A. Gising, M. Skoglund, and T.B. Schon. Autonomous landing of an unmanned aerial vehicle. In *In Reglermte (Swedish Control Conference)*, page 1–5, June 2010.

[72] F. Hoffmann, N. Goddemeier, and T. Bertram. Attitude estimation and control of a quadrocopter. In *2010 IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS*, pages 1072–1077, 2010.

[73] G. M. Hoffmann, S. Waslander, and C.J. Tomlin. Quadrotor helicopter trajectory tracking control. In *PROCEEDINGS OF THE AIAA NAVIGATION, GUIDANCE AND CONTROL CONFERENCE*, 2008. Honolulu, HI.

[74] J. P. HOW, B. BEHIHKE, A. FRANK, D. DALE, and J. VIAN. Real-time indoor autonomous vehicle test environment. *IEEE CONTROL SYSTEMS*, 28(2):51–64, April 2008. ISSN 1066-033X. doi: 10.1109/MCS.2007.914691.

[75] J. Howlett, G. Schulein, and M. Mansour. A practical approach to obstacle field route planning for unmanned rotorcraft. In *PROCEEDINGS OF THE 60TH ANNUAL FORUM OF TH AMERICAN HELICOPTER SOCIETY*, 2004. Baltimore, MD.

[76] 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, Sept 2008. doi: 10.1109/IROS.2008.4650775.

[77] S. Hrabar. Reactive obstacle avoidance for rotorcraft uavs. In *2011 IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS*, pages 4967–4974, 2011.

[78] S. Hrabar and G. S. Sukhatme. Omnidirectional vision for an autonomous helicopter. In *2003 IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION*, volume 1, pages 558–563 vol.1, Sept 2003. doi: 10.1109/ROBOT.2003.1241653.

[79] Stefan Hrabar and Gaurav Sukhatme. Vision-based navigation through urban canyons. *JOURNAL OF FIELD ROBOTICS*, 26(5):431–452, 2009. ISSN 1556-4967. doi: 10.1002/rob.20284.

[80] Hui-Min Huangh. *Autonomy levels for unmanned systems (ALFUS) framework. Vol. i: Terminology,*. NIST Special Publication 1011-I-2.0., 2008. Contributedby the Ad Hoc ALFUS Working Group Participants.

[81] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *ArXiv*, abs/1502.03167, 2015.

[82] Dongwon Jung, Jayant Ratti, and Panagiotis Tsiotras. Real-time Implementation and Validation of a New Hierarchical Path Planning Scheme of UAVs via Hardware-in-the-Loop Simulation. *JOURNAL OF INTELLIGENT & ROBOTIC SYSTEMS*, 54(1-3, SI): 163–181, MAR 2009. ISSN 0921-0296. doi: {10.1007/s10846-008-9255-0}.

[83] Wooyoung Jung, Seunghan Lim, Dongjin Lee, and Hyochoong Bang. Unmanned aircraft vector field path following with arrival angle control. *JOURNAL OF INTELLIGENT ROBOTIC & SYSTEMS*, 84(1):311–325, Dec 2016. ISSN 1573-0409. doi: 10.1007/s10846-016-0332-5.

[84] I. Kaminer, O. Yakimenko, A. Pascoal, and R. Ghabcheloo. Path generation, path following and coordinated control for time-critical missions of multiple UAVs. In *2006 AMERICAN CONTROL CONFERENCE*, volume 1-12 of *Proceedings of the American Control Conference*, pages 4906+, 2006. ISBN 1-4244-0209-3. doi: {10.1109/ACC.2006.1657498}.

[85] T. Kanade, O. Amidi, and Q. Ke. Real-time and 3d vision for autonomous small and micro air vehicles. In *2004 43RD IEEE CONFERENCE ON DECISION AND CONTROL (CDC)*, volume 2, pages 1655–1662 Vol.2, Dec 2004. doi: 10.1109/CDC.2004.1430282.

[86] Katie Kang, Suneel Belkhale, Gregory Kahn, Pieter Abbeel, and Sergey Levine. Generalization through simulation: Integrating simulated and real data into deep reinforcement learning for vision-based autonomous flight. In *2019 INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA)*, 2019.

[87] Derya Kaya and Ali Türker Kutay. Modeling, simulation and system identification of a quadrotor helicopter. In *2013 ANKARA INTERNATIONAL AEROSPACE CONFERENCE (AIAC)*, 2013.

[88] Farid Kendoul. Survey of advances in guidance, navigation, and control of unmanned rotorcraft systems. *JOURNAL OF FIELD ROBOTICS*, 29(2):315–378, 2012. ISSN 1556-4967. doi: 10.1002/rob.20414.

[89] Farid Kendoul, Isabelle Fantoni, and Kenzo Nonami. Optic flow-based vision system for autonomous 3d localization and control of small aerial vehicles. *ROBOTICS AND AUTONOMOUS SYSTEMS*, 57(6–7):591 – 602, 2009. ISSN 0921-8890. doi: 10.1016/j.robot.2009.02.001.

[90] Farid Kendoul, Zhenyu Yu, and Kenzo Nonami. Guidance and nonlinear control system for autonomous flight of minirotorcraft unmanned aerial vehicles. *JOURNAL OF FIELD ROBOTICS*, 27(3):311–334, 2010. ISSN 1556-4967. doi: 10.1002/rob.20327.

[91] Kristian Klausen, Thor I. Fossen, Tor Arne Johansen, and A. Pedro Aguiar. Cooperative Path-Following for Multirotor UAVs with a Suspended Payload. In *2015 IEEE CONFERENCE ON CONTROL AND APPLICATIONS (CCA 2015)*, pages 1354–1360, 2015.

[92] William Koch, Renato Mancuso, Richard West, and Azer Bestavros. Reinforcement learning for uav attitude control. *ACM TRANS. CYBER-PHYS. SYST.*, 3(2), 2019. ISSN 2378-962X. doi: 10.1145/3301273.

[93] P. V. Kokotovic and H. J. Sussmann. A positive real condition for global stabilization of nonlinear systems. *SYSTEMS AND CONTROL LETTERS*, 13(2):125–133, 1989.

[94] Mangal Kothari, Ian Postlethwaite, and Da-Wei Gu. A suboptimal path planning algorithm using rapidly-exploring random trees. *INTERNATIONAL JOURNAL OF AEROSPACE INNOVATIONS*, 2:93–104, 2009. ISSN 1757-2258.

[95] S. Kukreti, M. Kumar, and K. Cohen. Genetically tuned lqr based path following for uavs under wind disturbance. In *2016 NTERNATIONAL CONFERENCE ON UNMANNED AIRCRAFT SYSTEMS (ICUAS)*, pages 267–274, June 2016. doi: 10.1109/ICUAS.2016.7502620.

[96] Y. K. Kwag and C. H. Chung. Uav based collision avoidance radar sensor. In *2007 IEEE INTERNATIONAL GEOSCIENCE AND REMOTE SENSING SYMPOSIUM*, pages 639–642, 2007. doi: 10.1109/IGARSS.2007.4422877.

[97] Nathan O. Lambert, Daniel S. Drew, Joseph Yaconelli, Roberto Calandra, Sergey Levine, and Kristofer S. J. Pister. Low level control of a quadrotor with deep model-based reinforcement learning. *IEEE ROBOTICS AND AUTOMATIC LETTERS, volume = 4, number = 4*, 2019.

[98] Dah-Jye Lee, Paul Merrell, Zhaoyi Wei, and Brent E. Nelson. Two-frame structure from motion using optical flow probability distributions for unmanned air vehicle obstacle avoidance. *MACHINE VISION AND APPLICATIONS*, 21:229–240, 2010. doi: 10.1007/s00138-008-0148-9.

[99] B. Li and Y. Wu. Path planning for uav ground target tracking via deep reinforcement learning. *IEEE ACCESS*, pages 29064–29074, 2020.

[100] L. Li, Y. Lv, and F. Wang. Traffic signal timing via deep reinforcement learning. *IEEE/CAA JOURNAL OF AUTOMATICA SINICA*, 3(3):247–254, July 2016. ISSN 2329-9266. doi: 10.1109/JAS.2016.7508798.

[101] Lebao Li, Lingling Sun, and Jie Jin. Survey of Advances in Control Algorithms of Quadrotor Unmanned Aerial Vehicle. In *2015 IEEE 16TH INTERNATIONAL CONFERENCE ON COMMUNICATION TECHNOLOGY (ICCT)*, pages 107–111, 2015. ISBN 978-1-4673-7005-9.

[102] Yueqian Liang and Yingmin Jia. Combined vector field approach for 2d and 3d arbitrary twice differentiable curved path following with constrained uavs. *JOURNAL OF INTELLIGENT ROBOTIC & SYSTEMS*, 83(1):133–160, Jul 2016. ISSN 1573-0409. doi: 10.1007/s10846-015-0308-x.

[103] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. Continuous control with deep reinforcement learning. In *2016 INTERNATIONAL CONFERENCE ON LEARNING REPRESENTATIONS (ICLR)*, 2016.

[104] Wenjie Lou and Xiao Guo. Adaptive trajectory tracking control using reinforcement learning for quadrotor. *INTERNATIONAL JOURNAL OF ADVANCED ROBOTIC SYSTEMS*, 13(1):38, 2016. doi: 10.5772/62128.

[105] L. Lu, C. Sampedro, J. Rodriguez-Vazquez, and P. Campoy. Laser-based collision avoidance and reactive navigation using rrt* and signed distance field for multirotor uavs. In *2019 INTERNATIONAL CONFERENCE ON UNMANNED AIRCRAFT SYSTEMS (ICUAS)*, pages 1209–1217, 2019.

[106] Xinglong Lu, Yiwen Cao, Zhonghua Zhao, and Yilin Yan. Deep reinforcement learning based collision avoidance algorithm for differential drive robot. In *Intelligent Robotics and Applications ICIRA 2018*, 2018.

[107] B. Ludington, E. Johnson, and G. Vachtsevanos. Augmenting uav autonomy: Vision-based navigation and target tracking for unmanned aerial vehicles. *IEEE ROBOTICS AUTOMATION MAGAINE*, 13(3):63–71, Sept 2006. ISSN 1070-9932. doi: 10.1109/MRA.2006.1678140.

[108] Z. Ma, T. Hu, L. Shen, W. Kong, and B. Zhao. A detection and relative direction estimation method for uav in sense-and-avoid. In *2015 IEEE INTERNATIONAL CONFERENCE ON INFORMATION AND AUTOMATION*, pages 2677–2682, 2015.

[109] Zhaowei Ma, Chang Wang, Yifeng Niu, Xiangke Wang, and Lincheng Shen. A saliency-based reinforcement learning approach for a uav to avoid flying obstacles. *Robotics and Autonomous Systems*, 100:108 – 118, 2018. ISSN 0921-8890. doi: 10.1016/j.robot.2017.10.009.

[110] S. O. H. Madgwick, A. J. L. Harrison, and R. Vaidyanathan. Estimation of imu and marg orientation using a gradient descent algorithm. In *2011 IEEE INTERNATIONAL CONFERENCE ON REHABILITATION ROBOTICS*, pages 1–7, 2011.

[111] R. Mahony, T. Hamel, and J. Pflimlin. Nonlinear complementary filters on the special orthogonal group. *IEEE TRANSACTIONS ON AUTOMATIC CONTROL*, 53(5):1203–1218, 2008.

[112] R. Mahony, V. Kumar, and P. Corke. Multirotor aerial vehicles: Modeling, estimation, and control of quadrotor. *IEEE ROBOTICS AUTOMATION MAGAZINE*, 19(3):20–32, Sept 2012. ISSN 1070-9932. doi: 10.1109/MRA.2012.2206474.

[113] A. Manjunath, P. Mehrok, R. Sharma, and A. Ratnoo. Application of virtual target based guidance laws to path following of a quadrotor uav. In *2016 INTERNATIONAL CONFERENCE ON UNMANNED AIRCRAFT SYSTEMS (ICUAS)*, pages 252–260, June 2016. doi: 10.1109/ICUAS.2016.7502565.

[114] Andreas Bell Martinsen and Anastasios M. Lekkas. Curved path following with deep reinforcement learning: Results from three vessel models. *OCEANS 2018 MTS/IEEE CHARLESTON*, pages 1–8, 2018.

[115] D.Q. Mayne, J.B. Rawlings, C.V. Rao, and P.O.M. Scokaert. Constrained model predictive control: Stability and optimality. *AUTOMATICA*, 36(6):789 – 814, 2000. ISSN 0005-1098. doi: 10.1016/S0005-1098(99)00214-9.

[116] M. Meingast, C. Geyer, and S. Sastry. Vision based terrain recovery for landing unmanned aerial vehicles. In *2004 43RD IEEE CONFERENCE ON DECISION AND CONTROL (CDC)*, volume 2, pages 1670–1675 Vol.2, Dec 2004. doi: 10.1109/CDC.2004.1430284.

[117] Luis Mejías, Srikanth Saripalli, Pascual Campoy, and Gaurav S. Sukhatme. Visual servoing of an autonomous helicopter in urban areas using feature tracking. *JOURNAL OF FIELD ROBOTICS*, 23(3-4):185–199, 2006. ISSN 1556-4967. doi: 10.1002/rob.20115.

[118] Jose Luis Mendoza-Soto, Jose J. Corona-Sanchez, and H. Rodriguez-Cortes. Quadcopter path following control. a maneuvering approach. *JOURNAL OF INTELLIGENT ROBOTIC & SYSTEMS*, 2018. ISSN 1573-0409. doi: 10.1007/s10846-018-0801-0.

[119] T. Merz and F. Kendoul. Beyond visual range obstacle avoidance and infrastructure inspection by an autonomous helicopter. In *2011 IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS*, pages 4953–4960, Sept 2011. doi: 10.1109/IROS.2011.6094584.

[120] B. Mettler, N. Dadkhah, and Z. Kong. Agile autonomous guidance using spatial value functions. *CONTROL ENGINEERING PRACTICE*, 18(7):773 – 788, 2010. ISSN 0967-0661. doi: 10.1016/j.conengprac.2010.02.013. Special Issue on Aerial Robotics.

[121] Cun-Xiao Miao and Jian-Cheng Fang. An adaptive three-dimensional nonlinear path following method for a fix-wing micro aerial vehicle. *INTERNATIONAL JOURNAL OF ADVANCED ROBOTIC SYSTEMS*, 9:1, 11 2012.

[122] Alain Micaelli, Claude Samson, Programme Robotique, and Projet Icare. Trajectory tracking for unicycle-type and two-steering-wheels mobile robots. In *IFAC PROCEEDINGS VOLUMES*, volume 27, pages 249–256, 09 1994.

[123] N. Michael, D. Mellinger, Q. Lindsey, and V. Kumar. The grasp multiple micro-uav testbed. *IEEE ROBOTICS AUTOMATION MAGAZINE*, 17(3):56–65, Sept 2010. ISSN 1070-9932. doi: 10.1109/MRA.2010.937855.

[124] D. Mittall, K. Kumar, S. N. Hashmi, P. Kumar, A. Nanda, and S. Chandra. Performance comparison of deep and shallow network for quadcopter automation. In *2018 IEEE 13TH INTERNATIONAL CONFERENCE ON INDUSTRIAL AND INFORMATION SYSTEMS (ICIIS)*, pages 143–147, 2018. doi: 10.1109/ICIINFS.2018.8721383.

[125] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin A. Riedmiller, Andreas Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *NATURE*, 518:529–533, 2015.

[126] Matthias Mueller, Neil Smith, and Bernard Ghanem. A benchmark and simulator for uav tracking. In *PROCEEDING OF THE EUROPEAN CONFERENCE ON COMPUTER VISION (ECCV)*, 2016.

[127] Derek R. Nelson, D. Blake Barber, Timothy W. McLain, and Randal W. Beard. Vector field path following for miniature air vehicles. *IEEE TRANSACTIONS ON ROBOTICS*, 23(3):519–529, JUN 2007. ISSN 1552-3098. doi: {10.1109/TRO.2007.898976}.

[128] Chunyu Nie, Zewei Zheng, and Zhiping Cai. Three-dimensional path-following control of a robotic airship with reinforcement learning. In *2019 INTERNATIONAL JOURNAL OF AEROSPACE ENGINEERING*, 2019.

[129] P. Niermeyer, V. S. Akkinapalli, M. Pak, F. Holzapfel, and B. Lohmann. Geometric path following control for multirotor vehicles using nonlinear model predictive control and 3d spline paths. In *2016 INTERNATIONAL CONFERENCE ON UNMANNED AIRCRAFT SYSTEMS (ICUAS)*, pages 126–134, June 2016. doi: 10.1109/ICUAS.2016.7502541.

[130] A. Ollero and G. Heredia. Stability analysis of mobile robot path tracking. In *PROCEEDINGS 1995 IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS. HUMAN INTERACTION AND COOPERATIVE ROBOTS.*, volume 3, pages 461–466, Aug 1995. doi: 10.1109/IROS.1995.525925.

[131] Necdet Sinan Ozbek, Mert Onkol, and Mehmet Onder Efe. Feedback control strategies for quadrotor-type aerial robots: a survey. *TRANSACTIONS OF THE INSTITUTE OF MEASUREMENT AND CONTROL*, 38(5, SI):529–554, MAY 2016. ISSN 0142-3312. doi: {10.1177/0142331215608427}.

[132] J. Park and N. Cho. Collision avoidance of hexacopter uav based on lidar data in dynamic environment. *REMOTE SENSING*, 12(6), 2020. doi: 10.3390/rs12060975.

[133] Sanghyuk Park, John Deyst, and Jonathan How. Performance and lyapunov stability of a nonlinear path-following guidance method. *JOURNAL OF GUIDANCE CONTROL AND DYNAMICS*, 30:1718–1728, 11 2007.

[134] Guilherme V. Pelizer, Natassya B. F. Silva, and Kalinka R. L. J. C. Branco. 3D path-following algorithms for unmanned aerial vehicles adjusted with genetic algorithm. In *COMUNICATION IN CRITICAL EMBEDDED SYSTEMS*, pages 63–80, 2017. ISBN 978-3-319-61403-8.

[135] R. Polvara, M. Patacchiola, S. Sharma, J. Wan, A. Manning, R. Sutton, and A. Cangelosi. Toward end-to-end control for uav autonomous landing via deep reinforcement learning. In *2018 INTERNATIONAL CONFERENCE ON UNMANNED AIRCRAFT SYSTEMS (ICUAS)*, pages 115–123, 2018. doi: 10.1109/ICUAS.2018.8453449.

[136] Guilherme V. Raffo, Manuel G. Ortega, and Francisco R. Rubio. Backstepping/nonlinear H-infinity control for path tracking of a quadrotor unmanned aerial vehicle. In *2008 AMERICAN CONTROL CONFERENCE, VOLS 1-12*, pages 3356–3361, 2008.

[137] Ashwini Ratnoo, Shmuel Hayoun, Asaf Granot, and Tal Shima. Path following using trajectory shaping guidance. *JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS*, 38:106–116, 2015. doi: 10.2514/1.G000300.

[138] Z. Rochala, K. Wojtowicz, P. Kordowski, and B. Brzozowski. Experimental tests of the obstacles detection technique in the hemispherical area for an underground explorer uav. *IEEE AEROSPACE AND ELECTRONIC SYSTEMS MAGAZINE*, 34(10):18–26, 2019. doi: 10.1109/MAES.2019.2918043.

[139] Alejandro Rodriguez-Ramos, Carlos Sampedro, Hriday Bavle, Paloma de la Puente, and Pascual Campoy. A deep reinforcement learning strategy for uav autonomous landing on a moving platform. *JOURNAL OF INTELLIGENT & ROBOTIC SYSTEMS*, 93(1): 351–366, Feb 2019. ISSN 1573-0409. doi: 10.1007/s10846-018-0891-8.

[140] 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, 2013.

[141] Ashton Roza and Manfredi Maggiore. Path Following Controller for a Quadrotor Helicopter. In *2012 AMERICAN CONTROL CONFERENCE (ACC)*, pages 4655–4660, 2012.

[142] B. Rubí, A. Ruiz, R. Perez, and B. Morcego. CS2AC trainning benchmarks: Path-flyer, 2019. URL https://cs2ac.upc.edu/en/training-benchmarks/path-flyer.

[143] Bartomeu Rubí, Bernardo Morcego, and Ramon Pérez. Adaptive nonlinear guidance law using neural networks applied to a quadrotor. In *2019 15th IEEE INTERNATIONAL CONFERENCE ON CONTROL AND AUTOMATION*, 2019.

[144] Bartomeu Rubí, Ramon Pérez, and Bernardo Morcego. A Survey of Path Following Control Strategies for UAVs Focused on Quadrotors. *Journal of Intelligent & Robotic Systems*, 98: 241–265, 2019. ISSN 0921-0296. doi: 10.1007/s10846-019-01085-z.

[145] Bartomeu Rubí, Bernardo Morcego, and Ramon Pérez. Deep Reinforcement Learning for Quadrotor Path Following with Adaptive Velocity. *Autonomous Robots*, 2020. doi: 10.1007/s10514-020-09951-8.

[146] Bartomeu Rubí, Adrián Ruiz, Ramon Pérez, and Bernardo Morcego. Path-flyer: A benchmark of quadrotor path following algorithms. In *2019 15th IEEE INTERNATIONAL CONFERENCE ON CONTROOL AND AUTOMATION*, 2019.

[147] Bartomeu Rubí, Bernardo Morcego, and Ramon Pérez. A Deep Reinforcement Learning Approach for Path Following on a Quadrotor. In *2020 EUROPEAN CONTROL CONFERENCE (ECC)*, 2020.

[148] A. Rucco, A. P. Aguiar, and J. Hauser. Trajectory optimization for constrained uavs: A virtual target vehicle approach. In *2015 INTERNATIONAL CONFERENCE ON UNMANNED AIRCRAFT SYSTEMS (ICUAS)*, pages 236–245, June 2015. doi: 10.1109/ICUAS.2015.7152296.

[149] Alessandro Rucco, A. Pedro Aguiar, Fernando Lobo Pereira, and João Borges de Sousa. *A Predictive Path-Following Approach for Fixed-Wing Unmanned Aerial Vehicles in Presence of Wind Disturbances*, volume 427, pages 623–634. Springer, Cham, 2016. ISBN 978-3-319-27146-0. doi: 10.1007/978-3-319-27146-0_48.

[150] M. Ryll, J. Ware, J. Carter, and N. Roy. Efficient trajectory planning for high speed flight in unknown environments. In *2019 INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA)*, pages 732–738, 2019.

[151] Rolf Rysdyk. Uav path following for target observation in wind. *JOURNAL OF GUIDANCE, CONTROL, AND DYNAMICS*, 29(5):1092–1100, 2006. doi: 10.2514/1.19101.

[152] S. Saha, A. Natraj, and S. Waharte. A real-time monocular vision-based frontal obstacle detection and avoidance for low cost uavs in gps denied environment. In *2014 IEEE INTERNATIONAL CONFERENCE ON AEROSPACE ELECTRONICS AND REMOTE SENSING TECHNOLOGY*, pages 189–195, 2014.

[153] C. Sampedro, H. Bavle, A. Rodriguez-Ramos, P. de la Puente, and P. Campoy. Laser-based reactive navigation for multirotor aerial robots using deep reinforcement learning. In *2018 IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS)*, pages 1024–1031, 2018.

[154] M. Sanfourche, G. L. Besnerais, P. Fabiani, A. Piquereau, and M. S. Whalley. Comparison of terrain characterization methods for autonomous uavs. In *PROCEEDINGS OF THE 65TH ANNUAL FORUM OF TH AMERICAN HELICOPTER SOCIETY*, page 1–14, 2009. Grapevine, TX.

[155] A. Santamaria-Navarro, J. Solà, and J. Andrade-Cetto. High-frequency mav state estimation using low-cost inertial and optical flow measurement units. In *2015 IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS)*, pages 1864–1871, 2015.

[156] S. Saripalli, J. F. Montgomery, and G. S. Sukhatme. Visually guided landing of an unmanned aerial vehicle. *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION*, 19(3):371–380, June 2003. ISSN 1042-296X. doi: 10.1109/TRA.2003.810239.

[157] Sebastian Scherer, Sanjiv Singh, Lyle Chamberlain, and Mike Elgersma. Flying fast and low among obstacles: Methodology and experiments. *THE INTERNATIONAL JOURNAL OF ROBOTICS RESEARCH*, 27(5):549–574, 2008. doi: 10.1177/0278364908090949.

[158] Keith W. Sevcik, Noah Kuntz, and Paul Y. Oh. Exploring the effect of obscurants on safe landing zone identification. *JOURNAL OF INTELLIGENT & ROBOTIC SYSTEMS*, 57 (1):281, 2009. ISSN 1573-0409. doi: 10.1007/s10846-009-9358-2.

[159] M. Zamurad Shah, Raza Samar, and Aamer I. Bhatti. Lateral track control of UAVs using the sliding mode approach: from design to flight testing. *TRANSACTIONS OF THE INSTITUTE OF MEASUREMENT AND CONTROL*, 37(4):457–474, APR 2015. ISSN 0142-3312. doi: {10.1177/0142331214543093}.

[160] H. Shen and C. Guo. Path-following control of underactuated ships using actor-critic reinforcement learning with mlp neural networks. In *2016 6TH INTERNATIONAL CONFERENCE ON INFORMATION SCIENCE AND TECHNOLOGY (ICIST)*, pages 317–321, May 2016. doi: 10.1109/ICIST.2016.7483431.

[161] D. H. Shim, Hoam Chung, and S. S. Sastry. Conflict-free navigation in unknown urban environments. *IEEE ROBOTICS AUTOMATION MAGAZINE*, 13(3):27–33, Sept 2006. ISSN 1070-9932. doi: 10.1109/MRA.2006.1678136.

[162] Jinok Shin, Sanki Ji, Woonghee Shon, Hogil Lee, Kuk Cho, and Sangdeok Park. Indoor hovering control of small ducted-fan type oav using ultrasonic positioning system. *JOURNAL OF INTELLIGENT & ROBOTIC SYSTEMS*, 61(1):15–27, 2011. doi: 10.1007/s10846-010-9488-6.

[163] Roland Siegwart and Illah R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. Bradford Company, Scituate, MA, USA, 2004. ISBN 0262015356. 2nd Ed (2011.

[164] David Silver, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. Deterministic policy gradient algorithms. In *PROCEEDINGS OF TGE 31ST INTERNATIONAL CONFERENCE ON MACHINE LEARNING - Volume 32*.

[165] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Vedavyas Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy P. Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis. Mastering the game of go with deep neural networks and tree search. *NATURE*, 529:484–489, 2016.

[166] Shashiprakash Singh and Radhakant Padhi. Automatic Path Planning and Control Design for Autonomous Landing of UAVs using Dynamic Inversion. In *2009 AMERICAN CONTROL CONFERENCE, VOLS 1-9*, PROCEEDINGS OF THE AMERICAN CONTROL CONFERENCE, pages 2409–2414, 2009.

[167] E. D. Sontag and H. J. Sussmann. Further comments on the stabilizability of the angular velocity of a rigid body. *SYSTEMS AND CONTROL LETTERS*, 12(3):213–217, 1989.

[168] J. Stevens and R. Mahony. Vision based forward sensitive reactive control for a quadrotor vtol. In *2018 IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS (IROS)*, pages 5232–5238, 2018.

[169] S. Stevšić, T. Nägeli, J. Alonso-Mora, and O. Hilliges. Sample efficient learning of path following and obstacle avoidance behavior for quadrotors. *IEEE ROBOTICS AND AUTOMATION LETTERS*, 3(4):3852–3859, Oct 2018. ISSN 2377-3766. doi: 10.1109/LRA.2018.2856922.

[170] A. Stulgis, L. Ambroziak, and M. Kondratiuk. Obstacle detection and avoidance system for unmanned multirotors. In *2018 23RD INTERNATIONAL CONFERENCE ON METHODS AND MODELS IN AUTOMATION ROBOTICS (MMAR)*, pages 455–460, 2018. doi: 10.1109/MMAR43460.2018.

[171] Shanwei Su. Path Following Control of Non-minimum Phase VTOL Aircraft via Minimum Distance Projection Method. In *26TH CHINESE CONTROL AND DECISION CONFERENCE (2014 CCDC)*, Chinese Control and Decision Conference, pages 708–712, 2014. ISBN 978-1-4799-3706-6.

[172] P. B. Sujit, S. Saripalli, and J. B. Sousa. Unmanned aerial vehicle path following: A survey and analysis of algorithms for fixed-wing unmanned aerial vehicless. *IEEE CONTROL SYSTEMS*, 34(1):42–59, Feb 2014. ISSN 1066-033X. doi: 10.1109/MCS.2013.2287568.

[173] Richard S. Sutton and Andrew Barto. *Reinforcement Learning: An Introduction*. The MIT Press, 1992.

[174] A. Symington, R. De Nardi, S. Julier, and S. Hailes. Simulating quadrotor uavs in outdoor scenarios. In *2014 IEEE/RSJ INTERNATIONAL CONFERENCE ON INTELLIGENT ROBOTS AND SYSTEMS*, pages 3382–3388, Sept 2014. doi: 10.1109/IROS.2014.6943033.

[175] Colin Theodore, Dale Rowley, David Hubbard, Adnan Ansar, Larry Matthies, Steve Goldberg, and Matthew Whalley. Flight trials of a rotorcraft unmanned aerial vehicle landing autonomously at unprepared sites. In *PROCEEDINGS OF THE 62ND ANNUAL FORUM OF TH AMERICAN HELICOPTER SOCIETY*, 2006. Phoenix, AZ.

[176] Peter Tsenkov, Jason K. Howlett, Matthew S. Whalley, Greg J. Schulein, Marc D. Takahashi, Matthew H. Rhinehart, and Berenice F Mettler May. A system for 3d autonomous rotorcraft navigation in urban environments. In *AIAA GUIDANCE, NAVIGATION AND CONTROL CONFERENCE AND EXHIBIT*, 2008.

[177] L. P. Tuyen and T. Chung. Controlling bicycle using deep deterministic policy gradient algorithm. In *2017 14TH INTERNATIONAL CONFERENCE ON UBIQUITOUS ROBOTS AND AMBIEN INTELLIGENCE (URAI)*, pages 413–417, June 2017. doi: 10.1109/URAI.2017.7992765.

[178] David R. Valencia and Donghan Kim. Trajectory tracking control for multiple quadrotors based on a neurobiological-inspired system. *2019 3RD IEEE INTERNATIONAL CONFERENCE ON ROBOTIC COMPUTING (IRC)*, pages 465–470, 2019.

[179] Wannes Van Loock, Goele Pipeleers, Moritz Diehl, Joris De Schutter, and Jan Swevers. Optimal Path Following for Differentially Flat Robotic Systems Through a Geometric Problem Formulation. *IEEE TRANSACTIONS ON ROBOTICS*, 30(4):980–985, AUG 2014. ISSN 1552-3098. doi: {10.1109/TRO.2014.2305493}.

[180] R. Vidal, O. Shakernia, H. J. Kim, D. H. Shim, and S. Sastry. Probabilistic pursuit-evasion games: theory, implementation, and experimental evaluation. *IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION*, 18(5):662–669, Oct 2002. ISSN 1042-296X. doi: 10.1109/TRA.2002.804040.

[181] C. Wang, J. Wang, Y. Shen, and X. Zhang. Autonomous navigation of uavs in large-scale complex environments: A deep reinforcement learning approach. *IEEE Transactions on Vehicular Technology*, pages 2124–2136, 2019.

[182] Chen Wang, Bifeng Song, Panfeng Huang, and Changhong Tang. Trajectory tracking control for quadrotor robot subject to payload variation and wind gust disturbance. *JOURNAL OF INTELLIGENT ROBOTIC & SYSTEMS*, 83(2):315–333, Aug 2016. ISSN 1573-0409. doi: 10.1007/s10846-016-0333-4.

[183] T. Wang, Y. Chen, J. Liang, C. Wang, and Y. Zhang. Combined of vector field and linear quadratic Gaussian for the path following of a small unmanned helicopter. *IET CONTROL THEORY AND APPLICATIONS*, 6(17):2696–2703, NOV 2012. ISSN 1751-8644. doi: {10.1049/iet-cta.2012.0270}.

[184] Yajing Wang, Xiangke Wang, Shulong Zhao, and Lincheng Shen. Vector field based sliding mode control of curved path following for miniature unmanned aerial vehicles in winds. *JOURNAL OF SYSTEMS SCIENCE AND COMPLEXITY*, 31(1):302–324, 2018. ISSN 1559-7067. doi: 10.1007/s11424-018-8006-y.

[185] Matt Whalley, Marc Takahashi, and Gregory Schulein. Field-testing of a helicopter uav obstacle field navigation and landing system. In *PROCEEDINGS OF THE 65TH ANNUAL FORUM OF TH AMERICAN HELICOPTER SOCIETY*, 2010. Grapevine, TX.

[186] Keyu Wu, Mahdi Abolfazli Esfahani, Shenghai Yuan, and Han Wang. Depth-based obstacle avoidance through deep reinforcement learning. In *Proceedings of the 5th International Conference on Mechatronics and Robotics Engineering*, page 102–106, 2019. doi: 10.1145/3314493.3314495.

[187] M. Wzorek and P. Doherty. Reconfigurable path planning for an autonomous unmanned aerial vehicle. In *2006 INTERNATIONAL CONFERENCE ON HYBURD INFORMATION TECHNOLOGY*, volume 2, pages 242–249, Nov 2006. doi: 10.1109/ICHIT.2006.253618.

[188] M. Wzorek, C. Berger, and P. Doherty. A framework for safe navigation of unmanned aerial vehicles in unknown environments. In *2017 25TH INTERNATIONAL CONFERENCE ON SYSTEMS ENGINEERING (ICSEng)*, pages 11–20, 2017.

[189] Y. Xiao, X. Lei, and S. Liao. Research on uav multi-obstacle detection algorithm based on stereo vision. In *2019 IEEE 3RD INFORMATION TECHNOLOGY, NETWORKING,*

ELECTRONIC AND AUTOMATION CONTROL CONFERENCE (ITNEC), pages 1241–1245, 2019.

[190] Takeshi Yamasaki, S. N. Balakrishnan, and Hiroyuki Takano. Integrated Guidance and Autopilot for a Path-Following UAV via High-Order Sliding Modes. In *2012 AMERICAN CONTROL CONFERENCE (ACC)*, Proceedings of the American Control Conference, pages 143–148, 2012.

[191] Chao Yan, Xiaojia Xiang, and Chang Wang. Towards real-time path planning through deep reinforcement learning for a uav in dynamic environments. *Journal of Intelligent & Robotic Systems*, 98:297 – 309, 2020. ISSN 1573-0409. doi: 10.1007/s10846-019-01073-3.

[192] Songyue Yang, Zhijun Meng, Xuzhi Chen, and Ronglei Xie. Real-time obstacle avoidance with deep reinforcement learning three-dimensional autonomous obstacle avoidance for uav. In *Proceedings of the 2019 International Conference on Robotics, Intelligent Control and Artificial Intelligence*, page 324–329, 2019. doi: 10.1145/3366194.3366251.

[193] Xin Yang, Hongcheng Luo, Yuhao Wu, Yang Gao, Chunyuan Liao, and Kwang-Ting Cheng. Reactive obstacle avoidance of monocular quadrotors with online adapted depth prediction network. *NEUROCOMPUTING*, 325:142 – 158, 2019. ISSN 0925-2312. doi: doi.org/10.1016/j.neucom.2018.10.019.

[194] R. Yu, Z. Shi, C. Huang, T. Li, and Q. Ma. Deep reinforcement learning based optimal trajectory tracking control of autonomous underwater vehicle. In *2017 36TH CHINESE CONTROL CONFERENCE (CCC)*, pages 4958–4965, July 2017. doi: 10.23919/ChiCC.2017.8028138.

[195] Bo Zhao, Bin Xian, Yao Zhang, and Xu Zhang. Nonlinear robust sliding mode control of a quadrotor unmanned aerial vehicle based on immersion and invariance method. *INTERNATIONAL JOURNAL OF ROBUST AND NONLINEAR CONTROL*, 25(18): 3714–3731, 2015. ISSN 1099-1239. doi: 10.1002/rnc.3290.

[196] S. Zhao, X. Wang, W. Kong, D. Zhang, and L. Shen. A novel data-driven control for fixed-wing uav path following. In *2015 IEEE INTERNATIONAL CONFERENCE ON INFORMATION AND AUTOMATION*, pages 3051–3056, Aug 2015. doi: 10.1109/ICInfA.2015.7279812.

[197] M. Zhaowei, H. Tianjiang, S. Lincheng, K. Weiwei, Z. Boxin, and Y. Kaidi. An iterative learning controller for quadrotor uav path following at a constant altitude. In *2015 34TH CHINESE CONTROL CONFERENCE (CCC)*, pages 4406–4411, July 2015. doi: 10.1109/ChiCC.2015.7260322.

[198] L. Zheng, P. Zhang, J. Tan, and F. Li. The obstacle detection method of uav based on 2d lidar. *IEEE ACCESS*, 7:163437–163448, 2019. doi: 10.1109/ACCESS.2019.2952173.

[199] Bingyu Zhou, H. Satyavada, and S. Baldi. Adaptive path following for unmanned aerial vehicles in time-varying unknown wind environments. In *2017 AMERICAN CONTROL CONFERENCE (ACC)*, pages 1127–1132, May 2017. doi: 10.23919/ACC.2017.7963104.

[200] Dingjiang Zhou and Mac Schwager. Vector Field Following for Quadrotors using Differential Flatness. In *2014 IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION (ICRA)*, pages 6567–6572, 2014.