UNIVERSITY OF SALERNO

DEPARTMENT OF INDUSTRIAL ENGINEERING

Ph.D. Course in Industrial Engineering
Curriculum in Industrial Engineering - XXXI Cycle

ROBOTIC FRAMEWORK FOR DEVELOPING OF UNMANNED VEHICLES

Supervisor	**Ph.D. Student**
Prof. Domenico Guida	*Zandra Betzabé Rivera Chávez*

Scientific Referees

Prof. Jhoniers Gilberto Gerrero Erazo
Prof. Predrag Dasic

Ph.D. Course Coordinator

Prof. Francesco Donsì

Academic year 2017/2018

Acknowledgments

First of all, I would like to thank Professor Domenico Guida infinitely for the academic, professional, and personal support provided me during these years of doctoral studies outside my home. Without his help, his patience, his recommendations, and suggestions, it would not be possible to take this vital step in my professional and personal life in one of the most challenging moments of my life.

I would also give an exceptional thanks to my co-supervisor, Marco Claudio de Simone, for his permanent support and his valuable advice in all aspects that allowed me to continue with the thesis.

I want to express my immense gratitude to the entire team of Professor Guida, who received me with respect and affection, making this journey possible.

I want to give an exceptional thanks to my family, who have always been by my side, giving me support. I want to sincerely thank my mother, who knew how to overcome all the difficulties still to provide me with words of encouragement and hope. Other special thanks to my little nieces for being my "moral backbone."

Another very special thank you to my husband, who has been making back and forth traveling to support me and teach me. His presence helped me a lot to be active to face of all adversities.

- Rivera, Z. B., De Simone, M. C., & Guida, D. (2019). *Unmanned Ground Vehicle Modelling in Gazebo/ROS-Based Environments. Machines*, 7(2), 42.

- De Simone, M., Rivera, Z., & Guida, D. (2018). *Obstacle avoidance system for unmanned ground vehicles by using ultrasonic sensors. Machines*, 6(2), 18.

- Rivera, Z. B., De Simone, M. C., Guida D. *Waypoint navigation for wheeled mobile robots in ROS-based environments*. 18th International Conference "Research and Development in Mechanical Industry" RADMI-2018 13-16 September 2018 Vrnjačka Banja (Serbia)

Contents

i. List of Figures

ii. List of Tables

iii. Abstract

Unmanned Vehicles, known as UVs, have been developed to accomplish difficult, tedious, and unpleasant missions for human beings. Their usage started in military applications; subsequently, specific industries adopt them to increase the productive capacity of their factories already automated with industrial robots relying on the mobility and autonomy offered by these vehicles. However, unmanned vehicles could operate in many other sectors - like in agriculture, construction, logistic, customer service -, facilitating and improving the quality of life in general. It is necessary to increase its development and implementation substantially to achieve this.

As we will show in this document, progress in the area of mobile robotics, especially in the field of unmanned vehicles, has been essential and proliferates. However, it is not robust yet to be reliable and accepted beyond the controlled environments in which they operate nowadays. The enlarged area to cover, due to mobile capabilities, plus the risky missions they need to accomplish, increases the complexity of autonomous steering and control of such a vehicle. For this reason, the modeling considerations to achieve the task of autonomous driving is considered complex and reserved to humans (Litman T., 2018), due to the high frequency of interactions with other mobile objects, which requires sensing and acting capabilities in real-time bases, with an essential degree of intelligence and skills.

An essential step in their development is the robotics environments for developing and testing unmanned vehicles, these computational environments incorporate and centralizes all technologies, in their broadest sense, related to robotics. In both professional and academic literature, these environments are called with different names, such as robotic middleware, robotic platform, and robotic framework. Their degree of development and their capacities are not homogeneous, being those specialized and commercial branded who have reached essential levels of maturity and acceptance. As it is the case of X-Plane, a platform for the simulation of autonomous flights of many well-known aircraft, the Federal Aviation Administration (FAA) can even certify the implementation if they also count on certified hardware.

Some conventional robot models are offered in these robotic environments, which are highly used for research in robotics. It is an essential contribution to the robotics community to get the research efforts to concentrate on the central issues of their work. However, it does not help much if there is a need to test a new robot model from scratch, where the initial main effort is in the modeling and evaluation of behavior in order to redesign the model itself. These environments also offer standard robotic functionalities that come to help in both cases.

Then, the complete development of a new unmanned vehicle, from a 3D model creation to the real prototype testing, requires an ad-hoc platform in every step of the process. It also requires a good understanding of kinematic and dynamic modeling, added to programming skills and powerful simulation environments. To our knowledge, no open-source robotic environment or system can cover the complete process of UVs development robustly and quickly. Some of them are powerful in control and simulation; others help a lot in algorithm development; some others accept mathematical formulation naturally; others deal very well with the communication systems.

Thus, one solution is to integrate some of these platforms and benefit from their advantages. For this reason, we have thought to develop an integrated framework that facilitates the design of unmanned vehicles, initially in simulation, capable of fulfilling autonomous behavior, performing tasks like path planning, location, mapping, and safe navigation by avoiding obstacles in the ground and air environments. We initiate by evaluating the offer, by installing and testing some robotics middleware in order to choose the platform that allows the best integration capabilities with robust applications at each step of unmanned vehicle modeling from scratch.

Doing in this way, we integrate different robotic platforms and tools to build a framework in which it is possible to have all the standard functionalities by type of unmanned vehicle. Therefore, when the need for a new vehicle arrives, it is possible to create and add a new model, with its peculiarities (characteristics and capabilities). Thus, our robotic framework, called UNISA-UVF, is designed to facilitate the modeling, simulation, and testing of unmanned vehicles. UNISA-UVF is a sensor-based robotics system that uses model-based and learning-based approaches.

Also, in our framework, it is possible to create different versions of the same vehicles with slight variations in the description of their morphology, which facilitates the collaborative missions in which several UVs are required to carry out them together. Therefore, we reserve a workspace in which we have implemented some classic group activities such as leader-follower or cooperative-SLAM. Having built our framework on the open-source middleware Gazebo-ROS, we can take full advantage of code reusing.

Our framework will allow the Industrial Engineering Department of UNISA to build and test unmanned ground and air vehicles in simulated environments with the possibility of testing physical prototypes with much less effort. Our framework, designed to be completely reusable, also allows integration with MATLAB/Simulink and X- plane in order to increase this capacity, by using 3D design software for vehicle modeling.

iv. Resume

I veicoli senza pilota, noti come UV, sono stati sviluppati per compiere missioni difficili, noiose e spiacevoli per gli esseri umani. Il loro utilizzo è iniziato in applicazioni militari; successivamente, alcune industrie specifiche li hanno adottati per aumentare la capacità produttiva delle loro fabbriche già automatizzate con dei robot industriali affidandosi alla mobilità e all'autonomia offerta da questi veicoli. Però, i veicoli senza pilota potrebbero operare in molti altri settori - come l'agricoltura, l'edilizia, la logistica, il servizio al cliente -, facilitando e migliorando la qualità di vita in generale. Per raggiungere questo obiettivo è necessario aumentarne sostanzialmente lo sviluppo e l'implementazione gli UVs.

Come mostreremo in questo documento, i progressi nell'area della robotica mobile, ovvero il settore dei veicoli senza pilota, sono stati essenziali e proliferano. Tuttavia, non è ancora robusto per essere affidabile e accettato al di là degli ambienti controllati in cui operano al giorno d'oggi. La vasta area da coprire, vista la capacità mobile, e le missioni rischiose che devono compiere, aumenta la complessità della guida e del controllo autonomi di un veicolo di questo tipo. Per tale ragione, le considerazioni modellistiche per raggiungere il compito di guida autonoma sono considerate complesse e quindi riservate all'uomo (Litman T., 2018), a causa dell'alta frequenza di interazioni con altri oggetti mobili, che richiede capacità di rilevamento e di azione in tempo reale, con un grado di intelligenza e competenze fondamentali.

Un passo essenziale nel loro sviluppo sono gli ambienti robotici per lo sviluppo e la sperimentazione di veicoli non pilotati, questi ambienti computazionali incorporano e centralizzano tutte le tecnologie, nel loro senso più ampio, legate alla robotica. In ambito sia professionale che accademico, questi ambienti sono chiamati con nomi diversi, come middleware robotico, piattaforma robotica, e framework robotico. Il loro grado di sviluppo e le loro capacità non sono omogenee, in quanto sono quelli specializzati e commerciali di marca che hanno raggiunto livelli essenziali di maturità e accettazione. Come nel caso di X- Plane, una piattaforma per la simulazione di voli autonomi di molti velivoli rinomati, anche la Federal Aviation Administration (FAA) può certificarne l'implementazione se dispone di un hardware certificato.

In questi ambienti robotizzati vengono offerti alcuni modelli di robot convenzionali, molto utilizzati per la ricerca in robotica. Si tratta di un contributo essenziale alla comunità della robotica per far sì che gli sforzi di ricerca si concentrino sulle questioni centrali del loro lavoro. Tuttavia, non aiuta molto se c'è la necessità di testare da capo un nuovo modello di robot, dove lo sforzo principale di partenza è la modellazione e la valutazione del comportamento

per riprogettare il modello stesso. Tali ambienti offrono anche funzionalità robotiche standard che vengono in aiuto in entrambi i casi.

Quindi, lo sviluppo completo di un nuovo veicolo non pilotato, dalla creazione di un modello 3D al testaggio del prototipo reale, richiede una piattaforma ad hoc in ogni fase del processo. Richiede inoltre una buona conoscenza della modellazione cinematica e dinamica, oltre a competenze di programmazione e ad ambienti di simulazione potenti. A nostra conoscenza, nessun ambiente o sistema robotizzato open-source può coprire l'intero processo di sviluppo degli UV in modo robusto e rapido. Alcuni di essi sono potenti nel controllo e nella simulazione; altri aiutano molto nello sviluppo di algoritmi; altri accettano naturalmente la formulazione matematica; altri si occupano molto bene dei sistemi di comunicazione.

Quindi, una soluzione è quella di integrare alcune di queste piattaforme e beneficiare dei loro vantaggi. Per questo motivo, abbiamo pensato di sviluppare un framework integrato che faciliti la progettazione di veicoli non pilotati, inizialmente in simulazione, in grado di realizzare comportamenti autonomi, eseguendo compiti come la pianificazione dei percorsi, la localizzazione, la mappatura e la navigazione sicura, evitando ostacoli in ambienti aerei e terrestri. Siamo partiti valutando l'offerta, installando e testando alcuni middleware di robotica al fine di scegliere la piattaforma che permette le migliori capacità di integrazione con applicazioni robuste in ogni fase della modellazione di veicoli non pilotati da capo.

Facendo in questo modo, integriamo diverse piattaforme robotiche e strumenti per costruire un framework in cui è possibile avere tutte le funzionalità standard per tipo di veicolo non pilotato. Pertanto, quando arriva la necessità di un nuovo veicolo, è possibile creare e aggiungere un nuovo modello, con le sue peculiarità (caratteristiche e capacità). Così, il nostro framework robotico, chiamato UNISA-UVF, è stato progettato per facilitare la modellazione, la simulazione e il testing di veicoli non pilotati. UNISA-UVF è un sistema di robotica basato su sensori che utilizza approcci basati sul model-based e learning-based.

Inoltre, nel nostro framework, è possibile creare diverse versioni degli stessi veicoli con leggere variazioni nella descrizione della loro morfologia, facilitando così le missioni collaborative in cui sono necessari più UV per svolgerle insieme.

v. Introduction

The advances made in automatic control systems, artificial intelligence, and wireless communication make it possible to incorporate Unmanned Vehicles (UV) at home, office, and industry. Vacuum cleaners and logistics robots are commercially available with increasingly sophisticated autonomous functions. Also, some significant number of UV are using for monitoring, evaluation, and surveillance of different environments, including, among others, search and rescue operations, structural conformity assessments after disasters, environmental and biological ocean surveys and sampling.

Perhaps, the most important fact is that unmanned vehicles have revealed their enormous potential for action in a wide variety of military missions, pushing their development. Whereas, there is a need to have a keep going proof of concepts of new functionalities, to exploit its full potentiality also for civilian activities. For this reason, we are assembling a testing environment for unmanned vehicles, using the existing technologies, research results, and best practices, through reliable virtual simulations that help to prove innovative mechanical and control designs.

One of the central elements for the design, creation, development, and continuous testing of UV prototypes is the computational environment or platform for robotics, which contains aspects of hardware and software that allow conceiving, developing, and simulating UVs. Initially, with virtual prototypes, which after multiple tests of viability, operability, and feasibility, allow creating physical prototypes to continue with a new battery of tests until having a successful UV capable of fulfilling their missions safely and efficiently.

To date, these platforms exist in commercial and open-source versions and are useful because they make available to developers or amateurs in robotics common and most used functionalities, such as location, navigation, obstacle detection, mapping. However, those that seem to prevail in the market are those very specialized as flight simulators. As it is the case of X-Plane, a platform that simulates flights of various commercial aircraft, both in manual mode, i.e., by the control through pilot maneuvering and steering and in an autonomous way with the use of autopilots.

From the market perspective, the sectors of robotics overgrow, according to the International Federation of Robotics (IFR)[1], the sale in the industrial robotics sector will grow from USD 44.02 reached in 2018 to USD 69.140 billion in 2023, with a compound annual growth rate (CAGR) of 9.45%. It is

[1] https://ifr.org/downloads/press2018/Executive_Summary_WR_2018_Industrial_Robots.pdf

also expected that the service robotics market will grow from USD 11.27 billion in 2018 to USD 29.76 billion by 2023[2], with a CAGR of 21.44% (Research and market 2018) while the demand for unmanned aerial vehicles (UAV) was valued at USD 18.14 billion in 2017 projected to reach USD 52.30 billion by 2025, at a CAGR of 14.15% from 2018 to 2025[3] (Marketsandmarkets, 2017).

Geographically, the Asia Pacific region dominates the demand for mobile robotics, thus in 2016, it had a 32.20% share of the global market, thanks to its growing investment in the defense and logistics sectors. North America follows with its increasing demand for domestic robots, such as vacuum cleaning robots and floor cleaning robots, lawnmowers, and entertainment robots. It has also observed that the European market is increasing its demand for warehouse automation and the growing adoption of mobile robotics in various industries, such as medicine, defense, and agriculture.

The market ecosystem for Unmanned Aerial Vehicles (UAVs) are assorted and usually classified according to the type of UAV, its application, class, system, mode of operation, or region. United States of America (USA) firms, namely, General Atomics, Northrop Grumman, Textron, Boeing, are the ones that lead the frame, shortly after 3D Robotics. Other critical firms in the world are DJI (China), Parrot (France), and Aeryon Labs (Canada).

Although this growth of almost two figures, only traditional industries such as automotive, electrical, and electronics benefits from them. When these are incorporated efficiently into their production processes, they quickly increase their performance as it was reported in the Executive Summary World Robotics 2018 Industrial Robots. The well-known suppliers of industrial robotics mainly satisfy these "large account" customer automation needs.

For service robots, on the other hand, it seems that the higher speed of development benefits mainly from the progress of low-cost robotics, ready-to-use electronic equipment and components. In addition to the open-source software packages for standard functionalities that can be reused or tailored, with the support of active communities of practice around the world.

Another way to see the interest in the topic of mobile robotics and unmanned vehicles is through the number of references found in Google search.

[2] Robotics Market Research Reports & Consulting from ....
https://www.marketsandmarkets.com/robotics-market-research-112.html

[3] Unmanned Aerial Vehicle by 2025 |authorSTREAM.
http://www.authorstream.com/Presentation/Abhi.hole-3866937-unmanned-aerial-vehicle-2025/

By the end of November 2018, the following keywords search UGV, unmanned aerial vehicle, UAV, unmanned surface vehicle, USV, and unmanned submarine vehicle, UUV; results in 15,600 links and 615 links in Google academic in a variety of disciplines. If we review some of the first research articles shown, we see that the most cited ones have to do with mobile robots that work collaboratively, that is, teams of network robots that perform missions in a joint and coordinated manner. We can also see many articles that analyze the problems related to the use of UVs on aquatic surfaces and particularly underwater missions.

In addition to the technology, the possibilities of use, and the enthusiasm for the UVs, it requires regulation to accompanies the growth. The regulatory institutions of vehicles may expand the use of unmanned vehicles and create the necessary rules as they have the confidence and security in such a vehicle. Like investors to produce and market them, and end-users to incorporate UVs into their homes, offices, factories, lands, and other installations.

vi. Background and motivation

In the last decades, robotics has experienced a very marked development; robots tend to perform increasingly complex tasks with less human intervention. They become more autonomous and interact more and more with their environment to fulfill the mission that has been assigned to them, so the robot becomes an "intelligent machine." It means that robotics enthusiasts and researchers are moving in this direction, giving mobile robots advanced capabilities that allow them to reach a level of increasing autonomy in dynamic and unpredictable environments.

This trend in complex robotics development needs practical integration tools to implement valuable scientific contributions in the area of research into mobile robotics and related hardware and software technologies. For this reason, it is essential that robotics, developers, and implementers know how the models, methods, platforms, and algorithms available deal with the underlying physical and numerical paradigms of robotic environments. As well as develop the skills and technical abilities to interact with them.

The robotics community undertakes significant developments by launching engaging robotics platforms to support research and testing. Thus, we could say that the offer is available, with the right level of development also in the opensource arena. On the other hand, the Campania region in Italy hosts a critical number of companies related to aeronautics, vehicles' industry, and services. The companies related to modeling and delivering vehicle prototypes and components need to create virtual models to do many tests in a simulation.

There is ample and vital space for applied research matching the offer and the demand. In this case, an opportunity to take an increased and challenging hands-on in a complex field. Because small UVs is getting affordable with opensource platforms and cheap microelectromechanical (MEMs) system, it will also be necessary to transfer this know-how to partner academic institutions in Latin America.

Gazebo-ROS encapsulates all the robotics complexity, but to keep it general-purpose, performing very well with any accessory. Some cutting decisions have been made; for example, it is not a graphically rich platform, although it has necessary graphical tools that help in visualization, simulation, and control is not intuitive. It is easy to understand the choice that prioritizes performance over ease of use, also considering the flexibility and integration options with other robotic environments. Therefore, it requires substantial knowledge to take advantage of the full potential of the Gazebo-ROS platform. For this reason, it is known that the first steps and initial learning become complex and leave the feeling of being insurmountable. However, once this great difficulty of the initial barrier is overcome, a minimum level of autonomy can be reached to develop confidently and safely in this environment.

Due to the above and even though there are several books and tutorials, we believe that our work on unmanned vehicles seen from the mechanics and robotics perspective at each stage of development could help to understand the platform and the behavior of UVs on it. Therefore, in this thesis work, we focus on the configuration of an open and profitable code framework for unmanned vehicles, which will allow UNISA mobile robotics researchers and students to enter the world of Gazebo-ROS modeling and control of unmanned vehicles.

vii. Thesis Objectives and Organization

The objective of this thesis work is to develop a framework for the creation and simulation of unmanned vehicles with functionalities such as location, navigation, and control in individual and group missions on an open-source platform. The chosen platform is Gazebo-ROS for the simulation and control of unmanned virtual vehicles. Besides, we have considered it appropriate to interconnect and integrate the robotic platforms Simulink-MATLAB and X-plane, recognized as standards in the robotics and aeronautics industry, respectively. By doing so, we can reuse existing models and UV systems within the university as in the robotic community.

So, the framework UNISA-UVF will allow the mechanical department of UNISA to build and test land and unmanned air vehicles in simulated environments, with the possibility of testing prototypes of physical vehicles with much less effort. The open-source environment, Gazebo-ROS, is designed to be completely reusable and interoperable, now with MATLAB Simulink and X-plane, but is open to adding other ones.

To limit the scope of this thesis work, we remain in the field of mobile robotics, focusing on the control and engineering problems of unmanned vehicles (UVs) in the ground and aerial environments that perform autonomous operations over UNISA-UVF framework, independently or in groups. The structure of the thesis is as follows:

- In chapter one, we present the literature review of unmanned vehicles and the available computational robotic structures that support them. For each type of unmanned vehicle, we show its characteristics, common usages, and evolution to nowadays.
- In chapter two, we present the open-source Gazebo-ROS robotic platforms selected as the leading platforms in the development of UNISA-UVF, a detailed description of the main characteristics and functionalities, as well as, those of the integrated commercial platforms Simulink-MATLAB and X- Plane.
- In chapter three, we present the characteristics of autonomous vehicles developed for the simulations that we test in our framework, as well as the kinematic, dynamic, and control models of the Unisa_bots, a UGV of a differential type and a fixed-wing UAV.
- In chapter four, we present some useful robotics techniques and how Gazebo-ROS implement them.
- Finally, in chapter five, the practical use of the UNISA-UVF framework is presented in two case studies, the first Unisa_Gbots is a group of UGV in a simulation that performs a SLAM of a free of use 3D indoor environment; the second UAV (rosmilvus) a new unmanned aerial vehicle in a simulation that delivers Dubin's waypoint navigation.

CHAPTER I
Literature Review

I.1 Introduction

This chapter presents a literature review related to the research objective of this thesis, starting with the study of the state of the art of mobile robotics with a focus on unmanned vehicles, followed by the analysis of robotic platforms with a focus on opensource that allow the modeling and control of unmanned vehicles. In both cases, we make a brief definition of the concepts, methods, and related techniques, to put the reader on the terms used and the meaning we have taken in each case.

The discussion and conclusion of the chapter have an opportunistic approach since it considers the views of mechanical engineers and robotics professionals on the relevant aspects of the design, modeling, control, and in general, in the processing of information related to Unmanned Vehicles. Within these two perspectives, we can identify and evaluate the best alternatives to build the Unisa framework for Unmanned Vehicles (UNISA-UVF).

The vision of mechanical engineers in robotics pays special attention to the physical and mechanical aspects of robots and their interaction with their environment. Thus, for example, to the selection and management of the most suitable sensors and actuators in mobile robots according to the missions they must fulfill. In the same way, the professionals in robotics pay more attention to the computational platform of the equipment embarked on the mobile robots as the central or distributed systems, like their integration and communication capacities, the algorithms, and programming languages. Therefore, they ignore or overlook the problems related to the generation of energy, the electrical distribution, and some of the kind.

I.2 Unmanned Vehicles

The term "Unmanned Vehicle" refers to the capability of a vehicle to navigate in the environment for which it was designed (ground, air, water) without an onboard human presence. The type and level of autonomy vary, ranging from the absence of automation to full automation. The form and degree of control are implemented based on their detection system, and it usually relies on the usage of model-based and learning approaches to increase their levels of driving over time (Rivera Z.B et al., 2016).

Unmanned vehicles are within the realm of mobile robotics; they are robots with motion capabilities in different environments. Its roots include many disciplines of engineering and science, from mechanical, electrical, and electronic engineering to computational, cognitive, and even social sciences, due to the increased interaction with human beings in daily life activities. Their essential components include at least one controller, a power source, a software or control algorithm, some sensors, and actuators.

The following definition of a mobile robot by Arkin (1998) identifies their feature and purpose: "*An intelligent robot is a machine able to extract information from its environment anduse knowledge about its world to move safely in a meaningful and purposive manner.*" (Chapter I, pg.2). This definition easily contrasts with reality, because unmanned mobile robots have been used in difficult, dangerous, and highly unpleasant tasks to be carried out by human beings, either because the costs of accessibility, safety, survival are high, or fatigue, time, or unpleasantness are unsupportable.

Therefore, nowadays, missions for ground, aerial, over and underwater-unmanned vehicles fulfill tasks like monitoring infrastructures as bridges, canals, offshore oil, and gas installation. Also, mobile robots are in inhospitable and remote environments where they can be impossible for a human being to go, like Mars.

Depending on these environments of action, a first classification includes:

- Terrestrial robotics covers both wheeled vehicles operating on regular surfaces (roads, parking lots, homes, offices) and field robotics, which deals with off-road vehicles or lands for agriculture.
- Aerial robotics includes all flying vehicles (mainly fixed-wing and rotor drones), both at low altitudes and at high altitudes and in all types of environments (includes interiors for micro and mini versions).
- Finally, marine robotics includes submarines and autonomous vessels.

Recently, the degree of automation increases in the automotive industry, employing "intelligent vehicles," which deals with the mobility of people and goods on paved surfaces commonly; however, there are some critical off-road developments. For these cars, the adapted equipment includes a high quantity and preformat automation components, and usually, parallel steering and traction systems for automation since levels of security required to transport humans are higher.

In addition to mobility, we can appreciate that the size and nature of the working environments have essential characteristics to consider. Thus, an immediate consequence is that its creation, testing, and implementation, require quite a consistent means (robots in themselves, experimental sites, computer infrastructure and possible infrastructures for command posts, specialists and researchers), both in costs and in the level of knowledge in various domains related to the type of mobile robot and the environment of action. Therefore, there is an essential "knowledge" and "know-how" to master, from conception to experiments passing for implementations.

The areas of knowledge involved in the field of mobile robotics are Mechanical engineering, responsible for the design of vehicles, in particular, the mechanisms of motion. Computer science, accountable for visualization, simulation, and control with algorithms for detection, planning, navigation, control, mapping. Electrical engineering, responsible for integrating systems, sensors, and communications. Cognitive psychology, perception, and neuroscience, for the study of biological organisms to understand how they analyze information and how they solve problems of interaction with the environment. Finally, Mechatronics, which is a combination of mechanical engineering with computing, computer engineering, and electrical engineering.

The expansion of robotics has shown a significant transformation in its scope and dimensions, especially since the new millennium, in some fields, it has a level of maturity, backed by advances in related technologies. However, seen from the spectrum of scientific challenges, we can appreciate that being broad, multidisciplinary and complex environment there are still many unanswered questions, or they need to be perfected; among others, we have the autonomy of the movement, freedom of decision, the conception and management of "system systems", the human-machine interaction.

As it is known, innovation drivers can come from the side of technology that allows the development of a specific type of mobile robots, or from the knowledge side that encourages the development of technology to materialize or supports the new findings. For this reason, the degree of development of mobile robots varies significantly according to the time and the impetus with which the research topics, technologies, and initiatives were approached, the same ones that allowed their progress until now. To describe these achievements in mobile robots, we will do a broad classification according to the environment where they operate (ground, air, or water). We include the "intelligent vehicles" within the mobile ground robots, because to date, it is the only environment in which tests have been carried out by transporting human beings.

The standard classification for unmanned vehicles, in the academic community, are Autonomous Ground Vehicles (UGV), autonomous land vehicles (ALV), or mobile robots for vehicles traveling on land, which are also known as "Intelligent vehicles." Autonomous Aerial Vehicles (UAV) generally classified as rotors and fixed-wing for those who move in the air. Autonomous Submarine Vehicles (UUV) or Unmanned Surface Vehicles (USV) for those traveling below and above the surface of the water. The following paragraphs cover, for each of these three groups of unmanned vehicles, a brief definition, uses, characterization, and evolution.

I.2.1 Unmanned Ground Vehicles (UGV):

Brief definition

The unmanned ground vehicle is a vehicle that operates while in contact with the ground and without a human presence on board. Its development begins as an application domain for Artificial Intelligence research at the end of the 1960s; the initial purpose was to recognize, monitor, and acquire objectives in military environments. In the civil area, they are used for disaster management responses throughout the world (Murphy, 2014).

Characterization

In the realm of mobile robotics, the locomotion mechanisms on land can be diverse and rely on the choice made during the conception and implementation. Thus, the robots can walk, jump, run, slide, crawl, and roll. The mechanism of locomotion on ground preferred and chosen by researchers, and the robotics industry has been by far the wheel, being mechanically more straightforward and more efficient, especially on flat surfaces.

For this reason, key components that influence the total kinematics of the UGV are undoubtedly the wheels, so the selection and the arrangement of these in the vehicle are essential. Four types of wheels are commonly used (see Figure I.1), with advantages and disadvantages, and have very different kinematics, as:

- Standard wheel: two degrees of freedom, rotation around the wheel axle (usually motorized), and one point of contact.
- Rotating wheel: two degrees of freedom, rotation around a controllable displaced joint.
- Swedish wheel (Swedish): three degrees of freedom, rotation around the wheel axis (usually monitored), the rollers or bearings, and the point of contact.
- Ball or spherical wheel: three degrees of freedom with technically tricky realization.

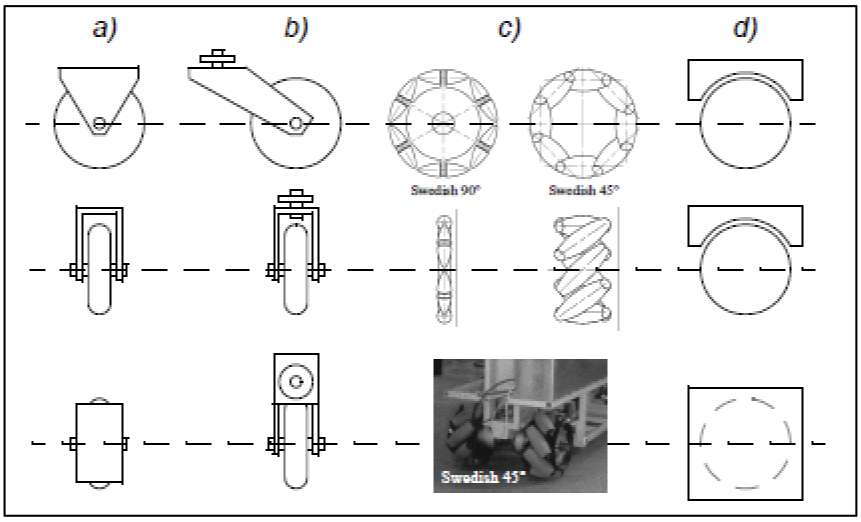


Figure I.1 *Common Robotics Wheels*
(Source: Introduction to Autonomous Mobile Robots (p .36)

The common UGV steerability based on wheels are depicted in Figure I.2, for highly directional wheels, usually standard wheels are fixed in primary rotational axis, or combined with castor wheels. Standard wheels could be configured as traction and direction, with the center of rotation passing through the contact patch with the ground, while the castor wheel rotates around an offset axis, causing a force to be imparted to the robot chassis during steering. There is a need for Swedish wheels to have an omnidirectional UGV that thanks to the rollers or bearings of its surface allow them to move in all directions; there are other ways but has increased complexity.

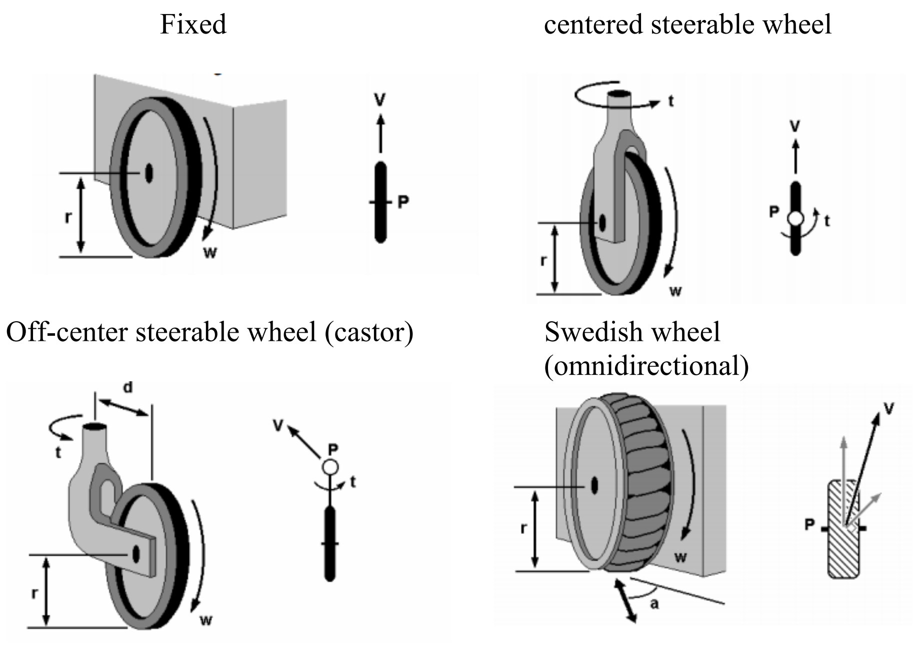

Fixed
centered steerable wheel
V
P
r
w
t
Off-center steerable wheel (castor)
Swedish wheel (omnidirectional)
d
a

Figure I.2 *Common Wheels Steerability*
(Source: Adapted from Introduction to Autonomous Mobile Robots)

According to the number, selection, and disposition of these wheel types, the UGV will have different degrees of freedom, which will characterize its maneuverability, how easily roll in a straight line, or make turn motions. The Unicycle type robot has a simple and known mechanical and electronic structure, which makes it the preferred one for its kinematics in laboratory tests. It has two conventional fixed wheels arranged on the same axis and controlled independently, each one that allows the direction and synchronization, and a locator or castor wheel conveniently arranged to give stability. Its traction-steering system allows controlling the linear and angular speed independently.

Two other configurations also used are the tricycle and the quadricycle or quad, so-called by the number of wheels used, generally conventional. The tricycle resembles the unicycle with the difference that the traction-steering function is given in the steerable centered front wheel, while the rear wheels are fixed on the same axis. It also has simple kinematics but can lose traction during movement when its center of gravity is too close to the permitted limits, making it unstable and unreliable in the estimation of the vehicle's position. Thus, it is usually used to transport heavy loads to low speed.

The Ackerman steering quadricycle, have the axes of the two front steerable wheels intercepting in an Instantaneous Center of Rotation (ICR or ICC). The ICR point belongs to the projection of the common axis of the rear fixed wheels. In this way, a set of concentric arcs is observed in the plane traced by each wheel around this point ICR, whose instantaneous velocity vectors of each wheel are the tangent to these arcs. Its kinematics, mechanical structure, and control electronics are not as simple as the previous one, but it provides more excellent stability, prevents slippage in the wheels, and therefore reduces odometry errors, especially for all-terrain vehicles.

The efficiency of wheeled robots depends to no small extent on the quality of the terrain, particularly the smoothness or hardness of the ground, the type of surface (flat or non-flat), the number of obstacles (free or dense). Conventional wheeled vehicles usually move on flat and hard enough terrain, while on non-flat, irregular and non-dense terrains, track wheels with gears and adapted diameters are required. For example, at home, for indoor floor cleaning missions, the mobile robotic vehicles need an appropriate configuration to move on polished and carpeted floors in general. Nevertheless, in devastated outdoor places, the ground mobile robots used for monitoring will have a diverse configuration to be capable of adapting to irregular terrain, with debris and other conditions that will limit its displacement.

Another essential feature is the traction and steering system, which is related to the arrangement of wheels in the vehicle. Three types are depending on how they are linked to the axes; so, the traction and direction can be:

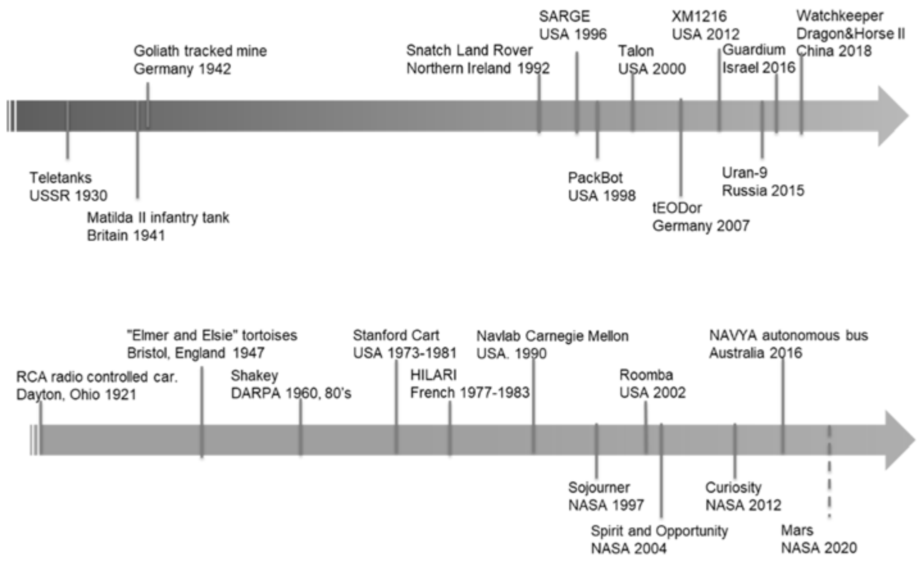
- On independent axles, traction on the rear wheels, and steering control on the front wheels, the accuracy is related to the level of adherence of the wheels because its mass is negligible concerning that of the complete vehicle, and the turning radius is high (distance to ICR).
- In the same axle or differential traction, using independent motors in the wheels of the same axle and idle wheels or castor arranged appropriately, this configuration allows turns even of the size of the vehicle. Engines with the same characteristics are usually suggested to simplify the control.
- On all axes or integral, used in vehicles that require high adherence, the odometrical system is more complicated due to the uncertainty in the associated turning radius.

There are also unique configurations that associate the systems described above in order to expand the functionality of the robot, granting it more degrees of freedom or improving the errors related to wheel slippage and odometrical measurements. However, the associated mechanics, electronics, and computing will be more complex, given that the control and direction systems are associated with the algorithms of local motor control and the mechanics associated with them.

Evolution and Usage

Autonomous unmanned vehicles have had a significant development in the military field since the beginning of World War II, the remote tanks of the USSR "Teletanks" were used in the Winter War (1939-1940) against Finland, and the beginning of the Eastern Front after Germany invaded the USSR in 1941. While the Germans used the Goliath mine with 60 kg of explosive charge directed through a control cable. Nowadays, the Unmanned Ground Vehicles are mostly used for Intelligence, Surveillance, and Reconnaissance (ISR), Search & Rescue, Combat Support, Transportation, Explosive Ordnance Disposal, Mine Clearance, Firefighting and others (Counter CBRN, Hazmut).

The following timeline shows some of the military UGVs that have been developed and deployed by several armies around the world, it is certainly not a complete list of them but shows the diversity of army efforts (upper Figure I.3). The investments in research for military usage of UGVs are always increasing, in order to make them more autonomous, to operate on their own (through artificial intelligence) for long periods, capable of carrying on large payloads or being very light and small to enter through the enemy lines.



Goliath tracked mine Germany 1942
Snatch Land Rover Northern Ireland 1992
SARGE USA 1996
Talon USA 2000
XM1216 USA 2012
Guardium Israel 2016
Watchkeeper Dragon&Horse II China 2018
Teletanks USSR 1930
Matilda II infantry tank Britain 1941
PackBot USA 1998
tEODor Germany 2007
Uran-9 Russia 2015
"Elmer and Elsie" tortoises Bristol, England 1947
Stanford Cart USA 1973-1981
Navlab Carnegie Mellon USA. 1990
NAVYA autonomous bus Australia 2016
RCA radio controlled car. Dayton, Ohio 1921
Shakey DARPA 1960, 80's
HILARI French 1977-1983
Roomba USA 2002
Sojourner NASA 1997
Spirit and Opportunity NASA 2004
Curiosity NASA 2012
Mars NASA 2020

Figure I.3 *"UGVs lifetime in Military and Civilian fields (upper & lower)" (Source: self-drawing based on a literature review)*

While lifetime UGVs for civilian and commercial applications that have been developed and deployed for different purposes like agriculture, manufacturing, mining, supply chain, and for Aerospatiale missions (Figure I.3.

lower) are mostly created in academia for research purposes, in the industry, UGVs are used for map building, transporting materials and goods, stock scanning, and inventory taking with autonomous vehicles, forklifts, and conveyors.

The first car was unmanned and controlled wirelessly via radio created by a subsidiary of General Electric back in 1921, then “Elmer and Elsie” tortoises, considered as the ancestors of ground robots and "intelligent" weapons, because in order to identify the sources of dim light and approached them, they had capabilities of locomotion, detection, and evasion of obstacles. They are also recognized as the pioneers of Artificial Intelligence (AI) because of their ability to react to stimuli as "conditioned reflex." However, Shakey, created in 1966-67, is considered the pioneer of unmanned vehicles. It was able to navigate by himself from one room to another and even to transport an object; it established the functional and performance baselines for mobile robots (Nilsson, 1969).

Shakey carried on several sensors: a camera, a distance measuring device, and tactile sensors to perceive obstacles, actuators as step by step motors. It was the pioneers of mobile robots able to "feel" his surroundings (requiring enormous computational resources for calculation at that time). It served as a testbed for AI's work funded by DARPA at the Stanford Research Institute (SRI) (Nilsson, 1969).

Both Sharkey and Bristol turtles defined the research agenda of Artificial Intelligence (AI) in areas like planning, vision, conditioned reflex processing, and natural language (Flynn, 1985). The Sharkey mobile robot continued his evolution, by the end of the 80s, was an eight-wheel all-terrain with standard hydrostatic steering, able to move on roads and in rough terrains; as well as, had incorporated the electronics and software for an unmanned navigation and objectives search (Everett, 1996).

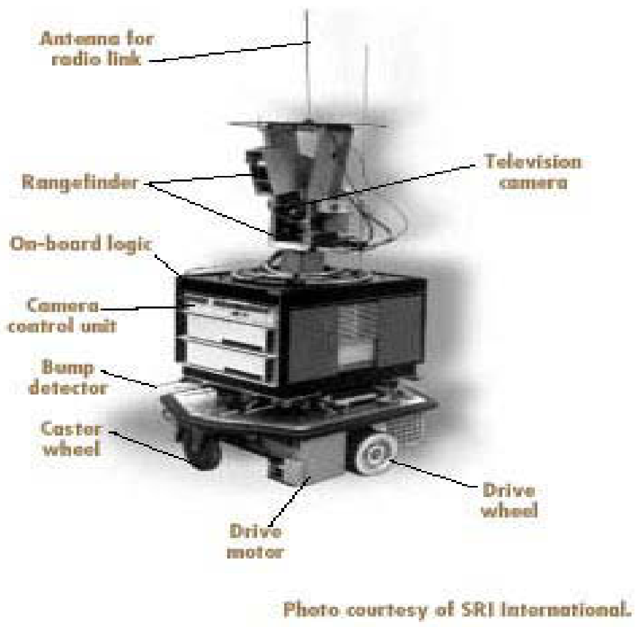


Figure I.4 *"Sharkey" mobile robot - First Unmanned Robot (Source: http://www.historyofinformation.com)*

NASA, in 1997, launched the "Sojourner" rover (a type of UGV) to explore, analyze, and photograph the surface of Mars. It was active for two months on this planet in a radius of action of 20 meters around the landing and continuous communication platform called PathFinder. After "Spirit" and "Opportunity," two twin rovers were launched in January 2004 to explore a broader area. Then "Curiosity" was sent in August 2012 in the mission Scientific Laboratory on Mars, and it is expected to send "Mars2020" by 2021 to continue the mission of exploration to know if there was life on that planet and if it is possible to send human beings.

Table I.1 *Rovers Characteristics and mission*

Rovers	**Structure**	**Weight (lbs)**	**Vel. Max(mph)**	**Scientific Inst.**	**Mission**
Sojourner	Chassis with solar panel Six independent metal drive wheels studded with nails bogie tipper suspension system	2. 3	0.02	two	Move, explore and photograph within a radius of 20m around the communication platform
Spirit and Opportunity		374 each	0.1 each	5 each	Explore and look for evidence of water on Mars
Curiosity		1982	0.09	10	Discover if Mars ever had everything necessary for life: durable water and adequate chemical ingredients
Mars2020	?	?	?	7	Look for signs of past life or current possibilities, see if a human being can explore it one day

Source: based on NASA public website information

The rovers sent to Mars by NASA have six steel wheels with different types and sizes of steel tines, to improve grip, has solar panels on the top and a bogie-type swinging suspension system that gives excellent freedom of movement, being developed in each new rover launched on Mars (see Figure I.5). With this type of leaned suspension, the rovers can overcome an obstacle one and a half times greater than the diameter of its wheels, moving at a cruising speed of 40 cm per minute in Sejourner, reaching up to 180 cm per minute in Curiosity.

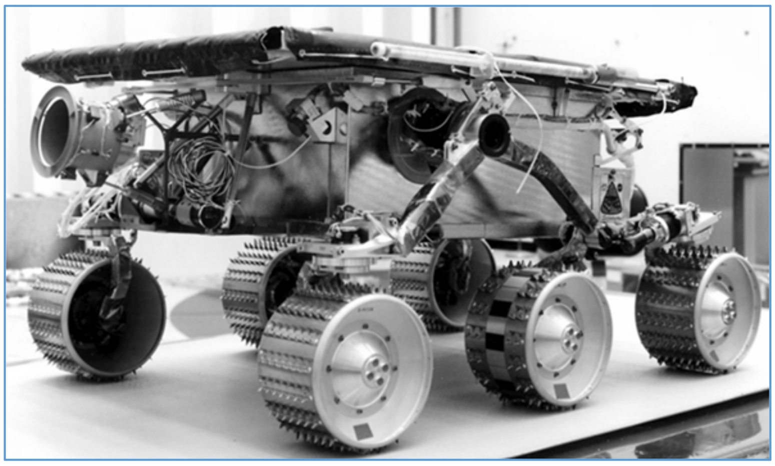


Figure I.5 *"Sojourner" - NASA First Rover on Mars*
(Source: NASA website)

The size of the rovers was growing and increasing in weight, going from only 11 kg to 900 kg to support an increasing number of communications, sensing, sampling, and research equipment, going from only 2 to 10 scientific instruments. The technology present in the rovers are those of propulsion to provide them with energy to reach Mars and conduct long-term studies, sources of power to increase the efficiency of the central system and its subsystems, telecommunications to send control commands and to receive data in real-time and in large quantities. Finally, avionics and software engineering, to provide the electronics, computing, and commands necessary for the operation of the spacecraft and its subsystems.

As we can see, there is an essential effort from the scientific community, but the state-of-the-art technology remains on research centers of these crucial institutions and universities. Therefore, it is interesting to see how the Defense Advanced Research Projects Agency (DARPA) of the Department of Defense of the United States, a pioneer institution in the technological development related to robotics, becomes an essential driver of the sector. Because to increase its research efforts, launch challenge programs in order to involve external civil agents at a global level. Thus, DARPA provides excellent resources for research and development through challenges in the creative mobile robotics sector from the late 1960s to the present.

The competitions of intelligent unmanned vehicles "Grand Challenge" in 2004 and 2005 and "Urban Challenge" in 2007 of DARPA are followed by other initiatives aimed at promoting the development of technologies and research topics related to mobile robotics. Both competitions of DARPA unmanned vehicles are based on the levels of autonomy defined by the Society of Automobile Engineers (SAE), a global association of engineers and related technical experts in the aerospace, automotive and commercial vehicle industries. The SAE defined five levels of autonomy for driving cars, depending on the degree of capabilities reached in four categories, namely the execution of the steering and the control of the acceleration; the monitoring of the driving environment; the alternative performance of the dynamic driving task; and system capacity or driving modes.

The Urban Challenge was the more significant trigger from where it is looked for because it pushes competitors to manager complex maneuvers required to overcome obstacles in a dynamic, uncontrolled environment (other vehicles, pedestrians). In addition to respecting a series of traffic rules imposed for the city as for any other type of vehicle, such as negotiating the passage at an intersection, respecting the signals of the traffic lights, parking only in permitted places, among others.

With the DARPA challenges, the capacities of the UGVs were increased, and a real interaction of unmanned vehicles under challenging conditions like the desert (Grand Challenge 2005) and in the complexity of an urban environment (Urban Challenge 2007) had been tested for the first time. Also, the rovers launched by NASA show the crucial advances of these vehicles in completely unsafe environments, having examined all the technologies implemented in the design and modeling of the mobile robot itself as in the complementary systems to allow navigation, exploration, communication and data analysis with a significant payload of scientific instruments. By the same period, another DARPA Advanced Ground Vehicle Technology (AGVT) project was created, specialized in military applications (Gage, 1995).

However, despite these critical advances, today, only some vehicles can operate autonomously under certain conditions, as is the case of autonomous vacuum cleaners inside homes or offices and the use of some cargo robots, who perform some logistical maneuvers. In order to generalize the use of mobile robots, unmanned vehicles, and "intelligent vehicles," more research and tests must be carried out to solve all the theoretical and technical problems, to increase their level of accuracy in sensing and actuating. Also, the intelligence for making timely and accurate decisions in a highly interactive environment in number and frequency, such as urban environments and public roads, in order to ensure safe driving in all conditions.

I.2.2 Unmanned Aerial Vehicles (UAV):

Brief definition

The Unmanned Aerial Vehicle Systems (UAS) is known by different names and acronyms, such as an unmanned aerial vehicle (UAV), flying robot, remotely piloted vehicle (RPV), or just "drone. " It is an airplane characterized by the absence of a human pilot, which can be controlled through a computer system on board and remotely through a navigator. This equipment respects the same procedures of a conventional airplane.

When the ground control station (GCS) is incorporated, and a communication data link for command and control is usually called UAS. However, other components are considered critical, such as autopilots, navigation sensors, image sensors, mechanical servos, and wireless systems.

Characterization

There are mainly two types of Unmanned Aerial Vehicles by their kind of take-off; those that can do a vertical takeoff and those that do not, in the second group, are the fixed or flexible wing. In general, fixed-wing UAVs excel due to their speed of travel, resistance to external disturbances such as wind gusts, load capacity, while the rotors (usually Quad-rotors) for their maneuverability, vertical flight capacity, indoor flight capabilities. It is possible to see in the literature that is being built hybrid UAV to take advantage of the characteristics of both, so there are airplanes with rotors strategically willing to give such a hover or capacity vertical takeoff and landing.

You can also find in the literature a commonly used classification that takes as reference the maximum weight to the takeoff (MTOW) of the unmanned aerial vehicles, as well as the comparative table that follows (Table I.2), obtained from the international organization UVS, allows to perceive the weight, range, altitude and flight duration of the UAV. In the table, the essential features reserved for particular tasks in the military field are listed, where research and development of UAV have more time and have reached a certain degree of maturity. Recently the European Airbus launch Zephyr, which is the world's leading, solar-electric, stratospheric UAV, combines the endurance of a satellite with the versatility of a UAV [4]. The Airbus is working on a new large model Zephyr T with a wingspan of 33m and weighs 140kg to accommodate payloads with larger masses.

[4] Zephyr - UAV - Airbus. http://www.airbus.com/defence/uav/zephyr.html

Starting from the most basic classification, according to aircraft weight, that can go from Micro air vehicle (MAV) weighing less than 1g to heavier ones around 5t. They have specific physical components, materials, and shapes, powerful propulsion technologies, control systems (electronic, environmental), varying functions, and feature sets. For example, a research field in MUAVs function is dealing with Flapping-wing ornithopters, imitating birds or insects, exploring miniature optic-flow sensors. To realize how to transmit data to neuromorphic, they are testing chips able to treat optic flow as well as light intensity discrepancies.

It is also possible to classify the UAVs by their level of autonomy, being able to go from not being autonomous until carrying on board a complete intelligent autopilot. If there is no level of autonomy, the flight path should be planned and scheduled in advance, in order to guide and control it continuously from a dedicated command post on the ground, called the ground segment. It is linked to the air segment by systems of communications. Even though the UAV has complete autonomy, there are tasks carried out in the ground segment, such as the definition of the mission and the supervision of the development of the mission. In the air, on the other hand, the tasks related to on-board sensors and actuators are carried out, to obtain information and control the flight, respectively.

As in the human-crewed aircraft, to transmit and receive digital signals a data links, it is always necessary to have onboard controllers, those low-level systems allow telecommunications between the aircraft and the control station; while the protocol that establishes the transmission rules governs the communication. (De Simone M.C. & Guida D. 2018). Unmanned Aerial Vehicles (UAVs) implements full or half-duplex systems to send control signals and receive telemetry signals. The onboard computer (generally with GPS navigation) is connected to the aircraft control system to be capable of flight and operating system control. Usually, it includes one or more control stations, communication links, data terminals, launch and recovery systems, pieces of equipment, ground support, and an air traffic control interface.

Table I.2 *Classification of UAVs by Flight capacity*

	Mass (Kg)	Range (Km)	Flight alt. (m)	Endurance (h)
Micro	< 5	< 10	250	1
Mini	< 20/25/30/150	<10	150/250/300	<2
Tactical				
Close Range (CR)	25-150	10-30	3000	2-4
Short Range (SR)	20-250	30-70	3000	3-6
Medium Range (MR)	150-500	70-200	5000	6-10
MR Endurance (MRE)	500-1500	>500	8000	10-18
Low Attitude Deep Penetration (LADP)	250-2500	>250	5-9000	0.5-1
Low Altitude Long Endurance (LALE)	15-25	>500	3000	>24
Medium Altitude Long Endurance (MALE)	1000-1500	>500	3000	24-48
Strategic				
High Altitude Long Endurance (HALE)	2500-5000	>2000	20000	24-48
Stratospheric (Strato)	>2500	>2000	>20000	>48
Exstratospheric (EXO)	TBD	TBD	>30500	TBD
Special task				
Unmanned combat AV (UCAV)	>1000	1.5	12000	2
Lethal (LET)	TBD	300	4000	3-4
Decoys (DEC)	150-250	0-500	50-5000	<4

Source: Unmanned aerial vehicles with international UVS information (MTOW)

The applications of UAVs in military environments are at the service of intelligence, surveillance, and reconnaissance (ISR), combat operations, and other related. In civil and commercial settings, they are used for precision agriculture; remote sensing, security and border management; monitoring of inspection, traffic, public structures and roads; photography, media coverage and film production; topography and mapping, research and conservation of wildlife, scientific research; shipment of packages, among many others.

Developing a good architecture, a model is a crucial element; it serves to evaluate the full functionality, reducing ambiguity, and increasing the robustness of the system. From the structure (model), for a new aircraft, it is important to do many tests in simulation environments as much as possible to test the model and each component, understanding their behaviors.

Evolution

From the early times, the Unmanned Aerial Vehicles (UAV) technology has been used for military purposes, with a large size and purpose range. The latest UAVs generations have sophisticated and miniaturized sensors, allowing remote control of the aircraft, to complete their mission without losing lives. Within the time, the UAVs are increased their usage by different military forces, by government agencies and by businesses (De Simone M.C. & Guida, D. 2018). For example, in the United States, government agencies use some UAVs, like the RQ-9 Reaper, to patrol the borders of the nation, to explore and identify fugitives and migrants.

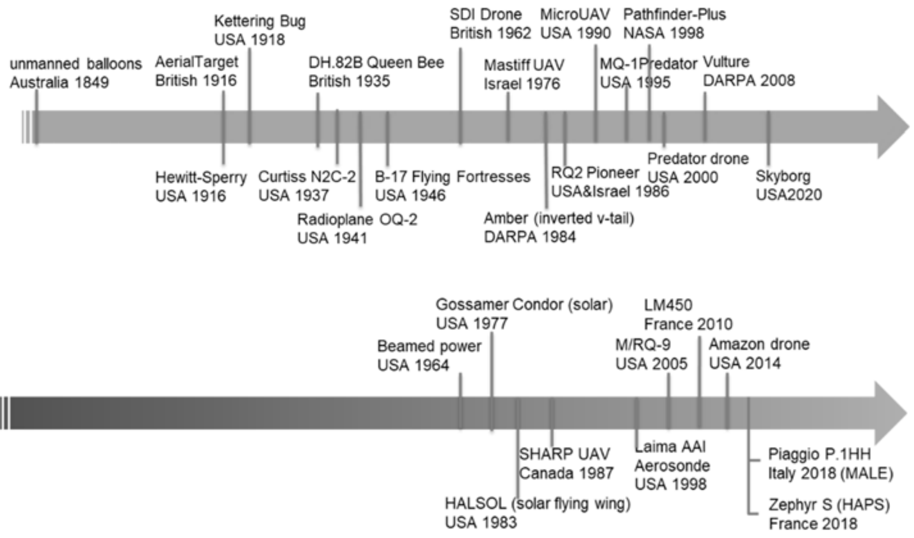

Kettering Bug USA 1918
unmanned balloons Australia 1849
AerialTarget British 1916
DH.82B Queen Bee British 1935
SDI Drone British 1962
MicroUAV USA 1990
Pathfinder-Plus NASA 1998
Mastiff UAV Israel 1976
MQ-1Predator USA 1995
Vulture DARPA 2008
Hewitt-Sperry USA 1916
Curtiss N2C-2 USA 1937
B-17 Flying Fortresses USA 1946
RQ2 Pioneer USA&Israel 1986
Predator drone USA 2000
Skyborg USA2020
Radioplane OQ-2 USA 1941
Amber (inverted v-tail) DARPA 1984
Gossamer Condor (solar) USA 1977
LM450 France 2010
Beamed power USA 1964
M/RQ-9 USA 2005
Amazon drone USA 2014
SHARP UAV Canada 1987
Laima AAI Aerosonde USA 1998
Piaggio P.1HH Italy 2018 (MALE)
Zephyr S (HAPS) France 2018
HALSOL (solar flying wing) USA 1983

Figure I.6 *"UAVs lifetime in Military and Civilian fields (upper & lower)"*
(Source: self-drawing based on a literature review)

The first military unmanned aerial systems (UAS) go back to the year 1916 (see Figure I.6 upper), the ' AerialTarget' of the British professor AM Low, and the airplanes Hewitt-Sperry Automatic of the Wright brothers also in 1916. Then many developments have been seen, mostly related to war conflicts. Later, in 1988, Amber (see Figure I.7), the first unmanned aerial vehicle of resistance, flew more than 38 hours straight to 25,000 feet; it had digital flight control, microprocessors, and satellite navigation. In the same year, the DARPA/Navy unmanned vehicle program was initiated to serve as a stand-alone test-bed vehicle and then integrated with specific missions such as mine location, avoidance, and remote monitoring. More recently, the Air Force Research Laboratory announced "Skyborg," a UAV with artificial intelligence that by the end of 2023 could take off and land autonomously, fly in bad weather, and avoid other aircraft, terrains, and obstacles. It is expected to be combat-ready at this time.

Unmanned aerial vehicles are having essential drivers that are boosting the supply, demand and research to increase the capabilities of UAVs or drones (as they are commonly known), as well as to improve the models, methods, and techniques of modeling, control and simulation, through enhanced performance of the associated hardware, software and telecommunications components. For these reasons, the UAV, especially the rotors, have been popular commercial devices, which due to their low cost, have been used for distraction purposes, to take photographs or other tasks in private events, even though this depends on the regulation of each country or region. However, there are a professional use drones, which have better flight routines (automatic flights, GPS navigation, corrections by altitude) and can carry onboard components with better technological features such as high-resolution cameras, thermal sensors, gas, multispectral, radars, and higher performance batteries, some of them are draw in the Figure I.6 (lower).

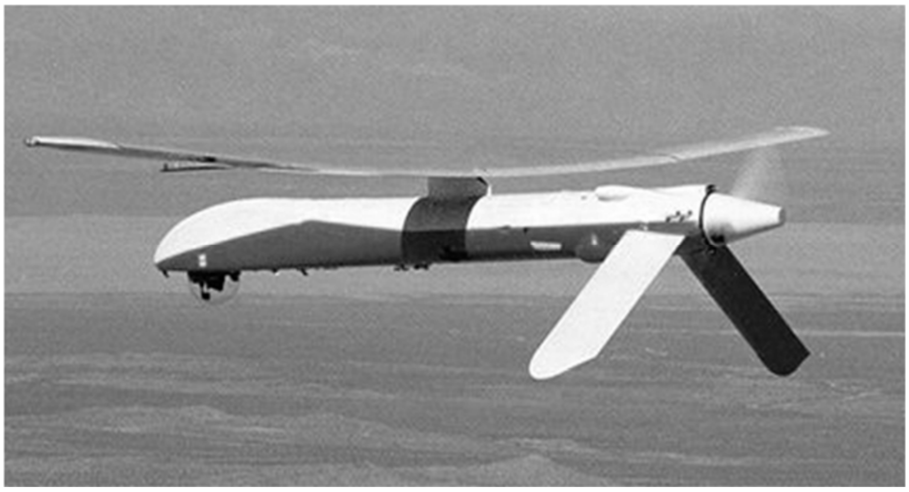


***Figure* I.7** *"Amber" - First Unmanned Aerial Vehicle*
(Source: DARPA https://www.darpa.mil/about-us/timeline/amber-predator-golden-hawk-predator)

Also, the unmanned aircraft is intended to fly in the stratosphere, Zephyr S (see Figure I.8), and Zephyr T, the new models of the European Airbus they operate exclusively with solar energy, flying about 70,000 feet above conventional air traffic[5]. They are cataloged as a HAPS (High Altitude Pseudo-Satellite), the wingspan of 25m and 33m, and weighs less than 75kg and 140kg, respectively, give them the ability to fly for months at a time.

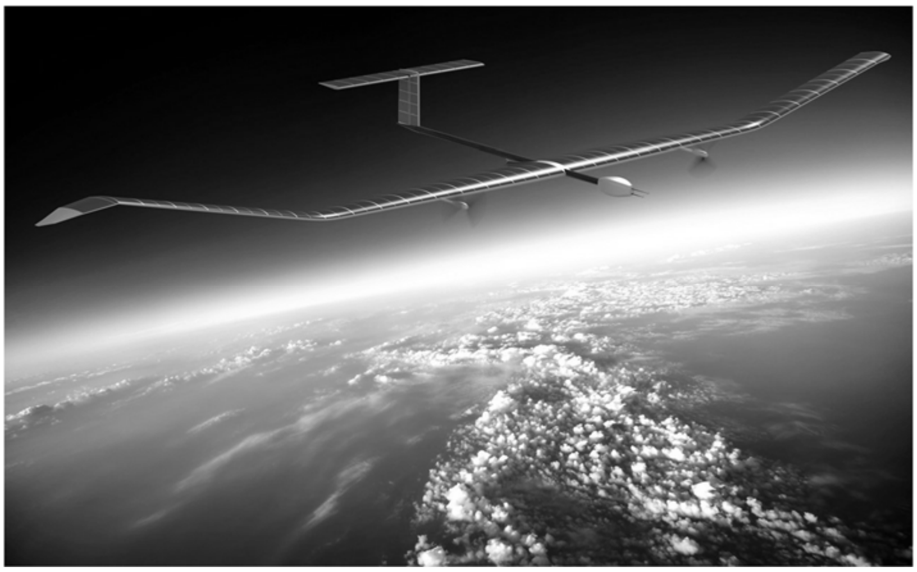


Figure I.8 *"Zephyr S" - First solar-electric UAV pioneering the stratosphere (Source: AIRBUS https://www.airbus.com/defence/uav/zephyr.html)*

The growth projections of the unmanned vehicle market were valued at USD 18.14 billion in 2017 and are projected to reach USD 52.30 billion by 2025, with a CAGR of 14.15% from 2018 to 2025, according to Marketsandmarkets. The market leaders are the North American groups occupying the first places, among them General Atomics, Northrop Grumman, Textron, Boeing, and 3D Robotics, followed by DJI from China, Parrot from France, and Aeryon Labs of Canada.

The latent potential of unmanned aerial systems for civil applications is and has always been perceived as favorable. However, sustainability and operational complexity remain essential. The experiences of the missions with unmanned aerial vehicles in the military sector are more frequent and with significant technological advances. In the civil areas, they are developed in the field of university research, apart from the light transport aircraft, that is currently venturing into the best distribution sector established in some developed countries, and there has been significant progress in the design of robust control software and hardware in this specific type of UAV as quad-rotors.

[5] https://www.airbus.com/defence/uav/zephyr.html

I.3 Robotics Frameworks

The nowadays interest and demand in robotics requires to have robust computational platforms and tools to make rapid prototyping, robot design, simulations of virtual models and sensors, provision, and evaluation of models and controllers. It is also crucial for developers and implementers to be aware of the available platforms, methods, algorithms, and most used hardware components, as well as their underlying physical and numerical paradigms, advantage, and disadvantage. Those reasonable reasons push us to select and design an ad-hoc platform.

To build a UNISA unmanned vehicle framework (UNISA-UVF), knowing state of the art, the tendencies, and best practices are essential. In this chapter, we present our findings and analyze the most relevant robotics framework available from a broader perspective, with an emphasis on those based on open-source. A comparison of their main characteristics, components, relevance, and adoption is made. We rely on a seminal work of Kramer & Scheutzin in 2007, they established a systematic evaluation of available Robotic Development Environments (RDEs) for mobile robots, building a comprehensive list of evaluation criteria targeted at robotics applications, comparing their strengths and weaknesses.

We start with a paper of MIRA middleware (Einhorn, E. et al. 2012); their essential characteristics are summarized in Table I.3. The authors present their robotic framework MIRA comparatively, through a benchmarking with the robotics platforms available at that moment, between them the ROS middleware. Once updated and completed the benchmarking, we could realize which platform will fit better to specific robotics needs and purposes.

Table I.3 *Middleware Platforms Summary*

Name	Organization	Description	Webpage
ASEBA	Aseba	The engine of the educational Thymio mobile robot, to program in a user-friendly using a cozy integrated development environment. A modular architecture for event-based control for complex robotic systems.	https://www.thymio.org/home-en:home
CARMEN	Carnegie Mellon Robot Navigation Toolkit	The platform provides basic navigation primitives, including base and sensor control, logging, obstacle avoidance, localization, path planning, and mapping.	http://carmen.sourceforge.net/home.html

MIRA	Middleware for Robotic Applications	Applications of several different processes (algorithms for specific tasks) on different machines (either in real-time) in a distributed layout.	http://www.mira-project.org/joomla-mira/
MIRO	Middleware for Robotics	A distributed object-oriented framework for mobile robot control, based on CORBA technology designed for high performance and real-time applications	https://www.openhub.net/p/miro-middleware
MOOS	Mission Oriented Operating Suite	Star-shaped topology network. Data as named messages stored in MOOSDB. Other clients can also fetch the history of changes.	http://www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php/Main/HomePage
OROCOS	Open Robot Control Software	Real-time control of robots and machine tools: A Kinematics and Dynamics library, Bayesian Filtering Library and Orocos Toolchain	http://www.orocos.org/
Player	Player/Stage Project	Fits well for simple, non-articulated mobile platforms. It offers more hardware drivers, provides easy access to sensors and motors on laser-equipped.	http://playerstage.sourceforge.net/index.php?src=index
ROS	Robot Operating System	Distributed environment for complex mobile and manipulator robots, based on algorithms and actuated sensing.	http://www.ros.org/
Urbi	Universal Robotic	Distributed at runtime. Determined by UObject (C++ API) for drivers and algorithms exposed to urbiscript (event-based) used to connect components in an application.	https://github.com/urbiforge/urbi
YARP	Yet Another Robot Platform	Modular, code reuse, transport-neutral interposes communication-based on Ports with different protocols.	http://www.yarp.it/

Source: based on a literature review.

A recent paper makes a detailed technical comparison of thirty-two most popular robotic frameworks, architectures, and middlewares, for our analysis, we select and complete them with the listed above (see Table I.4). The authors, Tsardoulias, E., Mitkas, A.P. (2017 p1-2), start with a clarification on the definitions of those three words which are used almost interchangeably in the literature. For them,

- The robotic framework is "*a collection of software tools, libraries, and conventions, aiming at simplifying the task of developing software for a complex robotic device*," as the APIs.
- Robotic middleware is "*the glue that holds together the different modules of a robotic system… it provides the essential software-hardware interfaces between the high level (software) and the low level (hardware) components of the system,* " like the communications infrastructure between the software nodes running in a robotic system.
- Robotic architecture is "*a more abstract description of how modules in a robotic system should be interconnected and interact with each other*," for example, to provide the communication infrastructure between the different modules (software or hardware).

Table I.4 *Robotic framework and middleware comparison*

RFWs	**OS**	**Programming language**	**Open-source**	**Distributed architecture**	**HW interfaces & drivers**	**Robotic algorithms**	**Simulation**	**Cntrol / Realtime oriented**
ASEBA	Linux	aseba	✓	✓	✓	✗	~	✓
CARMEN	Linux	C++	✓	✓	✓	✓	✓	✗
MIRA	Linux, Windows	C++, Python, JavaScript	✓	✓	✓	✓	✗	✓
MIRO	Linux	C++	✓	✓	✓	✗	✗	✗
MOOS	Windows, Linux, OS/X	C++	✓	~	✓	✓	✗	✗
MSRS (MRDS)	Windows	C#	✗	✓	~	✗	✓	✗
OROCOS	Linux, OS/X	C++	✓	✓	✓	✓	✗	✓
Player/Stage/ Gazebo	Linux, Solaris, BSD	C++, Tcl, Java, Python	✓	~	✓	✓	✓	✗
ROS	Unix	C++, Python, Lisp, Java adapters for Oc-tave/MATLAB	✓	✓	✓	✓	~	✗
Urbi (language)	Linux, OS/X, Windows	C++ like	✓	✗	✓	✗	✗	✗
YARP	Windows, Linux, OS/X	C++	✓	✓	✓	✗	✓	✗

Source: updated from Robotic frameworks, architectures and middleware comparison (Tsardoulias, E., Mitkas, A.P. 2017)

For a robotic system to function successfully, all the layers need to be covered, from the low-level embedded systems, with software for controlling the physical robot actuators, all the way up to high-level tasks such as collaboration and reasoning. All these layers of computation must be able to communicate and perfectly integrate, ideally. Common to many robotic applications are tasks such as localization, mapping, navigation. For that reason, a layer of software above the operating system but below the application program appears in the market as middleware to wrap this complexity and to provide a common programming abstraction across a distributed system. Table I.5 summarizes the advantages of these platforms.

Table I.5 *Middleware Advantage Summary*

Advantage	Description
Portability	It relies on standard programming model across language and platform boundaries, as well as across distributed end systems.
Reliability	The software modules can be reused over many applications and optimized if needed with confidence.
Managing complexity	Managing complexity by decomposition and abstraction of low-level layers increases the availability of suitable (object-oriented) libraries. However, it can be extremely tedious and error-prone the Programmation of combinations of these abstractions. Pattern aware middlewares reduce both the programmer pain and the chances to introduce errors in the code (Schmidt et al. 2000).

CHAPTER II
Robotics Frameworks for UNISA Unmanned Vehicles

II.1 Introduction

Robotics community uses Gazebo-ROS to simulate and control any robot. This powerful combination is supporting complex distributed environments with multiple robots performing tasks in a coordinated manner. It offers credible simulations and flexible, robust, and standards for robots' development.

With all these functionalities, the management of this complexity that permits both flexibility and integration with other robotic environments requires some substantial knowledge to exploit the full potential of this Gazebo-ROS platform. Thus, the first steps and learning become complicated and leave a feeling of being difficult to overcome that initial barrier that allows a minimum level of autonomy to develop with confidence in this environment.

Because of the above, and even though there are several books and tutorials, we think that a presentation of this platform from a focus on the Gazebo-ROS functionalities in the stages of modeling of mobile robots is still of interest. First, we will present an extensible Gazebo and ROS introduction, in order to identify their points of strength, the terms used, the structure and the modeling techniques of their main functionalities. Then, kinematics and dynamics over Gazebo-ROS are presented. Finally, the discussion and conclusion will suggest some range of parameters that usually works to summarize the strengths and weaknesses found in the modeling of mobile robots in this platform.

II.2 Gazebo 3D Simulator

Gazebo development starts at the University of Southern California in 2002 as part of a Ph.D. research project, after in 2009 it had been integrated with ROS in a PR2 robot at Willow Garage Company, which become the most critical financial support since 2011. Now the version 10 is on development;

it is expected to be launch by January 2019. While the v.11 is already describing their major new functionalities scheduled to be done by January 2020. Therefore, Gazebo launches a new significant version once a year, with a useful life of two years for even and five years for odd versions, respectively.

Gazebo, a powerful 3D simulator, could be integrated into different robotic platforms, ROS select it as a natural complement. It incorporates different physical engines that require to be invoked at run time from the launch scripts in ROS. The different physical engines have their level of development; some have a marked orientation to simulate certain types of robots, as in the case of Simbody for humanoids.

The rendering is crucial during robotics simulation to manage the appearance of the moving image. The rendering and the physical engine make the simulation plausible, and in the modeled environment, it is required a compromise to achieve between the accuracy of response to the physical phenomenon and the capacity to respond in computational terms. Thus, now, for the mobile robots of generalized use, it is still not possible to have both in their highest degree of reach at the same time. For example, in 3D games, many objects generally move at high speed, where the precision of the movements and the response to that set of interacting forces are not necessarily exact; however, in our eyes, they are credible. While for the development of a robotic component that must intervene in a medical environment, it will surely be of great importance the exact control of the movements and forces involved that the speed of presentation of the image.

In any simulator, a visual perspective and laws of physics of the situation they represent need to be managed. Gazebo calls it "world" to the graphical environment where various static and/or dynamic objects must be served. To each object Gazebo call "model." The configuration parameters of both "world" and "model" have a series of configuration parameters that are accessed from the graphic Gazebo environment, through plugins (executable with specific functionalities) or ROS' control platforms.

It seems very simple described in this way, but working in a virtual environment, simulating the necessary capabilities of a mobile robot, is not that simple. These powerful tools hide all this functionality's complexity, allowing us to interact with them just by setting some parameters. Those parameters related to the physical environment are interrelated, and in many cases, have immediate implications for each other. Therefore it is essential to know the basic concepts and laws that govern them, to understand why and how they have been incorporated into these tools and what they mean in each environment.

Thus, Gazebo is a tridimensional open-source dynamics simulator for a single and multi-robots' mechanisms, for inside and outside environments. Although it was created to close the gap of realistic robot simulation in outdoor environments, the users mostly use it for indoor simulations. The realistic worlds observed in Gazebo rely heavily on physics-based characteristics, which means that when the model is pushed, pulled, knocked over, or carried "reflect the physics" (Koenig, N. P., & Howard, A. (2004).

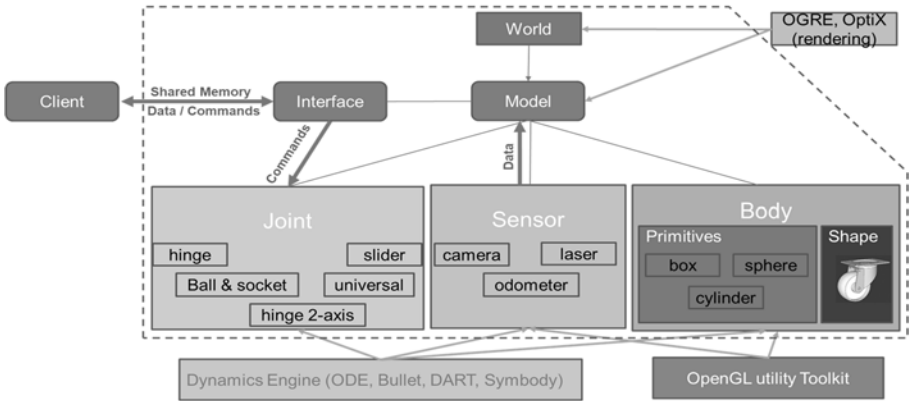


Figure II.1 *Gazebo General Structure*
(Source: Adapted from Koenig, N. P., & Howard, A. (2004 p.2150)).

Gazebo's general structure (Figure II.1) relies on third-party software packages as ODE for dynamics and kinematics of articulated rigid bodies; independent visualization toolkit, called GLUT, interactive applications (from standard library OpenGL) for the 2D and 3D creation. For that reason, it seems almost unchanged from its creation in 2004. This characteristic makes Gazebo be platform-independent, which permits, for example, to add some Dynamics Engines as Bullet, Simbody, and DART for specific versions and platforms; however, the original ODE remains as a default engine.

By doing that way, Gazebo's models like robots, actuators, sensors (dynamic objects), planes, buildings, and other stationary objects can be created and added into its virtual environment. Those objects interact based on ODE Dynamics employing Newton-Euler equations and First-order time integrator for Motion; Frictionless joints for Constraints; Perfectly inelastic collision* for Collisions and Friction pyramid for Contacts. The environmental factors as gravity and lighting are defined into the World. Finally, Client programs use interfaces to communicate and control the dynamic objects (Hsu et al. 2014).

As we could see in Gazebo Architecture (Figure II.1), there is a division between a server and a client. Governed by two executable programs "gzserver" for simulating the physics, rendering, and sensors; and "gzclient" for a graphical interface to visualize and interact with the simulation. The Gazebo communication library (like Google Protobuf and boost::ASIO, used respectively for message serialization and transport mechanism) serves to put in touch clients and servers (Koening et al. 2014).

The Gazebo official website[6] describes its features and functionalities; there are also tutorials and models to use in order to get confidence in their usage. For our work, the functionalities on which we are interested are related to robot modeling, sensors data treatment, plugins control, and dynamics simulation.

Gazebo uses SDF (Simulation Description Format), an XML format file, to describe objects and worlds capable of representing all robots' properties and simulated environments. Those are models of links, joints, sensors, static and dynamic objects, lighting, terrain, and, indeed the physics. The links are described by Inertial (mass and moment of inertia), Collision, and Visual (geometry) properties, which are used by physics, collision, and render engines, respectively. The Joints connecting two links are used to constrain their movements, defining the DOF (degree of freedom) of the robot, which is determined by their configuration type (revolute, prismatic, revolute2, universal, ball, screw).

The supported SDF protocol versions are 1.4, 1.5, and 1.6. Also, Gazebo has a dependency on SDFormat (a C++ library) to brings protocol needed by Gazebo to describe every aspect of the simulation. The library (SDFormat) handles the version dependencies (SDF protocol) automatically, those are summarized in Table II.1:

[6] http://gazebosim.org/#status

Table II.1 *Gazebo - SDF compliant versions*

Gazebo version	SDFormat version	SDF protocol version
1.9	> 1	<= 1.5
2.2	> 1.4.7 and < 2.0	<= 1.5
3	> 2.0.1 and < 3.0	<= 1.5
4	> 2.0.1 and < 4.0	<= 1.5
5	> 2.3.1 and < 4.0	<= 1.5
6	> 3.1.1 and < 4.0	<= 1.5
7	> 4.0.1 and < 5.0	<= 1.6
8	5.0	<= 1.6
9	6.0	<= 1.6
10	6.0	<= 1.6

Source: Self-constructed, based on Gazebo SDF documentation.

Sensors are independent units and are usually attached to models in Gazebo; the plugins are used to request data from sensors and to send data to them for configuration management. By the time of this paper is written, there are almost 20 sensors definitions ready to be used, those go from different types of cameras to wireless transmitters, some of them also support Gaussian and custom noise output signals. It is possible to add new sensors plugins or use third-party ones.

Plugins in Gazebo allow controlling almost every functionality through C++ classes. These complementary coded routines are compiled as shared libraries and are used to manage one of the six specific Gazebo components, namely: World (all models and physics engine), Model (joints and links), Sensor (data generation and processing), System (load and init processes), Visual, and GUI. By this means, the users can include the functionalities that are best adapted to their simulations.

The dynamics in Gazebo are done by physics libraries providing a generic and straightforward interface to fundamental simulation; by now, in Gazebo-classic versions, there are four open-source physics engines (ODE, Bullet, Simbody, and DART) integrated with a choice possibility. The ODE engine is the default; the other engines must be installed and compiled before usage. Those engines provide access to different algorithm implementations and simulation features. In simulation environments, “dynamics” is mostly related to “articulated rigid body dynamics,” but in robotics, it could be a need for particle dynamics, cloth dynamics, wave dynamics, fluid dynamics, flexible body dynamics, and fracture dynamics.

The default physics engine is ODE (Open Dynamics Engine)[7], a sophisticated software system that includes multiple numerical algorithms to deal with mathematical models of dynamical systems. It could be any collection of things (bodies) or more precisely rigid body properties (position vector, linear velocity of a point of reference, orientation of a body, angular velocity, mass, center of mass, inertia) that moves or changes over time in environments of virtual reality (Gallagher et al. 2005).

A vital change arrived since Gazebo 5 when some internal core libraries were started to move into a new external library to get more modularity; by now (since Gazebo 9), the new libraries dependencies are ignition-cmake, ignition-common, ignition-fuel-tools, ignition-math, ignition-msgs, ignition-transport (see table II.2).

Table II.2 *Gazebo External Dependencies*

Gazebo version	Ignition Math version	Ignition Transport version	Ignition Messages version
6	2.0	-	-
7	2.4	1.0 or 2.0	-
8	3.0 The built-in gazebo::math library is completely dep-recated	3.0	0.4
9	4.0 The built-in gazebo::math library is wholly removed	4.0	1.0
10 (24 Jan 2019)	4.0	4.0	1.0

Source: Self-constructed, based on Gazebo Dependencies from source tutorial.

In this ever-evolving field, the Open Source Robotics Foundation, responsible for the Gazebo simulator, announced the release of *Ignition Acropolis*[8] (March 2019). This new architecture uses ignition libraries (see Table II.3), which means significant changes, one of them is the default physics engine being DART the new one and new render engines (see Table II.4 and Table II.5) with a promise of "*level of acc uracy surpassing game engines*." The main objectives of Ignition robotics are to have *Distributed*

[7] http://ode.org/

[8] https://ignitionrobotics.org/features

Simulation to gain in performance, to be *Cross-platform* supporting Linux, macOS, and Windows (late in 2019). Also, it will offer *Cloud Integration* and *Extensible* features throughout plugins.

Table II.3 *Gazebo Acropolis' Libraries*

Gazebo Acropolis		Gazebo Acropolis	
Library name	**Version**	**Library name**	**Version**
ign-cmake	2.x	ign-physics	1.x
ign-common	3.x	ign-plugin	1.x
ign-fuel-tools	3.x	ign-rendering	1.x
ign-gazebo	1.x	ign-sensors	1.x
ign-gui	1.x	ign-tools	0.x
ign-launch	0.x	ign-transport	6.x
ign-math	6.x	sdformat	8.x
ign-msgs	3.x		

Source: Gazebo Feature comparison

Table II.4 *Gazebo's classic and Ignition Physics Engine*

Feature	Gazebo-classic	Ignition Gazebo
ODE engine	✓ Default engine	
Bullet engine	✓	
DART engine	✓	✓ Plugin shipped with ign-physics
Simbody engine	✓	
Custom engine plugins	✕	✓

Source: Gazebo Feature comparison

Table II.5 *Gazebo's classic and Ignition Render Engine*

Feature	Gazebo-classic	Ignition Gazebo
Ogre 1.x engine	✓	✓
Ogre 2.x engine	✕	✓
Optix engine	✕	✓ Partial support
Custom engine plugins	✕	✓

Source: Gazebo Feature comparison

All those libraries, tools, dependencies gain access to always improved developments, more modularity, and a higher provision of components, but on another hand, complicates the versioning management during the implement-tation. It makes that some robot models that run entirely over a software architecture will require work to be done in order to use on another one. The hardest or not of the implementation will depend on the version and type distance of the software involved, and the ability to coding.

II.3 Robot Operating System (ROS) Middleware Platform

Until a few decades ago, the entire process of robot creation needs to be done from the very beginning, every time, making the research and development process tedious and lengthy. For this reason, platforms called frameworks, middleware, and robotic environments begin to emerge and have acceptance. These robotic platforms, initially proprietary and then open-source, have developed the standard functionalities of the robots, such as location, displacement, obstacle detection, and more. These functionalities obtain information from the environment through the sensors loaded in the robots, create the kinematic and dynamic models adapted to the geometry of the robots - their mass and the payload they carry.

ROS is one of this middleware for robotics that was born in the year 2000 at Stanford University, and then since 2007 it is supported by Willow -Garage, as a robotic development platform it is modular and distributed, capable of being integrated natively with other environments, as they state when presenting ROS as an "*open-source, robot-agnostic, multi-purpose robotics middleware*". Being still young, with his little more than ten years, is hugely active, so much so that there is a new distribution every year, which means that new options and functionalities are integrated and existing ones optimized.

This dynamic within the Willow-Garage and the robotics community has made ROS the "de facto standard" framework, which increasingly increases the packages available in the market. It is undoubtedly favorable but can also be overwhelmed for someone who starts, not to mention that robotics is a multidisciplinary field where different areas of science combine with state-of-the-art technologies (Quigley et al. 2009).

The primary function of ROS is to allow the development of robots, offering basic standard features that can work in a distributed, multi-language environment, capable of integrating to other robotic platforms natively as far as possible. That is to say, the robots implemented in ROS have a computational capacity embedded in the small cards that travel with the robot and have a ROS version configured locally to meet the immediate needs as well as they have hardware and software to communicate with remote computing units with all ROS functionalities for the execution of complex or cumbersome tasks in terms of information processing. This ROS distributed work mode is not limited to native platforms but can be integrated with other robots or robotic platforms through the simplicity of the "message-passing" with which it manages its communication system.

This ability of ROS to allow the interconnection of diverse environments, in multiple platforms, and written in different languages is a significant comparative advantage between distributed robotic systems. With this same logic, the ROS environment connects with various simulation environments, although the natural selection by default is that of Gazebo, a powerful 3D simulator that was born with ROS and then separated to have its development. The combination Gazebo-ROS allows the control and simulation of robots that allows going from the simulation to the real world in an almost transparent way, that is using the same packages, parameters and configurations file with slight modifications.

Another important capability of ROS is that it makes transparent the management of sensors and actuators, many of the most known and used by the robotic community already have the ROS plugins, enabled by Willow-garage, by the critical ROS community or by the same manufacturers of these components. For example, as we will see later, during the initial simulation processes, it will be enough to incorporate some of these fake prototypes, or rather, virtual description and control codes available for sensors and actuators.

The "core" of ROS itself is design and develop in a modular way; it can even be adapted for particular developments, and after knowing this environment deeply. In general, depending on the needs in functionalities and capabilities of the robot that we are modeling, we must make the selection of the functions already available within the ROS distribution that we have selected or after a search on the web, where we will find packages made possible by the research community and the fans. However, the latter is not well documented, requires essential knowledge of programming to discern if it will adapt well to our needs and if it will adequately complement the selection made for the complementary functionalities.

Like for any other tool, it is vital to know the terms, methods, and models used to interact with them. Thus, ROS identifies the robots by their names, configured as "robot name" and described through the environment parameter "robot_description" in the scripts used when launching them in execution. The environment or "world" can be a real or simulated one. The robots have defined the "links" and "joints" in .urdf extension configuration files, in addition to the name that identifies them, which are the solid parts and hinges of the "solid body" as mobile robots are generally classified (Martinez et Fernandez, 2013).

The .urdf files (the standard robot description format in ROS) contain the description of the robot and its physical components, whether they are real or false (simulated), that is, through "links" and "joints." Together with the "meshes" (3D models) or basic geometric shapes, thoroughly describe the objects that will later be used by the simulation, visualization, and control tools. In other words, the Gazebo simulator, the ROS visualization tools, the plugins, interprets these files or command algorithms to impose a force, a torque that will make them act in the middle.

The movement capacity of the robot is closely related to the arrangement and configuration of these "links" and "joints," this configuration determines the "degree of freedom" of the "rigid body." The identification of this degree of freedom by each robot or mobile body will serve for the "kinematic modeling." It will also allow finding the limits of the movement when the "dynamic modeling" is made that incorporates the forces that interact between the rigid body and its environment.

All this sequence of modeling, starting with the 3D model of the robot, followed by kinematic and dynamic modeling, as well as the individual or group missions that must be fulfilled, are already defined in specific "packages" in the Gazebo-ROS environment. These can be used as-is, with minimal changes, if the robot we are creating has similar characteristics and has to fulfill typical missions or very similar to the packages that we want to use.

The packages are a set of algorithms, configuration files, and methodologies that have a common goal and have been packaged together to be used "on-the-shelf" or "as-it" to fulfill a mission. For example, the packages "Localization," "Mapping," "Move_base," as their names imply, aim to locate the robot at a specific moment, draw a map or manage the movement of the base of a mobile robot respectively. ROS also has meta-package or "stacks", as we could see in Figure II.2, that are the grouping of packages that together allow us to fulfill a more complex mission; thus, for example, two of its powerful stacks are "Navigation" and "Moveit!" which will use, among others, the packages described shortly before (Marder-Eppstein et al., 2010).

All these stacks, packages, plugins, and the core of ROS have a relevant theoretical background that should be known in order to interpret the behavior of the mobile robot in its interaction with its environment, with other robots or other objects in its real environment or simulation. On the other hand, the algorithms that support the platform and its main functionalities have been modified with each "distro" in order to support the new hardware and software in which it is sustained. Therefore, to maintain the compatibility of the packages between one version and another, there is a considerable effort of those in charge of maintenance; however, this is not guaranteed, especially when the changes in the dependencies are significant and ROS must adapt accordingly. The same happens with the packages that must be tailored to these new requirements; consequently, the packages created as part of our implementation will require an adaptation in turn.

It is essential to visit its website[9], again and again, to make the tutorials and fully understand the terms used, to be confident using ROS. Repair in the documentation of the packages the dependencies and conditions necessary for proper implementation, as well as the restrictions or suggestions identified. It is also convenient to visit regularly the GIT site of ROS[10] and the community where it is possible to find the source libraries that can be "cloned" and installed in the work environment.

The software modularity and distributed computing require an important communication feature, in ROS these core functions are done by Message Passing that provides an anonymous publish/subscribe through Topics; Remote Procedures Calls for synchronous request/response interactions between processes through Services, and Distributed Parameter System to share configuration information through a Global key-value store.

ROS is a middleware for robotics; it “sits in the middle” of a variety of hardware components (sensors, actuators) and a high level of software functionalities (sensing, obstacle avoidance, orientation, and motion planning) (Pyo, 2015). In this sense, ROS manages the overall process providing hardware encapsulation, distributed computing, code reusing through nodes (processes) with granular specific functionalities or complete high functionalities packages, and meta-packages grouping related packages for broadening needs like SLAM (Simultaneous Localization and Mapping), or AMCL (Adaptive Monte Carlo Localization) integrated methods on ROS Navigation and MoveIt! (Crick et al., 2017).

Package, meta-packages, and specific functionalities in ROS are written mostly in C++ or phyton using roscpp and rospy libraries, respectively; the other languages are still in early development over ROS (Lisp, Java, Lua). It is possible to configure and integrate multiple sensors data and time stamps of different devices through drivers, plug-ins, parameters, services, and messages

[9] http://www.ros.org/
[10] https://github.com/ros

communication. This granularity of code supported by an active community makes code re-usability possible because there are already implemented device driver and interfaces for high-end sensors as Velodyne-LIDAR, Laser scanners, Kinect, and popular actuators such as Dynamixel servos.

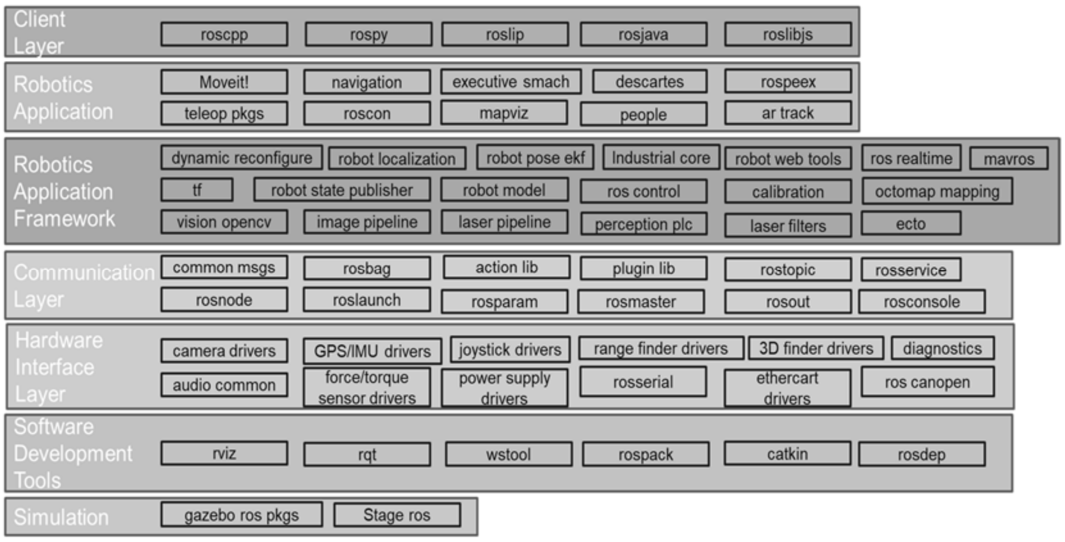


Figure II.2 *ROS Layers and Packages*
(Source: Book ROS Robot Programming p.13 (Pyo, Y. S. 2015))

The fundamental concepts of ROS implementation are nodes, messages, topics, and services (Figure II.3). Many nodes form a system, which are unit processes or modules that perform the computation of each functionality. Nodes communicate with each other through messages (typed data structure as an integer, floating-point, boolean, different messages, arrays of other messages) (Quigley, M. et al. 2009) that flows asynchronously and synchronously. Those messages are published to a given topic or service that could be seen as channels of communication, or as many to many asynchronous for the former and as synchronous feedback requirement for the later (Fankhauser et Hutter 2016).

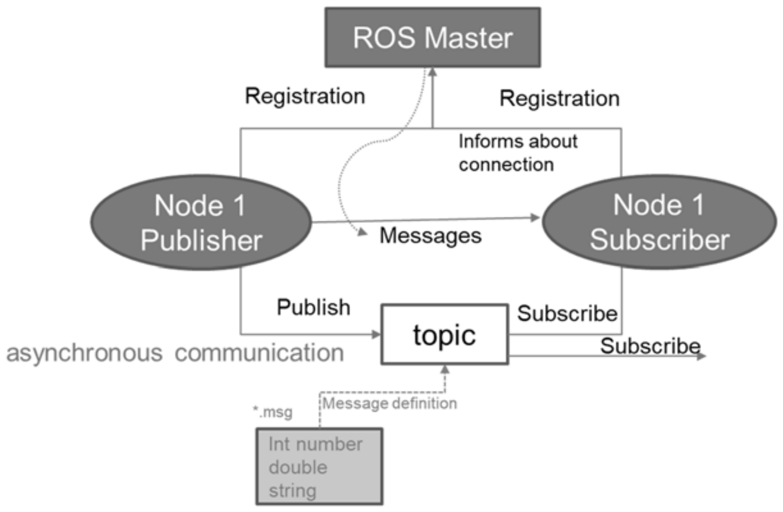


Figure II.3 *ROS Communication – a) Topic*

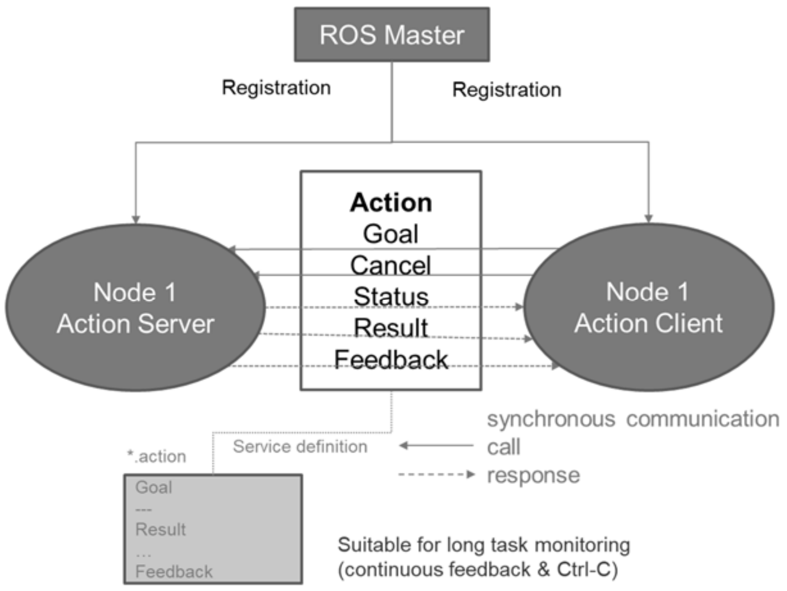


Figure II.4 *ROS Communication – b) Action*

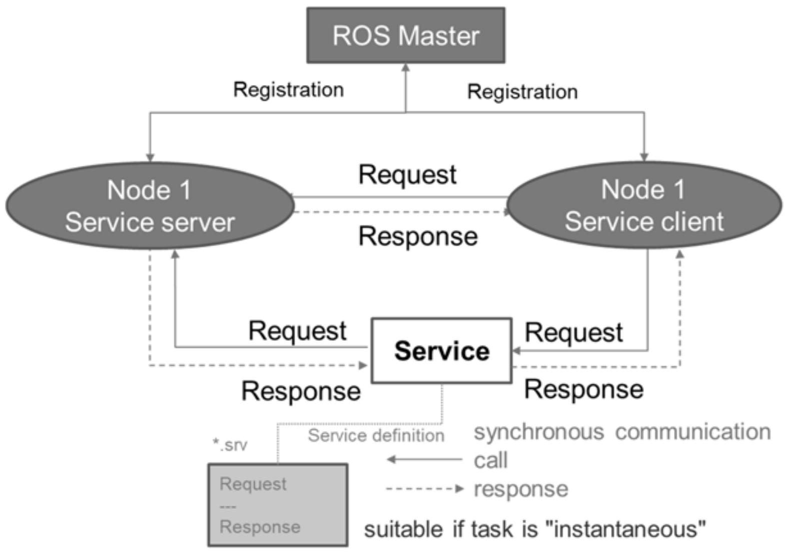


Figure II.5 *ROS Communication – c) Service*
(Source: ETH Zurich Robotic Systems Lab. Lecturer Péter Fankhauser (16-02-2018)

The execution and communication between nodes are independent of the coding language (C++ or phyton); what is essential is that nodes use the same typed messages. The roscpp and rospy libraries are it-selfs ROS packages with functionalities to initialize, handle and eliminate nodes in the system, and client libraries specific to each language program, which permits to use topics, services, and parameters to communicate with all applications running in the system.

As we already say, robotics is a rapidly changing field, and it is not an exception for ROS middleware that previously launched the third release of ROS2 call *ROS Crystal* last December 2018. However, the migration has not been undertaken jet, because mature roboticist considered it is still under heavy development (releasing new versions every six months). The motivations for this significant change to ROS2 pretend to reach the following goals[11]:

- Teams of multiple robots, ROS can do it now, but with no standards.
- Small embedded platforms instead of segregated from ROS by device drivers.
- Real-time systems in inter-process and inter-machine communication.
- Network robustness, keep working even over non-ideal networks.
- Evolve into ROS-based for production environments suitable for use in real-world applications and not only for research labs.
- Remain flexible but incorporate some prescribed patterns for building and structuring systems.

[11] Why ROS 2?. http://design.ros2.org/articles/why_ros2.html

II.4 UNISA-UVF Framework

To search, analyze, and select the platform that best suits the unmanned vehicle needs of the mechanical engineering department of the University of Salerno, we must understand what the objectives that motivate that need are. How they are doing until now and what they expect from this new platform. In order to relieve the necessary information, we use the requirements engineering techniques.

II.4.1 Requirements Engineering

Usually, when designing a technological product or service, the product or service must be identified, the objectives pursued, and the expected functionalities clarified. Also, there is a need to understand in which environment the new product or service will act. Three generic questions are formulated to find it out, namely: For whom is this service or product being created? What exactly is it about? Furthermore, What functions will it provide?

To start answering these questions ourselves, we look at the objectives formulated in this research thesis, which is to propose a technological platform that allows the mechanical engineering laboratory of the University of Salerno to carry out the modeling and simulation of unmanned vehicles of various types. Thus, from this initial approach, we can try first answers; it is mostly a software product, capable of doing modeling, simulation, and controlling of no-pilot mobile robots on any kind. The user will be a mechanical engineering team related to some advanced courses teaching, academy projects, and a spin-off to boost the university-industry relationship through real applied engineering projects.

In order to select the technological tool that best adapts to the requirement, we must fully understand the current, and the immediate future needs in order to propose an efficient environment. The first observation is that the user of this platform requires outstanding versatility and flexibility, so the element of integration between different platforms is a consideration to take into account for the reasons described above. Another observation linked to flexibility is the capacity of the current team for the creation of prototypes of non-piloted vehicles for various applications. Thus they are currently investigating areas related to the use of unmanned vehicles for precision agriculture, for the surveillance of zones extensive, for collaborative robotics, and so on.

Thus, the unmanned vehicles that must be incorporated into the platform are terrestrial and aerial; for the first, "indoor" navigation capabilities are required, and for the second, a fixed-wing aircraft capable of flying long distances and for long periods. In addition to these current projects, we have seen that the robots available in the laboratory can be reactivated to test new missions or investigate new issues related to kinematic, dynamic modeling, control, vibration, or other related to the area. Among others, they are a robotic arm, two UGVs both with traction system, one with wheels and the other with the caterpillar tractor type, that equipment that they use with more frequency. Two other projects on the way are related to an airship that should keep floating at a certain distance for extended periods and a mechanical platform with multiple uses in the industry.

Formally, in software engineering, this stage is related to the engineering of the requirements, which is an iterative process to identify the functional, data, environment, user, and usability needs of the system. The objective is to collect enough, relevant, and appropriate data to define a stable set of requirements. Even though this process has not been previously established as such, we could say that it has been most of the interview and observation type, by being in contact with the daily life of the users, it has allowed us, among others:

- understand the context of user activity
- collect qualitative information
- collect ideas about the uses that will give
- get much information

Using the requirements engineering techniques, we will define the functional needs of the UNISA-UVF framework through the description of its work environment, the identification of its primary functions, as well as the characterization of the service functions it will provide — finally, a classification of the functionalities according to usage and reusability.

Functional Expression of Need

We can summarize the framework that we build graphically in Figure II.4. This UNISA-UVF framework must be able to meet the requirements of the users that will interact in it through the functionalities they want for their autonomous vehicles. The researchers and students are building, currently a mobile robot with wheels, a small fixed-wing aircraft, and soon, they are going to develop an unmanned vehicle with caterpillar traction and an unmanned airship. Despite the unmanned underwater and boats are not coming up projects, the possibility exists because they have meaningful relationships with region companies in the maritime sector. In the same way, the framework must be able to support the sensors and actuators that are currently available in the laboratory as those that could be acquired in the future; a detailed list is not of interest in this case since these components are generally supported on all the robotic platforms.

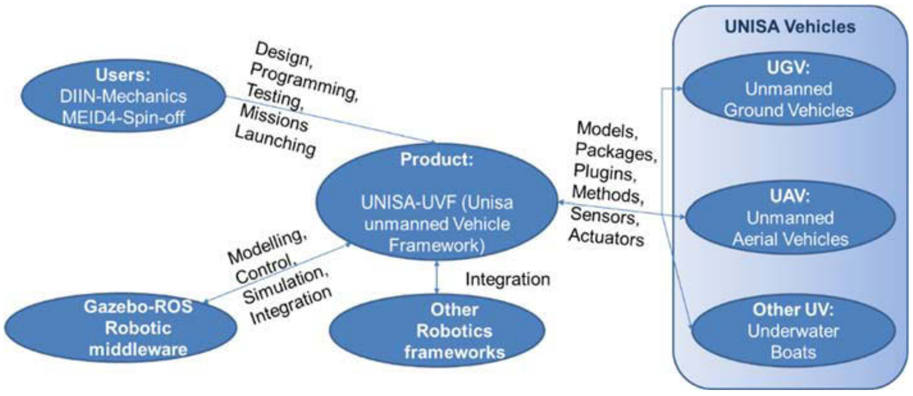

UNISA Vehicles
Users: DIIN-Mechanics MEID4-Spin-off
Design, Programming, Testing, Missions Launching
Product: UNISA-UVF (Unisa unmanned Vehicle Framework)
Models, Packages, Plugins, Methods, Sensors, Actuators
UGV: Unmanned Ground Vehicles
UAV: Unmanned Aerial Vehicles
Other UV: Underwater Boats
Modelling, Control, Simulation, Integration
Integration
Gazebo-ROS Robotic middleware
Other Robotics frameworks

Figure II.6 *UNISA-UVF Product/Service environment*

The following lines describe the components of needs expressed regarding this framework; we get the information through interviews with the main actors and by following the area internal meetings where it is discussed the priorities of the actual projects and those that are coming.

The Users

The users will be the engineers of the Department of Applied Mechanics belonging to the DIIN (Industrial Engineering) of the Salerno University in Italy, the same one that has affiliated a laboratory that supports the experimental work of the students of laureate and masters related to the career. Additionally, international students who come to do research or exchange internships for a double diploma from abroad universities, especially from Argentina, Colombia, and from the European community. Those students could be involved in unmanned vehicle projects as part of the thesis works or as part of their internships when some partners' companies are involved.

Another talented group that will make use of the framework will be the collaborators associated with the MEID4 spin-off those who are working on projects related to unmanned vehicles by means of university-industry projects, which are national and international companies that joint innovative projects when they ask for these consultancies, or when they are asked to be part of an innovative project in open founds competitions.

The UNISA Vehicles

The unmanned vehicles that must be supported by the platform are diverse, being able to be cataloged in land and air environments; without ruling out the possibility of also incorporating unmanned water vehicles. These vehicles must travel in "indoor" and "outdoor" environments for land vehicles and at different altitudes for drones. The duration of the displacement of all unmanned vehicles will vary depending on the mission that they will be assigned.

The vehicles to be used are usually modeled "from scratch," that is, conceived by the users described above since they are generally linked to innovative projects with peculiar characteristics depending on the nature of the innovation. It implies that the platform should offer them the ability to incorporate their designs.

Likewise, the configuration of the additional components that autonomous vehicles must carry out as payloads will be related to the mission assigned to them, at least during the simulation period, the platform should offer them the ability to incorporate various sensors and actuators, up to that have found the right combination for the purpose pursued. It is also to be assumed that these components will be those that are present in the laboratory or the market. Thus, the framework should make available some of these with standard configurations that allow them to speed up its use.

The Product Need: General Robotic Framework for Unmanned Vehicles

Given the level of complexity of unmanned vehicles, especially when there is a need to have to incorporate several of these for different types of projects and missions, we should immediately think of a robotic framework that offers the basic functionality already proven by other professionals of the sector. Other researchers or roboticists have been previously expressed their preferences while presenting their research paper tests by discussing the performance, essential characteristics, advantages, and disadvantages. Thus, also taken this information, we evaluate those preferred platforms looking for the features and strengths of the central platform and its components, which allow us to think that they will conveniently support the need expressed by the UNISA-UVF users, and their current and future projects related to the unmanned vehicles.

The platforms under evaluation are opensource, free to use and adapt, so the costs of the projects should not be impacted by a supplementary licensing fee, as it would affect significantly. That said, integration with robotic or related payment environments must be allowed in order to take advantage of the previous developments of the industrial engineering department, its students, and its clients or future partners when it comes to the formulation of projects. The tools or platforms initially identified are Simulink-Mathlab, Solidworks, X-plane.

One of the essential features of robotic platforms is the ability to incorporate conventional robots' functions. In our case, we are interested in mobile robotic standard functionalities, mainly for unmanned vehicles. Those general functionalities are the ability to locate in the environment, to travel in indoors and outdoors spaces, to get to get remote places in autonomous way, the ability to build a map, or to move in it, the ability to recognize and avoid obstacles, the ability to interact with other robots or objects in the environment.

Generally, in a robotic environment, especially mobile robots, have a computing capacity that moves with them, this will vary in their processing and communication capabilities, and decision-making autonomy, so it will go from the small plates with microchips to laptops or other computers with essential capabilities. Additionally, depending on the assigned mission, they will need to communicate with external, usually centralized computing units that offer better information processing capabilities. For this reason, we generally talk of distributed environments, even when a mission is being carried out by only one mobile robot. It is even more real when dealing with robots that perform collaborative tasks, which in some cases, could include various robotic platforms.

The Gazebo-ROS Robotic middleware

In the robotics community, it is known that the heterogeneity of the concepts in the field and the variability of the hardware makes the robotic applications development complex and fragile. In fact, for mobile robotics, developers must master the details related to the vehicles' locomotion medium, its morphology, and its sensory components, as well as the physics. All those will impact when coding and in kinematics and dynamics behavior during the simulation.

To respond to this observation of hardware variability, some robotic middleware such as ROS, MIRO, PyRO, and Player proposed abstractions of hardware components concerning their technical details. Their applications encapsulate specific data and provide those at a higher level. However, these abstractions do not allow the isolation of some hardware components changes, remaining at the end at a low level.

The framework, a robotic platform and work environment, has been built mainly over the chosen middleware "GAZEBO-ROS," exploring its integration capabilities to expand its usage possibilities.

The other Robotics frameworks

On the other hand, for the UNISA-UVF users, it will be essential to know the methods and techniques used by the functionalities offered on the base platform in order to understand and analyze the behaviors observed during the simulation or in real usage. Being mostly mechanical engineers, they will have a natural tendency to look for answers in kinematic and dynamic models related to the physics of the environment in which they are running their tests. Therefore, knowing what the theoretical foundations and the techniques used are essential.

Also, it would be easier for the UNISA-UVF users to manipulate, through tools of their daily interaction, the concepts, methods, and formulas of mobile robots while modeling and controlling them. Unlike the professionals linked to computer science or robotics per se who prefer to interact with high-level functional modules already created and incorporated in middleware. That is, the ability to manipulate kinematic and dynamic models through already made algorithms.

The Product Proposed: UNISA-UVF

After evaluating the different alternatives, which are detailed in chapter one, we have chosen Gazebo-ROS as our base middleware environment. Because we need support for simulation and control of complex robotic missions, with an excellent capacity for integration with tools such as Simulink, MATLAB, and Solidworks used at UNISA Labs. Gazebo-ros handles hardware components through low-level abstractions; that is, we could select existing hardware and software modules for sensors and actuators most used in the market.

To work in a Gazebo-ROS basis, we create a workspace to organize the unmanned vehicles under the latest ROS convention. This workspace is also called UNISA-UVF, houses the custom packages designed to manage the needs described so far. The designed and implemented packages are based on functionalities, unmanned vehicle type, unmanned vehicle name, or mission names. This delivered choice brings great flexibility, in order to reuse as much as possible, the code of algorithms, scripts, configuration files, and other of the kind.

Thus, we think that the users of the UNISA-UVF framework, built based on this Gazebo-ROS middleware, will be able to invoke the essential functions available for mobile robots, which are presented as meta-packages, packages, and plugins. They will also have the connectivity options with Simulink and MATLAB when they want to model the kinematics or dynamics of the vehicle on these platforms. If in the repertoire of functionalities, no package adapts to the required needs, it can be easily incorporated. On the other hand, the creation of programs in C ++ and Python, through which new features are included naturally in this platform, will always be available.

II.4.2 Technologies

Figure II.5 represents the architecture of the UNISA-UVF framework. It presents the stacks, packages, and tools of ROS configured adequately to support diverse missions of unmanned land and air vehicles for the moment, being able to incorporate the amphibious vehicles easily. It also shows the Gazebo environment, for which various useful plugins have been selected for mission simulations with unmanned vehicles. It also shows the integration with other robotic environments, whose configuration and implementation has been verified in some of the missions that will be presented in the next chapter. The integrating element is communication through various types of messages, managed by Gazebo-ROS.

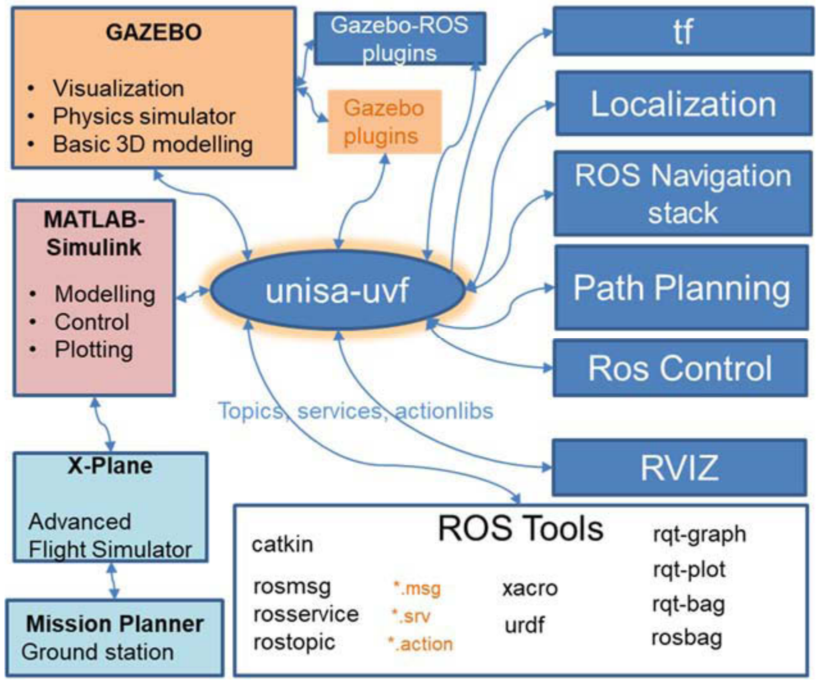


Figure II.7 *Architecture of the UNISA-UVF framework (Source: Self-elaboration)*

Table II.6 summarizes the technologies used in UNISA-UVF. Some of them (like Catkin) are part of ROS and are generally required for a project based on ROS. Others, such as Python, have been selected for their suitability and ease of use.

Table II.6 *Technologies Used in UNISA-UVF Platform*

Yam	**Application**	**Yam**	**Application**
ROS	Robotics Framework	XML	The base for URDF and SDF Formats
ros_control	Controlling Custom Robot	YAML	Data Definition Format
Gazebo	Simulator	URDF	Robot Definition Format
Python	Primary Programming Language	SDF	World Definition Format
CPP	Primary Programming Language	xacro	URDF Preprocessor
Catkin	ROS Build System Helper	Keras	Neural Network Library
CMake	Build a System Generator		

Source: Self-elaboration

II.4.3 General Design

This section shows the considerations, objectives, and priorities considered for the creation of the UNISA-UVF Framework, from a technical-operational perspective based on the requirements engineering previously carried out.

Focus

The thesis project for the creation of Framework UNISA-UVF follows a sequenced approach of objectives that must be progressively reached, even if some goals require to go back to previous steps when there is an incompatibility or low-performance behavior. However, we tried to fix the limits necessary to make them independent and measurable regarding compliance and progress. The objectives sets are the following:

1. Select and prepare the servers and personal computers that will serve as central and testing systems for the creation and simulation of unmanned vehicles, for the industrial engineering department of UNISA.
2. Prepare the equipment and install the operating systems required for the various compatibility tests (diverse Ubuntu distributions)
3. Install Gazebo, ROS, and other required main packages, taking care not to lose version compatibility.
4. Search and install ROS testing packages with functionalities like the missions suggested for the unmanned vehicles of the laboratory.
5. Install the different available distributions of Ubuntu, ROS, and Gazebo that can support the functional tests of different packages of the previous point in the additional test computers.
6. Create a UNISA-UVF directory structure to serve as a framework for autonomous vehicles in the current project portfolio of the department.
7. Create empty ROS projects for the main functionalities to be developed within the work structure.
8. For each functionality, in the case bases, create empty ROS projects for each type of unmanned vehicle (land, air, and aquatic) according to the vehicle's capabilities.
9. Create or adapt the algorithms and scripts for the functionalities by vehicle type. The added features can be:
    - For UGV, implement a custom robot controlled by a standard differential drive.
    - Use ros_control to allow precise control over each of the actuators of the robot.
    - Implement several individuals and collective robotic functionalities to measure the performance of the robot.
    - Implement sensors and actuators (fake ones for simulations).
    - Add custom worldS in Gazebo.
10. Launch nodes and topics created in Gazebo and ROS, make them communicate with each other, and correct the errors that will arise.
11. Create missions according to the types of unmanned vehicles.
12. Execute the missions created and document them.
13. Evaluate the performance of the packages created within the UNISA-UVF framework, both in the Gazebo-ROS environment and with external robotic platforms.

The objectives and programmed tasks have been fulfilled. In the case of the sensors, those with which they are counted in the mechanics' laboratory have been simulated as far as possible, while in cases where the type of sensor is not available, a generic configuration has been used or found on the web the widely used by the community due to its characteristics.

A significant fraction of our time was devoted to studying the Gazebo-ROS architecture, due to the number of available distributions, the functionalities to be implemented and the compatibility restrictions with third-party packages. The successive reinstallations served to understand the limits of integration of the platform.

The implementation of several robots usually used by both the scientific community and the robotics enthusiasts was used to perform the initial compatibility tests of the Gazebo-ROS platform. Subsequently, the fundamental concepts of mechanics were revised according to the modeled unmanned vehicle in order to select or develop an ad-hoc algorithm. The concepts of kinematics and dynamics of rigid bodies were also reviewed in detail to understand the implementation of these in various packages and functionalities of ROS and Gazebo.

It was also necessary to enter the understanding of the techniques and methods used in robotics to understand the packages and stacks of ROS as well as in the ODE physics engine of the Gazebo simulator to assign the correct values to the parameters, that govern the dynamics in said environments.

The complexity of the various disciplines involved and many packages available in the community has been gradually incorporated, and then understand that installing some dependencies caused the non-functioning of functionality already tested previously on the same computer with the identical versions of the base distributions (Ubuntu, ROS, Gazebo).

Structure Design

The following structure shows the implementation of the UNISA-UVF framework (see Figure II.6). In the first level of the structure, there are the functionalities that are available for unmanned vehicles. In the second level, the work environments are separated by vehicle type; now, each first level work environment has at least two directories, one for the UGV and another for the UAV. The third level and subsequent ones have the structure of the ROS package; that is, there are all the necessary elements to execute the Gazebo-ROS packages created with each functionality.

It is in the third and subsequent levels where the necessary details are found for the execution of the missions carried out so far, so in summary, these folders contain:

- **Launch**: stores the execution scripts that are XML type files, define the nodes to be launched, and passes the parameters required by the nodes for execution, if necessary. For some of our collaborative missions, we use the ability to define groups with independent namespaces.

- **Src**: stores the source files, meaning the programs in C ++ or phyton created or adapted for our current or future missions
- **Worlds**: contains the configuration files of synthetic environments used in the simulation with Gazebo-ROS
- **Include**: allows to point out to the catkin compiler, that the folders in the package structure can be used to find the required resources at the time of compilation and execution.
- **Param**: contains the .yaml or other files to pass the specific parameters required for the execution of the algorithms, such as the identification of active joints. It is mainly used by ros_control for the identification of transmission mechanisms.
- **Rviz**: contains the configuration of the ROS visualization tool
- **Maps**: includes the maps that will be used for navigation if they are configured.
- **Nodes**: they contain the configured files of the unmanned vehicles; they can be of the .urdf type or .xacro files (macros in XML) that using labels allows changing the configuration of the model and its characteristics.
- **Meshes**: contain a detailed description of the 3D models of mobile robots and their components
- **Models**: are the models of vehicles for missions that only use Gazebo.
- **Scripts**: contain the files of various definitions related to the other integrated robotic platforms.

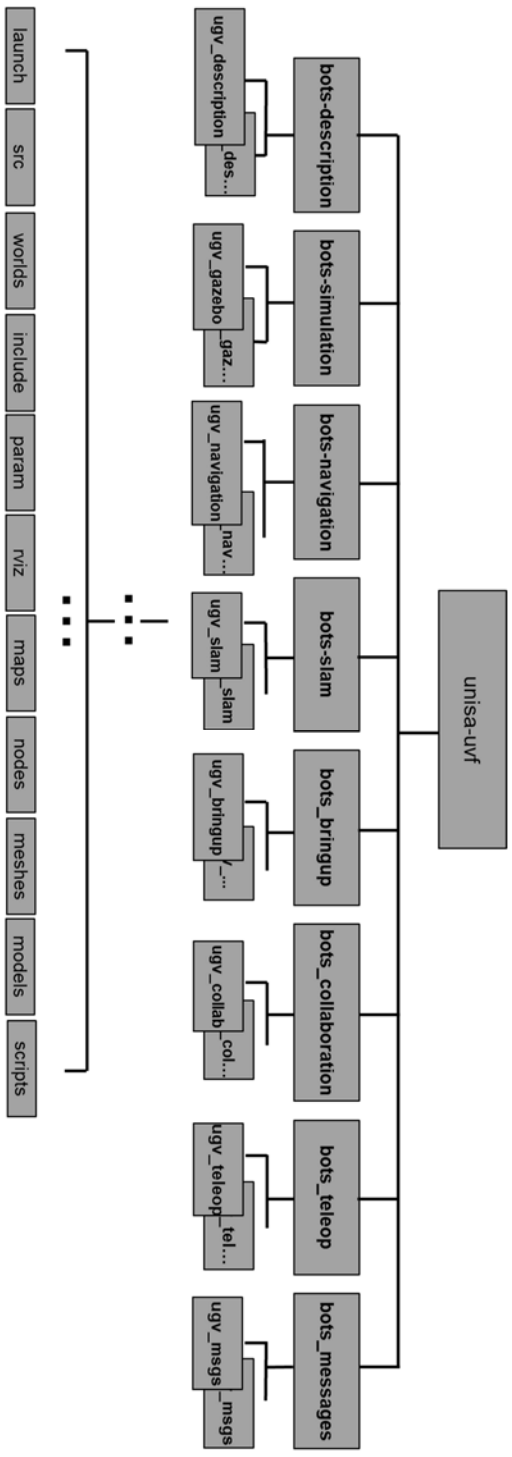


Figure II.8 *Structure implementation of the UNISA-UVF framework ROS packages implemented*

In the ROS environment, the functionalities are provided through packages and stacks created to satisfy different purposes. The former permit to develop and compile minimal collections of code for easy reuse, the latter to simplify the process of code sharing employing distribution (collection of packages). Thus, Stacks collect packages that collectively provide functionality, such as a navigation stack or a manipulation stack.

Table II.7 summarizes the ROS packages implemented in UNISA-UVF. All of them, as required, are configured in the various functionalities implemented in the framework and described later. The configuration scripts contain the execution commands and the .yaml or other parameters configuration files used. These parameter configuration files allow us to limit the assigned values and specify initial or default values that much help reusability.

Table II.7 *ROS packages implemented in UNISA-UVF*

Stack	Brief Description
Navigation	Takes in information from odometry and sensor streams and outputs velocity commands to send to a mobile base. The robot must be running ROS, have a tf transform tree in place, and publish sensor data using the correct ROS Message types. Also, the Navigation Stack needs to be configured for the shape and dynamics of a robot to perform at a high level.
Robot model	Packages that use XML to model robot’s information using URDF describing format. The URDF files parsed are used to constructs an object model (C++) of the robot[12].
ROS control	Joint state data from the robot's actuator's encoders and an input setpoint are taken, using a PID controller typically to control the output, like effort, sent to the actuators. ros_control gets more complicated for physical mechanisms that do not have one-to-one mappings of joint positions, and efforts. Nevertheless, these scenarios are accounted for using transmissions.
Vision_opencv (Open Source Computer Vision Library)	Provides packaging of the OpenCV library for ROS[13], it provides a common infrastructure for computer vision applications. It can be used to detect and recognize faces, identify objects, classify human actions in videos, track camera movements, track moving objects, extract 3D models of objects. Also, to produce 3D point clouds from stereo cameras, stitch images together to create a high-resolution image of an entire scene, find similar images from an image database, remove red eyes from images taken using flash, follow eye movements, recognize scenery and establish markers to overlay it with augmented reality.

[12] GitHub - ros/robot_model: https://github.com/ros/robot_model
[13] vision_opencv - ROS Wiki. http://wiki.ros.org/vision_opencv

Package	Brief Description
Teleop pkgs	Joy a ROS driver for a generic Linux joystick, it contains joy_node, a node that interfaces a generic Linux joystick to ROS. This node publishes a "Joy" message, which includes the current state of each one of the joystick's buttons and axes.
Robot localization	A collection of state estimation nodes per-sensor basis. Each node is an implementation of a nonlinear state estimator for robots moving in 3D space. It contains two state estimation nodes, ekf_localization_node, and ukf_localization_node. Besides, robot_localization provides navsat_transform_node, which aids in the integration of GPS data.
Robot pose ekf	Estimates the robot's 3D pose based on pose measurements coming from different sources, offering loosely coupled integration of different sensors (signals received as ROS messages). It uses an extended Kalman filter with a 6D model (3D position and 3D orientation) to combine measurements from wheel odometry, IMU sensor, and visual odometry [14].
Gmapping	Provides a ROS node for laser-based SLAM (Simultaneous Localization and Mapping), called slam_gmapping[15]. It permits to create a 2-D occupancy grid map from a laser and pose data collected by a mobile robot. A mobile robot with an odometry data source is needed plus horizontally-mounted, fixed, laser range-finder. The slam_gmapping node tried to transform each incoming scan into the Odom (odometry) tf frame. See the "Required tf transforms" for more on required transforms.
Mavros	Provides a communication driver for several autopilots with a MAVLink communication protocol. Also, it provides the UDP MAVLink bridge for ground control stations (i.e., QGroundControl).
Tf – Tf2	**tf** manage and track over time the relationships between coordinate frames. It maintains a tree structure buffered in time and enables the user to transform points and vectors. Between any two coordinate frames at the desired point in time[16]. **tf2** is the newest coordinate frame transformation library to track multiple coordinate frames over time. Operate in a distributed system, all computers and ROS components in the system get access to the information of the coordinate frames of a robot[17].

[14] Robot_pose_ekf - ROS Wiki. http://wiki.ros.org/robot_pose_ekf

[15] Cambridge Robotics RoboCup Virtual Rescue Simulation. https://staff.fnwi.uva.nl/a.visser/activities/robocup/RoboCup2018/TDPs/TDP-Cambridge-2018-draft.pdf

[16] 6 - ROS TF, Sensors.pdf - CSE 468/568 Robotic Algorithms https://www.coursehero.com/file/47074632/6-ROS-TF-Sensorspdf/

[17] tf2 - ROS Wiki. http://wiki.ros.org/tf2

Robot state publisher	Informs the state of a robot to tf, using a kinematic tree model of the robot takes the joint angles as input and publishes the 3D poses of the links. Once published, it is available to all components in the system that also use tf. The package can both be used as a library and as a ROS node[18].
cv_bridge	Converts between ROS Image messages and OpenCV images.
image_geometry	Different kinds of methods to treat the image geometry and pixels[19].

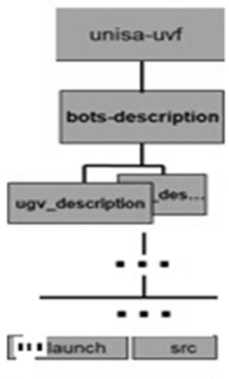
Source: self-constructed, based on the ROS website and other related documentation.

II.4.4 Functionalities

We created and implemented a series of packages to test our framework, related to a series of unmanned land and air vehicles, for each of them; we adhere as much as possible to the latest ROS name convention. As we showed earlier in figure II.6, the high-level folders organize the functionalities, and within these functionalities are the customized packages by type of unmanned vehicle. To present them, summarized in this document, we choose to do so by the name of the customized packaging, where the structure of the implementation file will be reported in the package description.

Bots_Description

For this functionality, we created three custom packages, one for unmanned ground vehicles call ugv_description and other ones for the aerial vehicles because we are intended to test two aircraft, a fixed-wing drone, and an airship. Thus, for unmanned aerial vehicles, we created the packages rosmilvus and NAS10, respectively. The number of ground vehicles we could manage is not limited, because it was created and configured to accept new models, either as a remap UGV name, with slight differences, or completely different models with the only restrictions that must be controlled as a differential driver; by knowing we have three different models.



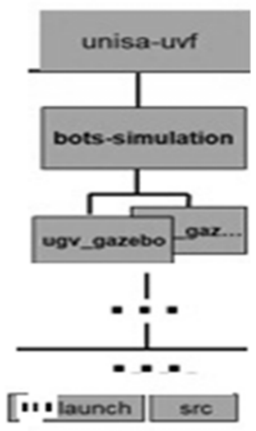
These packages contain all the files required to create ground or aerial unmanned vehicle models. Those vehicles have the meshes, in STL and DAE,

[18] robot_state_publisher - ROS Wiki. http://wiki.ros.org/robot_state_publisher
[19] vision_opencv - ROS Wiki. http://wiki.ros.org/vision_opencv

the robot description in URDF (Kunze et al. 2011), and launch files to visualize the vehicle in RVIZ. The STL and DAE files, along with the textures, are stored under meshes folders according to their categories (wheels, bases, sensors). Inside the URDF folder, the description filenames help to identify them conveniently, along with ugv_$(arg model). urdf files, some macros were used to organize and describe common characteristics. Thus this folder also contains the XACRO files of the ground vehicles, for example, ugv_$(arg model). gazebo.xacro. Finally, the launch files for visualizing the vehicle in R-viz is saved under the launch folder.

We created the different models of UGVs for test purposes, so we must set which model to use before using them in every respective functionality package that makes a call to the ugv_description package. For this, a UGV model name could be pass-through command line when launching the calling package; in calling launch files, it is possible to pass the name as an argument _$(arg model). Finally, as an environmental variable using an export command like this: export MYUGV_MODEL=my2bot or by setting this permanently in the startup launch file to use a specific one by default (.bashrc) in every user's session.

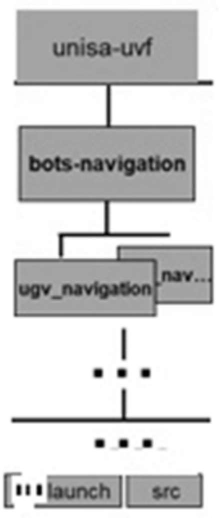
Bots_Simulation



The simulation functionality permits UNISA unmanned vehicles to be simulated in the Gazebo-ROS environment. The packages contain the files required to create an environment in Gazebo for each type of unmanned vehicle. Those have the STL/DAE, SDF, WORLD, YAML (Sinha et al. 2000), and launch files to launch the unmanned vehicles in their respective environments. The Models folder contains STL/DAE, textures, colors, SDF, and config files for the worlds. The Worlds folder contains the different custom worlds created based on community models like the turtlebot3 house, Watkins, and Cessna_demo. Finally, all the launch files for getting the vehicle in different worlds are in the Launch folder.

For UGVs, the ugv_gazebo is a calling package for bots' description; thus, the MYUGV_MODEL variable must be set before calling the vehicle model configuration in one of the exposed manners discussed previously. The gazebo models, characteristics, meaning the feature of unmanned vehicles, their components, and payload related to simulation, like physical engine definitions, plugins, and sensors/actuators. The xacro files used for those definitions are ugv_$(arg model).gazebo.xacro, common_properties.xacro, and are hosted in a ugv_definition folder to have all the vehicles' configurations in only one place.

Bots_Navigation



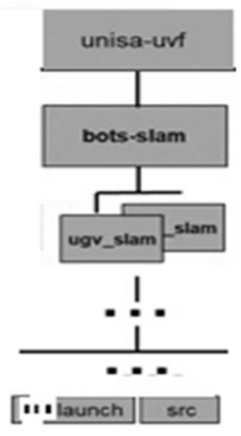
The navigation stack in one of the powerful and complete functionalities in ROS, it is presented as a stack meaning that related package is assembled to make one more complete task. It, combined with Gazebo, one with full suits of the physics engine, gives to this merge such high completeness for many mobile robotics tasks.

In our case, for ground and aerial navigation, we choose to develop them entirely independently, because the space dimensionality requires special treatment in each of these environments. The package ugv_navigation, for unmanned ground vehicles, permits to move the UGVs from one location (designated goal pose) to the specified destination (goal pose) in a given world using a map, robot's encoder, IMU sensor, and distance sensor. The maps previously obtained contains geometry information of furniture, objects, and walls of the given indoor environments we tested and are stored in the Map folder.

The Config folder contains the YAML file that includes ROS based controllers for the drive and steering of a vehicle in Gazebo. They Include folder contains all the header files, and the "src" folder contains the C++ source codes. The launch folder has the ROS launch files to launch the additional work.

For UGVs, the ugv_navigation is a calling package for bots' description; thus, the MYUGV_MODEL variable must be set before calling the vehicle model configuration in one of the exposed manners discussed previously.

Bots_SLAM



Bots_SLAM is one of the complete functionalities in our framework; it is implemented in a way that it is possible to choose between different unmanned ground vehicle models, also the number of bots for doing the job, as well as the method of SLAM that must be carried out. Also, we use configuration file .lua and .yaml files to set the parameters needed based on the technique, the mission exigencies.

The SLAM (Simultaneous Localization and Mapping) technique draws a map by estimating ground vehicles unmanned vehicle current location in an arbitrary space. Some tests are done with one and two different UGVs; for this reason, the package ugv_description is called by passing the MYUGV_MODEL name.

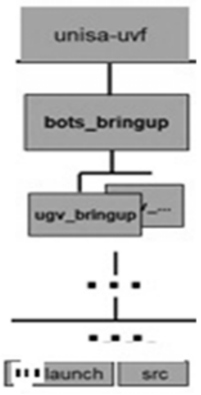
The SLAM based in Gazebo-ROS is part of navigation stack, it uses the distance measures that comes from the sensors and the robot' pose information [20]. It is possible to choose the SLAM method, either from command line passing the argument name value or from launch file as we do for ugv models. In our tests, we configured it in ugv_slam_gazebo.launch file in Launch directory, the piece of script that helps to personalize the execution looks like:

```
 <!-- Arguments -->
<arg name="model" default="$(env MYUGV_MODEL)" doc="model type [my2bot, my3bot, mybot05, burger]"/>
<arg name="slam_methods" default="gmapping" doc="slam type [gmapping, cartographer, hector, karto, frontier_exploration]"/>
<arg name="configuration_basename" default="ugv_lds_2d_gazebo.lua"/>
<arg name="open_rviz" default="true"/>
```

The Gmapping has been configured as a default slam method; it has many parameters to change performances for different environments. Some of them, with a significant impact, are maxUrange (maximum usable range of the lidar sensor), map_update_interval (how often the map is updated in seconds), minimumScore (for considering the result of the scan matching), linearUpdate and angularUpdate (a scan processes each time the robot translates and rotate respectively).

Finally, the map is drawn based on the ugv's odometry, tf information, and scan information of the sensor when the mobile robot moves, the created files map.pgm and map.yaml are conveniently renamed to be used later and stored in the Map folder. The data generated with the map_server package can be seen in the RViz visualization tool of ROS; thus, the conventional configuration file is created in the Rviz folder.

Bots_Bringup



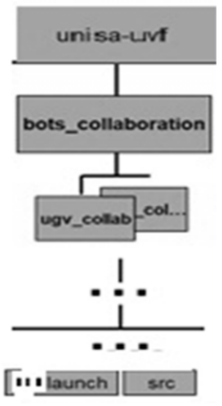
The bringup functionality permit to centralize communication, control, and visualization with real mobile robots. As ROS works in a highly distributed environment, the mobile robots communicate with each other through a master node handling the communication and data passing. The master system runs in a computer (a server, PC, or Laptop) with significant computing resources to hold the roscore, data processing, decision making, learning if it is configured for doing so. In addition, to local functionalities in a client (the real robot) carrying on a tinny computing

[20] TurtleBot3 - ROBOTIS e-Manual. http://emanual.robotis.com/docs/en/platform/turtlebot3/navigation/

system or a Laptop) depending on the applications chose to run over.

The configuration includes a varied number of launch files related to the way it is expected to deal with the robot, its components, and payload (local computing card/system, sensors, actuator, and tools). In our case, despite that we do not make tests with real mobile robots, we configure the environment for a future trial. Thus we use standard configurations for commonly robotics components. We create the ugv_bringup package and make some tests simulating fake robots, sensors, actuators, and parts over Gazebo-ROS.

Bots_Collaboration

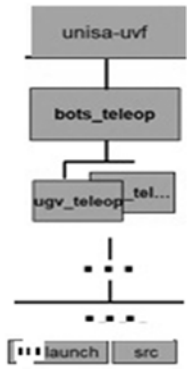


Nowadays robotics field is moving to the robotics mission that needs excellent collaborative tasks within mobile robots in ever a large environment. The collaboration could be between drones, and one or more ground vehicles, unmanned or teleoperated. Thus, this functionality groups other functionalities closely related to the missions they need to perform.

For this reason, we devoted an independent environment where it is possible to manage the configuration files to manage collaborative work, calling all the other packages already implemented in our UNISA-UVF framework. For the test, we did a collaborative SLAM with two slightly different UGVs, localizing, navigating and mapping an indoor environment, one of these environments was the Watkins lab and Turtlebot3 house, going from relaxed to a more crowded place. We also test, a kind of follower with two identical fixed-wing UAVs, flying one after the other performing Dubins pathway in the Gazebo simulator.

This package has different launch and configuration files, structured to be highly reusable using arguments, environmental variables, and namespaces.

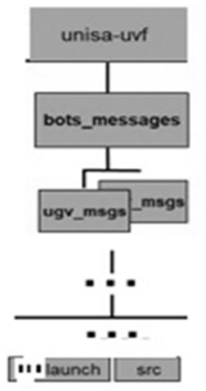
Bots_Teleop



The Teleop functionality helps to control mobile robots with some external control devices, like keyboard, joystick, and gamepads. They could also be embedded in some onboard black boxes, giving them the unmanned capabilities as autopilots in UAVs.

In our framework, we implemented the well-known keyboard packages and algorithms. Instead, for Joystick used in the tests, we start with the configuration of the device a Microsoft joystick over Ubuntu, then the joy ROS package, finally we decided to create a custom package writing an algorithm (ugv_teleop_joy.cpp) to personalize the linear and angular velocities, then, for control functionalities the ROS teleop_joy package is called. For collaborative tasks, the simultaneous control of two UGVs, we created a launch file teleop_onejoy_twobots that permit us to use only one joystick. For this reason, the ugv_teleop has all the folders of a classical ROS node.

Bots_Messages



The message passing system of ROS is a critical element of this distributed environment; with only three kinds of communication configuration, it is possible to manage large robotics environments. To be used, it is not necessary to create a custom package to use it by subscribing and publishing topics, services, or actionlibs. The message passing through this kind of channel is usually already available for the common needs of robotics missions.

However, the possibility to create a personalized message helps in many ways, from performance gains, to control robots and their components. The generated message could be compiled as a plugin to use in Gazebo, Gazebo-ROS, or ROS packages.

We created two message types to control a fixed-wing vehicle in the Gazebo environment, one to manage the mobile components of the aircraft model in the simulation environment and the other to control the graphics of this environment.

CHAPTER III
Kinematic and Dynamic Modeling of Unisa_bots

III.1 Introduction

The mathematical modeling of mobile robots, in general, is carried out in order to understand their behavior in an established acting environment. Thus, the kinematic and dynamic models are expressed mathematically and are the basis for the design and control of robots in general (Inoue et al., 1997, Lyshevski et al., 2000, O'Connor et al., 1996, Carlos, C. D. W et al., 1997, Samson 1995). The aim, when designing mobile robots, is to achieve the levels of reliability and maneuverability necessary to fulfill the desired functionalities, such as precision and speed, while having stable mechanical structures.

The kinematics and dynamics are diverse in mobile robots, depending on their morphology, the arrangement of their components, sensors, and actuators. Kinematic and dynamic characteristics may or may not consider the geometry of the robot. Several mathematical models could represent the same mobile robot, each of them having a utility based on the functionality we want to achieve, observe, or analyze. Based on these modeling, we could find the speeds at which the mobile robot moves and its position in the environment, for example.

We start with a brief introduction of the coordinate system required to identify and analyze the position and displacement of vehicles in typical environments where our UNISA_bots are likely to perform. We continue with the description of two unmanned vehicles used in our case studies. Then, the basic kinematics and dynamics of each of them are presented according to the classification of their respective categories. Finally, the Gazebo-ROS underlying parameters, models, and techniques to deal with kinematic and dynamics are discussed.

III.2 Coordinate system (reference frames)

The missions of mobile robots are closely associated with the displacement, the mobility required, and the space in which they can fulfill a specific purpose. Therefore, it is essential to represent the positions and orientations of these robots in the space and how those changes concerning time. The displacement depends on the input commands received, which could be external (remote control) or internal to the platform robotic (unmanned). We can also appreciate that the environments in which they operate, or "worlds" as they are usually called, are diverse so that the representation of the position and orientation can occur in two or three dimensions (2D or 3D).

The position and orientation are also required for the objects in the "world," they are necessary besides one of the robots, their components, and payload. Then, in robotics, any object included in the "world" for the real or simulated activity with mobile robots need to have a defined position and orientation concerning a fixed reference frame. Usually, the fixed frame is the "world coordinate frame" or “inertial frame” as it is typically called to an of static Cartesian coordinates defined in the center of the earth when dealing with 3D spaces or a central or lateral point of an area delimited in 2D spaces. The axes of the Cartesian coordinates of the static objects will remain constant for the reference frame, while they will vary in time for the objects that move, in this case, the mobile robots or the objects that in the activity are pushed, thrown, released, and loaded.

A coordinate system defines a plane (2D) or space (3D) by axes from a fixed point called the origin. This axis serves to identify the goal position to achieve in the missions, the locations of the robots, and the obstacles. They are located at some point along the axes of the coordinate systems established in the work environment. All frames in the workspace are related to the world's coordinate system, either directly or indirectly (i.e., through the main body in the case of an embedded sensor). The axes of coordinates are described as vectors; this allows making the position and orientation calculations of the represented object concerning the fixed axe of reference coordinates.

A robot could use several coordinate systems (Cartesian, polar), each one appropriate to the type of the robot, depending on the desired functionalities, even it is possible to configure more than one coordinate system. In Gazebo-ROS, the central coordinate system for the body or the chassis in UGVs are called base_link (or base_footprint), it is located at the base of the robot, to facilitate the movement of the robot from one position to another, or to facilitate the transformation calculations. Likewise, the moving parts of the robot and the sensors and actuators on board will have an ad-hoc location of their reference systems, generally for the vehicle frame. While the coordinate system of the static or dynamic object in the workspace is related to the component of the robot with which it interacts (a tool, actuator or sensor), or with the inertial frame to facilitate the programming tasks.

III.3 Modeling UNISA-UGV

III.3.1 UNISA-UGV Description

UNISA-UGV is the prototype of an unmanned ground vehicle, created in the laboratory of the faculty of industrial engineering at the University of Salerno-Italy. Because it has a conventional geometry, facilitates the study of control systems, and other related research issues, according to the department lines of research. In our case, we start with the creation of the 3D model for the Gazebo-ROS environment (the detail of this initial activity is presented in chapter 4). Then, we continue with the design and modeling of kinematic and dynamic, characterizing the vehicle, meanly the topology, sensors, actuators, and energy sources. Then, accordingly to the missions to fulfill, the configuration of a workspace is done based on underlying hardware and software capabilities available.

The chassis of our wheeled mobile robot (Unisa-UGV) is made of metal-acrylic, has a combination of ultrasonic sensors SRF05 and SRF06, four installed in the front of the vehicle, and three others on each remaining side (side and rear) for the recognition of objects. These sensors have a range of distance detection going from 2 cm to 450 cm for SRF05 and from 2 cm to 510 cm for SRF06, with an accuracy of 2 mm. Our two fixed-axle wheels use electric DC motor-reducers with incremental digital encoders. The integrated controller is an Arduino-Galileo, a board based on the Intel Quark SoC X1000 application processor. Our sensors, actuators, and microcontrollers work with a 12-volt battery (see Figure III.1).

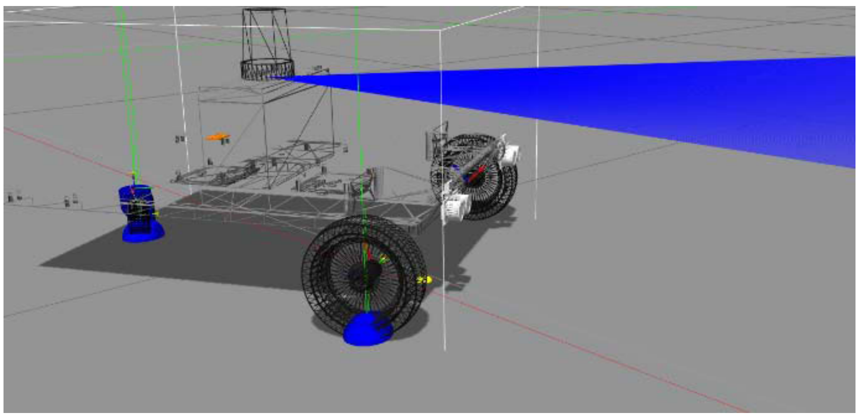


Figure III.1 *UNISA-UGV 3D prototype*
(Source: DIIN – UNISA Department of Industrial Engineering)

To give stability in a horizontal plane displacement to our UGV's structure has a rigid platform carrying two conventional fixed front wheels and a castor (rear-wheel). While doing a translational movement, the two front wheels rotate on the same horizontal axis, and the plane of each wheel remains vertical. For simulation purposes, the wheels' ground contacts have been ideally reduced to three individual points (see Figure III.2).

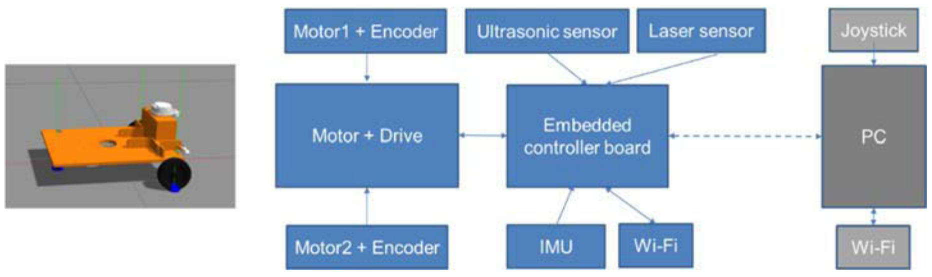


Figure III.2 *UNISA-UGV Structure*
(Source: self-elaboration)

The traction-steering system linked to our robot allows us to manage the linear and angular speed independently. Added to the advantages derived from their simple mechanical structure and conventional electronics. All together, make a clean solution that can permit different laboratory tests. The advantages could be:

- It has a simple mechanical structure facilitating kinematic modeling
- It has a low manufacturing cost
- It facilitates calculations of safe space (free of obstacles) by using the longest rigid platform side, for example, the radio of the Robot.
- It facilitates the calibration of various components that tend to present systematic errors. Those could be the unequal wheel diameters, misaligned wheel, invalid contact points of the wheel with the floor, loss of efficiency of encoders.

While the disadvantages are:

- The moving on uneven surfaces is not easy.
- The ground contact loss of one of the active wheels can change the orientation sharply.
- It is sensitivity to wheels' sliding due to slippery floors, in both external or internal forces. For example, when it collides with foreign bodies or the rotating wheels are in some arrangement.
- Only bidirectional movement (forward and backward) is available.

III.3.2 Unisa-UGV Kinematics

In general, the kinematic modeling of a UGV depends on the physical characteristics of the robot and its components (Muir et al., 1987; Campion et al., 1996; Alexander et al., 1989). There is a straightforward relation between the vehicle's structural peculiarities and the specific main task that needs to fulfill. It means that the vehicle characteristics will make them suitable for a specific task, and vice versa, the task itself will be the one that will determine in a first stage the structural particularities of the vehicle. The UGV design needs to take into account the required mobility for the vehicle to carry the assigned tasks, the efficiency of energy, the ratio of weight/payload, the dimensions, and maneuverability. In the same way, the environment characteristics for ground operations.

The traction and steering systems of ground mobile robots are distributed in their wheels axes in correspondence to speed demands, maneuverability, and target terrain characteristics. The capacities required according to the missions they will fulfill will determine the more convenient wheels, the number of them, the arrangement of these in the vehicle, as well as the traction and direction system, and finally, the physical form of the robot.

Therefore, several mathematical models can be constructed to represent the kinematic characteristics of the same UGV by incorporating properties that will be of interest to achieve or observe specific behaviors. Based on these models, the model determines the different positions in which it is located and the speeds at which it moves. When modeling, certain mathematical assumptions that help to operate them could be made, such as the restriction of movement of the wheels according to the type of vehicle, called holonomic constraint, and the assumption that the wheels do not slide on the ground (Antonelli G. et al., 2005) known as a non-holonomic restriction.

To get appropriate maneuverability of the UGV (degree of maneuverability δ_M), meaning the ability to move in the environment, are related to the degree of mobility (δ_m) and with the degree of steerability (δ_S). Thus, the overall degree of maneuverability of the UGV is obtained by the degree of mobility plus the degree of steerability. Generally, more kinematic constraints and hence a less mobile system, and in the case of steerability, an increase in the rank of implies more degrees of steering freedom and, therefore, more excellent ultimate maneuverability.

For example, for a three-wheel UGV, many configurations are possible (see Figure III.3), where each wheel contributes differently to the vehicle motion. It means that each wheel imposes zero or more constraints to the chassis, which for kinematic modeling, there is a need to combine them appropriately.

Standard wheels could be fixed with an orientation configuration; some other could be steerable with a steering angle; then, to have the rolling constraints of all wheels, an aggregate matrix must be computed.

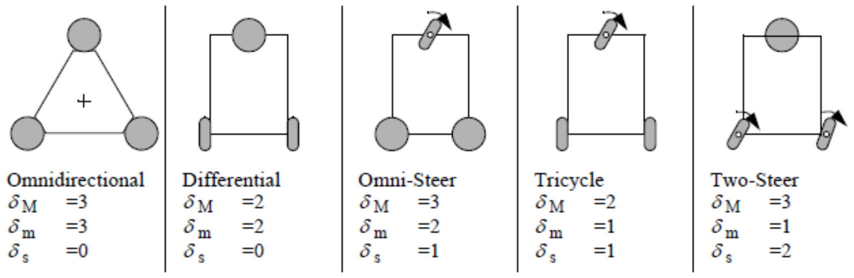


Figure III.3 *Basic types of three-wheel configurations*
(Source: Introduction to Autonomous Mobile Robots (p. 83))

It is common to find the unicycle type in mobile wheeled robots (Differential), as it is a simple configuration of wheels that at the same time develops high speeds with high traction if it has pneumatic wheels. For this reason, this configuration is most presented in UGV books and research literature. In the same way, this configuration is observed in the UGV commercialized for the home and the industry. In our case, we did not make any exceptions with the typology, and we have used this configuration, that is, the UGV used in our tests is a unicycle, so we have described the kinematics of unicycle-type UGV.

A differential drive robot can control both the rate of its change in orientation and its forward/reverse speed by merely manipulating wheel velocities. In other words, its ICC is constrained to lie on the infinite line extending from its wheels' horizontal axles. The mobile platform of our UNISA-UGV uses the mechanism of the differential drive; it allows calculating the position of the robot from trigonometric equations with the information of the two fixed front wheels arranged on the same axis of rotation.

Preliminary considerations

In UGVs, as in other ground mobile robots with wheels in general, it is assumed that the wheel remains vertical to the plane during the motion and that its orientation remains fixed or variable concerning the vehicle. Ideally, it is also assumed that the contact is reduced to a single point in the plane. So, for the three types of conventional wheels (fixed, steerable centered, steerable not centered), it is assumed that there is pure rotation without sliding (not slip), which means that the velocity at the point of contact is zero in both components of the vector.

It is also assumed that the wheels are non-deformable and that they move on a horizontal plane and are subject to the following restrictions:

- Movement is restricted to the axis of symmetry of the mobile robot: the mobile robot only moves in the direction in which the traction wheels are located, and the movement is due solely to these wheels. There is a holonomic restriction (relations between coordinates).

$$\dot{x} = u\cos\theta$$

$$\dot{y} = u\sin\theta$$

Where:

$\dot{x}$ is the speed on the x-axis

$\dot{y}$ is the speed on the y-axis

φ is the orientation of the vehicle

u is the component of the unit vector along its direction of displacement

By operating, the holonomic restriction is obtained:

$$\dot{y}\cos\theta - \dot{x}\sin\theta = 0$$

- The wheels do not slip, they do not lose adherence to the ground, so there is a direct relationship between the rotation movement of the wheels and the movement of the mobile robot. There is a non-holonomic restriction (non-integrable relations between differential coordinates).

$$\dot{\varphi} = (r)(\frac{\dot{\theta_d} - \dot{\theta_\iota}}{2\,b})$$

$$r\,(\dot{\theta_d} - \dot{\theta_\iota}) = 2\,b\,\dot{\varphi}$$

$$u = (r)(\frac{\dot{\theta_d} + \dot{\theta_\iota}}{2})$$

$$r\,\left(\dot{\theta_d} + \dot{\theta_\iota}\right) = 2\,u = 2\,(\dot{y}\sin\varphi + \dot{x}\cos\varphi)$$

By operating, non-holonomic restrictions are obtained:

$$r\left(\dot{\theta}_d\right) = (\dot{y}\sin\varphi + \dot{x}\cos\varphi) + (b\ \varphi)$$

$$r\left(\dot{\theta}_\iota\right) = (\dot{y}\sin\varphi + \dot{x}\cos\varphi) - (b\ \varphi)$$

Differential Drive System

As we have previously described, our unmanned vehicle, UNISA_UGV, considers two fixed wheels (frontal wheels), with a common rotation axis (a differential mechanism) plus an omnidirectional wheel (rear wheel), caster type.

The movement of our vehicle is controlled by the traction and steering of the two front wheels, with steering controls on the front wheels and the speeds provided by both, in the classic differential mechanism. We define standard wheels that meet the three design conditions :

- The front wheels have no lateral variations, rotate in a common axis, and are equidistant. The rear wheel is a castor providing a pure rotation contact without causing slips in the vehicle when moving.
- The two front wheels "fixed" mechanical design confers a speed restriction in the driving direction (only forward and backward) while the castor wheel has a free movement.
- The two front wheels have the movement controlled by actuators, while the idler wheel (castor rear-wheel) is passively controlled, meaning that it is influenced by the general flow of the chassis and does not provide any additional speed restriction in the movement.

The front wheels of our UGV are more significant concerning the castor wheel, so for the operations of sensing and controlling in the two instantaneous degrees of freedom of our vehicle, we need at least two actuators/sensors conveniently arranged, i.e., located on each front fixed wheel. Since the front wheels are commanded through angular velocities expressed in radians per second, it is required to find the linear velocity associated with each wheel as a result of the angular velocity performed in each unit of time.

The action on the two fixed wheels is done by a mechanism called differential drive, which guides the movement. In our vehicle, its frame origin has been located at the midpoint of the line that links the two fixed wheels and an axis orthogonal to this line.

In the vehicles that use a differential mechanism, it is possible to have three types of movements, as relived in literature. The first, when the speeds of both front wheels are the same, we have a straightforward move. The second when the wheels speeds are identical, but in opposite directions, we have a rotation in its central axis (the midpoint of the common axis). Third, when one of the wheels has zero speed, we have a rotation around that wheel. No lateral movement is possible, and this restriction is known as Singularity. Other singularities are related to errors in the relative speeds of the wheels, or the small ground-level variations which usually are mitigated by the castor wheel.

Because the front wheels can act independently, by changing their speeds, the mobile robot moves in linear trajectories or making turns, to the right or the left, depending on the lower speed value of one of the corresponding wheels. The movements are observed concerning the vehicle's frame. To have a vehicle moving in circles, it must turn around a point, called ICC - Instantaneous Curve Center or ICR, which is along the common axis of the right and left wheels (see figure III.4).

When the front wheels vary their speeds, acting independently, the mobile robot moves with linear trajectories or with turns, to the right or the left depending on the lower speed value of one of the wheels.

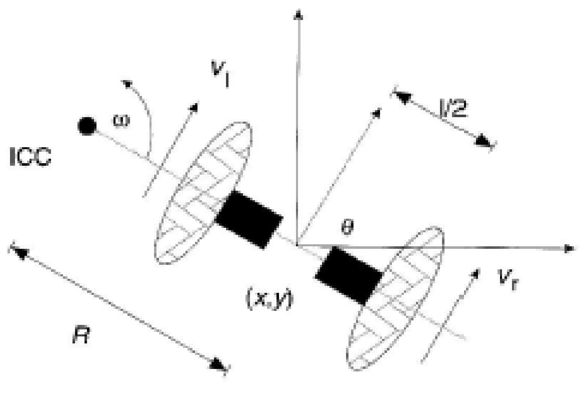


Figure III.4 *Differential Drive kinematics*
(Source: Dudek and Jenkin, Computational Principles of Mobile Robotics)

From Figure III.4, we could appreciate that the angle of rotation of the vehicle, *w* over the ICC needs to be the same for both wheels. For this reason, we can write the following equations:

$$w\left(R+\frac{l}{2}\right)=V_r$$

$$w\left(R+\frac{l}{2}\right)=V_l$$

Where:

ω is the angle of rotation with respect to ICC

l is the distance between the two wheels (from their centers)

V_r is the linear velocity of the right wheel

V_l is the linear velocity of the left wheel

R is the distance from the ICC to the midpoint between the wheels (signed).

For any moment during this movement, we can calculate the values of R and w:

$$R = \frac{l}{2}\left(\frac{V_l + V_r}{V_r - V_l}\right)$$

$$w = \frac{V_r - V_l}{l}$$

Direct Kinematics in Differential Drive System

Until the previous equation, we have only found the radius *R* to the Instant Curvature Center (ICC) and the angular velocity *w* of the robotic platform based on the speeds of the right and left wheels. With this data, we can find the positions of the robot in terms of time (δt). We can assume an initial position (x, y) in Figure III.4, pointing in the direction corresponding to the angle θ concerning the X-axis. When sending speeds to the wheels independently, these will generate different translation and rotation movements and will place the robot in a new position (x ', y') with an angle θ'.

We use the above equation to find the Instant Curvature Center (ICC), which will then serve to determine the new position at a time $t + \delta t$ as shown below:

$$ICC = [x - Rsin(\theta), \quad y + Rcos(\theta)]$$

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} \cos(w\delta t) & -\sin(w\delta t) & 0 \\ \sin(w\delta t) & \cos(w\delta t) & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x - ICC_x \\ y - ICC_y \\ \theta \end{bmatrix} + \begin{bmatrix} ICC_x \\ ICC_y \\ w\delta t \end{bmatrix}$$

This process is called Direct Kinematics; when we have the control parameters such as the speeds of the two wheels and the time, we can determine the new position *x* and the new orientation θ reached by the robot.

Reverse Kinematics in Differential Drive System

Contrary to the previous case, the Reverse Kinematics tries to find the appropriate control parameters to take the robot to the desired position $(x, y, \theta\ )$ each time.

The formulas to achieve this are as follows:

$$x(t) = \frac{1}{2}\int_0^t [V_r(t) + \mathrm{V_l(t)}]\cos[\theta(t)]\delta t$$

$$y(t) = \frac{1}{2}\int_0^t [V_r(t) + \mathrm{V_l(t)}]\sin[\theta(t)]\delta t$$

$$\theta(t) = \frac{1}{2}\int_0^t [V_r(t) + \mathrm{V_l(t)}]\delta t$$

We must remember that the UGV with a differential drive system has holonomic and non-holonomic constraints. For this reason, certain positions are not easily reachable and require a series of previous maneuvers, such as when it is desired to reach a position parallel to the starting position, as well for other exceptional cases. The equations of movement that help to solve them are those that follow.

When the speeds of the left and right wheels are equal, then the speed of the robot will also be the same as these, and the robot will move in a straight line. So, when $V_l = V_r = V$ we have:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} \mathrm{x + V\cos}(\theta)\ \delta t \\ \mathrm{y + V\sin}(\theta)\ \delta t \\ \theta \end{bmatrix}$$

When $V_r = -V_l = V$ the robot revolves around the central axis of the front wheels, we have:

$$\begin{bmatrix} x' \\ y' \\ \theta' \end{bmatrix} = \begin{bmatrix} \mathrm{x} \\ \mathrm{y} \\ \theta + \dfrac{2V\delta t}{l} \end{bmatrix}$$

As we have seen, the restrictions of the arrangement of the wheels lead to a nonlinear formulation to solve the kinematics of this type of mobile robot, with mathematical descriptions in terms of trigonometric functions. However, its derivatives, as the speeds are not and it is for this reason that the computational platforms use these values (speed, acceleration, force or torque) instead of finishing the position itself (quote).

III.3.3 Unisa-UGV Dynamics

UNISA-UGV dynamic model is based on the rigid body concept that does not deform under the action of the applied forces. The main body is the chassis, where the wheels are configured as independent rigid bodies, each one placed conveniently as an articulated link through joints, in this case, revolute ones. The forces acting in the relation of UGV (with all the components) with the ground are modeled to understand the behavior in motion.

Rigid bodies can be joined through mechanical joints (modeled as kinematic restrictions) to build mechanisms (vehicles and other articulated bodies). This reduced number of parameters that describes the system of several bodies and the reference frames attached to each body helps in the formulation and solution of the dynamics through a description of the position, movement, and acceleration of the individual components and the system in general, as a function of time.

There are powerful methods to formulate the equations of motion of mechanical systems; the dynamic approach of Lagrange is one of them. It uses the equations of motion systematically considering the kinetic and potential energies of the given system. Another critical approach is Newton-Euler's ones, the first and most crucial step in Newton-Euler's dynamic modeling, is to draw the free-body diagram of the system and analyze the forces acting on it.

Behind the simulations of rigid bodies, different paradigms are taken by the physics engines that support dynamics in virtual environments. Understanding how simulators incorporate these paradigms into their functionalities could help explain some frequently strange behaviors in the robotic simulation. Brogliato et al. (2002) present a state of the art of the interactions of rigid bodies in numerical simulations, discussed the dynamics of extensible mechanical systems, making an interesting distinction between models of rigid and compatible bodies and the simulation of their contacts.

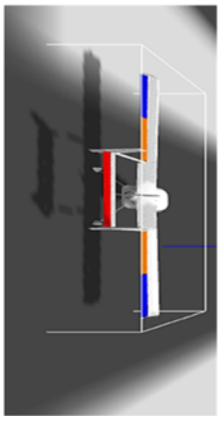
III.4 Modeling UNISA-UAV

III.4.1 UNISA-UAV Description

It is a prototype of an unmanned aerial vehicle (Fig. III.5) with fixed-wing electric motorization, created in collaboration with an aeronautical company of the Campania region, in Salerno-Italy. The prototype of the real airplane is in the company's facilities, while the 3D model had to be adapted to be able to take it to the Gazebo-ROS and X-Plane simulation environments used in this thesis work. It is an aerial that can be cataloged as LALE (Low Altitude Long Endurance) according to the proposal of the international organization UVS; based on the different parameters of the vehicles such as flight fee, flight duration, operating range and maximum take-off weight (MTOW).

To date, various works have been carried out with different software with this unmanned aerial, including preliminary dimensioning to determine the main characteristics such as the aerodynamic profile, the opening, and winding, the power of the engine, and the capacity of the battery. Then, through software such as DATCOM + and ADS, the aerodynamic characteristics were analyzed. With all this information and the preliminary designs of the air, the 3D models and the characterization of it were generated to configure it in X-Plane, a powerful flight simulator, certified by the FAA and with a commercial license. Subsequently, connections were made with MatLab to receive and send flight information in real-time, working together with X-Plane to identify an initial dynamic model of the new flight. This model was used to configure a PID-type flight controller, integrating it into the Ardupilot software to perform Software-In-the-Loop (SIL) activities with X-Plane.

Within this series of activities, the work of this thesis is framed, with the aim of modeling and controlling the air in a robust open-source environment such as Gazebo-ROS. For this, it began with the adaptation of the 3D model, having to go through the Solidworks to generate the air configuration files, its components, and the first sensors, later the creation of control algorithms through reuse, as far as possible. As much as possible, of packages available in Gazebo-ROS, the detail of the activity is described in chapter four (lines below). Now several tests are being made additions with different missions using Mission Planner and X-Plane.



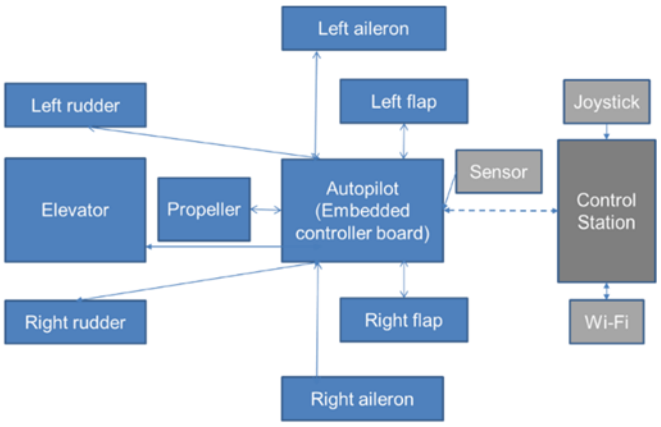


Figure III.5 *UNISA Unmanned Aerial Vehicle Structure*
(Source: self-elaboration)

III.4.2 Unisa-UAV Kinematics

Our UAV is an airplane with six degrees of freedom (DOF) that responds to the possibilities of displacement in space. These are controlled through the command inputs of the servos conveniently arranged in the moving parts of the aerial (elevator, spoiler, rudder, and accelerator). Additionally, degrees of freedom are conferred concerning the payload that one wishes to control, among them, for example, some instruments that measure wind and other disturbances.

The development of the design and model of the fixed-wing UAV required the consideration of kinematic, dynamic characteristics, and energy sources as inputs of the models performed, as well as the missions that must be fulfilled and the necessary hardware and software capabilities available.

The control of the "pose" (position and orientation) of the aerial and its sensors is a fundamental issue in the kinematics of any mobile robot. Consequently, it is also the transformation between their coordinate systems. In general, the coordinates of the UAVs start with three basic positions, namely:

- 1st Coordinate System: How different bodies are oriented towards each other.
- 2nd coordinate systems: How the aerial is oriented with respect to the earth.
- 3rd coordinate systems: How the sensors (cameras, antennas, lidar) are oriented with respect to the aerial.

In addition to these three coordinate systems, many more are added that will facilitate the transformations, mainly because the measurements are made in different frames. For example, the movement of air is more natural to describe in a coordinate system fixed to its structure (center of air mass).

Likewise, the aerodynamic and torques acting on the structure of the air, making it easier to express them in their coordinate system, as well as all sensors onboard (accelerometers, gyroscopes) take their measurements concerning the coordinates of the body. On the other hand, the GPS measures the position of the air and the angle of travel (course angle) concerning the earth (inertial).

While, most of the missions that are assigned to the air as trajectories of flights and overfly certain areas, are specified in the inertial reference systems.

These are many reasons for the various commonly used coordinates when modeling and controlling an air vehicle. Also, it cannot be forgotten that Newton's equations of motion are expressed in the fixed inertial referential system; usually, the chosen one is the referential system of the earth.

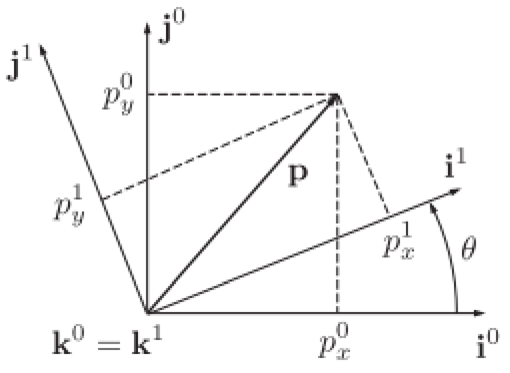


Figure III.6 *Rotation in 2D*
(Source: Small-Unmanned-Aircraft-Theory-and-Practice (p. 9))

Of the two coordinates shown in Figure III.6. The rotation matrices allow

$$p = p_x^0 i^0 + p_y^0 j^0 + p_z^0 k^0$$

$$p = p_x^1 i^1 + p_y^1 j^1 + p_z^1 k^1$$

$$p^1 \triangleq \begin{bmatrix} p_x^1 \\ p_y^1 \\ p_z^1 \end{bmatrix} = \begin{bmatrix} i^1 & i^0 & j^1 & j^0 & i^1 & k^0 \\ j^1 & i^0 & j^1 & j^0 & j^1 & k^0 \\ k^1 & i^0 & k^1 & j^0 & k^1 & k^0 \end{bmatrix} \begin{bmatrix} p_x^0 \\ p_y^0 \\ p_z^0 \end{bmatrix}$$

$$p^1 = R_0^1 p^0$$

III.4.3 Unisa-UAV Dynamics

The dynamics in three-dimensional space are highly complex and are usually model through different techniques that the main objective is to reduce them by decomposition, loops, and others. Beard, R. W., & McLain, T. W. (2012) in their book Small unmanned aircraft: Theory and practice, present in chapter four the forces and moments acting in a Micro Aerial Vehicle (MAV) due basically to gravity, aerodynamics, and propulsion sources. The formulas of total forces f and moments m are the sums of these three sources:

$$f = f_g + f_a + f_p$$

$$m = m_a + m_p$$

Where:

f	is the total force acting on the airframe
f_g	is the force acting due to the gravity
f_a	is the force acting due to the aerodynamics
f_p	is the force acting due to the propulsion
m	is the moment acting on the airframe
m_a	is the moment acting due to the aerodynamics
m_p	is the moment acting due to the propulsion

The authors in the book use the standard approach that capture the effect of the air pressure with a combination of forces and a moment applied at the quarter-chord point (see Fig. III.7), also known as the aerodynamic center, where it is possible to get enough stability despite changes in angle of attack and lift coefficient. At this point, the forces are modeled using a lift force, a drag force, and a moment.

Gravitational and Aerodynamic Forces and Moments

Beard, R. W., & McLain, T. W. (2012) model the earth's gravitational field effect as a proportional mass force acting at the Center of Mass, and hence considered no moments produced by gravity in their mathematical representation. This force represented its body-frame components follows:

$$f_g^b = R_v^b \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} = \begin{bmatrix} -mg\ sin\theta \\ mg\ cos\theta\ sin\phi \\ mg\ cos\theta\ cos\phi \end{bmatrix}$$

For aerodynamic analysis of fixed-wing airplanes, it is common to group longitudinal forces causing motion in the body $i^b - k^b$ plane, by a lift in the direction i^b and drag in k^b, and pitching moment in the direction j^b. While for the group of lateral forces causing translational motion in the lateral direction along the j^b axis and rotational motions in roll i^b and yaw k^b that will result in directional changes in the flight path of the airplane (Beard, R. W., & McLain, T. W. (2012)).

Longitudinal Aerodynamics

The longitudinal aerodynamic analysis to determine the forces of lift, drag, and pitching moment are formulated as follows:

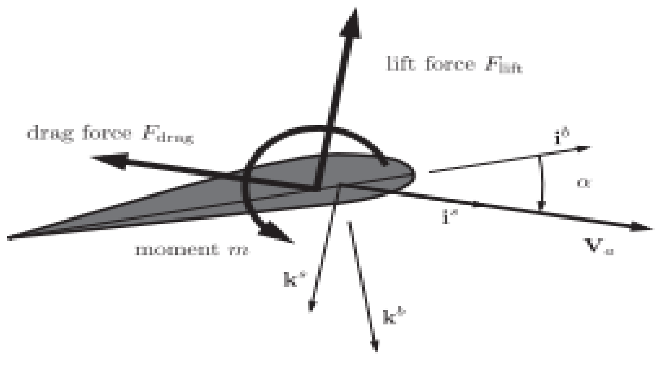


Figure III.7 *Air pressure at aerodynamics center of MAVs (source: Small unmanned aircraft: Theory and practice)*

$$f_{drag} = \tfrac{1}{2}\rho V_a^2 S C_D(\alpha, q, \delta_e)$$

$$f_{lift} = \tfrac{1}{2}\rho V_a^2 S C_L(\alpha, q, \delta_e) \qquad m = \tfrac{1}{2}\rho V_a^2 S_c C_m(\alpha, q, \delta_e)$$

Where:

ρ	is the air density
V_a	is the speed of the MAV through the surrounding air mass
S	is the planform area of the MAV wing
c	is the mean chord
$C_L\ C_D\ C_m$	are the nondimensional aerodynamic coefficients of the MAV wing
α	is the angle of attack
q	is the pitch rate
δ_e	is the elevator deflection

The longitudinal aerodynamic analysis also depends on the deflection of control surfaces and their dispositions in the aircraft (usually the elevator, the aileron, and the rudder, and others like spoilers, flaps, and canards), which are used to modify the aerodynamic forces and moments. For example, a positive aileron deflection δ_a is produced when the left aileron is trailing edge down, and the right aileron is trailing edge up. In consequence, the longitudinal forces and moments are heavily influenced by the angle of attack α , the pitch rate q, and the elevator deflection δ_e creates functional dependences.

The forces lift, drag, and pitching moment can be modeled with acceptable accuracy using Taylor linear approximations, when the angle of attack is small, because the shape of the flow field is predictable (laminar and attached to the body) and changes in response to changes in the angle of attack, pitch rate, and elevator deflection. However, the effects of wing stall must be incorporated into the longitudinal aerodynamic model in order to design appropriate control laws for real aircraft and simulate their performance (Beard, R. W., & McLain, T. W. (2012)).

Lateral Aerodynamics

The lateral aerodynamics are most significantly influenced by the sideslip angle β. They are also influenced by the roll rate p, the yaw rate r, the deflection of the aileron δ_a, and the deflection of the rudder δ_r. Denoting the lateral force as f_y and the roll and yaw moments as l and n respectively, we have:

$$f_y = \frac{1}{2}\rho V_a^2 S C_Y(\beta, p, r, \delta_a, \delta_r)$$

$$l = \frac{1}{2}\rho V_a^2 S_b C_l(\beta, p, r, \delta_a, \delta_r)$$

$$n = \frac{1}{2}\rho V_a^2 S_b C_n(\beta, p, r, \delta_a, \delta_r)$$

Where:

f_y	is the lateral force
n	is the roll moment
l	is the yaw moment
ρ	is the air density
V_a	is the speed of the MAV through the surrounding air mass
S	is the planform area of the MAV wing
b	is the wingspan of the aircraft
$C_Y\ C_l\ C_n$	are the nondimensional aerodynamic coefficients of the lateral
β	is the angle of sideslip
p	is the roll rate
r	is the yaw rate
δ_a	is the aileron deflection
δ_r	is the rudder deflection

These forces and moments are aligned with the body axes of the aircraft and do not require a rotational transformation to be implemented in the equations of motion. Also, the coefficients the lateral force coefficient $C_{Y_0}\ C_{l_0}\ C_{n_0}$ when $\beta = p = r = \delta a = \delta r = 0$ are typically zero for symmetric aircraft.

Aerodynamic Coefficients

The use of aerodynamic coefficients permits us to deal with perturbations to tend to restore the aircraft to its nominal flight condition. The $C_{m_\alpha}\ C_{l_\beta}\ C_{n_\beta}$ coefficients represent the change in the moment coefficients concerning changes in the direction of the relative airspeed, as described by α and β, behaving like torsional springs. While the $C_{m_q}\ C_{l_p}\ C_{n_r}$ coefficients behave like torsional dampers, all these values are seen as stability derivatives determining the MAV static and dynamic stability (Vidan, C., & Badea, S. I. 2016). The moments of inertia of the MAV body provide the mass.

The aerodynamic coefficients $C_{m_{\delta e}}$ $C_{l_{\delta a}}$ $C_{n_{\delta r}}$ are associated with the deflection of control surfaces and are referred to as the primary control derivatives. They define the off-axis moments that occur when the control surfaces are deflected; those can be thought of as gains.

Propulsion Forces and Moments

The propeller generates thrust and torque in the aircraft that has it in his configuration. Usually, Bernoulli's principle is used to calculate the pressure ahead of and behind the propeller and then applying the pressure difference, despite this approach considers an efficient propeller, it could be acceptable for starts the simulations. Beard, R. W., & McLain, T. W. (2012) design consider the thrust acts directly along the i^b body-axis of the aircraft. Therefore, the thrust does not produce any moments about the center of mass of the MAV, then we have:

$$f_p = \frac{1}{2}(\rho S_{prop} C_{prop}) \begin{bmatrix} (k_{motor}\delta_t)^2 - V_a^2 \\ 0 \\ 0 \end{bmatrix}$$

Where:

f_p	is the force due to the propulsion system
ρ	is the air density
S_{prop}	is the area swept out by the propeller
C_{prop}	is the aerodynamic coefficient for the propeller
k_{motor}	is the constant that specifies the efficiency of the motor
δ_t	pulse-width-modulation, it is the control signal denoting the throttle deflection
V_a	is the speed of the MAV through the surrounding air mass

The torque applied by the motor to the propeller (and then to the air) results in the same and opposite torque applied by the propeller to the engine that is fixed to the MAV body. This torque is contrary to the direction of the propeller rotation and proportional to the square of the propeller angular velocity. The effects of this propeller torque are usually relatively minor, the slow rolling motion that could cause is easily corrected by applying a small aileron deflection, which generates a rolling moment to counteract the propeller torque. The moments due to the propulsion system are, therefore:

$$m_p = \begin{bmatrix} -k_{T_p}(k_\Omega \delta_t)^2 \\ 0 \\ 0 \end{bmatrix}$$

Where:

m_p	is the moment due to the propulsion system
$(k_\Omega \delta_t)$	is the speed of the propeller
k_{T_p}	is a constant determined by experiment, it is opposite to the propeller rotation direction

Atmospheric Disturbances

Usually, wind disturbances are considered when modeling aircraft dynamics. The air mass relative to the ground (wind velocity Vw) added to the velocity of the airframe relative to the surrounding air mass related (Va) equals the velocity of the airframe relative to the ground (Vg). The wind velocity components include steady ambient wind, wind gusts, and other atmospheric disturbances, which are represented in different frames and directions. Approximation and transfer functions are used to model them, as the von Karmen model with the Dryden transfer functions. By the experiment's results, suitable parameters into the equations of motion for low and medium altitudes and light and moderate turbulence are possible to find in the literature.

Linear Design Model

As we pointed out initially, linearization and decoupage of the equations of motion to produce reduced-order transfer function and state-space models that are more suitable for control system design are done regularly because Low-level autopilot control loops for unmanned aircraft are designed based on these linear design models, which capture the approximate dynamic behavior of the system under specific conditions.

The lateral dynamics decomposition of the aircraft motion is given by the following Laplace equation, which expresses the relationship between the aileron deflection and the roll angle:

$$\phi(s) = \left(\frac{a_{\phi_2}}{s\left(s + a_{\phi 1}\right)}\right)\left(\delta_a(s) + \left(\frac{1}{a_{\phi_2}}\right) d_{\phi_2}(s)\right)$$

Where:

ϕ	is the roll angle
s	is the complex variable of Laplace
V_g	is the ground speed
a_{ϕ_1} a_{ϕ_2}	are the ailerons roll angle
δ_a	Is the control signal denoting the aileron deflection

The relationship between the roll angle and the course angle is given by the equation:

$$x(s) = \left(\frac{\frac{g}{Vg}}{s}\right)(\phi(s) + d_x(s))$$

Where:

x	is the course angle
ϕ	is the roll angle
s	is the complex variable of Laplace
g	is the gravity
V_g	is the ground speed
d_x	is the disturbance signal associated with reduced course model

For aircraft that have a rudder and the ability to measure the side slip angle, the following equation expresses the relationship between the rudder deflection and the sideslip angle.

$$\beta(s) = \left(\frac{a_{\beta_2}}{\left(s + a_{\beta 1}\right)}\right)\left(\delta_r(s) + d_\beta(s)\right)$$

Where:

β	is the sideslip angle
s	is the complex variable of Laplace
V_g	is the ground speed
$a_{\beta_1}\ a_{\beta_2}$	is the ailerons side slip angle
δ_r	Is the control signal denoting the rudder deflection
δ_β	Is the control signal indicating the side slip deflection

The transfer functions for the longitudinal dynamics are given by the next equations, which model the relationship between the elevator deflection and the pitch angle:

$$\theta(s) = \left(\frac{a_{\theta_3}}{(s^2 + a_{\theta 1}s + a_{\theta 2})}\right)\left(\delta_e(s) + \left(\frac{1}{a_{\theta_3}}\right) d_{\theta_2}(s)\right)$$

The following equation expresses the relationship between the pitch angle and the altitude:

$$h(s) = (\frac{V_a}{s})(\theta + \frac{1}{V_a} d_h)$$

The following equation expresses the relationship between the airspeed and the altitude:

$$h(s) = (\frac{\theta}{s})(V_a + \frac{1}{\theta} d_h)$$

Finally, the following equation expresses the relationship between the throttle and pitch angle to the airspeed, respectively:

$$\bar{V_a}(s) = \left(\frac{1}{(s + a_{V_1})}\right)\left(a_{v_2}\bar{\delta}_t(s) - a_{v_3}\bar{\theta}(s) + d_v(s)\right)$$

III.5 Modeling in GAZEBO-ROS

III.5.1 Gazebo-ROS Kinematic - main characteristics

Kinematics is the study of a body motion without asking for forces or torques that are at their origin. It is used in robotics to understand the motion of robots in order to design and control them using numerical tools and algorithms. The Kinematics for robotics are highly studied since seminal works of Denavit and Hartenberg (1955) coordinates into robotics, and it continues to be a challenging problem the determination of the position kinematics, from computational point of view, car nonlinearity of their mathematical formulations and results, basically comes from joins and constraints descriptions in terms of trigonometric functions (Roth et al., 1994).

Several methods, techniques, and tools had been created, as it is related in a comprehensive and accurate kinematics state-of-art presented by Roth et al. (1994). Some following conferences and assembly of papers continuous to point out the challenging of Kinematics for sophisticated robots, especially manipulators and humanoids and mobile robotics in off-road environments.

The general robot kinematics division falls on forward and inverse kinematics. Forward kinematics (angles respect to position) is a straightforward position's calculation based on each link length and each joint angle related to that link. While in inverse kinematics (position to angle), the length of each link and the position of some point on the robot are given, and the calculation is done to find the angles of each joint to obtain that goal position. Inverse kinematics is a far more difficult problem because of singularities and nonlinearities, increasing the cost to have a complete analytical solution (Kucuk & Bingul, 2004) and making it computationally expansive and time-consuming for real-time control.

In kinematics modeling, there are mainly two different spaces used, namely, Cartesian space and Quaternion space. Cartesian coordinates (x,y,z) represent a smooth and natural means of representing a position in 3D space if there is a translation, a new position of an object can be described as a new Cartesian coordinate translated from the origin. If the object experiments a rotation, an angular displacement in any of the Cartesian axes, there is a need to calculate the new orientation of that object from its original unrotated orientation.

Position and orientation are critical pieces in mobile robotics; because we need to track the robot itself and their sensing and acting components and over time while the robot is performing any mission. To manage these representations, roboticist in their numerical tools and algorithms, incorporate homogenous transformations based on 4x4 real matrices (orthonormal matrices), based on the theoretical robot kinematics representation of Denavit & Hartenberg (1955) who showed that a global transformation between two joints requires four parameters, known as DH parameter. Moreover, in the measure of, they are getting access to better computational capabilities; the quaternions are used for rotation representation.

There are many ways to represent rotation, Euler angles, Gibbs vector, Cayley-Klein parameters, Pauli spin matrices, axis and angle, orthonormal matrices, Hamilton's quaternions, and Dual quaternions. The select one depends on the complexity of the robot configuration, in the environment where it is acting and the mission it is performing, as well as on the computational power and how to cope with some issues (numerical, storage, user interaction, or interpolation). For example, Dual quaternion offers a considerable advantage in terms of computational robustness and storage efficiency for dealing with the kinematics of robot chains, because it can present rotation and translation in a compact form of transformation vector, simultaneously vs. the nine elements in homogenous transformations (Funda et al., 1990).

As we see before, the 3D simulated mobile is represented as many bodies or links attached through mechanical joints to build mechanisms. Those mechanisms have reference frames attached to each body that participated in the formulation and solution of dynamics by a description of the position, the motion, and the acceleration of the individual components and, consequently, in the overall system as a function of time.

For basic robots' configurations, the preferred Kinematics methods are Geometric and Algebraic approaches; the latter involves coordinate transformations. With more than three joints and with kinematic chains that do not lay on the plane, the geometric method is too difficult. Thus, a systematic approach is needed for each pair of consecutive joints; it usually includes algebraic solutions using homogeneous coordinates and Denavit-Hartenberg Notation. In this notation, to describe how a frame (i) relates to a previous frame (i -1), four DH parameters are needing to align two axes based on displacements and rotations.

Forward kinematics usually follows the following steps:

1) Identify the robot position in rest.
2) Assigns reference frames to joints and links.
3) Compute Denavit-Hartenberg parameters.
4) Compute transformation matrix Ai that allows passing from the reference frame of the i-th joint to the one of the (i+1)-th joint.

5) Multiply matrices Ai to get matrix T that allows moving from the original reference frame position of the base XYZ to the one of the XYZ in the final place.
6) Extract the coordinates of the current position from the matrix T, and finally.
7) Look at the rotation sub-matrix and extract orientation components.

The solutions methods for Inverse Kinematics could be closed or iterative. In the former, the geometric methods reduce the more significant problem to a series of plane geometry problems, and algebraic methods through trigonometric equations. While in the latter, the iterative numerical solutions are m equations with n unknowns, which represent an alternative nowadays. Behind the scenes, the necessary tools for kinematics' numerical solutions are matrix multiplication and scalar arithmetic simple algebra to manipulate equations based on graphical constructions and the sine and cosine laws of triangles.

The system models represent key characteristics as their number of degrees of freedom (DoF), based on the number of rigid bodies and the constraints imposed by their joins, the mass distribution, and their expected behaviors and functions in the selected physical or abstract system. To capture these characteristics, we use methods, algorithms, and equations in the mathematical models. Then a simulation, the process of running a model, would reproduce the outcomes of the mathematical models associated with the system.

The robot’s kinematics are closely related to the design of the robot itself and the environment where it is subject to perform their missions De Simone et al. (2017-2018). Thus, a robotic system typically has many 3D coordinate frames that change over time. Those could be the world frame, base frame, gripper frame in the case of a robotic arm attached to a mobile robot. In the Gazebo-ROS platform, the tf package keeps track of all these frames over time and allows, to know where a frame is relative to the other frame (world, map) in a specific time, before or where it would be expected to be in a near-future time to come.

The tf ROS-based package, and tf2, its' updated version, can operate in a distributed system. It means that all information about a robot's coordinate frames is available to all ROS components on any computer in the system. Therefore, there is no central transformation information server, this is still challenging due to the current trend of missions with cooperative robots working together, with sources of information on the transformations of their links and joints between different sets of coordinates tables that vary dynamically, acquiring greater complexity with the increase of robots in the mission (Koubâa et al. 2017).

Tf2 gets in charge of all the details of transformation in the ROS-based platforms, it is efficient (both at computational and bandwidth), and simple to use. There are mostly two tasks, listening for transforms and broadcasting transforms that the user invokes. The Listener, to receive and buffer all coordinate frames that are broadcasted in the system, and query for specific transforms between frames. The Broadcasting, to send out the relative pose of coordinate frames to the rest of the system. Different parts of the robot broadcast information about their relative pose to the system.

At the high level, the Design Goals provides developers and users a distance from the details of the specific coordinate frame data store for each one. Everything is broadcast and reassembled at the end consumer points now there is a need for a transformation (timestamped at times other than the current time). It is possible to have many different data sources for tf information, and data is not required to be synchronized by using interpolation when working in a distributed environment can also arrive out of order. There is only the need to know the name of the coordinate frame to work with using "frame_ids" as unique identifiers.

When evaluating a transform, the tf2 uses a directed tree structure allowing fast traversal (n depth tree). A link redefinition can serve ti reconfigure it. The core tf2 library is C++ class, a second class provides ROS interface and instantiates the core library. Moreover, Multi-Robot Support is allowed, included those with the same or similar configuration using "tf_prefix" that limits their scope of action.

The Developers of drivers, models, and libraries need a share convention for coordinate frames in order to integrate better and re-use software components. Shared conventions for coordinate frames provide a specification for developers creating drivers and models for mobile bases.

III.5.2 Gazebo-ROS Dynamics - main characteristics

Gazebo-ROS uses articulated rigid body structure to simulate dynamics, that is various shapes or bodies connected with joints of multiple types (ball-and-socket, hinge, slider or prismatic, hinge-2, fixed, angular motor, universal). The Lagrange multiplier velocity base model of Trinkle/Stewart and Anitescu/Potra serve to build the equations of motion (Cosmin Petra et al. 2009). Contact and friction model are based on the Dantzig LCP solver described by Baraff, although ODE implements a faster approximation to the Coulomb friction model.

It is possible to call functions to apply forces to the rigid body at each CPU execution time (integrator step). These forces applied to push it around are the sum of all forces, added into "force accumulators," in the rigid body object. The force accumulators are set to zero after each integrator step.

The system models represent key characteristics as their number of degrees of freedom (DoF), based on the number of rigid bodies and the constraints imposed by their joins, the mass distribution, their expected behaviors and functions in a selected physical or abstract system. To capture those characteristics, methods, algorithms, and equations are used in these models. Then a simulation, the process of running a model, would reproduce the outcomes of mathematical models associated with the system.

In simulated virtual environments, these models and their representative scenarios with several possible states could be represented alone or with other static and dynamic models. The interaction between all the elements must be considered to simulate the desired environment. In this context, dynamic system simulation is the computing process, over a time span, of a system's states and outputs using the information provided by the system's model.

The system's states are represented by a mathematical model of a physical system, usually by first-order differential equations (where functions are related to their derivatives) or difference equations (a type of recurrence relation). As a system, the value of external inputs variables impacts on state variables (which evolve through time), and subsequently influencing output variables values. The different changes experienced in the simulated environment over a period (a span of time) responds to (usually changing) input signals and an attempt to find a state in which the system is in equilibrium. There are different external forces that affect the motion of rigid bodies such as gravity or generated forces when contacts occurred (collisions).

In general, Newton's equations of motion are the foundation of all types of physics-based simulation. Since simulations often include multiple degrees of freedom (DoFs), the computation of relation of forces acting on a body, with constant mass, are expressed in vector notation. When the dynamical system is linear, time-invariant, and finite-dimensional, then the differential and algebraic equations may be written in matrix form. Another common aspect of the physics-based simulation is the methods and algorithms used for collision detection and collision response.

Brogliato et al. (2002) present a state-of-art of the rigid body interactions in numerical simulations, discussing the dynamics of mechanical systems extensible, and making an interesting distinction between a rigid body and compliant models in their contacts' simulation. From these other papers, we could see that behind rigid body simulations are different programming paradigms, among which three have been taken by most physics engines to supporting dynamics simulation in virtual environments. Those three are event-driven, time-stepping, and penalty-based (Taylor et al. 2016), a summarized difference is presented in Table III.1.

Dynamic solver libraries and simulation software are a tailor in mathematical models based on their expected domains of application (Robotics, Games). Differential-Algebraic Equations (DAEs) defines a relationship between the functions representing physical quantities and the derivatives representing their rates of change. It usually involves expensive numerical methods; for this reason, solvers generally prioritize accuracy over efficiency using specific and strongly simplified mathematical formulations. A differential Variational Inequality (DVI) is a system in which a function describes the time dependence of a point in a geometrical space (infinite, continuous, and three-dimensional); the dynamics, inequalities, and discontinuities are present. While arbitrary integration schemes, used in penalty-based methods, put multiple springs and damper models for multiple contact points and solves multiple contact forces at once, depending on how contacts forces are calculated could give only approximatively results (Baraff, 1989).

Table III.1 *Dynamics Paradigms in Numeric Simulation*

	Simulation Paradigm		
	Event-driven	**Time stepping**	**Penalty-based**
Contacts treatment	each impact event modeled	all events in a time-interval are collected and modeled into a single complementary problem	virtual spring and damper
Mathematica l model	differential algebraic equation	differential variation inequality	arbitrary integration schemes
Recommend ed	Integration forward in time	Integration forward in time	implicit integrators to avoid "stiff" equations

Source: Elaborated based on literature review

Steve Peters in a ROSCon2014 presentation talks make a comparison of the physics engine used for rigid body dynamic simulators for robotic simulation, the methods used by each of them for the most characteristic feature in simulations are described in the following Tables III.2, III.3, and III.4.

Table III.2 *Physics Engines for Simulation*

Features	DART	ODE	Bullet	Simbody
Contact Formulation	Pure Rigid	Pure Rigid	Pure Rigid	Compliant
Joint Spring/Damping	Implicit	Implicit	Implicit	Implicit
Coordinate representation	Generalized	Maximal	Maximal	Generalized
Most used in	Robotics, Animation	Robotics, Gaming	Gaming, Animation	Biomechanics
Started	In 2008 at Georgia Tech	In 2001 by Russell Smith	Sony, AMD, Google	Stanford
OpenSource Providers	Georgia Institute of Technology	ode.org	Sony, AMD, Google	SimTK
Sources	https://dartsim.github.io/index.html	https://sourceforge.net/projects/opende/files/	https://pybullet.org/wordpress/	https://simtk.org/projects/simbody
Supported Platforms	Linux, Mac OSX, and Windows	Linux and Windows	Windows, Linux, Mac OSX, iOS, Android	Linux, Mac OSX, and Windows

Source: ROSCon 2012 - The Gazebo Simulator as a Development Tool in ROS (John Hsu and Nate Koenig Slides).

Table III.3 *Coordinate Representation in Numeric Simulation*

	Coordinate representation	
	Maximal	**Generalized**
Coordinate type	Absolute	Independent
Inter-penetrating bodies	separated by constraint stabilization	---
DOF	6*links	links
Mass matrix	sparse 6*links x 6*links	dense links x links
Constrains to solve	6*links – links	0
Kinematics	Accuracy depends on a constraint solver	implicit in formulation

Source: ROSCon 2012 - The Gazebo Simulator as a Development Tool in ROS (John Hsu and Nate Koenig Slides).

Table III.4 *Spring-Damper Computation*

	Spring / Damper numerics	
	Explicit	**Implicit**
Velocity of time (i+1)	depends on state (i)	depends on state (i+1)
Computation	easier to compute	numerically stable

Source: ROSCon 2012 - The Gazebo Simulator as a Development Tool in ROS (John Hsu and Nate Koenig Slides).

In Gazebo, when two objects collide, like the wheels that roll on a surface or try to move through it, a frictional force is generated; there are physical motor systems defined in the simulator software to manage these forces. ODE is the default physical engine in Gazebo, where the friction consists of two parameters, " 'mu' " and " 'mu2' ", which represent:

1. " 'mu' " is the coefficient of friction of Coulomb for the first direction of friction.
2. " 'mu2' " is the coefficient of friction for the second direction of friction (perpendicular to the first direction of friction).

ODE will automatically calculate the first and second friction direction for us. However, we can manually specify the first friction direction in the model description file ".sdf" or in ".urdf" if used in the Gazebo environment -ROS The two objects in collision specify their own " 'mu' " and " 'mu2' ". The gazebo will choose the smallest " 'mu' " and " 'mu2' 'between the two colliding objects. The valid range of values for " 'mu' " and " 'mu2' " is any non-negative number, where 0 equals a contact without friction, and a significant value approaches a surface with infinite friction. Tables of the coefficient of friction values for a variety of materials can be found in engineering manuals or the online toolbox.

For a terrain-wheel interaction model, we consider non-deformable wheels because our real WRV has solid plastic materials, and the weight load during the simulation operation and the actual experiences would not be so important that they would change the rigidity. The selected interior environments were solid pavement. Therefore, the interaction between the wheel and the ground can be reasonably approximated as a point contact. It allows the use of classical Coulomb friction to describe the limits of available traction and lateral forces for the load function (basically the power, motors, sensors, actuators, and other components carried out) with a coefficient of friction (a parameter to be set in a simulated environment).

CHAPTER IV
Mobile Robotics Techniques used in Unisa_bots

IV.1 Introduction

The robotics techniques concretize several of the concepts reviewed so far in this document. It is here that the different areas of knowledge are integrated, each contributing its technologies in the broadest sense, namely the knowledge of the specialists, the experiences of the developers, implementers and the final users; thus, the techniques described below allow us to approach the complex world of mobile robotics from a practical perspective.

"From mechanics to computing for unmanned mobile vehicles" defines well the concept of robotic techniques that interest us in this study. Because unmanned vehicles concentrate, to a great extent, both of the phrase' disciplines, the former contributes to the design, locomotion mechanisms, and control of vehicles, while the latter provides the computational power. The combination of mechanical engineering with computer science, and related engineering fields permits the development of systems that go beyond hardware, algorithms or telecommunications to allow these vehicles to move and behave "intelligently" in environments with large-extensions and through various means, recognizing objects and places, sending and receiving information in real-time and deploying their capabilities in a concerted manner.

The described capabilities require that mobile autonomous vehicles could integrate different techniques to estimate their position and location, build or use a map, navigate knowing or not the environment previously, identifying trademarks, planning, and following routes. An abstract representation is needed to deal with it; one way is to see the mobile robot as a point (x, y) in a continuous or delimited space of two or three dimensions - generally, a Cartesian plane. To describe the mobile robot state, also called pose (position and orientation) at every timestamp. When the robot moves through free spaces, it changes its pose; each free space is called C_{free} and houses the robot in its trajectory.

A set of rigid bodies also represent the mobile robot, for example, the UGV has the chassis, the wheels, the actuators, and sensors onboard. The same for the UAV, it has the main body, ailerons, flaps, elevator, rudders, and propellers, also carries onboard sensors, actuators, and communication equipment. When a robot is seen as a point, the techniques can be applied to each component by reducing them to their mass center.

The robotics techniques for path planning, location, perception or sensing, mapping, and SLAM (simultaneous localization and mapping) are created with three fundamental abstractions "space," "pose," and "free space," giving the mobile robot a safe trip. Below we will describe how these techniques are used in the various functionalities of mobile robots. What hardware components or equipment are needed and how they are formulated in the Gazebo-ROS platform. We start with 3D modeling techniques, necessary for describing robots, their components, and onboard sensors/actuators, as well as to create virtual environments that will later serve for simulations.

IV.2 Design Modeling Techniques (3D models)

We use a compatible XML-like language (Xacro, URDF, and SDF) to create the virtual model of our unmanned vehicles, in order to work smoothly in the Gazebo-ROS environment. Despite the primitive's shapes of our UGV, there was not easy to construct the model directly on Gazebo's building editor; because of the limited building tools of the environment. Thus, we use two strategies for modeling; in both, it was necessary to pay special attention to geometry and the relationship of all the components. Also, we must adhere, as much as possible, to the three most important ROS standards: the first related to Standard Units of Measure (REP-103), the second to Coordinate Conventions (REP-105, in robotics, the orthogonal coordinate systems are commonly called frames) and the third to ROS Package Naming (REP-144).

Our Wheeled Mobile Robot (WMR) 3D model is made up of a rigid platform equipped with two front and one back caster non-deformable wheels, it moves on a horizontal plane. During motion, the plane of each wheel remains vertical, and the wheel rotates around its horizontal axle, the orientations concerning the cart platform are fixed. Also, the contact between the wheels and the ground is reduced to three single points (see thin green line in fig.3a).

To design our ground mobile vehicle, we use different strategies in order to choose the appropriate tools for the robotic frameworks we are building. First, we tried adapting another similar wheeled mobile robot; we choose a four-wheeled vehicle provided by the ROS community through the GitHub platform. Then we use a SolidWorks to design our WMR model and import it as a .dae file using a plugging to converts the resulting 3D model into a URDF file. Comparatively (Fig. IV.1 and Fig. IV.2), both roads give as a virtual 3D model, but the time expends to create a model from scratch and to export and adequate the model from a CAD tool are both comparable time-consuming.

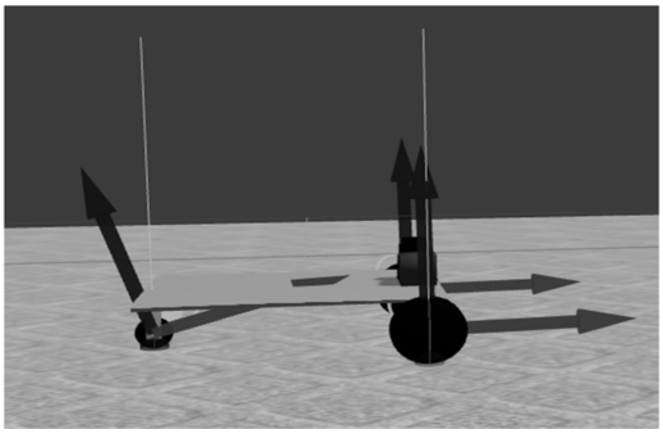


Figure IV.1 *UNISA-UGV designed in ROS and presented on Gazebo (Source: Self-elaboration)*

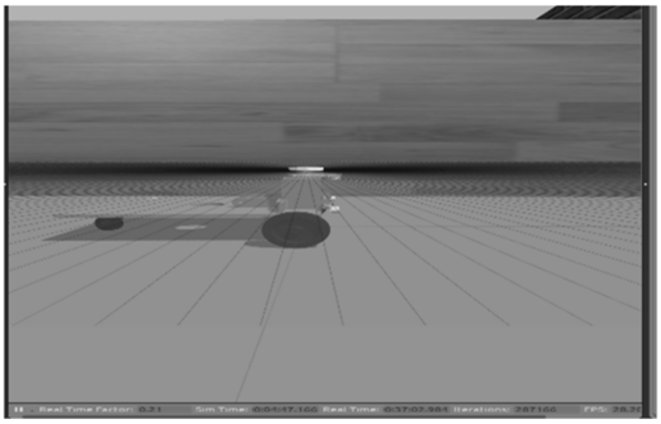


Figure IV.2 *UNISA-UGV designed in Solidworks and presented on Gazebo (Source: Self-elaboration)*

We come out with our UGV's virtual 3D model with both strategies. The one that comes from Solidworks gives us the possibility to export with the information of links inertial calculated directly from the model, and the characteristics of onboard sensors. In this way, the model is ready to use in Gazebo-ROS; however, the available plugin to convert the 3D model to an urdf or sdf file was not a straightforward task, we need to do many workarounds in order to align the reference frames and the poses. The following lines present the UNISA_UAV using the Solidworks model and the process to get those.

3D CAD for Multi-Body advanced modeling – From Solidworks to Gazebo

Because of the complete information of the robot and its components that we get for the 3D model in Solidworks, it is vital to master the export process. The Gazebo-ROS platform accepts two kinds of files for a robot description, the Universal Robot Description Format (URDF) and the Simulation Description Format (SDF). The former file type is used heavily in ROS for robot visualization and control, while the latter, the SDF files, for simulation in Gazebo.

It is advisable to have a simplified assembly of the 3D CAD model in Solidworks to avoid errors during the export process. It is better to assemble the bodies part if those will not act independently in any way, and separate if they will participate in the motion. The resulting body elements are considered as rigid bodies and must be identified correctly.

Once exported, it is essential to check if the resulting rigid bodies are positioned in the right way by opening the robot in the Gazebo simulator. The collision models need to be in the same place as the visual model; otherwise, it could be an error in the origins of the SolidWorks model. It could also be advisable to check the meshes in software as a Blender and move the part origin to the exact position if necessary.

The flowchart shows the outcomes of the two processes to obtain the robot description files from Solidworks (see Figure IV.3). The first uses a SW2URDF plugin to get a kind of ROS package to be used by any application in the Gazebo-ROS environment. The second is a procedure to follow called "Gazebo Exporter," the result is a .sdf file with the robot description with all the additional files needed to be launch in the Gazebo simulator.

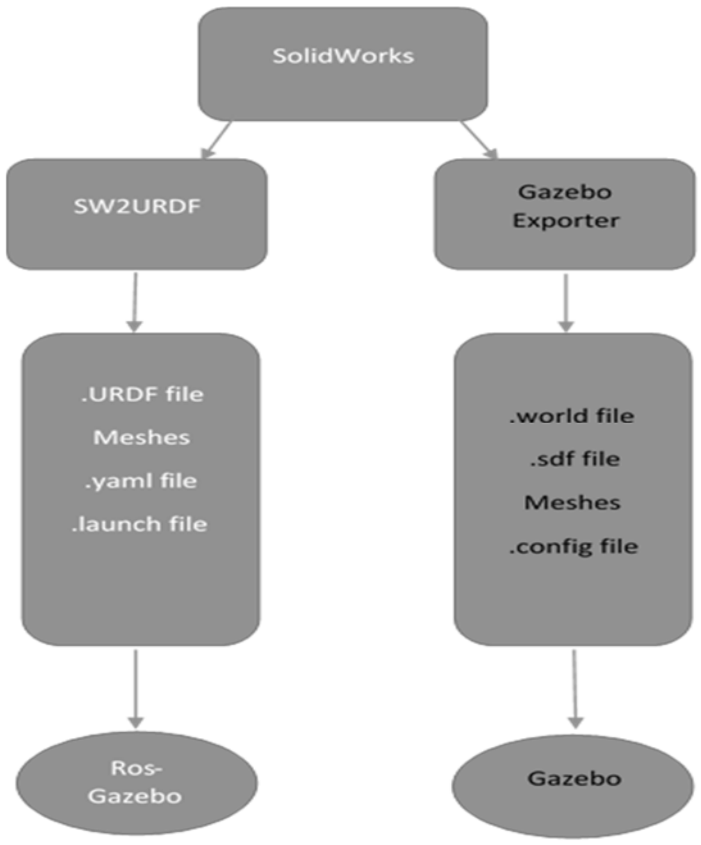


Figure IV.3 *Flow chart of Solidworks to Gazebo (Source: Self-elaboration)*

SW2URDF

The sw2urdf plugin for Solidworks helps with the task of getting the needed files to be used in the Gazebo-ROS environment. Thus it must be installed and configured conveniently. The exporter button activates it, showing the "Export to URDF" link in the File menu, as it is shown in Figure IV.4.

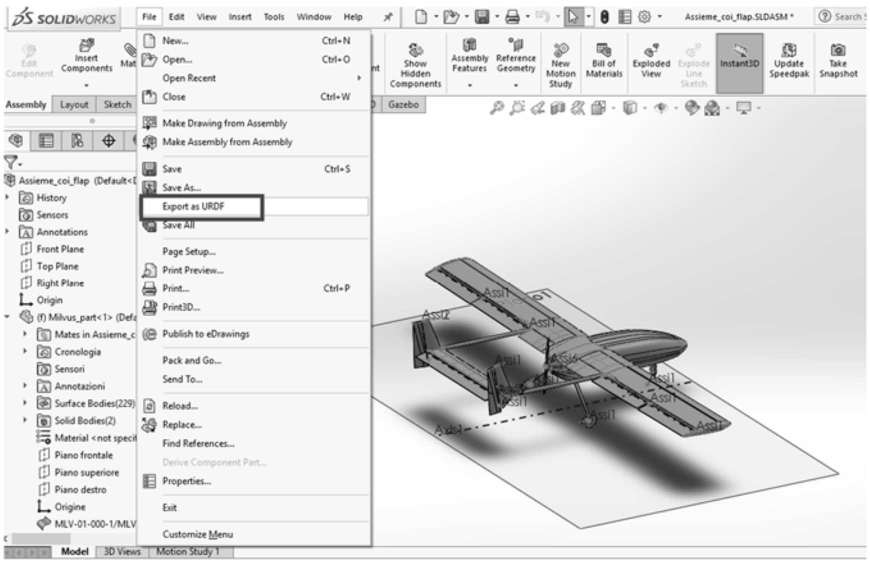
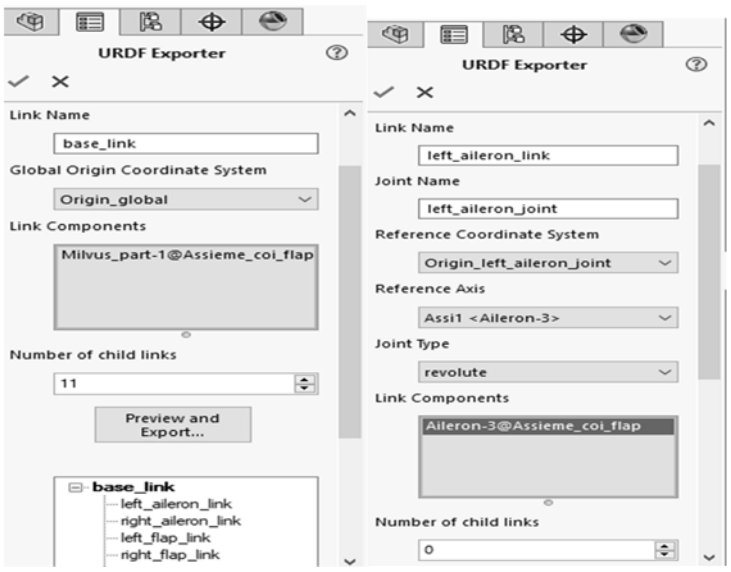


Figure IV.4 *File->Export as URDF*
(Source: Screenshot)

There are 3 phases of the exporter (see Figure IV.5); the first part appears as a Property Manager on the left side of the screen. In this part, it is needed to enter the file name for the STL associated with the link, the reference geometry, and the joints types.


URDF Exporter

Link Name: base_link
Global Origin Coordinate System: Origin_global
Link Components: Milvus_part-1@Assieme_coi_flap
Number of child links: 11
Preview and Export...

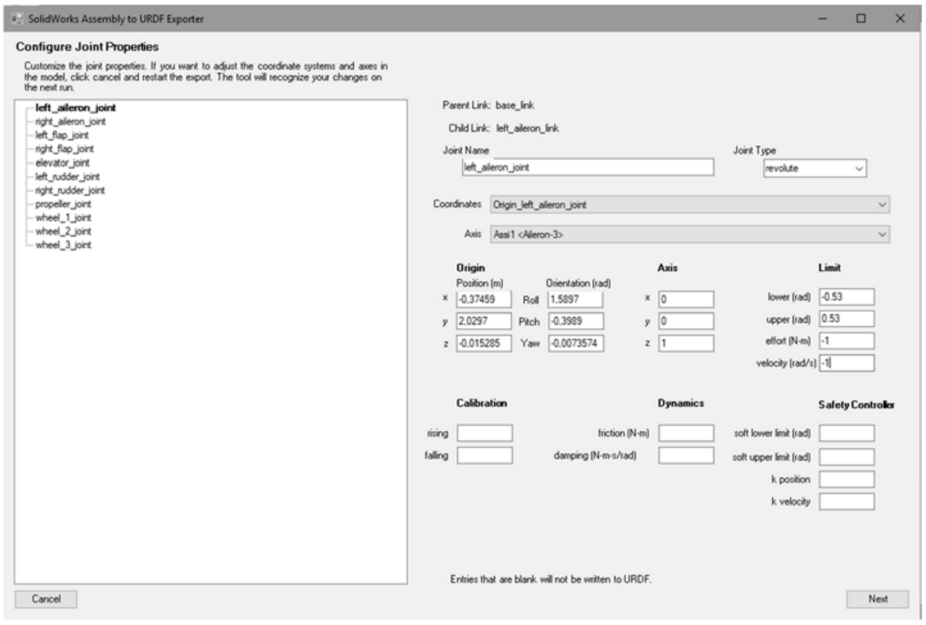
- base_link
  - left_aileron_link
  - right_aileron_link
  - left_flap_link
  - right_flap_link

URDF Exporter

Link Name: left_aileron_link
Joint Name: left_aileron_joint
Reference Coordinate System: Origin_left_aileron_joint
Reference Axis: Assi1 <Aileron-3>
Joint Type: revolute
Link Components: Aileron-3@Assieme_coi_flap
Number of child links: 0

Figure IV.5 *Exporter Property Manager*
(Source: Screenshot)

The second page of the exporter is the joint properties windows (see Figure IV.6), where coordinate reference settings for the joint can be changed, and joint limits applied.


SolidWorks Assembly to URDF Exporter

Configure Joint Properties

Customize the joint properties. If you want to adjust the coordinate systems and axes in the model, click cancel and restart the export. The tool will recognize your changes on the next run.

- **left_aileron_joint**
- right_aileron_joint
- left_flap_joint
- right_flap_joint
- elevator_joint
- left_rudder_joint
- right_rudder_joint
- propeller_joint
- wheel_1_joint
- wheel_2_joint
- wheel_3_joint

Parent Link: base_link

Child Link: left_aileron_link

Joint Name: left_aileron_joint

Joint Type: revolute

Coordinates: Origin_left_aileron_joint

Axis: Assi1 <Aileron-3>

Origin

	Position (m)		Orientation (rad)
x	-0,37459	Roll	1,5897
y	2,0297	Pitch	-0,3989
z	-0,015285	Yaw	-0,0073574

Axis

x	0
y	0
z	1

Limit

lower (rad)	-0.53
upper (rad)	0.53
effort (N-m)	-1
velocity (rad/s)	-1

Calibration

rising	
falling	

Dynamics

friction (N-m)	
damping (N-m-s/rad)	

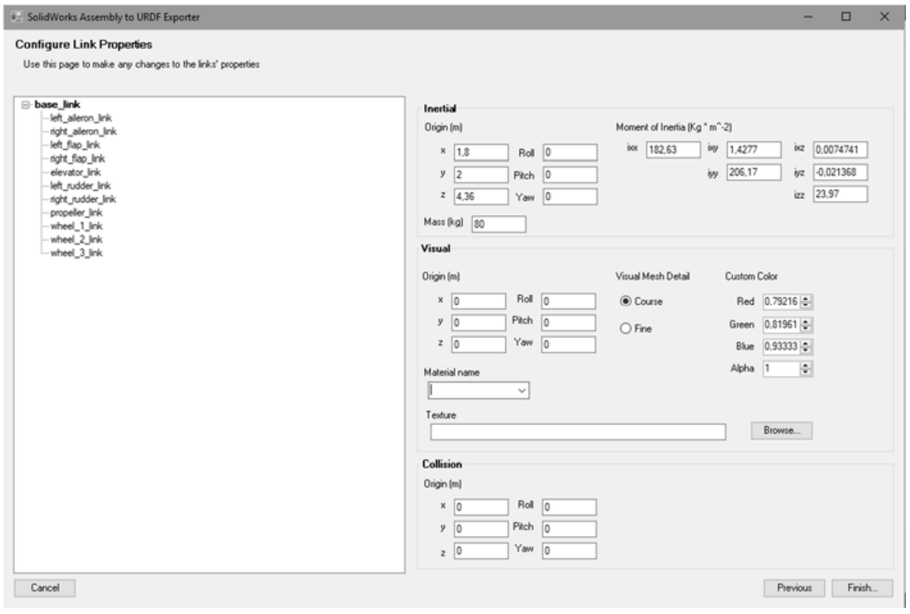
Safety Controller

soft lower limit (rad)	
soft upper limit (rad)	
k position	
k velocity	

Entries that are blank will not be written to URDF.

Figure IV.6 *Exporter Joint Properties*
(Source: Screenshot)

The third step of the exporter is the link properties Tab (see Figure IV.7). The most important feature of this step is to check if the link has proper inertia; if they are all 0, then there is a need to quit and re-export the model.


SolidWorks Assembly to URDF Exporter

Configure Link Properties

Use this page to make any changes to the links' properties

- base_link
  - left_aileron_link
  - right_aileron_link
  - left_flap_link
  - right_flap_link
  - elevator_link
  - left_rudder_link
  - right_rudder_link
  - propeller_link
  - wheel_1_link
  - wheel_2_link
  - wheel_3_link

Inertial

Origin (m)

x	1,8	Roll	0
y	2	Pitch	0
z	4,36	Yaw	0

Moment of Inertia (Kg * m^-2)

ixx	182,63	ixy	1,4277	ixz	0,0074741
		iyy	206,17	iyz	-0,021368
				izz	23,97

Mass (kg) 80

Visual

Origin (m)

x	0	Roll	0
y	0	Pitch	0
z	0	Yaw	0

Visual Mesh Detail: [✓ Course] Fine

Custom Color

Red	0,79216
Green	0,81961
Blue	0,93333
Alpha	1

Material name

Texture

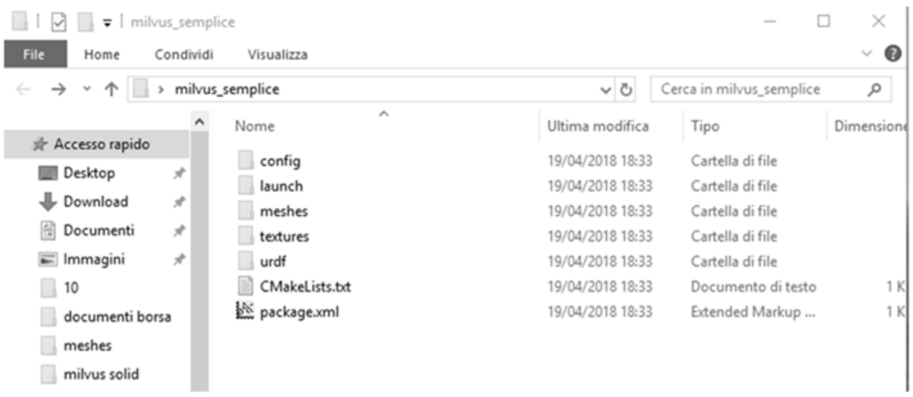
Collision

Origin (m)

x	0	Roll	0
y	0	Pitch	0
z	0	Yaw	0

Figure IV.7 *Link Properties Tab*
(Source: Screenshot)

The last step is to click "Preview and Export." The exporter generates a folder with the .urdf file, the meshes, and two launch files: the first to use in ROS environment, for an immediate visualization the model could be open in RVIZ, and the file with the scripts to launch the model in Gazebo (see Figure IV.8).

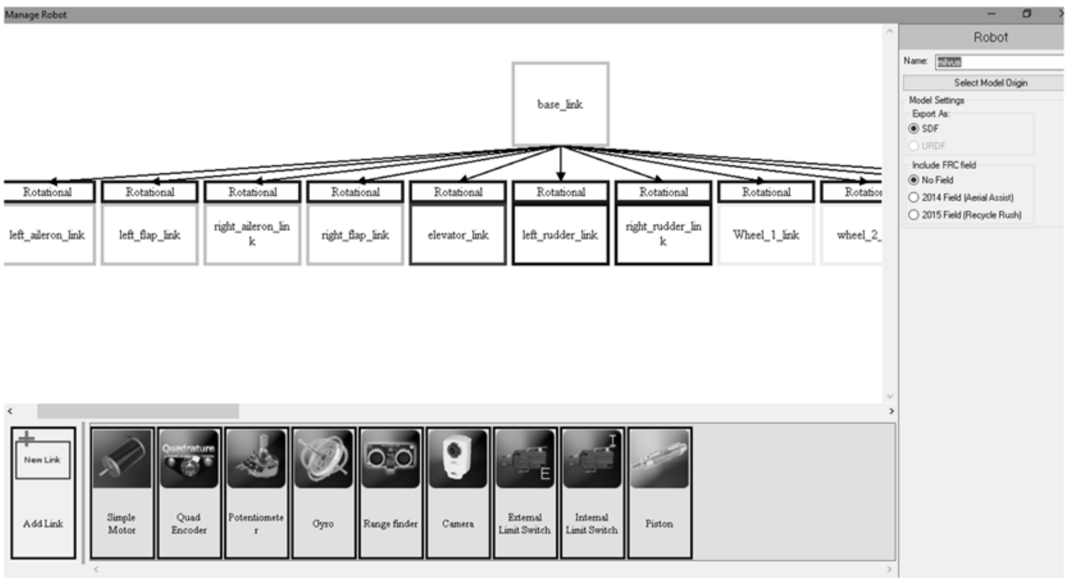


Nome	Ultima modifica	Tipo	Dimension
config	19/04/2018 18:33	Cartella di file	
launch	19/04/2018 18:33	Cartella di file	
meshes	19/04/2018 18:33	Cartella di file	
textures	19/04/2018 18:33	Cartella di file	
urdf	19/04/2018 18:33	Cartella di file	
CMakeLists.txt	19/04/2018 18:33	Documento di testo	1 K
package.xml	19/04/2018 18:33	Extended Markup ...	1 K

Figure IV.8 *Folder and files for Gazebo - sw2urdf plugin*
(Source: Screenshot)

Gazebo Exporter

In the Gazebo Export, a graphical scheme helps to choose the components of the main body (see Figure IV.9) to generate the .sdf file. First, enter the model name (without spaces), select the base plane and the axis directions. Still, on the first screen, configure the various links to the base they are attached to it. Each link needs to have a unique name, collision, and visual component (selecting the element itself from the SolidWorks model), as well as the mass and the inertia matrix. Still, it is also possible to configure sensors, cameras, or motors attached to specific links.



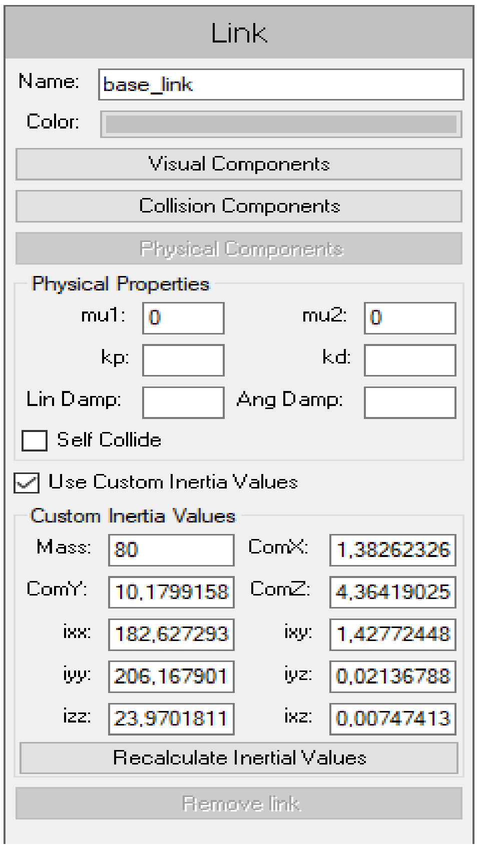
Name: milvus

Export As: [✓ SDF] URDF

Include FRC field: [✓ No Field] 2014 Field (Aerial Assist) 2015 Field (Recycle Rush)

Figure IV.9 *Graphical interface for robot model assembling from Solidworks (Source: Screenshot)*

For each link attached to the base, the joint type, axes, and limits need to be specified. The physical properties' values for each rigid body component could be set up at this moment or later in a global configuration stage. While the inertia values are generated directly by Solidworks, helping a lot in the Gazebo-ROS simulation environment (see Figure IV.10).



Link

Name: base_link

Color:

Visual Components

Collision Components

Physical Components

Physical Properties

mu1:	0	mu2:	0
kp:		kd:	
Lin Damp:		Ang Damp:	

☐ Self Collide

☑ Use Custom Inertia Values

Custom Inertia Values

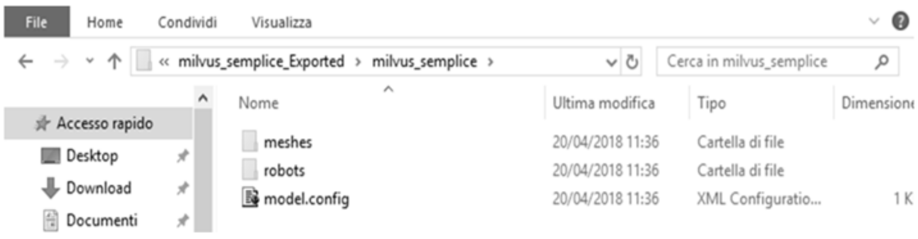
Mass:	80	ComX:	1,38262326
ComY:	10,1799158	ComZ:	4,36419025
ixx:	182,627293	ixy:	1,42772448
iyy:	206,167901	iyz:	0,02136788
izz:	23,9701811	ixz:	0,00747413

Recalculate Inertial Values

Remove link

Figure IV.10 *Link configuration Tab in Gazebo Exported procedure (Source: Screenshot)*

Once added all the data related to the robot, it can generate the SDF file by simply clicking on the export button. A folder containing the SDF file of the robot will be created with information related to mass, a matrix of inertia, the position of the parts, and joint. A second folder will contain STL files with meshes of all parts of the robot (see Figure IV.11).



Nome	Ultima modifica	Tipo	Dimensione
meshes	20/04/2018 11:36	Cartella di file	
robots	20/04/2018 11:36	Cartella di file	
model.config	20/04/2018 11:36	XML Configuratio...	1 K

Figure IV.11 *Generated Folder from Gazebo Exported procedure (Source: Screenshot)*

IV.3 Sensors and Sensing Techniques

The sensors and associated algorithms give the capability of sensing to robots, those most strongly associated with mobility measure the distance that the vehicle has moved, the inertial changes, and external structure in their surroundings. Two different classes of sensors exist to sense the environment: visual sensors, which use light reflected from objects, and non-visual sensors, which use various audio, inertial, and other methods.

Based on their typical usage; the proprioceptive (PC) or exteroceptive (EC) meaning internal-state sensors that provide feedback on the internal parameters or external-state sensors dealing with the observation of aspects of the world outside the robot itself; active (A) o passive (P) characteristics related to the energy direction exchange with the environment. Table IV.1 provides a classification of the most useful sensors for mobile robot applications.

Table IV.1 *Classification of sensors for mobile robots*

General classification (typical use)	Sensor/Sensor System	PC or EC	A or P
Tactile sensors (detection of physical contact or closeness; security switches)	Contact switches, bumpers	EC	P
	Optical barriers	EC	A
	Noncontact proximity sensors	EC	A
Wheel/motor sensors (wheel/motor speed and position)	Brush encoders	PC	P
	Potentiometers	PC	P
	Synchros, resolvers	PC	A
	Optical encoders	PC	A
	Magnetic encoders	PC	A
	Inductive encoders	PC	A
	Capacitive encoders	PC	A
Heading sensors (for orientation of the robot concerning a fixed reference frame)	Compass	EC	P
	Gyroscopes	PC	P
	Inclinometers	EC	A
Acceleration sensor	Accelerometer	PC	P
Ground beacons (localization in a fixed reference frame)	GPS	EC	A
	Active optical or RF beacons	EC	A
	Active ultrasonic beacons	EC	A
	Reflective beacons	EC	A
Active ranging (reflectivity, time-of-flight, and geometric triangulation)	Reflectivity sensors	EC	A
	Ultrasonic sensor	EC	A
	Laser rangefinder	EC	A
	Optical triangulation (1D)	EC	A
	Structured light (2D)	EC	A
Motion/speed sensors (speed relative to fixed or moving objects)	Doppler radar	EC	A
	Doppler sound	EC	A
Vision sensors (visual ranging, whole-image analysis, segmentation, object recognition)	CCD/CMOS camera(s)	EC	P
	Visual ranging packages		
	Object tracking packages		
Legend: A, active; P, passive; P/A, passive/active; PC, proprioceptive; EC, exteroceptive.			

Source: Introduction to Autonomous Mobile Robots (Siegwart et al. 2011 p.104)

The sensing techniques for vision sensors are based on the fields of computer and robot vision in order to build computer representations of the environment from light. Then to interpret this information, artificial intelligence studies deals with the task of reasoning and motion planning based on the resulting environmental representation.

There are different visions' techniques for the extraction of salient components of the image and the scene when the active vision system is implemented, like depth information, or to identify image points that are important to the task at hand, corners, echoes in signals or other features by the use of detectors algorithms; rather than processing the entire image blindly as it is the case when images are captured by passive video cameras.

The reduction of the three-dimensional position of a point to its two-dimensional projection within a camera is a core process to determine the direction and the length of the environment objects; the most extensive technique in computer vision and robotics use "pinhole camera model" described by a 3×4 matrix called the "projection matrix", other methods uses "parallel projection", "weak perspective" or "scaled orthographic camera".

Another fundamental visual tasks in robot vision is matching two or more views of the same object by its color histogram ((Swain, M. J., & Ballard, D. H. (1991), Engelson, S. P., & McDermott, D. V. (1992)), or to use a patch or window as the measurement rather than a single pixel correlation, trough the cepstrum and cepstral analysis as a tool for detecting echoes in signals with Fourier transform, feature detectors or interest operators which applies some simple heuristic to identify image points that are important to use these points to represent the image like corners, "Moravec interest operator" is one of most famous (Moravec, (1977)), Harris corner detector (Harris, C., & Stephens, M. (1988), SIFT Scale-invariant feature transforms (Lowe, D. G. (1999, 2004)), SURF Speeded up robust features (Bay H. et al. (2008)).

The techniques are many for each type of feature detection, as "visual targets" by binary acquisition target (BAT) (Maitland, N., & Harris, C. (1994)), "edge detection and extraction" by techniques identifying image locations that are associated with a significant change in intensity, the edge detectors includes operators as Laplacian of the Gaussian, generalized Hough transform, Line approximation and others to find and identify local maxima/minima (known as zero-crossing, peaks) in the image. Finally, some other techniques to match the convolved images directly (Jones, D. G., & Malik, J. (1992)) is suggested rather than compare two images by convolving the images with filters, extracting the edges, and then comparing them.

Sensing and ROS perception - main features

To interact with the environment, it is necessary to have detection and activation instruments, it will be the capabilities of the robot and the mission (indoor or outdoor navigation, mapping, grip, facial recognition) or expectations of achievement of objectives (real-time or expected precision) that will guide the selection, the number, and the precision capabilities of sensors and actuators.

In ROS, the support for computer vision is provided by means of camera drivers for different kinds of cameras like FireWire, USB or Gigabit Ethernet cameras; the integration of OpenCV libraries; tools to set the frame transform (tf) of the optical camera frame with respect to the robot; and a number of third-party drivers and tools, which comprise algorithms for visual odometry, augmented reality, object detection, and perception.

ROS comes with an image pipeline that permits the conversion of RAW images acquired by the camera into monochrome (grayscale) and color images; also uses the distortion coefficients computed during the calibration process. The stereo vision capability lets to obtain depth information from the world by computing the disparity image of the baseline between the left and right cameras, up to some extent, and with certain conditions. It is also possible to inspect that information as a 3D point cloud once it has been fine-tuned in order to get the best quality.

The ROS image_view package and the visualization nodes are available for monocular and stereo vision; there are some new stacks like viso2_ros wrapper of the libviso2 visual odometry library or fovis package. Visual odometry requires good cameras; however, it is possible to improve the results with RGB-D sensors, such as Kinect, or even sensor fusion in the case of monocular vision.

Gazebo and Ros have add-ons for most of the sensors and actuators used. If a different type or brand of the sensors, motors, or other components is needed, it is possible to modify an available complement, changing the characteristics that are normally described in the card component. These definitions are usually implemented in the call files ".launch" or. "World", or ".yaml".

In the vehicles tested in our framework, we use a lidar laser distance sensor 360 LDS-01 and IMU with three axes for gyroscope, accelerometer, and magnetometer. We define their geometries and the location using links and joints characteristic of both in the files ".urdf.xacro" and ".gazebo.xacro" associated with the complements "libgazebo_ros_laser.so" and "libgazebo_ros_imu.so" respectively. We also use the differential driver add-on called "libgazebo_ros_diff_drive.so".

IV.4 Positioning and Transformations Techniques

One of the primary and initial functionalities of a mobile robot is the ability to estimate its position and orientation concerning a stable reference frame, generally the inertial system because Newton's laws that are typically used in control models are valid concerning the inertial system. In order to do this, several techniques have been developed, the simplest being based on a series of internal measurements obtained by orientation sensors.

Table IV.2 below shows a summary of the types of conventional positioning estimators, the means of measurements obtained, the methods, techniques, and kind of sensors used. As we could see experimentally and in the literature, there is no method, technique and/or sensor that can give an exact result or absolute position, this will depend on several internal and external factors so in general one or more of these techniques are used depending on the nature of the mission and the type of mobile robot that is being modeled, simulated or controlled. For applications in mobile robotics, it is necessary to consider the characteristics of the sensor such as immunity to variations in environmental conditions, robustness to vibrations, size, consumption, wear and safety of operation, as well as its main features such as resolution, precision, and reach.

Table IV.2 *Positioning - Methods, Techniques, and Sensors*

Types of Estimators	**Type of measurement**	**Methods and Measurement Techniques**		**Sensors**
Explicit	Internal measures	Odometry		Doppler sensors
				Optical encoders
		Inertial Navigation		Gyroscopes
				Accelerometers
	Transmission stations	Fixed	Triangulation Trilateration	Ultrasound
		Mobile	GPS phones	Infrared
				Radiofrequency
Based on the perception of the environment	Brand-based Positioning	Artificial marks (conveniently added)		Inductive
				Thermal
				Chemicals
				Infrared
				Video-cameras
		Natural marks (geometric)		Video-cameras
	Positioning based on Maps	Construction of maps		Ultrasonic systems
		Data Comparison techniques		Laser systems
		Topological and geometric maps		Video-cameras

Source: Una introducción a los robots móviles (p.31)

The explicit estimators try to determine the location of the mobile robot through various techniques that use internal measures, generally with equipment shipped in the robot and through external measurements, located in the environment. Among them, the odometry for the case of mobile robots that use wheels, which estimates the position and orientation of the vehicle through the number of turns of the wheels, through an integration of the movement in time. This technique is simple, inexpensive and allows rapid sampling; however, there is an accumulation of measurement errors, due to the inaccuracies related to the wheels, when they slip or when there is wear, so they need to be calibrated; The inaccuracies are also due to irregularities in the terrain and variations in the cargo transported.

Another essential group is the estimators based on the perception of the environment, which uses active sensors, so-called because they have some energy such as ultrasound or laser, and passive sensors, which are limited to capture the energy of the medium such as video cameras and infrared sensors. Among the techniques used is the estimation by distinctive marks or beacons, which are characteristics of the environment that can be recognized by the mobile robot through its sensors. Two types of environment estimators are used, the marks or beacons and the maps, the first ones are objects or characteristics of the environment that facilitate the navigation of the robot, while the maps that can be CAD models of the environment previously made or constructed while navigating they allow to recognize the static objects of the environment that will serve to identify the position of the robot.

The estimation by means of natural and artificial marks is made using the techniques of triangulation with the objects identified in the environment, from the measurements of distances, angles or both; thus the number of required marks will depend on the possibilities of triangulation from these, that is, of the absolute reference angles, observed angles between marks, distances observed between marks, and angle and distance to a mark. The distinction of characteristic features of natural marks is one of the main problems of this method.

The techniques used are Computer Vision, which identify natural objects through vertical segments that stand out in the environment, such as columns, doors, relevant light sources, etc.; while the artificial marks are conveniently arranged in the environment, in a fixed position, and may include information additional to that of their shape (usually geometric). The other technique is the Navigation Line, which is a continuous mark, detected through sensors of electromagnetic, optical (reflection), thermal or chemical type; the restriction of the immediate vicinity between the robot and the line to be followed is still observed, the sensors must be very close with the consequent problems of limitation in the movement.

The estimation by maps of the environment, known as "map matching" is a technique that compares a previously created global map loaded in memory with a local map that is acquired from the environment while the robot moves and obtains information from it through its onboard sensors. Using data comparison techniques, they search for and find direct correspondences or through a set of similar characteristics, then the calculation of the position of the robot begins.

The information of the environment to generate the local map is obtained from sensors with distance measurement capabilities with different methods, among them, for example, flight time, phase displacements, and frequency. It will be useful for navigation if the map is mostly static, there are few elements or objects in the environment, and they are easily identifiable and characterized. The precision of the position depends in the first instance on the quality of the sensor or the sensors, the computational capacity for the census and information processing, the fusion of data from various sensors and the automatic generation of the model of the environment (degrees of abstraction), in the second instance of the robot's autonomous exploration capacity, of the exploration strategy, with movements that maximize the area covered and time; and finally of the comparison algorithms by feature extraction or iconic comparison.

Positioning in ROS - main techniques and modes of implementation

To correctly position the robot and its components in his working environment, and keep track of them, ROS uses the tf library, a core library in the ROS ecosystem, which can provide the resulting transformation between different coordinate frames, handling estimated errors in case of noise or other latencies.

The tf does the transformation in two stages, the first receiving and processing information from different sources (sensors, actuators) at different frequencies and with different frames of coordinates system at each time, call Stamp. The tf can manage synchronous (continues data source) and asynchronous information (latencies, delays, and packet drops), then the Stamp can process according to the reference tree it builds in advance, which can remain fixed or change over time, meaning that it can manage dynamically. The second receives the transformation and stores it for further use, releasing it when requested.

Since the listener does not generate future values, he will use the Spherical Linear Interpolation (SLERP) to approximate the movement of the joint between the two samples even if they are time-spaced, increasing the robustness of the system even when there are lost packets. The tf achieves a compromise between the precision achieved with higher stamp collection frequencies and bandwidth consumptions (Foote T. 2013).

IV.5 Path Planning Techniques

Path planning for a robot seen as a point

Planning a path from the current position to a goal position involves identifying the starting position and the final position, or rather the "space" and "position" of each. The pose would be represented by s = (x, y) and s'= (x', y ') both in a plane (space R ^{2}), also generally referred to as the initial state "q" and "goal" for the final position, the latter must be a free space (C_{free}). Between both positions, there must be continuous free spaces (C_{free}) that allow tracing a continuous path from start to end position (Dudek, G., & Jenkin, M. (2010)). Several solutions have been created with mechanisms that allow finding possible paths through free spaces (Lozano-Pérez, T., & Wesley, M. A. (1979), Laumond, J. P., Sekhavat, S., & Lamiraux, F. (1998), Hwang, Y. K., & Ahuja, N. (1992)). among them, “the shortest path.”

C-space (Configuration space)

Although a mobile robot could be wholly represented and controlled as a point moving in an infinite plane, there could be a need for a more general representation to incorporate the complexity of those UVs whose acting systems are distributed among its components. Each component is identified by a point, usually the center of mass of each, which will be used as a point of reference of the body component (Dudek, G., & Jenkin, M. (2010)). All those component bodies are added to the main body by joints with an appropriate type. For example, the ailerons or the propeller in the aircraft, both have their central points that are identified with a coordinate frame, then attached to the main body by revolute joints with acceptable upper and bottom angle limits.

In this way, a vehicle that can move and rotate can entirely be represented as q = [x, y, θ] concerning a fixed reference axis F_w (generally the world frame), for a robot with several components the structure of q will be more complicated. The configuration space (or C-space) represents all the possible kinematic states of a robot; this space has a dimension for each of its degrees of freedom (Kavraki, L., Svestka, P., & Overmars, M. H. (1994)). Since the set of poses q = [x, y, θ] includes a rotation as the third dimension in its state (θ), it is said that it could generate non-Euclidean spaces (the topological connection of its points in space may not have the properties of flatness), for example when it has the possibility of rotating 360° defining a cylindrical space.

The obstacles in the environment can limit the set of possible C-spaces of the robot, that is, they will not allow specific configurations of the space of the robot and instead give rise to the creation of C-obstacles, and the union of several of these spaces with obstacles give place to the so-called "C-obstacle region". Thus, the intersection of this region with the potential position of the robot will form the free space C_{free}. It is also possible that semi-free spaces are generated when the robot touches the limits of the obstacles without penetrating them (Dudek, G., & Jenkin, M. (2010)).

In addition to robots and obstacles spaces considerations, the robot itself has holonomic and non-holonomic restrictions. The holonomic's ones also restrict the possibilities while planning paths, since despite having the adjoining free space, it can be given that the robot cannot reach it because it is mechanically prevented from doing so (such as lateral displacement in vehicles with conventionally fixed wheels).

It can be observed that finding free spaces to plan a route can become a complicated task if the environment is saturated with obstacles or if the mobile robot has significant restrictions, so it is usual to make simplifications to reduce this complexity of the representation of the C-space. The most classic approach is to assume that the robot is a point, including real information about its sizes and shapes and its non-holonomic restrictions in the process of executing the path. For manage it, a mechanism is used by which the obstacles are dilated or increased according to the radius of the robot (taking the largest dimension of this), this process is known as "Minkowski sum" (Varadhan, G., & Manocha, D. (2004, October). However, it is not guaranteed that complexity will be reduced since it will depend on other factors, such as representing obstacles as primitive polygons.

The problem of path planning in mobile robots is summarized to find a path "τ" that allows the robot to arrive at the "goal" state from an initial state "q." The solutions to the problem of path planning are based on probabilistic estimates, which emphasize certain aspects of interest and make simplifications and assumptions regarding other aspects of the environment. Thus, algorithms are constructed based on different theoretical assumptions and requirements. Those are based on the relationship between the structure of the environment and the capabilities of the robot (form, means of locomotion), the solidity, and the guarantee of the proposed path. All those guarantees to have a path and being free of collisions; the cost of the path vs. an ideal path; and other considerations such as storage space or computing time used to find the solution.

Discrete search space

There are several approaches to constructing the path (Latombe, J. C. (2012)); one of them is the representation of free space as a representation of "*graph*" networks with adjacent interconnected cells, the cells susceptible to generating a robot-obstacle collision are removed from the graph. In this way, the planning of the path has only to deal with the task of finding the most efficient route between the initial position and the goal position through the *cells* that were available in the graph. This method of representing the work-space is called *graph-based*, and the technique that produces the least-length path is called *V-graph* (Visibility graph planning). The techniques that generate the graphics are diverse and are also expensive in computational terms, so there are several algorithms designed to improve them, including the tangent graph algorithm.

Other approaches such as Retraction Methods try to reduce the dimensionality of C_{free} to a one-dimensional subset of itself; among them is the generalized Voronoi diagram that has the useful property of maximizing the space between points and obstacles; however, the routes are usually long.

Other than methods, there are several techniques and search algorithms for C-space construction, as well as, many of them for C-space path planning development. One mainly used is the general search algorithm "Graph search," which determines whether a path exists from the start node to the target node. The algorithm works by maintaining a list of nodes that have been visited (CLOSE) and a list of nodes that have been visited but can directly or indirectly lead to the goal (OPEN) using the "state transition" function. The algorithm continues to visit the directly adjacent nodes until the target is found (in which case the variable "found" is true), or it remains empty then is set as OPEN (in which case "found" is false). If it cannot find a solution, this method starts an exhaustive search that takes a long time. Different variations to this method try to improve the efficiency, among them, the techniques "depth-first search," "Breadth-first search" including search costs according to the proximity by the number of nodes visited or other types of metrics (Korf, R. E. (1985)).

All mobile robots' system has been incorporated some system to avoid obstacles (local generator of trajectories). These vary in complexity, going from the primitive algorithms that detected an obstacle and stopped the robot at a short distance from it in order to avoid a collision until arriving at the algorithms more sophisticated than, for example, allows the robot to surround the obstacle to reach the destination point. The task gets complicated when it comes to dealing with everyday external environments, not only for the extensions, as we had already highlighted, but also for the dynamics they present. For example, when it comes to UAV or UUV that must travel extensive areas by air and water, the environment while traveling cannot be controlled; even UGVs like rovers when moving outdoors can find holes, stones, or other objects that they should anticipate and avoid promptly.

Path planning with Potential Fields

Another route planning alternative is through the simulation of the work environment as if it were an electric field, in which the robots and obstacles are assigned electrical charges and when they approach each other they must be repelled, thus the search for free spaces goes faster, by not having to do it exhaustively cell to cell. The robot sees as an electrical particle acting under the influence of a potential field U that is modulated to represent the structure of the free space. The resulting scalar potential field is used to represent the free space. The attraction to the goal is modeled by an additive field, which in the absence of obstacles, attracts the electrically charged robot towards the goal (Kim, D. H., Wang, H., & Shin, S. (2006)).

Probabilistic route planning

For complex environments or robots with various degrees of freedom, the described techniques of searching for C-space configuration spaces are not practical and could be impossible given the vast areas to be mapped, and the computational capabilities, so probabilistic methods are used for path planning. These techniques are heuristic by nature, so there is a need for determining the frequency and the limit of the different random searches that allow finding a path before surrendering and suppose that the goal is not attainable. Consequently, it is expected that the path planning performed with these techniques will be described at a high level, which is, starting from an initial location of the robot and arriving at a goal location, with some intermediate points of reference.

One of these techniques is RPP (Randomized Path Planner), one of the most sophisticated for the planning of probabilistic paths known as probabilistic route maps or PRM. The basic concept is the use of probabilistically tests, instead of trying to sample the entire C space. This algorithm operates in two phases, a learning phase, in which a roadmap is built inside space C, and a consultation phase, in which probabilistic searches are carried out using the roadmap to accelerate the search (Dudek & Jenkin, 2010).

Planning in ROS - main techniques and modes of implementation

Gazebo-ROS has various techniques in its multiple packages to deal with the critical aspect of path planning; some of them are summarized here. One of the search techniques implemented in Gazebo-ROS Dijkstra's algorithm for keeping the path based on a metric. The metric, identified and selected previously, could be a distance, the lowest-cost for each identified node. This process serves to classify the list of the set of OPEN nodes and to evaluate the nodes according to the length of the path defined based on the Cartesian axis. This algorithm, together with the previous two, belongs to the strategy of uninformed or blind search.

Other search techniques, also used in the Gazebo-ROS platform, are based on heuristic models such as "Best first search." The simplest one, this technique selects the node closest to the goal first and go back until the node of the initial position. It works fine in an environment with few obstacles, when used in conjunction with the algorithm A*, an optimal cost estimator, this model can also be used in slightly more demanding environments, through an optimal cost function from start to finish, however in larger environments with several degrees of freedom their computational cost is very high.

Dynamic programming is another general-purpose technique used by Gazebo-ROS for path planning. It is a recursive (or iterative) procedure to evaluate the minimum cost path to any point in the environment from some starting point, and the optimal path can be achieved by optimizing the intermediate segments. Also, Bug 1 and Bug 2 algorithms, guarantee to find a route if the objective position is accessible. To success, the algorithm must develop a robot's abilities to know when it has returned to a specific point in space or the straight line to the goal, as well as, the ability to accurately following the limit of an object. For example, when the robot is contouring an obstacle to know when it has been returned to the intersecting point with the straight line to the goal position. Failure to comply with these requirements leads to the collapse of the path planning algorithm (Dudek & Jenkin, 2010).

Therefore, with the techniques described, we can see, for example, how Gazebo-ROS implements them in one of its main macro-packages related to our study, which is the stack Navigation. One of the packages included in the stack is the planning one, in which he uses the cost-based approach with the A * algorithm. The costmap in ROS is a two-dimensional grid of cells that represent the map and the location of known obstacles. In each of these cells, a value is registered that describes the condition of it, meanly: free (0), occupied (1), or unknown (-) as we saw above.

Later, the global planner to plan an optimal path from the initial pose to the goal pose uses this costmap. However, even though the generated route is based on a known map with the static obstacles already identified in the costmap, these objects could have been moved, or there may be dynamic objects in the environment that the robot can find along the route when executing the path plan. Therefore, the local ROS planner will also use the costmap in addition to the global path plan to generate instantaneous and reliable local path planning, so when the speed commands that move the robot through its immediate vicinity are produced, it will avoid the obstacles.

The techniques used in global and local planning are completed in ROS using the Dynamic Window Approach (DWA) in which the possible range of velocity commands is sampled, and the robot's progress is simulated in time (Fox, D., Burgard, W., & Thrun, S. (1997)). The results of the advanced simulations are compared with a cost function that has adjustable parameters based on the distance of the obstacles, the progress towards the goal, and the proximity of the plan. The set of speed commands that have the lowest cost is selected and sent to the base of the mobile robot. Usually, the planner runs at a speed of 30 Hz, which allows the robot to move towards a goal while safely avoiding dynamic obstacles.

IV.6 Localization, Mapping and SLAM Techniques

Mobile robot's navigation requires the components and algorithms to perceive the environment, localize himself in it, and be able to navigate effectively in this environment. Having a map and exact location on the map permits to predict a path to a target and to navigate safety if onboard sensors are precise enough. When neither of both is available, the robot needs to build a map of an unknown environment while simultaneously keeping track of his location within it, this method is known as SLAM (Simultaneous Localization and Mapping). Then a map, the representation of the environment where the mobile robot is acting, is a central component for an autonomous system because it is used during action planning and execution. For that reason, the map needs to be available and as much accurate as possible.

Usually the 3D representation of the environment, discretize the area mapped using a grid of cubic volumes of equal size, called voxels (Roth-Tabak and Jain (1989), Moravec (1996)), the rigid grids this approach produce are large in-memory requirement because needs to be available in advance, making it prohibitive for large outdoor areas. To improve this, 3D range measurements are stored directly by modeling 3D point clouds that return range sensors, such as laser range finders or stereo cameras in the 3D SLAM systems (Cole & Newman (2006), Nüchter, A., et al. (2007)), however the number of measurements of this technique is still high with no upper limit. The models in this approach do not include free spaces nor unknown areas, and it is not possible to deal with sensor noise and dynamic objects directly. So, point clouds are only suitable for high precision sensors in static environments (Hornung, A. et al. 2013).

Then, the 2.5D maps, which discretize the vertical dimension as a function of the height of the robot, was used and proof that is enough for route planning and outdoor terrain navigation with a fixed form of a robot ((Hebert et al. (1989), Hadsell et al. (2009)). The map does not represent the real environment, since overhanging obstacles that are higher than the vehicle, such as trees, bridges or underpasses are ignored and could be not enough for localization (Hornung et al. (2013)), To overcome this problem, different workarounds were used as a list of occupied voxels for each cell in 2D grid (Ryde and Hu (2010)), a Multi-Volume Occupancy Grid approach of Dryanovski et al. (2010) and Douillard et al. (2010) combine a course elevation map for background structures with object voxel maps at a higher resolution.

Another proposed technique uses octrees for mapping, proposed initially by Meagher (1982), then Payeur et al. (1997), Fournier et al. (2007) and Pathak et al. (2007) used octets to adapt the mapping of the occupation grid from 2D to 3D with a probabilistic way of modeling occupied and free space. Fairfield et al. (2007) presented a map structure, called Deferred Reference Counting Octree, designed to allow efficient map updates, especially in the context of SLAM particle filter. Recently, Hornung et al. (2013) propose a general framework that stores clouds of unprocessed points, integrated into a map of volumetric and memory-efficient occupation (compact), which uses a tree-based representation, a probabilistic estimate of the occupation to guarantee the update and to deal with to the noise of the sensor.

An integrated approach was suggested by Darmanin, R., and Bugeja, M (2016), see Figure IV.12, mainly because nowadays autonomy in mobile robotics missions requires that different strategies need to be performed in real-time, using just sensory data and appropriate algorithms. For example, in dangerous situations, exploration strategies include map creation of as much as possible of the unknown environment, and in the shortest time. This accurate mapping requires exploitation actions, such as place revisiting actions (Makarenko et al., 2002). Thus, it gives rise to Active SLAM strategies that seek to improve the localization estimate of the robot rather than explore as much of the environment as possible in the shortest time. Both exploration and Active SLAM strategies provide a sequence of locations that the mobile robot needs to visit in order to meet the specified exploration criteria.

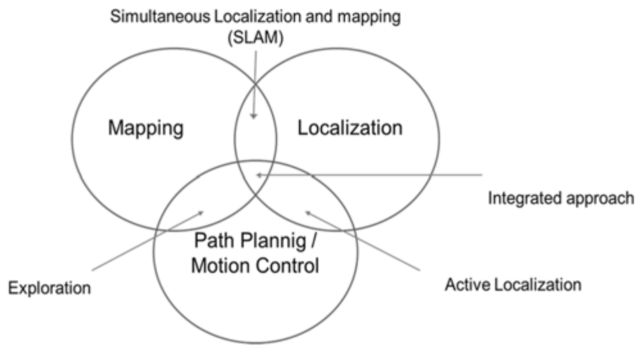

Simultaneous Localization and mapping (SLAM)
Mapping
Localization
Integrated approach
Path Plannig / Motion Control
Exploration
Active Localization

Figure IV.12 *Integrated approach for Mapping, Localization and Path Planning (Source: Autonomous Exploration and Mapping using a Mobile Robot Running ROS (Darmanin, R. and Bugeja, M. 2008)).*

The robot can use various sensors, like laser rangefinder, IMU-sensor, sonar, altimeter, a depth camera, or conventional RGB camera with appropriate algorithms. Lately, more than ten years now, SLAM methods based on computer vision algorithms are heavily implemented (Borenstein et al. (1996), Boyen, X., & Koller, D. (1998), Dellaert et al (1999), Durrant-Whyte et al. (2003), Fox, D., et al. (1999, 2000)). To generate a map many techniques are available (see Table IV.3), as for example Lisa a mobile service robot, uses the robot's odometry data and the sensor readings of a laser range finder, then the map building process uses a particle filter to match the current scan onto the occupancy grid (Wirth, S., & Pellenz, J. (2007)).

SLAM (Simultaneous Localization and Mapping) is usually solved by a probabilistic Bayes formulation, by a particle filter (Guivant, J. E., & Nebot, E. M. (2001), Gutmann, J. S., & Fox, D. (2002)). Based on the position estimate (from odometry or another estimator), the current laser scan is registered against the global occupancy grid. The occupancy grid stores the occupation probability for each cell and is used for path planning and navigation. The distance transform holds the distance per cell to a given target Navigation (Zelinsky's path transform). On the other hand, the obstacle transform keeps the distance to the closest obstacle for each cell. It enables the calculation of short paths to target locations while at the same time maintaining a required safety distance to nearby obstacles.

There are many combinations of techniques that could be used for the integrated approach, which is related to the environment to cover the available time and priorities assigned to the mission activities. One very popular is FastSLAM (Montemerlo et al. (2002)) that uses a grid map and odometry for static objects in the environment; an extension method includes dynamic environments (Avots et al. (2002)). The last one was presented in the ROS conference of 2002, it uses two occupancy grid maps, one map (S) to represent occupancy probabilities which correspond to the static parts of the environment and the other map (D) is used to represent occupancy probabilities of the moving parts; also use landmarks features (corners) with the nearest neighbor filter for localization that can be detected by the sensors of the robot.

Localization, Mapping, and SLAM in ROS

The amcl package is a probabilistic localization system for a robot moving in 2D. It implements the adaptive (or KLD-sampling) Monte Carlo localization approach, which uses a particle filter to track the pose of a robot against a known map. To estimate the robot's pose, it takes as inputs the laser-based map, laser scans, and transform messages.

The map_server is the ROS node that provides offers map data as a ROS Service. It also provides the map_saver command-line utility, which allows dynamically generated maps to be saved to file. While octomap_server provides map building and serving capabilities, the mapping approach is based on octrees and probabilistic occupancy estimation. The OctoMap library provides data structures and mapping algorithms in C++.

In ROS to perform SLAM, a ROS wrapper is available for OpenSlam's Gmapping, which implements a Particle Filter (PF), a technique for model-based estimation, with the slam-gmapping package it is possible to create a 2-D occupancy grid map from a laser and pose data collected by a mobile robot. GMapping Particle Filter (PF) is a technique for model-based estimation, in SLAM, it estimates two things: the map and the robot's pose within this map, meaning the probability of the map and the robot's pose given the control inputs (e.g., motor encoder counts) and sensor readings (i.e., LiDAR). Also, there is a motion model and a sensor model involved in the calculation of the probability distribution.

The particle filter could be adapted as it is the case of the Rao-Blackwellized particle-filter based approach developed in map an environment accurately and efficiently (Grisetti et al., 2007). In contrast to EKF-based SLAM (Kalman, R. E. (1960)), this technique uses multiple hypotheses represented by the particle set in the particle filter.

Another vital package is hector_mapping, which offers a SLAM approach that can be used without odometry as well as on platforms that exhibit roll/pitch motion (of the sensor, the platform, or both). It uses LIDAR systems like the Hokuyo UTM-30LX and provides 2D pose estimates at a scan rate of the laser sensors (40Hz for the UTM-30LX). While the system does not give an explicit loop closing ability, it is sufficiently accurate for many real-world scenarios.

IV.7 Collision checking and Recovery Techniques

In mobile robotics, one of the fundamental functionalities is the collision checking in order to make a path planning, obstacle avoidance, and safe navigation. A geometric reasoning system serves to detect potential contacts and prevent them by steering the robot away from these places. In cluttered and dynamic environments, it could be difficult to avoid every possible contact point in advance.

The collision detection methods look for determining the minimum Euclidean distance between two objects, which is a measure of proximity or penetration (Lin, M., & Gottschalk, S. (1998, May)). The solutions usually determine the minimum separation or maximum penetration.

The dimensions of the obstacles and their position in the environment can be partially or entirely unknown, located on a map if they are static. Systems to avoid obstacles are linked to the generator of local trajectories, and the algorithms that manage them have different techniques that range from stopping the march to evasion through various strategies such as surrounding the obstacle, using a variety of sensors (ultrasonic, laser rangefinder, video camera) and methods (see table IV.3).

Table IV.3 *Sensors and Methods for Collision Checking*

Method	Input	Description	Advantage	Disadvantage
Detection of edges or corners	Ultrasonic sensor Laser rangefinders	Determines the position of vertical "visible" edges of obstacles, joins two visible edges to represent their limits	Very used, simple	The robot must stop in front of the obstacle. I roll sensors, False edges and obstacles are very close or very far
Grid of certainty	Ultrasonic sensor The work area in cells or grids (two-dimensional)	Grid with degrees of certainty, updated on time. It is a function that projects probability contour with high values around the acoustic axis of the conical field of vision.	Incremental cell value allows inaccurate sensors to be used	It does not specify the angular position of the object. The robot must stop in front of the obstacle. computationally intensive

Field of Potential	Predefined geometrical shapes Ultrasonic sensors	A resultant force (accelerator) is determined to the robot, calculating repulsive forces of the obstacles and attractive forces of the target point.	many variants to the method that improves the performance of the robot's speed	Speed variation near obstacles.
Field of virtual forces (VFF)	two-dimensional cartesian grid-histogram (C) Ultrasonic sensors Low pass filter in VFF control loop	Each grid has a certainty value; only one cell is incremented in the grid for each set of readings. Probabilistic distribution, an obstacle occupies an "active region" (C *) Field of virtual forces, filled cells exert the repellents of magnitude proportional to the value of certain of the grid and inversely proportional to the distance	Avoid obstacles in real-time, control of fast movements in front of unexpected obstacles. Computationally efficient by a rapid and continuous sampling of each sensor	At lower visibility between two obstacles, the repulsive forces do not let the robot pass. Considerable fluctuations in the direction control, requires filter in VFF control.
Vector field histogram (VFH)	two-dimensional cartesian grid-histogram (C) Ultrasonic sensors Low pass filter in VFF control loop	three levels of data representation in the two-dimensional Cartesian grid-histogram C: . detailed description updated in time . a one-dimensional polar histogram h, around the robot, each grid has a "polar density of obstacles" reference values for the vehicle's steering and speed controller	Avoid obstacles in real-time, increase the detail of information regarding the VFF method, giving it more reliability	Possible errors in the selection of the reference address It requires a large storage capacity in the robot.

Impedance-based control	Sensors ultrasound Motion controller Correction function	Differentiated control in free space and restricted space, based on extended impedance (the relationship between dummy forces and modified motion error). Design of controllers with stability analysis of the control system using the Lyapunov theory of non-linear systems.	Increase system stability	Non-linear systems Due to the change of position of the moving target to avoid the obstacle, it can move away from the route completely
Control based on optical flow	A video camera Speed of movement Apparent of the brightness patterns of an image	Two discrete controllers are designed, one controls the linear speed and the other the angular speed of the robot. The dynamics of the mobile robot and the nonlinear kinematics of the video camera must be known	Preventive control strategy. The mobile robot adjusts its speed	Sensitive to changes in lighting Specially conditioned environment
Control based on a 2D½ vision	parameters and data of the video camera	2D½ vision system relates the depth coordinate (distance) between the linear laser light emitter and the position of the projection of that a point in the image through geometric equations Control strategy to generate the reference direction, the orientation of the robot, the desired speed	Preventive control strategy. The mobile robot adjusts its speed	

Source: Based on "Una introducción a los robots móviles (p.46-56)"

IV.8 Summary of ROS Methods and Techniques

As a summary, Table IV.4 shows the methods and techniques used in the ROS stacks and packages used in the UNISA-UVF.

Table IV.4 *Methods and Techniques for mobile robotics in ROS*

Method	3-D perception	Planning			Path Following	Mapping	Localization	SLAM	Collision checking	Recovery behavior
		Global Planner	Local Planner	Motion Planning						
Occupancy map monitor (Octomap)	x	x								
Dijkstra's algorithm		x			x					
Carrot planner		x							x	
A*, RA*, Anytime D*, ANA*		x	x							
Elastic Band			x							
Probabilistic Roadmap				x						
Sampling-based planner				x						
Rapidly-exploring Random Trees (RRT)				x						
Kinematic Planning by Interior-Exterior Cell Exploration (KPIECE)				x						
Dynamic Window Approach (DWA)			x		x					
Trajectory Rollout			x							
Costmap _2d, Cost-functions					x	x				
Occupancy grid map (Gmapping)						x	x			
Point clouds							x			
Adaptive Monte Carlo Localization (AMCL)						x	x	x		
Particle filters EKF						x	x	x		
Sequential Importance Resampling (SIR) filter							x	x		
Fast SLAM								x		
Graph-based SLAM								x		
Flexible Collision Library (FLC)		x					x	x	x	
Collision Matrix									x	
Conservative reset										x
Aggressive reset										x
Clearing rotation										x
Aborted (infeasible stuck)										x

CHAPTER V
Case Studies using UNISA Unmanned Vehicles

V.1 Introduction

In order to test the framework UNISA-UVF proposed, we test different missions for unmanned vehicles, ground, and aerial ones. Some of these tests are presented in this chapter; the selection is based on the different characteristics and functionalities, having in mind to offer a wide variety of potentialities instead of which approach is more relevant for each case.

For UGVs, we selected three use cases, in the first one, the main functionality of mobile robotics is presented, meaning Navigation with all the components as localization, path following, and so on. In the second use case, two UGVs are simulated in order to do a SLAM (Simultaneous Localization and Mapping) in order to cover an indoor environment while mapping this place together, creating at the end only one map. Moreover, finally, a third one to show the integration capabilities of GAZEBO-ROS with another robotics platform like MATLAB.

For the UAVs, we selected a fixed-wing aircraft simulated and controlled in two different ways. The first one launched the UAV milvus_dae only in the Gazebo simulator in order to be controlled by the keyboard and a plugin to control each joint (moving components like flaps and ailerons). The second use case uses the same model of UAV, this time in the Gazebo-ROS environment, performing an unmanned path following.

V.2 UNISA-UGV Use Cases

As we already presented in the previous chapters, the 3-D model was exported from Solidworks, making all the necessary modifications to be used in the Gazebo-ROS framework. The UGVs used in the use cases presented here looks like in the simulator as in the following Figures V.1 and V.2, while the structure of the components is represented as a frames tree (Figure V.3)

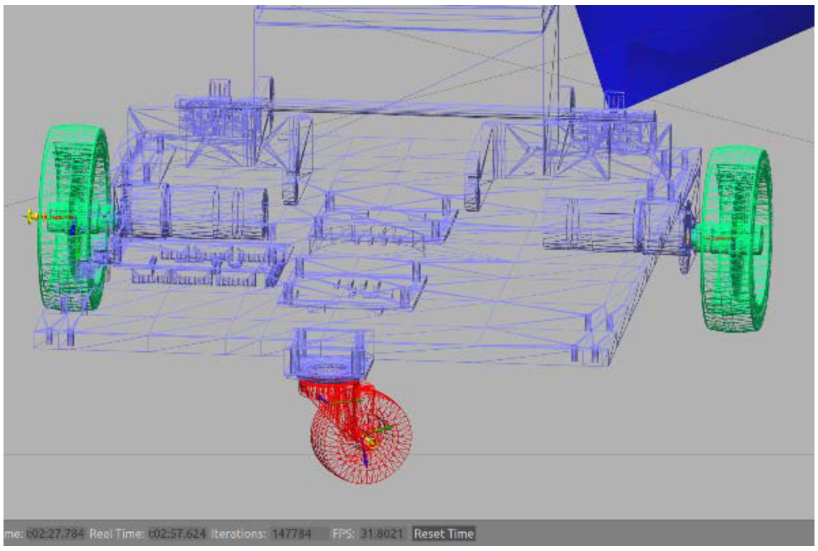

me: 02:27.784 Real Time: 02:57.624 Iterations: 147784 FPS: 31.8021 Reset Time

Figure V.1 *UNISA UGV 3-D model joints disposition*
(Source : Self-elaboration)

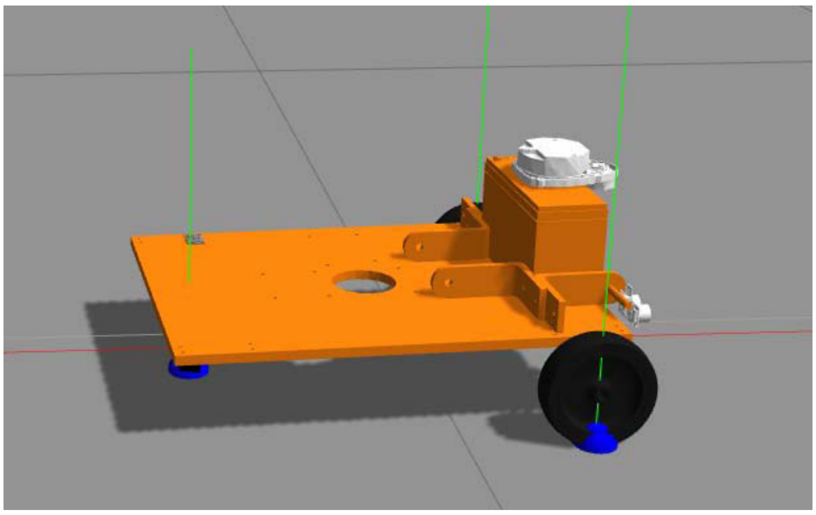


Figure V.2 *UNISA UGV 3-D model contact points*
(Source : Self-elaboration)

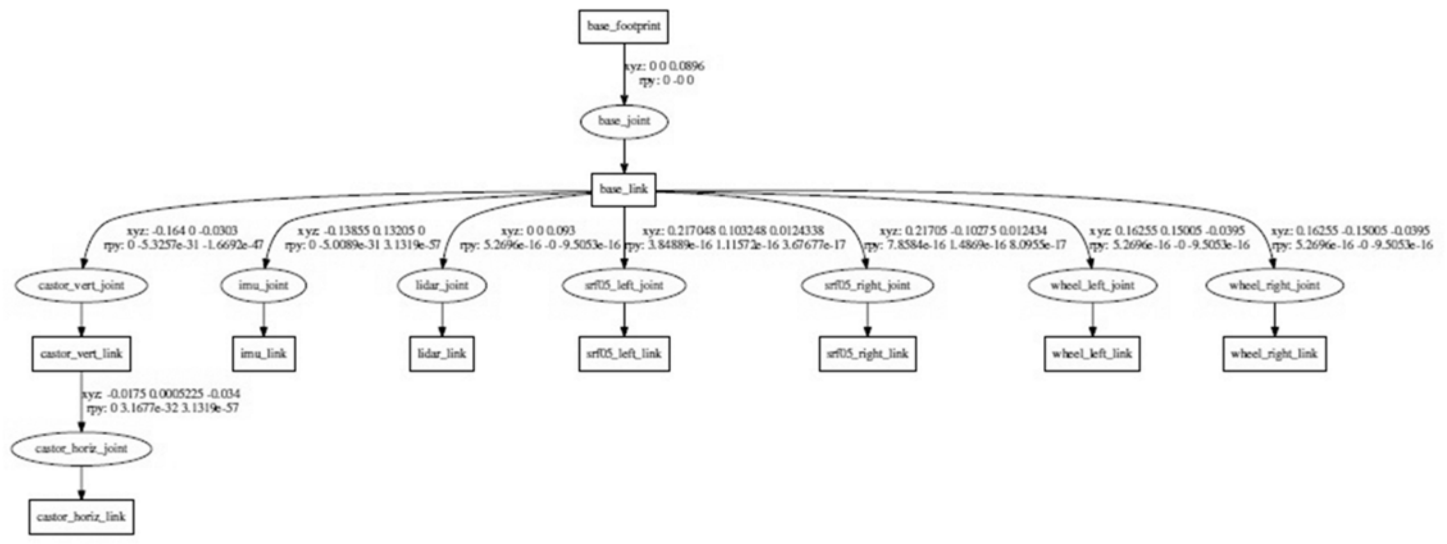

base_footprint
xyz: 0 0 0.0896
rpy: 0 -0 0
base_joint
base_link
xyz: -0.164 0 -0.0303
rpy: 0 -5.3257e-31 -1.6692e-47
xyz: -0.13855 0.13205 0
rpy: 0 -5.0089e-31 3.1319e-57
xyz: 0 0 0.093
rpy: 5.2696e-16 -0 -9.5053e-16
xyz: 0.217048 0.103248 0.0124338
rpy: 3.84889e-16 1.11572e-16 3.67677e-17
xyz: 0.21705 -0.10275 0.012434
rpy: 7.8584e-16 1.4869e-16 8.0955e-17
xyz: 0.16255 0.15005 -0.0395
rpy: 5.2696e-16 -0 -9.5053e-16
xyz: 0.16255 -0.15005 -0.0395
rpy: 5.2696e-16 -0 -9.5053e-16
castor_vert_joint
imu_joint
lidar_joint
srf05_left_joint
srf05_right_joint
wheel_left_joint
wheel_right_joint
castor_vert_link
imu_link
lidar_link
srf05_left_link
srf05_right_link
wheel_left_link
wheel_right_link
xyz: -0.0175 0.0005225 -0.034
rpy: 0 3.1677e-32 3.1319e-57
castor_horiz_joint
castor_horiz_link

Figure V.3 *UNISA UGV Frames Tree*
(Source: Screenshot)

V.2.1 Use Case 1: One UGV Autonomous Navigation – from MATLAB/Simulink

Mission:

The mission is to teleoperate the mybot05 unisa_ugv model in a virtual environment. In this use-case, we choose to use a virtual Watkins laboratory model provided by robotics inc., in a Gazebo simulator using specified waypoints in MATLAB/Simulink.

A summary of the result for most interesting steps while modeling and simulating in Matlab/Simulink and Gazebo-ROS are presented. We also include the detailed screenshots of configuration files and plots of positions, velocities, and acceleration of the UGV while performing at a specific stage. Also, a brief video recording is saved in the documentation folder.

Packages:

Simulation: gazebo_ros

Gazebo 3D simulator wrapped as a ros package to permit the communication and control facilities

Implemented physics engine: ODE

Alternative physics engine: Bullet,

Teleoperation: ugv_teleop

Joystick configuration over Ubuntu is managed by the joy ROS package. For teleoperation, an algorithm (ugv_teleop_joy) is created to personalize the linear and angular velocities, then, for control functionalities, the ros teleop_joy package is called.

Called package: joy, ros_teleop_joy

Alternative packages: teleop_twist_joy, turtlebot_teleop

Waypoints Teleoperation: From MatLab/Simulink

Matlab/Simulink installed over Ubuntu. For usage, it must be started and active in the network. Then initialized as a ROS node.

Called package: ROS Toolbox

Alternative packages

Status_publisher: rosbot_state_publisher

This ROS package allows publishing the state of any robot in the environment to tf packages. It uses kinematics tree model of the robot in order to convert the following inputs to outputs.

- Inputs: joint angles of the robot
- Outputs: 3D poses of the robot links

Rviz: rviz (visualization)

This tool provides a visualization environment, the UGV "understanding" of his environment, based on the topic subscribed selection (configuration file .rviz). Called package: rviz

Input variables or configuration files:

Environment variable:

```
export MYUGV_MODEL=mybot05
```

Model: model variable chose based on MYUGV_MODEL and MYUGV_WORLD environment variables

```
<!-- arguments -->
<arg name="model" default="$(env MYUGV_MODEL)" doc="model type
[my2bot, my3bot, mybot05, burger]"/>
<arg name="sim_world" default="$(env MYUGV_WORLD)" doc="sim_world
type [empty, house, watkins, plaza, world]"/>
```

The configuration files are:

```
ugv_description/urdf/ugv_$(arg model).urdf.xacro
ugv_gazebo)/worlds/ugv_$(arg sim_world).world
```

RVIZ:

Visualization configuration file: ugv_gazebo/rviz/ugv_simulation.rviz

Execution commands:

In Ubuntu with ROS computer, define the environmental global variable

```
$ export MYUGV_MODEL=mybot05
$ export MYUGV_WORLD=watkins
$ roslaunch ugv_gazebo ugv_gazebo_rviz.launch
```

Here, two possible options to control the UGV, the first through the joystick and the second by a connection of MATLAB/Simulink as a ROS node to send and control the waypoints execution:

1. Run the following commands in another terminal

```
$ export MYUGV_MODEL=mybot05
$ roslaunch ugv_teleop ugv_teleop_joy.launch
```

2. From the desktop screen double click the Matlab R2018a icon, or execute MatLab in any computer on the network

One started, initialize the rosnode and call the simulink

```
>> rosinit
>> simulink
```

From the MATLAB toolstrip, select HOME > Open > Simulink Model to open a new Simulink model.

Select the filename "myugv05_waypointWatkins_PID.slx"

Once loaded, click the Run button to start simulation (green play sign). You should see the XY plot that starts drawing the X-Y trajectory that the ugv follows.

To see graphically if all nodes are well configured and interconnected, run:

```
$ rosrun rqt_graph rqt_graph
```

Complementary commands:

If you want to plot the current execution, you could use the plotjuggler packages

```
$ roslaunch plotjuggler plotjuggler.launch
```

If you need to analyze the simulation behavior in another moment, you need to create bag files, executing the following in another terminal:

```
$ roscd ugv_bag/data
$ export MYUGV_MODEL=mybot05
$ export MYUGV_WORLD=watkins
$ export MYUGV_NAVTYPE=wpSimulink   ("joy" or "wpSimulink")
$ rosbag record --all --output-
name=navigation_${MYUGV_MODEL}_${MYUGV_WORLD}_${MYUGV_NA
VTYPE}_${date}.bag
```

Results - Execution Images and Plots:

<u>Simulation</u>: gazebo_ros
<u>Control</u>: waypoints and PID from Matlab/Simulink

Follows the image of the UGV with a plot of the autonomous waypoints path followed (Figure V.4 and V.5), it has a PID controlled. As we could see in the plot, at the start, the velocity in X grows as well as the acceleration in X and Z. During linear traveling, we get progressively to an excellent performance either on velocities and straightness. The images show different PID controller adjustments. The first image shows the configuration of the P (proportional) component only P= -0.05; it gives us a very stable and straight path following but runs too slowly.

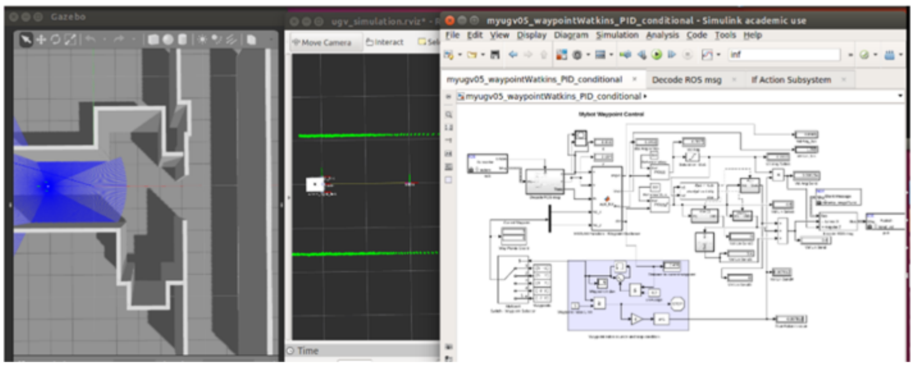

Gazebo
ugv_simulation.rviz*
Move Camera
Interact
Time
myugv05_waypointWatkins_PID_conditional - Simulink academic use
File Edit View Display Diagram Simulation Analysis Code Tools Help
myugv05_waypointWatkins_PID_conditional
Decode ROS msg
If Action Subsystem
Mybot Waypoint Control

Figure V.5 *UNISA UGV in Gazebo_ROS – Waypoints Navigation and PID control from MATLAB/Simulink at start*
(Source: Screenshot)

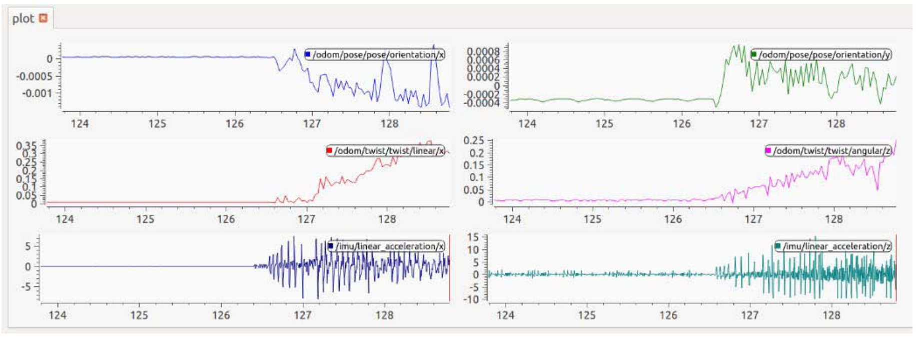


Figure V.6 *UNISA UGV Waypoints Navigation plot at the start (Source: Screenshot)*

The next image shows the second straight path traveling just after the left turn (Figure V.6), the PID configured as P= -0.05, I=-0.001, and D=-0.01 permits a better speed. Finally, the plot of all the paths followed by the autonomous waypoint navigation is presented in Figure V.7.

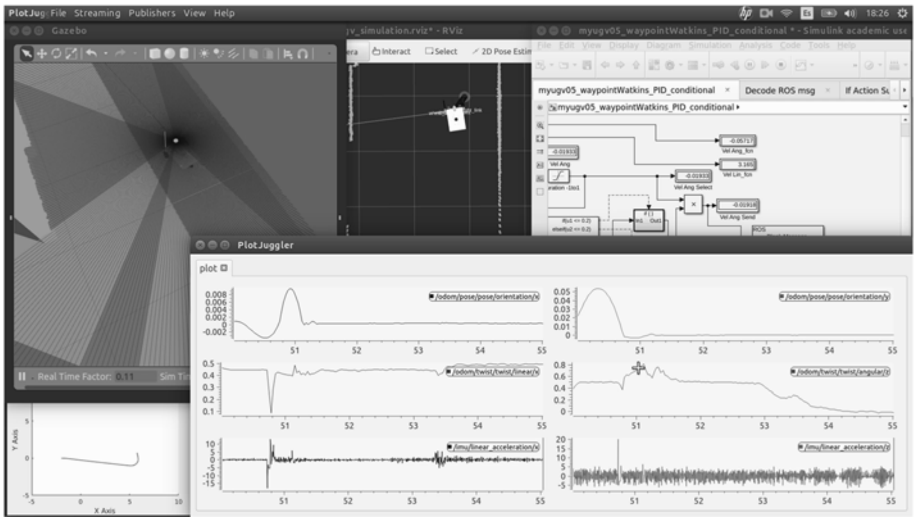

PlotJuggler
Gazebo
RViz
myugv05_waypointWatkins_PID_conditional
Decode ROS msg
plot
/odom/pose/pose/orientation/x
/odom/pose/pose/orientation/y
/odom/twist/twist/linear/x
/odom/twist/twist/angular/z
/imu/linear_acceleration/x
/imu/linear_acceleration/y
Real Time Factor
X Axis
Y Axis

Figure V.7 *UNISA UGV Waypoints Navigation plot at the turn (Source: Screenshot)*

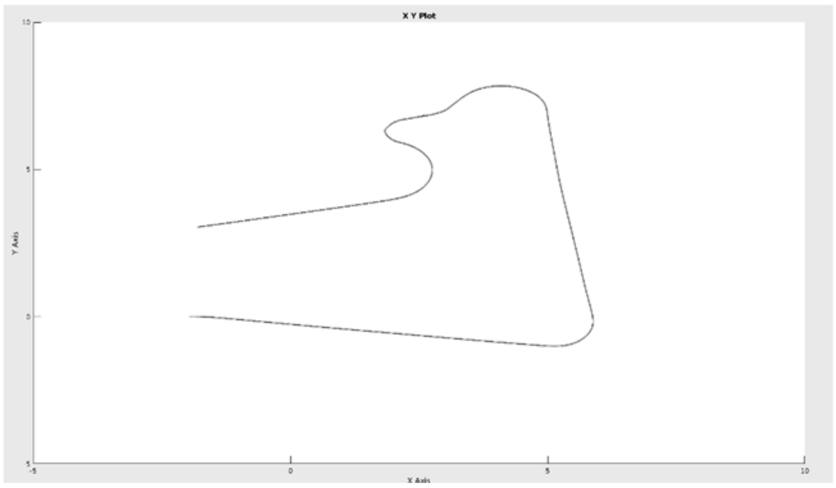


Figure V.8 *UNISA UGV Autonomous Waypoints Navigation – Path followed (Source: Screenshot)*

The PID configuration based on the linear distance to goal (dist) and rotational distance (dhdg), influences in a significant manner to the stability and performance of the UGV while performing the waypoints following. Thus, there is a need to tune PID gains, the Proportional, Integral, and Derivative to conceal a mid-point between stability and performance. Also, there is a need to manage the overall velocities to control the stability by limiting the linear and angular ones to 0.5 and angular velocity to 1.0, respectively. However, in order to increase the performance, the linear velocity was limited while during the turning by decreasing it by 10% or 20%. The maximum rate of velocity change is around 0.10 - after this range, the robot fall-down; it gets increased when the control program sends angular velocities.

V.2.1 Use Case 2: One UGV performing a SLAM

Mission

The mission is to create an indoor map using the my2bot unisa-ugv model in a virtual environment. In this use-case, we choose to use a virtual house model provided by Robotics Inc. in a Gazebo simulator and a Microsoft joystick to teleoperate the unmanned ground vehicle (UGV).

A summary of the ROS packages used, the variables and files to configure the simulated environment is presented, followed by the detailed script and configuration files, and finally, the execution of the command console displays. The objective of this document is to repeat the experiment with the same or other differential drive robot, in a kind of fully documented guide.

Packages:

Simulation: gazebo_ros
Gazebo 3D simulator wrapped as a ROS package to permit the communication and control facilities
Implemented physics engine: ODE
Alternative physics engine: Bullet

Teleoperation: ugv_teleop
Joystick configuration over Ubuntu is managed by the joy ROS package. For teleoperation, an algorithm (ugv_teleop_joy) is created to personalize the linear and angular velocities, then, for control functionalities, the ros teleop_joy package is called.
Called package: joy, ros_teleop_joy
Alternative packages: teleop_twist_joy, turtlebot_teleop,

Slam: ugv_slam
This package provides laser-based SLAM (Simultaneous Localization and Mapping), calls ROS nodes for gmapping method usage and to save the created map.
Called package: map_server, slam_gmapping
Alternative packages: Cartographer, Hector, Karto, Frontier_exploration, RTAB-Map

Status_publisher: rosbot_state_publisher
This ROS package allows publishing the state of any robot in the environment to tf packages (see Figure V.8). It uses kinematics tree model of the robot in order to convert the following inputs to outputs
Inputs: joint angles of the robot
Outputs: 3D poses of the robot links

Inputs variables or configuration files :

Environment variable :
export MYUGV_MODEL=my2bot

Model: model variable choosed based on MYUGV_MODEL environment variable
ugv_description/urdf/ugv_$(arg model).urdf.xacro

SLAM:
Configuration file: ugv_slam/launch/ugv_$(arg slam_methods).launch
Slam methods variable: name="slam_methods" default="gmapping"
Map and trajectory builder: name="configuration_basename" default="ugv_lds_2d_gazebo.lua"

Execution commands:

```
$ roslaunch ugv_collab teleop_mapping_gazebo.launch
$ roslaunch ugv_gazebo ugv_house.launch
$ roslaunch ugv_teleop ugv_teleop_joy.launch
$ rosrun map_server map_saver -f map_my2bot_house_ugv_teleop_joy
```

ROS nodes and topics:

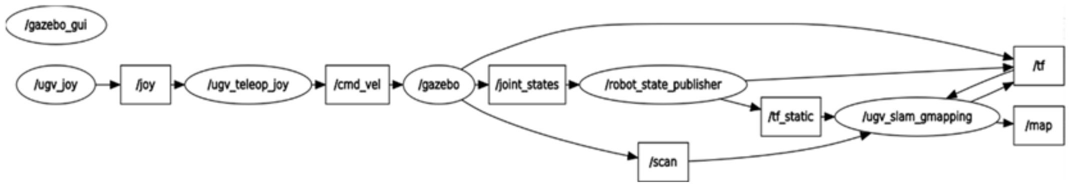
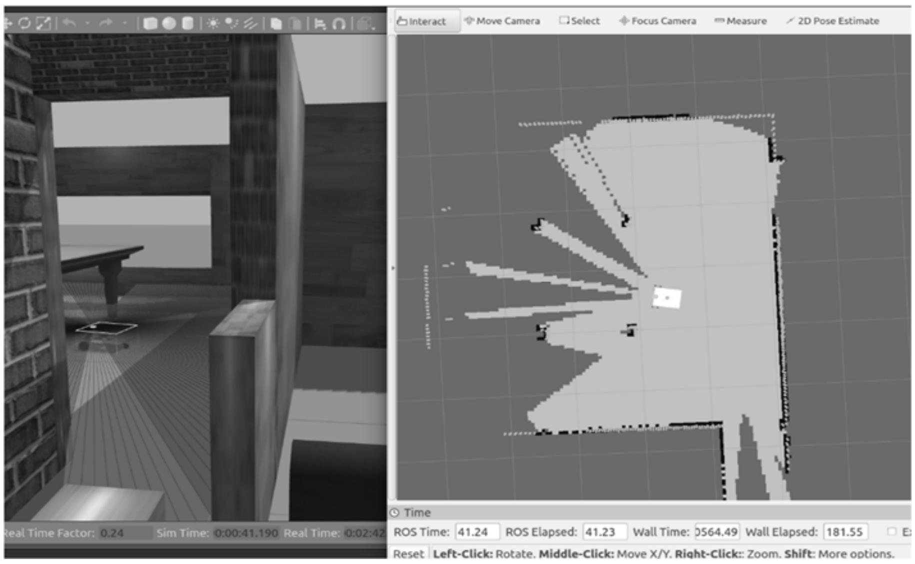


Figure V.9 *rosgraph Nodes and Topics*
(Source: Screenshot)

Results - Execution Images and Plots:

For this use case, the velocities of the joystick are modulated by a scalar coefficient, either for linear and angular velocities in order to decrease the velocities, limiting them by scale_linear value="0.5" and scale_angular value="0.2". These parameters are configurable in the launching file and are tightly related to vehicle geometry. The ROS package move_base that we had used has a PID controller and EKF (Extended Kalman Filter).

The UGV used in this case-use called my2bot in Gazebo simulator and Rviz visualization (see Figure V.9), after some traveling (launching scripts and teleoperated by a Joystick).



Real Time Factor: 0.24 Sim Time: 0:00:41.190 Real Time: 0:02:42

ROS Time: 41.24 ROS Elapsed: 41.23 Wall Time: 0564.49 Wall Elapsed: 181.55

Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click:: Zoom. Shift: More options.

Figure V.10 *One UGV performing SLAM and Obstacle Avoidance*
(Source: Screenshot)

The next Figure V.10 shows the UGV facing an obstacle; the driver stops slowly in order not to knock the table neither to fall. In the plot (Figure V.11), the angular acceleration has been controlled despite the high quick variation, as we could see in the plots.

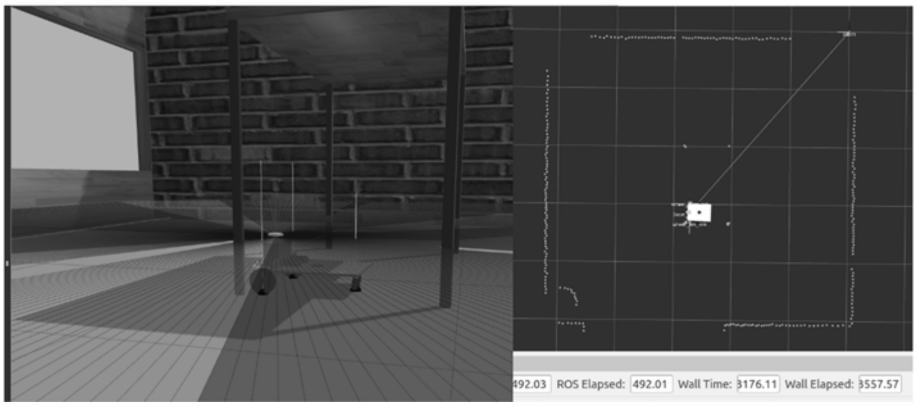

492.03 ROS Elapsed: 492.01 Wall Time: 3176.11 Wall Elapsed: 3557.57

Figure V.11 *One UGV facing an Obstacle*
(Source: Screenshot)

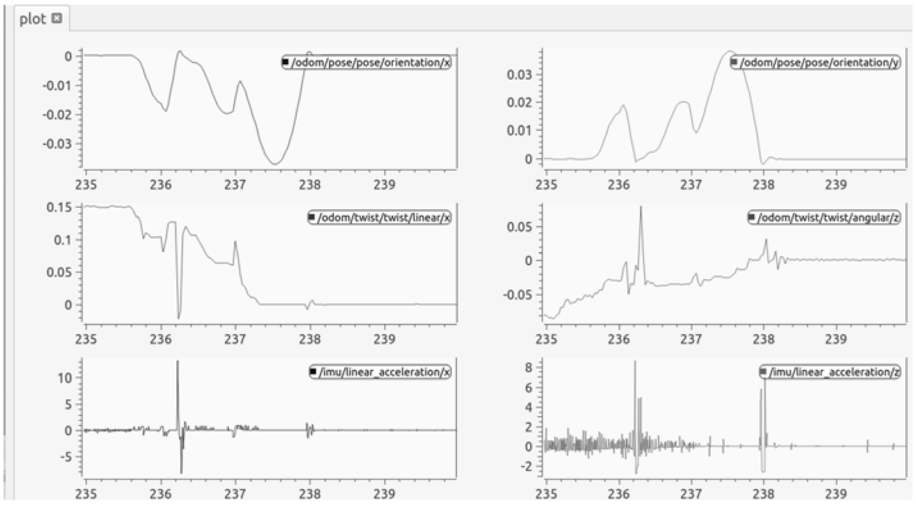


Figure V.12 *One UGV facing an Obstacle – Plots of Position, Velocity, and Acceleration*
(Source: Screenshot)

In the Figure V.12 that follows, the UGV is facing a door, in order to pass through it must do some stop and back driving, as the driver does not have enough space to turn with the maximum velocities it could get. The variation we could see in the plot graphic (Figure V.13) is due to the PID controller and

EKF gains. Despite the high variability, the UGV arrives to perform well as it could be possible to see in the video registration.

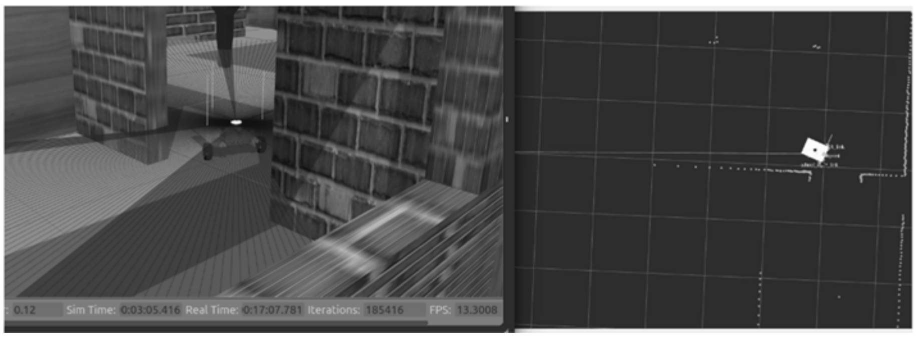


Figure V.13 *One UGV facing a door (Source: Screenshot)*

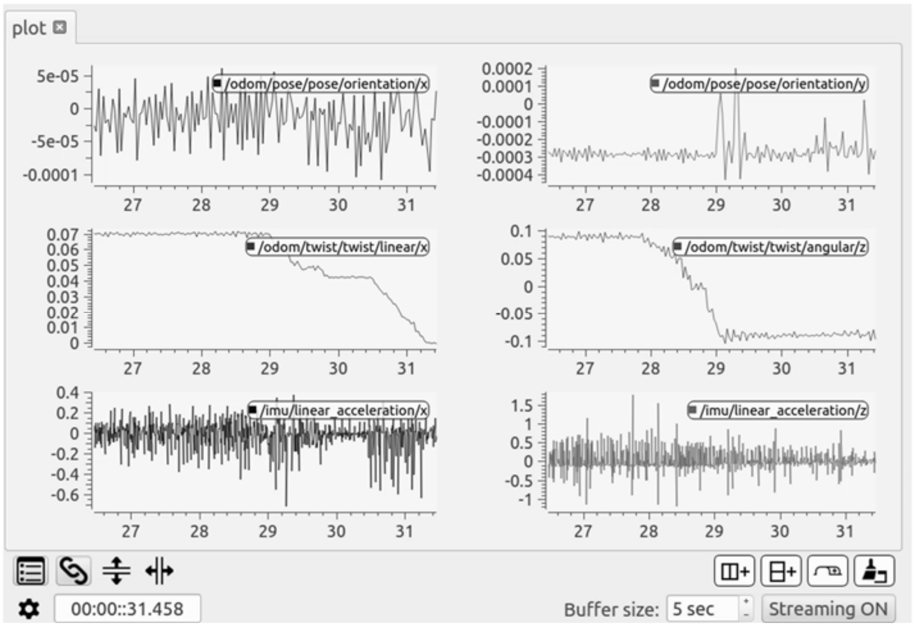


Figure V.14 *One UGV facing a door – Plots of Position, Velocity, and Acceleration (Source: Screenshot)*

V.2.3 Use Case 3: Two UGV creating a single Map

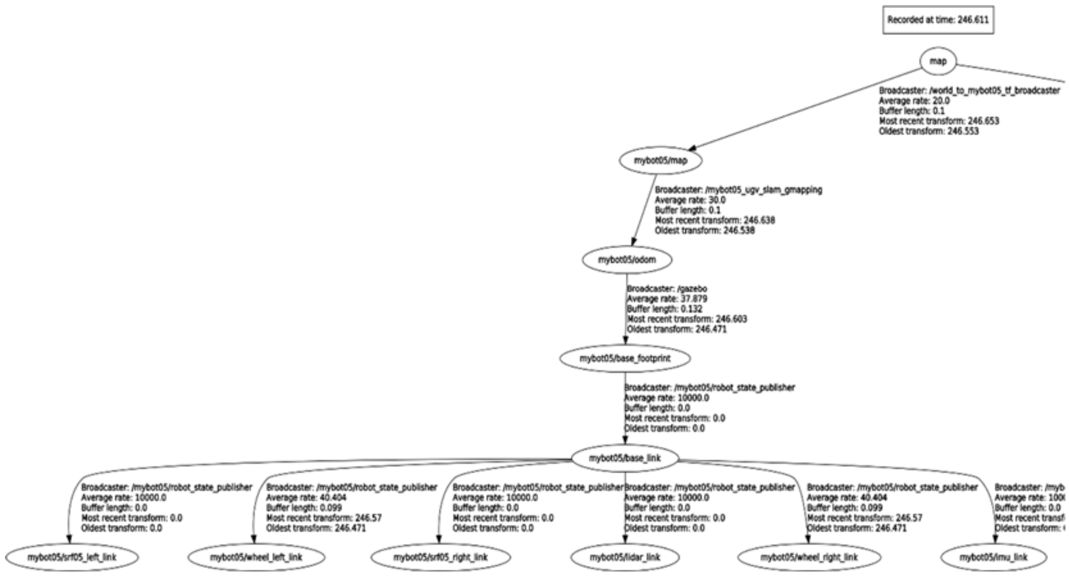
Mission:

The mission is to create an indoor map using two bots my2bot and mybot05 unisa_ugv models in a virtual environment. We will be using several ROS packages to implement, mapping, navigation, and path-planning in a collaborative way. In this use-case, we choose to use a virtual Watkins model provided by robotics inc., in a Gazebo simulator and a Microsoft joystick to teleoperate both unmanned ground vehicles (UGVs).

A summary of the ROS packages used, the variables and files to configure the simulated environment is presented in this chapter, followed by the detailed script and configuration files and finally the execution of the command console displays. The objective of this document is to repeat the experiment with the same or other differential drive robot, in a kind of fully documented guide.

Frames:

A summary of the transformation tree of the frames running when two Unisa UGVs are performing together building a map is shown in the following Figure V.14. The root is the "global map frame" as it remains fixed during the simulation period, follows the "map frame" of each UGV is creating. At this point, two branches of the tree represent the components starting by their "Odom frame," followed by the "base_footprint frame," the "base_link" frame, and each of the vehicle links (meaning the wheels and sensors).


Recorded at time: 246.611
map
Broadcaster: /world_to_mybot05_tf_broadcaster
Average rate: 20.0
Buffer length: 0.1
Most recent transform: 246.653
Oldest transform: 246.553
mybot05/map
Broadcaster: /mybot05_ugv_slam_gmapping
Average rate: 30.0
Buffer length: 0.1
Most recent transform: 246.638
Oldest transform: 246.538
mybot05/odom
Broadcaster: /gazebo
Average rate: 37.879
Buffer length: 0.132
Most recent transform: 246.603
Oldest transform: 246.471
mybot05/base_footprint
Broadcaster: /mybot05/robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
mybot05/base_link
Broadcaster: /mybot05/robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /mybot05/robot_state_publisher
Average rate: 40.404
Buffer length: 0.099
Most recent transform: 246.57
Oldest transform: 246.471
Broadcaster: /mybot05/robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /mybot05/robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
Broadcaster: /mybot05/robot_state_publisher
Average rate: 40.404
Buffer length: 0.099
Most recent transform: 246.57
Oldest transform: 246.471
Broadcaster: /myb
Average rate: 100
Buffer length: 0.0
Most recent transf
Oldest transform:
mybot05/srf05_left_link
mybot05/wheel_left_link
mybot05/srf05_right_link
mybot05/lidar_link
mybot05/wheel_right_link
mybot05/imu_link

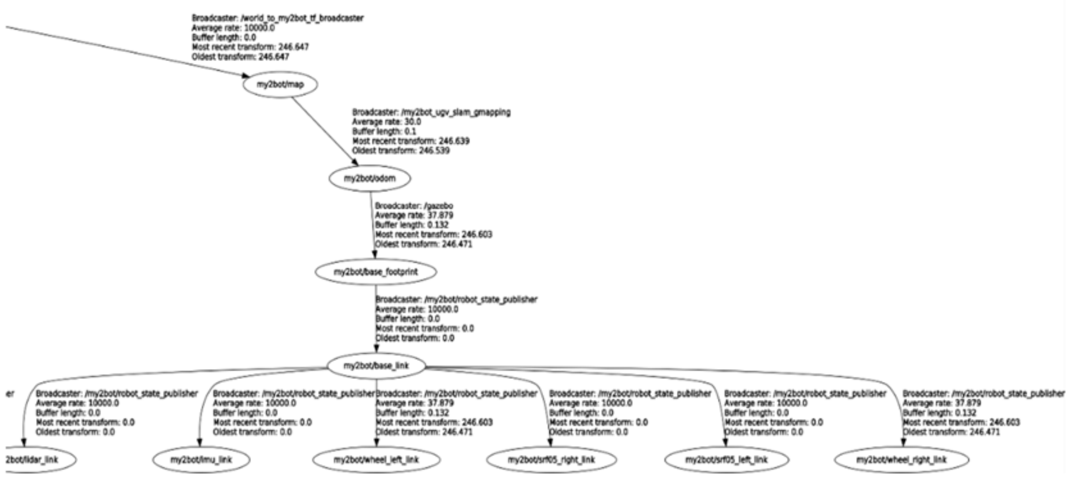

Broadcaster: /world_to_my2bot_tf_broadcaster
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 246.647
Oldest transform: 246.647
my2bot/map
Broadcaster: /my2bot_ugv_slam_gmapping
Average rate: 30.0
Buffer length: 0.1
Most recent transform: 246.639
Oldest transform: 246.539
my2bot/odom
Broadcaster: /gazebo
Average rate: 37.879
Buffer length: 0.132
Most recent transform: 246.603
Oldest transform: 246.471
my2bot/base_footprint
Broadcaster: /my2bot/robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
my2bot/base_link
er
Broadcaster: /my2bot/robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
2bot/lidar_link
Broadcaster: /my2bot/robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
my2bot/imu_link
Broadcaster: /my2bot/robot_state_publisher
Average rate: 37.879
Buffer length: 0.132
Most recent transform: 246.603
Oldest transform: 246.471
my2bot/wheel_left_link
Broadcaster: /my2bot/robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
my2bot/srf05_right_link
Broadcaster: /my2bot/robot_state_publisher
Average rate: 10000.0
Buffer length: 0.0
Most recent transform: 0.0
Oldest transform: 0.0
my2bot/srf05_left_link
Broadcaster: /my2bot/robot_state_publisher
Average rate: 37.879
Buffer length: 0.132
Most recent transform: 246.603
Oldest transform: 246.471
my2bot/wheel_right_link

Figure V.15 *Coordinate Frames Tree – two UGVs mapping together (Source: Screenshot)*

Packages:

<u>Simulation: gazebo_ros</u>

Gazebo 3D simulator wrapped as a ROS package to permit the communication and control facilities

We use the namespace concept to operate two simulated UGVs in the same Gazebo world.

Implemented physics engine: ODE

Alternative physics engine: Bullet,

<u>Teleoperation: ugv_teleop</u>

Joystick configuration over Ubuntu is managed by the joy ROS package. For teleoperation, an algorithm (ugv_teleop_joy.cpp) is created to personalize the linear and angular velocities. Then, for control functionalities, the ROS teleop_joy package is called. To simultaneously control both robots, a launch file teleop_onejoy_twobots was created.

Called package: joy, ros_teleop_joy

Alternative packages: teleop_twist_joy, turtlebot_teleop,

<u>Slam in Simulation: ugv_gazebo (special launch script)</u>

Permits to characterize Gazebo for each mission related to UNISA_UGV

In this case, there are customized lunch files to:

• have two instances of gmapping with appropriate parameters for two slightly different UGVs
• work in two different namespaces
Called package: slam_gmapping, spawn_urdf

Slam: slam_gmapping
Provides laser-based SLAM (Simultaneous Localization and Mapping), calls ROS nodes for gmapping method usage and to save the created map.
Called package: map_server, gmapping
Alternative packages: Cartographer, Hector, Karto, Frontier_exploration, RTAB-Map

Collaboration: ugv_collab
This package provides a Gazebo simulated environment with the Watkins Lab world and spawns two bots in two different namespaces
Call Multi-map merge script to combine individual robot maps made by each ugv spawned, into a single, consistent, global map
Called package: multi_robot_map_merge, tf
Alternative packages: Cartographer, Hector, Karto, Frontier_exploration, RTAB-Map

Map Merge: multi_robot_map_merge
Permits to characterize the Map Merge to manage the number of UNISA_UGV mapping
Called package: ros-kinetic-multirobot-map-merge

Status_publisher: rosbot_state_publisher
This ROS package allows publishing the state of any robot in the environment to tf packages. It uses kinematics tree model of the robot in order to convert the following inputs to outputs (see Figure V.6)
Inputs: joint angles of the robot
Outputs: 3D poses of the robot links

Inputs variables or configuration files :

Environment variable :
export MYUGV_MODEL=my2bot (for all sessions)
export MYUGV_MODEL=mybot05 (only when launching multi_ugv_slam for this ugv type)
Model: model variable chose based on the MYUGV_MODEL environment variable
ugv_description/urdf/ugv_$(arg model).urdf.xacro

SLAM:

Configuration file: ugv_slam/launch/ugv_$(arg slam_methods).launch
Slam methods variable: name="slam_methods" default="gmapping"
Map and trajectory builder: name="configuration_basename" default="ugv_lds_2d_gazebo.lua"

Execution commands:

```
$ roslaunch ugv_collab watkins_multi.launch
$ roslaunch ugv_gazebo multi_ugv_slam.launch ns:=my2bot
$ roslaunch ugv_gazebo multi_ugv_slam.launch ns:=mybot05 (with
MYUGV_MODEL=mybot05)
$ roslaunch ugv_collab multi_map_merge.launch
$ roslaunch ugv_teleop teleop_onejoy_twobots.launch
$ rosrun rviz rviz -d `rospack find ugv_collab`/rviz/multi_map.rviz
```

Once the area to map was covered appropriately, create a common map by running:

```
$ roscd ugv_slam/maps_created
$ rosrun map_server map_saver -f ./multi_map_watkins_$date
```

Run map_server twice for individual maps, changing the MYUGV_MODEL with the ugv names (my2bot and mybot05) and running:

```
$ export MYUGV_MODEL=my2bot (then mybot05)
$ rosrun map_server map_saver -f ./$MYUGV_MODEL_map_watkins_$date map:=/$MYUGV_MODEL/map
```

ROS nodes and topics:

Figure V.15 shows the nodes and topics of two UGVs building a map of a current environment together; the nodes are represented by ellipses while the topics by rectangles. It is provided by the rosgraph tool.

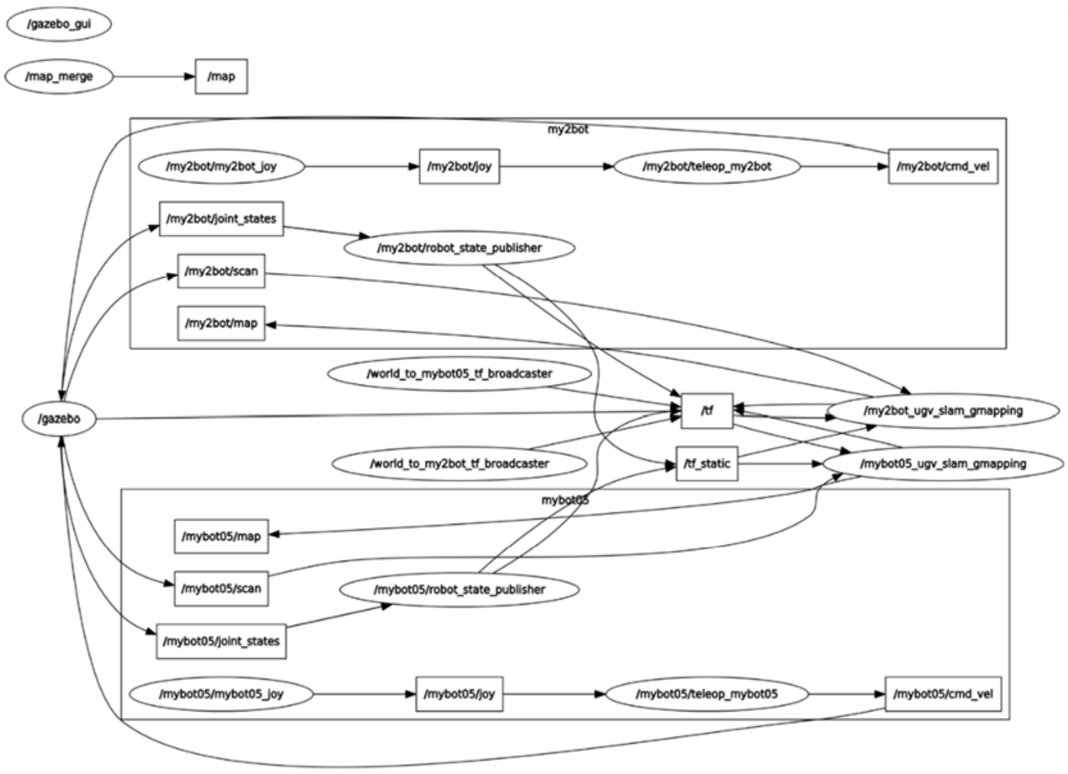


Figure V.16 *rosgraph nodes and topics*
(Source: Screenshot)

Execution Images:

- **Gazebo-ROS with two UGVs**

The Gazebo-ROS environment starts with two UGVs, my2bot and mybot05 (see Figure V.16) in a virtual environment. By the execution of roslaunch call the configuration script watkins_multi.launch placed in the launch folder of package ugv_collab
The two UGVs have a fake lidar configuration for mapping purpose; they are launched at different initial positions in order to facilitate the navigation and map creation performance.
The environment is a simple one, just for test purposes. The objective in this use case is the collaboration while performing a single task.

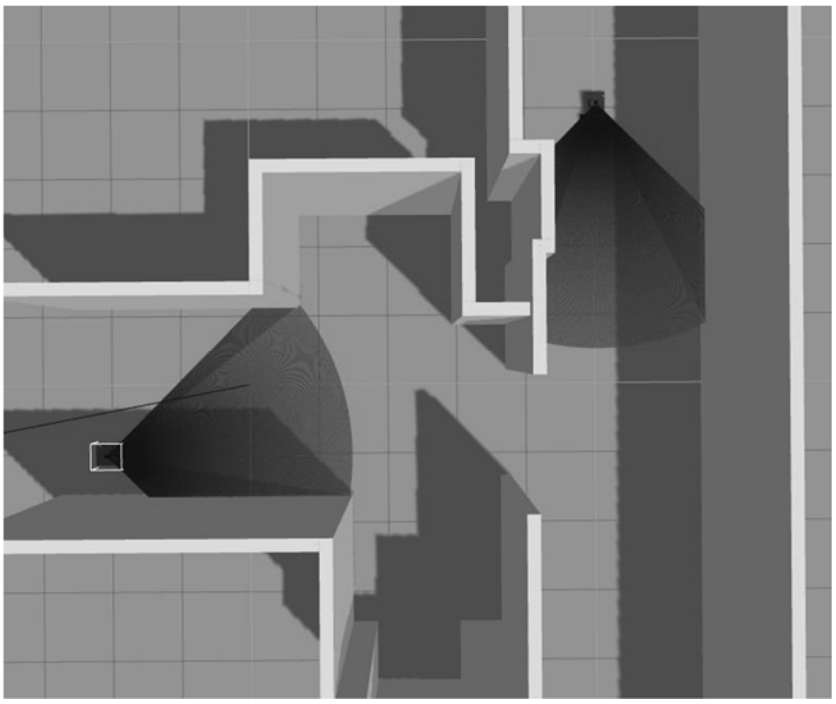


Figure V.17 *Two UGVs in Gazebo for Collaborative Work (Source: Screenshot)*

- **Launching RVIZ to visualize all running nodes of these two UGVs mapping mission**
  The ROS tool RVIZ permit visualization of the Gazebo environment, meaning all it is launched and displayed in the simulator and all other nodes that we launched so far in ROS. This tool is also highly configurable. At some minutes after running, we could see the followed (Figure V.17).

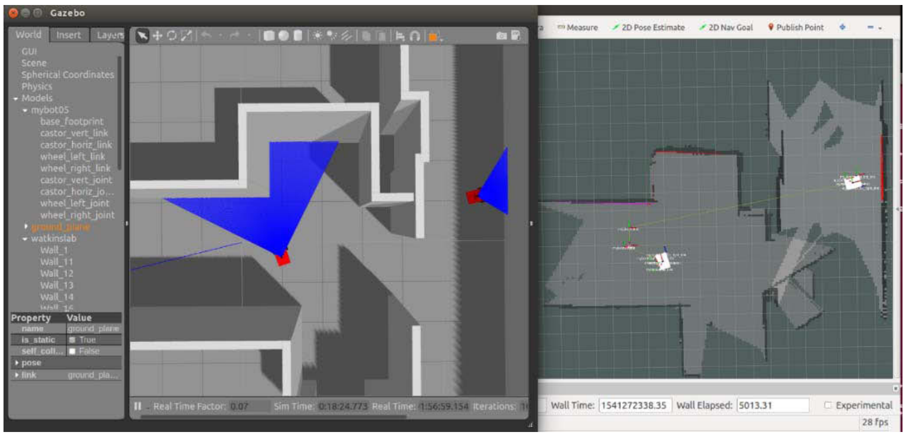


Figure V.18 *Two UGVs in Gazebo and RVIZ Navigating and Mapping together (Source: Screenshot)*

- **Doing SLAM for a moment, Gazebo and RVIZ visualization**
  The mapping progress is visualized in real-time in RVIZ, while at the same time, we see in Gazebo simulator the two UGVs moving around in concordance with the velocities send to /cmd_vel of each one. In this case, we use the rqt_steering tool to show different options for teleoperation.

- **Generating the maps, one for each UGV and another merged**

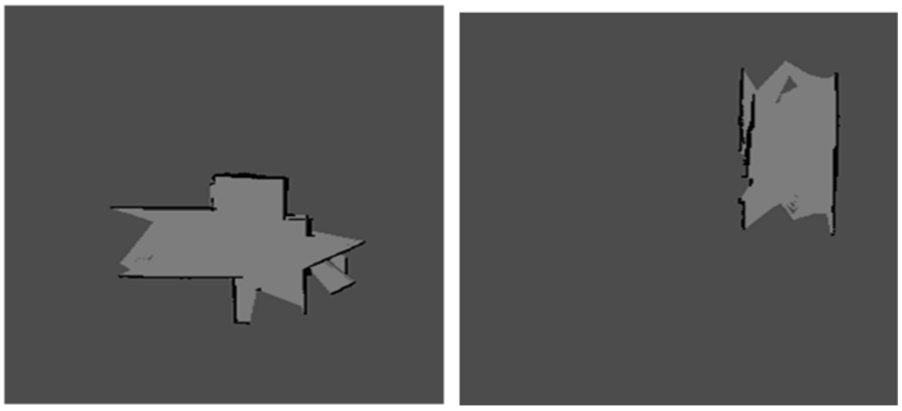


Figure V.19 *Two Maps created by each UGVs Collaboratively (Source: Screenshot)*

The map_server package gives the possibility to get the individual map of each mobile robot participating in the task and to get another merged. The following figures show the singular maps (Figure V.18).

V.3 UNISA-UAV Use Cases

V.3.1 Use Case 1: fixed-wing UAV controlled by a Gazebo plugin

Mission:

The mission is to fly the fixed-wing UAV in order to do all the steps to perform a safety flight, by the control of each mobile component using a plugin, that sends the appropriate commands by teleoperation from a keyboard. For this purpose, Gazebo messages and a C++ program was modified. Graphically we could summaries the general functionality and the steps needed as follow Figure V.19.

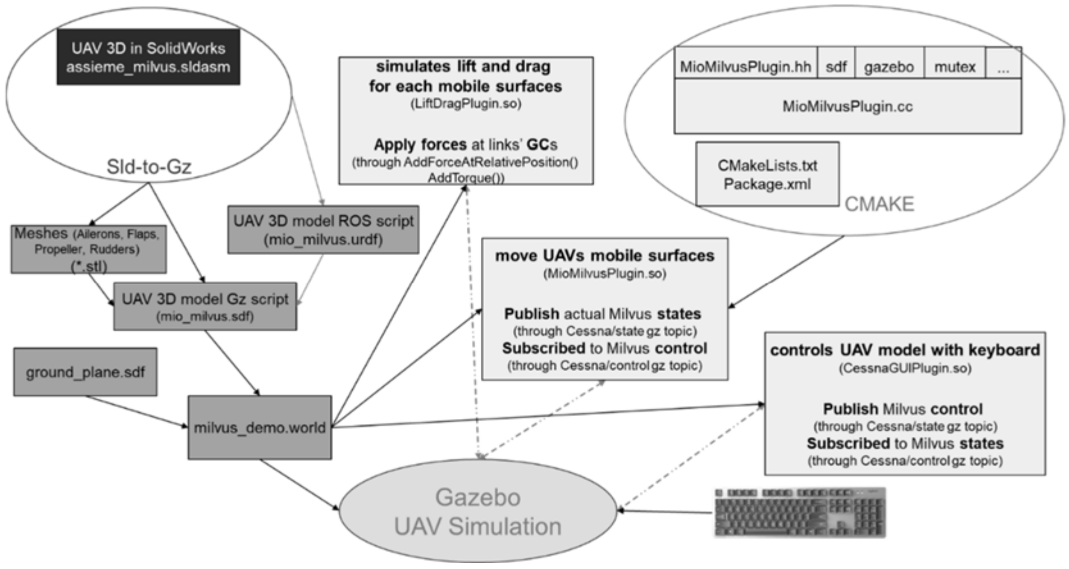


Figure V.20 *UAV Processes to bring and control a Model into Gazebo (Source: Self-elaboration)*

UAV Description:

Figure V.20 describes the mio_milvus fixed-wing UAV components graphically, meaning the links and joints configurations. The mobile components are the left and right ailerons, the left and right flaps, the left and right rudders, the elevator, the wheels, and the propeller.

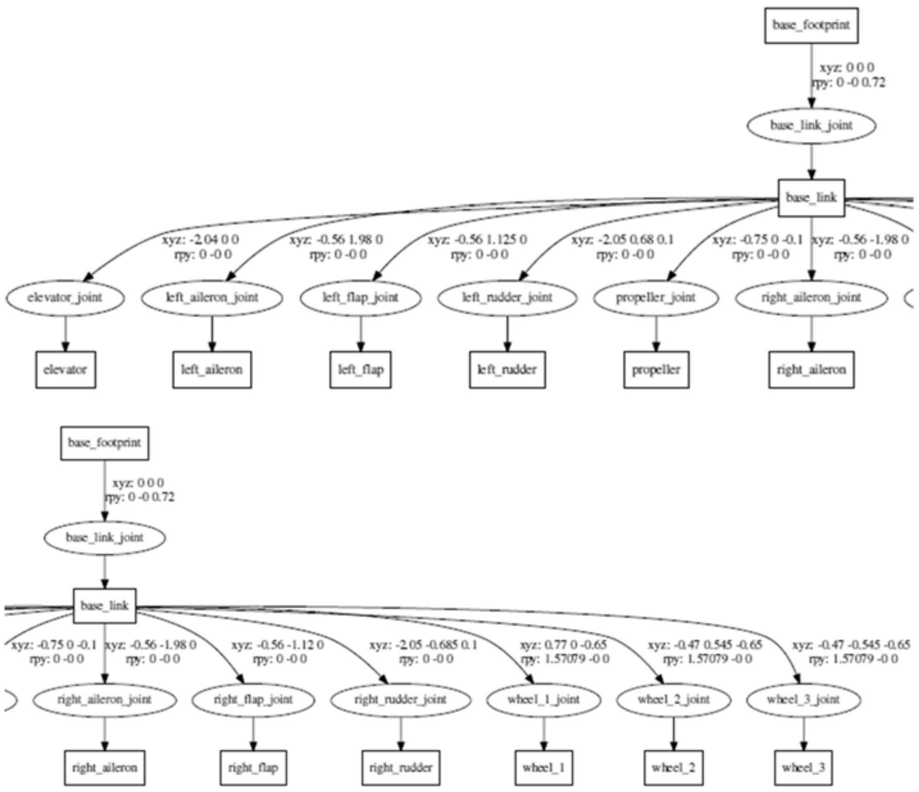


Figure V.21 *UAV Description – links and joints*
(Source: Self-elaboration)

Programs:

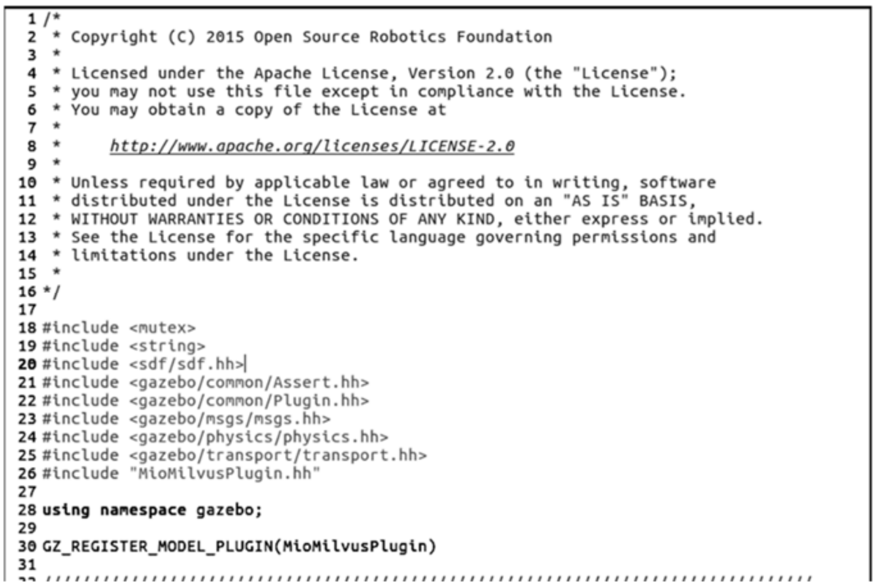
<u>Simulation: gazebo</u>

Gazebo 3D simulator to permit the communication and control facilities in the virtual world of the UAV

Implemented physics engine: ODE
Alternative physics engine: Bullet

<u>Plugin: MioMilvusPlugin</u>

We modify the CessnaPlugin slightly in order to be possible to use with CessnaGUIPlugin in another fixed-wing UAV over Gazebo. The resulting new C++ code call MioMilvusPlugin requires to be compiled, which in turn asks for some code adaptation. We will explain the program elements and the compilation process as an example (Figures V.21 to V.30).



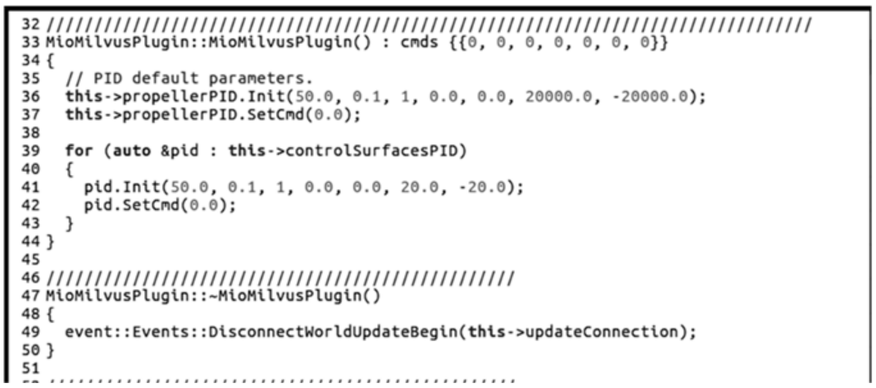
```
 1 /*
 2  * Copyright (C) 2015 Open Source Robotics Foundation
 3  *
 4  * Licensed under the Apache License, Version 2.0 (the "License");
 5  * you may not use this file except in compliance with the License.
 6  * You may obtain a copy of the License at
 7  *
 8  *     http://www.apache.org/licenses/LICENSE-2.0
 9  *
10  * Unless required by applicable law or agreed to in writing, software
11  * distributed under the License is distributed on an "AS IS" BASIS,
12  * WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
13  * See the License for the specific language governing permissions and
14  * limitations under the License.
15  *
16 */
17
18 #include <mutex>
19 #include <string>
20 #include <sdf/sdf.hh>
21 #include <gazebo/common/Assert.hh>
22 #include <gazebo/common/Plugin.hh>
23 #include <gazebo/msgs/msgs.hh>
24 #include <gazebo/physics/physics.hh>
25 #include <gazebo/transport/transport.hh>
26 #include "MioMilvusPlugin.hh"
27
28 using namespace gazebo;
29
30 GZ_REGISTER_MODEL_PLUGIN(MioMilvusPlugin)
31
```

Figure V.22 *MioMilvusPlugin for Gazebo – initial definitions (Source: Screenshot)*

MioMilvusPlugin requires some gazebo, sdf, and other header files that are included at the beginning of the C++ program (lines 18-26), the variable definitions. Using namespace gazebo (line 28) makes all members visible, then we do not need to call gazebo:: all the time. The MioMilvusPlugin is registered as a gazebo model plugin in the simulator (line 30), then this C++ library could be loaded by Gazebo at runtime to has access to Gazebo's API, to perform a wide variety of tasks including moving UAV mobile components (joints) and accessing sensor data (cameras, lidar).

<u>Plugin: MioMilvusPlugin</u> - Class Constructor and Destructor

The lines 33 to 44 creates a plugin constructor and instantiates initial values for cmd (array for control propeller speed and UAV mobile surfaces). It also defines their default PID values. While the lines 47 to 50 create a destructor to disconnect everything for a clear exit.

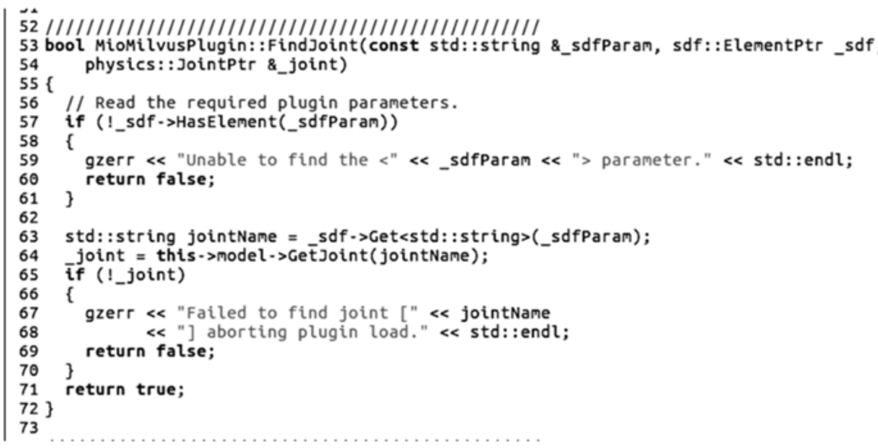


```
////////////////////////////////////////////////////////////////////////////////
MioMilvusPlugin::MioMilvusPlugin() : cmds {{0, 0, 0, 0, 0, 0, 0}}
{
  // PID default parameters.
  this->propellerPID.Init(50.0, 0.1, 1, 0.0, 0.0, 20000.0, -20000.0);
  this->propellerPID.SetCmd(0.0);

  for (auto &pid : this->controlSurfacesPID)
  {
    pid.Init(50.0, 0.1, 1, 0.0, 0.0, 20.0, -20.0);
    pid.SetCmd(0.0);
  }
}

/////////////////////////////////////////////////
MioMilvusPlugin::~MioMilvusPlugin()
{
  event::Events::DisconnectWorldUpdateBegin(this->updateConnection);
}

```

Figure V.23 *MioMilvusPlugin for Gazebo – Class Constructor and Destructor (Source: Screenshot)*

Plugin: MioMilvusPlugin - FindJoint method

Lines 53 to 72 create the FindJoint method, which requires as input the reference memory of sdf parameters and joints, provided at call time. The method looks into the .world file to obtain the parameter name and assigned value for each call, with those strings creates the working variables jointName and _joint. It could throw error messages and abort the loading process if one of the required parameters is not found.

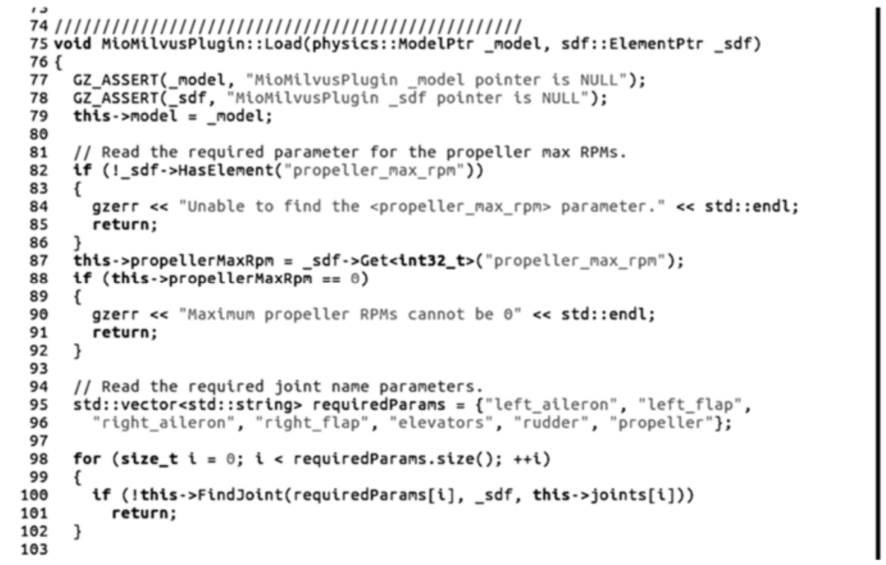


```
52 /////////////////////////////////////////////////
53 bool MioMilvusPlugin::FindJoint(const std::string &_sdfParam, sdf::ElementPtr _sdf,
54     physics::JointPtr &_joint)
55 {
56   // Read the required plugin parameters.
57   if (!_sdf->HasElement(_sdfParam))
58   {
59     gzerr << "Unable to find the <" << _sdfParam << "> parameter." << std::endl;
60     return false;
61   }
62
63   std::string jointName = _sdf->Get<std::string>(_sdfParam);
64   _joint = this->model->GetJoint(jointName);
65   if (!_joint)
66   {
67     gzerr << "Failed to find joint [" << jointName
68           << "] aborting plugin load." << std::endl;
69     return false;
70   }
71   return true;
72 }
73
```

Figure V.24 *MioMilvusPlugin for Gazebo – FindJoint method (Source: Screenshot)*

Plugin: MioMilvusPlugin - Load method

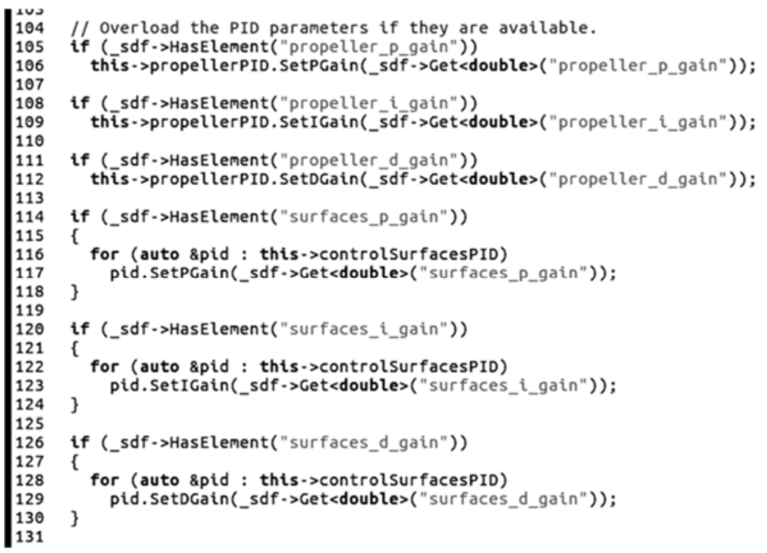
The lines 75 to 150 define the Load method, requires as input the _model and _sdf pointers. It asks for the values of the propeller_max_rpm and mobile surface variables that must be passed as parameters; otherwise, abort the load process if those are not found, then Call the FindJoint method to obtain the values of each of them.



```
74 ////////////////////////////////////////////////
75 void MioMilvusPlugin::Load(physics::ModelPtr _model, sdf::ElementPtr _sdf)
76 {
77   GZ_ASSERT(_model, "MioMilvusPlugin _model pointer is NULL");
78   GZ_ASSERT(_sdf, "MioMilvusPlugin _sdf pointer is NULL");
79   this->model = _model;
80 
81   // Read the required parameter for the propeller max RPMs.
82   if (!_sdf->HasElement("propeller_max_rpm"))
83   {
84     gzerr << "Unable to find the <propeller_max_rpm> parameter." << std::endl;
85     return;
86   }
87   this->propellerMaxRpm = _sdf->Get<int32_t>("propeller_max_rpm");
88   if (this->propellerMaxRpm == 0)
89   {
90     gzerr << "Maximum propeller RPMs cannot be 0" << std::endl;
91     return;
92   }
93 
94   // Read the required joint name parameters.
95   std::vector<std::string> requiredParams = {"left_aileron", "left_flap",
96     "right_aileron", "right_flap", "elevators", "rudder", "propeller"};
97 
98   for (size_t i = 0; i < requiredParams.size(); ++i)
99   {
100    if (!this->FindJoint(requiredParams[i], _sdf, this->joints[i]))
101      return;
102  }
103 
```

Figure V.25 *MioMilvusPlugin for Gazebo – Load method 1of3*
(Source: Screenshot)

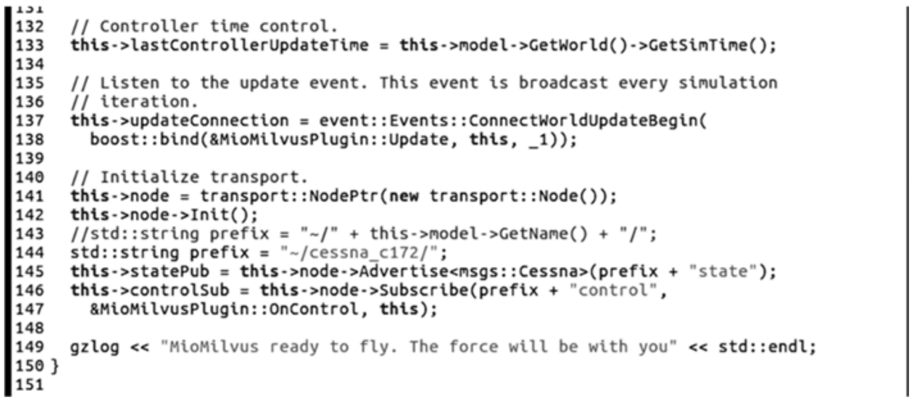
The lines 105 to 130 overload the initial PID values if those are provided in the world file. The PID values are obtained for the propeller and mobile surfaces, for the last one uses the same values for all of them.



```
  // Overload the PID parameters if they are available.
  if (_sdf->HasElement("propeller_p_gain"))
    this->propellerPID.SetPGain(_sdf->Get<double>("propeller_p_gain"));

  if (_sdf->HasElement("propeller_i_gain"))
    this->propellerPID.SetIGain(_sdf->Get<double>("propeller_i_gain"));

  if (_sdf->HasElement("propeller_d_gain"))
    this->propellerPID.SetDGain(_sdf->Get<double>("propeller_d_gain"));

  if (_sdf->HasElement("surfaces_p_gain"))
  {
    for (auto &pid : this->controlSurfacesPID)
      pid.SetPGain(_sdf->Get<double>("surfaces_p_gain"));
  }

  if (_sdf->HasElement("surfaces_i_gain"))
  {
    for (auto &pid : this->controlSurfacesPID)
      pid.SetIGain(_sdf->Get<double>("surfaces_i_gain"));
  }

  if (_sdf->HasElement("surfaces_d_gain"))
  {
    for (auto &pid : this->controlSurfacesPID)
      pid.SetDGain(_sdf->Get<double>("surfaces_d_gain"));
  }

```

Figure V.26 *MioMilvusPlugin for Gazebo – Load method 2of3*
(Source: Screenshot)

The last lines of the load method control the simulation time to deal with the update events. Initialize the gazebo node, which will be responsible for the communication transport, then creates the publisher and subscriber to state and control topics, both relay over the Cessna message type, which is part of standard Gazebo messages.

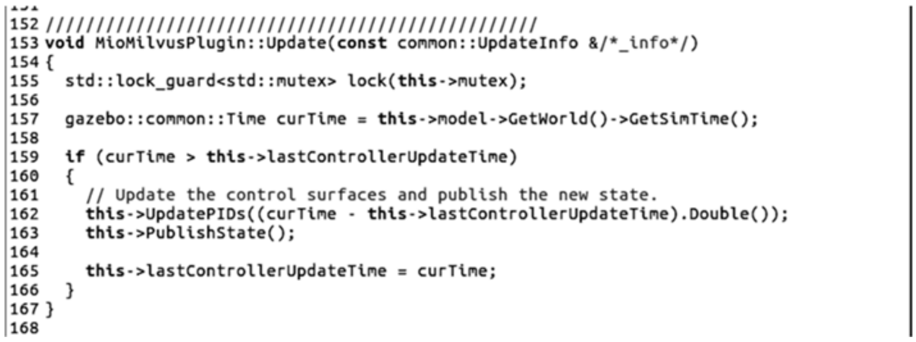


```
  // Controller time control.
  this->lastControllerUpdateTime = this->model->GetWorld()->GetSimTime();

  // Listen to the update event. This event is broadcast every simulation
  // iteration.
  this->updateConnection = event::Events::ConnectWorldUpdateBegin(
    boost::bind(&MioMilvusPlugin::Update, this, _1));

  // Initialize transport.
  this->node = transport::NodePtr(new transport::Node());
  this->node->Init();
  //std::string prefix = "~/" + this->model->GetName() + "/";
  std::string prefix = "~/cessna_c172/";
  this->statePub = this->node->Advertise<msgs::Cessna>(prefix + "state");
  this->controlSub = this->node->Subscribe(prefix + "control",
    &MioMilvusPlugin::OnControl, this);

  gzlog << "MioMilvus ready to fly. The force will be with you" << std::endl;
}

```

Figure V.27 *MioMilvusPlugin for Gazebo – Load method 3of3*
(Source: Screenshot)

Plugin: MioMilvusPlugin. Update method

The lines 153 to 167 defines the Update method, uses mutex lock_guard to be sure that the message to be published remains unchanged during the process. It makes calls to UpdatePIDs and PublishState methods.

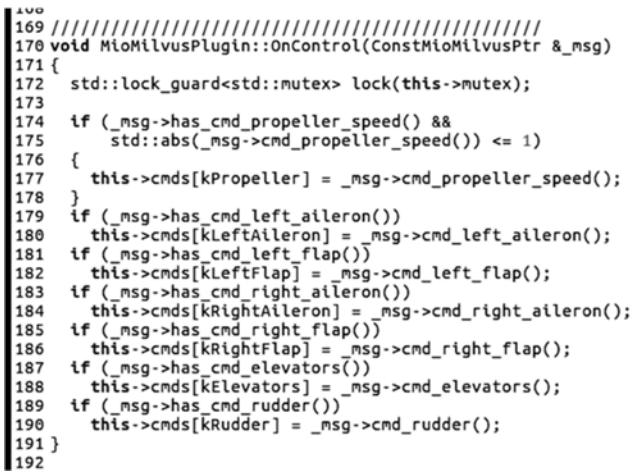


```
152 /////////////////////////////////////////////////
153 void MioMilvusPlugin::Update(const common::UpdateInfo &/*_info*/)
154 {
155   std::lock_guard<std::mutex> lock(this->mutex);
156
157   gazebo::common::Time curTime = this->model->GetWorld()->GetSimTime();
158
159   if (curTime > this->lastControllerUpdateTime)
160   {
161     // Update the control surfaces and publish the new state.
162     this->UpdatePIDs((curTime - this->lastControllerUpdateTime).Double());
163     this->PublishState();
164
165     this->lastControllerUpdateTime = curTime;
166   }
167 }
168
```

Figure V.28 *MioMilvusPlugin for Gazebo – the Update method (Source: Screenshot)*

Plugin: MioMilvusPlugin - Control method

It is a callback method (lines 170 to 191), it is activated when the new values for propeller and mobile surfaces comes from the keyboard through the CessnaGUIPlugin employing control topic to which the UAV in gazebo simulation environment was previously subscribed. The new values are stored in a Cessna message array, to be read at call time.



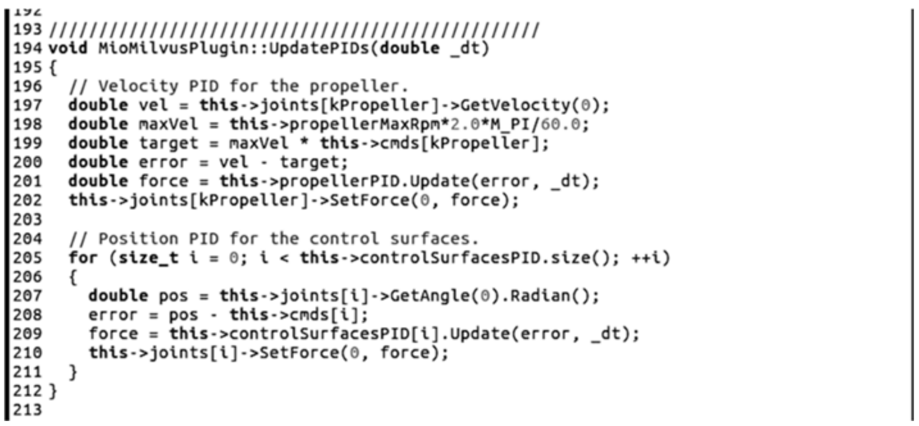
```
169 /////////////////////////////////////////////////
170 void MioMilvusPlugin::OnControl(ConstMioMilvusPtr &_msg)
171 {
172   std::lock_guard<std::mutex> lock(this->mutex);
173
174   if (_msg->has_cmd_propeller_speed() &&
175       std::abs(_msg->cmd_propeller_speed()) <= 1)
176   {
177     this->cmds[kPropeller] = _msg->cmd_propeller_speed();
178   }
179   if (_msg->has_cmd_left_aileron())
180     this->cmds[kLeftAileron] = _msg->cmd_left_aileron();
181   if (_msg->has_cmd_left_flap())
182     this->cmds[kLeftFlap] = _msg->cmd_left_flap();
183   if (_msg->has_cmd_right_aileron())
184     this->cmds[kRightAileron] = _msg->cmd_right_aileron();
185   if (_msg->has_cmd_right_flap())
186     this->cmds[kRightFlap] = _msg->cmd_right_flap();
187   if (_msg->has_cmd_elevators())
188     this->cmds[kElevators] = _msg->cmd_elevators();
189   if (_msg->has_cmd_rudder())
190     this->cmds[kRudder] = _msg->cmd_rudder();
191 }
192
```

Figure V.29 *MioMilvusPlugin for Gazebo – Control method (Source: Screenshot)*

Plugin: MioMilvusPlugin - UpdatePIDs method

This method (lines 194 to 212) uses the PID, proportional-integral-derivative controller, which is the most common type of controller used for UAV stabilization and autonomous control. Here it is used to drive propeller and mobile surfaces of the UAV flight in the simulation environment.

The specific update of propellerPID and controlSurfacesPID are called passing the calculated errors.

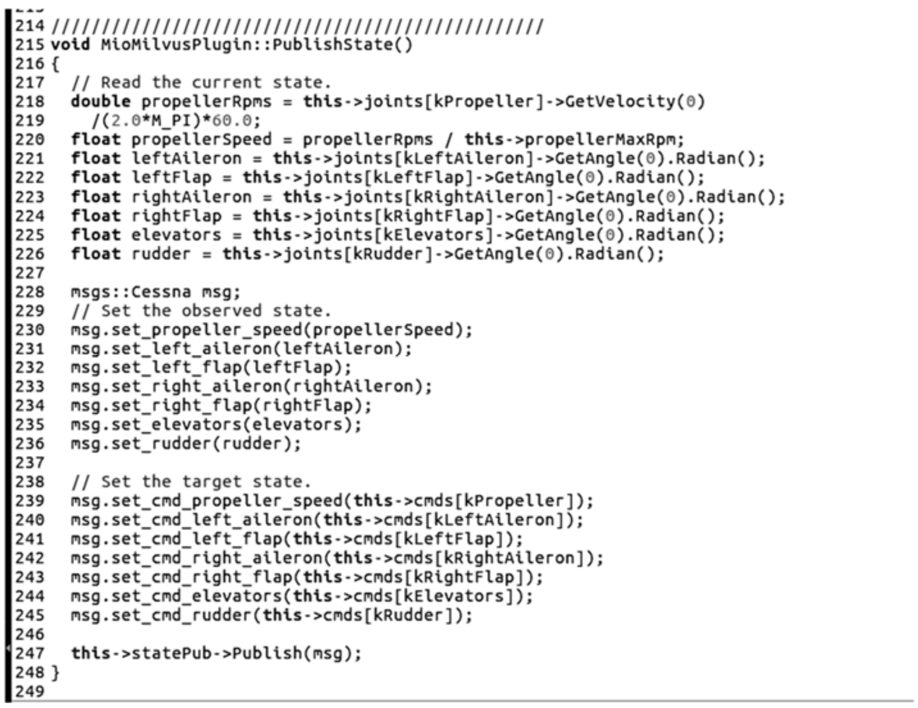


```
192
193 /////////////////////////////////////////////////
194 void MioMilvusPlugin::UpdatePIDs(double _dt)
195 {
196   // Velocity PID for the propeller.
197   double vel = this->joints[kPropeller]->GetVelocity(0);
198   double maxVel = this->propellerMaxRpm*2.0*M_PI/60.0;
199   double target = maxVel * this->cmds[kPropeller];
200   double error = vel - target;
201   double force = this->propellerPID.Update(error, _dt);
202   this->joints[kPropeller]->SetForce(0, force);
203
204   // Position PID for the control surfaces.
205   for (size_t i = 0; i < this->controlSurfacesPID.size(); ++i)
206   {
207     double pos = this->joints[i]->GetAngle(0).Radian();
208     error = pos - this->cmds[i];
209     force = this->controlSurfacesPID[i].Update(error, _dt);
210     this->joints[i]->SetForce(0, force);
211   }
212 }
213
```

Figure V.30 *MioMilvusPlugin for Gazebo – UpdatePIDs method (Source: Screenshot)*

Plugin: MioMilvusPlugin - PublishState method

This method (lines 215 to 248) prepares the Cessna msg values for the observed state (position and orientation) of the propeller and each mobile surface, as well as for the target state values from keyboard interactions. This method is called during the Update process, which publishes the message over the state topic.



```
////////////////////////////////////////////////
void MioMilvusPlugin::PublishState()
{
  // Read the current state.
  double propellerRpms = this->joints[kPropeller]->GetVelocity(0)
    /(2.0*M_PI)*60.0;
  float propellerSpeed = propellerRpms / this->propellerMaxRpm;
  float leftAileron = this->joints[kLeftAileron]->GetAngle(0).Radian();
  float leftFlap = this->joints[kLeftFlap]->GetAngle(0).Radian();
  float rightAileron = this->joints[kRightAileron]->GetAngle(0).Radian();
  float rightFlap = this->joints[kRightFlap]->GetAngle(0).Radian();
  float elevators = this->joints[kElevators]->GetAngle(0).Radian();
  float rudder = this->joints[kRudder]->GetAngle(0).Radian();

  msgs::Cessna msg;
  // Set the observed state.
  msg.set_propeller_speed(propellerSpeed);
  msg.set_left_aileron(leftAileron);
  msg.set_left_flap(leftFlap);
  msg.set_right_aileron(rightAileron);
  msg.set_right_flap(rightFlap);
  msg.set_elevators(elevators);
  msg.set_rudder(rudder);

  // Set the target state.
  msg.set_cmd_propeller_speed(this->cmds[kPropeller]);
  msg.set_cmd_left_aileron(this->cmds[kLeftAileron]);
  msg.set_cmd_left_flap(this->cmds[kLeftFlap]);
  msg.set_cmd_right_aileron(this->cmds[kRightAileron]);
  msg.set_cmd_right_flap(this->cmds[kRightFlap]);
  msg.set_cmd_elevators(this->cmds[kElevators]);
  msg.set_cmd_rudder(this->cmds[kRudder]);

  this->statePub->Publish(msg);
}
```

Figure V.31 *MioMilvusPlugin for Gazebo – PublishState method (Source: Screenshot)*

Results:

MioMilvus control (Keyboard Teleoperation)

The teleoperation of an aircraft by a keyboard pressing is not an easy job; however, it simulates a real flight controlling in the sense that the pilots use the yoke, gears, pedals to "drive" a real aircraft, in the same way, changing the values by endless variations. However, depending on the computational resources, it could be very sensible or too slow to accumulate the commands send to the simulated UAV.

As we could see in the execution images, it was possible to fly a 3D model of the fixed-wing aircraft call MioMilvus using a plugin that runs only in Gazebo. Doing in this way, the flight does not need a predefined path; it is up to the pilot to choose the best way to perform the navigation.

Execution Images:

Figures V.31 and V.34 shows the different stage of the fixed-wing UAV flight controlled by the increments and decrements of the angular positions of the mobile components, send to Gazebo simulator from the keyboard pressing.

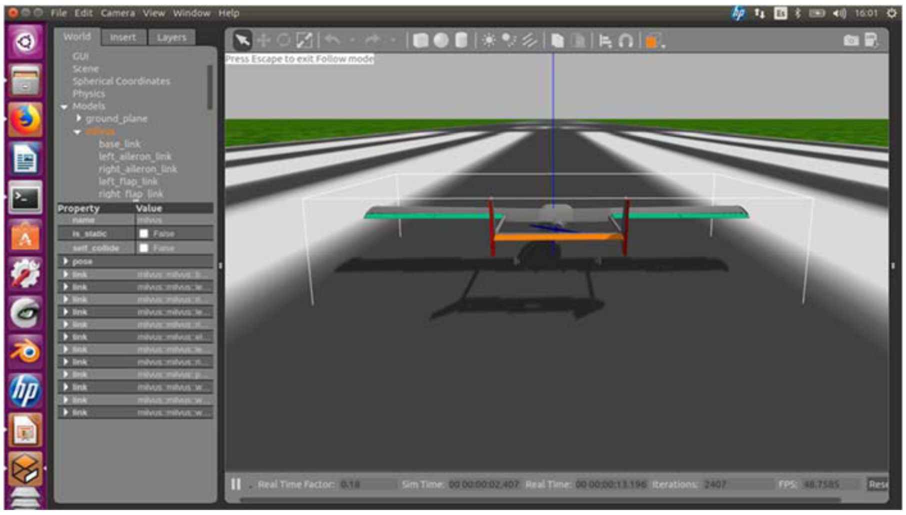


Figure V.32 *MioMilvus in Gazebo – at launch (start)*
(Source: Screenshot)

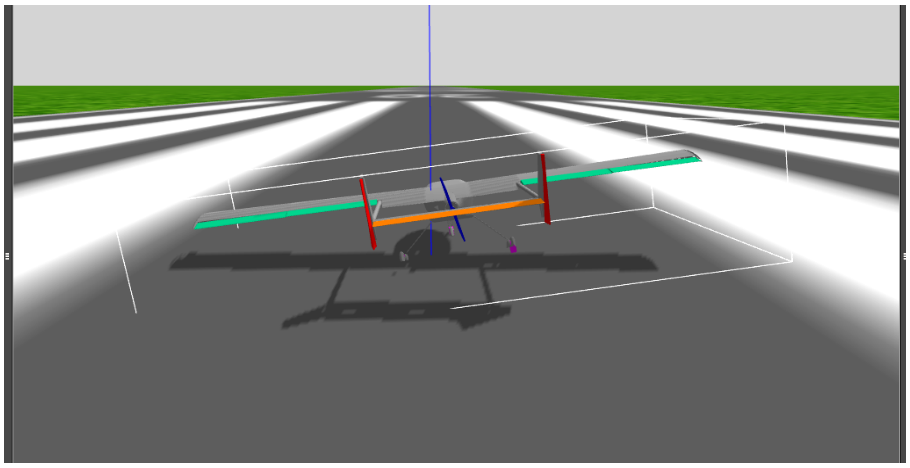


Figure V.33 *MioMilvus in Gazebo – taking-off*
(Source: Screenshot)

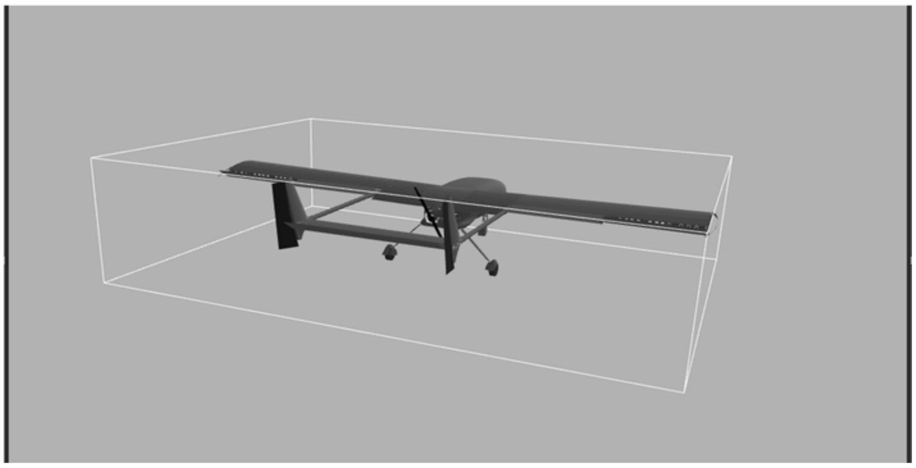


Figure V.34 *MioMilvus in Gazebo – flying (Source: Screenshot)*

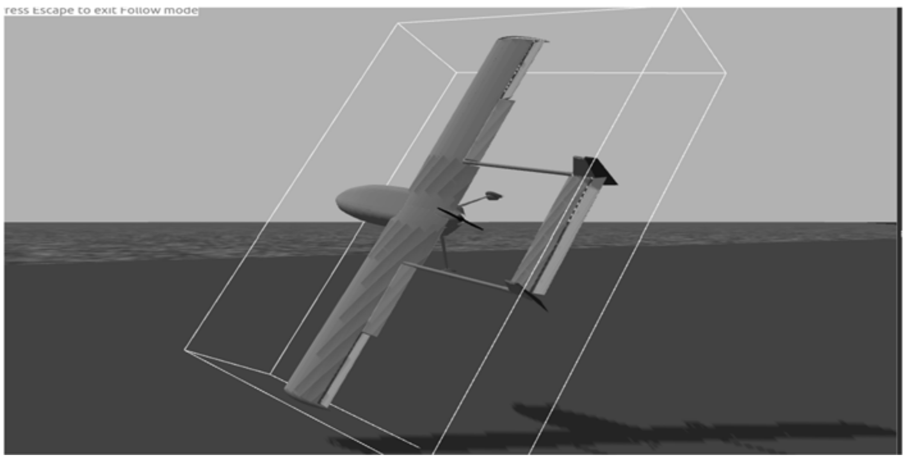


Figure V.35 *MioMilvus in Gazebo – turning for landing (Source: Screenshot)*

V.3.2 Use Case 2: fixed-wing UAV in Dubins waypoint navigation

Mission:

The mission is to fly the fixed-wing UAV in order to perform a path following between waypoints using Dubins paths; that is, to be on the path with no time dependency following an optimal path-planning, which for constant-altitude, constant-velocity vehicles with turning constraints, are also time-optimal paths between two configurations (nodes) (Beard & Mc.Lain, 2012).

A summary of the ROS packages used, based on the Small Unmanned Aircraft Theory and Practice book, are rosflight and rosplane. The physical parameters need for our aircraft configuration, like mass, geometry, propulsion, and aerodynamic parameters.

Also, we presented the variables and files to configure the simulated environment, followed by the detailed script and configuration files, and finally, the execution of the commands console displays. Then a summary of the position, velocity, and acceleration over time and the results. The objective is to repeat the experiment with the same or other fixed-wing UAV, in a kind of fully documented guide.

Packages:

<u>Simulation: gazebo_ros</u>

Gazebo 3D simulator wrapped as a ROS package to permit the communication and control facilities

We use the namespace concept to operate two simulated UGVs in the same Gazebo world.

Implemented physics engine: ODE
Alternative physics engine: Bullet

<u>Autopilot: rosflight_controller</u>

The autopilot is a high-level design model for the guidance loops. The initial test could be used to estimate the autopilot constants b∗ and to develop a reduced-order model. The guidance models are derived from the six-degree-of-freedom model, kinematic relations, and force balance equations that follow different kinematic design models, for example (in equations (9.19) through (9.22)):

- kinematic design models in equations $\ddot{\chi} = b\chi. (\chi.c - \chi.) + b\chi (\chi c - \chi)$
  - Inputs: command inputs χc, hc, and Vac,
  - Outputs: autopilot coefficients bVa, bh., bh, $b\chi.$, and $b\chi$
- kinematic design models in equations $\psi. = (g/Va) \tan \varphi$
  - Inputs: command inputs θc, hc, and Vac,
  - Outputs: autopilot coefficients bVa, bh., bh, $b\chi.$, and $b\chi$
- dynamic design model utilizes relationships drawn from free-body diagrams (see Fig.). The control variables are thrust, lift coefficient, and bank angle [Fthrust, CL, φ]T, resulting:
  - h. = Vg sin γ
  - h. = Va sin γ (in the absence of wind)

<u>Estimator: rosflight_estimator</u>

This package estimates the states required for the autopilot using the onboard sensors and increases the precision using the following filtering methods:

- low-pass filtering the rate gyros, for the angular rates in the body frame p, q, and r
- low-pass filtering the absolute and differential pressure sensors and inverting the sensor model, for the altitude h and airspeed Va
- extended Kalman filters, for the remaining states.
    - two-state EKF that can be used to estimate the roll and pitch angles (φ, θ).
    - seven-state EKF based on GPS measurements can be used for position, ground speed, course, wind, and heading to estimate the pn, pe, h, Va, φ, θ, ψ, p, q, and r states.

<ins>Pathfollower: rosflight_path_follower</ins>

This package is related to the guidance laws for tracking straight-line segments, constant-altitude circular orbits, or a combination of those in more complex paths, developing strategies to deal with two challenging disturbances in small UAVs as wind and turn radius. For the former, the primary tracking issue is wind speeds, which are commonly 20 to 60 percent of the desired airspeed; and for the latter is the fundamental limit on the spatial frequency of paths that can be tracked.

As Nelson, D. R. et al. (2007) suggests, the objective is "to be on the path" rather than at a certain point at a time, meaning directing the UAV onto the path. The following figures show the variables used for developing the modeled strategies in the algorithms (see Fig. V.38). These authors also say that "for straight-line paths," the approach approximates PD control; "for curved paths," an additional anticipatory control element to improves the tracking capability needs to be implemented. Moreover, finally, the approach that accommodates the addition of an adaptive element to account for disturbances such as wind is helpful (p.185).

- constant winds: wn, we

<ins>Pathmanager: rosflight_path_manager</ins>

Use guidance strategies can be used to follow a series of waypoints, developing strategies to switch from one waypoint to another by the definition of planes and corrected fillets to transit with confidence.

- series of waypoints

For Dubins paths, the objective is to transition from one configuration (position and course) to another, uses turn-straight-turn paths class, managing constant-altitude, constant-groundspeed scenarios.

- series of waypoints
- start configuration denoted as (p s, χ s)

- end configuration denoted as (p e, χ e),

The following figures show the variables used for developing the path_manager algorithms (see Fig.)

Pathplanner: rosflight_path_planner

This package uses the deliberative path planning approach, where the trajectories are planned explicitly; it is strongly dependent upon the models used to describe the state of the world and the motion of the vehicle. The package regularly executes an outer feedback loop. Use simple low-order navigation models for the vehicle and constant-wind models for the atmosphere (p.206).

The problems addressed are point-to-point problems, where the objective is to plan a waypoint path from one point to another through an obstacle field; a coverage problem, where the aim is to plan a waypoint path so that the small UAV covers all the areas, uniformly, in a specific region. In both cases, given the constraints of the obstacle field. It is configured for the use of the rapidly-exploring random tree (RRT) algorithm, closely related to the probabilistic roadmap technique, using Dubins paths between nodes.

Status_publisher: rostopic

This ROS package allows dynamic subscriptions and publications of information (see figure II.3). It permits us to interact with ROS communication for getting information about topics and for interaction with them.

Input variables or configuration files:

Global variable:

```
mav_name=milvus
```

Model: model configuration based on xacro files, calling meshes and configuration files

```
~/catkin_ws/src/unisa_uvf/bots_simulation/uav_gazebo/rosmilvus/rosmilvus_sim/xacro/milvus.xacro
```

Milvus Config:

```
    robot name is: milvus
---------- Successfully Parsed XML ---------------
root Link: milvus/base_link has three child(ren)
  child(1):  milvus/chase/camera_base_link
  child(2):  milvus/gimbal/base_link
    child(2.1):  milvus/gimbal/yaw_link
    child(2.1):  milvus/gimbal/roll_link
    child(2.1):  milvus/gimbal/pitch_link
  child(3):  milvus_vero/laser_link
```

The xacro and configuration files for model description and gazebo configuration are grouped in an original xacro file that calls the standard properties for gazebo and configuration depending on the model name.

Milvus Params:
Configuration file for the UAV, onboard sensors and coefficients

- Mass and Inertial variables
- Components variables
- Trim conditions
- Plugin Parameters

Autopilot and configuration files:

Rosfligh Programs:

```
zandra@ROS:~/catkin_ws/src/unisa_uvf/bots_simula-
tion/uav_gazebo/rosmilvus/rosmilvus/src$ ls -l
-rw-rw-r-- 1 zandra zandra  6888 set 15 16:22 controller_base.cpp
-rw-rw-r-- 1 zandra zandra  7351 set 15 16:23 controller_example.cpp
-rw-rw-r-- 1 zandra zandra  6871 set 15 16:24 estimator_base.cpp
-rw-rw-r-- 1 zandra zandra 12482 set 15 16:24 estimator_example.cpp
-rw-rw-r-- 1 zandra zandra  2759 set 15 16:25 path_follower_base.cpp
-rw-rw-r-- 1 zandra zandra  2099 set 15 16:26 path_follower_example.cpp
-rwxrwxr-x 1 zandra zandra  3075 set 15 16:27 path_manager_base.cpp
-rwxrwxr-x 1 zandra zandra 13525 set 15 16:27 path_manager_example.cpp
-rw-rw-r-- 1 zandra zandra  1048 set 20 22:43 path_planner.cpp
-rw-rw-r-- 1 zandra zandra  1048 set 20 13:59 path_planner.cpp_test1
```

Rosflight Header Files:

```
zandra@ROS:~/catkin_ws/src/unisa_uvf/bots_simulation/uav_ga-
zebo/rosmilvus/rosmilvus/include$ ls -l
-rw-rw-r-- 1 zandra zandra 3529 set 15 16:30 controller_base.h
-rw-rw-r-- 1 zandra zandra 1543 set 15 16:30 controller_example.h
-rw-rw-r-- 1 zandra zandra 3062 set 15 16:31 estimator_base.h
-rw-rw-r-- 1 zandra zandra 1374 set 15 16:31 estimator_example.h
-rw-rw-r-- 1 zandra zandra 2223 set 15 23:56 path_follower_base.h
-rw-rw-r-- 1 zandra zandra  387 set 15 16:32 path_follower_example.h
-rwxrwxr-x 1 zandra zandra 2673 set 15 16:33 path_manager_base.h
-rwxrwxr-x 1 zandra zandra 2274 set 15 16:34 path_manager_example.h
```

Controller Configuration:

zandra@ROS:~/catkin_ws/src/unisa_uvf/bots_simulation/uav_gazebo/ro-smilvus/rosmilvus/cfg$ cat Controller.cfg

```
#!/usr/bin/env python
PACKAGE = "rosmilvus"
from dynamic_reconfigure.parameter_generator_catkin import *
gen = ParameterGenerator()
# trim
trim = gen.add_group("Trim")
trim.add("TRIM_E", double_t, 0, "Elevator trim", 0, -1, 1)
trim.add("TRIM_A", double_t, 0, "Aileron trim", 0, -1, 1)
trim.add("TRIM_R", double_t, 0, "Rudder trim", 0, -1, 1)
trim.add("TRIM_T", double_t, 0, "Throttle trim", 0.6, 0, 1)
# course hold
course = gen.add_group("Course")
course.add("COURSE_KP", double_t, 0, "Course proportional gain", 0.7329, 0, 2)
course.add("COURSE_KD", double_t, 0, "Course derivative gain", 0, -1, 0)
course.add("COURSE_KI", double_t, 0, "Course integral gain", 0.0, 0, 0.2)
# roll hold
roll = gen.add_group("Roll")
roll.add("ROLL_KP", double_t, 0, "Roll proportional gain", 1.17, 0, 3)
roll.add("ROLL_KD", double_t, 0, "Roll derivative gain", -0.13, -1, 0)
roll.add("ROLL_KI", double_t, 0, "Roll integral gain", 0, 0, 0.2)
# pitch hold
pitch = gen.add_group("Pitch")
pitch.add("PITCH_KP", double_t, 0, "Pitch proportional gain", 1.0, 0, 3)
pitch.add("PITCH_KD", double_t, 0, "Pitch derivative gain", -0.17, -0.4, 0)
pitch.add("PITCH_KI", double_t, 0, "Pitch integral gain", 0, 0, 0.2)
pitch.add("PITCH_FF", double_t, 0, "Pitch feed forward value", 0, -1, 1)
# airspeed with pitch hold
as_pitch = gen.add_group("Airspeed with Pitch")
as_pitch.add("AS_PITCH_KP", double_t, 0, "Airspeed with pitch proportional gain", -0.0713, 0, 0.2)
as_pitch.add("AS_PITCH_KD", double_t, 0, "Airspeed with pitch derivative gain", -0.0635, -0.2, 0)
as_pitch.add("AS_PITCH_KI", double_t, 0, "Airspeed with pitch integral gain", 0, 0, 0.2)
# airspeed with throttle hold
as_thr = gen.add_group("Airspeed with Throttle")
as_thr.add("AS_THR_KP", double_t, 0, "Airspeed with throttle proportional gain", 3.2, 0, 10)
as_thr.add("AS_THR_KD", double_t, 0, "Airspeed with throttle derivative gain", 0, -5, 0)
as_thr.add("AS_THR_KI", double_t, 0, "Airspeed with throttle integral gain", 1.0, 0, 10)
# altitude hold
alt = gen.add_group("Altitude")
alt.add("ALT_KP", double_t, 0, "Altitude proportional gain", 0.045, 0, 0.1)
alt.add("ALT_KD", double_t, 0, "Altitude derivative gain", 0, -0.05, 0)
alt.add("ALT_KI", double_t, 0, "Altitude integral gain", 0.01, 0, 0.05)
# side-slip hold
sideslip = gen.add_group("Side Slip")
sideslip.add("BETA_KP", double_t, 0, "Side slip proportional gain", -0.1164, 0, 0.3)
sideslip.add("BETA_KD", double_t, 0, "Side slip derivative gain", 0, -0.15, 0)
sideslip.add("BETA_KI", double_t, 0, "Side slip integral gain", -0.0037111, 0, 0.05)
exit(gen.generate(PACKAGE, "rosmilvus", "Controller"))
```

Follower Configuration:

```
zandra@ROS:~/catkin_ws/src/unisa_uvf/bots_simulation/uav_ga-
zebo/rosmilvus/rosmilvus/cfg$ cat Follower.cfg
```

```
#!/usr/bin/env python
PACKAGE = "rosmilvus"
from dynamic_reconfigure.parameter_generator_catkin import *
gen = ParameterGenerator()
# Chi Infinity
gen.add("CHI_INFTY", double_t, 0, "Chi Infinity", 1.0472, 0 , 1.5708)
# K Path
gen.add("K_PATH", double_t, 0, "K Path", 0.025, 0, 1)
# K Orbit
gen.add("K_ORBIT", double_t, 0, "K Orbit", 4.0, 0, 15)
exit(gen.generate(PACKAGE, "rosmilvus", "Follower"))
```

Rosmilvus configuration files:

```
zandra@ROS:~/catkin_ws/src/unisa_uvf/bots_simula-
tion/uav_gazebo/rosmilvus/rosmilvus_sim/xacro$
```

```
-rw-rw-r-- 1 zandra zandra  825 set 11 12:06 aircraft_forces_and_moments.xacro
-rw-rw-r-- 1 zandra zandra  472 set 11 12:06 aircraft_truth.xacro
-rw-rw-r-- 1 zandra zandra 3857 set 17 00:02 fixedwing.xacro
drwxrwxr-x 2 zandra zandra 4096 set 16 09:34 meshes
-rw-rw-r-- 1 zandra zandra 2257 set 16 23:15 milvus_sil.xacro
-rw-rw-r-- 1 zandra zandra 4159 set 18 13:16 milvus.xacro
```

ROS nodes and topics:

Figure V.35 shows the nodes and topics of the Milvus UAV performing a Dubins path; the nodes are represented by ellipses while the topics by rectangles. It is provided by the rosgraph tool.

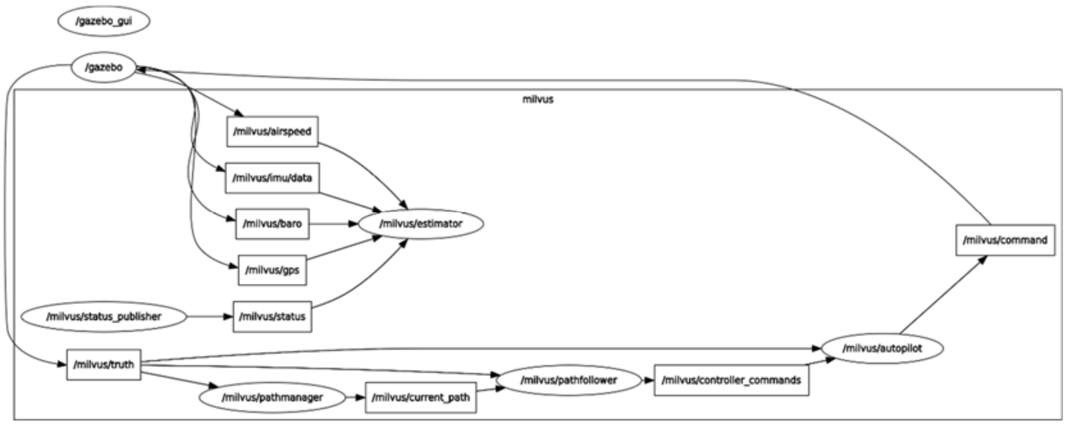


Figure V.36 *Nodes and topics in Gazebo-ROS*
(Source: Screenshot)

Results:

Dubins Path Following (Plots)

The plot shows the (x,y) positions once the desired altitude has been reached, the line in blue shows the ascension of the UAV (see Figure V.36 and V.37), then the plot of Dubins path-following (see Figure V.38).

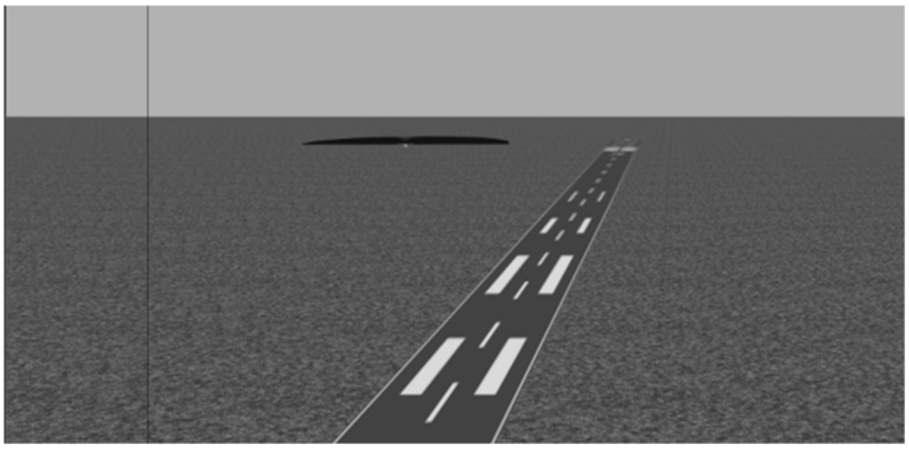


Figure V.37 *Milvus UAV ascension*
(Source: Screenshot)

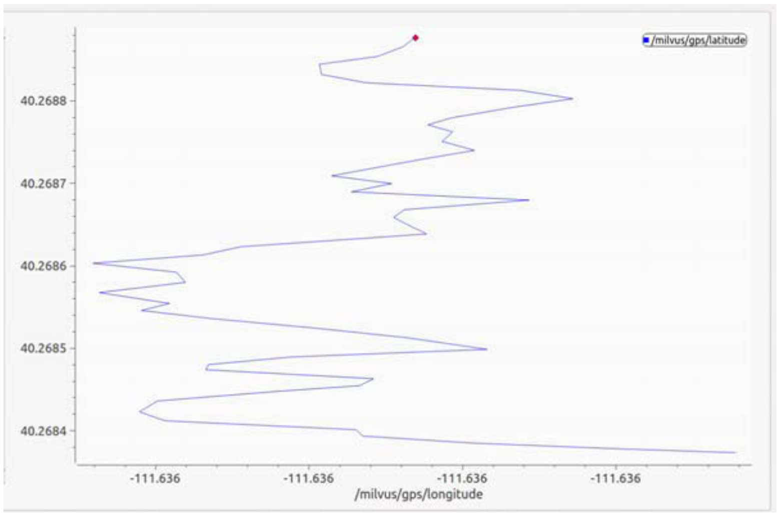


Figure V.38 *Milvus UAV Plot while ascending*
(Source: Screenshot)

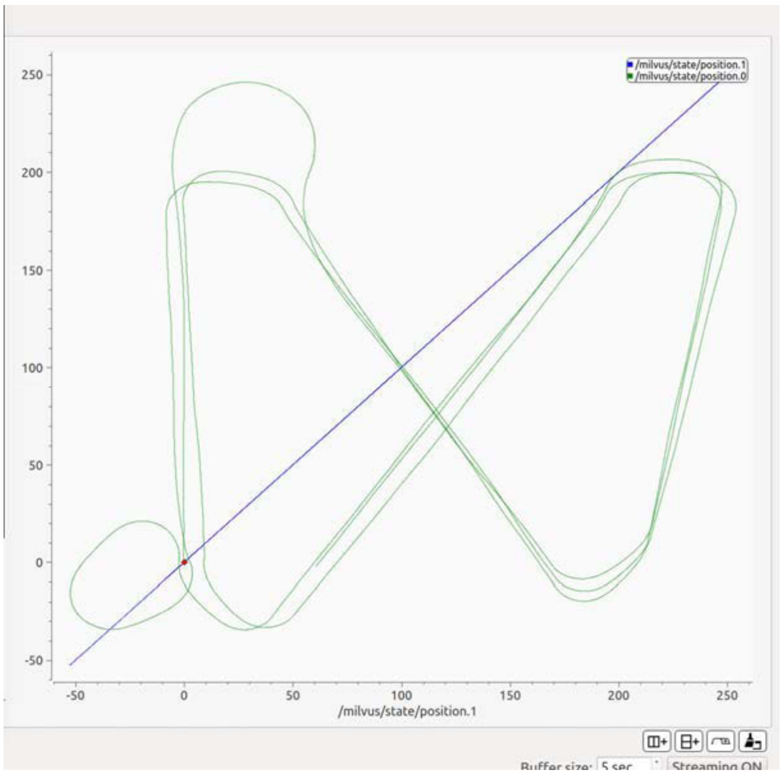


Figure V.39 *Milvus UAV Plot of the Dubins path followed*
(Source: Screenshot)

Execution Images:

Launching Milvus in Gazebo-ROS

The Gazebo-ROS environment starts with Milvus UAVs in a virtual environment, as we could see in Figure V.39. By the execution of **roslaunch,** call the configuration script **milvus.launch** placed in the launch folder of package **rosmilvus_sim**.

The Milvus UAV has a fake sensor configured for sensing, and navigation through autopilot control means that it is entirely autonomous. The environment is usually used for aircraft. The objective in this use case is the autonomous navigation following Dubins path waypoints.

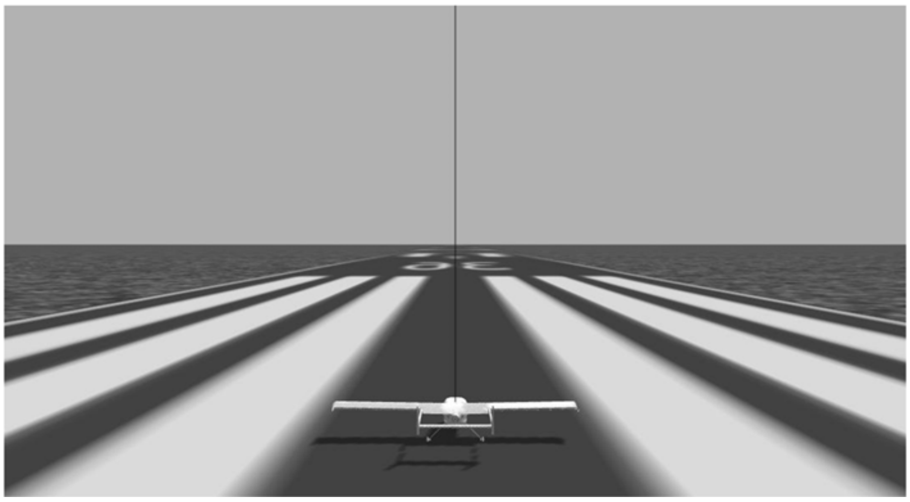


Figure V.40 *Milvus UAV in Gazebo at launch (Source: Screenshot)*

Milvus following a path-planning

The Milvus UAV, based on the configuration at launch time and parameters send, go along all the navigation steps. It starts **armed: 'true,'** and with an established **rosplane_msgs/Current_Path**, then once arrived at the altitude of 50m, start the path-planning programmed to follow a Dubins Path. The Images (Figures V.40 to V.43) show different moments of the flight

```
<node name="status_publisher" pkg="rostopic" type="rostopic"
    output="screen"
   args="pub status rosflight_msgs/Status '{header: {seq: 0, stamp: 0,
    frame_id: 'none'},
   armed: 'true', failsafe: 'false', rc_override: 'false', offboard: 'false',
   control_mode: 0, error_code: 0, num_errors: 0, loop_time_us: 0}'"/>
```

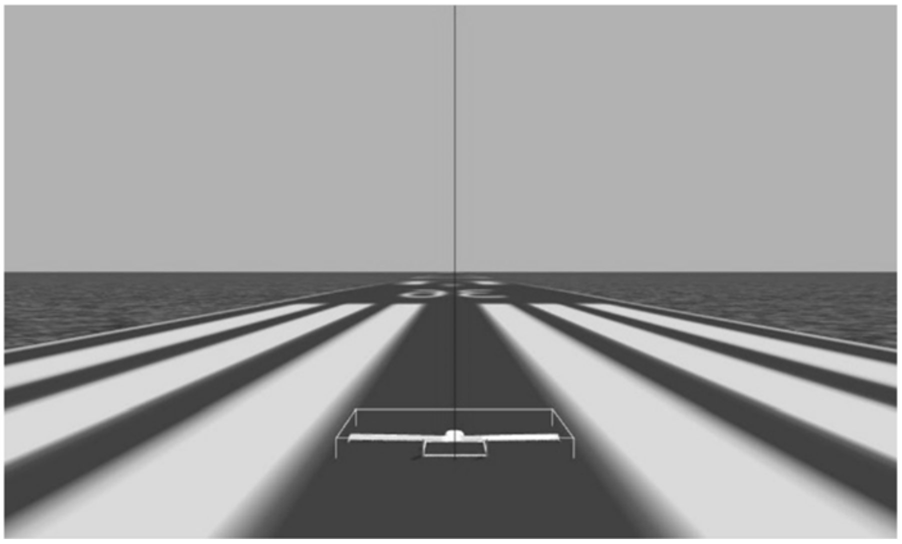


Figure V.41 *Milvus UAV in Gazebo at starting the execution (Source: Screenshot)*

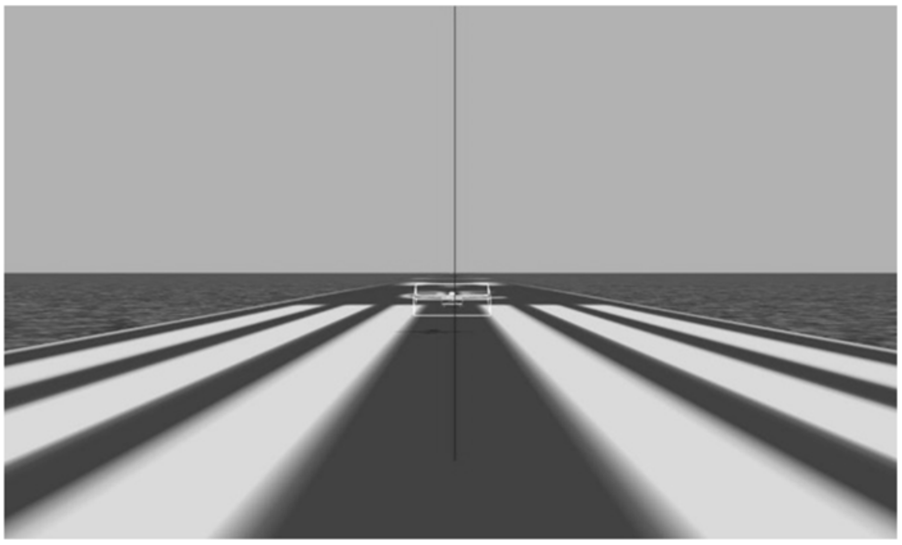


Figure V.42 *Milvus UAV in Gazebo at takeoff (Source: Screenshot)*

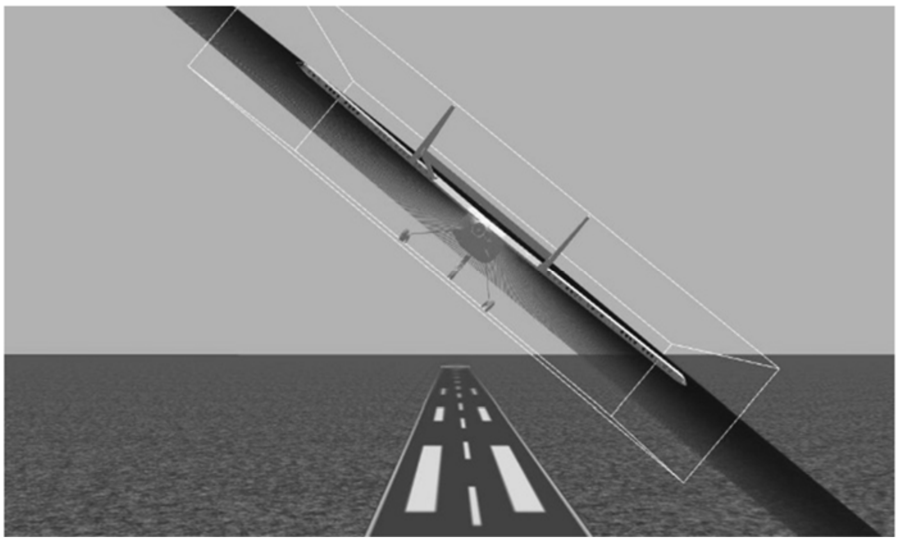


Figure V.43 *Milvus UAV in Gazebo at turning in Dubins path (Source: Screenshot)*

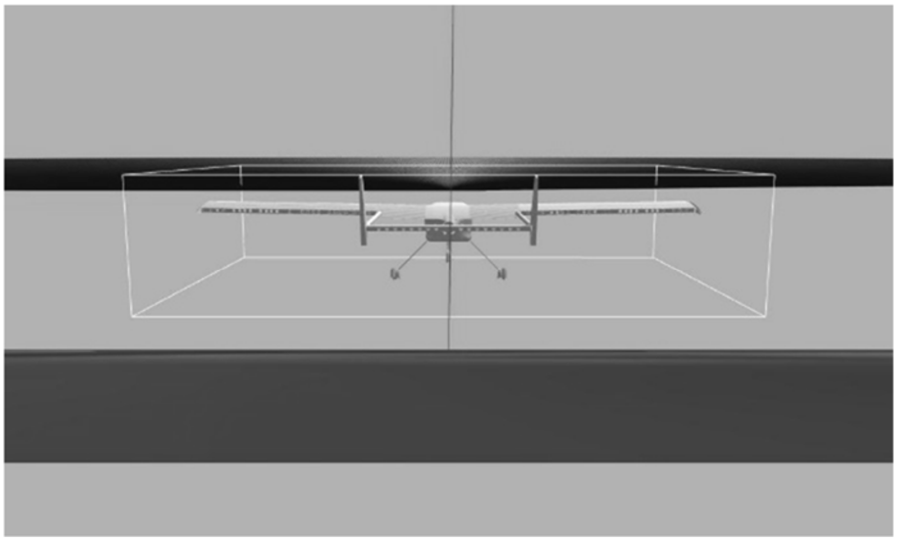
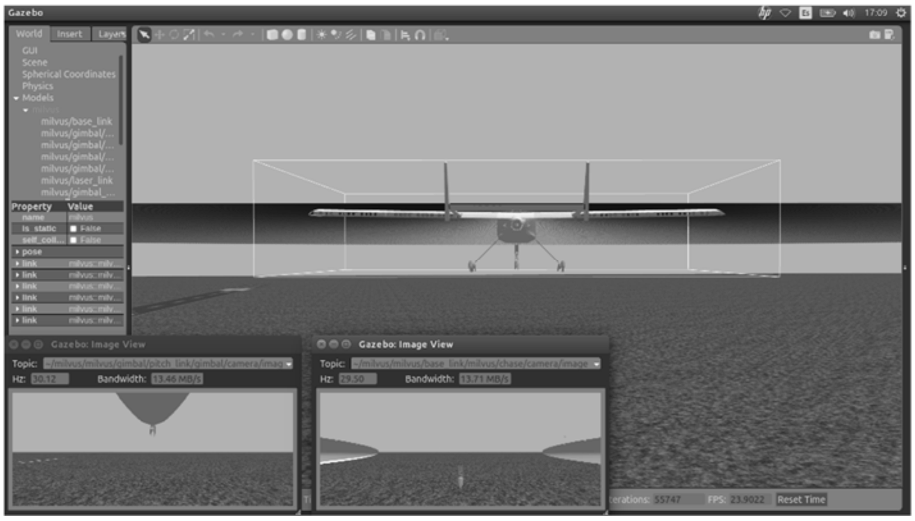


Figure V.44 *Milvus UAV in Gazebo straight flights in Dubins path (Source: Screenshot)*

Milvus fake cameras and lidar

Figure V.44 shows the Milvus UAV two onboard cameras images and lidar projections while performing a path following.



name: milvus
is_static: False
self_coll...: False

Topic: ~/milvus/milvus/gimbal/pitch_link/gimbal/camera/imag
Hz: 30.12 Bandwidth: 13.46 MB/s

Topic: ~/milvus/milvus/base_link/milvus/chase/camera/image
Hz: 29.50 Bandwidth: 13.71 MB/s

Iterations: 55747 FPS: 23.9022

Figure V.45 *Milvus two onboard cameras images (Source: Screenshot)*

Conclusions

The unmanned vehicles, like other mobile robots, are expected to increase in several units and complexity, enabling autonomous smart factories and homes, intelligent transportation, and intelligent production systems (agriculture, mining, fishing) and other civilian activities like rescue, structural and environmental monitoring. Then, the research activities need to encompass the continuously evolving and challenging end-user applications and technologies, in their broad sense, involved in mobile robotics platforms.

Our UNISA-UVF framework, based on Gazebo-ROS, has the main functionalities for unmanned vehicle's end-to-end development. It facilitates the modeling and simulation of UGV and UAV from scratch; also, it integrates other popular frameworks, like Solidworks, MatLab, Simulink, and X-Plane. Different UGVs and UAVs have been modeled and simulated successfully by using a 3-D CAD modeler. All the vehicles we implement in the platform had testing purposes of the selected software integration: either by using the model editor of the platform, either by using converted and implemented designs from other CAD tools. Furthermore, the kinematics and dynamics embedded in the different packages were reviewed and selected accordingly to the needs of vehicles' mechanical systems behavior on tested use-cases with different control packages. UNISA-UVF has been capable of coping successfully with the simulated physical environments while performing the different missions we tried. Even though Gazebo-ROS seems challenging to implement and use, it shows all its potential as a collaborative environment to create missions that extend the capabilities of X-Plane and other software by the addition of sensing capabilities and cooperative work.

For these reasons, the UNISA-UVF framework needs to be enhanced continuously as a testbed for unmanned vehicles for customized purposes, with new integrated tools, methods, models, hardware, and software components. From the end-user applications' perspective, one future development could be the incorporation of artificial intelligence to unisa-bots, another future work would be the enhance testing of mobile robots working collaboratively. From the UNISA-UVF framework evolution, one next research project would be the migration to ROS2, which is still under a massive development (releasing new versions every six months) but today are getting stable to start testing the use-cases already did until now, incorporating the highly distributed and real-time middleware capabilities. Another significant improvement will be the creation of a user-friendly environment - with a graphical interface in order to easily interact with the multiple files needed to launch simulated robotic environments, including tools for dynamic interaction with documented procedures of unmanned vehicles use-cases.

References

Alexander, R. M. (1989). Optimization and gaits in the locomotion of vertebrates. Physiological reviews, 69(4), 1199-1227.

Antonelli, G., Chiaverini, S., & Fusco, G. (2005). A calibration method for odometry of mobile robots based on the least-squares technique: theory and experimental validation. IEEE Transactions on Robotics, 21(5), 994-1004.

Arkin, R. C., & Arkin, R. C. (1998). Behavior-based robotics. MIT press.

Avots, D., Lim, E., Thibaux, R., & Thrun, S. (2002). A probabilistic technique for simultaneous localization and door state estimation with mobile robots in dynamic environments. In Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on (Vol. 1, pp. 521-526). IEEE.

Bambino, I. (2008). Una introducción a los robots móviles. AADECA, Buenos Aires.

Baraff, D. (1989, July). Analytical methods for dynamic simulation of non-penetrating rigid bodies. In ACM SIGGRAPH Computer Graphics (Vol. 23, No. 3, pp. 223-232). ACM.

Bay, H., Ess, A., Tuytelaars, T., & Van Gool, L. (2008). Speeded-up robust features (SURF). Computer vision and image understanding, 110(3), 346-359.

Brogliato, B., Ten Dam, A. A., Paoli, L., Genot, F., & Abadie, M. (2002). Numerical simulation of finite dimensional multibody non-smooth mechanical systems. Applied Mechanics Reviews, 55(2), 107-150.

Beard, R. W., & McLain, T. W. (2012). Small unmanned aircraft: Theory and practice. Princeton university press.

Borenstein, J., Everett, H. R., & Feng, L. (1996). Navigating mobile robots: Systems and techniques (pp. 1-225). Wellesley, MA: AK Peters. Navigating mobile robots: Systems and techniques (pp. 1-225). Wellesley, MA: AK Peters.

Boyen, X., & Koller, D. (1998, July). Tractable inference for complex stochastic processes. In Proceedings of the Fourteenth Conference on Uncertainty in artificial intelligence (pp. 33-42). Morgan Kaufmann Publishers Inc.

Campion, G., Bastin, G., & Dandrea-Novel, B. (1996). Structural properties and classification of kinematic and dynamic models of wheeled mobile robots. IEEE transactions on robotics and automation, 12(1), 47-62.

Carlos, C. D. W., Siciliano, B., & Bastin, G. (1997). Theory of Robot Control.

Crick, C., Jay, G., Osentoski, S., Pitzer, B., & Jenkins, O. C. (2017). Rosbridge: Ros for non-ros users. In Robotics Research (pp. 493-504). Springer, Cham.

Cole, D. M., & Newman, P. M. (2006, May). Using laser range data for 3D SLAM in outdoor environments. In Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on (pp. 1556-1563). IEEE.

Cosmin Petra, Bogdan Gavrea, Mihai Anitescu & Florian Potra. (2009). A computational study of the use of an optimization-based method for simulating large multibody systems, Optimization Methods and Software, 24:6, 871-894, DOI: 10.1080/10556780902806094.

Darmanin, R. N., & Bugeja, M. K. (2016, July). Autonomous Exploration and Mapping using a Mobile Robot Running ROS. In ICINCO (2) (pp. 208-215).

Dellaert, F., Burgard, W., Fox, D., & Thrun, S. (1999). Using the condensation algorithm for robust, vision-based mobile robot localization. In Computer Vision and Pattern Recognition, 1999. IEEE Computer Society Conference on. (Vol. 2, pp. 588-594). IEEE.

De Simone, M.C.; Russo, S.; Rivera, Z.B. & Guida, D. (2018). Multibody Model of a UAV in Presence of Wind Fields. In: Proceedings - 2017 International Conference on Control, Artificial Intelligence, Robotics and Optimization, ICCAIRO 2017, pp. 83-88. doi: 10.1109/ICCAIRO.2017.26.

De Simone, M.C. & Guida, D. (2017) On the development of a low-cost device for retrofitting tracked vehicles for autonomous navigation. In: AIMETA 2017 - Proceedings of the 23rd Conference of the Italian Association of Theoretical and Applied Mechanics, Vol. 4 (2017), pp. 71-82.

De Simone, M.C. & Guida, D. (2018). Control design for an under-actuated UAV model. FME Transactions, 46(4), 443-452.

De Simone, M.C.; Rivera, Z.B. & Guida, D. (2018). Obstacle avoidance system for unmanned ground vehicles by using ultrasonic sensors. Machines, Vol. 6, No. 2, 18 (2018). doi: 10.3390/machines6020018.

De Simone, M.C. & Guida, D.: Identification and control of an Unmanned Ground Vehicle by using Arduino. UPB Scientific Bulletin, Series D: Mechanical Engineering, Vol. 80, No. 1, (2018), pp. 141-154.

Denavit, J., & Hartenberg, R. S. (1955). Kinematic modeling for robot calibration. Trans. ASME Journal of Applied Mechanics, 22, 215-221.

Douillard, B., Underwood, J., Melkumyan, N., Singh, S., Vasudevan, S., Brunner, C., & Quadros, A. (2010, October). Hybrid elevation maps: 3D surface models for segmentation. In Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on (pp. 1532-1538). IEEE.

Dryanovski, I., Morris, W., & Xiao, J. (2010, October). Multi-volume occupancy grids: An efficient probabilistic 3D mapping model for micro aerial vehicles. In Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on (pp. 1553-1559). IEEE.

Dudek, G., & Jenkin, M. (2010). Computational principles of mobile robotics. Cambridge university press.

Einhorn, E., Langner, T., Stricker, R., Martin, C., & Gross, H. M. (2012, October). Mira-middleware for robotic applications. In 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems (pp. 2591-2598). IEEE.

Engelson, S. P., & McDermott, D. V. (1992, April). Image signatures for place recognition and map construction. In Sensor Fusion IV: Control Paradigms and Data Structures (Vol. 1611, pp. 282-294). International Society for Optics and Photonics.

Fankhauser, P., & Hutter, M. (2016). A universal grid map library: Implementation and use case for rough terrain navigation. In Robot Operating System (ROS) (pp. 99-120). Springer, Cham.

Fairfield, N., Kantor, G., & Wettergreen, D. (2007). Real ‐ time SLAM with octree evidence grids for exploration in underwater tunnels. Journal of Field Robotics, 24(1 ‐ 2), 03-21.

Flynn, A. M. (1985). Redundant sensors for mobile robot navigation.

Foote, T. (2013, April). tf: The transform library. In 2013 IEEE Conference on Technologies for Practical Robot Applications (TePRA) (pp. 1-6). IEEE.

Fournier, J., Ricard, B., & Laurendeau, D. (2007, May). Mapping and exploration of complex environments using persistent 3D model. In null (pp. 403-410). IEEE.

Fox, D., Burgard, W., Kruppa, H., & Thrun, S. (2000). A probabilistic approach to collaborative multi-robot localization. Autonomous Robots, 8(3), 325-344.

Fox, D., Burgard, W., & Thrun, S. (1999). Markov localization for mobile robots in dynamic environments. Journal of artificial intelligence research, 11, 391-427.

Fox, D., Burgard, W., & Thrun, S. (1997). The dynamic window approach to collision avoidance. IEEE Robotics & Automation Magazine, 4(1), 23-33.

Funda, J., Taylor, R. H., & Paul, R. P. (1990). On homogeneous transforms, quaternions, and computational efficiency. IEEE Transactions on Robotics and Automation, 6(3), 382-388.

Gage, D. W. (1995). UGV history 101: A brief history of Unmanned Ground Vehicle (UGV) development efforts. NAVAL COMMAND CONTROL AND OCEAN SURVEILLANCE CENTER RDT AND E DIV SAN DIEGO CA.

Gallagher, A. G., Ritter, E. M., Champion, H., Higgins, G., Fried, M. P., Moses, G., ... & Satava, R. M. (2005). Virtual reality simulation for the operating room: proficiency-based training as a paradigm shift in surgical skills training. Annals of surgery, 241(2), 364.

Guivant, J. E., & Nebot, E. M. (2001). Optimization of the simultaneous localization and map-building algorithm for real-time implementation. IEEE transactions on robotics and automation, 17(3), 242-257.

Gutmann, J. S., & Fox, D. (2002). An experimental comparison of localization methods continued. In Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on (Vol. 1, pp. 454-459). IEEE.

Hadsell, R., Bagnell, J. A., Huber, D. F., & Herbert, M. (2009, June). Accurate rough terrain estimation with space-carving kernels. In Robotics: Science and Systems (Vol. 2009).

Harris, C., & Stephens, M. (1988, August). A combined corner and edge detector. In Alvey vision conference (Vol. 15, No. 50, pp. 10-5244).

Herbert, M., Caillas, C., Krotkov, E., Kweon, I. S., & Kanade, T. (1989, May). Terrain mapping for a roving planetary explorer. In Proceedings, 1989 International Conference on Robotics and Automation (pp. 997-1002). IEEE.

Hornung, A., Wurm, K. M., & Bennewitz, M. (2010, October). Humanoid robot localization in complex indoor environments. In IROS (pp. 1690-1695).

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., & Burgard, W. (2013). OctoMap: An efficient probabilistic 3D mapping framework based on octrees. Autonomous robots, 34(3), 189-206.

Howard, A., Koenig, N., Hsu, J., & Dolha, M. (2014). Gazebo simulator website.

HSU, John M.; PETERS, Steven C. Extending open dynamics engine for the DARPA virtual robotics challenge. En International Conference on Simulation, Modeling, and Programming for Autonomous Robots. Springer, Cham, 2014. p. 37-48.

Hwang, Y. K., & Ahuja, N. (1992). Gross motion planning—a survey. ACM Computing Surveys (CSUR), 24(3), 219-291.

Inoue, K., Otsuka, K., Sugimoto, M., & Murakami, N. (1997, September). Estimation of place of tractor and adaptive control method of autonomous tractor using INS and GPS. In Preprints of the International Workshop on Robotics and Automated Machinery for Bio-Productions (pp. 27-36).

Jones, D. G., & Malik, J. (1992). Computational framework for determining stereo correspondence from a set of linear spatial filters. Image and Vision Computing, 10(10), 699-708.

Kalman, R. E. (1960). A new approach to linear filtering and prediction problems. Journal of basic Engineering, 82(1), 35-45.

Kramer, J., & Scheutz, M. (2007). Development environments for autonomous mobile robots: A survey. Autonomous Robots, 22(2), 101-132.

Kavraki, L., Svestka, P., & Overmars, M. H. (1994). Probabilistic roadmaps for path planning in high-dimensional configuration spaces (Vol. 1994). Unknown Publisher.

Kim, D. H., Wang, H., & Shin, S. (2006). Decentralized control of autonomous swarm systems using artificial potential functions: Analytical design guidelines. Journal of Intelligent and Robotic Systems, 45(4), 369-394.

Koenig, N. P., & Howard, A. (2004, September). Design and use paradigms for Gazebo, an open-source multi-robot simulator. In IROS (Vol. 4, pp. 2149-2154).

Koenig, Nate. et al. Gazebo. Retrieved, 2012, vol. 3, no 26, p. 2012.

Korf, R. E. (1985). Depth-first iterative-deepening: An optimal admissible tree search. Artificial intelligence, 27(1), 97-109.

Koubaa, A., Alajlan, M., & Qureshi, B. (2017). ROSLink: Bridging ROS with the Internet-of-Things for Cloud Robotics. In Robot Operating System (ROS) (pp. 265-283). Springer, Cham.

KPMG Global Automotive Executive Survey [Online] Available: https://assets.kpmg.com/content/dam/kpmg/nl/pdf/2018/sector/automotive/global-automotive-executive-survey-2018.pdf

KuCuk, S., & Bingul, Z. (2004). The inverse kinematics solutions of industrial robot manipulators. IEEE Conference on Mechatronics, 274-279.

Kunze, L., Roehm, T., & Beetz, M. (2011, May). Towards semantic robot description languages. In Robotics and Automation (ICRA), 2011 IEEE International Conference on (pp. 5589-5595). IEEE.

Latombe, J. C. (2012). Robot motion planning (Vol. 124). Springer Science & Business Media.

Laumond, J. P., Sekhavat, S., & Lamiraux, F. (1998). Guidelines in nonholonomic motion planning for mobile robots. In Robot motion planning and control (pp. 1-53). Springer, Berlin, Heidelberg.

Lin, M., & Gottschalk, S. (1998, May). Collision detection between geometric models: A survey. In Proc. of IMA conference on mathematics of surfaces (Vol. 1, pp. 602-608).

Litman, T. (2017). Autonomous vehicle implementation predictions. Victoria, Canada: Victoria Transport Policy Institute.

Lowe, D. G. (1999). Object recognition from local scale-invariant features. In Computer vision, 1999. The proceedings of the seventh IEEE international conference on (Vol. 2, pp. 1150-1157). Ieee.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. International journal of computer vision, 60(2), 91-110.

Lozano-Pérez, T., & Wesley, M. A. (1979). An algorithm for planning collision-free paths among polyhedral obstacles. Communications of the ACM, 22(10), 560-570.

Lyshevski, S. E., & Nazarov, A. (2000, September). Lateral maneuvering of ground vehicles: modeling and control. In American Control Conference, 2000. Proceedings of the 2000 (Vol. 1, No. 6, pp. 110-114). IEEE.

Maitland, N., & Harris, C. (1994, September). A Video Based Tracker for use in Computer Aided Surgery. In BMVC (pp. 1-10).

Marder-Eppstein, E., Berger, E., Foote, T., Gerkey, B., & Konolige, K. (2010, May). The office marathon: Robust navigation in an indoor office environment. In Robotics and Automation (ICRA), 2010 IEEE International Conference on (pp. 300-307). IEEE.

Martinez, A., & Fernández, E. (2013). Learning ROS for robotics programming. Packt Publishing Ltd.

Meagher, D. (1982). Geometric modeling using octree encoding. Computer graphics and image processing, 19(2), 129-147.

Montemerlo, M., Thrun, S., Koller, D., & Wegbreit, B. (2002). FastSLAM: A factored solution to the simultaneous localization and mapping problem. Aaai/iaai, 593598.

Moravec, H. P. (1977). Techniques towards automatic visual obstacle avoidance.

Muir, P. F., & Neuman, C. P. (1987). Kinematic modeling of wheeled mobile robots. Journal of robotic systems, 4(2), 281-340.

Murphy, R. R. (2014). Disaster robotics. MIT press.

Nelson, D. R., Barber, D. B., McLain, T. W., & Beard, R. W. (2007). Vector field path following for miniature air vehicles. IEEE Transactions on Robotics, 23(3), 519-529.

Nilsson, N. J. (1969). A mobile automaton: An application of artificial intelligence techniques. SRI INTERNATIONAL MENLO PARK CA ARTIFICIAL INTELLIGENCE CENTER.

Nüchter, A., Lingemann, K., Hertzberg, J., & Surmann, H. (2007). 6D SLAM—3D mapping outdoor environments. Journal of Field Robotics, 24 (8-9), 699-722.

O'Connor, M., Bell, T., Elkaim, G., & Parkinson, B. (1996). Automatic steering of farm vehicles using GPS. Precision Agriculture, (precisionagricu3), 767-777.

Pathak, K., Birk, A., Poppinga, J., & Schwertfeger, S. (2007, October). 3d forward sensor modeling and application to occupancy grid based sensor fusion. In Intelligent Robots and Systems, 2007. IROS 2007. IEEE/RSJ International Conference on (pp. 2059-2064). IEEE.

Payeur, P., Hébert, P., Laurendeau, D., & Gosselin, C. M. (1997, April). Probabilistic octree modeling of a 3d dynamic environment. In Robotics and Automation, 1997. Proceedings., 1997 IEEE International Conference on (Vol. 2, pp. 1289-1296). IEEE.

Pyo, Y. S. (2015). ROS Robot Programming.

Qian, W., Xia, Z., Xiong, J., Gan, Y., Guo, Y., Weng, S., ... & Zhang, J. (2014, December). Manipulation task simulation using ros and gazebo. In Robotics and Biomimetics (ROBIO), 2014 IEEE International Conference on (pp. 2594-2598). IEEE.

Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., ... & Ng, A. Y. (2009, May). ROS: an open-source Robot Operating System. In ICRA workshop on open source software (Vol. 3, No. 3.2, p. 5).

Ryde, J., & Hu, H. (2010). 3D mapping with multi-resolution occupied voxel lists. Autonomous Robots, 28(2), 169.

Rivera, Z.B.; De Simone, M.C.; Guida, D. (2016). Unmanned Ground Vehicle Modelling in Gazebo/ROS-Based Environments. Machines, 2019, 7, 42.

Roth, B. (1994). Computational advances in robot kinematics. In Advances in Robot Kinematics and Computational Geometry (pp. 7-16). Springer, Dordrecht.

Roth-Tabak, Y., & Jain, R. (1989). Building an environment model using depth information. Computer, 22(6), 85-90.

Samson, C., & Ait-Abderrahim, K. (1991, April). Feedback control of a nonholonomic wheeled cart in cartesian space. In Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on (pp. 1136-1141). IEEE.

Siegwart, R., Nourbakhsh, I. R., & Scaramuzza, D. (2011). Introduction to autonomous mobile robots. MIT press.

Sinha, V., Doucet, F., Siska, C., Gupta, R., Liao, S., & Ghosh, A. (2000, September). YAML: a tool for hardware design visualization and capture. In Proceedings of the 13th international symposium on System synthesis (pp. 9-14). IEEE Computer Society.

Smith, Russell, et al. (2005). Open dynamics engine.

Swain, M. J., & Ballard, D. H. (1991). Color indexing. International journal of computer vision, 7(1), 11-32.

Takaya, K., Asai, T., Kroumov, V., & Smarandache, F. (2016, October). Simulation environment for mobile robots testing using ros and gazebo. In System Theory, Control and Computing (ICSTCC), 2016 20th International Conference on (pp. 96-101). IEEE.

Taylor, J. R., Drumwright, E. M., & Hsu, J. (2016, December). Analysis of grasping failures in multi-rigid body simulations. In Simulation, Modeling, and Programming for Autonomous Robots (SIMPAR), IEEE International Conference on (pp. 295-301). IEEE.

Thomas, F., & Ros, L. (2005). Revisiting trilateration for robot localization. IEEE Transactions on robotics, 21(1), 93-101.

Tsardoulias, E., & Mitkas, P. (2017). Robotic frameworks, architectures and middleware comparison. arXiv preprint arXiv:1711.06842.

Varadhan, G., & Manocha, D. (2004, October). Accurate Minkowski sum approximation of polyhedral models. In Computer Graphics and Applications, 2004. PG 2004. Proceedings. 12th Pacific Conference on (pp. 392-401). IEEE.

Vidan, C., & Badea, S. I. (2016). Longitudinal automatic control system for a light weight aircraft. INCAS Bulletin, 8(4), 157.

Wirth, S., & Pellenz, J. (2007, September). Exploration transform: A stable exploring algorithm for robots in rescue environments. In Workshop on Safety, Security, and Rescue Robotics (Vol. 9, pp. 1-5).

Acronyms and Abbreviations

ALV	Autonomous Land Vehicles
API	Application Program Interface
DARPA	Defense Advanced Research Projects Agency
DART	Dynamic Animation and Robotics Toolkit
DOF	Degree of Freedom
DWA	Dynamic Window Approach
EKF	Extended Kalman Filter
FAA	Federal Aviation Administration
GPS	Global Positioning System
HMI	Human–Machine Interface
ICC	Instant Curvature Center
ICR	Instantaneous Center of Rotation
IFR	International Federation of Robotics
LADAR	Laser Detection and Ranging
LiDAR	Light Detection and Ranging
LAN	Local Area Network
NASA	National Aeronautics and Space Administration
NLP	Natural Language Processing
MAV	Micro Aerial Vehicle
ODE	Open Dynamics Engine

OpenGL	Open Graphics Library
PF	Particle Filter
PID	Proportional-Integral-Derivative feedback control
PRM	Probabilistic Route Maps
RF	Radio Frequency
RGB	Red, Green, Blue
ROS	Robotic Operating System
RPP	Randomized Path Planner
RSTA	Reconnaissance, Surveillance, and Target Acquisition
SDF	Simulation Description Format
SIL	Software-In-the-Loop
SLAM	Simultaneous Localization and Mapping
S&T	Science and Technology
TRL	Technology Readiness Level
UAV	Unmanned Air Vehicle
UGV	Unmanned Ground Vehicle
UNISA	Università degli Studi di Salerno
UNISA-UVF	UNISA Unmanned Vehicle Framework
URDF	Universal Robotic Description Format
UUV	Unmanned Underwater Vehicle
UVs	Unmanned Vehicles
VTOL	Vertical Takeoff and Landing