

**Multirotor UAV Simulation for Monitoring
Natural Disasters Using GAZEBO
(versão final pós correção)**

Guilherme António Cardoso Almeida

Dissertação para obtenção do Grau de Mestre em
Engenharia Aeronáutica
(Mestrado integrado)

Orientador: Professor Doutor Pedro Vieira Gamboa
Co-orientador: Mestre Joaquim Vasconcelos Reynolds Sousa

dezembro de 2023

Folha em branco

Declaração de Integridade

Eu, Guilherme António Cardoso Almeida, que abaixo assino, estudante com o número de inscrição 39429 de/o mestrado integrado em engenharia aeronáutica da Faculdade de Engenharias, declaro ter desenvolvido o presente trabalho e elaborado o presente texto em total consonância com o **Código de Integridades da Universidade da Beira Interior**.

Mais concretamente afirmo não ter incorrido em qualquer das variedades de Fraude Académica, e que aqui declaro conhecer, que em particular atendi à exigida referenciação de frases, extratos, imagens e outras formas de trabalho intelectual, e assumindo assim na íntegra as responsabilidades da autoria.

Universidade da Beira Interior, Covilhã 20 /12 /2023

Dedication

I want to dedicate this dissertation to my family, especially my mother, father, brother and uncle Paulo Vitor. I wouldn't be half the man that I am today without them. Thank you very much for everything.

Folha em branco

Acknowledgments

I would like to thank Professor Pedro Gamboa for accepting me on this project and for all his advice. My colleague, Joaquim Reynolds Sousa for his teachings and patience specifically in programming and informatics matters.

A special thanks to my graduation friends: Guilherme Guerreiro, Marcos Rosa, Nuno Santos, Joel Castro and José Medeiros. To my friends from Seia: Dinis, Pedro and Daniel whom I revisit every summer. To my cousins Henrique, Tomás and André with whom I spent a lot of time at the swimming pool and computer. To my Erasmus Student Network Covilhã family: Eric e Jaqueline; you are amazing.

And last but not the least, to my beloved father, mother and brother, because without you, I would not have conditions and support for the successful completion of my education.

Folha em branco

Resumo

O avanço das capacidades aeronáuticas e tecnológicas nos últimos anos tem aumentado significativamente a versatilidade dos Veículos Aéreos Não Tripulados. Os drones atualmente possuem várias configurações, incluindo asas fixas, asas rotativas e designs híbridos, atendendo a uma ampla gama de propósitos, como atividades recreativas, agricultura, competições desportivas e monitoramento civil e militar.

Os drones multirotores são acessíveis devido ao seu baixo custo e facilidade de uso e como consequência proliferaram atualmente. Aproveitando esta eficácia e popularidade e utilizando um quadricóptero de monitorização, esta dissertação concentra-se no desenvolvimento de uma simulação na qual um drone pode ser controlado por um algoritmo básico em Python. Esse algoritmo é responsável por enviar comandos ao drone, permitindo que ele siga autonomamente uma frente de fogo florestal.

O processo de criação da simulação envolveu várias etapas. Primeiramente, o drone virtual foi projetado em design auxiliado por computador (DAC), inspirado num modelo de drone real. Em seguida, o "script" do drone foi adaptado de um drone existente, permitindo que ele funcione dentro do ambiente de simulação. Um sistema de piloto automático foi implementado para fornecer controlo autónomo ao drone virtual, enquanto um programa de configuração de robôs foi utilizado para caracterizar o seu comportamento. Além disso, um ambiente virtual, baseado no laboratório de aerodinâmica e propulsão do Departamento de Ciências Aeroespaciais da Universidade da Beira Interior (UBI), foi criado para fornecer um cenário para a operação do drone.

A implementação bem-sucedida da simulação de um drone quadricóptero no simulador Gazebo tem como principal foco a sua aplicação na monitorização de incêndios florestais. Num futuro próximo este projeto envolve testar o algoritmo e a simulação desenvolvidos num drone real dentro dos limites do laboratório.

Palavras-chave

UAV, fogo florestal, monitorização, Gazebo, multirotor, voo virtual

Folha em branco

Abstract

The advancement of aeronautical and technological capabilities in recent years has significantly increased the versatility of Unmanned Aerial Vehicles. Drones nowadays come in various configurations, including fixed-wing, rotary-wing, and hybrid designs, catering for a wide range of purposes such as recreational activities, agriculture, sports competitions, military operations, and civil and military monitoring.

Multirotor drones are accessible due to their low cost and ease of use, and as a result, they are currently proliferating. Taking advantage of this efficiency and popularity, and using a monitoring quadcopter, this dissertation focuses on the development of a simulation in which a drone can be controlled by a basic algorithm in Python. This algorithm is responsible for sending commands to the drone, allowing it to autonomously follow a forest fire front.

The process of creating the simulation involved several stages. Firstly, the virtual drone was designed in computer aided design (CAD), inspired by a real drone model. Then, the drone's script was adapted from an existing drone, enabling it to function within the simulation environment. An autopilot system was implemented to provide autonomous control to the virtual drone, while a robot configuration program was used to characterize its behavior. Additionally, a virtual environment, based on the aerodynamics and propulsion laboratory of the Department of Aerospace Sciences at University of Beira Interior (UBI), was created to provide a scenario for drone operation.

The successful implementation of a quadcopter drone simulation in the Gazebo simulator focuses primarily on its application in monitoring forest fires. Soon, this project involves testing the developed algorithm and simulation on a real drone within the confines of the laboratory.

Keywords

UAV, wildfire, monitoring, Gazebo, multirotor, virtual flight

Folha em branco

Contents

1 Introduction	1
1.1 Motivation.....	1
1.2 Objectives.....	1
2 State of the art	4
2.1 Brief history of UAVs.....	4
2.2 UAV market.....	7
2.2.1 The American Case.....	10
2.3 UAV applications.....	11
2.3.1 Photogrammetry.....	11
2.3.2 Agricultural and forestry.....	12
2.3.3 Humanitarian aid.....	13
2.3.4 Merchandise delivery.....	14
2.3.5 Atmospheric observation, air analysis and pollution....	14
2.3.6 Monitoring natural disasters.....	14
2.3.7 Wildfires.....	15
2.4 UAV types and examples.....	20
2.5 Flight controllers software and simulators.....	22
2.5.1 Flight controllers software.....	22
2.5.2 Simulation platforms.....	25
3 Methodology	29
3.1 Software Tools.....	29
3.2 Test laboratory.....	31
3.3 Multirotor CAD model.....	34
3.4 UAV moments of inertia.....	35
3.5 Basic drone control functions.....	37
4 Model Implementation	43
4.1 Mathematical models of the flight simulation.....	43
4.1.1 Rigid bodies, equations of motion and state vector.....	44
4.1.2 Step sizes and simulation integration.....	46
4.1.3 Collisions and constraints.....	47

4.1.4	Aerodynamics and propulsion.....	49
4.2	Test lab virtual model.....	49
4.2.1	Parameters.....	49
4.2.2	Operation limits.....	52
4.2.3	Fire patterns.....	54
4.3	Physical models of the UAV and camera.....	55
4.3.1	Raspberry Pi V2.....	55
4.3.2	Holybro X500 V2.....	57
4.4	Multicopter virtual model.....	58
5	Results	63
5.1	UAV control algorithm and fire tracking.....	65
5.1.1	Continuous trajectories.....	65
5.1.2	Discontinuous trajectories.....	71
5.2	Simulation errors and bugs.....	73
5.3	Wall detection algorithm.....	75
6	Conclusion	79
6.1	Overview.....	79
6.2	Future work.....	80
	References	82
	Appendix	91

Folha em branco

List of Figures

Figure 2.1 - Kettering Bug [8]4

Figure 2.2 - Hewitt-Sperry[9]4

Figure 2.3 - DH.82B Queen Bee [10]5

Figure 2.4 - Radioplane OQ-2 [11]5

Figure 2.5 - MQM-57 Falconer [12]5

Figure 2.6 - Ryan Firebee[13]6

Figure 2.7 - Parrot ANAFI USA drone[15].....7

Figure 2.8 - Unmanned Aerial vehicles market growth by region (2022-2027)[18]7

Figure 2.9 - Global Unmanned Aerial Vehicle Market share, by application, 2019[17].....9

Figure 2.10 - USA Department of Defense spending on UAS [20]10

Figure 2.11 - Historical layer model obtained by use of photogrammetry 3D model[22].....12

Figure 2.12 - Spraying drone for herbicide applications [24]13

Figure 2.13 - Zipline's company drones delivering medical supplies [28]13

Figure 2.14 - Flood devastation caused by hurricane Katrina in the USA [30]15

Figure 2.15 - Different stages in the evolution of a fire [31]16

Figure 2.16 - IR image of wildfire hotspots from drone camera [30]17

Figure 2.17 - Conceptual UAV- based forest fire monitoring, detection and fighting system [32]19

Figure 2.18 - Different types of UAV. a) rotary-wing; b) fixed-wing; c) hybrid-wing [36]21

Figure 2.19 - DJI Mavic mini [38]22

Figure 2.20 - Development history originating from the original Multiwii flight controller software [42]23

Figure 2.21 - Development history of the OpenPilot series flight controller software [42]24

Figure 2.22 - Inside a gazebo world simulation [62]27

Figure 3.1 - Laboratory in aerospace sciences department at UBI32

Figure 3.2 - UAV testing area in the upper floor32

Figure 3.3 - Wildfire aerial view Yakutsk, Russia[68]33

Figure 3.4 - Wildfire aerial view near Izmir, Turkey[69]33

Figure 3.5 - Wildfire aerial view near Chiang Mai, Thailand[70]33

Figure 3.6 - Tessellation in CATIA V5.....34

Figure 3.7 - Side view of the UAV in CATIA V5.....35

Figure 3.8 - Upper view of the UAV in CATIA V5.....	35
Figure 3.9 - Nodes communication model in the ROS environment [74]	37
Figure 4.1 - The body coordinate frame[84]	44
Figure 4.2 - An example of an error in ball and socket joint [84].....	48
Figure 4.3 - Diagram with the ODE simulation parameters	51
Figure 4.4 - Test lab virtual model	53
Figure 4.5 - Fire patterns in Gazebo simulation	54
Figure 4.6 - Raspberry Pi V2 camera[93].....	56
Figure 4.7 - X500 V2 Holybro drone[96].....	58
Figure 4.8 - X500 V2 Holybro drone inside Gazebo simulation	59
Figure 5.1 - Terminal in Ubuntu 20.04 initializing the script	63
Figure 5.2 - Terminal in Ubuntu 20.04 initializing QGC	64
Figure 5.3 - Gazebo and QGC side to side with the drone up and ready to fly	64
Figure 5.4 - Takeoff position coordinate (0,0,2).....	66
Figure 5.5 - Starting point of the fire trail coordinate (1,1,2).....	67
Figure 5.6 - Starting point of the F shaped trail	67
Figure 5.7 - Coordinate (4,1,2).....	67
Figure 5.8 - Coordinate (5,-1,2).....	67
Figure 5.9 - Upper view of the coordinate (4,1,2).....	68
Figure 5.10 - Upper view of the coordinate (5,-1,2).....	68
Figure 5.11 - Perspective view of the drone in coordinate (3,1, 2.3).....	69
Figure 5.12 - Upper view of the drone in coordinate (7,0,2.3).....	69
Figure 5.13 - Coordinate (3,1, 2.3).....	70
Figure 5.14 - Coordinate (7,0,2.3).....	70
Figure 5.15 - Coordinate (3,-1,1).....	70
Figure 5.16 - Upper view of the drone in coordinate (3,0, 2.5).....	71
Figure 5.17 - Coordinate (3,0, 2.5).....	71
Figure 5.18 - Perspective view of the drone after returning from first fire path	72
Figure 5.19 - Upper view of the drone in coordinate (6,1,1).....	72
Figure 5.20 - Upper view of the drone in coordinate (6,-1,1).....	73
Figure 5.21 - Coordinate (6,1,1).....	73
Figure 5.22 - Coordinate (6,-1,1).....	73
Figure 5.23 - ROS master bug in Gazebo simulation	74
Figure 5.24 - QgroundControl altimetry and position data	74
Figure 5.25 - Time synchronizer error in Ubuntu terminal	75
Figure 5.26 - Terminal with data from takeoff position	76

Folha em branco

List of Tables

Table 2.1 – Total UAV systems market: revenue forecasts (world), 1998-2008 [19]

Table 3.1 - Mass and volumetric mass of each part of the UAV

Table 4.1 - Raspberry Pi Cameras hardware specifications[73]

Folha em branco

Acronyms

AI	Artificial Intelligence
API	Application Programming Interface
APM	ArduPilot Mega
CAE	Computer-Aided Engineering
CAD	Computer-Aided Design
CAGR	Compound Annual Growth Rate
CAM	Computer-Aided Manufacturing
CFM	Constraint Force Mixing
DAE	Digital Asset Exchange
DART	Dynamic Animation and Robotics Toolkit
DHL	Dalsey Hillblom and Lynn
FOV	Field of View
FPV	First-person view
GNC	Guidance, Navigation, and Control
GPS	Global Positioning System
GUI	Graphical User Interface
HALE	High Altitude Long Endurance
IMU	Inertial Measurement Unit
IR	Infrared radiation
ISU	International Space University
MALE	Medium Altitude Long Endurance
MAV	Micro Air Vehicle
MEMS	Micro Electro-Mechanical Systems
NASA	National Aeronautics and Space Administration
ODE	Open Dynamics Engine
PC	Portable Computer
PLM	Product Lifecycle Management
PX4	Pixhawk 4
QGC	QGroundControl
RC	Radio Controlled
RGB	Red, Green and Blue
ROS	Robot Operating System
RPM	Revolutions per minute
SDF	Simulation Description Format
SITL	Software-in-the-loop
UAS	Unmanned Aerial Systems
UAV	Unmanned Aerial Vehicle
UBI	University of Beira Interior
UK	United Kingdom
USA	United States of America
USD	United States Dollars
USSR	Union of Soviet Socialist Republics
XML	Extended Markup Language

Nomenclature

Roman symbols

h_m	motor height
I	Inertia matrix
K	motor constant
K_m	moment constant of the motor
K_v	velocity constant of the motor
r_0	outer radius of motor
r_i	inner radius of motor
V	Voltage

Greek symbols

ρ	density
π	Pi

Folha em branco

Chapter 1

Introduction

1.1 Motivation

Over the past few years, there has been a significant rise in natural disasters caused by human activities, leading to disturbances in ecosystems. Greenhouse gas emissions and ocean pollution have played a major role in this increase. Some common examples of these disasters include forest fires, floods, volcanic eruptions, and oil spills.[1], [2]

To effectively monitor and track the development of these environmental disturbances, a combination of satellite images, information from Unmanned Aerial Vehicles (UAVs), geological data, and reports from affected individuals are utilized. Particularly, the versatility and effectiveness of UAVs have resulted in substantial investments in this field. Between 2021 and 2026, the global drone market is expected to grow at a compound annual growth rate of over 16 %, with projections indicating it will reach 58.4 billion dollars at the end of that period[3]. Notably, China and the United States generated revenues of 1.25 billion and 1.22 billion United States Dollars (USD), respectively, in 2022[4]. Key industry players, such as NVIDIA Corporation, Lockheed Martin, and The Boeing Company, are fiercely competing for future market shares in the drone industry[5]. Consequently, drones are expected to play an increasingly significant role in our daily lives, evolving and assisting us in various tasks, including the monitoring of natural disasters.

Considering the growing importance of drones across different applications, particularly in environmental monitoring, the development of advanced algorithms for drone control has become a crucial area of research. Specifically, in the context of monitoring forest fires, the deployment of drones equipped with fire detection algorithms holds great promise. In this context, a project was proposed in order to develop a drone simulation where it detects wildfire with effectiveness. The project was divided into two parts: control and artificial intelligence (AI) fire tracking algorithm. This document describes the drone control part including state of the art, methodology and how the drone's virtual environment was implemented with its results.

1.2 Objectives

This dissertation aims to develop a robust drone control algorithm for monitoring forest fires. Tasks include calculating moments of inertia, conducting literature research, and modeling the drone and its operating environment. The drone's control functions will be designed based on target distance, provided by an advanced fire front tracking algorithm. Rigorous virtual

simulations will validate the control algorithm and drone model, ensuring their reliability in monitoring forest fires.

The objective is to create a tailored drone control algorithm that enhances fire detection and response, minimizing the impact of forest fires. By integrating the AI fire front tracking system, the effectiveness and efficacy of fire detection will be improved. It seeks to contribute to the field by designing and implementing a reliable system for monitoring forest fires, ultimately contributing to mitigate their devastating consequences.

Throughout the work, specifically in Chapter 2, State of the art, the history of drones is demonstrated, their value in the current economy both presently and throughout their existence. Their applications in a real context to aid society are explored. In Chapter 3 and 4, Methodology and Model Implementation, the process of applying the multirotor in a virtual environment is showcased, along with the demonstration of the mathematical equations that underpin virtual flight. Finally, in Chapter 5 Results, the control of the UAV is verified through functions that establish a connection with the local host.

Chapter 2

State of the art

2.1 Brief history of UAVs

UAVs, which are aircraft without on-board crew or passengers, have evolved significantly over time. They can either be automated drones or remotely piloted vehicles, capable of flying for extended periods at controlled speeds and altitudes and they play a crucial role in various aspects of aviation. The first type of unmanned aircraft was probably hot air balloons. Yet, these cannot be considered drones because their flight cannot be controlled like fixed-wing and multirotor aircraft.[6]

During World War 1, the first UAVs were developed using radio control techniques like the Hewitt-Sperry automatic airplane and the Kettering Bug unmanned aerial torpedo (see Figures 2.1 and 2.2). These were like flying bombs which had military purposes and were capable of striking ground targets. After World War 1, DH.82B Queen Bee (see Figure 2.3) was devised as a low-cost radio-controlled (RC) target aircraft, and if it survived the shooting, its controller would attempt to recover it for re-use. It was with the appearance of this aircraft that the term “drone” (a male bee) started being used for pilotless aircraft.[7]



Figure 2.1 - Kettering Bug [8]



Figure 2.2 - Hewitt-Sperry[9]



Figure 2.3 - DH.82B Queen Bee [10]

During World War 2, approximately 15000 Radioplane OQ-2 (see Figure 2.4) drones were produced by radio aircraft manufacturers for the US army. This drone was launched into the air and recovered using a parachute. The OQ-3, a subsequent version, was widely used throughout World War 2; roughly 9400 units were built during that time. After World War 2, drones were first utilized for aerial reconnaissance. The first drone of this kind, the MQM-57 Falconer (see Figure 2.5), had its first flight in 1955.

Throughout the Vietnam War, the US employed Ryan Firebee drones (see Figure 2.6), whose development began in 1951. These were launched from Hercules transport aircraft, which had two Firebee drones mounted under each wing for a total of four Firebee drones. In this conflict, reconnaissance was first deployed on a large scale. Drones also began to use a wide range of new roles, such as acting as decoys in combat, launching missiles against fixed targets and dropping leaflets for psychological operations.



Figure 2.4 - Radioplane OQ-2 [11]



Figure 2.5 - MQM-57 Falconer [12]



Figure 2.6 - Ryan Firebee[13]

Countries outside of the United Kingdom (UK) and the United States of America paid attention to the relevance of drones in the Vietnam War and began to investigate and develop their own versions of the unmanned aerial technology. New models became more sophisticated, with improved endurance and the ability to maintain greater height. In recent years, models have been developed that use technology such as solar power to tackle the problem of fueling longer flights. For example, in Japan RC helicopters were developed in the late 1970s and these hobby-use model radio control helicopters were going to be mass produced around the world in the early 1980s.

From 1990s to 2000s embedded system technology, small sensors, early micro-computers, communication devices and avionics systems such as autopilots appeared which led to a substantial growth period of the drone industry. Micro electromechanical systems (MEMS) were developed and made possible the development of a super lightweight Inertial Measurement Unit (IMU) that could be incorporated inside small multicopters making way to the consumers through the RC toy market. Research and development became active in universities and other academic fields. Most studies focused on issues of higher performance, hardware, software, aerodynamics and autonomous flight control.

From 2010 to today, a variety of flight controllers like ArduPilot (APM) and Pixhawk 4 (PX4) came out in the world. These open-source controllers combined with Linux Ubuntu, make drone controlling easy and flexible. Parrot Company and 3D Robotics are just some important lightweight manufacturing drone companies. Nowadays drones have many functions such as monitoring natural disasters, photography, filming, delivering goods, medical supplies etc. In Figure 2.7 it is shown a modern drone created by Parrot Company that is capable of performing such tasks. In the future it is foreseeable that drones will be truly autonomous, fully commanded with artificial intelligence.[14]



Figure 2.7 - Parrot ANAFI USA drone[15]

2.2 UAV market

The global UAV market is witnessing significant growth, with an expected increase from 26.03 billion USD in 2022 to 29.93 billion USD in 2023 at a Compound Annual Growth Rate (CAGR) of 15.0%. By 2027, it is projected to reach 52.21 billion USD, with a CAGR of 14.9%[16]. In 2019, the global UAV market size was valued at 10.72 billion USD.[17]

North America held a larger market share in 2019 due to increased drone usage in infrastructure surveys, aerial mapping, and military applications. The market in Europe is expected to grow at a moderate pace, while the Asia Pacific region is anticipated to exhibit significant growth, particularly in the demand for modern warfare unmanned aerial vehicles.

In a study from the year 2000, North America and Europe were projected to represent 60% of the total global UAV market in 2008, supporting diverse UAV capabilities. The Asia Pacific region, including countries like Australia, showed interest in tactical fixed wing capabilities and explored various civil uses of UAVs. Notably, the United States leads in drone use, but other countries like China, France, Germany, India, Israel, and Russia also possess or express interest in acquiring UAV technology.



Figure 2.8 - Unmanned Aerial vehicles market growth by region (2022-2027)[18]

In Figure 2.8 it is shown that Europe's market is expected to grow steadily due to increasing demand for cameras, software, avionics, and navigation systems. Major technology companies like Parrot Drones and Delair operate in European countries, addressing global demand. The Asia Pacific market is set for significant growth, driven by the demand for modern warfare unmanned aerial vehicles for strategic and tactical applications. Importer countries like India, Pakistan, and China are expected to generate substantial revenue. However, the rest of the world, including South Africa, Saudi Arabia, and Brazil, is likely to have a smaller market share due to limited market penetration.

In Table 2.1, throughout the forecast period (1998-2008), different classes of UAV systems and market segments were expected to exhibit varying growth rates. However, the compound annual growth rate for the entire market reached 12.2%. The market's growth will primarily be driven by military funding, accelerated worldwide by successful operations, expanded capabilities, and increased political support. Sales of fixed wing tactical UAVs, operating within specific range and altitude parameters, contribute to the majority of market revenues as expected.[19]

Table 2.1 - Total UAV systems market: revenue forecasts (world), 1998-2008 [19]

Year	Revenues (\$ billion)	Growth Rate (%)
1998	2.07	
1999	2.17	4.9
2000	2.38	9.6
2001	2.67	12.3
2002	3.02	13.1
2003	3.45	14.2
2004	3.98	15.1
2005	4.57	15.1
2006	5.22	14.3
2007	5.96	14.1
2008	6.87	15.2
Compound annual growth rate (CAGR)		12.2
Cumulative revenues (1998-2008)	42.4	

Increasing defense spending is a key driver for the growth of the UAV drone market. The procurement and purchase of UAV drones have become a force multiplier in combat operations worldwide. Notably, defense spending in the UK reached 42.4 billion British pounds (49.65 billion USD) in 2020-21, driving the market's growth.

The UAV market is divided into military, commercial, and recreational segments in terms of application (see Figure 2.9). During the forecast period, the military segment is expected to hold the largest share, driven by increasing demand for tactical and strategic UAVs for military purposes. Various countries have ramped up their purchases of specialized UAVs, such as

Medium Altitude Long Endurance (MALE) and High-Altitude Long Endurance (HALE) UAVs, for a range of missions, which is likely to further drive demand in the coming years.



Figure 2.9 - Global Unmanned Aerial Vehicle Market share, by application, 2019 [17]

The demand for UAVs is fueled by major operations such as border monitoring, surveillance, and border security worldwide. Additionally, the commercial segment is poised for significant growth in the future, fueled by the rapid adoption of UAVs in surveying, aerial mapping, crop monitoring, forest monitoring, and logistics. Major companies like Uber, investing in Air Taxi development, and logistics giant Dalsey Hillblom and Lynn (DHL) Logistics, innovating new air transportation using UAVs, are expected to contribute to the high growth of the commercial segment during the forecast period.[17]

The UAV market faced a moderate impact from the COVID-19 outbreak. Countries' budget reallocations towards developing medical infrastructure have limited defense budget expenditures for research, development, and acquisition of deep-tech products.

The shift towards unmanned systems reflects the military's inclination to leverage advanced technology and adapt to the emergence of asymmetrical warfare. Initiatives for indigenous aerospace platforms are on the rise, leading to an increase in the development of home-grown UAVs, which is expected to significantly boost the UAV market in the Asia-Pacific region. The growing number of initiatives by Original Equipment Manufacturers in indigenous UAV development will also enhance the adoption rate of this technology by providing easier access to low-cost UAVs.

The integration of unmanned vehicles in manned combat operations has opened up new business opportunities within the military segment. In November 2021, China's People's Liberation Army announced plans for joint manned and unmanned aerial operations, utilizing indigenously manufactured J-20 fighter jets in tandem with four drones for trial purposes. These operations

are poised to significantly enhance the country's aerial military capabilities, including surveillance, search, rescue, and close combat operations.[18]

2.2.1 The American Case

The increased demand for drone technology witnessed following the Gulf conflict was further amplified by the post-9/11 conflicts in Afghanistan and Iraq. These significant military engagements, intertwined with the broader Global War on Terror, created an unprecedented opportunity for the extensive utilization of drones. The surge in interest is evident in the Department of Defense's spending on Unmanned Aerial Systems (UAS). Post-9/11, the expenditure on drones surged from \$363 million in 2001 to \$2.9 billion in 2013 as shown in Figure 2.10. Moreover, between 2002 and 2010, the Department of Defense's UAS inventory witnessed a staggering 40-fold increase. While unmanned aircraft comprised only 5% of military aircraft in 2005, by 2012, UAVs accounted for one-third of the entire fleet. In 2003, a mere 163 UAS were in use, but by 2012, the inventory reported a remarkable total of 7,454 UAS—an astounding surge of over 4,400% in less than a decade. The political economy framework sheds light on the factors driving this dramatic expansion in the drone industry.

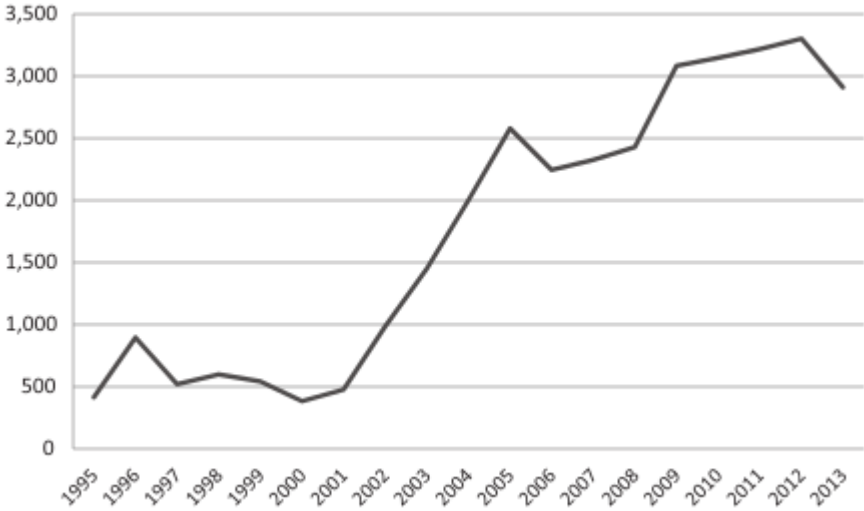


Figure 2.10 - USA Department of Defense spending on UAS [20]

Drones have revolutionized military operations with their compact design, advanced features, and ease of deployment. UAVs excel in data gathering and provide valuable situational awareness. Their attack capabilities offer a significant advantage in engaging targets quickly compared to ground units. Drones are highly adaptable, capable of operating in challenging environments and remaining airborne for extended periods. They serve multiple purposes, including land surveying

and measuring coverage across various terrains. The evolving nature of conflicts in Iraq and Afghanistan, spanning diverse landscapes, further highlighted the value of UAV technology.

While the military demand generated by Afghanistan, Iraq, and the broader War on Terror played a central role in propelling the drone industry's expansion, it was not the sole driver of growth. Established industry giants in the defense sector, such as Lockheed Martin, Northrop Grumman, Boeing, General Dynamics, and General Atomics, recognized the potential for significant profits presented by US military engagements. The allure of defense spending, including investments in UAV technologies, was a clear incentive. By 2012, global governments were already allocating over \$6.6 billion USD annually to UAVs, with projected profits expected to reach \$11.4 billion per year over the next decade, contributing to a market worth over \$89 billion. The Government Accountability Office revealed that the 2012 acquisition plan by the Department of Defense required substantial payments to drone manufacturers, totaling more than \$34.9 billion for the desired quantity of drones. Beyond manufacturing new drones, the industry also anticipates additional revenue from drone repairs and modifications. Given these promising prospects and the increased military demand, the political economy model correctly predicted intense rent seeking by stakeholders in the defense industry.[20]

2.3 UAV applications

The number of applications where UAVs become a useful tool seems almost unlimited and is continually growing. Yet the most relevant are photogrammetry, agricultural and forestry, humanitarian aid, merchandise delivery, cultural heritage and archaeology, atmospheric observation, air analysis and pollution, monitoring natural disasters and specifically wildfires. These mentioned areas are explained in the following subsections.

2.3.1 Photogrammetry

Photogrammetry is the art and science of extracting 3D information from photographs. Remote measurements from photos taken by UAVs with enough sensor technology can then be processed to create 3D terrain mapping (see Figure 2.11) with Digital Elevation Models containing forms, surface reconstruction, elevation contours, or features. Orthographic images are also final or intermediate products that are important for cartography and topography. In most cases, photogrammetry serves as the foundation for additional applications. It can also be used for inspection in cultural heritage and archaeological applications, for historical building restoration and damage evaluation after a natural disaster or acts of vandalism.[21]



Figure 2.11 - Historical layer model obtained by use of photogrammetry - 3D model[22]

2.3.2 Agricultural and forestry

Through extremely different and varied implementations, UAVs are just recently becoming a component of remote sensing applications in agriculture and forestry. Some of them are improving in performance while being more affordable than traditional platforms. Drones can fly at lower altitudes and slower speeds than satellites and aerial remote sensing systems, providing near-earth data including wildlife populations, biophysical characteristics, gaps in the forest canopy, and single tree identification.

In the context of agricultural and forestry, the control of biophysical variables is of special interest for various purposes, such as chlorophyll and biomass determination for forest site specific treatments. Crops and weed management in precision agriculture are two key activities for different purposes, such as yield estimations, herbicide applications (Figure 2.12), and pesticide control resulting in cost savings and minimal environmental impact. Many applications in crops are oriented to the generation of maps for monitoring weed infestations and coverage, biomass estimation, yield prediction, or crop stress. Imaging maps are commonly georeferenced and orthorectified, where positioning accuracy becomes an important consideration in map generation.[23]

The estimation of biophysical parameters can be done using several vegetation indices. The leaf area index, chlorophyll content, water stress detection, plant health, canopy analysis, photosynthesis, mapping of areas (including 3D), and soil analysis are just a few examples of the image-based products that can be produced based on these factors.[25]



Figure 2.12- Spraying drone for herbicide applications [24]

2.3.3 Humanitarian aid

Humanitarian aid, that is, providing essential supplies to otherwise inaccessible people in war or disaster areas is another useful application of drones. Similarly, medical delivery (see Figure 2.13) of organs and blood is another time-critical, life-saving application that has previously seen field trials[26]. Per example, during Covid-19 pandemic, it helped to reduce the virus infections spread as they were used to deliver automated external defibrillators, protective equipment like gloves and face masks, collect blood samples for analysis.[27]



Figure 2.13 - Zipline's company drones delivering medical supplies [28]

2.3.4 Merchandise delivery

Lightweight merchandise delivery has the potential to reduce the cost and carbon footprint of the enormous logistics and delivery industry. Several companies like Google and Amazon are taking

initial steps in the drone delivery business. The cargo transported by each drone should not be heavy because the drones utilized by these companies are small-sized.[26]

2.3.5 Atmospheric observation, air analysis and pollution

Drones are taking on new challenges in atmospheric observations, some have been specifically designed for such purposes. The most significant information gathered is: air gas concentrations, aerosol concentrations, ozone concentrations, temperature, humidity, pressure, and wind fields. Unmanned gliders are launched from heights and recover this data for analysis during the auto-piloted descent.[23]

2.3.6 Monitoring natural disasters

Since their inception, and even more recently, unmanned aerial vehicles have been considered beneficial for operations in disaster zones if they are conveniently fitted with cutting-edge sensors and technologies. Nuclear incidents, spills in the ocean, floods, avalanches, earthquakes and wildfires are clear causes of disaster where UAVs can play an important role. These are examples of man-made catastrophes where drones have been used, although they are not exclusive and can lay the groundwork for other disaster applications in the future.

For instance, nuclear leaks are a high-risk occurrence that present significant hazards at all levels, frequently at large distances from the incident, making it imperative to take rapid action in the aftermath. The nuclear leaks at Chernobyl (Union of Soviet Socialist Republics (USSR), 1986) and Fukushima (Japan, 2011) are well-known examples.[23]

Oil slicks and perhaps other pollutants on the sea surface are becoming more prevalent creating large, contaminated areas. Remote sensing is frequently employed in the marine environment to locate, monitor, and map oil spills on a water surface. Typically, a variety of airborne or spaceborne remote sensing techniques, such as thermal scanners, fluorosensor lasers, and hyperspectral imaging, are utilized to achieve this goal.[29]

In floods, drones can be used with antennas that identify transmitters on the ground, building a distribution map. When a flood occurs the movements of these devices on the ground will be detected. In hurricane Katrina, 2005 in the United States of America (USA), (see Figure 2.14) drones were essential for the aftermath because they could fly through inaccessible areas.



Figure 2.14 - Flood devastation caused by hurricane Katrina in the USA [30]

In avalanches, drones can be used for the study and detection of temporal evolution in wet snow in avalanche prone areas but also for search and rescue under the snow with infrared radiation (IR) cameras. For earthquakes, stereoscopic techniques can be used for 3D reconstruction of buildings for determining possible damage for post seismic analysis. After a disaster it can provide rapid intervention giving abundant and detailed information with imaging-based surveillance tasks.[23]

2.3.7 Wildfires

In nature, forests fulfill a number of significant roles. They have the ability to clean water, maintain stable soil, cycle nutrients, control climate, and store carbon. Additionally, they support environments rich in biological diversity and provide habitat for species. Economically speaking, forests support the forest products business, which generates billions of dollars in revenue and hundreds of thousands of jobs.

The use of UAVs, equipped with sensory technologies, was early identified for its potential use in fire detection. UAVs have developed to more advanced, precise, and high-performing technology and methodologies. The coordination between fire brigades during crisis management in fires is crucial for putting the fire out during its evolution stage (see Figure 2.15). As part of cooperative surveillance, UAVs scan different areas. If a fire is discovered, they position the incident using the system's GPS to provide the response brigades with the fire measurements and remotely sensed locations they need to plan their resources for fighting the fire or for civilian rescue. Flames can

be followed if they have thermal or IR imaging equipment. If the drone has chemical sensors, it detects components of gases that identify the existence of fire.[23]

As a result of forest fires spreading quickly through convection and a quick combustion cycle, it is essential to identify and put out them as soon as possible. This will help to reduce the amount of damage they may inflict. Traditional forest fire monitoring and detection techniques use either mechanical tools or people to keep an eye on the area, but both techniques can be risky and expensive in terms of the human resources needed.



Figure 2.15 - Different stages in the evolution of a fire [31]

Satellite systems' path planning and technological updates are less flexible, and their temporal and spatial resolution may be insufficient for operationally capturing precise data and battling forest fires. Typically, manned aerial vehicles are big and expensive. Furthermore, dangerous conditions and operator tiredness may pose a threat to the pilot's life.

The expansion of UAV applications in the fire domain area mainly pertains to the domain of wildfire remote sensing. When dealing with uncontrolled fires, manned aircraft wildfire monitoring is expensive and extremely dangerous. Thus, moving the aircraft operator from the air to the ground increases the effectiveness and efficiency of attempts to put out wildfires while freeing up resources for other firefighting-related tasks. UAVs are capable of performing a variety of functions in addition to remote sensing, such as aerial lighting of prescribed fires and even

firefighting, however the latter has not yet been developed in practical contexts due to the need to transport large volumes of water and fire retardant.[32]

Fire mapping, which highlights the spots that are on fire at a specific time on a map of a region using georeferenced aerial imagery, is the most frequent mission in the field of wildfire remote sensing. The fire maps can also be processed to establish the current fire perimeter and to provide an estimate of the fire's location in unseen areas. Monitoring is the process of continuously mapping, such as when tracking a fire's perimeter to update the fire map on a frequent basis.

Additionally, the obtained data aids in the development of 3D flame reconstruction algorithms using aerial stereo imagery, which helps academics understand how fire spreads. Aerial thermal IR imaging can also be used in automated wildfire monitoring (see Figure 2.16). Wildfire perimeters are tracked and this information is used to improve the parameters of its propagation simulator. Such wildfire prognosis capability could be integrated in an automated fire decision support tool.[33]

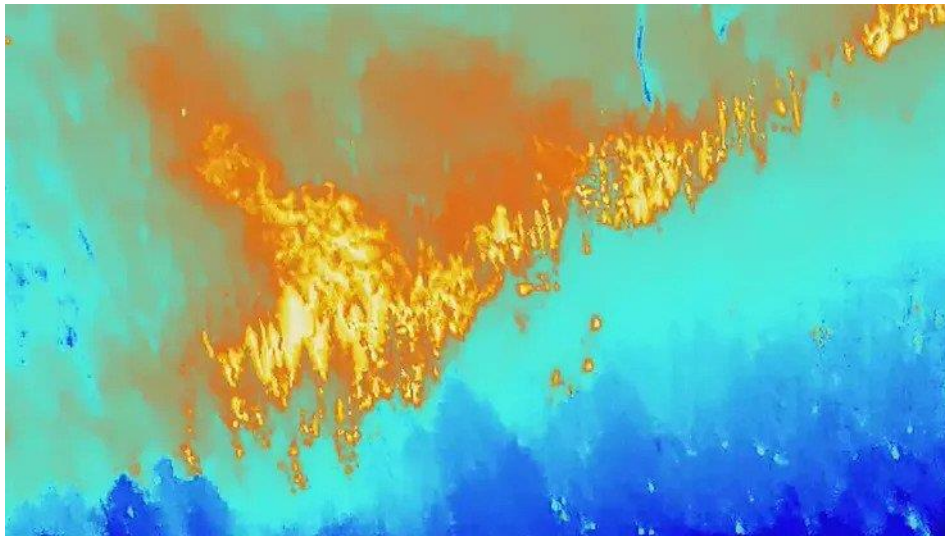


Figure 2.16 - IR image of wildfire hotspots from drone camera [30]

The earliest application of UAVs for gathering information on forest fires can be traced back to 1961 by the United States Forest Services Forest Fire Laboratory. UAV-based forest fire systems typically include the following:

- various frames and sensors, including GPS, IMUs, cameras;
- specific algorithms and strategies for fire monitoring, detection, diagnosis and prognosis;
- Guidance, Navigation and Control (GNC) systems for both single and multiple UAVs;

- cooperative localization, deployment and control systems for UAV fleets to optimally cover fire areas;
- a dedicated ground station that includes equipment for communication, ground computation, visualization of fire detection, tracking and prediction with automatic fire warning or alarm, as well as all equipment necessary for the safe and efficient operation of UAVs.

UAV forest fire fighting missions can generally be broken down into three stages: fire search, fire detection, and fire observation. Fire search with visual images can be done with color and motion features based on RGB (red, green and blue) model combined with fire detection algorithms. It can also be done in conditions of weak or no light using IR images.

It is also useful to use IR sensors during daytime since IR images are not affected by smoke. To improve the accuracy, reliability, and robustness of fire detection algorithms and to reduce the rate of false alarms, visual and IR images can be fused together, generally using fuzzy logic, probability, and statistical methods.

After detection, the segmented fire area requires geolocation information for measuring its geometrical features such as fire front location, fire site width and perimeter, flame length and height, inclination angle, coordinates of burnt areas and location of hot spots. Meanwhile, the location of the UAVs themselves is known using Global Positioning System (GPS) and with the help of Ground Central Stations for information exchange (Figure 2.17). The orientation of the UAVs camera is computed by composing the orientation angles of the pan and tilt system with the orientation angles of the UAV airframe, which is estimated by IMUs and compasses.

Fire observation is detrimental to predict the behavior of fire propagation and is essential for developing quick, effective, and advanced forest fire fighting strategies. Hence, it is important to know the evolution of the fire front, as well as other properties of the fire such as rate of spread as well as the ones mentioned earlier in the fire detection and confirmation period.

Rate of spread, in particular, is one of the most significant parameters in the characterization of forest fire behavior, as it is directly associated with fire intensity and flame front geometry, two key indicators of the danger levels of fire propagation.[32]

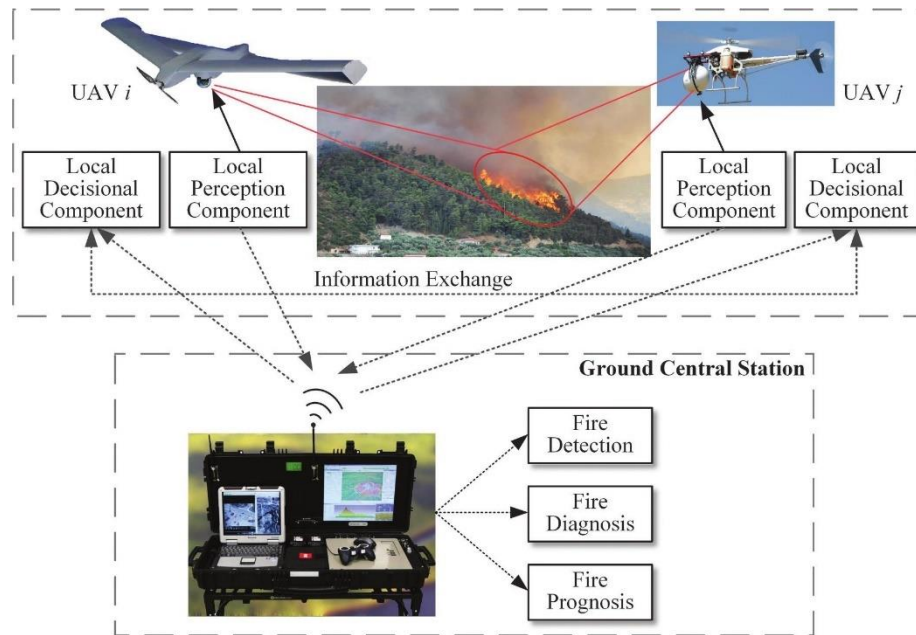


Figure 2.17 - Conceptual UAV- based forest fire monitoring, detection and fighting system [32]

Forest fires are highly sophisticated and non-structured so, it is crucial to use multiple sources of information at different locations. Furthermore, the rate at which this information is updated may be unsatisfactory if only a single UAV is deployed for either a single, large-scale forest fire or for multiple forest fires. More efficient forest fire monitoring, detection, and even fighting can be achieved when a fleet of multiple UAVs are deployed instead of a single UAV. However, this also requires that more practical algorithms for task assignment and cooperative control for multiple UAVs be developed. These algorithms are intended to achieve efficient cooperation between vehicles for optimal coverage, as well as deployment for the most efficient fire detection, tracking, and prediction in the cases of large, multiple, and simultaneous fire events.

In order to combat forest fires, single UAV systems can be used. Remotely piloted High Altitude Long Endurance UAVs are used alongside satellite monitoring systems for extended flights and carrying heavy payloads. However, they are expensive and do not offer more precise data compared to satellites.

Between 2006 and 2010, National Aeronautics and Space Administration (NASA) and the US Forest Service conducted new imaging missions using a HALE UAV equipped with a hyperspectral camera and on-board computing power. These missions successfully detected burning areas by applying a threshold to specific IR bands. Additionally, the OSIRIS project in Europe aims to develop a solar-powered HALE UAV for wildfire monitoring. This UAV can follow a predefined flight plan, which can be updated from a Ground Control Station. During its mission, the aircraft captures high-resolution images and transmits them in real-time to the ground station via a satellite link.

The use of multiple UAVs offers new forms of operation. The extent of a wildfire is frequently too vast to be covered by a solitary Unmanned aircraft, a challenge that can be addressed by fleets of UAVs. But systems built upon multiple UAVs also imply new design challenges, as vehicles must communicate and collaborate in some way to exploit the full potential of the fleet.

There are several approaches to using UAV systems for wildfire detection and monitoring. One method involves using a fleet of UAVs equipped with binary sensors to observe the fire front defined by control points. The observations are processed by a Kalman filter that estimates the location of the perimeter control. After that a planning algorithm prioritizes the observation of uncertain control points.

Another example involves an early fire detection platform that uses ground sensors and UAVs. Small quadrotor drones are used for early fire verification, while a blimp equipped with smoke detectors and radiometers monitors hot spots.[33]

2.4 UAV types and examples

The first characteristic is based on aerodynamic factors. UAVs can be classified into three types: a) fixed-wing, b) rotary-wing and c) hybrid (Figure 2.18).

The first type (fixed-wing) possesses a predefined airfoil and fixed -wings that enable lift based on the UAV forward airspeed. The control of such a UAV is accomplished through elevators, ailerons and rudder that are attached to the tailplane, wings and vertical tail, respectively. These construction characteristics enable UAV to turn around pitch, roll and yaw axes.

The second type (rotary-wing) is composed of rotors that generate the appropriate power necessary for vertical take-off. Based on this airflow and in contrast to the first one (fixed-wing), this type does not need a forward airspeed for lifting. Accordingly, the control of such a UAV is based on the torque and thrust of the rotors. For instance, the speed of the diagonal rotors determines the yaw movement. More specifically, depending on the number of rotors, a rotary-wing UAV can be classified into the following categories a) tricopters, b) quadcopters, c) hexacopters and d) octocopters. It is noteworthy that each of the types mentioned above presents the corresponding pros and cons. For example, a rotary-wing UAV can hover in place for extended periods which allows to perform tasks such as observation and firefighting with greater precision. On the other side, a fixed-wing UAV presents an efficient and simpler architecture facilitating the maintenance processes and is also characterized by a longer flight duration and larger coverage.[34]

Finally, there is a third type (hybrid-wing) which combines the previous ones. In particular, this type possesses rotors for taking off and landing, but also includes fixed wings utilized for covering large areas.[35]

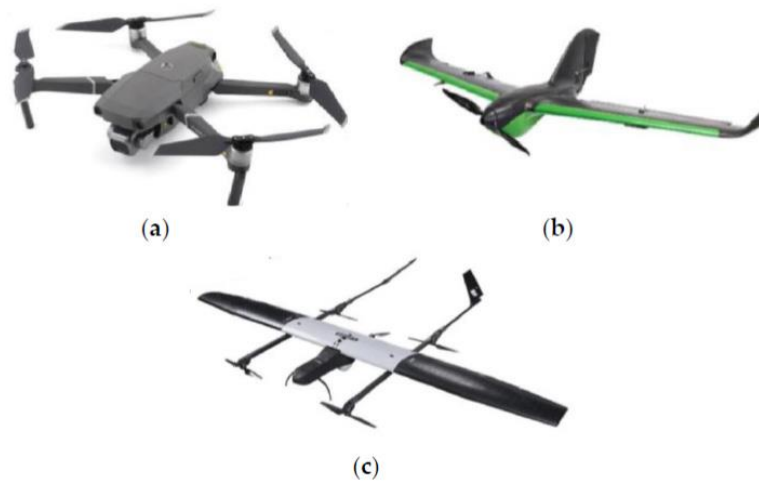


Figure 2.18 - Different types of UAV. a) rotary-wing; b) fixed-wing; c) hybrid-wing [36]

A second characteristic is the level of autonomy of the drone. The autonomy can vary from full autonomous operation to fully controlled by a remote pilot. There are four types of autonomy.

The first type, named human-operated system defines that the system operator is responsible for controlling all operations of the unmanned system. The second type called human delegated system is characterized by a higher level of autonomy compared to the first one, by maintaining the ability to take autonomously some restricted decisions. The third level is named human supervised system and can take various decisions based on the directions of the system operator. Specifically, in this case, both the system operator and the unmanned system can perform various actions based on the data received. Finally, the last level is named fully autonomous systems and is responsible for all its operations. In this case, the unmanned system receives data from the system operator and interprets it into specific tasks. Surely, in the case of an emergency, the system operator has the ability to intervene in the function of the unmanned system.

Another noteworthy characteristic is weight. It can vary from several grams to hundreds of kilograms. Several countries and researchers have categorized UAVs based on their size and weight. If a UAV exceeds the weight of 150 kg, then it is characterized as heavy. Otherwise, it is specified as light. Other classifications separate the fixed-wing and rotary-wing. It considers that fixed-wing UAVs whose weight is between 20 kg and 150 kg can be characterized as large. On the other side, if a fixed-wing UAV does not exceed 20 kg, then it can be characterized as small. Similarly, rotary-wing UAVs whose size ranges from 25 kg to 100 kg, are considered large. Accordingly, if a rotary-wing UAV does not exceed 25 kg, then it is small.

Moreover, it is considered that the small UAVs can be distinguished further, extracting a new subcategory called mini. These UAVs are those whose weight ranges from some grams to several kilograms.[37] DJI Mavic mini (Figure 2.19) has a takeoff weight of 249 grams.



Figure 2.19 - DJI Mavic mini [38]

Finally, UAVs can also be categorized based on the power source utilized for their flight. There are four main types of energy for a UAV: a) kerosene, b) battery cells, c) fuel cells and d) solar cells.

Kerosene is usually employed by large fixed-wing UAVs appropriate for military purposes. An example of such a UAV is the Predator. Conversely, the small rotary-wing UAVs incorporate battery cells, since their functional needs require less operating time. An example of such a UAV is DJI Phantom. A fuel cell is an electric device which transforms chemical substances into electrical energy.

Finally, solar cells can be used for both fixed-wing UAVs and rotary-wing UAVs. Google and Facebook have already directed their attention towards UAVs using this technology, with the aim of elevating such UAVs within the atmosphere, thus enabling a more extensive Internet connection.[35]

2.5 Flight controllers software and simulators

2.5.1 Flight controllers software

Software flight controllers are essential components in the development and operation of autonomous systems, particularly UAVs. They handle critical tasks such as sensor data processing, control algorithm execution, and ensuring stable flight. With a variety of software flight controllers available, it is crucial to observe the pros and cons of each.

The flight controllers showcased are the Multiwii Series flight controllers which include: Baseflight, Cleanflight, Hackflight, Betaflight, INAV and Raceflight (which has stopped developing). The OpenPilot Series flight controllers which include: LibrePilot, Tau Labs, and dRonin. Lastly, the two most famous: ArduPilot and PX4.

Multiwii Series - The 1st multiwii Series(see Figure 2.20) flight controller is known for its user-friendly interface and straightforward setup process. It is often recommended for beginners who are new to autonomous systems development. The flight controller may have limitations in terms of sensor integration options compared to more advanced flight controllers. This could impact the ability to incorporate specific sensors or advanced features in the system. Development of the Multiwii Series flight controller has stopped, with the focus shifting to more modern flight controllers. This may result in a lack of support for newer technologies or compatibility issues with contemporary software frameworks[39]–[41]. Baseflight came after the first multiwii series controller.

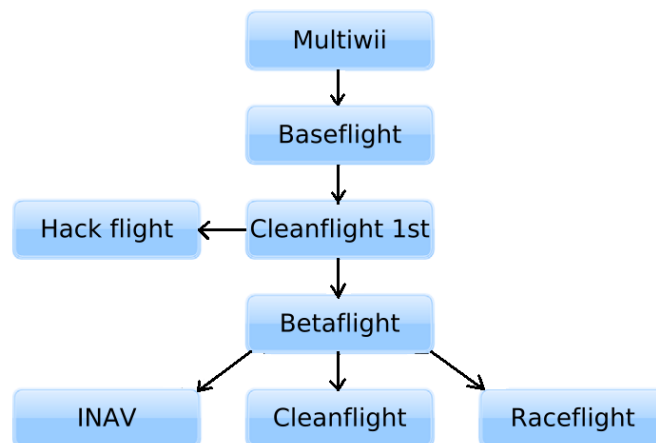


Figure 2.20 - Development history originating from the original Multiwii flight controller software [42]

Cleanflight – is an open-source flight controller software which is a 32-bit version of the original 8-bit MultiWii code. Cleanflight offers extensive configurability options, allowing users to fine-tune flight parameters, control settings, and sensor calibrations. This level of customization enables users to optimize their drone's performance and flight characteristics. Cleanflight is closely integrated with Betaflight, another popular open-source flight controller software. This integration allows users to benefit from the best features and optimizations of both Cleanflight and Betaflight, providing a comprehensive suite of options for drone control.[43], [44]

Betaflight – It is the largest flight firmware in the first-person view (FPV) drone racing and freestyle community due to its cutting-edge performance, features, reliability and wide range of hardware support.[44], [45]

INAV – offers seamless integration with GPS modules, enabling accurate position hold, return-to-home functionality, and autonomous missions based on GPS coordinates. This feature is beneficial for applications that require accurate positioning and navigation capabilities. INAV supports both multirotor and fixed-wing aircraft, making it a versatile option for users who work with different types of drones. While INAV offers a user-friendly interface, configuring advanced features and customizing complex flight parameters may require a deeper understanding of the software and its underlying concepts.[46]

OpenPilot Series – is a flight controller software package that has been developed through many projects and by different developers. OpenPilot (Figure 2.21) was the original flight controller software in this series, but development was stopped in 2015.

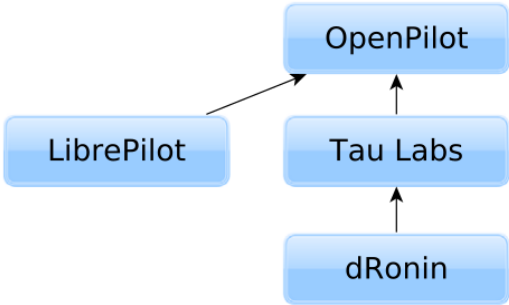


Figure 2.21 - Development history of the OpenPilot series flight controller software [42]

LibrePilot – started as a fork of OpenPilot in 2015 and supports a wide range of sensors, including gyros, accelerometers, magnetometers, and barometers. This allows users to gather accurate data for flight control and stabilization, enabling smooth and precise flying. LibrePilot offers a user-friendly interface, making it easier for users to configure and customize flight parameters. The intuitive interface simplifies the setup process and allows users to quickly get their drones up and running. LibrePilot may face compatibility issues with newer flight controller hardware or components.[47], [48]

ArduPilot – benefits from a large and active development community. This ensures frequent updates, bug fixes, and continuous improvements to the software, keeping it up to date with the latest advancements in autonomous systems. the controller can run under Linux, enabling it to be used on a large class of electronics from single-board computers all the way up to a full Portable Computer (PC) system. ArduPilot has a desktop Ground Control Station for mission planning, calibration, and vehicle setup for Windows, Linux and Mac ArduPilot requires specific hardware components for optimal performance.[49], [50]

PX4 – The PX4 flight stack and autopilot are a part of the DroneCode collaborative project which covers both Ground Control Station, Pixhawk hardware platforms and simulation. PX4 is built around the MAVLink communication protocol, which facilitates interoperability and integration with other systems and components. This allows seamless communication between the flight controller and companion computers and other MAVLink-compatible devices. It supports a wide range of vehicles, including multirotors, fixed-wing aircraft and more, providing versatility for various applications.

When encountering issues or bugs, troubleshooting and debugging in PX4 can be challenging, especially for users with limited technical expertise. Comprehensive debugging tools and resources may be required to identify and resolve complex issues.[51]–[54]

2.5.2 Simulation platforms

Simulation platforms play a crucial role in the development and testing of autonomous systems, such as drones, self-driving cars, and robotics. These platforms provide a virtual environment where developers can assess the performance, validate algorithms, and refine control strategies before deploying their solutions in the real world. However, with a multitude of available open-source simulation platforms, selecting the most suitable one for a specific application can be a daunting task.

Below a comprehensive comparative analysis of several popular open-source simulation platforms is provided, highlighting their advantages and disadvantages. By understanding the unique features and limitations of each platform, it is easier to make an informed decision about which simulator best aligns with this project requirements.

AirSim – provides a high-fidelity simulation environment for autonomous vehicle development, including drones and self-driving cars. It offers realistic physics modeling, accurate sensor

simulation, and realistic rendering for immersive simulations. It has a user-friendly interface, making it accessible to developers at different levels of expertise. It provides an easy-to-use Application Programming Interface (API) and graphical user interface (GUI) for scenario creation, configuration, and control of simulations.

However, running AirSim simulations with high-fidelity graphics and complex scenarios may require a powerful computer with a capable graphics processing unit (GPU) and sufficient computational resources.[55]–[57]

MORSE – supports the simulation of various sensors commonly used in robotics, including cameras, lasers, and range finders. This feature allows developers to generate simulated sensor data and evaluate the performance of perception algorithms without the need for physical sensors. Morse integrates well with Robot Operating Systems (ROS).

Compared to some other simulation platforms like Gazebo or AirSim, Morse may have a smaller user community and a more limited availability of resources.[58], [59]

jMAVSim – seamlessly integrates with the MAVLink communication protocol, which is widely used in the drone ecosystem. This integration enables developers to test and validate their software and control systems in a simulated environment that closely resembles real-world drone operations.

While jMAVSim provides a fast and efficient simulation of drone flight dynamics, its sensor and environment simulation capabilities may be more limited compared to more advanced simulators like AirSim or Gazebo.[59]

Gazebo – supports a wide range of sensors commonly used in robotics, including cameras, LiDAR, GPS, and IMU. It allows developers to generate simulated sensor data, enabling the evaluation and testing of perception algorithms without the need for physical sensors. It offers a highly customizable and flexible environment. It allows users to create and modify virtual worlds, models, and robot behaviors to match specific project requirements. This flexibility makes it suitable for a wide range of applications and research purposes. Gazebo seamlessly integrates with ROS. This integration allows for easy communication with ROS nodes and the use of ROS tools, libraries, and packages within the simulator environment. It is perfect for drone flight simulations (see Figure 2.22).

Gazebo has a steeper learning curve compared to some other simulation platforms. It requires users to understand concepts such as simulation world setup, models, and the plugin system. However, the extensive documentation and community support help mitigate this challenge.[60], [61]

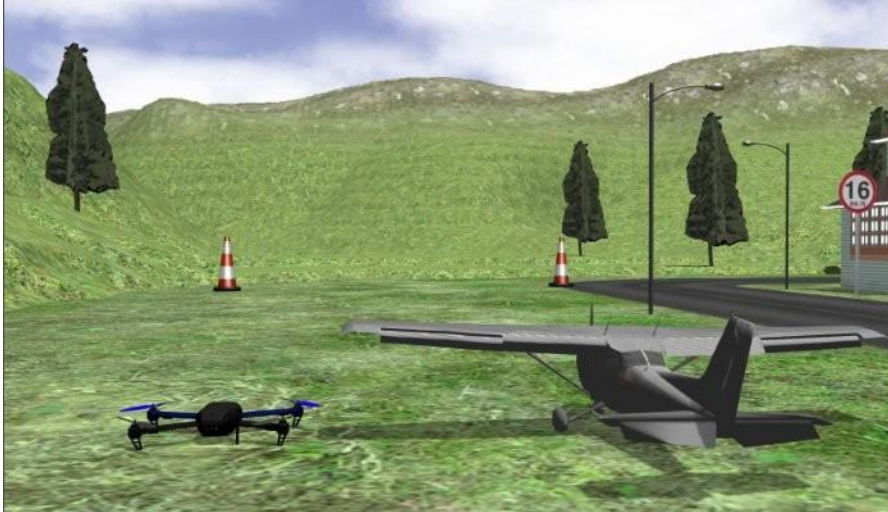


Figure 2.22 - Inside a gazebo world simulation [62]

Chapter 3

Methodology

This chapter outlines the methodology employed in this dissertation, elucidating the steps that culminated in the simulation of drone control, an integral component of the wildfire tracking drone project. The process commenced with the calculation of UAVs' moments of inertia, progressing through the depiction of the laboratory setup and the development of the multirotor CAD model, ultimately encompassing the fundamental drone control functionalities.

In a summarized format, this dissertation has proceeded in the following stages:

1. Measurement of the drone operating space inside the Laboratory of Aerodynamics and Propulsion in the Department of Aerospace Sciences at UBI.
2. Modeling of the referred space in a virtual environment (Gazebo Simulator), defining the limits of operation.
3. Calculating UAV moments of inertia and inserting that data in the UAV model code which allows realistic flight path to the final objective of wildfire tracking.
4. Virtually modeling the UAV based on a real multirotor and on its moments of inertia.
5. Creation of basic control functions (in Python) that send commands to the drone based on a distance to a certain target position.
6. Virtual implementation of the multicopter and its results.

3.1 Software Tools

The software tools used in this dissertation, are the following ones:

Robot Operating Systems

ROS is a widely used platform in the field of robotics, developed by a global community of millions of developers and users over a decade. ROS is used for teaching robotics, cutting-edge research projects, and deployed in production robots worldwide. It is versatile and adaptable, being an open-source platform that allows developers to customize it to suit their specific needs and seamlessly integrate with existing software stacks. This program is designed for a wide range of robotics applications (including drones) and is supported on multiple operating systems. Its

open-source approach fosters interoperability and collaboration within the robotics community.[63]

ROS has a package called MAVROS that acts like a bridge between the autopilot system such as PX4 and itself. It provides a set of ROS messages and services for MAVLink protocol.

Gazebo

Gazebo is a 3D dynamic simulator with the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. While similar to game engines, Gazebo offers physics simulation at a much higher degree of fidelity, a suite of sensors, and interfaces for both users and programs.

Typical uses of Gazebo include testing robotics algorithms, designing robots, performing model testing with realistic scenarios. A few key features of Gazebo include multiple physics engines, a rich library of robot models and environments, a wide variety of sensors, convenient programming and graphical interfaces.[64]

Microsoft Excel

Microsoft Excel is a software that utilizes spreadsheets for organizing data, programming, data analysis and calculations. For the present work, this software was used for volumetric mass calculations of the different drone parts and to obtain the proportion between a real wildfire and a laboratory test fire.

CATIA V5

CATIA V5, is a multi-platform software for CAD, Computer-Aided Manufacturing (CAM), Computer-Aided Engineering (CAE), 3D modeling, and Product Lifecycle Management (PLM) made by Dassault Systems. This software is utilized for obtaining the 3D model of the aircraft and for calculating the UAVs moments of inertia.

Microsoft 3D Viewer

The Microsoft 3D Viewer is a software application that allows users to view and interact with 3D models on Windows-based devices. It supports various 3D file formats (including Standard Tessellation Language (STL) and Digital Asset Exchange (DAE) format), provides viewing, editing, and basic texture capabilities, and is compatible with virtual reality and augmented reality environments.

Visual Studio Code

Visual Studio Code is a lightweight but powerful source code editor which runs in a desktop and is available for Windows, macOS and Linux. It comes with built-in support for JavaScript, TypeScript and Node.js and has a rich ecosystem of extensions for other languages and runtimes (such as C++, C sharp, Java, Python, Go).[65]

QGroundControl

QGroundControl (QGC) is a software that provides flight control and vehicle setup for PX4 or ArduPilot powered vehicles. It is user-friendly for beginners yet offers advanced features for experienced users. Key features include setup/configuration for ArduPilot and PX4 Pro vehicles, flight support for various autopilots, mission planning for autonomous flight, flight map display, video streaming with overlays, multi-vehicle management, and compatibility with Windows, Mac, Linux, iOS, and Android platforms.[66]

PX4 Autopilot

PX4 is an open-source flight control software stack for UAVs and autonomous vehicles. It provides a flexible and extensible platform for controlling, navigating, and executing missions. With its modular architecture, it supports various UAV platforms and offers features like firmware for flight controllers, middleware for communication, autonomous flight modes, simulation capabilities, and compatibility with ground control stations. PX4 is widely used and has an active community of developers.[67]

3.2 Test laboratory

In order to carry out this task, measurements were taken of the test area inside the Aerodynamics and Propulsion Laboratory in the Aerospace Sciences Department at UBI.

This area is located on the upper floor of the laboratory and has the following dimensions:

- 9 meters in length, 4.1 meters in width, and 2.6 meters in height.

It is a place without obstacles, trying to recreate its operating environment in a real context, that of monitoring a forest fire. Although in Figure 3.1 and 3.2, the testing space of the drone is occupied by scaled-down aerial models, they are removed before the UAV flight. The height is limited by a truss beam linked to the roof and the width by the ventilation ducts attached to the side wall.



Figure 3.1 - Laboratory in aerospace sciences department at UBI



Figure 3.2 - UAV testing area in the upper floor

The upper floor area is then represented at a virtual level through the Gazebo simulation software. In Section 4.2, the test laboratory virtual representation is explained in terms of its code, how it was adapted from a default file of the simulator and the creation of the fire pattern models with their corresponding sizes.

These fire patterns are a scaled representation of real fires, depicting their shape and size. They take on various forms: continuous and discontinuous, curved and straight. Aerial images from manned and unmanned vehicles reveal exactly that.

These three examples below reveal that wildfires have shapes similar to letters of the Latin alphabet but a little bit distorted. In Figure 3.3, an “F” shape is considered, in Figure 3.4 an “S” and in Figure 3.5 a “V” shape.



Figure 3.3 - Wildfire aerial view Yakutsk, Russia[68]



Figure 3.4 - Wildfire aerial view near Izmir, Turkey[69]



Figure 3.5 - Wildfire aerial view near Chiang Mai, Thailand[70]

3.3 Multirotor CAD model

In order to obtain the multirotor virtual display, the following steps were proceeded:

- The drone was disassembled into 25 distinct parts. Their mass (in grams) and geometric dimensions were measured.
- Each part was drawn in CATIA V5.
- From this moment on, the “CATProduct” file was converted into a “CATPart” file, making the drone one whole part.
- After that, tessellation is applied (see figure 3.6) and converted into a STL file.
- To preview and inspect the drone model before adding it into the simulation, Microsoft 3D viewer is utilized.
- Then, the STL file is placed in the “meshes” folder within the “typhoon_h480” drone folder that is used as an adaptation to this multirotor. This file is then called from within the code that defines the UAV, “typhoon_h480.sdf”, making way to the appearance of the multirotor virtual display, that is explained in more detail in section 4.4.

In figures 3.6, 3.7 and 3.8 it is demonstrated that the axis system has the x -axis facing the front of the drone, the y -axis facing to the right, and the z -axis facing downwards. It is noted that the GPS antenna is located on the rear of the drone. The images of the drone in CATIA V5 have rotors hidden from view to facilitate the visualization of the axes. The exact place of the center of the axis system is calculated in the next Section.

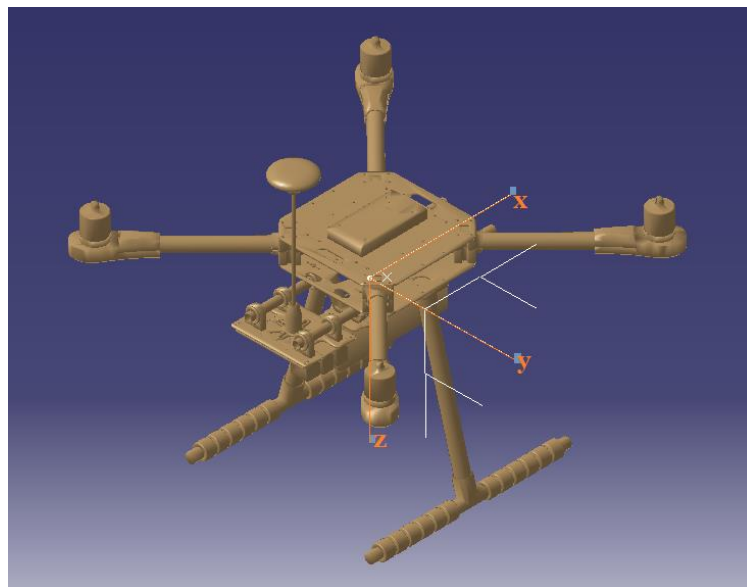


Figure 3.6 - Tessellation in CATIA V5

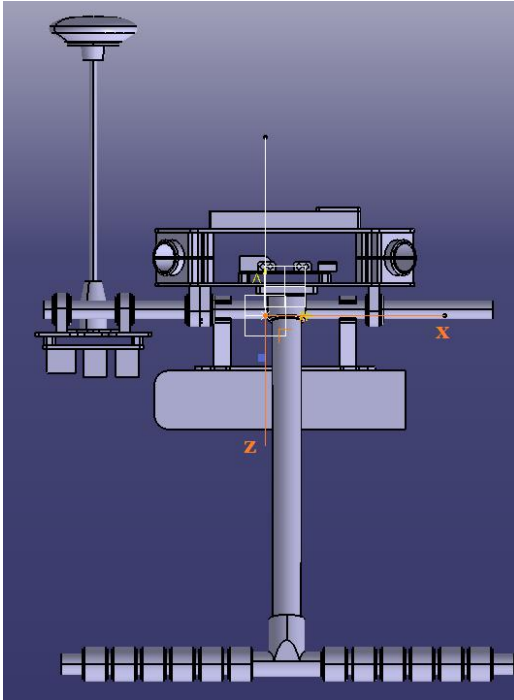


Figure 3.7 - Side view of the UAV in CATIA V5

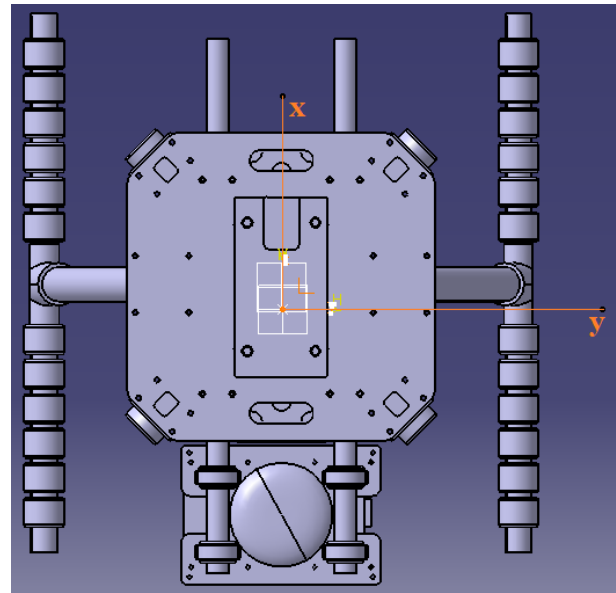


Figure 3.8 - Upper view of the UAV in CATIA V5

3.4 UAV moments of inertia

In order to obtain the moments of inertia, it was proceeded with the following steps:

- Using the “Measure Inertia” tool, the volume of each part was obtained in cm^3 .
- Gathering this data, the density of each part was calculated and placed in an Excel sheet.
- A file called "CATMaterial2" was created which has a material assigned to each part of the drone with its respective density.
- In the file that compiles all the parts, called "Product1", the "Measure Inertia" option was used and acquired the UAV center of mass.
- An axis system was created with the origin at the UAV's center of mass.
- With this axis system, it is finally computed the inertia matrix.

The following Table 3.1 indicates the mass, volume and density of each part that makes up the drone. The mass was measured a digital scale which measures to the nearest tenth of gram and the volume was calculated with a standard ruler.

Table 3.1- Mass and volumetric mass of each part of the UAV

part	mass, g	Volume, cm^3	Density (g/cm^3)	Density (kg/ m^3)	Number of parts
Arm	53.73	36.74	1.46	1462.44	4
Battery	435.47	192.89	2.26	2257.66	1
Battery plate	17.22	8.34	2.06	2064.75	1
Bottom main plate	58.52	35.21	1.66	1662.12	1
Connector arm frame	13.56	10.45	1.30	1297.98	4
Connector leg frame	10.00	6.10	1.64	1639.34	2
Connector leg land Gear	11.46	5.94	1.93	1928.64	2
GPS ant	32.87	20.68	1.59	1589.84	1
GPS base	10.61	4.89	2.17	2167.96	1
GPS top base	3.99	1.95	2.05	2048.26	1
GPS tube	2.61	1.76	1.48	1483.80	1
Hanger	1.90	1.46	1.30	1297.81	10
Hanger rubber	1.52	1.08	1.41	1411.33	10
Landing Gear	16.71	45.75	0.37	365.23	2
Leg	13.82	9.42	1.47	1467.87	2
Motor	65.98	19.62	3.36	3362.21	4
Nylod stud	0.12	0.10	1.25	1250.00	8
Payload tube	13.47	7.07	1.91	1905.50	2
pdb	22.18	7.70	2.88	2882.02	1
Pixhawk 4	31.82	37.68	0.84	844.55	1
Power Module	23.99	9.16	2.62	2619.85	1
Rpi	50.45	21.50	2.35	2346.73	1
Rpi plate	19.99	9.72	2.06	2056.16	1
Slide bar clip	6.51	5.49	1.19	1185.79	2
Top main plate	56.34	37.56	1.50	1500.08	1
TOTAL	1454.25	832.82			
TOTAL, kg	1.45				

The total mass of the drone is 1.454 kg and the total volume is 832.819 cm^3 . The calculated center of mass is at the coordinates of (0.7; -10.8; -16.6) and the matrix of inertia of the UAV (Equation 3.1), considering the axis represented in Figures 3.7 and 3.8:

$$I = \begin{bmatrix} 0.021 & -6.087 \times 10^{-5} & 4.868 \times 10^{-4} \\ -6.087 \times 10^{-5} & 0.023 & 2.238 \times 10^{-5} \\ 4.868 \times 10^{-4} & 2.238 \times 10^{-5} & 0.032 \end{bmatrix} \quad (3.1)$$

3.5 Basic drone control functions

The drone's fundamental control functions are designed to facilitate communication with various flight control software systems such as PX4 and MAVROS. This enables the drone to detect forest fire paths and safely navigate by avoiding potential obstacles it may encounter during its flight.

MAVROS stands for "MAVLink to ROS" and it is a popular open-source middleware that enables communication between a MAVLink-enabled autopilot system (such as PX4) and ROS. MAVROS allows users to control and monitor autonomous vehicles, such as drones, using ROS tools and libraries. It provides a convenient interface for exchanging messages, accessing vehicle telemetry data, and controlling vehicle behavior within the ROS ecosystem. MAVROS is widely used in the field of robotics and unmanned aerial systems for research, development, and implementation of various applications.[71], [72]

The way the drone control algorithm accesses the MAVROS package is through nodes, which are individual software processes that perform specific functions within a robotic system. They are like individual workers in a factory, each responsible for a particular task. Nodes (Figure 3.9) are coordinated by a main node called ROS master.[73]

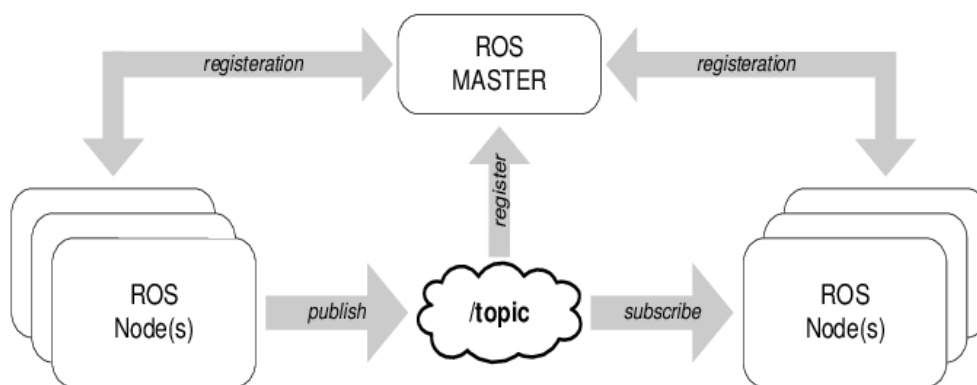


Figure 3.9 - Nodes communication model in the ROS environment [74]

Nodes communicate with each other through 5 different ways: messages, publishing and subscribing(topics), parameters, services, actions.

- **Messages:** ROS defines a set of standardized message types that nodes use to communicate. These message types define the structure and content of the data being exchanged between nodes. Nodes need to agree on the message type used for a specific topic to ensure proper communication.[75]
- **Publishing and subscribing (topics):** Topics in ROS serve as named communication channels through which nodes exchange messages. They follow an anonymous publish/subscribe model, allowing information production and consumption to be decoupled. Nodes are not aware of the specific nodes they are communicating with; instead, they subscribe to topics they are interested in, while publishers provide data by publishing messages to relevant topics. Multiple nodes can act as publishers or subscribers for a single topic. Topics are primarily designed for one-way, continuous communication.[76]
- **Parameters:** The parameter server in ROS acts as a network-accessible, shared dictionary. It allows nodes to store and retrieve parameters during runtime. While it is not optimized for high-performance operations, it serves as an efficient solution for managing static, non-binary data, particularly configuration parameters. The parameter server is designed to provide a global view of the system's configuration state, making it convenient for tools to inspect and modify the parameters as needed.[77]
- **Services:** ROS provides a Service-based communication model. Services are defined by a pair of messages: one for the request and one for the reply. A ROS node offering a service specifies a unique name, and clients can call the service by sending a request message and waiting for the corresponding reply. This interaction is often presented to programmers as if it were a remote procedure call. Services are defined using “srv” files, which are compiled into source code by a ROS client library. To enhance performance, a client can establish a persistent connection with a service. However, this approach may be less robust to changes in service providers.[78]
- **Actions (actionlib):** actions provide a way to perform long-running, asynchronous tasks that require feedback, cancellation, and goal status monitoring. Actions are useful when the outcome of a task is not immediate and may take a significant amount of time to complete. The action communication model in ROS involves three main components: the action server, the action client, and the action messages. Actions in ROS provide several benefits, such as the ability to track progress, receive intermediate feedback, preempt or cancel tasks, and handle tasks that require long-duration execution. They are commonly used for complex behaviors, motion planning, and coordination tasks in robotic systems. The action communication model in ROS complements the publish-subscribe messaging and service communication models.[79], [80]

Nodes are activated by assigning various functions within the drone control code. This algorithm is represented as follows:

```
# _____ Setup Publishers and Subscribers _____
def setup_pubsub(self):
    rospy.loginfo("----- Setting pub / sub -----")

    #Subscribers
    self.altitude_sub = rospy.Subscriber('/mavros/altitude',Altitude, self.altitude_cb)
    self.state_sub = rospy.Subscriber('/mavros/state',State,self.state_cb)
    self.extended_state_sub = rospy.Subscriber('/mavros/extended_state',ExtendedS
    self.setpoint_raw_local_sub = rospy.Subscriber('/mavros/setpoint_raw/local',Po
    self.local_position_odom_sub = rospy.Subscriber('/mavros/local_position/odom
    self.local_position_sub = rospy.Subscriber('/mavros/local_position/pose', PoseSt
    self.imu_sub = rospy.Subscriber("/mavros/imu/data", Imu, self.imu_cb)
    self.home_position_sub = rospy.Subscriber("/mavros/home_position/home", Hc

    self.guidance_target = rospy.Subscriber("/imageGuidance", PositionTarget, self.ir

    #Publishers
    self.setpoint_raw_pub = rospy.Publisher('/mavros/setpoint_raw/local',PositionT
    self.setmission_pub = rospy.Publisher('/mission/start',Bool,queue_size=10)

def setup_services(self):
    rospy.loginfo('----Waiting for services to connect----')
    try:
        rospy.wait_for_service('/mavros/param/get', self.service_timeout)
        rospy.wait_for_service('/mavros/cmd/arming',self.service_timeout)
        rospy.wait_for_service('/mavros/set_mode',self.service_timeout)
        rospy.wait_for_service('/mavros/cmd/takeoff',self.service_timeout)
        rospy.wait_for_service('/mavros/cmd/land',self.service_timeout)
        rospy.wait_for_service('mavros/param/set',self.service_timeout)
        rospy.loginfo('Services are connected and ready')
    except rospy.ROSException as e:
        rospy.logerr('Failed to initialize services')

    #get services
    self.get_param_srv = rospy.ServiceProxy('/mavros/param/get',ParamGet)

    #set services
    self.set_stream_rate_srv = rospy.ServiceProxy('/mavros/set_stream_rate', StreamRate)
    self.set_arm_srv = rospy.ServiceProxy('/mavros/cmd/arming',CommandBool)
    self.set_mode_srv = rospy.ServiceProxy('/mavros/set_mode',SetMode)
    self.set_takeoff_srv = rospy.ServiceProxy('/mavros/cmd/takeoff',CommandTOL)
    self.set_land_srv = rospy.ServiceProxy('/mavros/cmd/land',CommandTOL)
    self.set_param_srv = rospy.ServiceProxy('mavros/param/set', ParamSet)
```

```

# _____ Set Parameters Functions _____
def set_param(self, param_name, value1):
    try:
        myparam = ParamValue()
        myparam.integer = int(value1)
        self.set_param_srv(param_name, myparam)
        rospy.loginfo("---- RC Failsafe disabled ----")
        #rospy.loginfo("Param "+param_name+" set to "+str(value1))
    except rospy.ServiceException as ex:
        rospy.logwarn("Param "+param_name+" not successfully set")

def set_mavros_stream_rate(self):
    stream_rate = StreamRateRequest()
    stream_rate.request.stream_id = 3
    stream_rate.request.message_rate = 10
    stream_rate.request.on_off = 1
    try:
        self.set_stream_rate_srv(stream_rate)
    except rospy.ServiceException as exp:
        rospy.logerr('Stream rate service failed')

```

The “setup_pubsub” function is responsible for setting up the publishers and subscribers.

In this code, several subscribers are created using the “rospy.Subscriber” function. Each subscriber is associated with a specific topic and message type. For example, “self.altitude_sub” subscribes to the topic “/mavros/altitude” and expects messages of type “Altitude”. Similarly, other subscribers are created for topics such as “state”, “extended state”, “setpoint raw local”, “local position”, IMU data, “home position”, and “image guidance”.

On the other hand, two publishers are set up using the “rospy.Publisher” function. The “self.setpoint_raw_pub” publisher sends messages of type “PositionTarget” to the “/mavros/setpoint_raw/local” topic, and the “self.setmission_pub” publisher sends messages of type Bool to the “/mission/start” topic.

The “setup_services” function sets up various services that the node can use to interact with the system. Services allow the node to request specific actions or retrieve information from other nodes or components.

In this code, the “rospy.wait_for_service” function is used to wait for the availability of services with specified names. These services include parameter retrieval, arming, mode setting, takeoff, landing, and parameter setting. Once the services are available, the node logs a message indicating that they are connected and ready.

Afterward, the code creates service proxies using the “rospy.ServiceProxy” function. These proxies allow the node to call the respective services. For example, “self.get_param_srv” is a proxy for the “/mavros/param/get” service, which retrieves parameter values.

The “set_param” function sets a parameter value by using the “/mavros/param/set” service. It takes a parameter name and value as arguments, converts the value to an integer, and attempts to set the parameter. If successful, it logs a message indicating that the RC failsafe is disabled.

The “set_mavros_stream_rate” function configures the stream rate for MAVROS messages. It creates a “StreamRateRequest” object with the desired stream ID, message rate, and enables the stream. It calls the “/mavros/set_stream_rate” service to set the stream rate. If the service call fails, it logs an error message.

Also in the code, “mode functions” are written in order to control the state of a vehicle. These functions utilize various MAVROS services and publishers to control the vehicle's behavior and mode of operation.

- The arm() function attempts to arm the vehicle if it is not already armed.
- The disarm() function disarms the vehicle.
- The set_mode(mode) function changes the mode of the vehicle to the specified mode.
- The set_takeoff(takeoff_altitude_local, yaw_deg, yaw_rate) function sets the target pose for takeoff and engages the OFFBOARD mode. It publishes the target pose until the target position is reached.
- The set_position_local(target_x, target_y, target_z, yaw_deg, yaw_rate) function sets the target pose for local position control and engages the OFFBOARD mode. It publishes the target pose until the target position is reached.

Regarding the more relevant functions of this code, they are responsible for the drone's control behavior by following a fire path and avoiding obstacles. The main function initializes the drone, performs takeoff, sets position setpoints, and calls the “followGuidance” function.

The “followGuidance” function sets the target pose for the drone based on the fire path guidance and continuously publishes the setpoints. It ensures the drone is in the correct mode and armed. The code includes functions to calculate distances to walls and return True if it is too close to the wall.

Overall, this snippet of the code provides a basic framework for controlling the drone's movement, handling guidance, obstacle avoidance, and trajectory execution which is detailed in Chapter 5.

Chapter 4

Model implementation

4.1 Mathematical models of the flight simulation

The mathematical models of the project's flight simulation can be based on several physics' engines. These are software libraries or modules that simulate the physical laws and principles governing the behavior of objects in a virtual environment. They provide the necessary algorithms and calculations to realistically model and simulate various aspects of physics, such as motion, forces, collisions, and interactions between objects. Gazebo simulator supports four distinct physics engines[81]:

- Open Dynamics Engine (ODE) - for simplified robot dynamics
- Bullet - for gaming
- Dynamic Animation and Robotics Toolkit (DART) - for computer graphics and robot control
- Simbody - for biomechanics

These physics engines provide the underlying computational algorithms and calculations necessary to simulate the physical interactions and dynamics of objects in Gazebo. Each physics engine has its own set of mathematical models. Gazebo integrates with these physics engines to provide a realistic and accurate simulation environment. The simulator provides the flexibility to switch between these different physics engines based on the simulation file configuration.

As ODE is meant for simplified robot dynamics then it is the chosen physics engine for this simulation in Gazebo as well as because it is the most appropriate since it closely approximates the simulation of a multicopter for flight dynamics with the required performance and precision.[82], [83]

ODE's physical and mathematical concepts include an explanation in rigid bodies, step sizing, vectors, forces, collision and constraints.

4.1.1 Rigid bodies, equations of motion and state vector

A rigid body in simulation has various properties that can change over time and others that remain constant. The changing properties include the position vector of the body's point of reference (x, y, z) , the linear velocity of the point of reference (v_x, v_y, v_z) , the orientation represented by a quaternion or a 3x3 rotation matrix, and the angular velocity vector (w_x, w_y, w_z) . Constant properties include the body's mass, the position of the center of mass with respect to the point of reference (which must coincide in the current implementation), and the inertia matrix that describes how the body's mass is distributed around the center of mass. There are two coordinate frames: the body coordinate frame (Figure 4.1) and the global coordinate frame. The shape of a rigid body is not a dynamic property, except for its influence on the various mass properties.[84], [85]

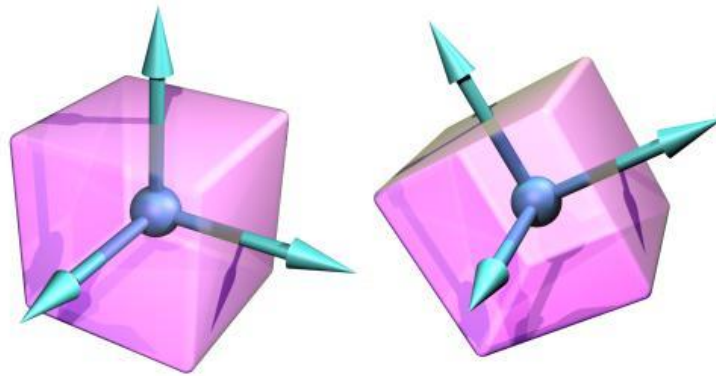


Figure 4.1 - The body coordinate frame[84]

The rigid bodies have equations of motion applied to them. These equations describe their motion state and dynamics. They can be derived from Newton's laws of motion and are typically expressed in terms of linear and angular quantities. The world-space location of the body is written in Equation 4.1:

$$\mathbf{p}(t) = \mathbf{R}(t) \mathbf{p}_0 + \mathbf{x}(t) \quad (4.1)$$

Being $\mathbf{x}(t)$ the translation of the body at time t . $\mathbf{R}(t)$ is a 3x3 matrix that characterizes the rotation around the center of mass, with each column indicating the directions of the body's axes in the world-space coordinates $(x, y$ and $z)$. Finally, \mathbf{p}_0 represents a fixed point in the object.

Besides the location, there is also the linear velocity that can be defined as the derivative of the position of the body in time t (Equation 4.2) and the angular velocity that is related to the direction of the body (Equation 4.3):

$$\mathbf{v}(t) = \dot{\mathbf{x}}(t) \quad (4.2)$$

$$\dot{\mathbf{r}}(t) = \mathbf{w}(t) \mathbf{r}(t) \quad (4.3)$$

$\mathbf{P}(t)$ is the sum of the products of the mass and velocity of each particle (Equation 4.4).

$$\mathbf{P}(t) = \sum m_i \dot{\mathbf{r}}_i(t) \quad (4.4)$$

The mass of each particle in the body is represented by m_i and the velocity of each particle is denoted as $\dot{\mathbf{r}}_i(t)$ and can be simplified by treating it as a single particle with mass M and a center of mass coordinate system. The Equation 4.5 can be expressed as follows:

$$\mathbf{P}(t) = \mathbf{M} \mathbf{v}(t) \quad (4.5)$$

$\mathbf{L}(t)$ is the total angular momentum. $L(t)$ is calculated by multiplying the inertia matrix and angular velocity (Equation 4.6):

$$\mathbf{L}(t) = \mathbf{I}(t) \mathbf{w}(t) \quad (4.6)$$

The angular velocity refers to the rate at which it rotates around a certain axis and the inertia matrix (3x3) describes how the mass of a rigid body is distributed around its center of mass.

Finally, the rigid body state vector can be defined as (Equation 4.7):

$$\mathbf{Y}(t) = \begin{pmatrix} \mathbf{x}(t) \\ \mathbf{r}(t) \\ \mathbf{P}(t) \\ \mathbf{L}(t) \end{pmatrix} \quad (4.7)$$

The derivative of the $\mathbf{Y}(t)$ vector is implemented and computed in the simulator physics. It serves the purpose of achieving accurate and real-time simulations. It allows for numerical integration, ensuring smooth motion updates. The derivatives enable the detection and response to collisions, incorporating external forces, and providing a realistic and interactive experience.[86], [87]

4.1.2 Step sizes and simulation integration

The process of simulating the rigid body system through time is known as integration. It involves updating the state of the rigid bodies at each time step. Two main considerations when choosing an integrator are accuracy and stability. Accuracy refers to how closely the simulated behavior matches real-life behavior, while stability ensures that calculation errors do not lead to non-physical behavior. ODE's current integrator is stable but lacks accuracy unless the step size is small. While ODE generally produces physically plausible results, it is not recommended for precise engineering applications until accuracy concerns are addressed in future releases.

After acknowledging the accuracy challenges within the current integrator framework, it becomes pivotal to explore numerical methods that mitigate these limitations. This subsection 4.1.2 delineates how diverse numerical techniques address these accuracy concerns, paving the way for improved simulations in complex systems like Gazebo. In a canonical initial value problem, the behavior of the system is described by equation 4.8 which is an ordinary differential equation of the form:

$$\dot{x} = f(x, t) \tag{4.8}$$

In the above equation f is a known function, x is the state of the system and \dot{x} is x time derivative. These are considered vectors of the system.

ODE uses Euler's basic numerical method, as well as Runge-Kutta of order 4 method for approximating solutions to ordinary differential equations. Euler's involves discretizing the independent variable into small intervals and iteratively updating the approximation based on the derivative at each point.[88]

It picks an initial value of integration $x_0 = x(t_0)$ and then computes $x(t_0 + h)$ by taking a step in the derivative direction. In Equation 4.9 it is considered h as a step size parameter.

$$x(t_0 + h) = x_0 + h\dot{x}(t_0) \tag{4.9}$$

While it is straightforward to implement, Euler's method is neither stable nor accurate. It produces significant errors especially for systems with complex or rapidly changing dynamics such as Gazebo.

It specifically introduces a local truncation error proportional to the square of the step size (h^2). As the step size increases, the error grows rapidly, limiting the accuracy of the approximation. To understand how this method can be improved, it is important to look closely at the error it produces. The key to understanding what is going on is the Taylor series in Equation 4.10:

$$x(\mathbf{t}_0 + \mathbf{h}) = x(\mathbf{t}_0) + \mathbf{h}\dot{x}(\mathbf{t}_0) + \frac{\mathbf{h}^2}{2!}\ddot{x}(\mathbf{t}_0) + \frac{\mathbf{h}^3}{3!}\dddot{x}(\mathbf{t}_0) + \dots + \frac{\mathbf{h}^n}{n!}\frac{\partial^n x}{\partial \mathbf{t}^n} + \dots \quad (4.10)$$

Euler's method would only be accurate if all derivatives beyond the first were equal to zero. Higher-order numerical methods, such as the Runge-Kutta methods, offer improved accuracy by reducing the truncation error through the inclusion of more terms from the Taylor series expansion.

Runge-Kutta of 2nd order (Midpoint method) improves Euler's method by evaluating the derivative at the current point, estimating the slope at the midpoint, and updating the approximation accordingly. It strikes a balance between accuracy and computational efficiency. The Equation 4.11 is as follows:

$$x(\mathbf{t}_0 + \mathbf{h}) = x(\mathbf{t}_0) + \frac{\mathbf{h}}{2} \left(f(x_0) + f(x_0 + \mathbf{h} f(x_0)) \right) \quad (4.11)$$

The function $f(x)$ represents the derivative function or rate of change of x with respect to time. This formula first evaluates a Euler step, then performs a second derivative evaluation at the midpoint of the step, using the midpoint evaluation to update x . Hence the name midpoint method.

The most popular procedure for reducing the computation errors in simulators and dynamics engines (possibly in Gazebo), is the Runge-Kutta of order 4 and has an error step of h^4 . The Equations 4.12-4.16 for computing $x(t_0 + h)$ are listed below:

$$k_1 = \mathbf{h}f(x_0, t_0) \quad (4.12)$$

$$k_2 = \mathbf{h}f\left(x_0 + \frac{k_1}{2}, t_0 + \frac{\mathbf{h}}{2}\right) \quad (4.13)$$

$$k_3 = \mathbf{h}f\left(x_0 + \frac{k_2}{2}, t_0 + \frac{\mathbf{h}}{2}\right) \quad (4.14)$$

$$k_4 = \mathbf{h}f(x_0 + k_3, t_0 + \mathbf{h}) \quad (4.15)$$

$$x(\mathbf{t}_0 + \mathbf{h}) = x_0 + \frac{1}{6}k_1 + \frac{1}{3}k_2 + \frac{1}{3}k_3 + \frac{1}{6}k_4 \quad (4.16)$$

The Butcher algorithm offers superior precision and adaptability compared to the fixed-coefficient Runge-Kutta method. Its adaptive coefficients allow for fine-tuning accuracy and stability, ideal for simulations requiring high precision. Particularly adept at handling stiff

equations with rapid solution changes, the Butcher algorithm maintains accuracy with larger step sizes, reducing computational load for complex systems.

However, its increased computational complexity limits real-time applications and resource-constrained environments. In contrast, the Runge-Kutta of order 4 method, preferred in Gazebo simulations due to its reasonable accuracy and lower computational demands, remains a practical choice despite slightly lower precision.

4.1.3 Collisions and constraints

Collision detection is the only aspect concerned with the detailed shape of the body. In the collision detection algorithm, bounding boxes are computed for each rigid body during a preprocessing step. The goal is to efficiently determine overlapping pairs of bounding boxes among the given set. If two rigid bodies' bounding boxes do not overlap, they can be disregarded. However, overlapping pairs require further consideration. The algorithm then checks for interpenetration or contact points between convex polyhedra representing the rigid bodies. Two bodies can also interact not by collision but through joints and constraints. However, there can be errors in the joints.[89]

The presence of joint error can occur when the positions and orientations of bodies connected by a joint do not meet the required constraints. Joint error can arise from user-defined positions/orientations or simulation errors causing the bodies to deviate from their required alignment (see Figure 4.2). To mitigate joint errors, a corrective force is applied to each joint during each simulation step. The magnitude of this force is determined by the Error Reduction Parameter (ERP), which ranges between 0 and 1. In the present simulation the value is set to 0.2.

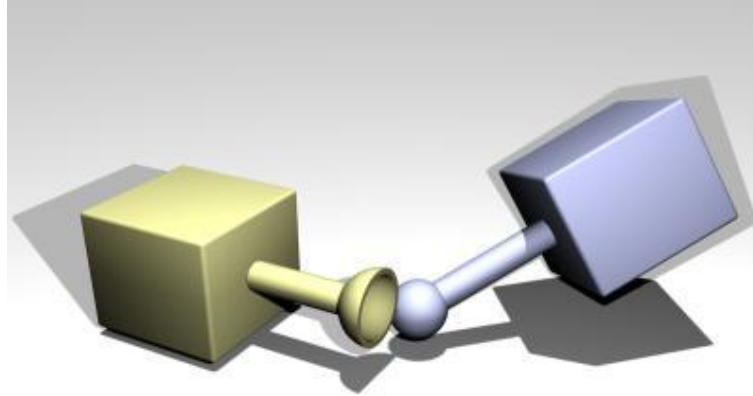


Figure 4.2 - An example of an error in ball and socket joint [84]

In physics simulation, constraints can be classified as either "hard" or "soft". Hard constraints represent conditions that are never violated, while soft constraints are designed to allow for some degree of violation.

Most constraints are hard by default, ensuring strict adherence to the specified conditions. However, unintentional errors can introduce violations, which can be corrected using the ERP.

Soft constraints intentionally allow for some violation. For example, the contact constraint that prevents object penetration can be made soft to simulate softer materials, allowing for natural penetration when objects collide.

The classification of constraints as hard or soft is controlled by two parameters. The ERP controls the reduction of error in hard constraints, while the Constraint Force Mixing (CFM) value determines the softness of constraints. By adjusting these parameters, the behavior of constraints can be customized to achieve desired simulation effects. In the current simulation, CFM value is set to 0, which makes "hard" constraints.

$$ERP = \frac{hk_p}{hk_p + k_d} \quad (4.17)$$

$$CFM = \frac{1}{hk_p + k_d} \quad (4.18)$$

Equations 4.17 and 4.18 are shown above, where h is the step size, k_p is the spring constant that characterizes the stiffness or elasticity of a spring, k_d is the damping constant which represents the resistance to motion in a system.[84]

4.1.4 Aerodynamics and propulsion

The ODE is primarily a physics engine focused on simulating rigid body dynamics, collisions, and constraints for a wide range of applications, including robotics and virtual environments. It does not inherently include aerodynamic modeling or specific equations for propulsion. Instead, it focuses on the basic physics of rigid body motion and collision detection/response.

In order to incorporate propulsion modeling into a simulation using ODE, it is needed to include plugins in the quadrotor's code to implement these equations. These plugins are shown in Section 4.4.

Regarding aerodynamics, it is simulated by applying force accumulators that are continuously applied to the drone over several time steps. During each integrator step, forces can be applied to the rigid body. These forces are accumulated in the rigid body object's "force accumulators". When the next integrator step occurs, the total sum of all applied forces is utilized to determine the body's movement. After each integrator step, the force accumulators are reset to zero.

4.2 Test lab virtual model

4.2.1 Parameters

The virtual model of the testing laboratory was adapted from a file that comes from the ROS installation. In this way, the physics of the virtual environment/space that are already included and defined in the base code are leveraged. This "world" is also defined in the launch files of the simulation "mavros_posix_sitl.launch," making it easier to adapt to this case and preventing unintentional corruption of installation files. The parameters that are defined under the physics tag in the world simulation are[90], [91]:

- **type:** ODE, bullet, simbody, and DART.

- **max step size:** when using "ODE," only the "step size" is defined because this "physical engine" is a "fixed step solver" (default is 0.001, it is 0.004 in this simulation). Smaller step sizes make the simulation more accurate.
- **real time update rate:** this is the frequency at which simulation time is advanced (default is 1000). It is 250 in this simulation because a higher number requires more computing power.
- **real time factor:** $\text{max step size} \times \text{real time update rate}$ (if < 1 , then the simulation is slower than real-time). In this simulation it is 1.
- **max contacts:** the maximum number of contacts between two entities with defined collision (walls and drone). In this simulation, the definition of this variable is omitted because the simulation is not meant to simulate interactions between the drone and other entities. Therefore, there is no need to allocate data resources for this variable.

ODE (the chosen one) has 4 main parameters: solver, constraints, friction, and contact. Only the first two are defined in the virtual environment because since it is a drone simulation, contact and friction between the multicopter and other surfaces should be avoided and then the simulation assumes default values since they are not so relevant. It was decided that it would be best not to modify any parameters since the ones already defined are suitable for real-time circumstances and simplify the simulation.

- **Solver:** there are two types of solvers, quick and world. The 1st iterates several solutions in order to reach an accurate enough solution. The 2nd gives an accurate solution if it can solve the problem. Quick is selected in this simulation.
- **Constraints:** constraint parameters are used to define relationships and limitations between rigid bodies or joints, enabling the simulation of realistic physical interactions. ERP and CFM are enabled in this part.
- **Friction:** these parameters control the magnitude of the friction forces between objects. Coefficient of friction and direction are specified.
- **Contact:** represented by stiffness and damping constants.

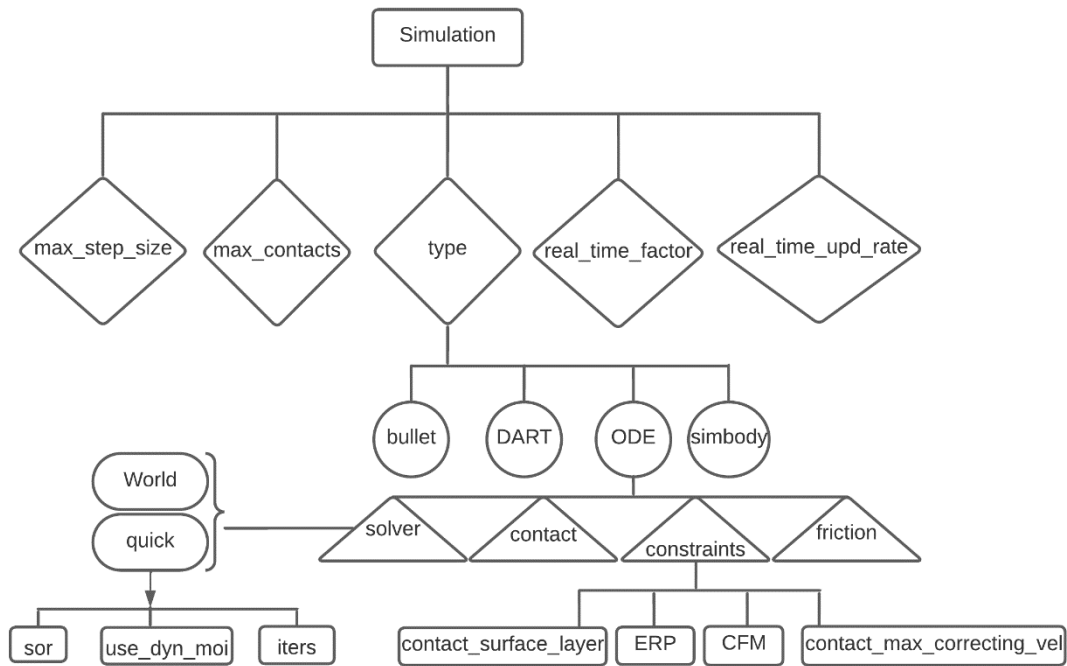


Figure 4.3 - Diagram with the ODE simulation parameters

The following code defines the simulation conditions for the virtual environment created. It has the physics of the world where the drone operates.

```

<!-- PHYSICS OF THE WORLD-->
<physics name='default_physics' default='o' type='ode'>
  <gravity>0 0 -9.8066</gravity>
  <ode>
    <solver>
      <type>quick</type>
      <iters>10</iters>
      <sor>1.3</sor>
      <use_dynamic_moi_rescaling>0</use_dynamic_moi_rescaling>
    </solver>
    <constraints>
      <cfm>0</cfm>
      <erp>0.2</erp>
      <contact_max_correcting_vel>100</contact_max_correcting_vel>
      <contact_surface_layer>0.001</contact_surface_layer>
    </constraints>
  </ode>
  <max_step_size>0.004</max_step_size>
  <real_time_factor>1</real_time_factor>
  <real_time_update_rate>250</real_time_update_rate>
  <magnetic_field>6.0e-6 2.3e-5 -4.2e-5</magnetic_field>
</physics>

```

- **Iters:** The number of iterations for the solver to run for each time step. It is set to 10 in this simulation, prioritizing quicker simulation with some accuracy.
- **Sor:** Successive Over Relaxation method is an iterative technique used to solve linear systems of equations. If it is close to 1 then the convergence is slower during each iteration. The value is set to 1.3 prioritizing stability.
- **Use_dynamic_moi_rescaling:** This parameter is related to moments of inertia. It is set to false/zero assuming that the moments of inertia of objects remain constant throughout the simulation.
- **Cfm:** By setting CFM to zero, the simulation assumes that the constraints are perfectly rigid, and any violation of the constraint conditions results in an immediate correction to enforce the constraints precisely. This leads to a very accurate and stable simulation.
- **Efm:** When the ERP is set to 0.2, it means that the position correction applied to resolve constraint violations is 20% of the total error. In other words, if there is a constraint violation between two objects in the simulation, the position correction will attempt to reduce the error by 20% during each iteration of the simulation.
- **Contact_max_correcting_vel:** it means that the maximum velocity at which contact correction is applied between objects in contact is limited to 100 meters per second.
- **Contact_surface_layer:** it means that a virtual layer with a thickness of 0.001 meters is added to the contact surface of objects. This layer helps prevent objects from penetrating each other due to numerical errors or small inaccuracies in the simulation.

4.2.2 Operation limits

The operating limits were defined based on 4 default files from the ROS installation: "big_box," "big_box2," "big_box3," and "big_box4." In Figure 4.4, one of the walls was hidden to better understand the depth/length in this perspective.

The Extended Markup Language (XML) language code related to one of the operating limits is demonstrated below. Within this XML code, the visual and collision properties are delineated like the chosen limit. This is a pivotal aspect of the simulation, as it defines how the system perceives and interacts with its surroundings.

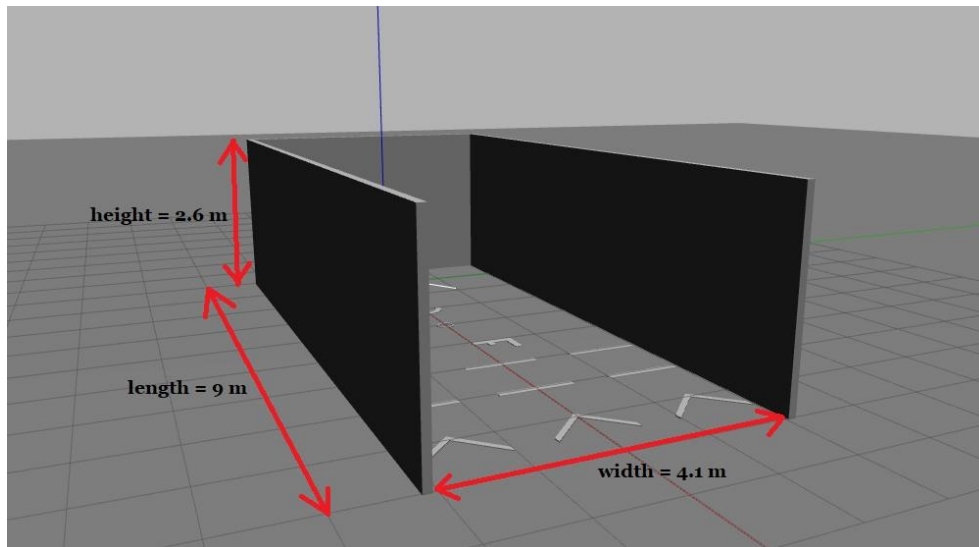


Figure 4.4 - Test lab virtual model

It's noteworthy that the values within this code, pertaining to the size of the walls, have been configured to mirror the dimensions of the real laboratory test. This ensures that the simulation faithfully replicates the conditions and constraints encountered during actual testing.

```

<?xml version="1.0" ?>
<sdf version="1.5">
  <model name="Big box">
    <pose>0 0 0 0 0 0</pose>
    <static>true</static>
    <link name="link">
      <collision name="collision">
        <geometry>
          <box>
            <size>9 0.1 2.6</size>
          </box>
        </geometry>
        <surface>
          <contact>
            <ode>
              <max_vel>0.1</max_vel>
              <min_depth>0.001</min_depth>
            </ode>
          </contact>
        </surface>
      </collision>
    </link>
  </model>
</sdf>

```

```

<visual name="visual">
  <geometry>
    <box>
      <size>9 0.1 2.6</size>
    </box>
  </geometry>
  <material>
    <script>
      <uri>
file://media/materials/scripts/gazebo.material
</uri>
      <name>Gazebo/White</name>
    </script>
  </material>
</visual>
</link>
</model>
</sdf>

```

4.2.3 Fire patterns

The code for the fire patterns was written in the same file as the simulation file "empty.world." Some figures that emit the shape of forest fires were obtained using XML code functions called "polyline," where the vertices need to be defined in coordinates/points. Other figures were drawn using the CATIA V5 program, and then their visual "mesh" file is called within a function.

In Figure 4.5, The "F" shape and line patterns are described with a polyline and the semi-circles and circular crowns with a STL file generated in CATIA V5.

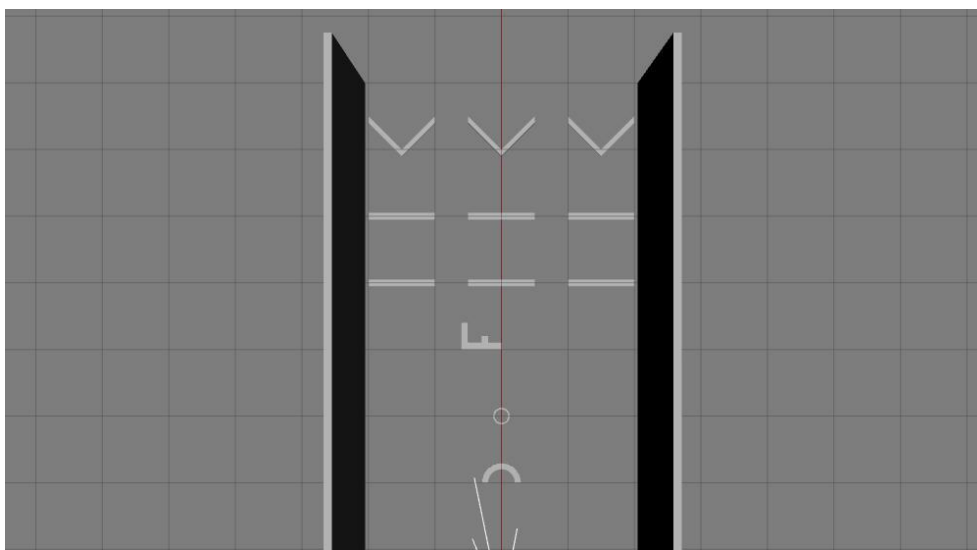


Figure 4.5 - Fire patterns in Gazebo simulation

```

<visual name="letter_f">
  <geometry>
    <polyline>
      <point>0 0</point>
      <point>0 .6</point>
      <point>.4 .6</point>
      <point>.4 .5</point>
      <point>.1 .5</point>
      <point>.1 .3</point>
      <point>.2 .3</point>
      <point>.2 .2</point>
      <point>.1 .2</point>
      <point>.1 0</point>

      <height>0.01</height>
    </polyline>
  </geometry>
  <material>
    <script>
      <uri>file://media/materials/scripts/gazebo.material
</uri><name>Gazebo/White</name>
    </script>
  </material>
</visual>
</link>
<static>1</static>
</model>

<!-- circular crown -->
<include>
  <uri>model://circular_crown</uri>
</include>
<!-- semi circle -->
<include>
  <uri>model://semi_circle</uri>
</include>

```

4.3 Physical models of the UAV and camera

4.3.1 Raspberry Pi V2

The chosen camera brand was "Raspberry Pi" because their cameras have several advantages over others. They are more affordable, making them suitable for this project with budget constraints.

Raspberry Pi cameras are compact and lightweight, allowing for flexible integration into the chosen multirotor. They are user-friendly with clear documentation and strong community support.

The Raspberry Pi V2 Camera (Figure 4.6) [92] offers several advantages over the other camera modules of the same brand. It is cheaper than the Module 3, Module 3 Wide, and High Quality Camera, making it a more affordable option. Additionally, it provides improved image quality compared to the Module v1. Since the camera and drone setup was purchased in 2022, the Module 3, which was launched earlier this year, was not chosen. The Camera Module v2, with its affordability and enhanced quality, was selected as the preferred option.



Figure 4.6 - Raspberry Pi V2 camera[93]

Table 4.1 provides a comprehensive comparison of camera module specifications, including size, weight, still resolution, video modes, sensor type, sensor resolution, sensor image area, focus capabilities, focal length, as well as horizontal and vertical field of view (FOV). This detailed comparison simplifies the process of selecting the most suitable camera for this specific project, enabling to make a well-informed choice.

Table 4.1 - Raspberry Pi Cameras hardware specifications[94]

	Module v1	Module v2	Module v3	V3 Wide	HQ camera	GS camera
Net price	25\$	25\$	25\$	35\$	50\$	50\$
size	25x24x9 mm	25x24x9 mm	25x24x11.5 mm	25x24x12.4 mm	38x38x18.4 mm	38x38x19.8 mm
weight	3g	3g	4g	4g	30.4g	34g

Still resolution	5 MP	8 MP	11.9 MP	11.9 MP	12.3 MP	1.58 MP
Video modes	1080p30, 720p60 and 640 × 480p60/90	1080p47, 1640 × 1232p41 and 640 × 480p206	2304 × 1296p56, 2304 × 1296p30 HDR, 1536 × 864p120	2304 × 1296p56, 2304 × 1296p30 HDR, 1536 × 864p120	2028 × 1080p50, 2028 × 1520p40 and 1332 × 990p120	1456 × 1088p60
sensor	OmniVision OV5647	Sony IMX219	Sony IMX708	Sony IMX708	Sony IMX477	Sony IMX296
Sensor resolution	2592 × 1944 pixels	3280 × 2464 pixels	4608 × 2592 pixels	4608 × 2592 pixels	4056 × 3040 pixels	1456 × 1088 pixels
Sensor image area	3.76 × 2.74 mm	3.68 × 2.76 mm (4.6 mm diagonal)	6.45 × 3.63mm (7.4mm diagonal)	6.45 × 3.63mm (7.4mm diagonal)	6.287mm × 4.712 mm (7.9mm diagonal)	6.3mm diagonal
focus	fixed	adjustable	motorized	motorized	adjustable	adjustable
Focal length	3.60 mm	3.04 mm	4.74 mm	2.75 mm	Depends on lens	Depends on lens
Horizontal FOV	53.50°	62.2°	66°	102°	Depends on lens	Depends on lens
Vertical FOV	41.41°	48.8°	41°	67°	Depends on lens	Depends on lens

4.3.2 Holybro X500 V2

The chosen brand for the UAV was Holybro. Their collaboration with the Pixhawk project provides advanced flight control capabilities and compatibility with various peripherals. They offer good value for money with competitive pricing. Additionally, Holybro has a strong community support network that provides helpful resources and knowledge sharing.

The X500 series consists of versatile drones designed for professional applications. These drones are used in aerial photography, videography, mapping, and surveying tasks.

It is a drone that has a modular design for customization and stable flight performance. It offers extended flight time and supports autonomous flight modes for efficient mission planning and

execution. The Holybro X500 V2 (Figure 4.7) is made with full carbon fiber twill, with carbon fiber tube arms supported by the newly designed fiber reinforced nylon connectors with convenient notches on both motor and body sides, providing a much easier & more straightforward installation. Its flight time is 18 minutes hover with no additional payload and with 5000mAh battery, the maximum payload is 1kg. The wheelbase is 500mm, frame body 144x144mm and landing gear height 215mm. Motor mount pattern is 16x16mm.[95]



Figure 4.7 - X500 V2 Holybro drone [96]

4.4 Multirotor virtual model

For the virtual model of the multirotor, the base code is the same as the "typhoon_h480.sdf" that comes with the ROS installation, and the parameters have been modified to fit the projects' multirotor. The base code used already utilizes the same source software-in-the-loop (SITL) as the PX4 flight controller. Gazebo allows for defining the complete model, with various libraries already existing, including the propulsion libraries. The modifications to the virtual model file can be grouped into 4 topics: inertia, collision, visual, and plugins (propulsion and MAVlink).

In the inertia part, the data is presented in Section 3.4, and applied in the drone's code. For collision, it was decided to define a parallelepiped that surrounds the drone instead of a perfect contour around the rotors, landing gear, antenna, and central block of the drone. This way, the program calculates fewer collision points between surfaces, reducing the chance of program crashes.

In the visual part, the file is set to look for a mesh file named "x500v2.stl," which is the file extracted from the CATIA V5 program, as demonstrated in section 3.3. The rotors visually default to the "typhoon_h480.sdf" file, but the power and rotation parameters are altered in the code. In Figure 4.8 the X axis (red) has the same direction as in CATIA V5, while the Z (blue) and Y (green) axes have opposite directions in Gazebo.

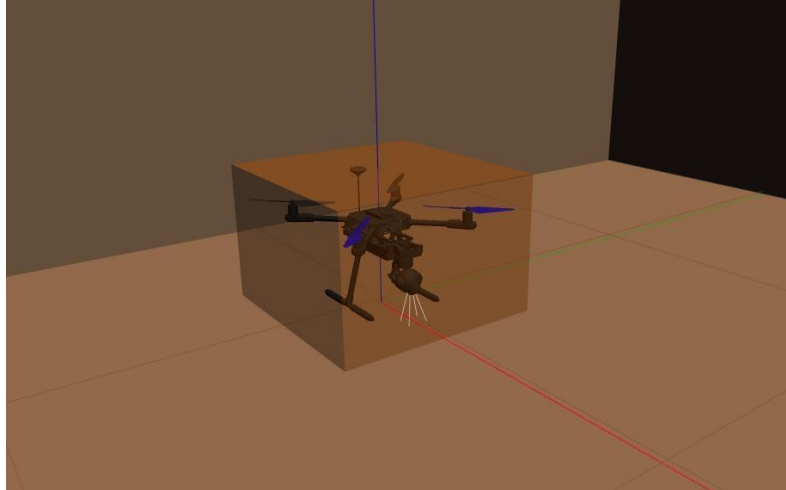


Figure 4.8 - X500 V2 Holybro drone inside Gazebo simulation

The code below defines the mass and inertia matrix of the drone. Subsequently, in the collision part, a box with dimensions of 0.50x0.50x0.4 meters is defined, which has its origin at the center of mass and encompasses the entire drone. Finally, the code that retrieves the graphical representation of the Holybro X500 V2 drone, previously designed in CATIA V5, is also demonstrated.

```
<mass>1.454</mass>
<inertia>
  <ixx>0.021</ixx>
  <ixy>6.087e-05</ixy>
  <ixz>-4.868e-04</ixz>
  <iyy>0.023</iyy>
  <iyz>-2.238e-05</iyz>
  <izz>0.032</izz>
</inertia>
```

```

<collision name='base_link_collision'>
  <pose>0 0 -0.066797 0 0 0</pose>
  <geometry>
    <box>
      <size>0.50 0.50 0.4</size>
    </box>
  </geometry>

<visual name='base_link_visual'>
  <pose>0 0 -0.032 0 0 -1.57079633</pose>
  <geometry>
    <mesh>
      <scale>0.001 0.001 0.001</scale>
      <uri>model://typhoon_h480/meshes/x500V2.stl</uri>
    </mesh>
  </geometry>

```

In a Simulation Description format (SDF) file for a virtual drone simulation, plugins provide additional functionality. Propulsion plugins simulate the behavior of the drone's propulsion system, including thrust and control inputs. MAVLink plugins facilitate communication between the virtual drone and ground control software using the MAVLink protocol. They exchange telemetry data and commands. These plugins enhance simulation realism and integration with external systems.

Some propulsion data used in the code was obtained from the Holybro Brushless 2216-880KV - CW (520049) motor. Complementary data was calculated using the following formulas:

$$K_m = \rho \frac{\pi}{2} h_m (r_0^4 - r_i^4) \quad (4.19)$$

$$K = \frac{V}{\pi K_v} \quad (4.20)$$

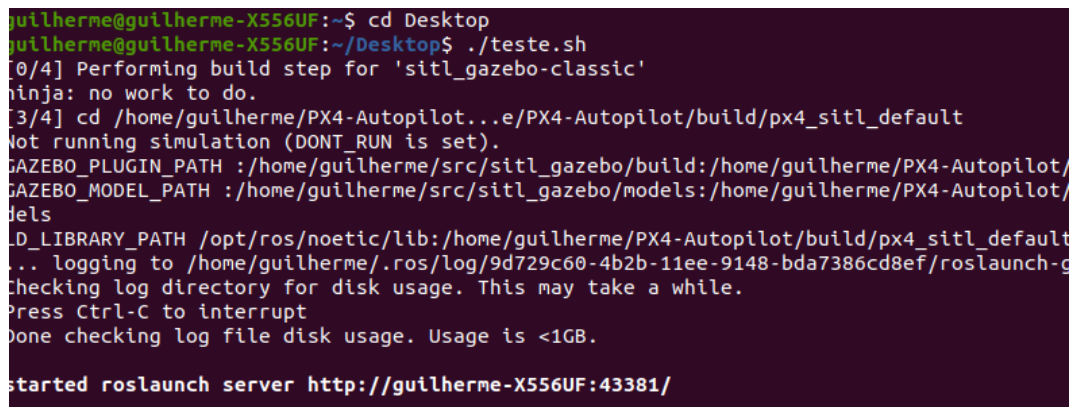
After meticulous calculations in Equations 4.19 and 4.20 the values of K_m and K (moment constant and motor constant) are $1.656 \times 10^{-4} \text{ kg m}^2$ and 0.00579 N m . The ensuing code unveils the intricacies of the rotor plugin employed for the front right motor, bearing in mind that its values remain consonant with those of its motor counterparts., the code undergoes a transformation as the parameters within are altered. . The rotor drag coefficient, rolling moment coefficient and the time constants remain the same as the default values in the code. The maximum rotation velocity is 10464 RPM [97].

```
<plugin name='front_right_motor_model' filename='libgazebo_motor_model.so'>
  <robotNamespace/>
  <jointName>rotor_o_joint</jointName>
  <linkName>rotor_o</linkName>
  <turningDirection>ccw</turningDirection>
  <timeConstantUp>0.0125</timeConstantUp>
  <timeConstantDown>0.025</timeConstantDown>
  <maxRotVelocity>10464</maxRotVelocity>
  <motorConstant>0.00579</motorConstant>
  <momentConstant>1.656e-04</momentConstant>
  <commandSubTopic>/gazebo/command/motor_speed</commandSubTopic>
  <motorNumber>0</motorNumber>
  <rotorDragCoefficient>0.000175</rotorDragCoefficient>
  <rollingMomentCoefficient>1e-06</rollingMomentCoefficient>
  <motorSpeedPubTopic>/motor_speed/0</motorSpeedPubTopic>
  <rotorVelocitySlowdownSim>10</rotorVelocitySlowdownSim>
</plugin>
```


Chapter 5

Results

In this chapter, the conclusive results of the simulation are presented. To access and view these results, a set of practical steps are required. Firstly, two software programs are initiated, namely Gazebo and QGC, in this specific order. The commencement process is carried out within the Ubuntu terminal using a bash script that contains the paths to activate the necessary packages. This is accomplished with the command “./teste.sh”, shown in Figure 5.1 and also the command “./QGroundControl.AppImage” show in Figure 5.2.



```
guilherme@guilherme-X556UF:~$ cd Desktop
guilherme@guilherme-X556UF:~/Desktop$ ./teste.sh
[0/4] Performing build step for 'sitl_gazebo-classic'
inja: no work to do.
[3/4] cd /home/guilherme/PX4-Autopilot...e/PX4-Autopilot/build/px4_sitl_default
Not running simulation (DONT_RUN is set).
GAZEBO_PLUGIN_PATH :/home/guilherme/src/sitl_gazebo/build:/home/guilherme/PX4-Autopilot/
GAZEBO_MODEL_PATH :/home/guilherme/src/sitl_gazebo/models:/home/guilherme/PX4-Autopilot/
dels
D_LIBRARY_PATH /opt/ros/noetic/lib:/home/guilherme/PX4-Autopilot/build/px4_sitl_default
.. logging to /home/guilherme/.ros/log/9d729c60-4b2b-11ee-9148-bda7386cd8ef/roslaunch-g
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://guilherme-X556UF:43381/
```

Figure 5.1 - Terminal in Ubuntu 20.04 initializing the script

These pathways lead to critical components of a ROS simulation: ROS Simulation Packages are pre-configured modules that build the virtual environment. The "catkin_ws" Directory serves as a flexible workspace for creating and adjusting simulation elements. The "mavros_posix_sitl.launch" Launch File initiates "mavros" and sets up the SITL environment, where a virtual aircraft is controlled and observed with precision.


```

guilherme@guilherme-X556UF: ~/Desktop$ ./QGrounControl.AppImage
Settings location "/home/guilherme/.config/QGroundControl.org/QGroundControl.ini" Is writable?: true
Filter rules "*Log.debug=false\nGStreamerAPILog.debug=true\nqt.qml.connections=false"
System reported locale: QLocale(English, Latin, United States) ; Name "en_US" ; Preferred (used in maps)
MAVLinkLogManagerLog: MAVLink logs directory: "/home/guilherme/Documents/QGroundControl/Logs"
Map Cache in: "/home/guilherme/.cache/QGCMapCache300" / "qgcMapCache.db"
qml: QGCCorePlugin(0x55a106c0d880) []
setCurrentPlanViewSeqNum
setCurrentPlanViewSeqNum
_recalcFlightPathSegments homePositionValid false
_recalcFlightPathSegments homePositionValid false
Adding target QHostAddress("127.0.0.1") 18570
"v4.2.0"
_recalcFlightPathSegments homePositionValid false
setCurrentPlanViewSeqNum
setCurrentPlanViewSeqNum
_recalcFlightPathSegments homePositionValid false
_recalcFlightPathSegments homePositionValid true
_recalcFlightPathSegments homePositionValid true

```

Figure 5.2 - Terminal in Ubuntu 20.04 initializing QGC

To establish a successful connection with QGC and maintain consistent settings, it is needed to modify two files: "6011_typhoon_h480" and "6011_typhoon_h480.post." Originally designed for a hexacopter, was quickly adapted to work with a quadcopter.

The script "6011_typhoon_h480" configures a quadrotor within the PX4 autopilot system. Originally designed for a hexarotor, it sets default flight control parameters, specifies the quadrotor type, and defines the mixer. The "6011_typhoon_h480.post" script configures the mixer for motor control and sets up MAVLink communication with specific port configurations. It also defines MAVLink streams for data transmission. Overall, these scripts configure communication and control for the quadrotor in the PX4 autopilot system, ensuring effective communication with external systems and control responses. After configuration, the drone goes through an arming process, preparing for flight and awaiting commands like in Figure 5.3.

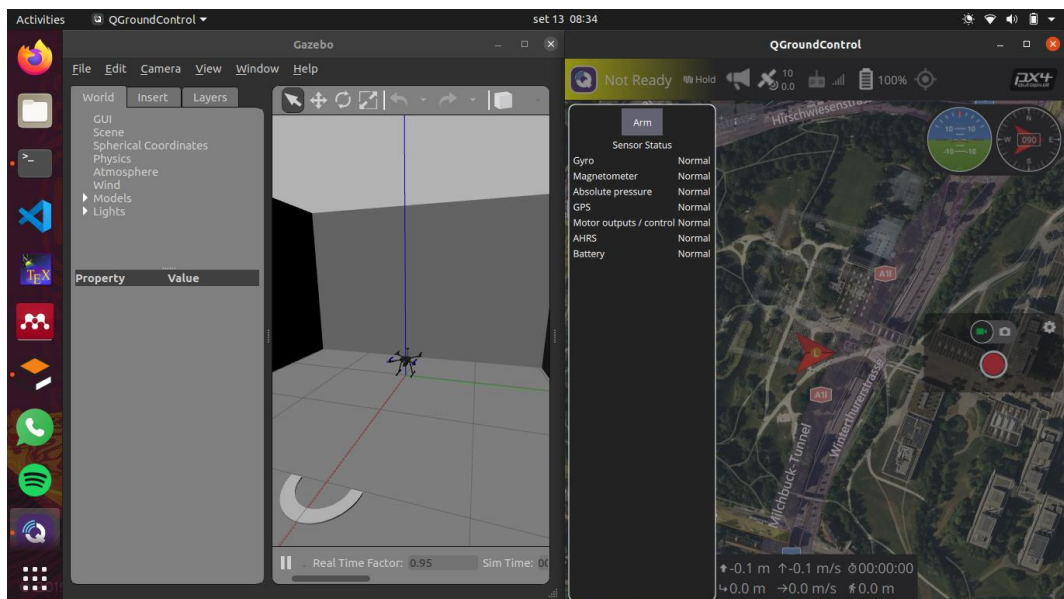


Figure 5.3 - Gazebo and QGC side to side with the drone up and ready to fly

5.1 UAV control algorithm and fire tracking

This specific section investigates the use of various drone control functions for continuous or discontinuous tracking fire line trajectories and various fire front shapes. The functions "set_position_local," and "followGuidance" belong to the Python script titled "droneControl4.py." This Python script plays a vital role in controlling the drone's behavior during simulated flights within the PX4 autopilot system.

5.1.1 Continuous trajectories

To achieve continuous tracking of fire line trajectories, the "droneControl4.py" script is employed, leveraging its "set_position_local" and "followGuidance" functions. The primary role of the "set_position_local" function is to guide the drone to precise local coordinates (X, Y, Z), while also considering the Yaw angle.

In a real-world context, this implies that the drone receives global coordinates, potentially marking the origin of a fire outbreak. These coordinates can be disseminated by local authorities through various means, such as satellite imagery analysis, civilian reports, or forest ranger observations.

Meanwhile, the "followGuidance" function empowers the drone to be controlled interactively via a keyboard interface, facilitated by the utilization of the "pynput" library[98], [99]. This versatile function enables the operator to make real-time adjustments, including alterations in altitude, forward and backward movements, as well as lateral shifts to the left or right.

In a simulation scenario, is considered an initial movement where the drone is directed to the coordinates (1, 1, 2). This specific location serves as the starting point for tracking a fire line configured in the shape of the letter "F," as illustrated in Figure 3.3 from Chapter 3.

To gain deeper insights into the execution process, the underlying code is exhibited, which showcases the implementation of these functionalities within the main function of the drone control algorithm:

```

if __name__ == '__main__':
    rospy.init_node('drone_control_node', anonymous = True)

    #Takeoff Parameters
    takeoff_altitude = 2

    try:
        control = DroneControl()

        #Arm
        control.arm()

        #Takeoff
        control.set_takeoff(takeoff_altitude,270)
        control.set_position_local(1,1,2,90)
        rospy.loginfo('Before followGuidance')
        control.followGuidance()
        rospy.loginfo('Ending followGuidance commands')

        #RTL and Land
        control.set_RTL()

    except rospy.ROSInterruptException as exception:
        pass

```

Next, Figures 5.4 and 5.5 depict the drone's takeoff position and the starting point of the fire trail. Upon detecting the initiation point, the keyboard operator transmits the coordinates the drone will follow along the path to the terminal. These specific coordinates are then visualized in Figures 5.6, 5.7, and 5.8, providing a comprehensive view of the drone's flight path and its relation to the fire trail's starting location. In Figures 5.9 and 5.10 upper views of the drone are demonstrated.

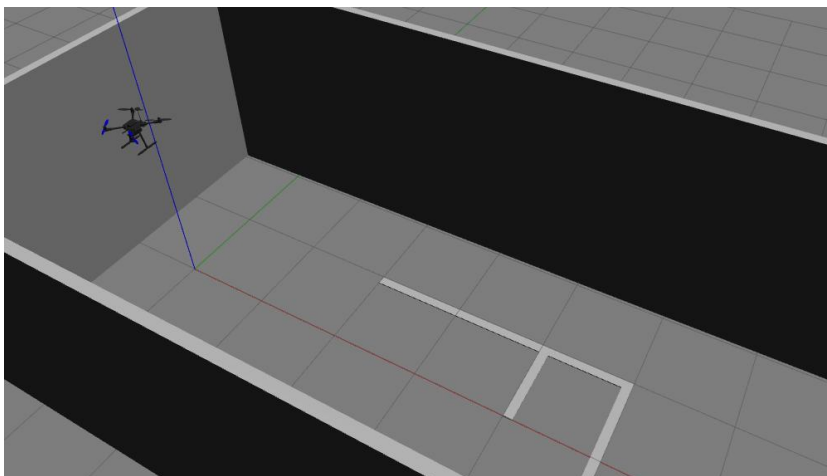


Figure 5.4 – Takeoff position coordinate (0,0,2)

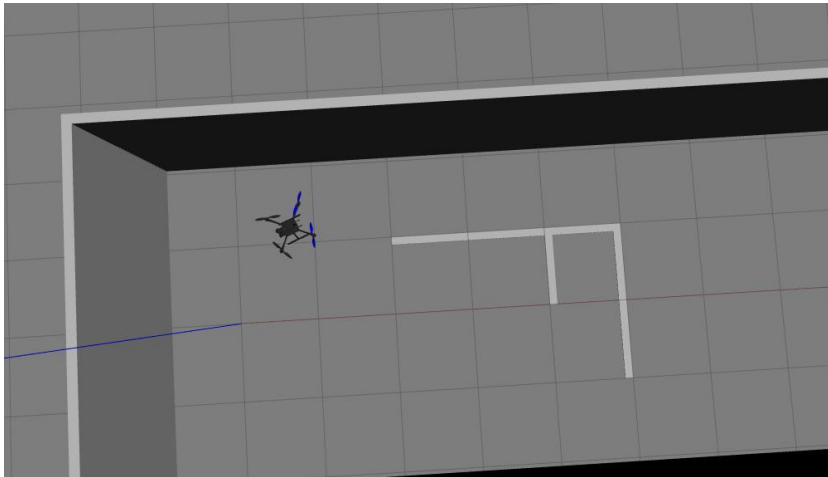


Figure 5.5 – Starting point of the fire trail coordinate (1,1,2)

```
[INFO] [1695138224.611779, 147.224000]: header: [INFO] [1695138314.835296, 228.672000]: header:
  seq: 934                                     seq: 2563
  stamp:                                       stamp:
    secs: 147                                 secs: 228
    nsecs: 196000000                          nsecs: 644000000
  frame_id: ''                                frame_id: ''
coordinate_frame: 1                           coordinate_frame: 1
type_mask: 0                                  type_mask: 0
position:                                     position:
  x: 1.0                                       x: 4.0
  y: 1.0                                       y: 1.0
  z: 2.0                                       z: 2.0
```

Figure 5.6 – Starting point of the F shaped trail

Figure 5.7 – Coordinate (4,1,2)

```
[INFO] [1695307128.712198, 183.556000]: header:
  seq: 2928
  stamp:
    secs: 183
    nsecs: 504000000
  frame_id: ''
coordinate_frame: 1
type_mask: 0
position:
  x: 5.0
  y: -1.0
  z: 2.0
```

Figure 5.8 – Coordinate (5,-1,2)

During this fire surveillance journey, the drone operator only knows the starting point of the fire trail. In this case, as it involves an "F" shaped pattern consisting of two lines in the same direction, the drone must follow the closest line to itself, which consequently is also the shorter one and begins at the coordinates (4,1,2). After traversing this line, it returns to the main line and proceeds to follow the second line, which is further away, until reaching the coordinates (5,-1,2) marking the end of the trail.

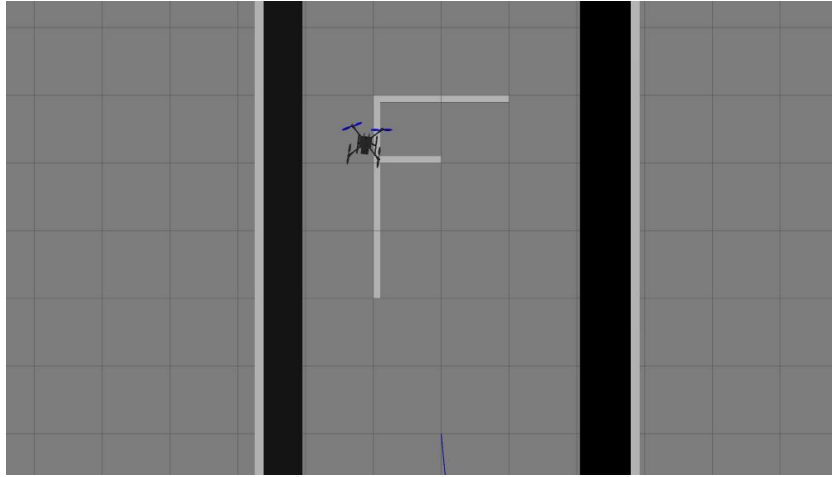


Figure 5.9 – Upper view of the coordinate (4,1,2)

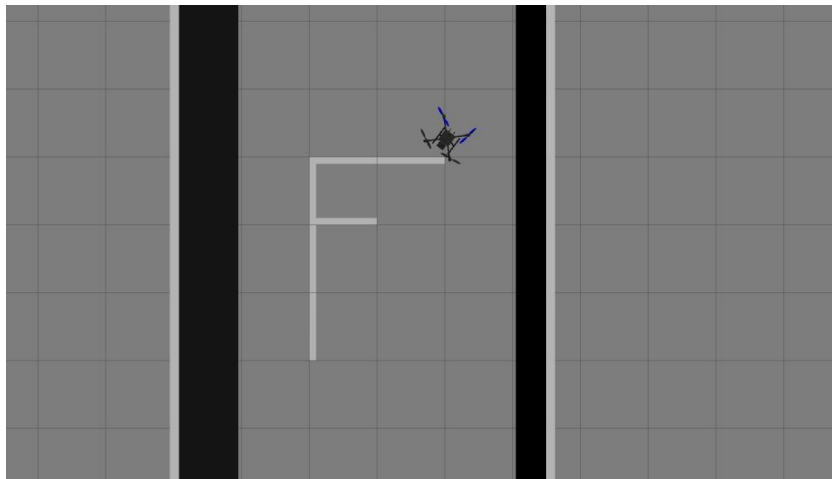


Figure 5.10 – Upper view of the coordinate (5,-1,2)

After inspecting the entire length of the fire front, the multirotor reverts its path to assess the fire's progression, promptly communicating any observed discrepancies to the base (in a real-world scenario).

Next, another example is examined resembling the "V" shape illustrated in Figure 3.5 from Section 3.2. This scenario represents another fire front characterized by straight-line segments. This example showcases the multirotor's versatility in monitoring different fire scenarios and reporting its findings for further analysis and decision-making.

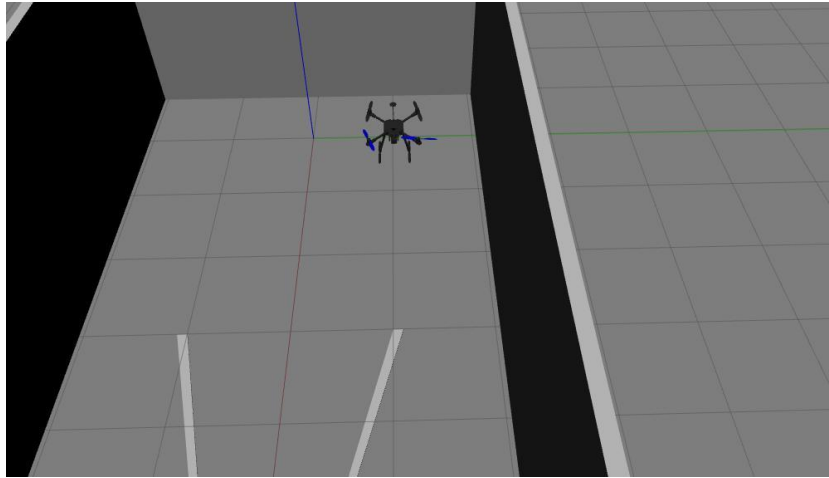


Figure 5.11 – Perspective view of the drone in coordinate (3,1, 2.3)

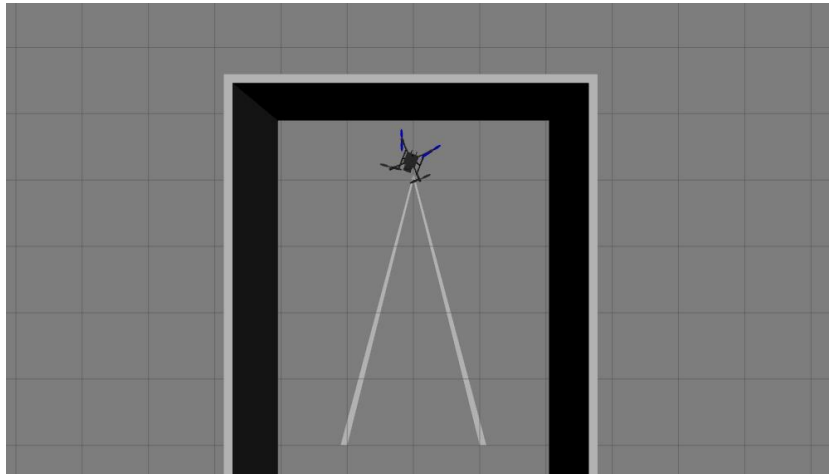


Figure 5.12 – Upper view of the drone in coordinate (7,0,2.3)

Once again, in Figures 5.11 and 5.12, the drone is showcased positioned directly above the fire's starting point and at the furthest vertex of the "V" shape. Subsequently, it descends to a height of $z=1.0$ m to gain a closer and more detailed view of the flames' size and thickness.

Following this descent, in Figures 5.13, 5.14, and 5.15, we can observe the Ubuntu terminal's interface. This occurs after executing the "followGuidance" function and utilizing the keyboard to guide the drone. This user interface interaction allows for real-time control over the drone's movements, offering a hands-on approach to navigate and inspect the fire front from various angles and altitudes. This capability provides valuable insights into the dynamic behavior of the fire and supports informed decision-making during firefighting operations.

```
[INFO] [1695139998.324100, 228.304000]: header:  
seq: 1437  
stamp:  
  secs: 228  
  nsecs: 280000000  
frame_id: ''  
coordinate_frame: 1  
type_mask: 0  
position:  
  x: 3.0  
  y: 1.0  
  z: 2.3
```

Figure 5.13 – Coordinate (3,1, 2.3)

```
[INFO] [1695140097.833022, 319.004000]: header:  
seq: 3251  
stamp:  
  secs: 318  
  nsecs: 980000000  
frame_id: ''  
coordinate_frame: 1  
type_mask: 0  
position:  
  x: 7.0  
  y: 0.0  
  z: 2.3
```

Figure 5.14 – Coordinate (7,0,2.3)

```
[INFO] [1695307190.973046, 238.256000]: header:  
seq: 4022  
stamp:  
  secs: 238  
  nsecs: 204000000  
frame_id: ''  
coordinate_frame: 1  
type_mask: 0  
position:  
  x: 3.0  
  y: -1.0  
  z: 1.0
```

Figure 5.15 – Coordinate (3,-1,1)

As evident in this simulation, velocity and acceleration data are deliberately excluded from the data display through the use of ignore flags that were added to the “set_position_local” method. This exclusion serves two primary purposes. Firstly, it prevents computational overload within the simulation since numerous iterations occur every second, typically ranging from 10 to 20 Hz. Secondly, it is worth noting that velocity and acceleration cannot reach high levels due to the confined nature of the simulation space, enclosed by four walls. As a result, these parameters can be safely ignored.

This selective exclusion of velocity and acceleration data is a practical approach to optimize the simulation's computational efficiency while maintaining the accuracy of other critical data. It ensures that the simulation runs smoothly without being bogged down by excessive calculations, making it a practical and efficient tool for analyzing and simulating scenarios in limited spatial environments.

5.1.2 Discontinuous trajectories

For discontinuous flight paths, the process of takeoff and initiating fire front tracking remains the same. However, in this scenario, the objective is for the drone to have a new takeoff executed,

utilizing a QGC feature. The QGC is used to ensure the drone's takeoff since sometimes, upon restarting the drone control algorithm, the simulator assumes the drone is already in "takeoff" mode when, in fact, it hasn't taken off yet (this aspect is detailed in section 5.2) .It is crucial to adjust the takeoff altitude in the mission settings to ensure that the drone does not exceed the hangar's maximum ceiling of 2.6 m.

Subsequently, the drone control algorithms are relaunched in the Ubuntu terminal, and the drone is poised to navigate the new fire pattern. This operational approach ensures adaptability and safety when executing discontinuous flight paths.

Figure 5.16 illustrates the drone at the end of the first fire trail which resembles the one from Figure 3.4 in Section 3.2. Subsequently, the drone increases its altitude to check for any additional fire sources. Since it does not detect any more forest fire sources and does not receive reports of sightings in other areas, it returns to the base.

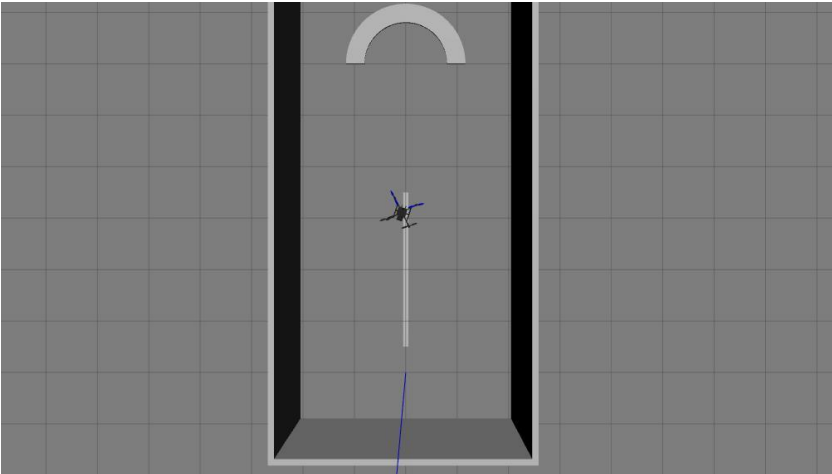


Figure 5.16 – Upper view of the drone in coordinate (3,0, 2.5)

```
[INFO] [1695146595.754750, 280.496000]: header:  
  seq: 2531  
  stamp:  
    secs: 280  
    nsecs: 444000000  
  frame_id: ''  
coordinate_frame: 1  
type_mask: 0  
position:  
  x: 3.0  
  y: 0.0  
  z: 2.5
```

Figure 5.17 – Coordinate (3,0, 2.5)

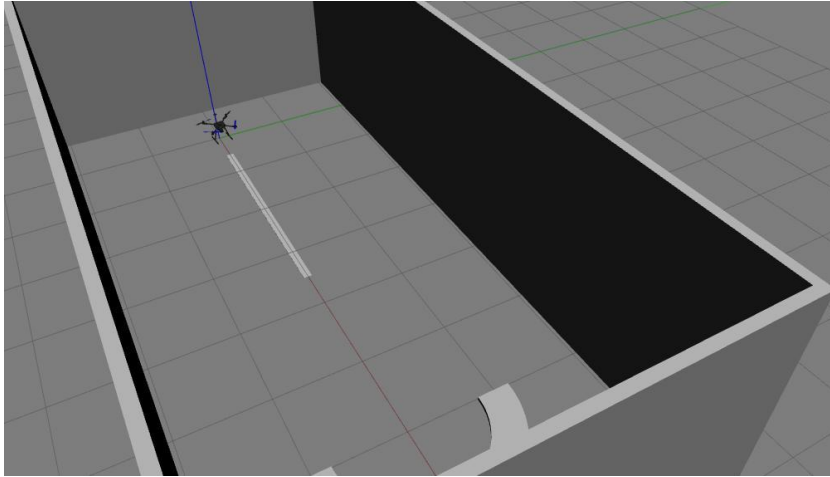


Figure 5.18 – Perspective view of the drone after returning from first fire path

Upon returning to the base, as shown in Figure 5.18, the drone receives information about a new fire outbreak further ahead, the QGC's takeoff functionality is activated, initiating the drone's journey toward the new fire source, which assumes a semi-circular shape. As it reaches the semi-circular pattern, as shown in Figure 5.19 and 5.20, the drone meticulously navigates along the trail through keyboard commands issued by the operator. By observing the drone's camera image, the operator determines the direction of movement required. Upon reaching the end of the fire trail, the drone can relay to the firefighting teams the direction of fire movement. Finally, the drone returns to the base at the conclusion of the operation. The drone demonstrates its capability to successfully navigate around this configuration.

This operation showcases the adaptability and maneuverability of the drone as it responds to changing fire patterns. The ability to effectively navigate a semi-circular fire pattern highlights the utility of such autonomous systems in responding to dynamic and evolving scenarios, ensuring efficient monitoring and response to wildfires.

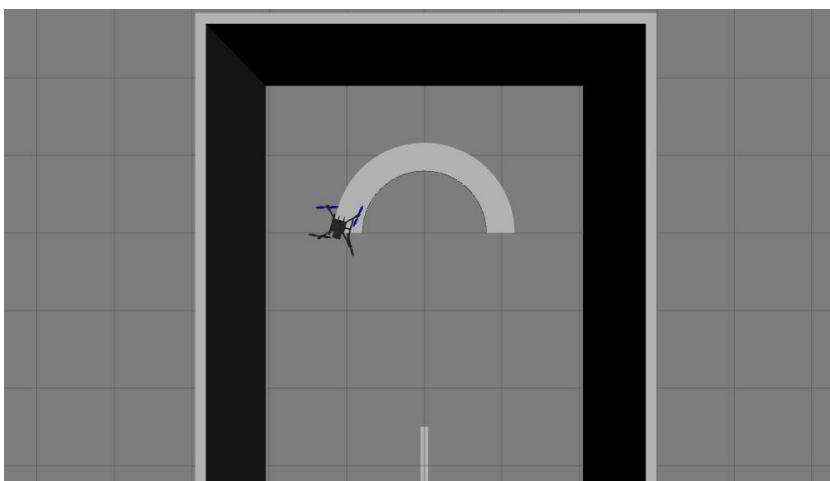


Figure 5.19 – Upper view of the drone in coordinate (6,1,1)

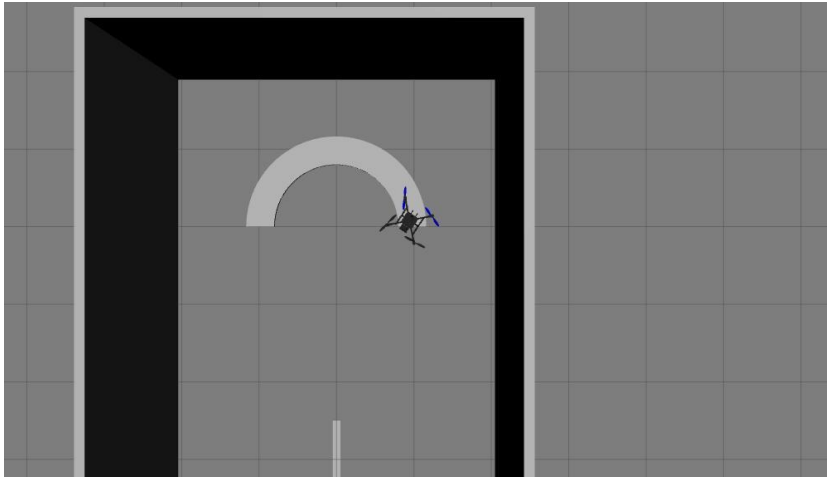


Figure 5.20 - Upper view of the drone in coordinate (6,-1,1)

```
[INFO] [1695147283.343857, 464.812000]: header:
  seq: 7356
  stamp:
    secs: 464
    nsecs: 784000000
  frame_id: ''
coordinate_frame: 1
type_mask: 0
position:
  x: 6.0
  y: 1.0
  z: 1.0
```

Figure 5.21 – Coordinate (6,1,1)

```
[INFO] [1695147323.007929, 500.664000]: header:
  seq: 8073
  stamp:
    secs: 500
    nsecs: 636000000
  frame_id: ''
coordinate_frame: 1
type_mask: 0
position:
  x: 6.0
  y: -1.0
  z: 1.0
```

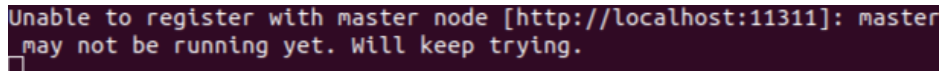
Figure 5.22 – Coordinate (6,-1,1)

The next section demonstrates the errors and bugs that might be associated with this simulation and how they can be avoided. All the examples presented so far need to be tested in a real context to verify their functionality.

5.2 Simulation errors and bugs

In addition, errors and bugs may occur along these flight paths. One error encountered in this simulation indicates that Gazebo is having trouble registering with the ROS master node. This can happen occasionally because the ROS master node is not running. It is essential that it is

running because it manages different parts of the programs' system. This can be solved by typing "roscore" in the Ubuntu terminal. Another reason can be because the simulation is relaunched again too quickly. In Figure 5.23 the problem in the terminal is shown.



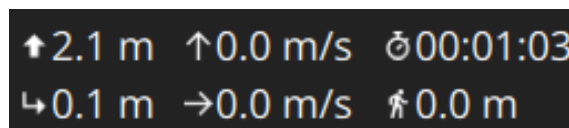
```
Unable to register with master node [http://localhost:11311]: master
may not be running yet. Will keep trying.
```

Figure 5.23 – ROS master bug in Gazebo simulation

Another important point to highlight is the existence of a positioning discrepancy between Gazebo and what is displayed in QGC. Recognizing these factors is crucial for accurately interpreting and utilizing simulation data in testing and decision-making processes, considering their potential impact on simulation fidelity. In Figure 5.24, the QGC data is depicted while the drone is positioned for takeoff at coordinates (0,0,2) within Gazebo. Hence, a discernible discrepancy comes to light: the QGC data shows a positional error of 0.1 m along the z-axis and a 0.1-meter deviation in the X and Y axes. Interestingly, there is no corresponding discrepancy evident in the velocity data.

This variance can arise for various reasons such as differences in simulation data between Gazebo and QGC. Firstly, Gazebo operates in discrete time steps, synchronizing with communication to QGC, potentially introducing discrepancies due to the discrete nature of updates and the complexity of calculations involved. Secondly, data transmission and processing delays between Gazebo and QGC can contribute to differences in reported data. Lastly, when using floating-point calculations, minor rounding errors can accumulate over time, leading to slight position discrepancies.

Although these errors may seem relatively small, in a real-world context, they would translate to a significant positioning error spanning several meters. This underscores the importance of cross-referencing data with other altimetry and positioning devices. Within the confines of a hangar, a 0.1-meter discrepancy can be the difference between avoiding collision with a wall and a potential mishap. Also, this value of 0.1 m represents the UAV's position change after pressing a specific key on the keyboard. This means that a single misstep could potentially lead to a collision for the drone. Therefore, it is crucial to always exercise caution and allow for a margin of safety, even when seemingly minor discrepancies arise.



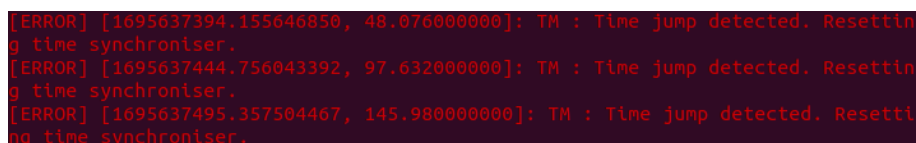
```
↑2.1 m  ↑0.0 m/s  ⌚00:01:03
↳0.1 m  →0.0 m/s  🌀0.0 m
```

Figure 5.24 – QgroundControl altimetry and position data

Concerning another bug is that it is crucial to understand that the “droneControl4.py” script keeps track of the most recent coordinate it receives from imageOutputScript.py. Once the drone successfully identifies all fires, the operator sends the multirotor to come back to its initial takeoff point using manual keyboard commands. This approach is designed to prevent a situation where, upon the next launch, the drone immediately heads towards the last known fire location which is not the same location for the next mission.

Also, after running the drone control program once, it is necessary to initiate a "takeoff" operation in QGC. This is because the “droneControl4.py” script retains the drone's flight mode as "armed" since its last launch. As a result, the script still perceives the drone as being in an "armed" state when in QGC it is not. To transition the drone from this mode to an actual flight-ready state, it is essential to perform a "takeoff" operation using QGC. This action is vital to ensure that the drone is ready for flight operations, regardless of its previous state.

The last error that occurs in this ROS-based simulation is related to the time synchronizer. It is essential that the timing of messages sent and received by different components of the system remain synchronized. This error is caused by hardware or system clock issues in the computer, delays in message processing or transmission or even high system resource usage that affects message processing speed. This makes the time synchronizer reset several times as shown in Figure 5.25. However, this error does not affect the flight path environment.



```
[ERROR] [1695637394.155646850, 48.076000000]: TM : Time jump detected. Resetting time synchroniser.  
[ERROR] [1695637444.756043392, 97.632000000]: TM : Time jump detected. Resetting time synchroniser.  
[ERROR] [1695637495.357504467, 145.980000000]: TM : Time jump detected. Resetting time synchroniser.
```

Figure 5.25 – Time synchronizer error in Ubuntu terminal

5.3 Wall detection algorithm

Also, in the course of this research, a wall detection algorithm was developed to enhance the spatial awareness and navigation capabilities of the UAV within a 3D environment. This algorithm serves as a crucial component of the overall simulation framework, contributing significantly to the drone's ability to navigate safely and efficiently.

The primary function of the wall detection algorithm is to compute the distances between the UAV's current position and the surrounding walls in the 3D environment. Specifically, it determines the distances to four distinct walls, taking into account their coordinates within the

simulated space. These distances are then utilized to assess the drone's proximity to its immediate surroundings.

The algorithm's logic is straightforward: it checks whether the minimum distance to any of the nearby walls is less than or equal to a predefined threshold, set at 0.5 units. If the minimum distance falls within this threshold, it indicates that the UAV is in close proximity to a wall.

One critical aspect of the wall detection algorithm's functionality is its real-time integration with the primary drone control algorithm. This integration ensures that the drone's control system is continually informed of its proximity to walls and can make instantaneous adjustments to its flight path as needed. Essentially, the wall detection algorithm operates continuously in the background, providing essential input for real-time decision-making.

The algorithm's effectiveness is visually confirmed through the terminal output, where distance measurements and proximity alerts are displayed. For instance, when the drone takes off, the terminal reports that it is approximately 1 m away from the nearest wall, typically the one parallel to the Y-axis and closest to the origin as illustrated in Figure 5.26.

```
INFO] [1695914173.154315, 66.944000]: False
INFO] [1695914173.269185, 67.044000]: 0.931019889935851
INFO] [1695914173.270703, 67.044000]: False
INFO] [1695914173.372685, 67.148000]: 0.9331174165010452
INFO] [1695914173.374927, 67.148000]: False
INFO] [1695914173.477771, 67.252000]: 0.9344354493543505
INFO] [1695914173.480453, 67.256000]: False
INFO] [1695914173.583057, 67.360000]: 0.9361586112529039
INFO] [1695914173.585543, 67.360000]: False
INFO] [1695914173.688182, 67.464000]: 0.9376161748543381
INFO] [1695914173.690631, 67.468000]: False
INFO] [1695914173.792688, 67.568000]: 0.9406992372125387
INFO] [1695914173.794695, 67.568000]: False
INFO] [1695914173.897993, 67.672000]: 0.943449832778424
INFO] [1695914173.901016, 67.676000]: False
INFO] [1695914174.005828, 67.776000]: 0.9455450408160686
INFO] [1695914174.014829, 67.780000]: False
INFO] [1695914174.125536, 67.788000]: 0.9455450408160686
INFO] [1695914174.136262, 67.792000]: False
INFO] [1695914174.255348, 67.896000]: 0.9483526715077459
INFO] [1695914174.256843, 67.900000]: False
INFO] [1695914174.358470, 68.000000]: 0.95120393501129
INFO] [1695914174.362362, 68.004000]: False
INFO] [1695914174.467943, 68.104000]: 0.9531264289282262
INFO] [1695914174.473640, 68.108000]: False
INFO] [1695914174.580343, 68.220000]: 0.9542988499626517
INFO] [1695914174.584810, 68.224000]: False
INFO] [1695914174.691232, 68.328000]: 0.957824252359569
INFO] [1695914174.695983, 68.332000]: False
INFO] [1695914174.799985, 68.436000]: 0.961889717541635
INFO] [1695914174.802048, 68.436000]: False
INFO] [1695914174.904346, 68.540000]: 0.9653911234810948
INFO] [1695914174.907097, 68.544000]: False
INFO] [1695914175.009753, 68.644000]: 0.968444251269102
INFO] [1695914175.012506, 68.648000]: False
```

Figure 5.26 – Terminal with data from takeoff position

As the drone maneuvers to positions with different coordinates, the terminal progressively reports distances to the nearest wall. When the drone approaches a wall to within half a meter, the terminal output switches to "True," indicating the drone's proximity to the wall.

Immediately following the confirmation of "True" in the terminal, the multirotor will transition to a designated safety position located at the center of the virtual test laboratory. Subsequently, it will be poised for further guidance to a new set of coordinates, which can be provided by the operator.

This operational procedure is a pivotal aspect of the multirotor functionality, as it ensures a secure and controlled starting point within the laboratory environment. The utilization of a safety position minimizes the potential for collisions, accidents, or unexpected interactions with the surroundings during the initial stages of operation.

Chapter 6

Conclusion

6.1 Overview

In this dissertation, the primary objective was to develop drone control capabilities for wildfire detection in a simulation environment. Throughout the research process, several noteworthy challenges were encountered. Given the initial objectives, it is worth noting that the project was able to achieve the control of the drone through local position coordinates and keyboard commands. While the full scope of simulating autonomy with artificial intelligence remains a future challenge, these accomplishments signify a step forward in enhancing the drone's practicality and control capabilities.

It's worth noting that the wall detection algorithm has successfully integrated with both the "imageOutputScript.py" (the keyboard command interface) and the "droneControl.py" (the primary drone control algorithm). These components operate in a cohesive manner, demonstrating effective collaboration.

One significant challenge was the limited scope of existing scientific literature pertaining to UAV market studies. It became apparent that there was a dearth of comprehensive research collaboration among researchers in this field. This limitation constrained the availability of valuable resources for this study.

Another noteworthy challenge I encountered was the need to acquire proficiency in Python and specifically for ROS through various online courses and tutorials. This endeavor presented a learning curve that inevitably affected the pace of development.

A critical constraint in this project was the unavailability of essential sensors, necessitating the creation of a custom wall detection algorithm from the ground up. This development added a layer of complexity to the project and had implications for its overall performance.

These challenges collectively underscore the complexities and limitations faced during this project. Addressing them will undoubtedly pave the way for further advancements in the field of drone-based wildfire detection.

6.2 Future work

In future work, a top priority will be to address the compatibility issue between the wall detection algorithm and the “imageOutputScript.py” and “droneControl.py” components. This resolution will enhance the overall functionality and usability of the simulation.

The project will progress by conducting validation tests within a controlled environment, such as a hangar. This step is crucial to verify the accuracy and reliability of the simulation.

Soon, the project will expand beyond the hangar setting to real-world applications. This transition will involve adapting the simulation to handle real-world variables and challenges such as making it applicable in disaster monitoring scenarios.

As part of future endeavors, careful consideration will be given to testing the simulation in the laboratory of the Department of Aerospace Sciences. This testing will be essential as the Gazebo simulator lacks realistic aerodynamic and propulsive properties, and adjustments will be needed to address these limitations.

References

- [1] “What Is Climate Change?” <https://www.nrdc.org/stories/what-climate-change> (accessed Aug. 26, 2023).
- [2] G. Ceballos, P. R. Ehrlich, A. D. Barnosky, A. García, R. M. Pringle, and T. M. Palmer, “Accelerated modern human-induced species losses: Entering the sixth mass extinction,” *Sci Adv*, vol. 1, no. 5, Jun. 2015, doi: 10.1126/SCIADV.1400253.
- [3] “Commercial drone market size worldwide 2027 | Statista.” <https://www.statista.com/statistics/878018/global-commercial-drone-market-size/> (accessed Jul. 14, 2023).
- [4] “Global: drone market revenue by country 2022 | Statista.” <https://www.statista.com/forecasts/1302524/revenue-of-the-drone-market-worldwide> (accessed Jul. 14, 2023).
- [5] “15 Biggest Drone Companies In the World.” <https://finance.yahoo.com/news/15-biggest-drone-companies-world-175046410.html> (accessed Jul. 14, 2023).
- [6] B. Custers, “Drones Here, There and Everywhere Introduction and Overview,” 2016, pp. 3–20. doi: 10.1007/978-94-6265-132-6_1.
- [7] “A Brief History of Drones | Imperial War Museums.” <https://www.iwm.org.uk/history/a-brief-history-of-drones> (accessed Jul. 14, 2023).
- [8] “Kettering Aerial Torpedo ‘Bug’ > National Museum of the United States Air Force™ > Display.” <https://www.nationalmuseum.af.mil/Visit/Museum-Exhibits/Fact-Sheets/Display/Article/198095/kettering-aerial-torpedo-bug/> (accessed Jul. 14, 2023).
- [9] “Hewitt-Sperry Automatic Airplane 1918 - Hewitt-Sperry Automatic Airplane - Wikipedia.” https://en.wikipedia.org/wiki/Hewitt-Sperry_Automatic_Airplane#/media/File:Hewitt-Sperry_Automatic_Airplane_1918.jpg (accessed Jul. 14, 2023).
- [10] “de Havilland DH82B Queen Bee – de Havilland Aircraft Museum.” <https://www.dehavillandmuseum.co.uk/aircraft/de-havilland-dh82b-queen-bee/> (accessed Jul. 14, 2023).
- [11] “Radioplane OQ-2A > National Museum of the United States Air Force™ > Display.” <https://www.nationalmuseum.af.mil/Visit/Museum-Exhibits/Fact-Sheets/Display/Article/196292/radioplane-oq-2a/> (accessed Jul. 14, 2023).
- [12] “Radioplane/Northrop MQM-57 Falconer > National Museum of the United States Air Force™ > Display.”

- <https://www.nationalmuseum.af.mil/Visit/Museum-Exhibits/Fact-Sheets/Display/Article/195784/radioplanenorthrop-mqm-57-falconer/> (accessed Jul. 14, 2023).
- [13] “San Diego Air & Space Museum - Historical Balboa Park, San Diego.” <https://sandiegoairandspace.org/collection/item/teledyne-ryan-firebee-photograph-collection> (accessed Jul. 15, 2023).
- [14] K. Nonami, “Research and development of drone and roadmap to evolution,” *Journal of Robotics and Mechatronics*, vol. 30, no. 3. Fuji Technology Press, pp. 322–336, Jun. 01, 2018. doi: 10.20965/jrm.2018.p0322.
- [15] “Parrot launches ANAFI USA” <https://www.parrot.com/en/newsroom/anafi-usa>
- [16] “UAV Drones Market Size, Trends and Global Forecast To 2032.” <https://www.thebusinessresearchcompany.com/report/uav-drones-global-market-report> (accessed Jul. 14, 2023).
- [17] “Unmanned Aerial Vehicle Market Size | UAV Industry Share, Report 2030.” <https://www.fortunebusinessinsights.com/industry-reports/unmanned-aerial-vehicle-uav-market-101603> (accessed Jul. 14, 2023).
- [18] “UAV Market - Unmanned Aerial Vehicles - Size, Analysis, Growth.” <https://www.mordorintelligence.com/industry-reports/uav-market> (accessed Jul. 14, 2023).
- [19] Katrina HERRICK, “Development of the Unmanned Aerial Vehicle Market: Forecasts and Trends”, doi: 10.1016/S1290-0958(00)80035-0.
- [20] A. R. Hall and C. J. Coyne, “The political economy of drones,” *Defence and Peace Economics*, vol. 25, no. 5, pp. 445–460, 2014, doi: 10.1080/10242694.2013.833369.
- [21] “Photogrammetry - an overview | ScienceDirect Topics.” <https://www.sciencedirect.com/topics/agricultural-and-biological-sciences/photogrammetry> (accessed Jul. 15, 2023).
- [22] J. Szulwic, P. Ziółkowski, and P. Ziolkowski, *GEODESY MEASUREMENT TECHNIQUES AS AN ENRICHMENT OF ARCHAEOLOGICAL RESEARCH WORKFLOW*. 2016. [Online]. Available: <https://www.researchgate.net/publication/309016515>
- [23] G. Pajares, “Overview and current status of remote sensing applications based on unmanned aerial vehicles (UAVs),” *Photogramm Eng Remote Sensing*, vol. 81, no. 4, pp. 281–329, 2015, doi: 10.14358/PERS.81.4.281.
- [24] Y. Unpaprom, N. Dussadeeb, and R. Ramaraj, “Modern Agriculture Drones.” [Online]. Available: <https://grabcad.com/library/>

- [25] J. Zhang, J. Hu, J. Lian, Z. Fan, X. Ouyang, and W. Ye, "Seeing the forest from drones: Testing the potential of lightweight drones as a tool for long-term forest monitoring," *Biol Conserv*, vol. 198, pp. 60–69, Jun. 2016, doi: 10.1016/j.biocon.2016.03.027.
- [26] E. Frachtenberg, "Practical Drone Delivery," *Computer (Long Beach Calif)*, vol. 52, no. 12, pp. 53–57, Dec. 2019, doi: 10.1109/MC.2019.2942290.
- [27] M. Ayamga, S. Akaba, and A. A. Nyaaba, "Multifaceted applicability of drones: A review," *Technological Forecasting and Social Change*, vol. 167. Elsevier Inc., Jun. 01, 2021. doi: 10.1016/j.techfore.2021.120677.
- [28] "Zipline's drones are delivering medical supplies and PPE in North Carolina - The Verge." <https://www.theverge.com/2020/5/27/21270351/zipline-drones-novant-health-medical-center-hospital-supplies-ppe> (accessed Jul. 14, 2023).
- [29] S. Asadzadeh, W. J. de Oliveira, and C. R. de Souza Filho, "UAV-based remote sensing for the petroleum industry and environmental monitoring: State-of-the-art and perspectives," *Journal of Petroleum Science and Engineering*, vol. 208. Elsevier B.V., Jan. 01, 2022. doi: 10.1016/j.petrol.2021.109633.
- [30] "Drones Are Ready to Take on Bigger Roles in Natural Disasters." <https://www.theearthandi.org/post/drones-are-ready-to-take-on-bigger-roles-in-natural-disasters> (accessed Jul. 15, 2023).
- [31] L. Merino, F. Caballero, J. R. Martínez-De-Dios, I. Maza, and A. Ollero, "An unmanned aircraft system for automatic forest fire monitoring and measurement," *Journal of Intelligent and Robotic Systems: Theory and Applications*, vol. 65, no. 1–4, pp. 533–548, Jan. 2012, doi: 10.1007/s10846-011-9560-x.
- [32] C. Yuan, Y. Zhang, and Z. Liu, "A survey on technologies for automatic forest fire monitoring, detection, and fighting using unmanned aerial vehicles and remote sensing techniques," *Canadian Journal of Forest Research*, vol. 45, no. 7, pp. 783–792, Apr. 2015, doi: 10.1139/cjfr-2014-0347.
- [33] R. Bailon-Ruiz and S. Lacroix, "Wildfire remote sensing with UAVs: A review from the autonomy point of view," in *2020 International Conference on Unmanned Aircraft Systems, ICUAS 2020*, Institute of Electrical and Electronics Engineers Inc., Sep. 2020, pp. 412–420. doi: 10.1109/ICUAS48674.2020.9213986.
- [34] "Advantages And Disadvantages Of Fixed Wing Versus Rotary Wing Aircraft - AviationOutlook." <https://aviationoutlook.com/advantage-and-disadvantages-of-fixed-wing-versus-rotor-wing-engineering-essay/#advantages-of-rotary-wing-aircraft> (accessed Aug. 29, 2023).

- [35] P. Radoglou-Grammatikis, P. Sarigiannidis, T. Lagkas, and I. Moscholios, “A compilation of UAV applications for precision agriculture,” *Computer Networks*, vol. 172, May 2020, doi: 10.1016/j.comnet.2020.107148.
- [36] S. A. H. Mohsan, Q. ul A. Zahra, M. A. Khan, M. H. Alsharif, I. A. Elhaty, and A. Jahid, “Role of Drone Technology Helping in Alleviating the COVID-19 Pandemic,” *Micromachines*, vol. 13, no. 10. MDPI, Oct. 01, 2022. doi: 10.3390/mi13101593.
- [37] B. Vergouw, H. Nagel, G. Bondt, and B. Custers, “Drone Technology: Types, Payloads, Applications, Frequency Spectrum Issues and Future Developments,” 2016, pp. 21–45. doi: 10.1007/978-94-6265-132-6_2.
- [38] “Mavic Mini - DJI.” <https://www.dji.com/pt/mavic-mini> (accessed Jul. 14, 2023).
- [39] “MultiWii.” <http://www.multiwii.com/> (accessed Jul. 14, 2023).
- [40] “MultiWii - MultiWii.” <http://www.multiwii.com/wiki/index.php?title=MultiWii> (accessed Jul. 14, 2023).
- [41] “Google Code Archive - Long-term storage for Google Code Project Hosting.” <https://code.google.com/archive/p/multiwii/> (accessed Jul. 14, 2023).
- [42] E. Ebeid, M. Skriver, K. H. Terkildsen, K. Jensen, and U. P. Schultz, “A survey of Open-Source UAV flight controllers and flight simulators,” *Microprocess Microsyst*, vol. 61, pp. 11–20, Sep. 2018, doi: 10.1016/j.micpro.2018.05.002.
- [43] “Cleanflight .” <https://cleanflight.com/> (accessed Jul. 14, 2023).
- [44] “Betaflight - Pushing the Limits of UAV Performance | Betaflight.” <https://betaflight.com/> (accessed Jul. 14, 2023).
- [45] “GitHub - betaflight/betaflight: Open Source Flight Controller Firmware.” <https://github.com/betaflight/betaflight> (accessed Jul. 14, 2023).
- [46] “iNAV for BetaFlight users · iNavFlight/iNav Wiki · GitHub.” <https://github.com/iNavFlight/iNav/wiki/iNAV-for-BetaFlight-users> (accessed Jul. 14, 2023).
- [47] “About LibrePilot - LibrePilot Documentation - LibrePilot.” <https://librepilot.atlassian.net/wiki/spaces/LPDOC/pages/4128772/About+LibrePilot> (accessed Jul. 14, 2023).
- [48] “LibrePilot – Open – Collaborative – Free.” <http://www.librepilot.org/site/index.html> (accessed Jul. 14, 2023).
- [49] “ArduPilot Documentation – ArduPilot documentation.” <https://ardupilot.org/ardupilot/> (accessed Jul. 14, 2023).
- [50] “History of ArduPilot – Dev documentation.” <https://ardupilot.org/dev/docs/common-history-of-ardupilot.html> (accessed Jul. 14, 2023).

- [51] “The Dronecode Foundation - We are setting the standards in the drone industry with open-source - Join the Community!” <https://www.dronecode.org/> (accessed Jul. 14, 2023).
- [52] “Discussion Forum for PX4, Pixhawk, QGroundControl, MAVSDK, MAVLink - Forums for the open-source drone development community.” <https://discuss.px4.io/> (accessed Jul. 14, 2023).
- [53] “PX4 User Guide.” <https://docs.px4.io/main/en/> (accessed Jul. 14, 2023).
- [54] “Introduction · MAVLink Developer Guide.” <https://mavlink.io/en/> (accessed Jul. 14, 2023).
- [55] “GitHub - microsoft/AirSim: Open source simulator for autonomous vehicles built on Unreal Engine / Unity, from Microsoft AI & Research.” <https://github.com/microsoft/airsim> (accessed Jul. 14, 2023).
- [56] “Home - AirSim.” <https://microsoft.github.io/AirSim/> (accessed Jul. 14, 2023).
- [57] S. Shah, D. Dey, C. Lovett, and A. Kapoor, “AirSim: High-Fidelity Visual and Physical Simulation for Autonomous Vehicles,” *Springer Proceedings in Advanced Robotics*, vol. 5, pp. 621–635, 2018, doi: 10.1007/978-3-319-67361-5_40.
- [58] “What is MORSE? — The MORSE Simulator Documentation.” https://www.openrobots.org/morse/doc/latest/what_is_morse.html (accessed Jul. 14, 2023).
- [59] “jMAVSim with SITL | PX4 User Guide.” <https://docs.px4.io/main/en/simulation/jmavsim.html> (accessed Jul. 14, 2023).
- [60] “What is Gazebo Simulation - The Construct.” <https://www.theconstructsim.com/ros-5-mins-028-gazebo-simulation/> (accessed Jul. 14, 2023).
- [61] “Gazebo.” <https://gazebo.org/home> (accessed Jul. 14, 2023).
- [62] “Gazebo Simulation plugin for Ardupilot.” <https://www.linkedin.com/pulse/gazebo-simulation-plugin-ardupilot-maxime-lafleur> (accessed Jul. 14, 2023).
- [63] “ROS: Why ROS?” <https://www.ros.org/blog/why-ros/> (accessed Jul. 14, 2023).
- [64] “Gazebo : Tutorial : Beginner: Overview.” https://classic.gazebo.org/tutorials?cat=guided_b&tut=guided_b1 (accessed Jul. 14, 2023).
- [65] “Documentation for Visual Studio Code.” <https://code.visualstudio.com/docs> (accessed Jul. 14, 2023).

- [66] “Overview · QGroundControl User Guide.” <https://docs.qgroundcontrol.com/master/en/index.html> (accessed Jul. 14, 2023).
- [67] “Open Source Autopilot for Drones - PX4 Autopilot.” <https://px4.io/> (accessed Jul. 16, 2023).
- [68] “Wildfires have erupted across the globe, scorching places that have never burned before | CNN.” <https://edition.cnn.com/2021/07/22/world/wildfires-siberia-us-canada-climate-intl/index.html> (accessed Jul. 14, 2023).
- [69] “Drones and Sensors Could Spot Fires Before They Go Wild - IEEE Spectrum.” <https://spectrum.ieee.org/drones-sensors-wildfire-detection> (accessed Jul. 14, 2023).
- [70] “2020 Northern Thailand forest fires snapshot. | WWF.” <https://www.natureza-portugal.org/?362337/2020-Northern-Thailand-forest-fires-snapshot> (accessed Jul. 14, 2023).
- [71] “ROS with MAVROS Installation Guide | PX4 User Guide.” https://docs.px4.io/main/en/ros/mavros_installation.html (accessed Jul. 14, 2023).
- [72] “mavros - ROS Wiki.” <http://wiki.ros.org/mavros> (accessed Jul. 14, 2023).
- [73] “Nodes - ROS Wiki.” <http://wiki.ros.org/Nodes> (accessed Jul. 14, 2023).
- [74] M. S. H. Achmad, G. Priyandoko, R. Roali, and M. R. Daud, “Tele-Operated Mobile Robot for 3D Visual Inspection Utilizing Distributed Operating System Platform,” *International Journal of Vehicle Structures and Systems*, vol. 9, no. 3, Sep. 2017, doi: 10.4273/ijvss.9.3.12.
- [75] “msg - ROS Wiki.” <http://wiki.ros.org/msg> (accessed Jul. 14, 2023).
- [76] “Topics - ROS Wiki.” <http://wiki.ros.org/Topics> (accessed Jul. 14, 2023).
- [77] “Parameter Server - ROS Wiki.” <http://wiki.ros.org/Parameter%20Server> (accessed Jul. 14, 2023).
- [78] “Services - ROS Wiki.” <http://wiki.ros.org/Services> (accessed Jul. 14, 2023).
- [79] “actionlib - ROS Wiki.” <http://wiki.ros.org/actionlib> (accessed Jul. 14, 2023).
- [80] “Understanding actions — ROS 2 Documentation: Foxy documentation.” <https://docs.ros.org/en/foxy/Tutorials/Beginner-CLI-Tools/Understanding-ROS2-Actions/Understanding-ROS2-Actions.html> (accessed Jul. 14, 2023).
- [81] “Gazebo : Blog: Gazebo supports four physics engines.” https://classic.gazebosim.org/blog/four_physics (accessed Jul. 14, 2023).
- [82] “Open Dynamics Engine.” <https://ode.org/> (accessed Jul. 14, 2023).
- [83] “Open Dynamics Engine.” <https://ode.org/ode-latest-userguide.html> (accessed Jul. 14, 2023).

- [84] “Manual - ODE.” <http://ode.org/wiki/index.php/Manual#Concepts> (accessed Jul. 14, 2023).
- [85] “Manual - ODE.” <http://ode.org/wiki/index.php/Manual> (accessed Jul. 14, 2023).
- [86] “Physically Based Modeling.” <http://www.cs.cmu.edu/~baraff/sigcourse/index.html> (accessed Jul. 14, 2023).
- [87] D. Baraff, “An Introduction to Physically Based Modeling: Rigid Body Simulation I-Unconstrained Rigid Body Dynamics Rigid Body Simulation.” Accessed: Jul. 16, 2023. [Online]. Available: <http://www.cs.cmu.edu/~baraff/sigcourse/notesd1.pdf>
- [88] A. Witkin and D. Baraff, “Physically Based Modeling: Principles and Practice Differential Equation Basics”, Accessed: Jul. 14, 2023. [Online]. Available: <http://www.cs.cmu.edu/~baraff/sigcourse/notesb.pdf>
- [89] D. Baraff, “An Introduction to Physically Based Modeling: Rigid Body Simulation II-Nonpenetration Constraints”, Accessed: Jul. 14, 2023. [Online]. Available: <http://www.cs.cmu.edu/~baraff/sigcourse/notesd2.pdf>
- [90] “Gazebo : Tutorial : Physics Parameters.” https://classic.gazebosim.org/tutorials?tut=physics_params&cat=physics (accessed Jul. 14, 2023).
- [91] “SDFormat Specification.” <http://sdformat.org/spec?ver=1.6&elem=physics> (accessed Jul. 14, 2023).
- [92] “RASPBERRY PI Módulo de câmara V2 oficial Raspberry Pi - 8MP.” https://mauser.pt/catalog/product_info.php?products_id=096-4061&gclid=EAIaIQobChMI8s_G_-P7_AIVMI9oCR2AtATvEAQYASABEgIIvD_BwE (accessed Jul. 14, 2023).
- [93] “Raspberry Pi Documentation - Camera.” <https://www.raspberrypi.com/documentation/accessories/camera.html> (accessed Jul. 14, 2023).
- [94] “Buy a Raspberry Pi Camera Module 2 – Raspberry Pi.” <https://www.raspberrypi.com/products/camera-module-v2/> (accessed Jul. 14, 2023).
- [95] “PX4 Development Kit - X500 v2 – Holybro.” <https://holybro.com/products/px4-development-kit-x500-v2> (accessed Jul. 14, 2023).
- [96] “Holybro X500 v2 kits” <https://holybro.com/products/x500-v2-kits>

- [97] “Holybro Brushless Motor 2216-880KV CW / CCW | Flying Tech.”
<https://www.flyingtech.co.uk/electronics/holybro-brushless-motor-2216-880kv-cw-ccw> (accessed Jul. 14, 2023).
- [98] “pynput · PyPI.” <https://pypi.org/project/pynput/> (accessed Sep. 12, 2023).
- [99] “keyboard · PyPI.” <https://pypi.org/project/keyboard/> (accessed Sep. 12, 2023).

Appendix

