POLITECNICO DI TORINO

MSc in Aerospace Engineering

MSc Thesis

SIMULATED MONITORING OF SMART COMMUNITIES USING UNMANNED AERIAL VEHICLES

Advisors:

Guglieri Giorgio

Quagliotti Fulvia

Rutherford Matthew J.

Valavanis Kimon P.

Student:

Pannozzi Pierluigi

March 2019

Abstract

Nell'era del progresso tecnologico, i velivoli a pilotaggio remoto (RPAS) stanno diventando sempre più diffusi e importanti per un ampio spettro di operazioni nel campo civile. Le tecnologie abilitanti per le smart cities si stanno sviluppando rapidamente e i velivoli autonomi sono gli attori principali per la loro implementazione. Il monitoraggio urbano è il primo requisito che ogni città intelligente dovrebbe supportare, basato su sistemi di volo autonomi, per fornire un'alternativa "eye-in-the-sky" al monitoraggio da terra, per assicurare il controllo delle aree urbane, e la sicurezza, e infine rilevare eventuali anomalie nel quartiere sorvegliato. Per ricostruire uno scenario operativo realistico, è stato sviluppato un modello di città (Torino), implementato nel software di simulazione fisica Gazebo. Ciò fornisce la possibilità ai cosiddetti supervisori umani remoti (operatori della Ground Control Station) e ai piloti, in pilotaggio manuale, di addestrarsi e sviluppare le capacità di addestramento richieste senza rischio e con un costo associato minimo. Lo stesso strumento permette di valutare la fattibilità delle missioni pianificate in funzione del profilo di rischio. Verranno presentate simulazioni di volo autonomo, parzialmente autonomo e di pilotaggio manuale rappresentative dello scenario di monitoraggio di un'area urbana della città di Torino: Politecnico di Torino.

In the era of technological progress, Remotely Piloted Air System (RPAS) have become increasingly prevalent and important for a wide spectrum of civilian operations. The qualifying technologies for smart cities are developing quickly, and UAVs are the main actors for their implementation. Urban monitoring is the first requirement that every smart city should support, based on autonomous flight systems. Then, unmanned aerial vehicles are used to provide the eye-in-the-sky alternative to ground monitoring, assuring safety and detecting any kind of anomaly in the monitored district. In order to build a realistic operative scenario, a city model (Turin) has been developed, and implemented in the Gazebo software physics simulator. This gives the possibility to"internal pilots" (Ground Control Station piloting) and "external pilots" (manual direct piloting) to train and develop the required skills for professional and safe piloting without the potential risks and costs associated with real-life training. The simulations will show autonomous flight, partial autonomous flight and manual piloting of the monitoring scenario for an urban area of the city of Turin: Polytechnic of Turin.

Acknowledgments

It is always difficult to write a gratitude text when the number of people involved is large. It does not mean that others who are not referred to, do not deserve my attention and recognition, as well. To my dad Cesare and my mom Cristina, I express my gratitude for their help and sacrifices that made my graduation possible in Aerospace Engineering. To my close friends and my girlfriend, always there for me. My thanks in USA to my advisors Dr. Kimon P. Valavanis and Dr. Matthew J. Rutherford for the orientation and long discussions not only about the contents of this dissertation but also for the resources and opportunities provided while I was working at the University of Denver Unmanned Systems Research Institute (DU^{2}SRI). My thanks in Italy to professor Giorgio Guglieri and Fulvia Quagliotti for their support and for the possibility of this international experience that they gave to me, putting me in contact with Kimon P. Valavanis. This thesis built the bridge between the Polytechnic of Turin and the University of Denver.

Contents

Nomenclature

AGL: Above Ground Level

AI: Artificial Intelligent

API: Application Program Interface

BSD: Berkeley Software Distribution

CRP: Cloud Robotic Platform

DSS: Decision Support System

DOD: Department Of Defense

EASA: European Aviation Safety Agency

ENAC: Ente Nazionale per l'Aviazione Civile

FAA: Federal Aviation Administration

FDM: Flight Dynamic Model

FMU: Flight Management Unit

FOV: Field Of View

FPV: First Person View

GCS: Ground Control Station

GPL: General Public License

GPU: Graphic Power Unit

GPS: Global Positioning System

HITL or HIL: Hardware In The Loop

ICT: Information Communication Technology

IMU: Inertial Measurement Unit

IO: Input Output

IOT: Internet Of Things

IT: Information Technology

JOSM: Java Open Street Map editor

LOS: Line Of Sight

LTE: Long Term Evolution

M2M: Machine to Machine

MGTW: Maximum Gross Takeoff Weight

UAV: Unmanned Aerial Vehicle

MGTW: Maximum Gross Takeoff Weight

MP: Mission Planner

MSL: Mean Sea Level

ODE: Open Dynamic Engine

OS: Operative System

OSM: OpenStreetMap

OSRF: Open Source Robotics Foundation

PNG: Portable Network Graphics

QGC: QGroundControl

RC: Radio Control

RFID: Radio Frequency Identification

RGB: Red Green Blue

ROS: Robotic Operating System

RTL: Return To Launch

SDF: Spatial Data File

SITL or SIL: Software In The Loop

TCP: Transmission Control Protocol

UAS: Unmanned Aircraft System

UAV: Unmanned Aerial Vehicle

UE: Unreal Engine

UDP: User Datagram Protocol

UGCS: Universal Ground Control Software

URDF: Unified Robot Description Format

UTM: Unmanned Traffic Management

VGI: Volunteered Geographic Information

VTOL: Vertical Take Off and Landing

XML: Extensible Markup Language

1 Introduction

Unmanned aviation has witnessed exponential growth over the last years, and civilian applications are supposed to dominate the field in the near future [1]. UAVs are expected to be implemented in a wide range of applications.

Regardless of the UAV type, the architecture of a typical UAV consists of components comprising the vehicle itself, the onboard sensors, the ground control station, and the communication channel.

According to the U.S. Department of Defense (DoD) [2], UAVs are classified into five classes based on size, takeoff weight, operating altitude, and airspeed see Table 1. There are official sites in the US that experiment using UAVs to support civil

Category	Size	MGTW (lbs)	Normal Operating Altitude (ft)	Airspeed (knots)
Class 1	Small	0-20	<1200 Above Ground Level	<100
Class 2	Medium	21-55	<3500 AGL	<250
Class 3	Large	<1320	<18000 Mean Sea Level	<250
Class 4	Larger	>1320	<18000 MSL	Any airspeed
Class 5	Largest	>1320	>18000 MSL	Any airspeed

Table 1: UAVs classification according to the US DoD [2, 3].

operations. Common applications include law enforcement, surveillance, firefighting, disaster relief, search and rescue missions, to name a few. However, cities must obtain permission from the FAA to use UAVs for civil domain purposes; the process includes submitting plans and having a qualified pilot in place. Similar applies for EASA (Europe), while national aviation agencies (i.e. ENAC for Italy) are in charge of UAV management and integration until next European Union (EU) regulations are fully adopted in the second half of 2019 [4].

This research addresses the use of UAVs for smart city monitoring; that is, to

Category	Weight of UAV	Radius of Mission	Endurance	Use case
Micro	<2kg	5kg (LOS)	A few hours	Reconnaissance, Inspection, Surveillance
Mini	2-20kg	25kg (LOS)	Up to 2 days	Surveillance, Data gathering
Small	20-150kg	50kg (LOS)	Up to 2 days	Surveillance, Data gathering

Table 2: UAVs classification, from [7].

provide support for a variety of applications where the main function of the UAV is to serve as the eye-in-the-sky component. However when it comes to civil application, UAVs can be classified in different ways, indeed see the three categories shown in Table 2, where the first two rows correspond to Class 1, shown in Table 1. In general, there is no consistent standardization but there are different classifications in terms of various aspects like endurance, altitude, range etc. Different classifications may be found in [5, 6].

1.1 Motivation and Rationale

A student at MIT has said "Everyone is building drones these days, but nobody knows how to get them to stop running into things"[8]. This quote has underlined the necessity of experimentations and training through simulation in order to integrate UAVs in smart cities safely.

The concept of smart cities begins with the integration of technology that needs to provide services efficiently and fast to residents. On the same page, the similar concept of "*senseable cities*" [9]: it must be possible to monitor the state of health of the city to improve the livability of citizens. Compared to the term smart city, senseable city wants

to underline the "human" dimension that despite the role of technology, continues to play an important role.

However, for smart communities, challenges such as safety, security, and privacy in densely populated regions remain a concern when integrating within the city infrastructure. Under this consideration, UAVs have the capability to quickly transform futuristic ideas to reality and to contribute to a better way of living [10]. Some advantages and support for a variety of applications where society may benefit by using UAVs are listed below:

- Defense: while UAVs have been used by the military (the Predator UAV is among the most well known), smaller, portable UAVs are now being used by ground forces, on a regular basis. They explore, collect data and, most important, they are expendable [11].

- Emergency response: innovations in camera technology have a significant impact on using UAVs. UAVs equipped with thermal imaging cameras may provide emergency response teams with data to identify victims, difficult to spot with a naked eye. In addition to emergency response, UAVs have proven useful in times of natural disaster [12]. In the aftermath of hurricanes and earthquakes, UAVs have been used to assess damage, locate victims, and deliver aid. And in certain circumstances, they are being used to prevent disasters.

- Tracking disease expansion: tracking animals allows for researchers to track diseases. UAVs with thermal imaging cameras have been used by the London School of Hygiene and Tropical Medicine to track macaque movements in the province of Palawan in the Philippines — a region where malaria is an active threat [13].

- Monitoring the degree of pollution in the atmosphere [14].

- Agriculture: farmers strive to reduce cost and expand yield. With the use of UAVs, farmers are able to gather data, automate redundant processes, and generally maximize efficiency ([15], [16]).

- Waste management: recycling and biodegradation have dramatically improved global waste management. However, innovations in waste collection are still

emerging. Fortunately, UAVs operate at the forefront of these initiatives and have helped to clean oceans [17].

- Construction: one of the most popular commercial use cases of UAVs is construction planning and management. Software developers have created solutions that analyze construction progress with regularly captured data. While ground surveying is still a critical part of construction planning and monitoring, the use of UAV data has become increasingly important [18].

- Urban planning: as urbanization continues, cities must adapt to larger populations and chronic congestion. Urban planning has become increasingly important for cities but requires a thorough understanding of metropolitan rhythms and flows. With the use of UAVs, urban planners are able to better understand their environments and implement data-driven improvements [19].

- Pilotless personal transport called Self-flying Air Taxi ([20], [21]) that is the unmanned version of the Uber air project [22].

- Delivery: UAVs generally use 4-8 propellers and rechargeable batteries to provide thrust and attach packages underneath the body. Delivery UAVs are operated autonomously or remotely, with operators potentially overseeing multiple UAVs at once. A famous example is Amazon Prime Air [23].

- Cinema: one of the first industries to adopt UAVs was professional filming. UAVs have allowed producers to capture dramatic aerial perspectives without the use of helicopters (more expensive). This has had a major impact on Hollywood's bottom line, pushing the limits in cinematography [24].

- Urban Monitoring and surveillance: there has been a growing interest in using unmanned aerial vehicles for information collection tasks, in civil domains. With the increasing numbers of UAVs, there is a growing need to enable the UAVs to perform the information collection mission autonomously without the need for direct human control, which is costly. Intelligent multi-agent techniques have been employed to address this problem and complex surveillance algorithms have been developed [25].

Regardless, urban monitoring is the focus of this thesis. Mohammed et al. [26] underline that UAVs are making cities smarter, more connected with the people and bringing them degrees of freedom that were before unthinkable. Moreover, UAVs will be an integral component of the future business. Although the tech giants tend to catch headlines with their UAV initiatives, start-ups actually drive most UAS activities. For example, more than 300 smart companies have entered the market since 2000 in the USA. These companies typically focus on hardware, support services, or operations. The latter is a broad category that includes software and services related to navigation and Unmanned Traffic Management (UTM), mitigation of threats related to unmanned aerial vehicles, construction, and maintenance of UAS-related infrastructures, such as vertiports to accommodate UAVs that take off and land vertically (VTOL). These startups receive funding because financiers believe in new emergent smart technologies. Figure 1 and 2 show the ongoing research and related investments. These are recent

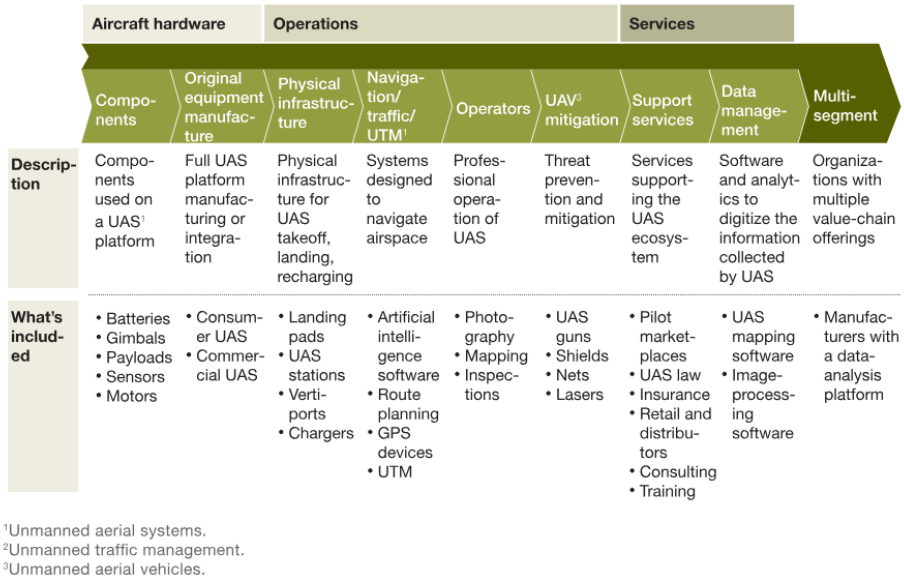


Figure 1: Research, from [27].

data, taken from an analysis conducted by McKinsey [27]. In addition, according to SESAR [28], the unmanned industry market will reach equals to 10 € billion annually by 2035, over 15 € billion annually by 2050 with civil applications nominating.

The European Platform for Intelligent Cities and the European Network of Living Labs defined smart cities as "The use of discrete new technology applications such as Radio Frequency IDentification (RFID) and Internet of Things (IoT) through more

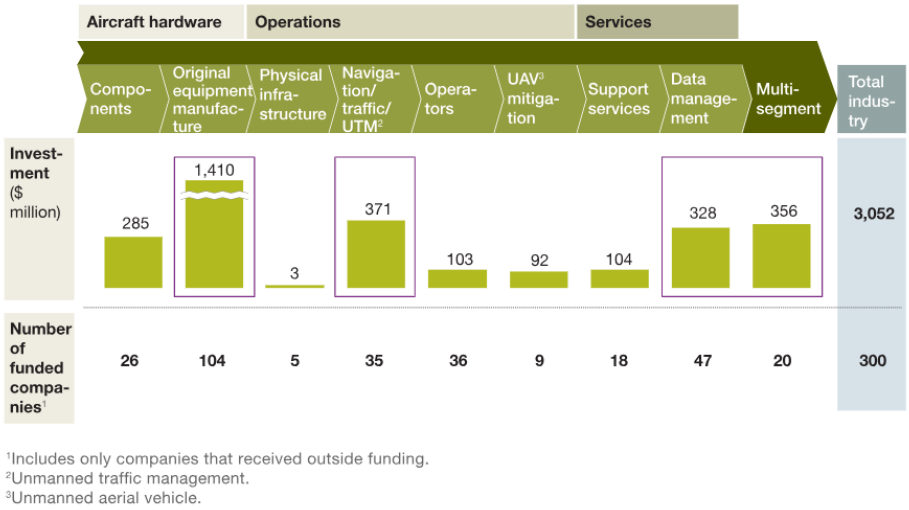


Figure 2: Investements, from [27].

holistic conception of intelligent, integrated working that is closely linked to the concept of living and user-generated services" [29]. Generally, a smart city is the city that seeks to achieve the objectives of a future city by utilizing ICT solutions and trends. Design of such a smart city requires huge and complete integration of ICT and its tendencies. UAVs contribute to these goals. However, during the coming decade, advances are expected to continue in the areas of cloud computing, wireless sensors, networked unmanned systems, big data, open data, and internet of things. As result, billions of devices will be connected together. Consequently, there will be a substantial opportunity for using UAVs in smart cities. There are several important core components that define the concept of a smart city, see Figure 3, they are provided below [30]:

- Management and organization: the alignment of management and organizational goals is the key point for a smart city to work effectively and efficiently.
- Technology: a smart city relies on a collection of smart computing technologies applied to critical infrastructure components and services. Smart computing refers to a new generation of integrated hardware, software and network technologies that provide Information Technology (IT) systems with real time awareness of the real world and advanced analytics to help people make more intelligent decisions and giving them more degrees of freedom.
- Governance: it involves the implementation of procedures with constituents who

exchange information according to rules and standards in order to achieve objectives. Several factors like collaboration, communication, leadership, and data exchange are needed for effective smart city governance.

- Policy-context: the policy context is crucial to the understanding of the use of information systems in proper ways. It mainly characterizes institutional and non-technical urban issues and creates conditions that enable urban development.
- People and communities: smart communities initiatives allow citizens to participate in the governance and management of the city and become active members.
- Economy: it is one of the major drivers of smart city initiatives and a city with a high degree of economic competitiveness is thought to have one of the properties of a smart city. The outcomes are mostly business creation, job creation, workforce development, and improvement in productivity.
- ICT infrastructure: the implementation of an ICT infrastructure is vital to a smart city's development and depends on some factors related to its availability and performance.
- Natural environment: one of the main focus of a smart city is to increase sustainability and to enhance natural resource management.

To develop smart city solutions, the complexity of how smart cities are operated, financed, regulated and planned needs to be considered. Falconer et al. [31] claimed that any smart city structure consists of four layers:

- Objectives: social, technological environmental and economic aims.
- Indicators.
- Components.
- Contents: solutions and services.

The main goal of any smart city design is to create a sustainable place where people can live, work, play and enjoy. Therefore, the smart city development is divided into elements. These are smart city infrastructure, smart database resources, smart building

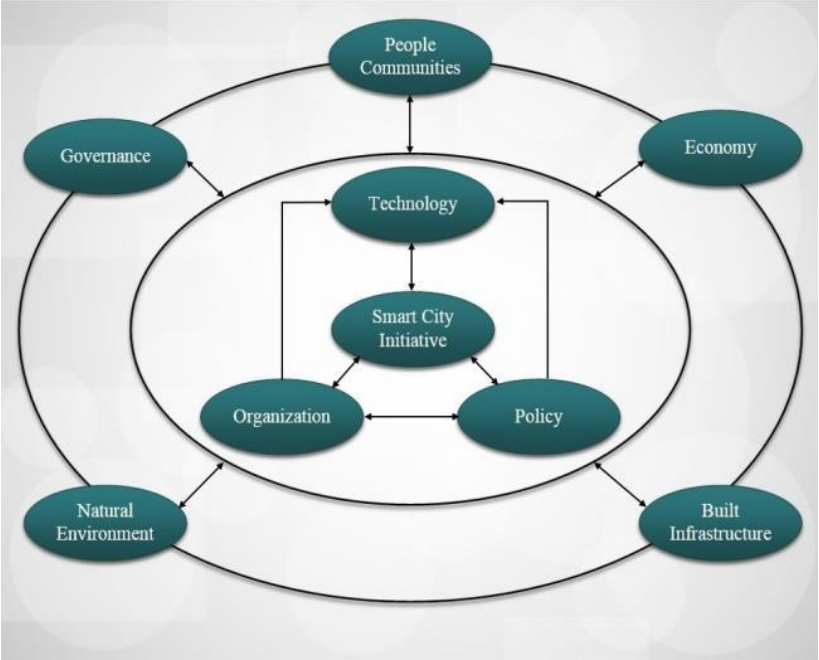


Figure 3: Smart city key factors, from [26].

management systems, and smart interface. These elements, integrated together, make up the smart city.

The integration of UAV solutions with M2M [32], RFID [33], LTE [34], and live video streaming increased the role of UAVs in urban safety areas. In addition, trends towards intelligence and data mining give UAVs opportunities to be involved in civil security activities. Furthermore, the involvement of UAVs in surveillance activities will reduce costs and will increase the efficiency of operations and interventions. The efficiency of security and safety systems in a city has become a serious concern not only for smart cities but also for any type of communities. In addition, the integration of mobile applications, protected and reliable wireless networks, forensic mapping software, and UAVs can help smart cities become a secure place for living. Using UAVs in disaster situations like fires, floods, earthquakes, etc. will help authorities to control such emergency situations efficiently and accurately. UAVs will inspect and evaluate the situation perfectly and also help in acting properly in certain disastrous situations because the UAVs can reach in areas that humans, instead, cannot.

Security management (urban/civilian security) is also a crucial point. Usage of UAVs in such area will allow the city to deploy a quick operations room, updated with efficient data flow and will allow for the city to manage big public events with huge

numbers of attendee's regularly. Mohammed et al. [26] state that future challenges can be classified into business and technical ones. Following a focus on these challenges.

1.1.1 Socio-Economic Challenges

Regarding ethics and privacy, many would not approve the use of UAVs for monitoring and surveillance of the general population as they may think of it as an invasion of their privacy [35].

When it comes to cost [36] the UAVs development can be expensive because of their technical and deployment issues, training and integration of systems. Designing a UAV for a specific service is also expensive as it needs to function properly and correctly.

Moreover, to use UAVs in a country it should be at first registered in that location. Flying UAVs might affect the airplanes and the navigation of their routes. So, a country must develop related regulation, licensing and legislation for UAVs deployment and use.

1.1.2 Techincal Challenges

First of all, it is needed the development of fail-safe systems, to guarantee high safety confidence levels in the event of aircraft failure, or loss of all communications between the UAV and the control center, or generally talking, all of the kind of hazards related to the concept of risk [37].

Secondly, it is required the development of very efficient, low vibration, engines, and gyro-stabilized platform technology, for high-resolution imaging and accurate measurements (through camera sensor, etc.).

Furthermore, it also matters of technical challenges demonstration of precision flying, in terms of altitude and flight path, over extended periods of time, in all weather conditions during both day and night periods.

In addition, wireless sensors can be used for smooth operations of UAVs. For example, surveillance and a live feed from wireless sensors can be used to control traffic systems.

Moreover, development of sense and avoid mechanisms enable a UAV to become aware of its environment enabling it to take evasive action if necessary and development

of automated image data compression algorithms, stitching of aerial imagery are key elements. Data fusion software can intelligently fuse many pieces of information from a large number of sensors. Subsequently, automated computer-based interpretation of data can take place.

Lastly, the development of a network-centric infrastructure, to enable any member of a team to control the UAV and retrieve imagery and sensor information, in real time.

Overall, it is forecasted that integrating properly UAVs with smart cities will create a sustainable business environment and a peaceful place of living for the smart community citizens.

1.2 Problem Statement

Real experiments using UAVs are costly and in the urban contest still risky; therefore, the performance of UAV systems should be validated in simulated world environments before actual deployment. This requires trustable simulation tools and models. For this research, the focus is on urban monitoring with safety, surveillance and traffic monitoring [38] being vital elements of it. The goal is to create a simulated framework characterized by a 3D city model (including walking or running people, cars, buses, trams etc.) that corresponds to a realistic urban scenario. At the same time, a UAV will be placed in this simulated world environment to monitor the area, autonomously, with an uploaded dynamic path characterized by waypoints. Furthermore, the UAV could be piloted by a joystick (RC, PlayStation controller, etc.) or a smartphone application, and this gives the possibility for an "internal pilot" (ground control station operator) or "external pilot" (manual piloting) to train in a simulated world before real-time flights, without risking the integrity of the expensive prototype.

Software problems to be based implementation includes challenges, some of which are listed next:

- Create a realistic 3D world with a simulation physics software, in which different types of multi-rotor UAVs will monitor and survey an urban district with minimum or no interface with humans. Stated differently, UAVs will act as the eye-in-the-sky component in order to safeguard humans and detect or predict an eventual anomaly.

- Path planning (dynamic) based on information received by the ground control station software and implemented using with Robot Operating System (ROS) [39], a meta operative system. This will be followed by Software In The Loop (SIL) [40] studies. Furthermore, it will perform an act of recording and collecting data and images thanks to the simulated camera attaches on the simulated UAV.

The expected result is to develop an accurately simulated city monitored and surveyed by UAVs. The part of the city of Turin (Piedmont, Italy) that has been chosen is the Polytechnic of Turin district.

1.3 Proposed Method of Solution

The simulated environment and urban monitoring mission will be achieved as follows:

- Using the OSM [41] tool in order to select the area (2D) to be monitored.
- Since there are limitations about exporting a selected area with OSM, in order to circumscribe a specific area, it is necessary to use the OSM editor: JOSM [42]. It gives the possibility of removing streets, squares etc. Another option is downloading a bigger area and execute urban monitoring in an inner district.
- After having selected and edited the area, the next step is performing a 3D extrusion in order to obtain the 3D city model (the district). Then, convert it in the required format for being uploaded in the physics software simulator.
- Finally, the 3D model is imported into the software simulator, and it is now possible to place animated models inside the city making it more complex and realistic.
- Planning a mission for the UAV (through GCS) which will be inserted in the world, it will fly autonomously (or it can be piloted) and will scan and monitor the district area. The output will be a streaming video, First Person View (FPV) or Pan and Tilt sensors are available.

The work will be conducted is creating a realistic world in Gazebo [43], adding people, cars, trains, trams, etc. in the main file.world that at the beginning has just the 3D map. Since the simulation is about urban monitoring in smart communities, it is required the

use of a trustworthy simulation tool: SIL. It is a simulation that gives the opportunity to operate plane, copter or rover, without the need for any hardware. SIL becomes an extraordinary practical tool since end-product can misbehave in-flight. It influences on avoiding hazard situations and preserving costly equipment being damaged.

1.4 Summary of Contributions

The primary contribution of this work is the development of a 3D city model, with a basic degree of detail, placing it in a physics software simulator and reproducing a true traffic scenario where a UAV will perform an act of monitoring, safeguarding for the districts moving around and above the city. Regardless of urban monitoring, it will collect and record data through the first person view camera. The FPV will stream in a specific window while the simulation is running in Gazebo [43]. The achievements and contributions are summarized as follows:

- Create a Gazebo city world starting from OpenStreetMap [44].
- Insert the 3D model in Gazebo always maintaining a discrete level of detail (advanced textures).
- Filling it with models (animated box and actor) and making them in action with path planning and a waypoints/timepoints loop (it is possible to decide the exact time models will reach the desired point).
- Simulated urban monitoring thanks to the FPV video streaming in a dedicated window. The urban monitoring will be done in three ways: autonomously, namely using the waypoints of the GCS software, semi-autonomously with the GCS and the partial intervention of a pilot (Playstation joystick, gamepad, RC) and piloting directly the UAV with a smartphone application. This comparison has been done in order to show all of the kind remote control intervention a GCS operator or an external pilot can conduct.

1.5 Thesis Outline

The remainder of this dissertation is organized as follows: Chapter 2 presents a literature review to provide background information on computational and experimental

work done on a simulated environment for unmanned aircraft. Chapter 3 explains the problem statement, its importance, and its complexity. Chapter 4 includes a proposed solution, a step-by-step guide while Chapter 5 details the results and the case studies. Two case studies have been analyzed, the first one regard the urban monitoring realized by the ground control station operator that uploads waypoints in the GCS software. In the second one, the GCS operator interacts with the UAV using a Playstation Joystick, and in another experiment, a pilot controls directly the UAV with the usage of a smartphone thanks to its UAV app. In general, the two case studies reflect the use and the tests on two different packages: PX4 and Sphinx-Parrot. Chapter 6 summarizes the research performed throughout this dissertation and describes the work needed to further advance the technology for the future works.

2 Literature Review

This Chapter provides a comprehensive related literature review. It presents selecting a 2D map, editing and exporting it in 3D format. It discusses choices of 3D software physics simulators based on open source and commercial software comparisons. Furthermore, it summarizes autopilot and ground control station tools, waypoints, path planning, and Gazebo animated models. There are also other applications being considered like, for example "tracking objects", or the Hardware In The Loop (HIL) simulation and the visualization tools of ROS.

2.1 3D City Model

OpenStreetMap [44] is one of the most promising and popular projects for Volunteered Geographic Information (VGI). The community bears an enormous potential of "humans acting as remote sensors". OSM aims at creating a comprehensive and free online map with global coverage, it is open source. Following the collaborative Wikipedia approach, everybody can contribute, edit and improve the data of OSM. Regarding the data model, OSM is kept as simple as possible:

- Users provide so-called nodes, that is geo-referenced points with longitude and latitude information.
- For the description of line string geometries, several nodes can be combined with ways. Thereby, a closed way represents a polygon (e.g., a building footprint), whereas a non-closed way represents a line (e.g., a street).
- Possibility to add elements (lakes, streets, etc.), so modifying and editing maps with criteria.

Avanesov et al. [45] use a 3D city model, it is a complex task and in order to speed up the process, it has been used data from OpenStreetMap and its satellite project. First of all, it is needed to crop the OSM country description to reflect the area of interest. Secondly, it has been used OSM2World to create an initial 3D scene of the area selected and then exported to a Wavefront format. Thirdly, it has been imported the file into Blender [46], a 3D open content creation suite, where the scene can be

edited (e.g. by adding building facade textures). Part of the Avanesov's work about how to create a world with OpenStreetMap is well explained by a specific chapter of the Sphinx-Parrot Guide [47]. This is a step-by-step guide, in which all of the code lines are explained in order to export a 3D city model from OpenStreetMap to Blender, then from Blender to Gazebo [43].

Goetz et al. [48] study a similar way to do the 3D extrusion, he uses OSM-3D that is the OpenStreetMap Globe tool. Trying to use OSM data and to create a more realistic virtual perspective on the real world, OSM-3D utilized this data and integrated reoccurring 3D features. When processing the OSM data, all trees and streetlights are stored in a database table with their point-geometry.

OpenStreetMap has some limits regarding the export function of a 2D map, for instance, there is a rudimentary select button. It is not possible to decide the orientation of the map but instead allows the user to select an arbitrary rectangular area that can be useless sometimes. The solution is using an editor called JOSM ([42], [49]) that gives the possibility to modify the map in order to have the 2D area desired. This editor requires Java [50] to be run on Windows. It is also possible to install it on Ubuntu 16.04 LTS (Xenial Xerus). The output file from JOSM is still an OSM file.

An example of commercial software is CityEngine [51]. It is advanced 3D modeling software for creating, with a high-quality rendering, a more realistic scenario in comparison with the open source software.

2.2 Simulation Software

It is no longer necessary to build a robotic simulator from the ground up. There are many available open source and commercial simulators and game engines that can be used to simulate a robotic vehicle with high physical and functional fidelity. Any modifications or improvements made to these existing simulators should be released to the community (if open source) to drive the capabilities of available robot simulators. Murphy et al. [52] explain and underline the open source simulators:

- MissionLab [53].
- Gazebo [54].
- SimRobot [55].

- FlightGear [56].
- SubSim [57].

The next tables compare different software (open source and commercial software) regarding any area of interest and a final decision about what simulation software will be used is conducted by argumentations. In Table 3 it is possible to see that Gazebo and Unity [58] are the easiest ones to work with but Gazebo is open source, while Unity is not. Regarding Table 4 (where 1 is the minimum value and 5 is the highest) Gazebo looks being the most compatible software with ROS. From Table 5 the focus has been done on the comparison between Gazebo and JMAVSim that are the easiest ones to develop (open source); Gazebo more than JMAVSim has the possibility to modify the onboard sensors and the availability of any robots that can also detect obstacles.

Also, Ebeid et al. [60] compared open source software and made a huge analysis of them. His survey covers open-source hardware, software, and simulation UAV platforms and compares their main features. A comparison of selected Open-Source Simulator (OSSIM) UAV platforms has been conducted, including the implementation language, the supported operating system, and the licensing terms. C and C++ are the most popular implementation languages, with Python and Java. All projects support Linux, some additionally support MacOS and/or Windows. More details are reported in [60].

2.2.1 AirSim

Aerial Informatics and Robotics Platform (AirSim, see Figure 4) was developed by Microsoft [61] in 2017. AirSim aims to support the development and testing of algorithms for autonomous vehicle applications, such as deep learning, computer vision, and reinforcement learning algorithms. The AirSim physics engine is based on Unreal Engine 4 (UE4) [62] and can operate at a high frequency for real-time HIL simulations with support for prominent protocols (for example MavLink). The simulator follows modular design architecture with an emphasis on extensibility. The only supported UAVs are Iris in a multi-rotor model and in an X-configuration for PX4 quadrotor.

Simulator	Physical Fidelity	Functional Fidelity	Ease of Development	Cost
USARSim	High	Medium	Medium	Medium
X-Plane	High	High	Medium	Medium
FlightGear	High	Medium	Medium	Low
MS Flight Simulator	High	Medium	Medium	Medium
Webots	Medium	Medium	Medium	Medium
Simbad	Medium	Low	Medium	Low
Player, Stage, GAZEBO	Medium	Low	High	Medium
eyeWyre	Medium	Low	Medium	Medium
MS Robotics Studio	High	High	Medium	Low
Matlab and Simulink	Low	Medium	Medium	High
MissionLab	Medium	Low	Medium	Medium
SimRobot	Medium	Low	Medium	Low
SubSim	Medium	Low	Medium	Medium
Unity	High	High	High	High

Table 3: Simulators comparison, from [52].

2.2.2 Gazebo

Gazebo was originally developed at the University of Southern California [63], the USA in 2002 and later at Open Source Robotics Foundation (OSRF). Gazebo is the default simulator included with Robot Operating System, making it one of the most popular 3D dynamics multi-robot simulators with a very operating community, see Figure 5 [54]. Gazebo allows the use of different physics engines and sensor models, and supports easy creation of 3D worlds, enabling testing of robot designs and algorithms,

	V-REP	Gazebo	MORSE	Webots	USARSim	STDR, Stage	Unity
Main Program Language	C++	C++	Phyton	C++	C++	C++	C++
Operating System	Mac, Linux	Mac, Linux	BSD, Mac, Linux	Mac, Linux	Linux	Linux	Linux
Simulation Type	3d	3d	3d	3d	3d	2d	3d
Physics Engine	ODE, Bullet, Vortex, Newton	ODE, Bullet, Dart	Bullet	ODE	Unreal	OpenGL	Unity 3d
3d Rendering Engine	Internal, External	OGRE	Blender game	OGRE	Karma	-	OGRE
Portability	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Support	4	5	4	4	3	4	2
ROS Compatibility	4	5	4	3	2	4	1

Table 4: Simulators comparison, from [59].

and training of AI systems using realistic frameworks and scenarios. Gazebo uses a distributed architecture with separate libraries for physics simulation, rendering, user interface, communication, and sensor generation. Although Gazebo is a feature-rich platform, it has some limits regarding the rendering techniques that are not as advanced as, e.g., Unreal Engine [62] or Unity [58].

Simulator	FlightGear	X-Plane	JMavSim	Gazebo	Air Sim	UE4Sim
Commercial, Free	Free	Commercial	Free	Free	Free	Free
Vehicles	Airplanes, some multirotor	Airplanes, some multirotor	Multirotor	Multirotor and any robots	Multirotor	Multirotor, cars
Interface ROS	No	No	Yes	Yes	No	No
Sensors	Diversity of sensors	Easy incorporation of sensors	No incorporation of sensors	Easy modification of sensors	Monocular, depth cameras, no lidar	Easy modification of sensors
Motion capture	No	No	No	No	Yes	Yes
Obstacles	Yes	Yes	No	Yes	Yes	Yes
SITH-HITL	Yes	Yes	Yes	Yes	Yes	No
MAVLink	Yes	Yes	Yes	Yes	Yes	No
Easy of Development	Medium	Medium	High	High	Medium	Medium

Table 5: Simulators comparison, from [7].

2.2.3 Morse

Modular Open Robots Simulation Engine (MORSE, see Figure 6) is a simulator for research in robotics that is developed jointly at Laboratory for Analysis and Architecture of Systems (LAAS) and Office National d'Etudes et de Recherches Arospatiales (ONERA) [64] since 2011. MORSE relies on Blender [46] for physics simulation and for a realistic display of the simulated world. MORSE is invented as a general purpose, modular system simulation of various moving robots in many different kinds

Figure 4: AirSim rendering, from [60].

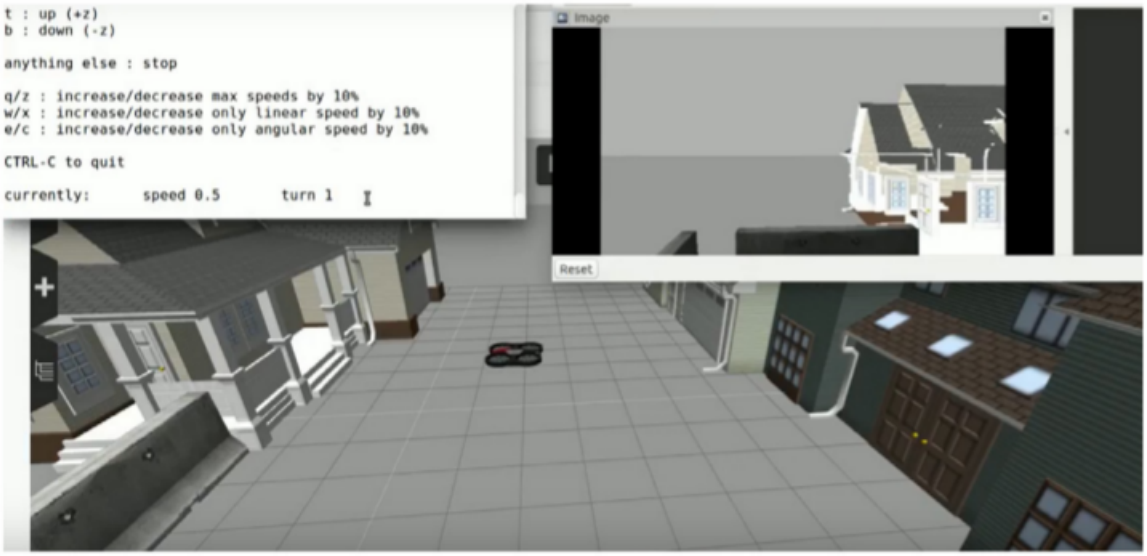


Figure 5: Gazebo rendering, from [60].

of environments. MORSE relies on a component-based architecture. Each MORSE component consists of a Blender and a Python file. The Blender file expresses the physical and visual properties of the object in the simulated framework. The Python file delineates an object class for the component type. The MORSE component library provides a quadrotor dynamic which is a simple definition of a quadrotor with a rigid

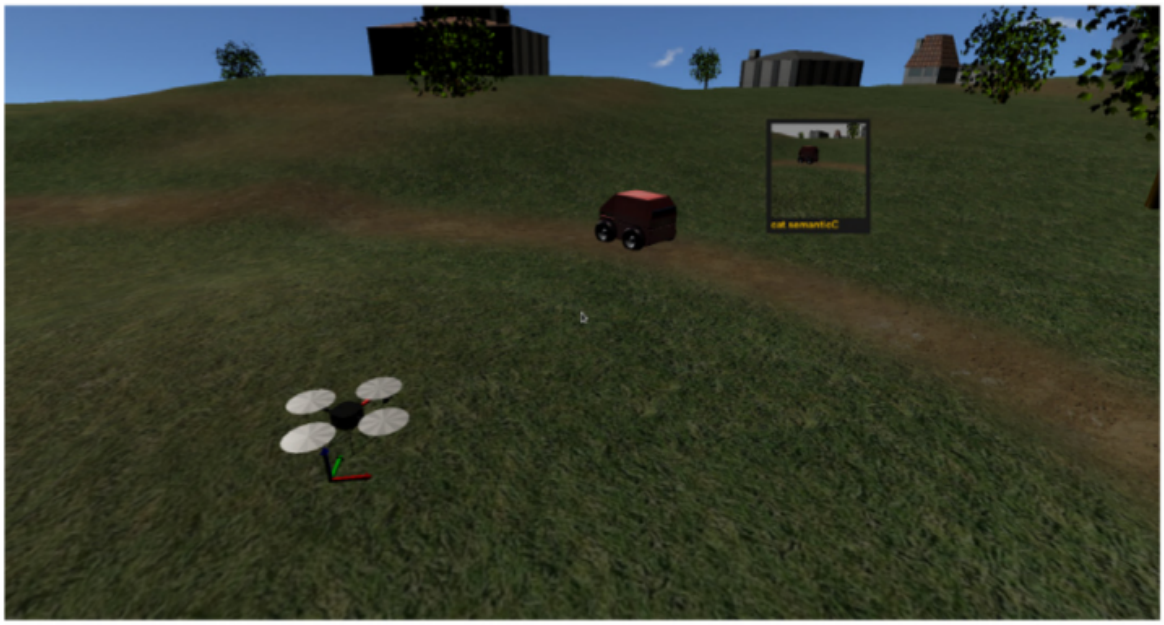


Figure 6: Morse rendering, from [60].

body. MORSE lacks graphical user interface since it only provides a command line one. MORSE does not embed any advanced algorithms (e.g., path planning) since it expects to run such algorithms in the simulated software stack of the desired object.

2.2.4 jMAVSim

Java Micro Air Vehicle Simulator (jMAVSim, see Figure 7) is a basic and lightweight multirotor simulator developed by the PIXHAWK engineering team [65]. jMAVSim supports the MAVLink protocol, uses the Java3D library for rendering, and connects directly to the HIL via a serial connection or to SIL via User Datagram Protocol (UDP) communication to the autopilot. The implementation of jMAVSim is as a single component based on a relatively simple object-oriented design with no explicitly documented architecture.

2.2.5 New Paparazzi Simulator

New Paparazzi Simulator (NPS, see Figure 8) [66] is an advanced simulator with sensor models developed at Ecole Nationale de l'Aviation Civil (ENAC) UAV Lab. The default flight model in NPS is JSBSim which supports a range of relatively complex airframes. NPS permits the use of different Flight Dynamic Model (FDM) back-ends, enabling developers to choose their own FDM model, for example connecting the

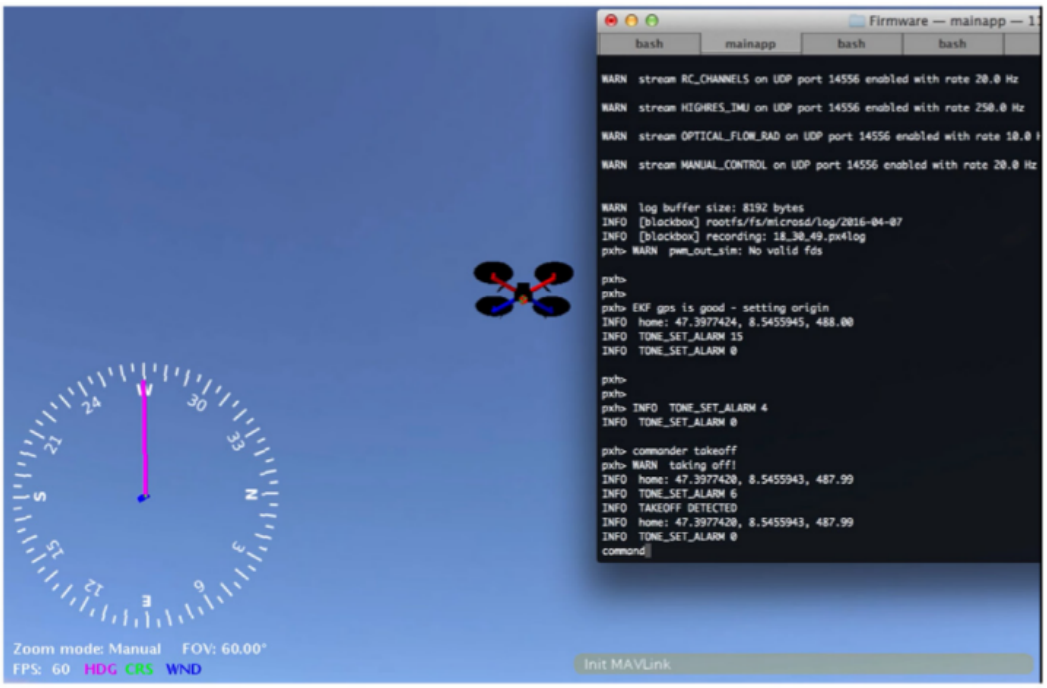


Figure 7: jMAVSim rendering, from [60].

simulator to an FDM expressed in MATLAB - Simulink. Different visualization options are available, including FlightGear [56] and the Morse simulator. Supported UAVs include rotor-craft and fixed-wing models.

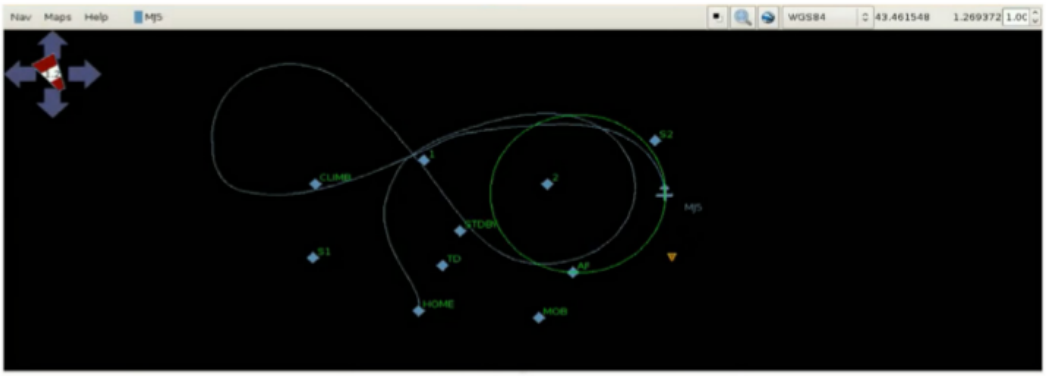


Figure 8: New Paparazzi rendering, from [60].

2.2.6 HackflightSim

HackflightSim [67] is a basic cross-platform quadcopter simulator developed by Simon D. Levy, Washington and Lee University Lexington, the USA in 2017. HackflightSim is implemented in C++ programming language, uses Unreal Engine 4 [62], and is based on the Hackflight firmware which is a Simple C++ quadcopter flight control

firmware for Arduino [68]. The project is analogous to AirSim however, it focuses only on quadcopter firmware. The implementation of HackflightSim, as shown in Figure 9, is as a single component that uses the Unreal Engine for visualization, similarly to jMAVSim the main component is based on a relatively simple object-oriented design with no explicitly documented architecture.

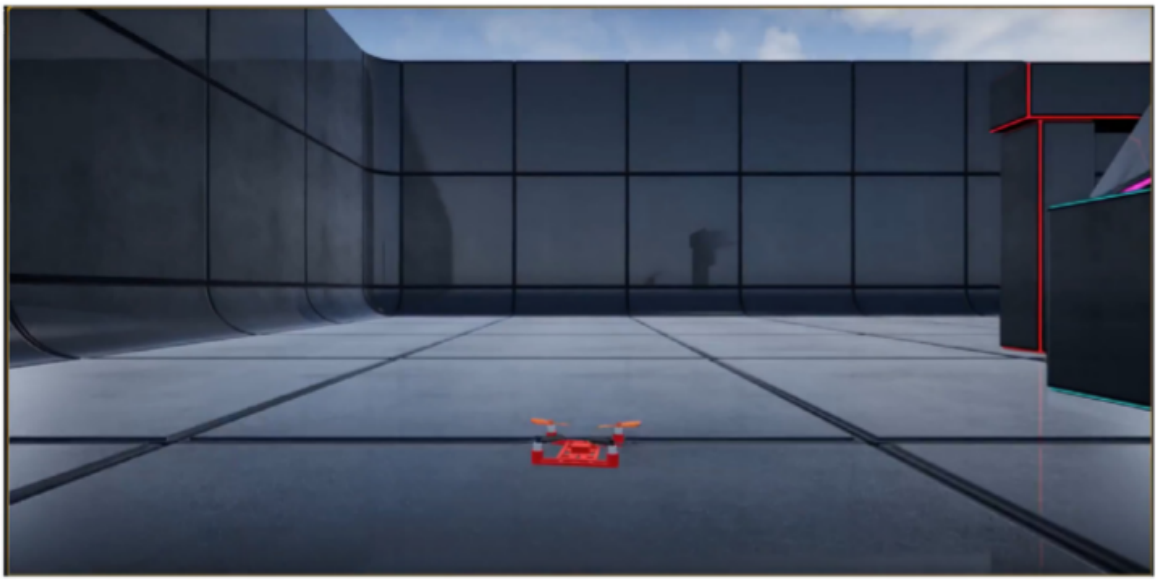


Figure 9: HackflightSim rendering, from [60].

2.3 Software choice: Gazebo

Gazebo is chosen because:

- Open source.
- Easy to develop.
- Sensors interface in the simulation.
- Best compatibility with ROS.

Therefore, the reasons for this choice are that Gazebo is the easiest SW to work with and it is open source. Moreover, Gazebo looks being the most compatible software with ROS and it is sensors compatible that is crucial in order to make possible the urban monitoring with the UAV camera.

Gazebo is a 3D dynamic simulator with the ability to accurately simulate populations of robots in a complex indoor and outdoor environments. It is possible to use Gazebo for:

- Testing robotics algorithms.
- Designing robots.
- Performing regression testing.
- All system training using realistic scenario.

Talking about features, Gazebo:

- Focuses on accurate physical simulation.
- Supports common robot control software (custom client code, ROS interface).
- Supports the sharing of resources (sensors, actuators, etc.).
- Makes universal test environment for robotics applications.
- Is free and open source.

Gazebo runs two processes: server and client, as shown in Figure 10. The former runs the physics loop and generate sensor data; it is the core of Gazebo and it can be used independently of any graphical interface. The latter is a user interconnection and visualization of a simulation.

Figure 10: Gazebo server and client.

Client code (user program) can interface to Gazebo in two ways:

- Libgazebo: direct interface with simulator client.
- ROS (recommended).

In Gazebo, it is thinkable to upload all of the kind of models, already available or created by the user itself and therefore this allows to create a realistic simulated dynamic scenario for a huge variety of simulations. The collision element specifies the shape used by the collision detection engine (cube, parallelepiped circumscribing the models or the model itself). The visual element specifies the shape used by the rendering engine. For most application cases, the collision and visual elements are the same.

In order to run a base simulation on Gazebo of multiple actors, Huang et al. [69] say that the limit is putting 16 actors for a real-time simulation with the same textures, for more simulated people with different textures it is necessary the use of some add-on that can work with Gazebo.

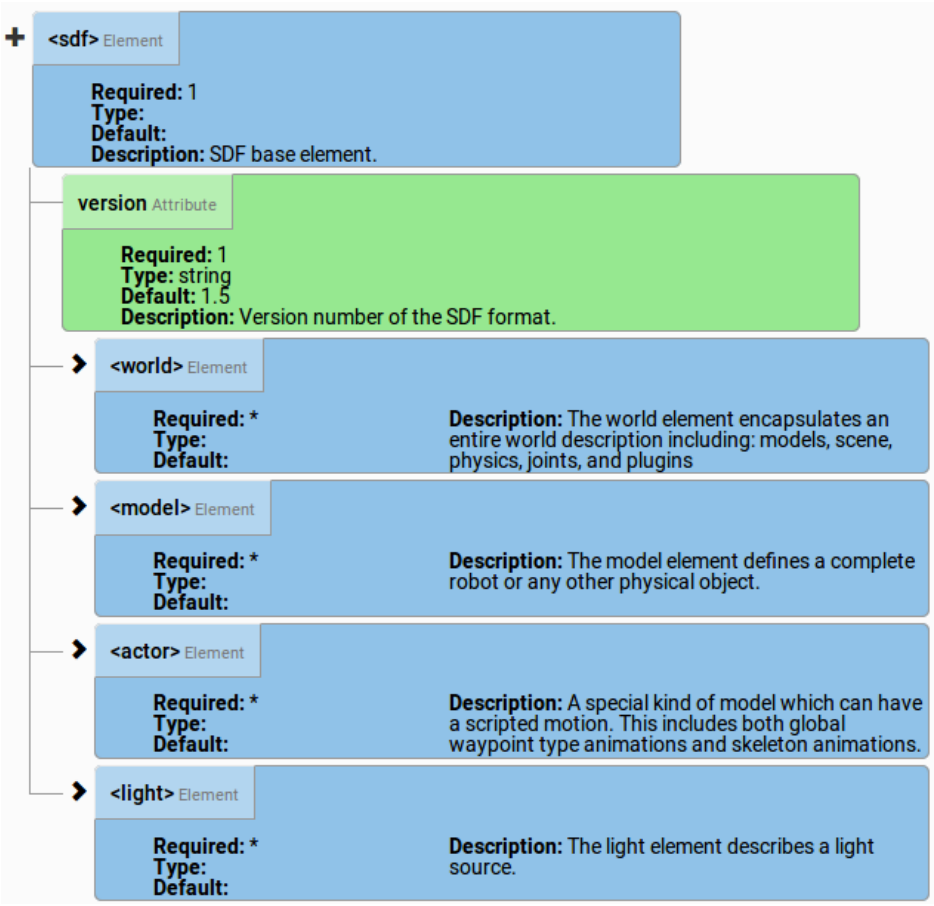
It is possible adding all of the kind of models, like said, actors (people that are walking/running) but at the same time also cars, train etc. For example Zofka et al. [70] created a traffic scenario in Gazebo, considering also the dynamic of the car in terms of internal wheels dynamic and motions.

A Gazebo file is an SDF file that can be a model, a world, an actor or a light for example. The root SDF element is shown in Figure 11.

2.4 Autopilot

The autopilot is the main component of the UAV: it gives the pilot the possibility to drive the UAV easily and to perform several different tasks by adapting its behavior to any circumstance. That component usually consists in a small box that combines all the inputs coming from the instrumentation and the pilot's commands and, thanks to that, it is able to give the correct power to each rotor, making the flight stable and secure. There are different kind of solution regarding the choice of autopilot (firmware), Brunner et al. [71] choose the PX4 [72] instead of Ardupilot [73] for different reasons. Some of them are:

- PX4 was meant for advanced UAV applications. It can be used in hybrid systems, with safety critical algorithms running on a real-time OS that can communicate with ROS [39] running on Linux on a companion computer. The main advantage of using ROS is that the communication between different processes and machines is easily solved.




```
<?xml version='1.0'?>
<sdf version='1.6'>
  <world name='default'>
    ...
  </world>
</sdf>
```

Figure 11: The basic structure of a SDF file and the example for a world tag.

- It offers a more mature software in the loop [40] simulation with built-in support of Gazebo simulator; it can be compiled for POSIX system.

- PX4 features an off-board flight mode where the vehicle obeys a position, velocity or attitude setpoint provided through the MAVLink communication protocol.

Moreover, he shows a system architecture of the simulated autonomous delivery drone. The simulated hardware blocks of the system, as shown in Figure 12, are represented by rounded orange rectangles, and the ROS nodes are represented by the blue rectangles. The different modules of the system communicate with each other through the data structures described by the ellipsoids.

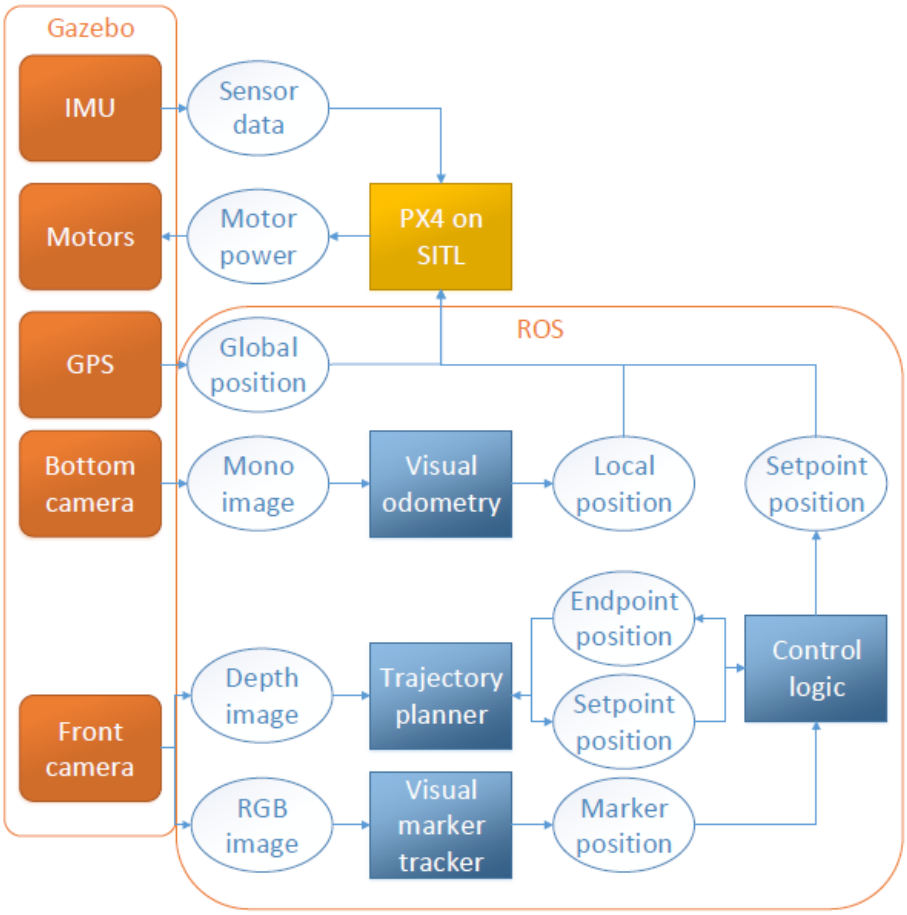


Figure 12: System architecture of Gazebo, SITL and ROS, from [71].

Rolando et al. [74] focus on PIXHAWK autopilot, where PIXHAWK is a high-performance autopilot module suitable for fixed wing, multi rotors, helicopters, cars, boats and any other robotic platform that can move. It is targeted towards high-end research, amateur and industry need and combines the functionality of the PX4FMU + PX4IO. In the same way, he choices PX4 firmware. PX4 firmware only supports the new specification, the joystick, and the virtual joystick.

This autopilot has characterized by different layers, as shown in Figure 13. Every layer has a different function, indeed:

- Apps API: this interface is intended for app developers, e.g. using ROS or Drone API.
- Applications Framework: this is the set of core default applications (nodes) which operate the core flight controls.
- Libraries: this layer contains all system libraries and functionality for the core

vehicle operation.

- Operating System: the last layer provides hardware drivers, networking, UAVCAN, and failsafe systems.

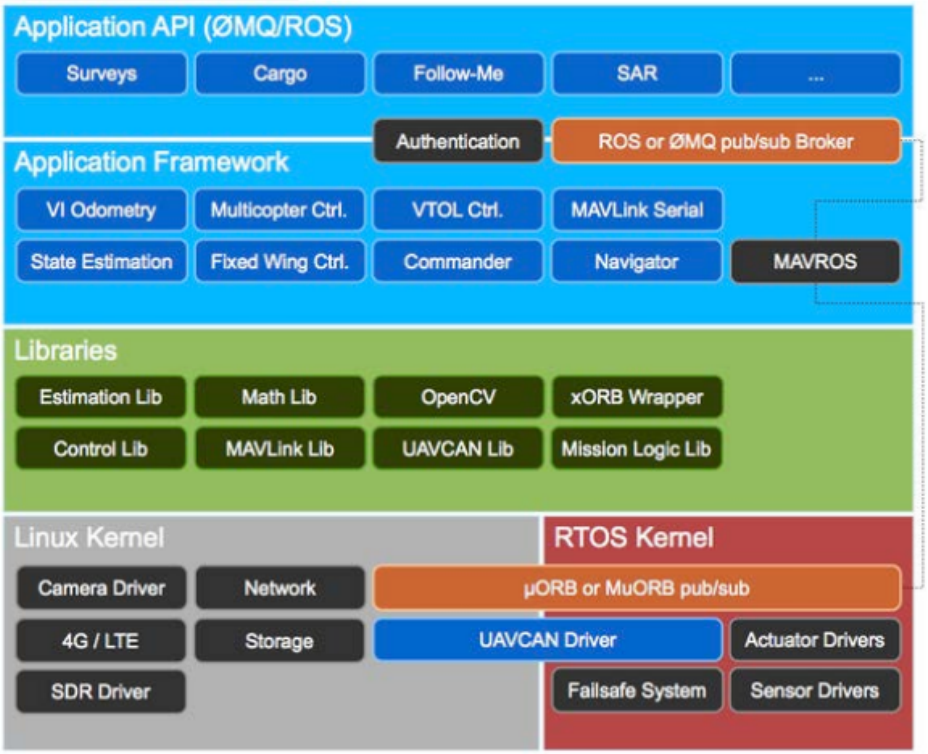


Figure 13: PX4 autopilot layers, from [75].

This autopilot is well compatible with ROS.

2.5 Software in the Loop

Real experiments using UAVs are expensive and risky regarding injuring humans and damaging the UAVs hardware; therefore, the performances of UAV systems should be analyzed before their deployments. For this reason, many researchers utilize SIL simulation. SIL is used as the first instrument in order to validate the software developed, in positive case the next stage is real tests. It is a simulation that gives the opportunity to operate plane, copter or rover, without the need for any hardware. It consists of a build of the autopilot code, using C++ programming language, which serves to run an autopilot on your computer directly for testing, consequently, the simulation is self-contained. In conclusion, SIL gives the possibility to test the C++ code, and the understanding of required improvements the code itself needs.

Nguyen et al. [76] propose an interesting way to work with SIL. For any UAV development, the SIL simulation has been known as a valuable simulation process to verify the designed UAV model and the flight control algorithms installed in the UAV. He said that some researchers build the flight control algorithms based on Matlab/Simulink [77] or PX4 source [78]. The UAV model is built based on X-Plane [79], FlightGear [80], JMAVSim [81], or Gazebo [82]. Among them, he says, PX4 source and Gazebo simulation have more advantages due to the following reasons. Firstly, PX4 [72] is an open source that includes many libraries to drive unmanned aerial and ground vehicles. In addition, PX4 can be uploaded to an open hardware Pixhawk [83] which is very popular hardware for UAV applications. Second, Gazebo simulation is also an open source, and it can provide the dynamic model of UAV, sensor model, and 3D visualization. In particular, this software contains the ODE, which can present a system model robot with high accuracy in real-time conditions [84].

In order to realize a real flight of UAV, the simulation process is always performed to ensure correctness of the control algorithms or the safe functions of real flight UAV. The open source named PX4 allows modifying the code for the control algorithms. The open software Gazebo makes the dynamic model, the sensor model and visualizes the state of the UAV. The ground control station consents to set the desired trajectory for UAV flight. The configuration of the SIL simulation is shown in Figure 14. To connect

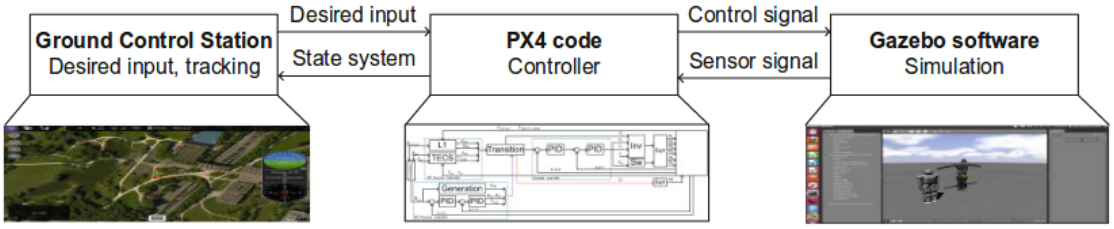


Figure 14: The bridge function of PX4, from [76].

to the SIL, two ports can be used. The port 14550 using the UDP protocol or the port 5760 using the TCP protocol. PX4 uses external developer APIs like DroneCore or ROS (which listen on port 14540), PX4 uses the normal MAVLink module to connect to ground control stations (which listen on port 14550) and uses a simulation-specific module to listen on UDP port 14560, see Figure 15. Simulators connect to this port, then exchange information using the Simulator MAVLink API.

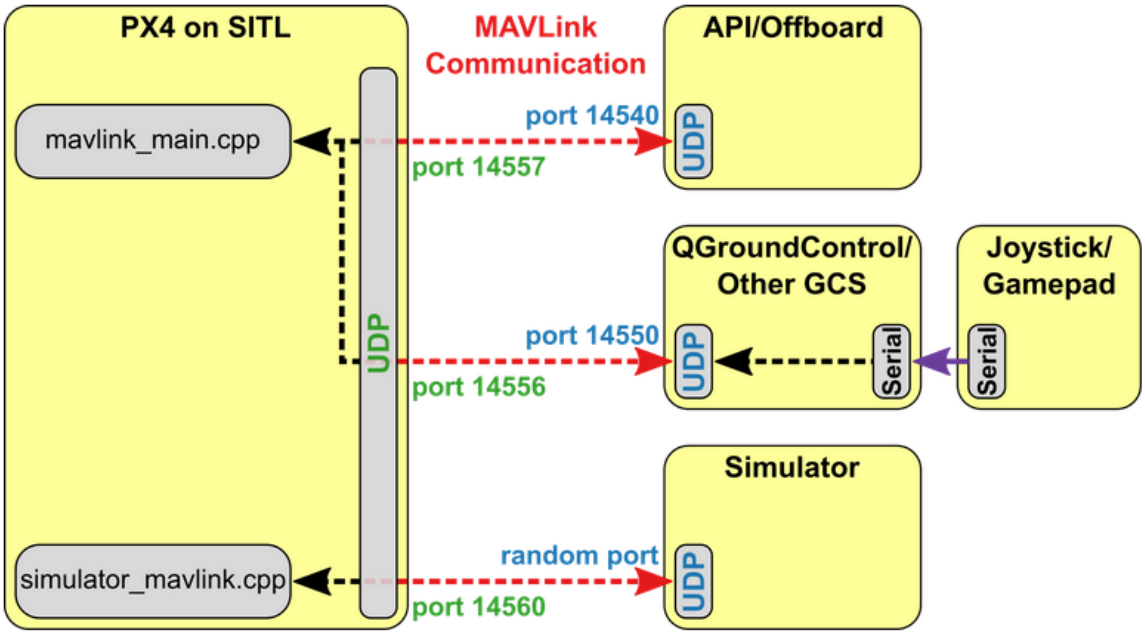


Figure 15: PX4 interconnection diagram, from [72].

2.6 Hardware in the Loop

Hardware in the loop is a simulation mode in which normal firmware is run on real flight controller hardware. This approach has the benefit of testing most of the actual flight code on the real hardware (instead in SIL simulation there is no hardware). In addition, HIL makes possible to verify that the latencies due to the calculation times of the microprocessor have no effect on the control laws. PX4 supports HIL for multicopters (using jMAVSim or Gazebo) and fixed wing (using Gazebo or X-Plane demo/full version). The autopilot is characterized by two independent sub-modules, the Flight Management Unit (FMU) and the Input-Output unit (IO).

Meier et al. [78] have made FMU + IO available as an all-in-one board as well (Pixhawk). The main hardware features are:

- 168 MHz Cortex M4F, 256 KB RAM, 2 MB flash.
- MPU6000 gyro/acc, L3GD20 gyro, LSM303D mag/acc.
- 14 PWM (servo) outputs total (8 with hard override).
- Triple-redundant power supply inputs with failover.
- 5 serial ports (2 with hardware flow control).

- 2 CAN ports, 1 I2C port, 1 SPI port, 3x ADC.
- RC inputs: PPM, S.BUS1/2, DSM2/X, RSSI input.

However, Meier says that the re-usability of these platforms depends on their modularity. Table 6 summarizes aspects relevant to the adaption and general re-usability, including potential limits induced by the license. The BSD license does not limit the reuse in academic and industrial applications, while GPL licensed code is subject to certain restrictions. The nodes column describes whether one software module is self-contained and can be easily exchanged against a different module without modifying the core system (equivalent to a ROS node). The column IPC describes if the system is multi-threaded and offers a suitable generic interprocess communication layer. The column ROS-IF (ROS interface) captures the ROS platform interface. The column ROS-N captures the native ROS support of flight control and guidance software. The license column underlines the license used by Systems.

System	Nodes	IPC	ROS-IF	ROS-N	SIL	Lic.
PX4	yes	yes	yes	yes	yes	BSD
OpenPilot [85]	yes	yes	no	no	no	GPL
APM [73]	no	no	yes	no	yes	GPL
PPZ [66]	no	no	yes	no	yes	GPL
MultiWii [86]	no	no	yes	no	no	GPL

Table 6: Software systems comparison, from [78].

2.7 GCS: Ground Control Station

UAVs are usually controlled by a number of operators (internal pilots) inside one or more ground control stations, depending on the size of the mission. GCS software has several features that will be detailed as follows [7]:

- Mission planning: GCS prepares the mission plans and paths for the UAV, according to the environment and mission requirements, then, UAV has to achieve the mission depending on the planned trajectories (waypoints).
- Navigation and position control: during the mission, UAVs are placed in several positions at different altitudes (waypoints settings), in order to check out the

target area. For that, GCS has to display and control the movements of the UAVs to succeed in the mission.

- Payload control: UAVs are mounted by devices such as cameras, sensors etc. GCSs must supervise the payload parameters during the mission, providing feedback.

- Communication and data exchange: GCS and UAVs should have direct and bi-directional communication between them. The GCS sends commands and orders for the UAV according to the mission and the UAV sends telemetry and data (images, videos, etc.) to the control station. The communication links between the various nodes is a necessary component in the flight systems. In fact, there are two different types of links: UAV-UAV and UAV-GCS, as shown in Figure 16. UAV-UAV communication link ensures the collaboration and the coordination between UAVs to improve the performance of the system.

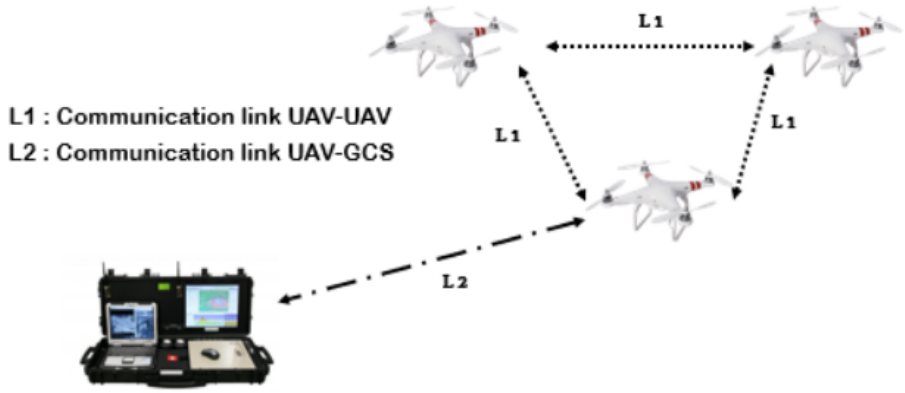


Figure 16: Communication links, from [7].

There is a large variety of GCS software application which runs on a ground-based computer such as QGroundControl, Mission Planner, MAVProxy and UGCS [7], see Table 7.

- QGroundControl: it is an open source ground control station developed by Lorenz Meier and written in C++ using the Qt libraries. This GCS operate on different platforms such as Windows, Mac OS X, Linux, Android, and iO. It supplies configuration for both PX4 Pro and ArduPilot firmware, it supports the MAVlink protocol (which employed to communicate with micro air vehicle) and offers the

opportunity to visualize details of the MAVLink protocol messages, exchanged between it and the flying vehicle. Moreover, QGroundControl proposes a graphical interface, which includes a 2D map, to facilitate the management of one or multiple UAVs and to control the location of UAVs. Additionally, it provides video streaming, displays the vehicle position, waypoints, etc; and offering the possibility to create a mission for autonomous flight.

- Mission Planner: this GCS software is developed by Michael Oborne using Python Programming Language. Unlike QgroundControl which is compatible with all platforms, MP is compatible with Windows only, but there is also a specific version for Windows, Linux and Mac OSX called APM Planner [87]. It provides a graphical interface which displays information about the UAV like GPS status, battery, airspeed, video etc.; additionally, MP allows to download the log files of a mission and examine them.

- Universal Ground Control Software: UGCS is a desktop software with a graphical interface that allows users to create a mission, to supervise UAVs, to be informed about the state of the vehicle and to follow the telemetry. The efficiency of the UGCS is that it supports various autopilots such as DJI, Ardupilot, PX4, Micropilot, Microdrones, and other MAVLink compatible, etc. This product has solid support which contains articles, tutorials, and videos. The free version of UGCS is limited and it is dedicated to beginners for a simple test or a simple mission within line of sight. For the proper functioning of UGCS, it is recommended to operate it in an environment with the following configurations: 2 GB RAM minimum, 2 GB of free space in the hard disk and a processor Core 2 Duo or Athlon X2 at 2.4 GHz.

- MAVProxy: it is an extensible command-line ground station written in Python which can be run on different OS such as Linux, OS X, Windows, and others. MAVProxy manages any UAV supporting the MAVLink protocol by using Micro Air Vehicle Marshaling/Communication library. Furthermore, it provides different modules, like console module that displays information about the UAV's current state and map module that shows the UAV's current position and waypoints. MAVProxy is commonly employed by developers to interact with SITL. It

has the ability to forward messages from a UAV via UDP protocol (like QGroundControl) to others GCS on other devices.

GCS Software	QGroundControl	Mission Planner	UGCS	MAVProxy
Interface	Graphical	Graphical	Graphical	Command
Commercial, Free	Free	Free	Free version with limited capabilities	Free
Support MAVLink	Yes	Yes	Yes	Yes
Platform for Android	Yes	No	Yes	No
Pilot	PX4 Pro, ArduPilot (APM), MAVLink compatible	Ardupilot, PX4	DJI, Parrot, MAVLink compatible, etc.	ArduPilot, MAVLink compatible

Table 7: GCSs, from [7].

From Table 7, QGC appears the best choice. UGCS has limited capabilities, MP is compatible with Windows only (it had not the MAVlink support, but nowadays flight logs are stored on the flight controller's onboard dataflash memory and can be downloaded after a flight through MAVLink [88]) and the MAVProxy is not compatible with the PX4 firmware. Automated mission planning over a swarm of UAVs remains to date a challenging research trend in regards to this particular type of aircraft (currently the mission planning can be made for a fleet of UAVs). This problem involves generating tactical goals, commanding vehicles, risk avoidance, coordination and timing.

Ramirez et al. [89] use QGC by adding a mission designer that permits the operator to build complex missions with tasks and other scenario items; an interface for automated mission planning and replanning for UAVs path planning. It works as a

test bed for different algorithms and a Decision Support System (DSS) that helps the operator in the selection of the plan.

Another important value of the GCS software choice is that QGC provides full flight control and vehicle setup for PX4 or ArduPilot powered vehicles. It provides easy and straightforward usage for beginners, while still delivering high end feature support for experienced users.

2.7.1 Video Streaming

PX4 on SIL for Gazebo supports UDP video streaming from a Gazebo camera sensor attached to a vehicle model. It is possible to connect this stream from QGroundControl (on UDP port 5600) and view the video of the Gazebo environment from the simulated vehicle, just as you would from a real camera. The video is streamed using a GStreamer pipeline [90]. Similarly, it is possible having the same video in Gazebo. Video streaming can be enabled/disabled using the Gazebo UI Video ON/OFF button, see Figure 17. So this means that it is possible to have a window in which the first person view will

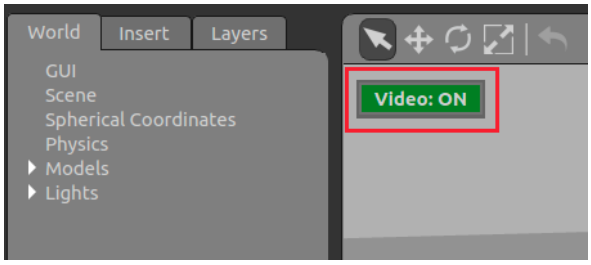


Figure 17: Video ON/OFF button in Gazebo, from [91].

appear. Another option is to open the video monitor going in the “window” option, “topics visualization”, “image stamped”.

2.8 Communication Protocol

Three levels of communication exist:

- Inner firmware communication between MAVLink and uORB.
- Communication between autopilot and GCS: MAVLink.
- Communication between autopilot and ROS middleware: MAVLink and MAVROS.

The uORB is an asynchronous publish/subscribe messaging API. It is used for inter-thread/inter-process communication. MAVLink is one of the most commonly used communication protocol when it comes to micro air vehicles. Many commercialized UAVs are already using the MAVLink protocol. It is both used for communication between GCS and unmanned vehicles and in the inter-communication of the subsystem within the vehicle. It works via USB or telemetry (not both at the same time: in that case, USB has the priority), and allows to send a common set of messages, used for transmitting controls, parameters to be set, and a lot of information like orientation, GPS location, speed etc. MAVROS is one of the packages available within the ROS community. This one, in particular, is aimed to provide a link between MAVLink and ROS, acting as a bidirectional bridge between these two languages, translating MAVLink messages into ROS messages and vice versa. It creates, in fact, two MAVROS nodes, one for publishing MAVLink stream from autopilot, one for subscribing ROS messages to be sent to the autopilot. It is, then, a package that can be included in a ROS application providing communication drivers for various autopilots with MAVLink communication protocol. It provides a sub-group of the standard MAVLink messages and commands and can be extended by using plug-ins. A complete description of the package and the set of messages are provided at [92].

2.9 Robotic Operating System: ROS

ROS is the Robot Operating System [39]. ROS is an open-source, meta-operating system for robots. It provides the same services from an operating system, including hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. It also provides tools and libraries for obtaining, building, writing and running code across multiple computers.

The supporting period of ROS is different for each version, but generally, two years of support is available after its release. Every two years, Robotic Operating System and Linux release Long Term Support (LTS) version and ROS is supported for the next five years. For instance, the Kinetic Kame Xenial Xerus version (used for this thesis), which is supported by Ubuntu 16.04 LTS (used as well), will be supported until April 2021.

ROS is:

- Distributed: individual programs can be run on multiple computers and communicate over the network.
- Peer to peer: individual programs communicate over defined API.
- Multilingual: individual programs can be written in C++, Python, Java, etc.
- Free and open source.
- Everything is modular.
- Huge and active community.

ROS is the Cloud Robotic Platform (CRP), see Figure 18.

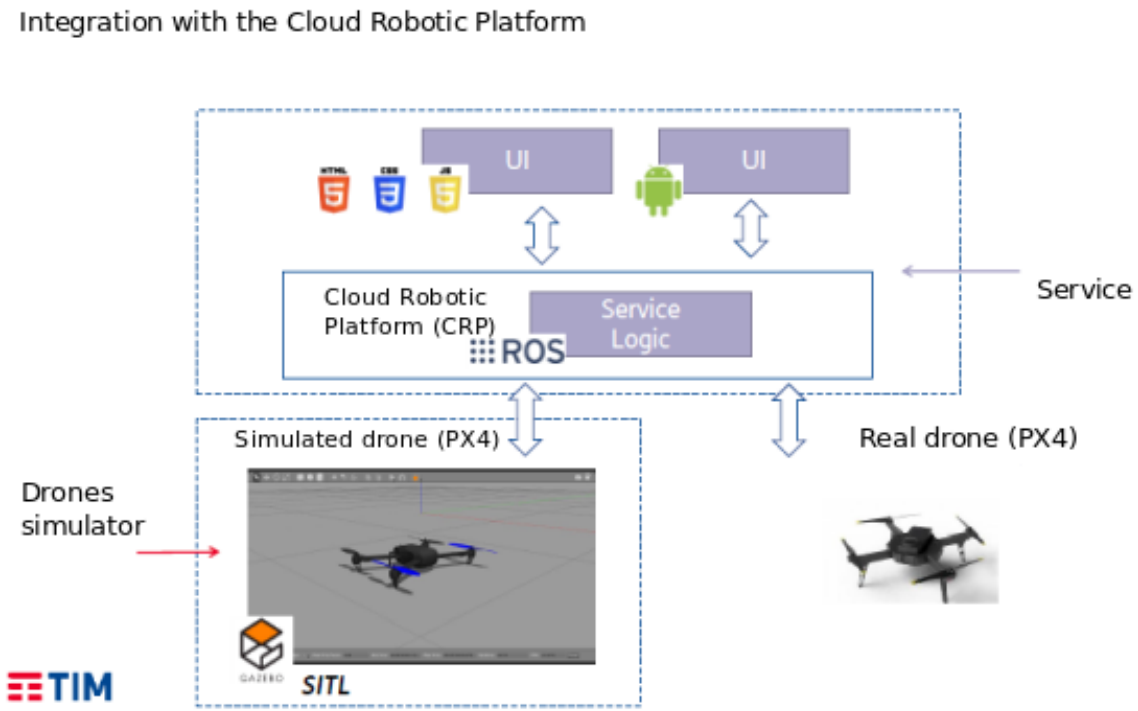


Figure 18: Sketch made by Telecom Italia (TIM).

2.9.1 Tracking targets

UAV design, path planning, and remote sensing are active research fields. The topic of automated vision-based target tracking and following has also progressed to a certain extent, as it plays an essential role in automating the tasks performed by UAVs

such as search and rescue, environmental monitoring, and infrastructure inspection [93]. Interesting work has been done by Hines et al [93]. He made a tracking targets simulation in which a UAV is flying and hovering on targets into a Gazebo world. The simulation setup uses Gazebo robotic simulator in combination with the PX4 autopilot software. Gazebo simulator performs physics simulation and supplies sensor values to the PX4 autopilot stack via specialized MAVLink messages. In return, the PX4 flight stack sends motor and actuator control values to Gazebo. The framework is developed using ROS. An example of UAV on a target shown in Figure 19.

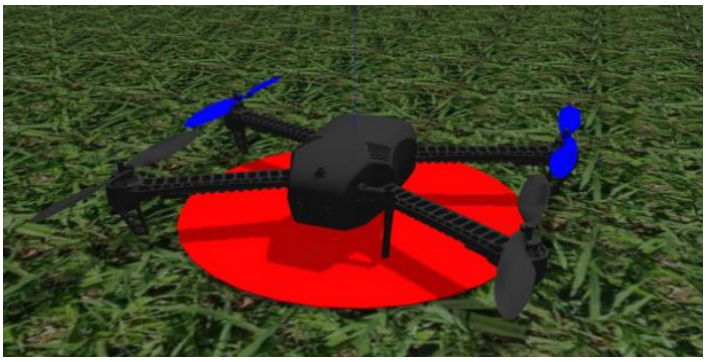


Figure 19: Simulated 3DR Iris multirotor and a target, from [93]

2.9.2 Model Status Monitoring: Inside the UAV with ROS

It is important to gain a robot point of view visualizations. Using such visualization achieves a better understanding of what the robot "sees", drawing a representation of the data in a simulated environment. Furthermore, a detailed list of feedbacks is very crucial in order to monitor the status of the UAV.

Campusano et al. [94] explain ROS already comes with a standard 3D visualization environment: RVIZ [95]. RVIZ visualizes sensor data and state information from the robot, combining the data and visualizing it all into a single image that overlays the data on a simulated environment. Moreover, it can be used with the software of computer vision. In addition to RVIZ, ROS provides other packages of visualization and data plots: rqt plot [96] and rqt graph [97]. The former shows a scrolling time plot of data of a robot. It is a readily available plotting application. This tool allows viewing a time plot of arbitrary numeric values that are extracted from ROS messages. In the same direction also Florea [98] works, using RVIZ and rqt plot. The latter package

helps in visualizing the ROS computational graph. Both Javed et al. [99] and Feng et al. [100] use RVIZ, rqt plot and rqt graph.

3 Problem Statement and Definition

The objective of the proposed work is to provide the foundations of a simulation framework to evaluate UAV utilization to monitor activities in an urban environment (city) with minimum interaction with, and intervention of human activities. In addition, the goal is considering collision avoidance among UAVs and urban structures or buildings.

Safety, surveillance and traffic monitoring [38] are crucial elements of urban monitoring. So the aim is to create a simulated environment characterized by a 3D world model in which animation like walking people, running people, cars, trams, etc., will be inserted to create a realistic scenario. In parallel, a UAV will be placed in this world and it monitors the area autonomously with path planning uploaded through the GCS software or directly piloted. The macro software problems to be faced are two:

- Creating a realistic 3D world with Gazebo. The thesis goal is to realize a simulated environment in which different types of UAVs will monitor and survey an urban district with a minimum or no interface with humans. Stated differently the UAVs will act as the eye-in-the-sky to safeguard humans and detect or predict any kind of anomaly.

- Simulating an urban monitoring mission: load waypoints from GCS to the autopilot, fly a UAV in SIL [40] monitoring its status with ROS [39]. It also records and collects images and shows the first person view perspective from a dedicated monitor.

It will be assumed that a UAV is available and its mathematical models are known [101] (kinematics, dynamics, equations of motion, etc.). Each UAV will be equipped with a set of sensors to facilitate sensor-based navigation and control. Then, the thesis objectives will be path planning for each UAV, collision avoidance with any urban structure (i.e., buildings, trees, etc.), navigation in an urban environment for monitoring activities, communication between UAV and GCS, and safety and reliability determination. Figure 20 shows the main three types of drones; this study will focus on multicopter UAV only. Regarding sensors, for urban monitoring purposes, camera sensors will be considered. First Person View and Pan and Tilt configuration for scanning and collecting urban images will be tested.

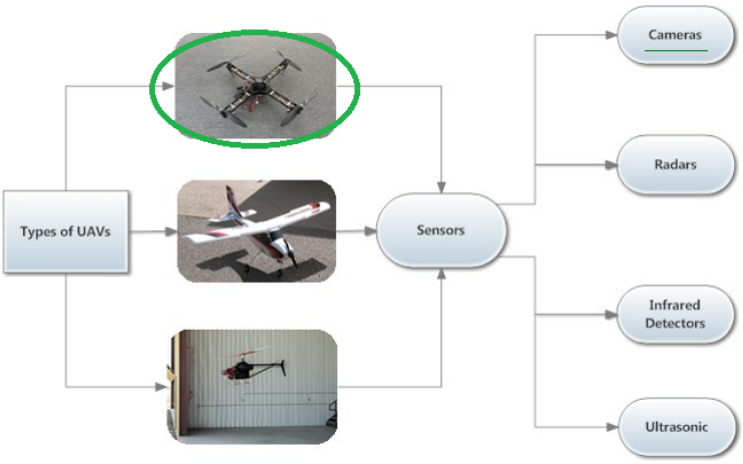


Figure 20: Sketch for urban monitoring, quadcopter and camera sensors used, from [38].

3.1 Tools

Demonstrations (performed in the case studies) will be accomplished using a set of available tools and packages (ROS, Gazebo, SIL, PX4 package and QGroundControl for the first and main case study and the package Sphinx-Parrot will be added and tested with Gazebo and SIL, for the second test). Following an overview of the combined usage of the main tools.

3.2 ROS with SIL in Gazebo

The SIL simulation can run in Gazebo with or without the use of ROS. There are many advantages in using Gazebo with the middleware ROS. To achieve ROS integration with stand-alone Gazebo, a set of ROS packages named *gazebo_ros_pkgs* provides wrappers around the stand-alone Gazebo, see Figure 21. They provide the necessary interfaces to simulate a robot in Gazebo using ROS messages, services and dynamic reconfigure [102]. The set of packages:

- Supports a stand-alone system dependency of Gazebo, that has no ROS bindings on its own.
- Builds with catkin (workspace).
- Treats URDF and SDF as equally as possible.

- Integrates real-time controller efficiency improvements from the DARPA Robotics Challenge.
- Cleans up the old code from previous versions of ROS and Gazebo.

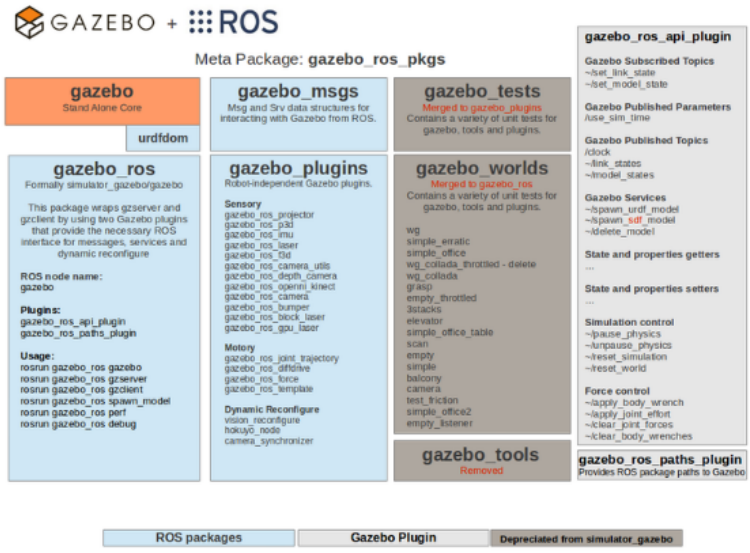


Figure 21: Gazebo and ROS packages.

4 Proposed Solution

First of all, the purpose of this research is to perform a 3D extrusion and therefore implement the 3D city model in Gazebo. It requires a detailed degree of rendering so advanced textures can be added. The 3D simulated environment has to be converted in the right format in order to be uploaded in the physics software simulator. Below is just two basic scripts to show the structure of Gazebo models, see Figures 22, 23.

Each model must have a model.config file in the model's root directory that contains meta information about the model, namely the name, version of the model, SDF tag etc. These are the crucial tags, but it is possible to add tag regarding the author's information and a description of the model itself.

The model.sdf file is an XML format that describes objects and environments for robot simulators, visualization, and control. Originally developed as part of the Gazebo robot simulator, SDF was designed with scientific robot applications in mind. Over the years, SDF has become a stable, robust, and extensible format capable of describing all aspects of robots, static and dynamic objects, lighting, terrain, and even physics [103]. In Figure 23 there are different tags, among them, the most significant is the collision-related tag, it activates the collision.

```
<?xml version="1.0"?>
<model>
  <name>My Model</name>
  <version>1.0</version>
  <sdf version="1.5">model.sdf</sdf>
  <description>
    My model.
  </description>
</model>
```

Figure 22: File.config.

This is a conceptual and general description depicting some research to achieve the 3D city model from a lot of codes and open source tools. At this point, OSM, JOSM OSM2World, Blender, Gazebo are the tools that mostly have helped this work [47].

The operating system is Ubuntu 16.04 LTS (Xenial Xerus). Up to this point, the Gazebo world file is characterized just by the 3D city model (the district).

```
<?xml version="1.0" ?>
<sdf version="1.5">
  <model name="My Model">
    <static>true</static>
    <link name="link">
      <collision name="collision">
        <geometry>
          <mesh>
            <uri>model://my_model/meshes/my_model.dae</uri>
          </mesh>
        </geometry>
      </collision>
      <visual name="visual">
        <geometry>
          <mesh>
            <uri>model://my_model/meshes/my_model.dae</uri>
          </mesh>
        </geometry>
      </visual>
    </link>
  </model>
</sdf>
```

Figure 23: File.sdf.

4.1 District to be monitored

This paragraph has the aim to show and describe the district of interest in order to compare it with the 3D model created. The area that will be monitored is a quarter of Turin, Italy. It is the "Politecnico di Torino". This area is characterized by the Polytechnic University of Turin, the high school Galileo Ferraris, parking places, a lot of bars, homes and the famous locomotive in front of the complex of Officine Grandi Riparazioni. It is possible to include in this university area also some other buildings and the cited before Officine; moreover, also the Intesa Sanpaolo skyscraper has been included. Furthermore, there is the presence of some green areas, trees etc. so in the 3D model, it is forecasted to see some vegetation. Practically, it is impossible to delimit a district perfectly but the idea is that the urban monitoring will be done on the Polytechnic and bordering places. Considering also that the University is divided into two big blocks by the street called Corso Castelfidardo, so the neighboring places to be considered are more. There are different roads in this area, but the most important streets that mark the current district are:

- Corso Vittorio Emanuele.
- Corso Duca Degli Abruzzi.

- Corso Luigi Einaudi.
- Via Pier Carlo Boggio that converges in Via Paolo Borsellino.

The skyscraper is outside of the area respect the four streets listed above. A view of the district is presented in Figure 24.

Figure 24: Polytechnic of Turin area view, from [104].

In addition, it is important to underline that for the extrusion it has been used OpenStreetMap and actually it is not feasible to make a specific manual selection of the area, the selection is a rectangular 2D area because OSM has limits in the selection mode, for example, there is no a South-North orientation option. So, it has been utilized an OpenStreetMap editor called JOSM [42], thanks to it, it is possible to remove streets, building, and areas that are not needed in the 3D model. It has been edited the original map from OpenStreetMap (OSM format) and after that, exported the file `mappa_torino.osm` (still OSM format). Following the steps to obtain the file.dae:

- Export a 2D map from OSM, with the option "manually selected area".
- Modify this map on the OSM editor called JOSM, then save like file.osm. There is also another approach that gives the possibility to skip this step; exporting a bigger area of the city and later on the GCS software uploading waypoints just where the urban monitoring is required.
- Convert the file from OSM to OBJ. Then import it in Blender like file.obj and next export like file.dae in order to use it in Gazebo.
- Upload it in Gazebo that works with DAE files.

The previous four steps are linked to the following four Figures 25, 26, 27, 28.

Figure 25: Openstreetmap (2D model).

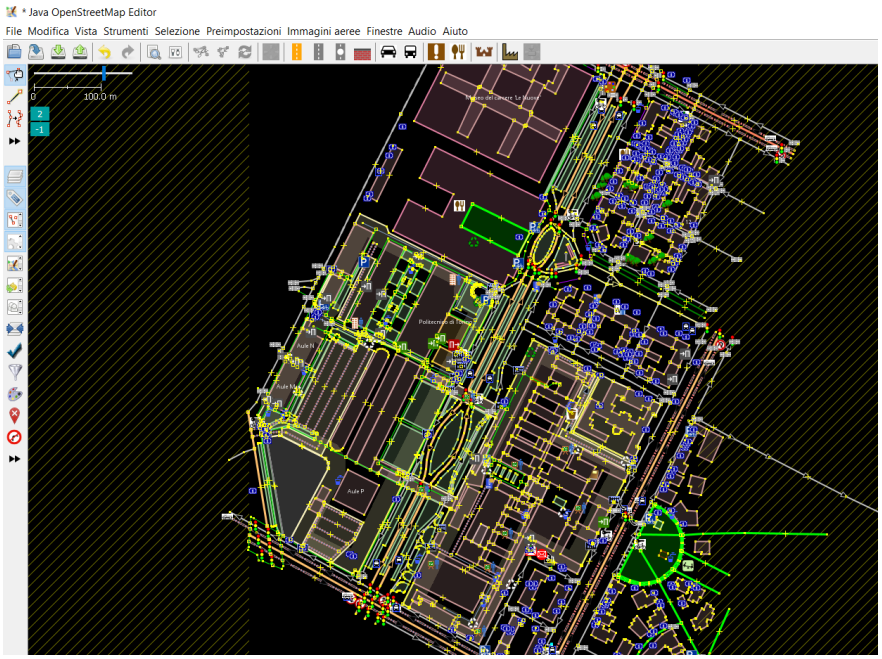


Figure 26: Josm editor (2D edited model).

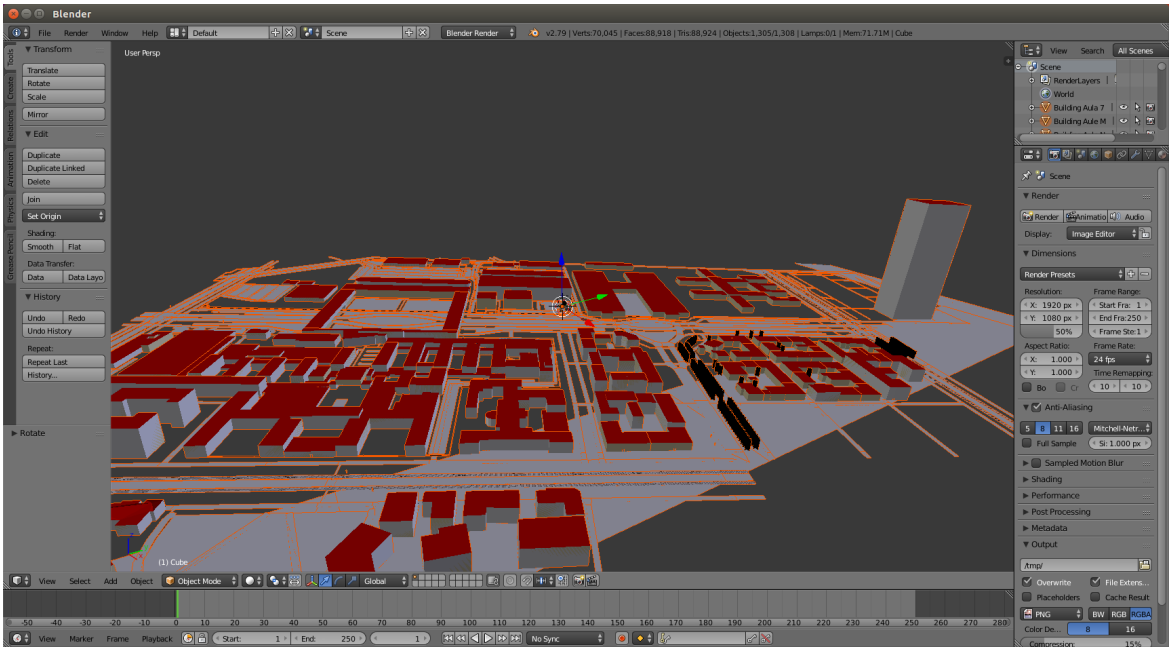


Figure 27: Blender (3D model converted from Wavefront format to Collada).

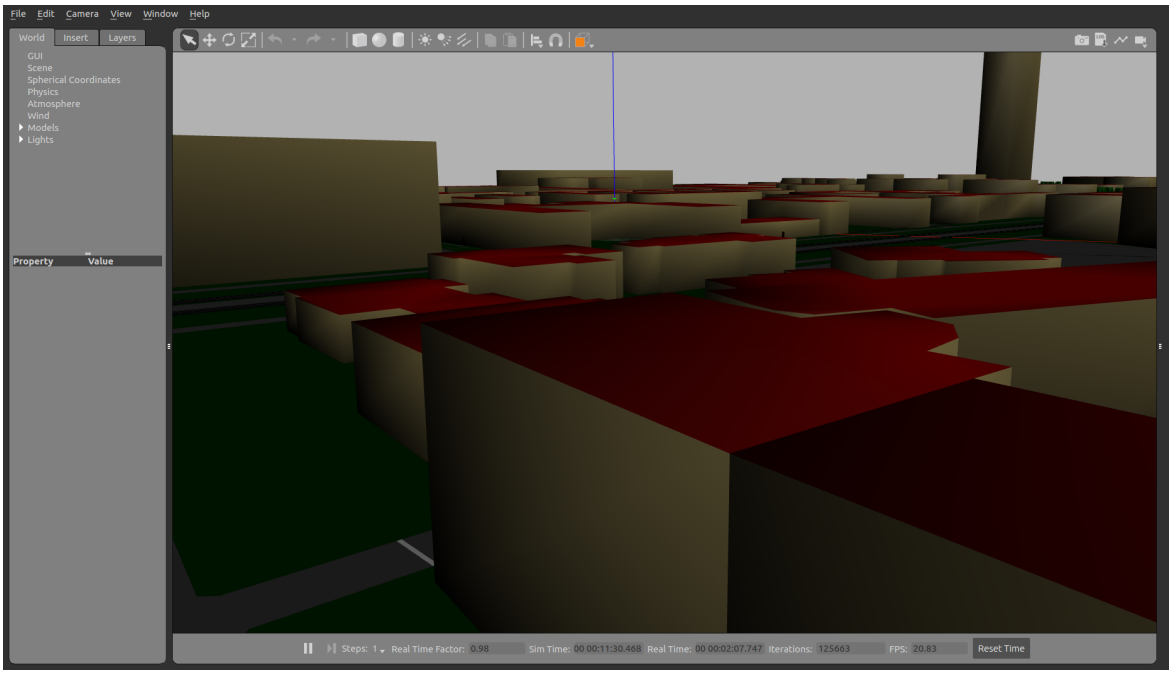


Figure 28: Gazebo (3D model, Collada format).

Overall, making a comparison between the real district and the simulated one, it is possible to state that the height (in meters) of the buildings is matching with the reality (buildings of the city model have been measured in Blender, and real ones have been manually measured for comparison). Although the Gazebo city model is an accurate representation, there are some elements missing in it, as for example

electrical lines, monuments, benches, flowerbeds, trees, etc. This is due to some limits of the OpenStreeMap tool, in it, all of the maps are constantly being upgraded and improved (in order to help the VGI community editing and adding elements in those maps an OSM account is mandatory), therefore there is still something missing in them. However, the model degree of detail is fair enough so tests and experiments can be accurately conducted and related to actual tests in the correspondent real area/district.

In conclusion, regarding Gazebo and the model framework, there is the option to test it during the night and with daily sunlight (adding light elements).

4.2 Textures

After having converted the 2D OSM file in the 3D OBJ file (Wavefront format) with the OSM2World tool [41], the next stage is importing the model in Blender where it is possible to export it with another format, Collada. The Collada file is a DAE file that can be uploaded in Gazebo. In Blender exporting the model without adding textures will give like output the default textures, as shown in Figure 29. Adding the

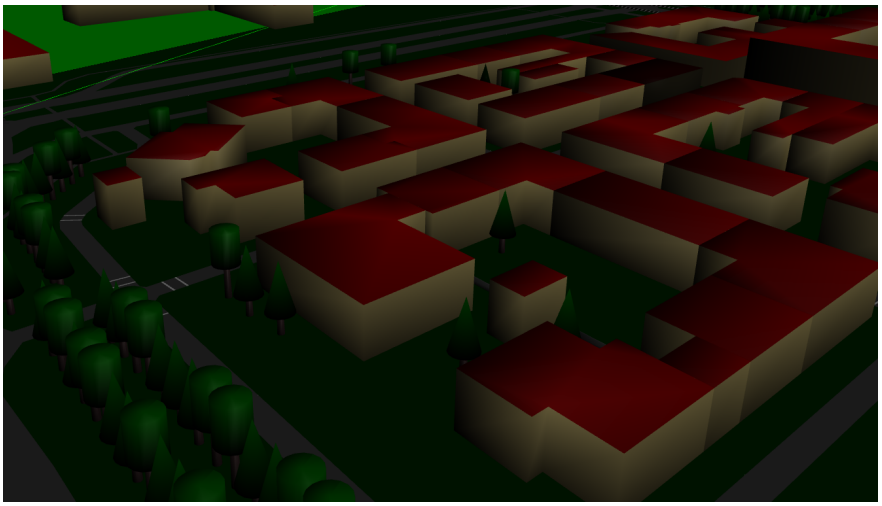


Figure 29: OSM2World default textures.

textures gives more realistic rendering, always in the limits of the OSM2World tool textures, see Figure 30. This means that OSM2World offers some textures that are PNG files and they are called in the DAE file for the rendering in Gazebo. There is a quality difference regarding the rendering respect to default textures but it is just about graphics output, see Figure 31.

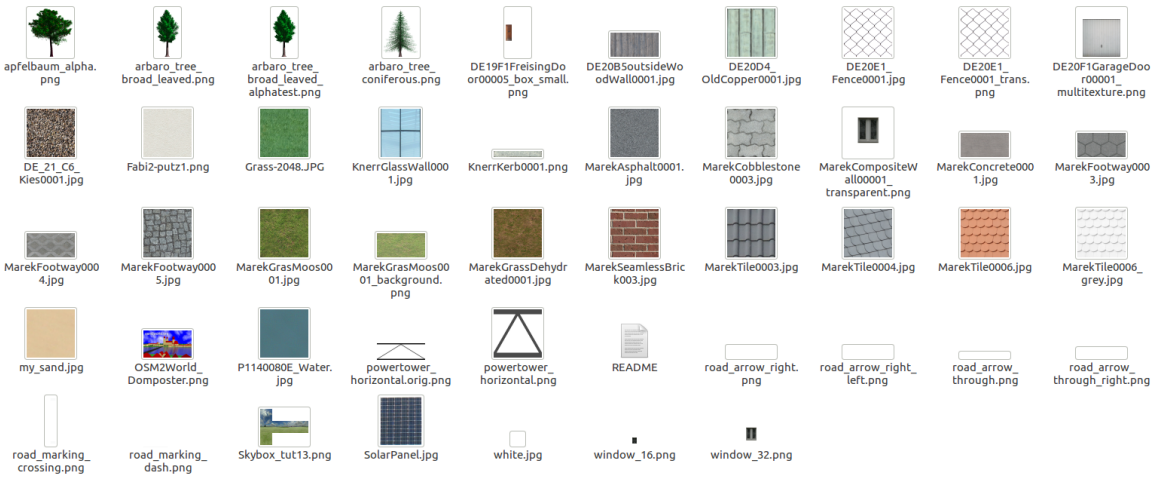


Figure 30: Textures offerd by OSM2World.

4.3 Animated Gazebo models

At this point, the 3D city model has to be filled by Gazebo's "actors" and models in order to create scripted animations and have like the result a realistic urban scenario to be monitored by the UAV. Animations are useful for having entities following predefined paths (loop) in the simulation without being affected by the physics engine. This means they will not fall due to gravity or collide with other objects, for example. They will, however, have a 3D visualization which can be seen by RGB [105] cameras, and 3D meshes which can be detected by GPU [106] based depth sensors, see Figure 32. In conclusion, actors are always static (i.e. no forces are applied to them, contact or anything else) [107]. Although these models are static, they have a 3D visualization and this is crucial in terms of urban monitoring simulation.

Figure 33 demonstrates that in the simulated world the UAV with its camera can collect the static model visualization and this is shown by the FPV screen, namely the second window in Gazebo.

A world file is characterized by many SDF files, the city 3D map is one of them. The final result will be having a world file characterized by the union of all of SDF files (UAV, models).

4.4 Ground Control Station: QGroundControl

Running Gazebo after QGC will impact on the vehicle connection indeed the GCS software will wait until Gazebo is open in order to connect with the simulated UAV (see

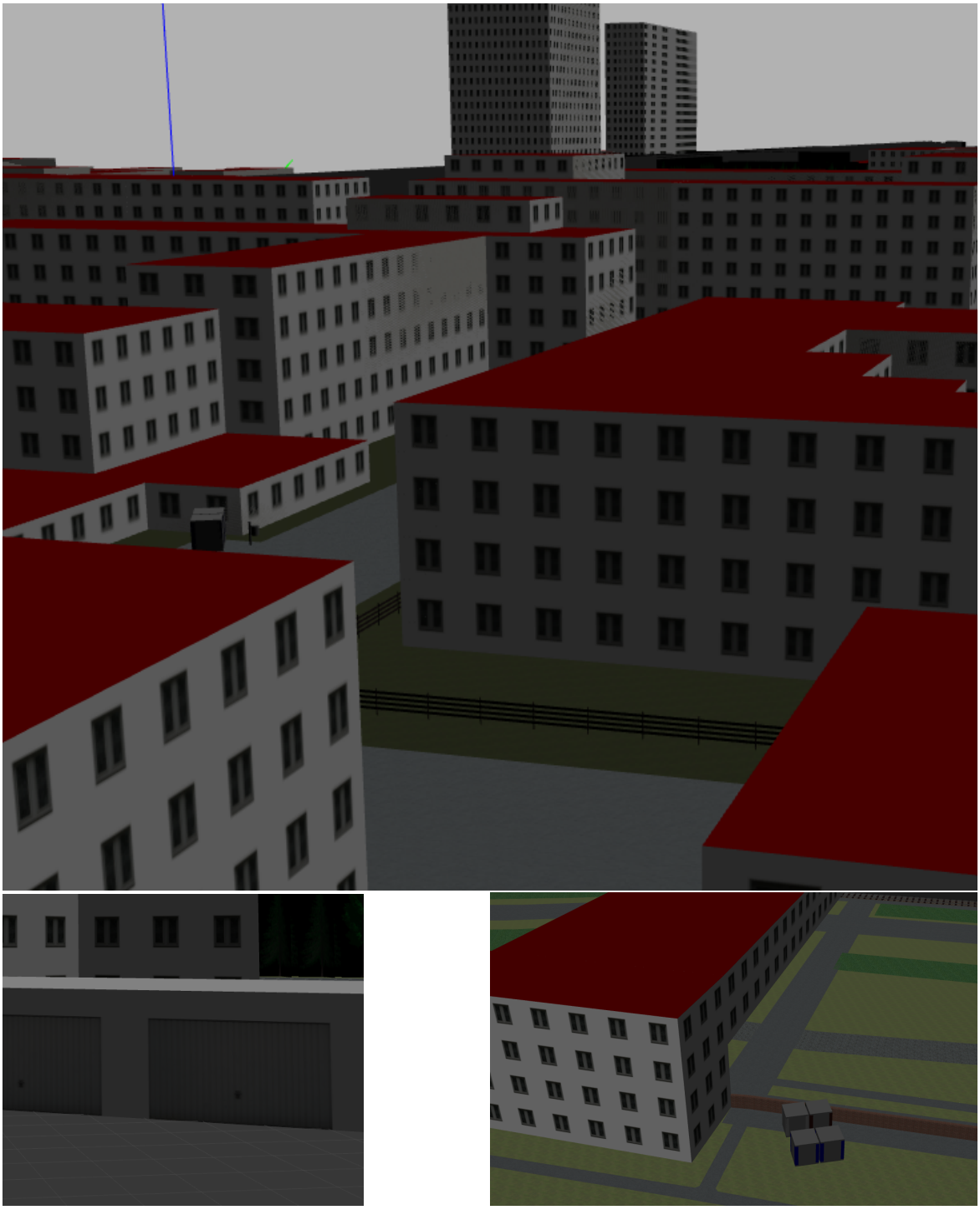


Figure 31: OSM2World textures.

"Waiting for Vehicle Connection" in Figure 34); if Gazebo is already open when QGC is run, therefore, will connect instantaneously with it. It is crucial to set the coordinates (latitude, longitude, altitude) in the ground control station software carefully. The real aim is setting the start mission point in the same place where the UAV is in the simulated environment in Gazebo before running the simulation. The carefulness in

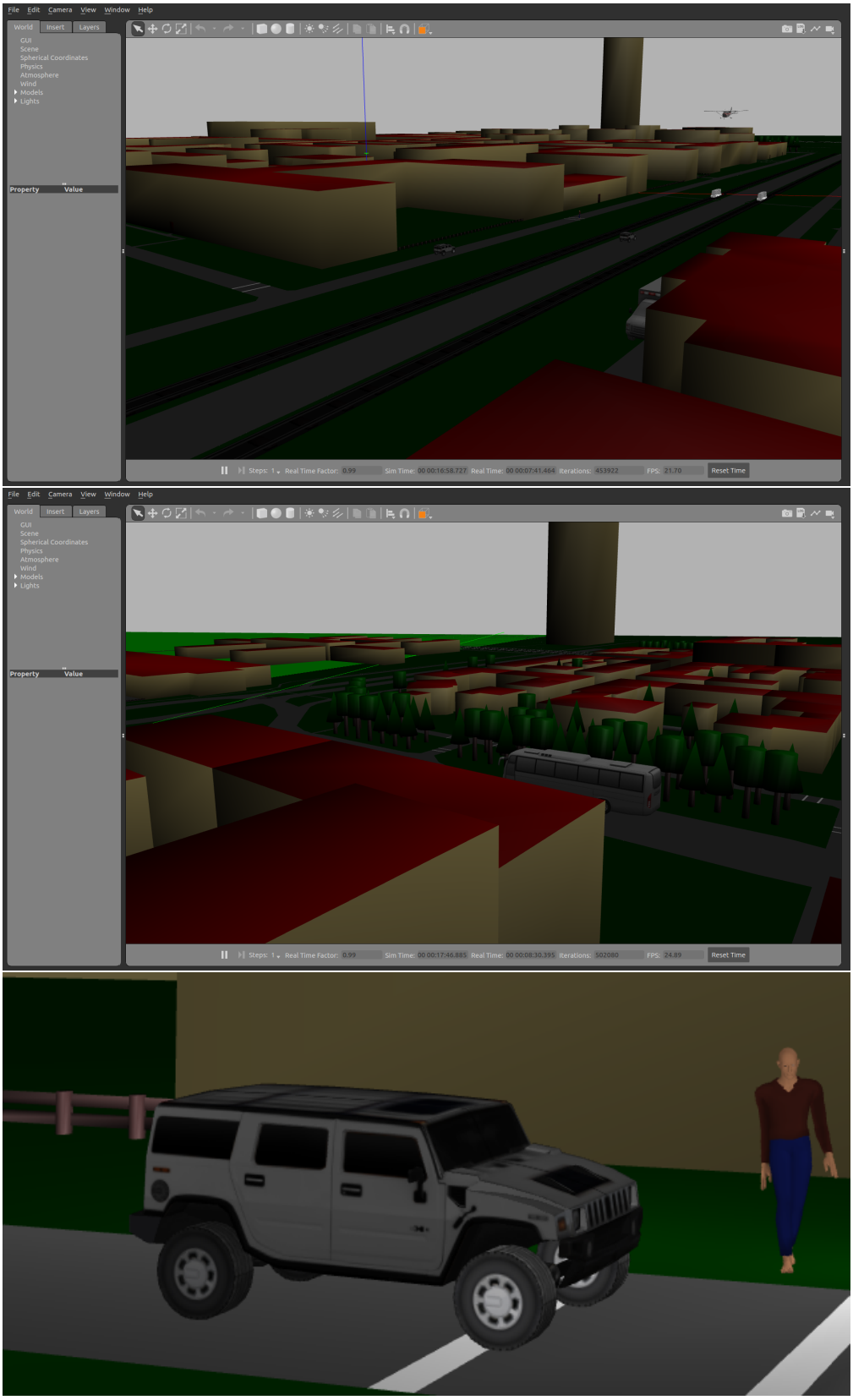


Figure 32: World with models.

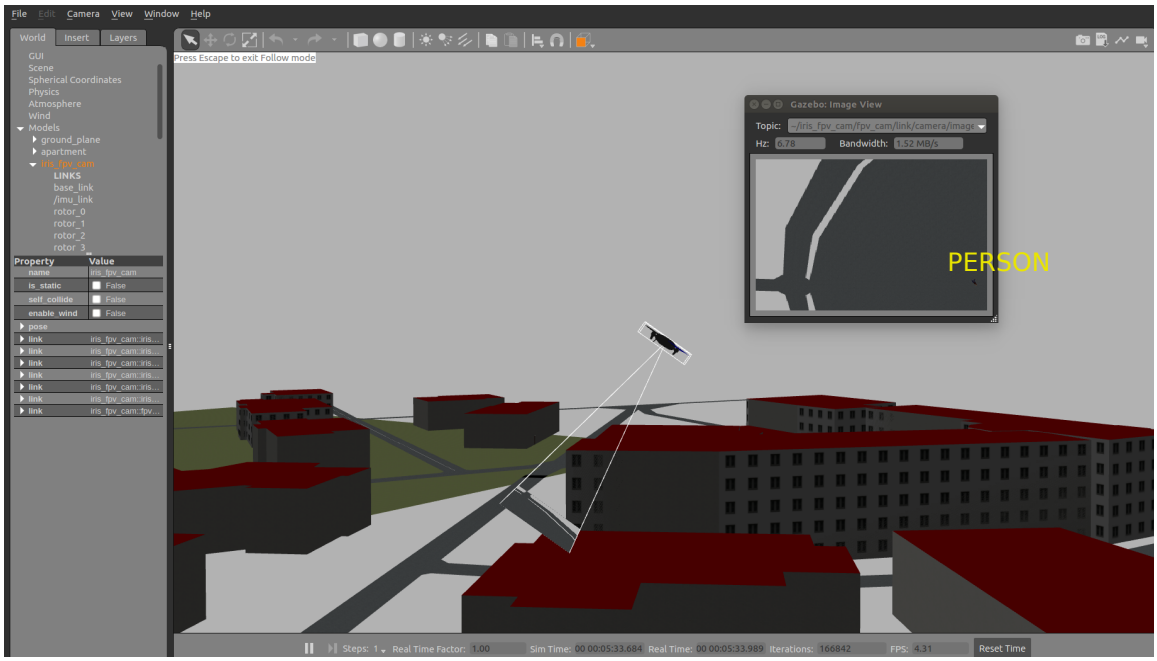


Figure 33: The FPV of the UAV monitoring the behavior of a person in the square.

Figure 34: QGC, Polytechnic of Turin area, waiting for vehicle connection.

setting the coordinates should be on which block of the Polytechnic of Turin, or the Piazzale Duca d'Aosta the UAV is, before its takeoff, for example. It is workable to add all of the waypoints needed for the mission, and for every waypoint, it is asked to insert the altitude, the flight speed and also there is the possibility to make the UAV hovering for some seconds, see Figure 35. Other options are available regarding the camera output window that is quite the same of Gazebo dedicated window (video

Figure 35: Waypoints settings in QGroundControl.

streaming). After that, the next step is to upload all of the inputs inserted, using the "Upload Required" button. After having uploaded the waypoints and set all of the inputs required for the current mission, it is asked to start the task with the "Slide to confirm" button. The UAV will fly in Gazebo world and it is thinkable to continue to command the UAV from the GCS or leave it following the path. On the GCS monitor, it is possible to see an arrow that will follow the UAV during its mission while it is flying into the simulated world. The "Slide to confirm" button will start the mission with the takeoff from the Planned Home waypoint (the first waypoint). The choice of the Planned Home waypoint is vital because in case of a collision with a wall of a building, for example, the UAV will lose the connection with the GCS tool, so, there is the option to make it fly automatically back to its first waypoint and re-start the mission or modify the old path in a proper way. The Fly View displays the actual home position set by the vehicle firmware when it arms (where the vehicle will return in Return/RTL mode). It is possible also going back to the Planned Home waypoint with the RTL mode, after the conclusion of the whole mission.

Another interesting GCS feature is "Goto here" button that allows detaching the UAV from its original path to another location while the simulation is running. Therefore it is possible to add a waypoint during the simulation and interrupt the current

mission going somewhere else. After this deviation, it is plausible to continue the previous mission with the "Continue Mission" button, as shown in Figure 36.

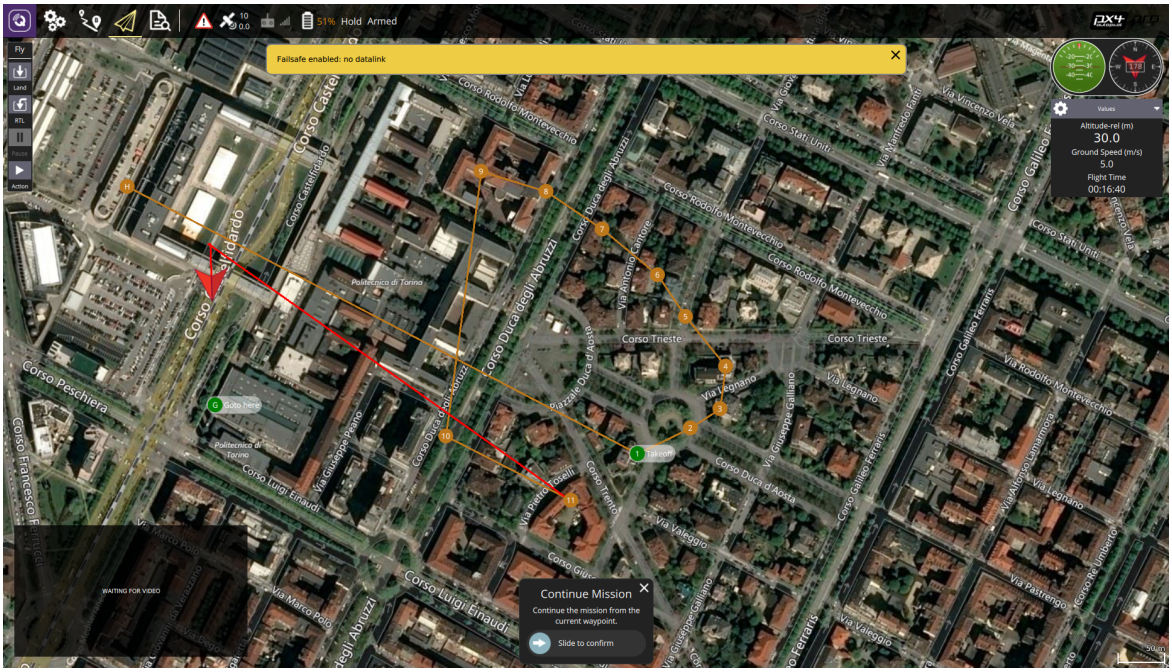


Figure 36: The UAV after its mission is going back to the Home waypoint with the RTL mode, but there is a deviation through "Goto here" option.

5 Results and Case Studies

In both case studies, the 3D model of the Polytechnic district will be used.

5.1 Case study 1: Simulated Autonomous Urban Monitoring

The case study 1 is a simulation of urban monitoring of the cited Polytechnic of Turin quarter. This simulation will be conducted with the autopilot PX4 (firmware) on the SIL simulation and with the physics software simulator Gazebo, in which the world file is determined by the 3D city file (SDF), the animated model files (SDFs) and the UAV (main world file containing all of the SDF files). It will be assumed that a UAV is available and its mathematical models are known (kinematics, dynamics, equations of motion, etc.). In the specific, the current PX4 Gazebo SIL model uses notation that is different from the standard propeller model notation, indeed, for example, the thrust is calculated like: real motor velocity^2 * motor constant. For more information see [101].

The UAV is able to communicate with the GCS software and therefore it is controllable by this tool (QGroundControl), the animated models are defined by script in which there are waypoints and timepoints for their loop path and finally the map of the area that is not static like the actors (Gazebo animated models) but it has the collision mode active. The UAV has the collision mode active too, same as the map. The collision of the UAV occurs when the cube or parallelepiped that circumscribes it collides a building surface [107]. In addition, it is possible to add a different kind of camera sensors on the UAV. It has been tested the Pan and Tilt camera (on the Typhoon model) and the FPV camera (on the Iris model). Overall, this case study is characterized by the action of urban monitoring conduct through the utilization of the GCS software (simulation of the GCS operator work). The package used is PX4 (3DR UAV models).

5.1.1 Pan and Tilt

The Pan and Tilt (yaw-pitch) camera has been tested on the Typhoon model (six rotors). When the UAV was flying there were some problems related to stability (oscillations), so the UAV was oscillating, the path planned was not followed with fidelity (see in Figure 37 that the orange line, namely the path planned, is not very well fol-

lowed by the red line that is the real path). To name a dramatic consequence, the streaming video of the camera was shaking. This could be due to the Pan and Tilt

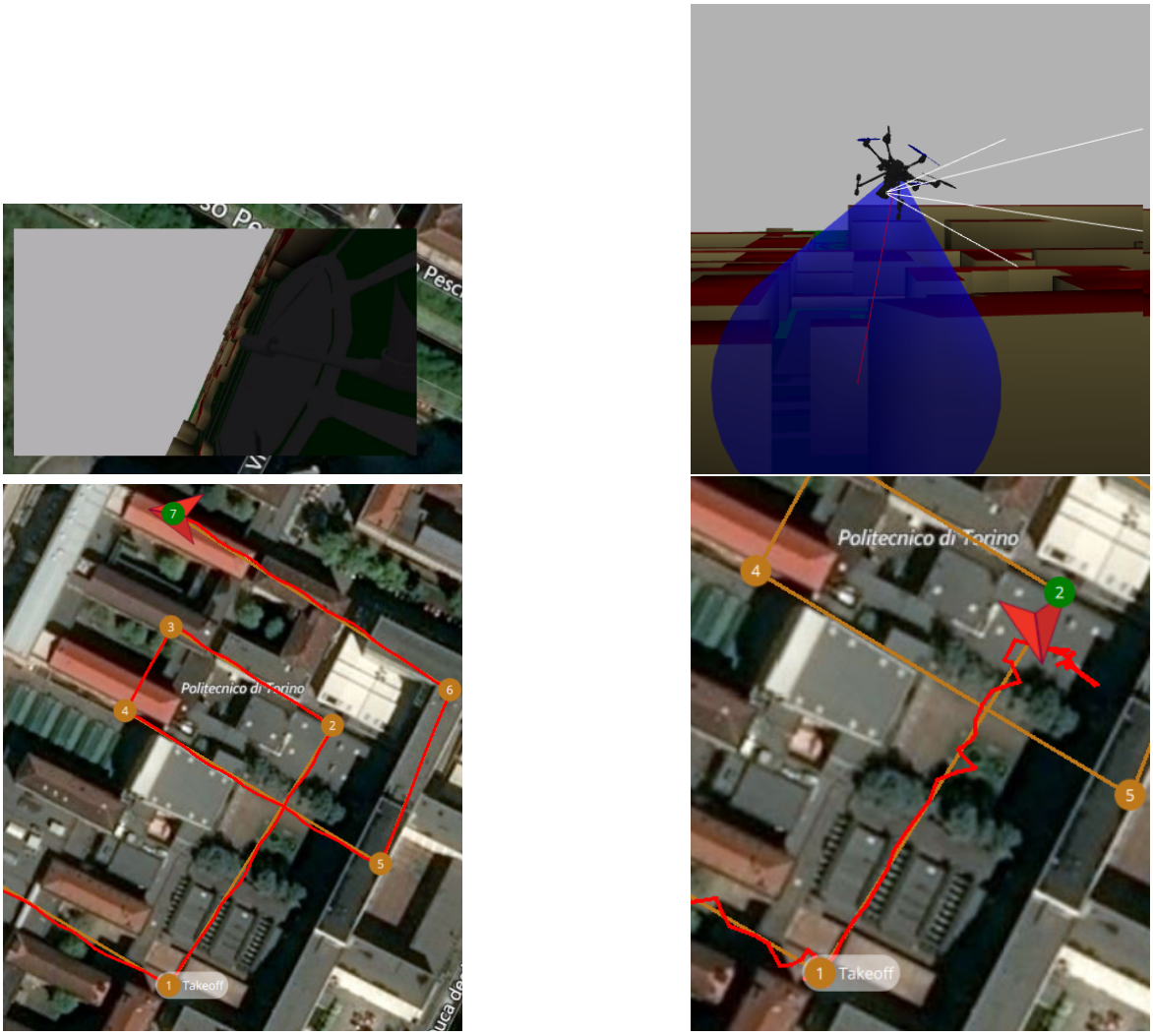


Figure 37: Video streaming shaking, UAV not stable and the path not well followed; in the case of camera mass near to zero the path planned is almost not followed.

motion that destabilized the UAV (maybe a bug of the package [108]). This test was done on a six-rotor UAV model (Typhoon) offered by the PX4 package. The idea could be working on the inertial property of the camera in order to control the Pan and Tilt motion and avoid the oscillations. It has been noticed that if the mass of the camera increases the UAV will not take off; if the camera mass goes down the fly qualities are worse than before. Moreover, if the camera mass is equal to zero the simulation will not run. Regard the inertia of the axis x, y, z (Ixx, Iyy, Izz) and without modifying the crossed inertias (Ixy, Ixz, Iyz), that were set at zero by default, the result was that with smaller camera inertia the path is followed worse than in the other tests. Overall,

changing the camera inertia, will produce a smaller effect as the camera is mounted on a gimbal ([109] [110] [111]). This analysis has been conducted fixing the inertia and changing the mass and vice versa.

The UAV seems to be stable only in the hovering configuration where there are no maneuvers going on and the only motion is the Pan and Tilt of the camera sensor.

5.1.2 First Person View

The second test was conducted with the Iris model, on which it has been attached the FPV camera. It is possible to choose also orientation in order to monitor and record below the UAV (90 degrees) or in front of it (0 degrees) etc. The model is working fine in this case.

The first stage was putting the UAV in an empty Gazebo world, then in the city model, as shown in Figure 38. After that, another world has been created in order to test the animated models, adding to them the textures and scale them (through Blender) with rationale proportion in comparison to the building sizes of the city (in Blender buildings are measured in meters). Lastly, all of the work has been reunited in a single world file: the 3D city with the animated models and the UAV acting as the eye-in-the-sky.

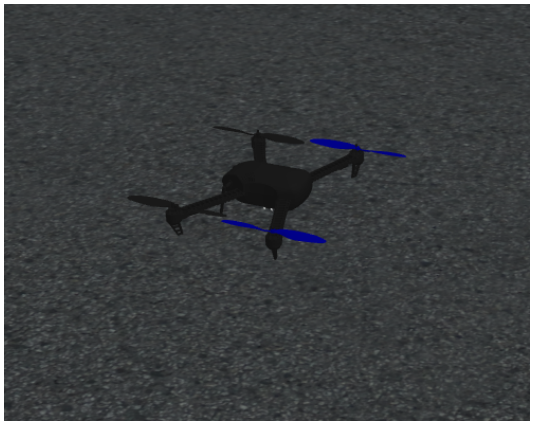


Figure 38: Iris Gazebo model in a world.

In Figure 39, it is possible to have a look at two important details. The former is that the UAV is inscribed in a parallelepiped and the collision with the buildings will consider the parallelepiped, so for example, this means having a collision also before the effective wing touches the wall (in general it is possible to relate the collision to

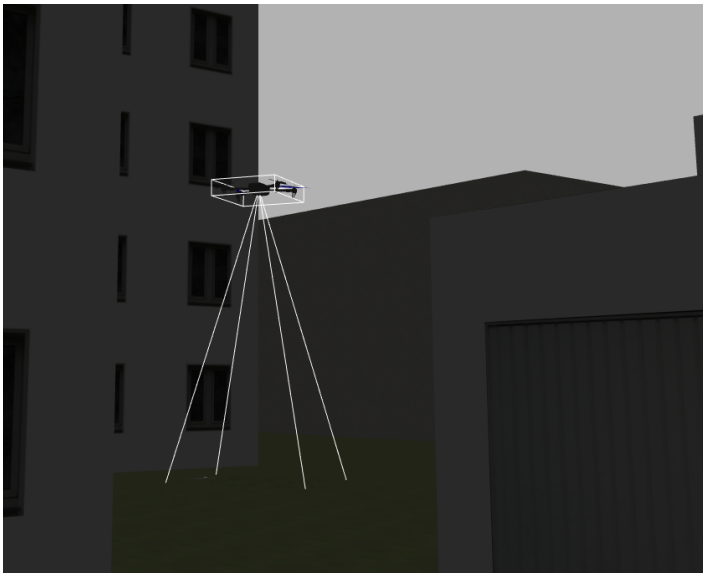


Figure 39: Iris Gazebo model in the 3D city scanning with the FPV.

the model itself). The latter is that there is the Field of View (FOV) of the FPV of the UAV sensor and in this case, the orientation has been set to monitor below.

The second window of Gazebo (FPV output, see Figure 40) is shown. There is also the FOV so three different streaming videos are running in the same simulation. The Iris is flying above the city (also without following it with the first window but remaining in the Planned Home waypoint). A dedicated window in Gazebo gives as output the First Person View from the onboard camera. In this case, the UAV is working as eye-in-the-sky providing a real-time overview of the city.

Another important test was the collision (valid for all of the UAV models of the PX4 package). The animated models pass through the walls (if their waypoints are set in order to try the wall collision) because as said before they are static and they are just useful for creating a realistic city scenario. Instead, for the UAV is vital having a collision mode in action because part of the urban monitoring is also the avoidance of obstacles (like the buildings, trees etc.). If the UAV collides, as shown in Figure 41, it can stick on the wall or it can fall down and in a realistic way, there will be errors and warnings in the ground control station software that will show the failure of the mission. At this point, it is mandatory to restart the simulation and modify the path in a accurate way.

ROS middleware system has been used through RVIZ, RQT Plot etc. In Figure 42, it has shown the Gazebo world, the GCS software and the RQT Plot window where

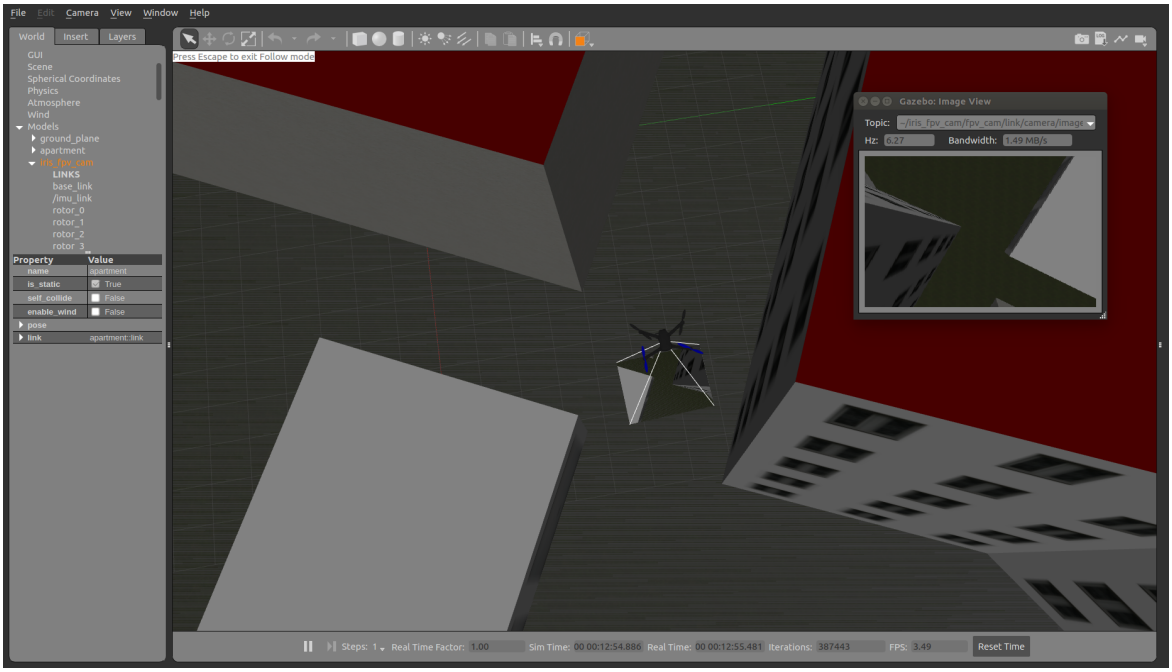


Figure 40: The two windows of Gazebo.

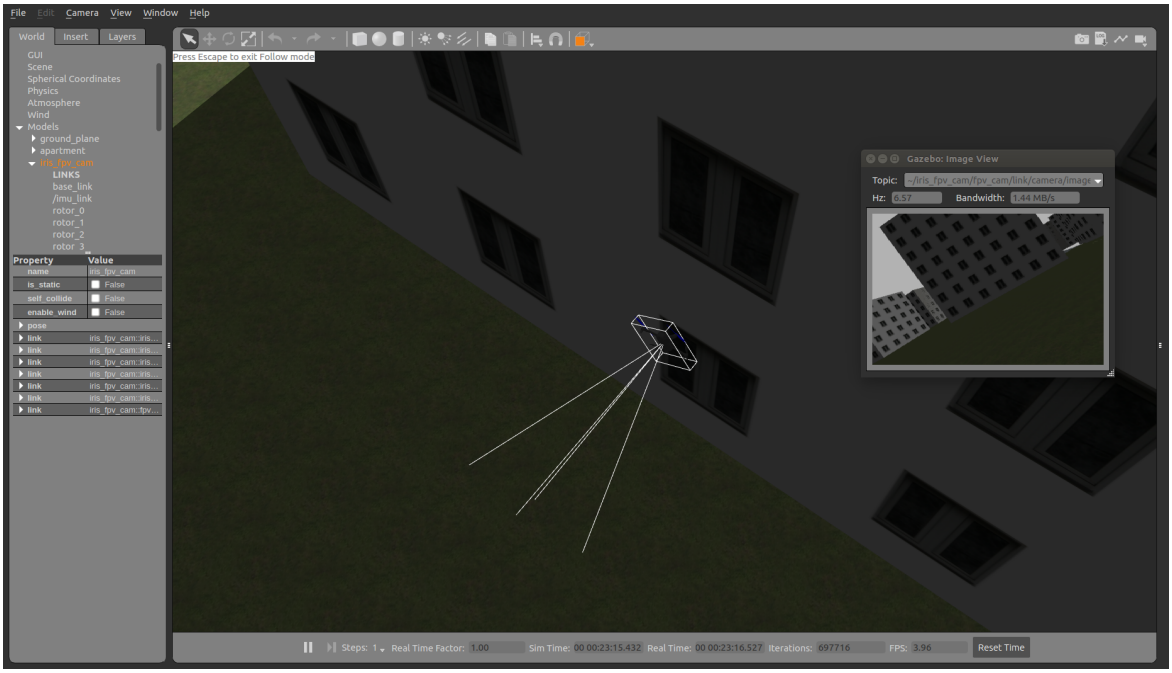


Figure 41: Collision test.

altitude and accelerations are plotted. In Figure 43, instead of the Gazebo world, there is the Ubuntu terminal which shows altitude data from the simulated IMU sensor onboard the UAV.

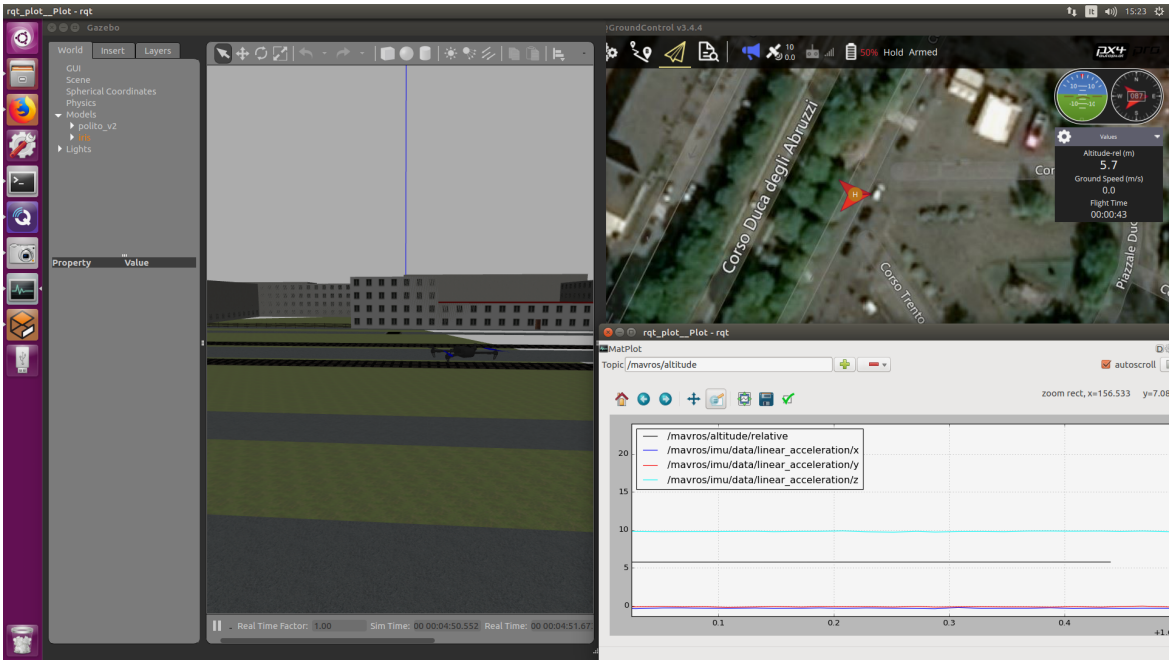


Figure 42: Gazebo, GCS and rqt plot.

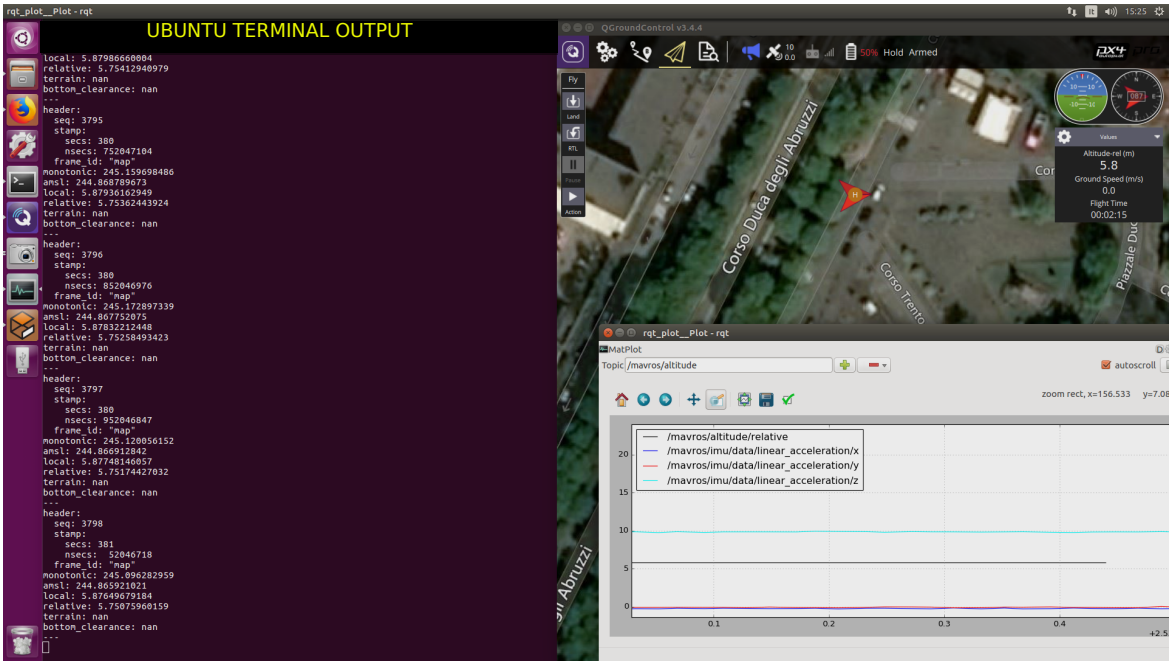


Figure 43: ROS outputs, GCS, rqt plot.

5.2 Case study 2: Simulated Pilot Intervention

In this case study, the packages PX4 (already used) and Sphinx-Parrot will be tested. McCarley et al. [112] describes a UAV system as a system that will vary in the degree to which airframe control is automated either en route or during takeoff and landing. For any system that is not fully automated, including systems that allow for a human

operator to intervene in vehicle control by overriding automation, it will be necessary to provide operators with a control interface through which to manipulate the vehicle. The form of this interface will differ for internal pilots, those who interact with the vehicle through an interface of sensor displays and controls inside a ground control station (first case study and second part of the current case study), and external pilots, those who interact with the vehicle while in visual contact with it at the site of takeoff or landing (at least), full manual control (first part of the current case study).

In any case, it is very vital to offer a simulated framework where pilots can train with a simulated UAV using a joystick or easier a personal smartphone through the UAV app. Guglieri et al. [113] state that using a flight simulator the operator can train without the risk of losing the prototype.

Pleban et al. [114] say that apps offer an easy to use interface, which makes the control of the UAV relatively easy, compared to more professional RC controlled UAVs. Most of the controlling aspects are handled automatically, like the start and landing phases, calibration and position holding.

The current case study is divided into two sub-case studies. In both the possibility to pilot a simulated UAV in a simulated world of Gazebo (outdoor, Turin model). Therefore, the aim is piloting to train without risking the integrity of a real expensive UAV model.

5.2.1 Full Manual Flight Control

The simulation is built on Sphinx-Parrot tools. Sphinx takes over the wifi interface and provides it to the simulated firmware [115]. The firmware runs an instance of host access point daemon (hostapd), which creates a wifi access point. The app (there are more than one, the tested one is FreeFlight Pro [116]) will connect with the simulated UAV, through its stolen WI-FI (the simulated UAV stoles the available WI-FI to which the PC connected and the smartphone will catch the simulated UAV WI-FI). Once connected, the UAV can be controlled exactly like in real life because the current app is the same that is used for real Parrot UAVs. Therefore, FreeFlight Pro is the same app for piloting the Parrot Bebop 2 Power, the Parrot Bebop 2 and the Parrot Bebop. In addition, instead of controlling a UAV in the sky, the UAV will fly in a Gazebo world.

An important test is the collision analysis because training without a simulated collision is useless. So a gas station model has been put in Gazebo and the UAV made to fly around and into a wall. The result achieves is that the collision is working perfectly, and the collision is about the parallelepiped or cube circumscribing the UAV model (same result with PX4). Moreover, when the UAV collides the app shows a warning message related to the failure of the motors (or general malfunction), see Figure 44.

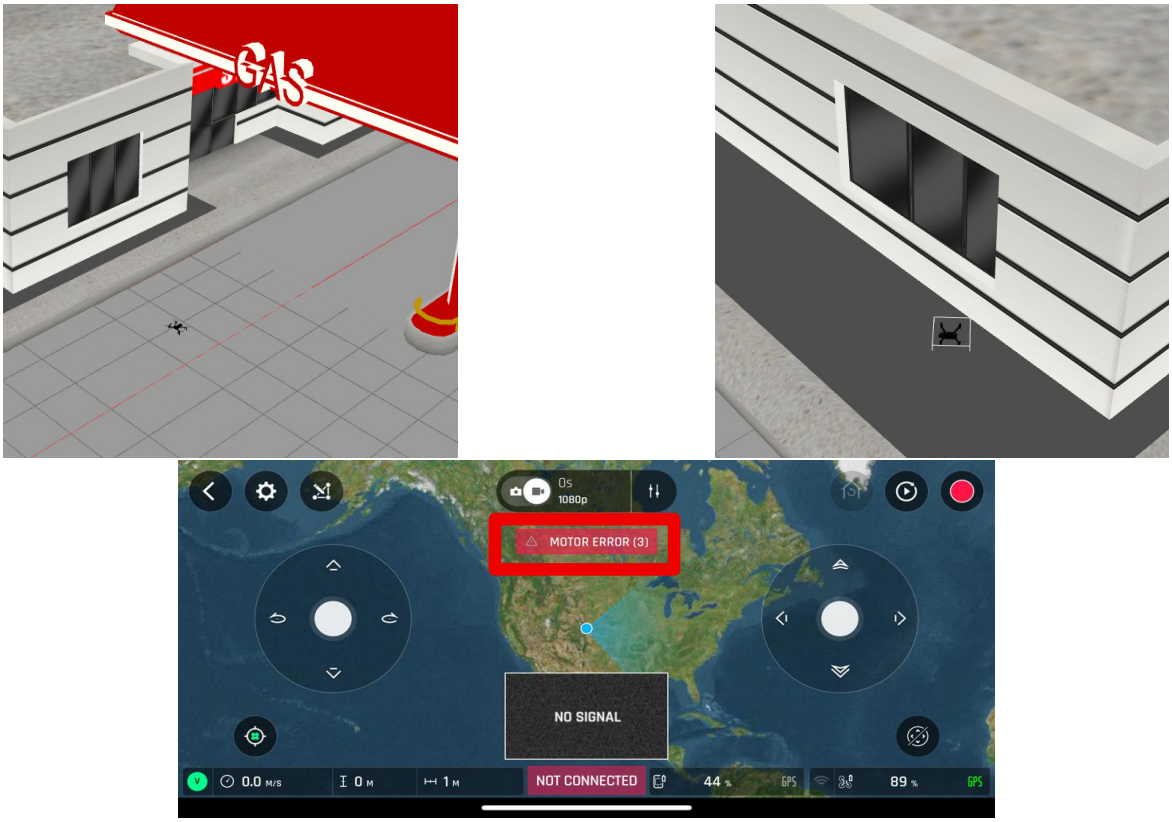


Figure 44: Piloting the UAV in Gazebo, test and result.

After that, the 3D city model has been added together with the UAV model (main file.world). Through the FreeFligth Pro app the UAV has been piloted around and above the Polytechnic of Turin 3D model, as shown in Figure 45.

The only limitation of this package is that the FPV streaming video cannot work without a specific graphics card. An NVIDIA graphics card using the latest proprietary drivers is necessary; then a high-end GPU (e.g. GTX 1060 Ti) is required for the eye-in-the-sky output window in Gazebo.

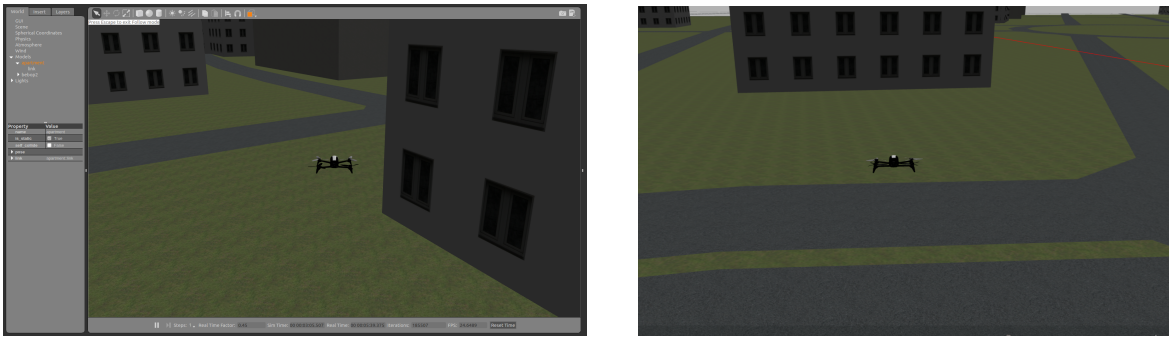


Figure 45: Piloting the UAV in Gazebo Turin 3D model.

5.2.2 Partially Automated Flight Control

Coming back to the PX4 package, now the focus is on the flight control integration between the GCS and the manual intervention of an operator (internal pilot). QGroundControl allows controlling a vehicle using a joystick, a gamepad or an RC. After having connected a PlayStation 4 joystick, it is needed to calibrate it following the on-screen instruction and moving the sticks. Select the flight modes/vehicle functions activated by each joystick button. A maximum of 16 joystick buttons can be linked to 16 actions. Therefore every button has a function and can command the UAV in a different way, for example from Figure 46:

- Arm.
- Rattitude.
- Stabilized.
- Hold.
- VTOL.
- Mission etc.

It is also thinkable to switch at the full manual control but for now, the aim of this test is understanding how an operator can leave the UAV flight above and around the city through the path planned and only, if necessary intervene and take the command. Indeed, an interesting discovery was that the UAV can be stopped during its path and hold its position (if for example, it is asked to focus on a car, a person etc.). After, easily it can resume its mission again, following its previous path.

In this test, in conclusion, after having uploaded the waypoints, and started the mission through the GCS, with the joystick it was possible making the UAV hovering every time needed for focusing, for example, on a zone where more actors and models were present and the monitoring was required for longer time.

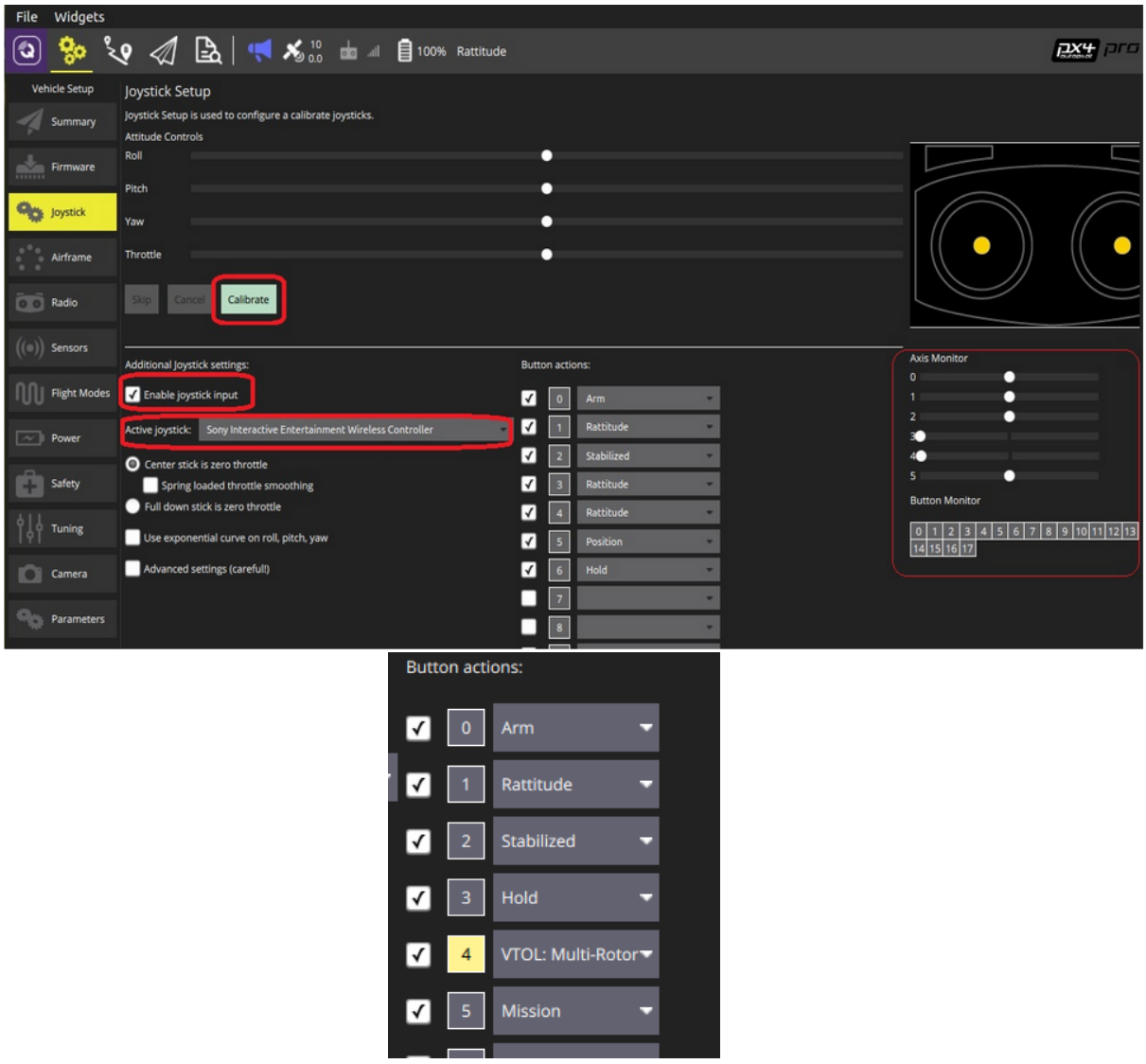


Figure 46: Calibration and settings of the PlayStation 4 joystick in the GCS tool; pressing a button will be underlined by the yellow color on the respective action.

6 Conclusions

6.1 Summary of Results

In this research, first of all, a 3D city model has been created and imported in the physics software simulator Gazebo. Secondly, it has been filled by models, those are static, they do not feel the collision as the 3D world and the UAV do, but they are useful in creating a realistic scenario. Thirdly, two case studies have been analyzed. The first one has been characterized by comparison in terms of UAV models. Within the PX4 package, work has been done on an analysis between the Iris model with the FPV and Typhoon with the Pan and Tilt motion. Both of these models were tested in autonomous flight, the mission was planned on the GCS software through waypoints, with the usage of the SIL simulation. The UAV goes autonomously following the path implemented and simulation of urban monitoring is achieved. Moreover, in the middle of the simulation it is possible to intervene on the GCS panel and add a new waypoint, therefore the UAV changes path reaching the newest waypoint then follows again the old path.

In parallel, second case study, another package has been added: the Sphinx-Parrot. This case study focuses on full manual piloting and partial autonomous flight. This means the pilot has full control (testing Sphinx-Parrot), and he is piloting the UAV with the same app used for real Parrot UAVs. Moreover, the PX4 package has been used again for the semi-autonomous flight. The UAV flies using by waypoints into GCS software and SIL, but if pilots require an intervention, it is plausible to command the UAV with a controller. Then, the mission continues and the UAV follows again the path uploaded at the beginning of the simulated mission. This simulates a GCS operator intervention.

6.2 Future Research

This is the beginning stages of urban monitoring simulations. First of all, some improvements regarding the SIL simulation can be done: using and testing the wind, battery, and motor failure plug-ins. This could be done in order to increase the credibility and the realistic fidelity of simulations. In addition, in order to enlarge the fidelity of the simulated scenario, it is needed to solve the problem regarding the static

feature of the Gazebo models. When it comes to the collision, they have to be active. Lastly, the city model has to be more accurate, all of the kind of vegetation, road signs, electric wires etc. have to be inside the model.

Regarding the streaming video of urban monitoring, different kind of sensors can be tested (more than the FPV and Pan and Tilt camera).

In addition, a lot of other new interesting studies can be conducted. Among them, the crucial interest goes on target acquisition and detection (using RVIZ in all of its potential, and not only for video streaming or feedback like in the current dissertation), HIL simulation, advanced obstacles avoidance and the use of a fleet of UAVs. The former can be useful to recognize an animated Gazebo model from another one (different textures). It could be done using a computer vision and machine learning software such as OpenCV [117]. The HIL simulation is very vital for testing hardware performances. This approach has the benefit of testing most of the actual flight code on the real hardware. When it comes to obstacle avoidance, different techniques can be used, for instance, optical flow or vision based obstacle avoidance, and image processing techniques (using OpenCV). The optical flow provides very important information about the robot environment as the obstacle disposition, the time to collision and the depth of objects [118]. Lastly, the use of a fleet of UAVs (or a swarm) is crucial to monitor a bigger area of the city or the same zone but with more accuracy. It is required to establish a high degree of communication and collaboration between the fleet while the mission is running.

References

[1] K. P. Valavanis, *Advances in unmanned aerial vehicles: state of the art and the road to autonomy*, vol. 33. Springer Science & Business Media, 2008.

[2] "U.S. Department of Defense." https://www.defense.gov/. Accessed: 2018-10.

[3] "Department of defense. unmanned aircraft system airspace integration plan. version 2.0 march 2011 prepared by: Uas task force airspace integration integrated product team, 2011.,"

[4] E. Capello, M. Dentis, G. Guglieri, L. N. Mascarello, and L. S. Cuomo, "An innovative cloud-based supervision system for the integration of rpas in urban environments," *Transportation Research Procedia*, vol. 28, pp. 191–200, 2017.

[5] K. Dalamagkidis, "Classification of uavs," in *Handbook of unmanned aerial vehicles*, pp. 83–91, Springer, 2015.

[6] M. Arjomandi, S. Agostino, M. Mammone, M. Nelson, and T. Zhou, "Classification of unmanned aerial vehicles," *Mech Eng*, vol. 3016, 2006.

[7] A. I. Hentati, L. Krichen, M. Fourati, and L. C. Fourati, "Simulation tools, environments and frameworks for uav systems performance analysis," in *2018 14th International Wireless Communications & Mobile Computing Conference (IWCMC)*, pp. 1495–1500, IEEE, 2018.

[8] "MIT Artificial Intelligence Lab." https://www.csail.mit.edu/news/self-flying-drone-dips-darts-and-dives-through-trees-30-mph. Accessed: 2018-12.

[9] I. Greco and M. Bencardino, "The paradigm of the modern city: Smart and senseable cities for smart, inclusive and sustainable growth," in *International Conference on Computational Science and Its Applications*, pp. 579–597, Springer, 2014.

[10] M. A. Khan, B. A. Alvi, A. Safi, and I. U. Khan, "Drones for good in smart cities: A review,"

[11] "Avionics International." https://www.aviationtoday.com/2018/03/28/air-force-aims-overwhelm-opposition-expendable-drones/. Accessed: 2018-09.

[12] S. M. Adams and C. J. Friedland, "A survey of unmanned aerial vehicle (uav) usage for imagery collection in disaster research and management," in *9th International Workshop on Remote Sensing for Disaster Response*, p. 8, 2011.

[13] "38 Ways Drones Will Impact Society: From Fighting War To Forecasting Weather, UAVs Change Everything ." http://www.cbinsights.com/research/drone-impact-society-uav/. Accessed: 2018-11.

[14] A. Danilov, U. D. Smirnov, and M. Pashkevich, "The system of the ecological monitoring of environment which is based on the usage of uav," *Russian journal of ecology*, vol. 46, no. 1, pp. 14–19, 2015.

[15] P. Tripicchio, M. Satler, G. Dabisias, E. Ruffaldi, and C. A. Avizzano, "Towards smart farming and sustainable agriculture with drones," in *Intelligent Environments (IE), 2015 International Conference on*, pp. 140–143, IEEE, 2015.

[16] "MIT Technology Review." https://www.technologyreview.com/s/601935/six-ways-drones-are-revolutionizing-agriculture/. Accessed: 2018-11.

[17] "The latest weapons in the fight against ocean plastic? Drones and an algorithm." https://www.weforum.org/agenda/2018/06/this-ai-is-learning-to-recognize-ocean-plastic-using-drone-photos/. Accessed: 2018-11.

[18] K. Han, J. Lin, and M. Golparvar-Fard, "A formalism for utilization of autonomous vision-based systems and integrated project models for construction progress monitoring," in *Proc., 2015 Conference on Autonomous and Robotic Construction of Infrastructure*, 2015.

[19] N. Noor and N. A. Rosni, "The evolution of uavs applications in urban planning," 06 2017.

[20] "This people-moving drone has completed more than 1,000 test flights." https://www.popsci.com/ehang-passenger-carrying-drone. Accessed: 2018-12.

[21] "Il taxi volante senza pilota ha terminato il suo primo volo." http://www.mobinews.it/article/95892. Accessed: 2019-01.

[22] "Fast-Forwarding to a Future of On-Demand Urban Air Transportation." https://www.uber.com/elevate.pdf. Accessed: 2018-12.

[23] "Amazon Prime Air." https://www.amazon.com/Amazon-Prime-Air/b?ie=UTF8&node=8037720011. Accessed: 2018-09.

[24] J. Fleureau, Q. Galvane, F.-L. Tariolle, and P. Guillotel, "Generic drone control platform for autonomous capture of cinema scenes," in *Proceedings of the 2nd Workshop on Micro Aerial Vehicle Networks, Systems, and Applications for Civilian Use*, pp. 35–40, ACM, 2016.

[25] E. Semsch, M. Jakob, D. Pavlicek, and M. Pechoucek, "Autonomous uav surveillance in complex urban environments," in *Proceedings of the 2009 IEEE/WIC/ACM International Joint Conference on Web Intelligence and Intelligent Agent Technology-Volume 02*, pp. 82–85, IEEE Computer Society, 2009.

[26] F. Mohammed, A. Idries, N. Mohamed, J. Al-Jaroodi, and I. Jawhar, "Uavs for smart cities: Opportunities and challenges," in *Unmanned Aircraft Systems (ICUAS), 2014 International Conference on*, pp. 267–273, IEEE, 2014.

[27] "Commercial drones are here: The future of unmanned aerial systems." http://www.mckinsey.com/industries/capital-projects-and-infrastructure/our-insights/commercial-drones-are-here-the-future-of-unmanned-aerial-systems. Accessed: 2018-10.

[28] "European Drones Outlook Study." https://www.sesarju.eu/sites/default/files/documents/reports/European_Drones_Outlook_Study_2016.pdf. Accessed: 2018-09.

[29] "EPIC-Smart Cities." http://www.epic-cities.eu/content/smart-cities. Accessed: 2019-01.

[30] H. Chourabi, T. Nam, S. Walker, J. R. Gil-Garcia, S. Mellouli, K. Nahon, T. A. Pardo, and H. J. Scholl, "Understanding smart cities: An integrative framework," in *System Science (HICSS), 2012 45th Hawaii International Conference on*, pp. 2289–2297, IEEE, 2012.

[31] G. Falconer and S. Mitchell, "Smart city framework: a systematic process for enabling smart+ connected communities," *Cisco IBSG*, pp. 1–11, 2012.

[32] G. Wu, S. Talwar, K. Johnsson, N. Himayat, and K. D. Johnson, "M2m: From mobile to embedded internet," *IEEE Communications Magazine*, vol. 49, no. 4, 2011.

[33] A. Juels, "Rfid security and privacy: A research survey," *IEEE journal on selected areas in communications*, vol. 24, no. 2, pp. 381–394, 2006.

[34] K. Doppler, M. Rinne, C. Wijting, C. B. Ribeiro, and K. Hugl, "Device-to-device communication as an underlay to lte-advanced networks," *IEEE Communications Magazine*, vol. 47, no. 12, 2009.

[35] R. L. Finn and D. Wright, "Unmanned aircraft systems: Surveillance, ethics and privacy in civil applications," *Computer Law & Security Review*, vol. 28, no. 2, pp. 184–194, 2012.

[36] P. Malone, H. Apgar, S. Stukes, and S. Sterk, "Unmanned aerial vehicles unique cost estimating requirements," in *Aerospace Conference, 2013 IEEE*, pp. 1–8, IEEE, 2013.

[37] D. W. King, A. Bertapelle, and C. Moses, "Uav failure rate criteria for equivalent level of safety," in *International helicopter safety symposium, Montreal*, vol. 9, 2005.

[38] K. Kanistras, G. Martins, M. J. Rutherford, and K. P. Valavanis, "A survey of unmanned aerial vehicles (uavs) for traffic monitoring," in *Unmanned Aircraft Systems (ICUAS), 2013 International Conference on*, pp. 221–234, IEEE, 2013.

[39] "ROS Tutorials." http://wiki.ros.org/ROS/Tutorials. Accessed: 2018-10.

[40] "Simulation." https://dev.px4.io/en/simulation/. Accessed: 2018-10.

[41] "OSM2World." https://wiki.openstreetmap.org/wiki/OSM2World. Accessed: 2018-10.

[42] "JOSM." https://josm.openstreetmap.de/. Accessed: 2018-10.

[43] "Gazebo Tutorials." http://gazebosim.org/tutorials. Accessed: 2018-11.

[44] "OpenStreetMap." https://www.openstreetmap.org/#map=16/45.0621/7.6617. Accessed: 2018-10.

[45] T. Avanesov, N. Louveton, R. McCall, V. Koenig, and M. Kracheel, "Towards a simple city driving simulator based on speed dreams and osm," in *Adjunct Proceedings of 4th International Conference on Automotive User Interfaces and Interactive Vehicular Applications*, vol. 2, pp. 32–33, ACM SIGCHI, 2012.

[46] "Open Source 3D creation. Free to use for any purpose, forever.." https://www.blender.org/. Accessed: 2018-10.

[47] "Create a world with OpenStreetMap." https://developer.parrot.com/docs/sphinx/openstreetmap.html. Accessed: 2018-11.

[48] M. Goetz and A. Zipf, "Openstreetmap in 3d–detailed insights on the current situation in germany," *City, Proc. Agil*, pp. 24–27, 2012.

[49] "JOSM." https://josm.openstreetmap.de/wiki/Download. Accessed: 2018-10.

[50] "Java." https://www.java.com/it/download/. Accessed: 2018-10.

[51] "Esri CityEngine." https://www.esri.com/en-us/arcgis/products/esri-cityengine/overview. Accessed: 2010-09.

[52] J. Craighead, R. Murphy, J. Burke, and B. Goldiez, "A survey of commercial & open source unmanned vehicle simulators," in *Robotics and Automation, 2007 IEEE International Conference on*, pp. 852–857, IEEE, 2007.

[53] T. Balch and R. C. Arkin, "Behavior-based formation control for multirobot teams," *IEEE transactions on robotics and automation*, vol. 14, no. 6, pp. 926–939, 1998.

[54] "Gazebo." http://gazebosim.org/. Accessed: 2018-10.

[55] "SimRobot - Robotiksimulator." http://www.informatik.uni-bremen.de/simrobot/. Accessed: 2018-10.

[56] "FlightGear." http://www.flightgear.org. Accessed: 2018-10.

[57] "Robotics and Automation Lab." http://robotics.ee.uwa.edu.au/simulation.html. Accessed: 2018-10.

[58] "Unity." https://unity3d.com/. Accessed: 2018-10.

[59] F. M. Noori, D. Portugal, R. P. Rocha, and M. S. Couceiro, "On 3d simulators for multi-robot systems in ros: Morse or gazebo?," in *Safety, Security and Rescue Robotics (SSRR), 2017 IEEE International Symposium on*, pp. 19–24, IEEE, 2017.

[60] E. Ebeid, M. Skriver, K. H. Terkildsen, K. Jensen, and U. P. Schultz, "A survey of open-source uav flight controllers and flight simulators," *Microprocessors and Microsystems*, vol. 61, pp. 11–20, 2018.

[61] "AirSim." https://github.com/Microsoft/AirSim. Accessed: 2018-10.

[62] "Unreal Engine." https://www.unrealengine.com/. Accessed: 2018-10.

[63] T. Balch and R. C. Arkin, "Design and use paradigms for gazebo, an open-source multi-robot simulator," *IEEE/RSJ International Conference on Intelligent Robots and System*, 2004.

[64] T. Balch and R. C. Arkin, "Modular open robots simulation engine: Morse," *IEEE International Conference On Robotics and Automation*, 2011.

[65] "jMAVSim Simulation." https://dev.px4.io/en/simulation/jmavsim.html. Accessed: 2018-11.

[66] B. Gati, "Open source autopilot for academic research-the paparazzi system," in *American Control Conference (ACC), 2013*, pp. 1478–1481, IEEE, 2013.

[67] "HackFlight Simulator." https://diydrones.com/profiles/blogs/hackflightsim-a-simple-quacopter-flight-simulator-using-actual-c. Accessed: 2018-10.

[68] "Arduino." https://www.arduino.cc/en/Main/Software. Accessed: 2018-10.

[69] M. Huang, N. Dr. Chakraborty, and M. H. Dr. Nguyen, "The effect of quadcopter guidance in crowd emergency evacuation scenarios: Simulation and analysis,"

[70] M. R. Zofka, S. Klemm, F. Kuhnt, T. Schamm, and J. M. Zöllner, "Testing and validating high level components for automated driving: simulation framework for traffic scenarios," in *Intelligent Vehicles Symposium (IV), 2016 IEEE*, pp. 144–150, IEEE, 2016.

[71] G. Brunner, B. Szebedy, S. Tanner, and R. Wattenhofer, "The urban last mile problem: Autonomous drone delivery to your balcony," *arXiv preprint arXiv:1809.08022*, 2018.

[72] "PX4 Autopilot." http://px4.io/. Accessed: 2018-11.

[73] "Ardupilot Autopilot." http://ardupilot.org/. Accessed: 2018-11.

[74] A. CARDAMONE, "Implementation of a pilot in the loop simulation environment for uav development and testing," 2017.

[75] "DCULab." http://www.modulabs.co.kr/board_GDCH80/1543. Accessed: 2018-12.

[76] K. D. Nguyen, C. Ha, and J. T. Jang, "Development of a new hybrid drone and software-in-the-loop simulation using px4 code," in *International Conference on Intelligent Computing*, pp. 84–93, Springer, 2018.

[77] A. Bittar, H. V. Figuereido, P. A. Guimaraes, and A. C. Mendes, "Guidance software-in-the-loop simulation using x -plane and simulink for uavs," *Conference Paper*, 2014.

[78] L. Meier, D. Honegger, and M. Pollefeys, "Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms," *Conference Paper*, 2015.

[79] D. F. d. Castro and D. A. d. Santos, "A software-in-the-loop simulation scheme for position formation flight of multicopters," *Journal of Aerospace Technology and Management*, vol. 8, no. 4, pp. 431–440, 2016.

[80] "FlightGear Flight Simulator." http://home.flightgear.org/. Accessed: 2018-10.

[81] S.-h. Cheon, S.-w. Ha, and Y.-h. Moon, "Hardware-in-the-loop simulation platform for image-based object tracking method using small uav," in *Digital Avionics Systems Conference (DASC), 2016 IEEE/AIAA 35th*, pp. 1–5, IEEE, 2016.

[82] M. Odelga, P. Stegagno, H. H. Bülthoff, and A. Ahmad, "A setup for multi-uav hardware-in-the-loop simulations," in *2015 workshop on research education and development of unmanned aerial systems, Mexico*, pp. 204–210, 2015.

[83] "Pixhawk Hardware for Open Source Autopilot." http://pixhawk.org/. Accessed: 2018-11.

[84] C. E. Agüero, N. Koenig, I. Chen, H. Boyer, S. Peters, J. Hsu, B. Gerkey, S. Paepcke, J. L. Rivero, J. Manzo, *et al.*, "Inside the virtual robotics challenge: Simulating real-time robotic disaster response," *IEEE Transactions on Automation Science and Engineering*, vol. 12, no. 2, pp. 494–506, 2015.

[85] "OpenPilot." https://opwiki.readthedocs.io/en/latest/user_manual/gcs_install_op.html. Accessed: 2018-11.

[86] "Multiwii." http://www.multiwii.com. Accessed: 2018-10.

[87] "APM Planner 2." http://ardupilot.org/planner2/. Accessed: 2019-02.

[88] "Data Logs in Mission Planner." http://ardupilot.org/planner/docs/common-downloading-and-analyzing-data-logs-in-mission-planner.html?highlight=mavlink. Accessed: 2019-02.

[89] C. Ramirez-Atencia and D. Camacho, "Extending qgroundcontrol for automated mission planning of uavs," *Sensors*, vol. 18, no. 7, p. 2339, 2018.

[90] "GStreamer." https://gstreamer.freedesktop.org/documentation/application-development/introduction/gstreamer.html. Accessed: 2018-11.

[91] "Gazebo Simulation with PX4." https://dev.px4.io/en/simulation/gazebo.html. Accessed: 2018-11.

[92] "ROS and MAVROS." wiki.ros.org/mavros. Accessed: 2018-12.

[93] H. Ajmal, R. Ragel, J. M. Roberts, and L. F. Gonzalez, "A framework for vision-based multiple target finding and action using multirotor uavs," 2018.

[94] M. Campusano and J. Fabry, "From robots to humans: Visualizations for robot sensor data," in *Software Visualization (VISSOFT), 2015 IEEE 3rd Working Conference on*, pp. 135–139, IEEE, 2015.

[95] "RVIZ." http://wiki.ros.org/rviz. Accessed: 2019-01.

[96] "RQT PLOT." http://wiki.ros.org/rqt_plot. Accessed: 2019-01.

[97] "RQT GRAPH." http://wiki.ros.org/rqt_graph. Accessed: 2019-01.

[98] A. G. Florea, "Integrating a v-rep simulated mobile robot into ros," *UNIVERSITY POLITEHNICA OF BUCHAREST SCIENTIFIC BULLETIN SERIES C-ELECTRICAL ENGINEERING AND COMPUTER SCIENCE*, vol. 80, no. 3, pp. 3–16, 2018.

[99] R. Mishra and A. Javed, "Ros based service robot platform," in *2018 4th International Conference on Control, Automation and Robotics (ICCAR)*, pp. 55–59, IEEE, 2018.

[100] Y. Feng, C. Ding, X. Li, and X. Zhao, "Integrating mecanum wheeled omni-directional mobile robots in ros," in *Robotics and Biomimetics (ROBIO), 2016 IEEE International Conference on*, pp. 643–648, IEEE, 2016.

[101] "Gazebo Motor and Propeller Model Notes." https://github.com/mvernacc/gazebo_motor_model_docs/blob/master/notes.pdf. Accessed: 2018-12.

[102] "ROS integration overview." http://gazebosim.org/tutorials?tut=ros_overview. Accessed: 2019-02.

[103] "Config and SDF." http://gazebosim.org/tutorials?tut=model_structure. Accessed: 2019-02.

[104] "Google Maps." https://www.google.it/maps. Accessed: 2018-12.

[105] "RGB Color Model." https://en.wikipedia.org/wiki/RGB_color_model. Accessed: 2018-12.

[106] "Graphics Processing Unit." https://en.wikipedia.org/wiki/Graphics_processing_unit. Accessed: 2018-11.

[107] "Gazebo Tutorial Actors." http://gazebosim.org/tutorials?tut=actor. Accessed: 2018-10.

[108] "MS Windows NT kernel description." http://discuss.px4.io/t/typhoon-model/8823. Accessed: 2018-12.

[109] "Gimbal." https://www.dronezon.com/learn-about-drones-quadcopters/drone-gimbal-design-components-parts-technology-overview/. Accessed: 2019-01.

[110] M. Quigley, M. A. Goodrich, S. Griffiths, A. Eldredge, and R. W. Beard, "Target acquisition, localization, and surveillance using a fixed-wing mini-uav and gimbaled camera," in *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE international Conference on*, pp. 2600–2605, IEEE, 2005.

[111] B. Ekstrand, "Equations of motion for a two-axes gimbal system," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 37, pp. 1083–1091, July 2001.

[112] J. S. McCarley and C. D. Wickens, "Human factors implications of uavs in the national airspace," 2005.

[113] E. Capello, G. Guglieri, and F. B. Quagliotti, "Uavs and simulation: an experience on mavs," *Aircraft Engineering and Aerospace Technology*, vol. 81, no. 1, pp. 38–50, 2009.

[114] R. C. Johann-Sebastian Pleban, Ricardo Band, "Hacking and securing the ar.drone 2.0 quadcopter: investigations for improving the security of a toy," 2014.

[115] "Stolen Wifi." https://developer.parrot.com/docs/sphinx/firststep.html. Accessed: 2019-02.

[116] "Parrot Drones." https://www.parrot.com/global/freeflight-pro. Accessed: 2018-10.

[117] "Open CV and Python Color Detection." https://www.pyimagesearch.com/2014/08/04/opencv-python-color-detection/. Accessed: 2019-02.

[118] K. Souhila and A. Karim, "Optical flow based robot obstacle avoidance," *International Journal of Advanced Robotic Systems*, vol. 4, no. 1, p. 2, 2007.