POLITECNICO DI TORINO

Master Degree Course in Mechatronic Engineering

Master Degree Thesis

Autonomous obstacle avoidance strategy for Unmanned Aerial Vehicles

Supervisors

Prof. Alessandro Rizzo
Rrof. Stefano Primatesta

Condidate

Kseniia Nikitina

October, 2021

Acknowledgements

First of all, I would like to thank Professors Alessandro Rizzo and Stefano Primatesta for giving me the opportunuty of studying for my Master Thesis guided by them. Without his supervision, resources and connections this project would not have been possible.

Finally, I am thankful to my parents, friends and loved ones for their continued moral and material support throughout the course and in helping me to finalize the project.

Abstract

The diffusion of Intelligent Unmanned Aerial Vehicles (UAVs) requires the development of autonomous flying techniques. Specifically, path and trajectory planning enable UAVs to avoid obstacles and reach a specific target efficiently.

The focus of this work is the development of an obstacle avoidance strategy to autonomously move an aerial robotic platform in an outdoor environment in presence of obstacles.

In this work, a global and local path planning for autonomous flight of UAVs is proposed. The adopted solution is based on the RRT* (Rapidly-exploring Random Tree star) and the VFH (Vector Field Histogram) algorithms to provide the global and local paths, respectively.

The RRT* constructs a unidirectional search tree by randomly selecting and interconnecting obstaclefree states until a tree branch reaches the goal node. The VFH algorithm is computationally efficient, very robust and insensitive to misreadings, and it allows continuous and fast motion of the drone without stopping for obstacles. Thanks to VFH, the UAV can move between very densely cluttered obstacle courses at high average speeds and is able to pass through narrow openings or negotiate narrow corridors without oscillations.

The developed strategy is based on the well-known Robot Operating System (ROS) framework. Specifically, autonomous missions were performed through the continuous interaction between the ground station and the UAV exploiting the mavros ROS node. In particular, the UAV detects obstacles through 2D Lidar and generates the local and global paths using the RRT* and VFH. The ground station supports the flight mission by generating a two-dimensional map using on-board sensor data and exploiting a SLAM algorithm, transmits the destination point to the UAV, and manages the state of the aircraft. The autonomous flight system and the proposed obstacle avoidance strategy are verified using the Gazebo simulator performing a simulated flight in the three-dimensional space. Simulation results demonstrate that the simultaneous use of RRT* and VFH enables the autonomous flight while avoiding obstacles and reaching a goal position.

Contents

List of Figures

List of Tables

List of Abbreviations

CC Companion Computer
DHCP Dynamic Host Configuration Protocol
CCW Counter Clockwise
CW Clockwise
DOF Degree of Freedom
FC Flight Controller
ENU East North Up
ESC Electronic Speed Controller
GPS Global Positioning System
EKF Extended Kalman Filter
FCU Flight Controller Unit
HAPP Heterogeneous Agent Path Problem
IMU Inertial Measurement Unit
IP Internet Protocol
LIDAR Light Detection and Ranging
LTS Long-Term Support
LP Low Pass
MAVLINK Micro Air Vehicle Communication Protocol
MAVROS ROS interface for MAVLINK
MGRS Military Grid Reference System
MPC Model Predictive Control
MAV Micro Air Vehicle
NED North East Down
OS Operating System
PID ProportionalIntegralDerivative controller
PWM Pulse Width Modulation
ROS Robot Operating System
RPY Roll Pitch Yaw
RTF Ready to Fly
SSH Secure Socket Shell
SITL Software in the Loop
TCP Transmission Control Protocol
NED North East Down
LAN Local Area Network
MAC Media Access Control
UAS Unmanned Aerial System

UAV Unmanned Arial Vehicle
UDP User Datagram Protocol
URI Uniform Resource Identifier
URL Uniform Resource Locator
USB Universal Serial Bus
WGS World Geodetic System
QGC QGroundControl

Glossary

Unmanned Aerial System An aircraft without a human pilot on board and a type of unmanned-vehicle.
Robot Operating System A flexible framework for writing robot software including a collectionof tools, libraries, and conventions that aim to simplify the task of creating complex androbust robot behavior across a wide variety of robotic platforms.
GitHub a Git repository hosting service.
Dronecode A nonprofit hosted under the Linux Foundation, dedicated to fostering open-source components and their communities.
PX4 Autopilot An open-source BSD-Licensed flight control software for drones and otherunmanned vehicles, a project by Dronecode.
MAVLink An open-source lightweight communication protocol for UAV systems andcomponents, widely used for communications between ground-stations, autopilots andcompanion computers, a project by Dronecode.
QGroundControl An open-source feature complete and fully customizable control station forMAVLink based Drones, a project by Dronecode.
MAVROS A MAVLink extendable communication node for ROS with proxy for GroundControl Station.
Gazebo Simulator A ROS compatible open-source 3D robotics simulator.
EKF2 Estimation System An attitude and position estimator using an Extended Kalman Filter.

1 Introduction

This chapter introduces the thesis, focusing on its concept and definition of this thesis development, such as motivation, target goals and its research methods and questions.

1.1 Overview

Humans are always searching for a better solution to resolve day-to-day problems. There is always something that needs to be more practical, efficient, affordable or even easier to interact with human beings. Due to that fact, there was a necessity of creating something that resolve those problems in a way that can guarantee same operational functions as human operators or even new ways to do the same thing with less effort. To accomplish that, Unmanned Aerial Vehicles (UAVs) were developed to preform actions that were difficult for human beings to execute.They are different from Manned Aerial Vehicles. UAVs are remotely controlled and Manned Aerial Vehicles are locally controlled. UAVs are characterized by their size, weight and maneuver. They are very small and light-weighted when compared to real manned aerial vehicles, such as commercial travel planes or even civil aircrafts.

In the past decade, the interest in UAVs and autonomy has constantly increased, as cited in [1].

1.2 UAVs applications

Since UAVs provide supremacy over conventional remote sensing technologies and their benefits lie in terms of less power consumption, less risk to human life, ease to data collection, hovering, and ultra-high spatial resolution forges them an excellent choice for surveying and mapping. The studies demonstrate the relevance and uniqueness of drones in the civil, logistics, agriculture and defense sectors. Figure 1.1 depicts the potential applications of UAV in civil, environment and defence sectors.

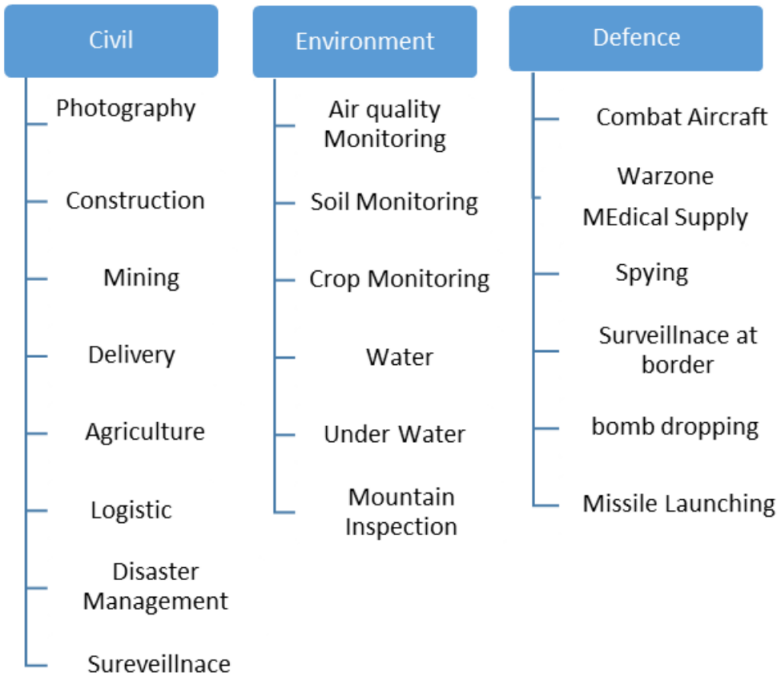


Figure 1.1: Potential Applications of UAVs.

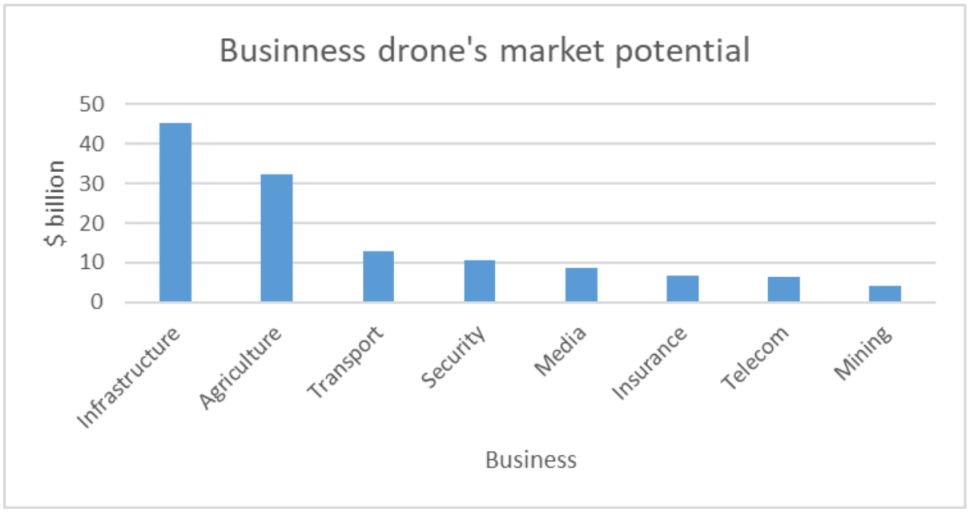


Figure 1.2: Market Potential of UAVs.

1.2.1 Drones in Defence

The advent of UAV was started initially with the aim of transacting the war missions like intelligence, spying, reconnaissance vigilance and target detection, later they were introduced for civil and logistic applications, therefore UAVs technology is used in the military sector, which is also the sector where this technology is most advanced. UAVs are used by armies to patrol battle zones but also to provide supplies to soldiers or civil people.

Outside the military sector, UAVs are also used by police, sea rescue, fire departments etc. to carry out some of their operations. About this, for instance, UAVs have become valuable in replacing helicopters in operations for which an aircraft is needed, but where UAVs can bring peculiar benefits such as their greater flexibility, as well as their lower operational costs.

1.2.2 Drones in Agriculture

Then, the second sector where UAVs are used the most is agriculture. Here they are mainly used for crop monitoring and distribution of pesticides, but also for surveillance. UAVs are particularly useful for the careful monitoring of large areas of farmland, considering factors such as slope and elevation, for example, to identify the most suitable seeding prescriptions.

1.2.3 Drones in Forestry, Fisheries and Wildlife protection

Drone bridges the gap of satellite imagery of less availability and cloud cover by performing various tasks like measuring canopy height, tracking of forest wildlifes, support in forest management, forest fires detection and control, survey forests and mapping canopy gaps easily. Animal poaching and adverse climate conditions have a destructive impact on wildlife. UAV equipped with thermal sensors along with satellite are being approached for monitoring, tagging and counting of animals which help to curb the poaching of animals and conserve the wildlife.

1.2.4 Drones for Civil applications

UAVs are also changing the world of filming and video shooting, both as regards the world of cinema but also at an amateur level. Indeed, in the last 20 years, the use of UAVs in the cinema business has increased exponentially, but UAV started also to get a lot of attention from the consumer public, due to the reduction in costs and the greater availability. Despite this, to date the regulations are still vague regarding the use of UAVs, mostly for autonomous ones, for reasons of privacy and safety.
Last but not least, international delivery companies have lately shown a particular interest in using autonomous UAVs for the delivery of commercial goods. This fact proves the necessity for a common regulation between nations for what regards the flight of autonomous aerial vehicles.

Figure 1.3: Examples of applications where UAVs are used nowadays.

1.3 Thesis Objective

The main goal of this work is to develop obstacle avoidance strategy to autonomously move an aerial robotic platform in outdoor environment with obstacles. The developed solution is expected to acccount for static obstacles. Some of these obstacles are generally known a priory, so the proposed solution will follow a classic two layered architactures. In this work, a global and local path planning for autonomous flight of UAVs was chosen. The adopted solution is based on the RRT* (Rapidly-exploring Random Tree star) and the VFH (Vector Field Histogram) algorithms to provide the global and local paths, respectively. The RRT* constructs a unidirectional search tree by randomly selecting and interconnecting obstaclefree states until a tree branch reaches the goal node. The VFH algorithm is computationally efficient, very robust and insensitive to misreadings, and it allows continuous and fast motion of the drone without stopping for obstacles. Thanks to VFH, the UAV can move between very densely cluttered obstacle courses at high average speeds and is able to pass through narrow openings or negotiate narrow corridors without oscillations.

Another goal of this project focus on the testing and validation of the collision avoidance algorithms and demonstrate it in different scenarios using the Gazebo world.

1.4 Motivation of this thesis

The work presented in this thesis regards the implementation of an autonomous path planning algorithm for aerial vehicles in overground zones. In this research, tremendous efforts have been dedicated to the collision avoidance of UAVs and to improve the performance of autonomous navigation to further ensure the safety and reliability of UAVs. To address the above-mentioned problems, the research objectives are to implement a novel and adaptive UAV path planning

algorithm for an urban environment that can handle multiple UAVs and unmolded obstacles in a static environment and real-time collision avoidance using 2D LiDar sensor.

1.5 Thesis Outline

This thesis is made up of several distinct chapters.

- Chapter 1 - Introduction: is introduction and outlines the motivations that stimulate the work and get them interested in the autonomous flight system of the drone.
- Chapter 2 - Literature Review: is related to the literature review for this thesis, hence the topics involved are an overview of path planning algorithms, the state-of-the-art and its implementations.
- Chapter 3 - Overview of Quadrotor Technology: Identification of generic quadrotor model proposed for development research. Introduces the background of the project and describes the UAVs components needed to achieve the project objectives
- Chapter 4 - Quadrotor model: It is divided it two parts, the firstcone where a description of the considered reference frames and the available control actions is performed. In the second part, a dynamic model is developed with respect to different reference frames.
- Chapter 5 - Autonomous Flight System of UAV through Global and Local Path Generation: Explain the path planning metods and obstacle avoidance approach and the implementation of them.
- Chapter 6 - System Development: presents the test and simulation result on each algorithm in different scenarios, and the combination of all algorithms for urban simulation test, with suggestions for possible future works on the subject.
- Chapter 7 - Simulation and Experiments of the Simulations: Implementation of the simulations techniques, outline also the advantages/disadvantages and limitation of the algorithms.
- Chapter 8 - Conclusion: Analyses andconclusions and discusses on what and how can be improved the work done.

2 Literature Review

This section covers the following topics: first, an brief classification about the UAVs by the flying principle. Then, an overview of the history of the approaches used to solve the path planning problem is done in order to introduce the topic. Study a selection of works regarding obstacle avoidance and background information needed to understand and motivate the method developed in this thesis. Most collision avoidance techniques have been developed for actors in two-dimensional environments. However, many of these techniques can be extended to three-dimensions. In this chapter, is present various collision avoidance approaches, their strengths and drawbacks and various approaches to UAV collision avoidance provided by others, focusing the attention on the most important aspects for the path planning.

2.1 Classification

The quadrotor is one category of the aerial vehicles which can be classified according to its flying principle. The classification diagram is represented in Figure 2.1.

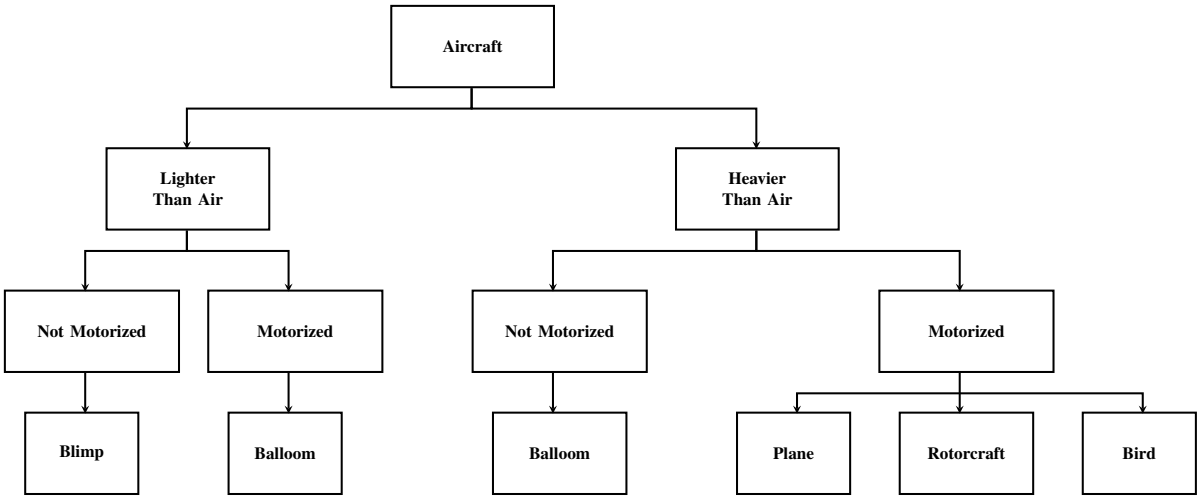


Figure 2.1: Aircraft Classification Depending on Flying Principle.

The micro aerial vehicle is evaluated as a perfect example of quick navigation among the motorized category. In addition, vertical take-off and landing quadcopters are categorized under the same class. On the other hand, UAVs can be classified according to the use of it in our life. In civil usage, fixed wing is used, for instance, in wide-area and high attitude tasks. Mostly, quadcopters are more useful in the scientific field, for example, meteorological and surveillance. In military usage, they equipped with advanced materials and sensors to realize more complicated missions. Furthermore, [8] indicate that most of the flapping wings is categorized with the micro UAVs but they are still under construction because of their limitation such as low payload capacity and low stability level. However, its ability to vertical take-off and landing and less power consumption are valuable characteristics for them. Rotary wing UAVs are usually utilized on tasks that need to hover. Compared to the previous category with the same size they are oversensitive to air disturbance. Quadcopters have become widely utilized due to some unparalleled characteristics, for example, they have small size, flexible maneuverability and easy to control. Generally, search and rescue tasks are the most important application of quadcopters, as well as, it's used in in the military fields such as homeland border protection and reconnaissance. Additionally, quadrotors have a wide utilization in science applications where they can be utilized to study environmental change. For example, ice quadratures are nimble airplanes manipulated by the rotational speed of the four rotors. Sheet elements and volcanic movement and then again for climatic inspecting. Elaboration on several applications of quadrotor.

2.2 Historical overview of Path Planning algorithms

Path planning has been one of the important problems in robotics. The primary need in robotics applications is to have an algorithm able to convert high-level instructions into low-level commands to make the robot understand how to solve a specific problem. Specifically, when the task is the motion of a mobile robot, this process is described as motion path planning.

The terms of UAVs motion planning are challenging because these vehicles have fast, com-

plex and uncertain dynamics and strict payload limitations. They are combined with real-time navigation issues in 3D space and involve continuous interaction with the environment[9]. A motion path planning is the sequence of moves and configurations that a mobile robot needs to accomplish in order to move safely towards a destination. This process usually includes also other high-level constraints, such as avoiding static or dynamic obstacles or looking for the shortest or fastest track. A motion planning algorithm works basically taking as input these constraints and producing as output the set of low-level commands that the robot has to carry out. Path planning is finding a continuous collision-free path, from a start point, to a goal point or region, and obstacles in the space.

Furthermore, since the environment where the robot operates is not always known a-priori, the motion planning path algorithms sometimes differentiate each other on the type of environment they are designed for (fully-known, partly-known or unknown) and also they can be differentiated on the type of collision avoidance they can perform, dealing either with static obstacles or with dynamic obstacles or both.

In a static and known environment, the robot knows the entire information of the environment before it starts moving. Because of this the optimal path could be computed offline before to the movement of the robot begins.

Generally, path planning methods for UAVs is mainly can be divided two levels: global planning and local planning. The global planner calculates (off-line) a path for a well-known environmental map, allowing the robot to move from start point to end point, avoiding known static obstacles, while optimizing certain goals such as finding the shortest path (to reduce travel time and energy consumption). The local planner updates (on-line) the robot's path based on information from its sensors to allow it to follow the path created by the global planner if possible, but changes it to avoid unexpected obstacles that were not taken into account.

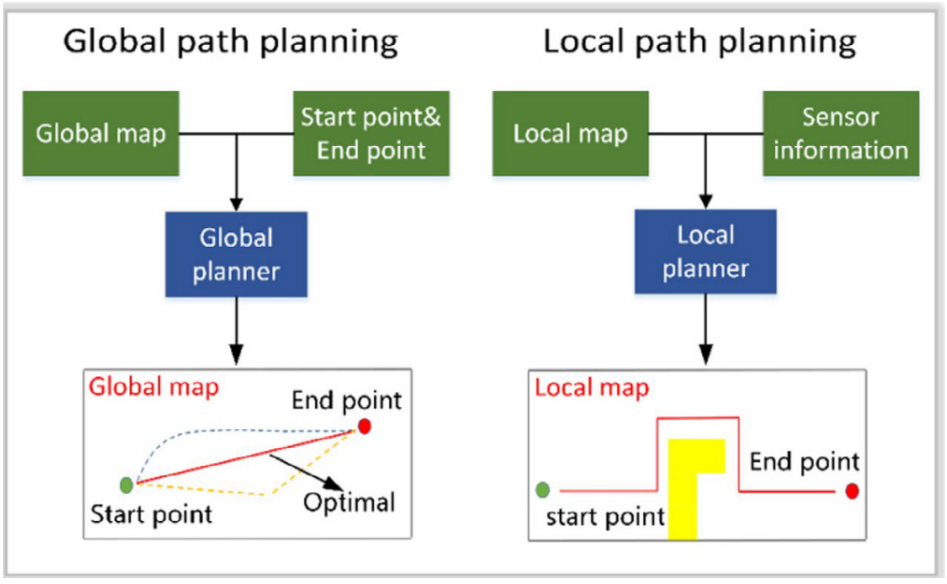


Figure 2.2: Global and local path planning.

Such as Visibility Graph method [9], Voronoi diagrams method [10], the Cell Decomposition method [11], the Potential Field method [12], [13], Virtual Force Field method (VFF),

Genetic Algorithm (GA), rapidly-exploring Random Trees (RRT) [14] and other dynamic path planning algorithms.

Visibility Graph is using in computational geometry and robot path planning, it is a graph of intervisible locations, typically for a set of points and obstacles in the Euclidean plane. Every node in the graph means a point location, and every edge represents a visible connection between them. If the line segment connecting two locations does not cross with any obstacle, an edge is drawn between them in the graph. When the set of locations lies in a line, this means as an ordered series. Visibility graphs have been extended to the real of time series analysis.

Voronoi Diagram is a partitioning of a plane into regions predicated on distance to points in a concrete subset of the plane. That set of points is designated beforehand, and for each seed there is a corresponding region consisting of all points more proximate to that seed than to any other.

Cell Decomposition is that a path between the initial and goal configuration can be resolute by subdividing the free space of the robots set into more minuscule regions called cells. After this decomposition, a connectivity graph, is constructed according to the adjacency relationships between the cells, where the nodes represent the cells in the free space, and the links between the nodes show that the corresponding cells are adjacent to each other. From this connectivity graph, a perpetual path, or channel, can be tenacious by simply following adjacent free cells from the initial setpoint to the goal point.

Potential Field and ***Artificial potential field*** methods. *Potential Field approach* is to treat the robots configuration as a point (customarily electron) in a potential field that amalgamates magnetization to the goal, and repulsion from obstacles. The resulting trajectory is output as the path. This approach has advantages in that the trajectory is engendered with little computation. However, they can become trapped in local minima of the potential field, and fail to find a path. *Artificial potential field* is a an application of artificial potential fields for avoidance the obstacles was first created by [15]. This design uses repulsive potential fields around the obstacles to push the robot away and an attractive potential field around goal to attract the robot. Therefore, the robot experiences a generalized force equal to the negative of the total potential gradient. This force runs the robot downhill towards its goal configuration until it arrives a minimum and it stops. The artificial potential field approach can be applied to both global and local methods ([16], [17]).

The potential force has two components: attractive force and repulsive force. The goal position produces an attractive force which makes the mobile robot move towards it. Obstacles generate a repulsive force, which is inversely proportional to the distance from the robot to obstacles and is pointing away from obstacles. The robot moves from high to low potential field along the negative of the total potential field. Consequently, the robot moving to the goal position can be considered from a high-value state to a low-value state.

The artificial potential fields can be achieved by direct equation similar to electrostatic potential fields or can be drive by set of linguistic rules [18]. The artificial potential field methods provide simple and effective motion planners for practical purposes. However, there is a major problem with the artificial potential field approach. It is the formation of local minima that

can trap the robot before reaching its goal. The avoidance of local minima has been an active research topic in potential field path planning. As one of the powerful techniques for escaping local minima, simulated annealing has been applied to local and global path planning.

The avoidance of local minimum has been an effective research topic in the APF based path finding. However, the previous solutions are limited to simple formations of obstacles or available for known environments. But Lee and Park designed a virtual obstacle concept is proposed as an idea to escape a local minimum. The imaginary obstacle is located around local minimum point to force the robot from the point. This technique is useful for the local pathfinding in unknown areas. The sensor based discrete modeling method is also planned for the simple modeling of a mobile robot with range sensors. This modeling is easy and good because it is designed for a real-time path planning [19].

Vector field histogram in robotics, Vector Field Histogram (VFH) is a real time motion planning algorithm proposed by [13]. The VFH utilizes a statistical representation of the robot s environment through the so-called histogram grid, and therefore places great emphasis on dealing with uncertainty from sensor and modeling errors. Unlike other obstacle avoidance algorithms, VFH takes into account the dynamics and shape of the robot, and returns steering commands specific to the platform. While considered a local path planner, i.e., not designed for global path optimality, the VFH has been shown to produce near optimal paths.

The original VFH algorithm was based on previous work on Virtual Force Field, a local spath-planning algorithm. VFH was updated and renamed VFH+ [20]. The approach was updated again and was renamed VFH* [21]. VFH is currently one of the most popular local planners used in mobile robotics, competing with the later developed dynamic window approach. Many robotic development tools and simulation environments contain built-in support for the VFH.

At the center of the VFH algorithm is the use of statistical representation of obstacles, through histogram grids. Such representation is well suited for inaccurate sensor data, and accommodates fusion of multiple sensor readings.

In VFH*, the algorithm verifies the steering command produced by using the A* search algorithm to minimize the cost and heuristic functions. While simple in practice, it has been shown in experimental results that this look-ahead verification can successfully deal with problematic situations that the original VFH and VFH+ cannot handle (the resulting trajectory is fast and smooth, with no significant slowdown in presence of obstacles).

Dijkstras algorithm Dijkstras algorithm is an algorithm for finding the shortest paths between nodes in a graph, which may represent, for example, road networks. It was conceived by computer scientist Edsger W. Dijkstra in 1956 and published three years later [22].

The algorithm exists in many variants. Dijkstras original variant found the shortest path between two nodes, but a more common variant fixes a single node as the source node and finds shortest paths from the source to all other nodes in the graph, producing a shortest-path tree.

For a given source node in the graph, the algorithm finds the shortest path between that node and every other [23]. It can also be used for finding the shortest paths from a single node to a single destination node by stopping the algorithm once the shortest path to the destination node has been determined. For example, if the nodes of the graph show cities and edge path costs show driving distances between pairs of cities connected by a direct road, Dijkstras algorithm

can be used to find the shortest way between one city and all other cities. As a result, the shortest path algorithm is generally used in network routing protocols, most notably IS-IS and Open Shortest Path First (OSPF). It is also employed as a subroutine in other algorithms such as Johnsons.

A* algorithm A researcher Nils Nilsson was trying to improve the pathfinding done by a robot in 1968, the robot that could navigate in a room with obstacles. This path-finding algorithm which is called A1, was faster than the best method, Dijkstras algorithm, for finding shortest way in graphs. Bertram Raphael did some significant improvements on this algorithm, naming the revision A2. Then Peter E. Hart designed an argument that established A2, with only small changes, to be the best possible algorithm for finding the shortest paths. Rapheal, Hart and Nilsson developed a proof that the revised A2 algorithm was perfect for finding shortest ways under certain well-defined conditions.

This algorithm is generally used in pathfinding and graph travelsal, the process of plotting and efficiently traversable way between multiple points, named nodes. Noted for its performance and accuracy, it enjoys widespread use. On the other hand, in practical travel-routing designs, it is generally less performed by algorithms which can pre-process the graph to attain better performance [24], even though other works has found A* to be superior to other ways [25].

Hart and others (1968) first explained the algorithm. It is an extension of Edger Dijkstras 1959 algorithm. A* shows better performance by using heuristics to guide its search.

D* algorithm D* (pronounced D star) is any one of the following three related additional search algorithms:

- The original D*, [26], is an informed incremental search algorithm.
- Focused D* is an informed incremental heuristic search algorithm by [26] that combines ideas of A* [27] and the original D*. Focused D* resulted from a further development of the original D*.
- D* Lite is an incremental heuristic search algorithm by [28] that builds on LPA*, an incremental heuristic search algorithm that combines ideas of A* and Dynamic SWSF-FP [29].

All three algorithms solve the same assumption-based path finding problems, including planning with the freespace assumption, where a robot has to navigate to given coordinates in unknown terrain. It makes expectations about the unknown part of the terrain (for example: that it doesnt contain obstacles) and finds a shortest way from its actual coordinates to the goal coordinates under these assumptions [30]. The robot then follows the way. When it observes new map information (such as previously unknown obstacles), it adds the information to its map and, if necessary, plans a new shortest way from its current coordinates to the given goal coordinates. It repeats the process until it reaches the goal coordinates or determines that the goal coordinates cannot be reached. When traversing unknown terrain, new obstacles may be discovered frequently, so this planning needs to be fast. Incremental (heuristic) search algorithms speed up searches for sequences of similar search problems by using experience with the

previous problems to speed up the search for the current one. Assuming the goal coordinates do not change, all three search algorithms are more capable than repeated A* searches. D* and its variants have been actively used for mobile robot and autonomous vehicle navigation. Current systems are typically based D* Lite rather than the original D* or Focused D*. In fact, even Stentzs lab uses D* Lite rather than D* in some implementations [28]. Such navigation designs include a prototype system tested on the Mars rovers opportunity and spirit and the navigation system of the winning entry in the DARPA Urban Challenge, both developed at Carnegie Mellon University.

The original D* was submitted by Anthony Stentz in 1994. The name D* comes from the term Dynamic A*, because the algorithm behaves like A* except that the arc costs can change as the algorithm works.

Rapidly exploring random tree A rapidly exploring random tree (RRT) is an algorithm designed to efficiently search nonconvex, high-dimensional spaces by randomly building a space-filling tree. The tree is constructed incrementally from samples drawn randomly from the search space and is inherently biased to grow towards large unsearched areas of the problem. RRTs were submitted by [31]. They easily handle problems with obstacles and differential constraints (nonholonomic and kinodynamic) and have been widely used in autonomous robotic path planning [32].

RRTs can be viewed as a technique to generate open loop trajectories for nonlinear systems with state constraints. An RRT can also be considered as a Monte-Carlo method to bias search into the largest Voronoi regions of a graph in a configuration space. Some variations can even be considered stochastic fractals.

In fact, sampling based algorithms, RRT being one of them, generally gained much popularity because of their less computational complexity and their ability to find paths without specific information of the obstacles in the configuration space. Instead, they make obstacle-free trajectories leading to a roadmap between sampled configurations in the obstacle free region. Though RRT has the advantage in providing the quickest first path and also probabilistic completeness but it does not ensure asymptotic optimality. Eventually, [33] overcame this problem of asymptotic optimality by introducing the extended version of RRT known as RRT*. Rapidly Exploring Random Tree Star made it possible to find an optimal solution, but this has been proven to take infinite time [33]. RRT*-Smart, which is used to solve the infinite time constraint and provides a significantly faster convergence rate, which has proven to be beneficial both in terms of cost and time compared to RRT* in various environments [34]. This is achieved through the introduction of two new concepts of intelligent displacement and path optimization. This study presents a dynamic bias factor diagram for RRT*-Smart, which leads to more efficient results, and also makes the algorithm parameter, bias factor, independent of the environment, resulting in a balanced trade-off between exploration speed and bias speed.

3 Overview of Quadrotor Technology

To determine a suitable quadcopter to use for this project, a variety of products from various manufacturers were studied. The research requires asmall quadrotor. Specifically, we have chosen the Iris+ aircraft as the quadrotor used and evaluated in this thesis. This choice is mainly due to the posssibility of easy simulate this aerial platform. The product that was examined are detailed rapresenter in this chapter.

3.1 Quadcopter

The modeling and simulation of the Iris+ quadrotor must be performed to determine its flight characteristics and designing its attitude controllers. The derivations of the dynamics equations for a quadrotor model were well elaborated by Boudallah in [2] , Beard in [3] , Corke in [4], Sidea in [5] and Bresciani in [6]. However, the modelling approaches in these researches were only applicable for a limited set of quadrotor configurations. The IRIS quadrotor featured a cross-style configuration and therefore, modifications to the dynamics equations of the plus-style configuration quadrotor were required to obtain the dynamics equations for a quadrotor in cross-style configuration. An approach to model a quadrotor in the cross-style configuration was elaborated by Partovi in [7] for the X650 quadrotor. This approach was suitable and was adopted for the modelling of the Iris+ quadrotor.

3.1.1 IRIS

The IRIS quadcopter was made available by the project adviser at the start of the project. The IRIS, a product of 3DRobotics, is equipped with a Pixhawk autopilot system, which led to the decision to use an on-board processing platform with a Raspberry Pi as the two can communicate MAVlink. MAVlink is a communication standard used to send commands and information to and from UAV autopilot boards. Developers can implement the MAVlink standard in various ways. One of the possible solution is implementation MAVProxy for use on Raspberry Pi. The

Iris+ was used throughout the entirety of this project, however other potential options were explored. While researching other possible flight platforms, the Iris+ was prepared for flight tests by performing calibrations and constructing a mount for the on-board system.
The IRIS was designated to be the flight platform for a camera payload. Mission Planner software was installed on a ground station laptop to collect and process telemetry data and perform some calibration procedures.
As mensioned below it can sum it up that Iris+ is the quadrotor vehicle that has been used for this research. The vehicle is manufactured by 3D Robotics (3DR), as show in Figure 3.2. The Iris+ has a mid-sized X shape body. The four 9.5 inch diameter propellers are mounted at the end of each leg. The propellers are designed as self-tightening propellers, where the mounting threads of the propellers get tightened by the motor torque during the flight. The Iris+ has good maneuver ability, and it can generate a lot of thrust to carry some external payload, such as a self-stabilizing gimbal and a GoPro camera. However, the gimbal and camera are not installed for weight-saving reasons, which allows our IRIS additional maneuver margin. The battery compartment of the IRIS can hold a 5100 mAh Li-On battery, which gives the Iris+ 16-22 minutes of flight. 3DR radio is a wireless telemetry communication module, which comes with the Iris+ package. This radio is small but it has a good transmittin grange of more than one mile. It is operated at a 57600 band rate, both up and downlink. However, based on some of the tests, the 3DR radio is only reliable for sending data with a rate up to about twenty Hz. Significant delay and data drop outs were observed when trying to send data to the Iris+ at a faster rate.

Figure 3.1: The standard IRIS drone kit.

3.1.2 Iris Features

- Multiple control options provide redundancy and flexibility: RC, computer, phone, and tablet.
- Built-in data radio for real-time mission monitoring, data-logging, and control.

- Powerful cross-platform ground station/mission planning and analysis software that runs on Windows, OS X and Linux, providing simple point-and-click programming and configuration.
- Mobile apps allow intuitive draw a path mission planning.
- Easy mounting system integrated in the arms provides painless mounting for future accessories.
- Camera options include a live video link with programmable on-screen display, and will soon support a fully integrated stabilized camera gimbal with autopilot control.
- GoPro compatible camera mount.
- Available with a 9-channel RC transmitter pre-programmed for the most popular flight modes.
- Pre-programmed GPS waypoints allow for professional-grade mission capabilities, such as: mapping, scripted cinematography, scientific research, and other applications where repeatable flight plans are required.
- Robust arms and feet produced from Zytel Nylon for the ultimate in wear, abrasion and impact resistance over a wide temperature range. They are easily and inexpensively replaced if required.
- Auto takeoff and landing along with return-to-launchpoint command at the press of a button or under programmable failsafe conditions.
- Follow-me function for the ultimate "selfies". In this mode, IRIS will follow (at an adjustable distance) any ground station device equipped with a GPS antenna and one of our 3DR telemetry/control radios.
- Geo Fencing provides a virtual box to keep your drone within a user-selectable space.
- Failsafe programming options bring peace of mind in the event of lost control signal, GPS or low battery conditions.
- External micro-USB port.
- Multicolor LED status indicator.
- Buzzer for audible status and warning messages.
- Safety switch adds a second level of protection against inadvertent start-ups.
- Open source flight code, ground station software, and electronics are all freely distributed under standard open source licenses. This means that I' capabilities are always improving and expanding with a simple firmware update!

3.1.3 IRIS Specifications

- Motor to motor dimension: 550mm
- Height: 100mm
- Weight (with battery): 1282 grams
- Average flight time: 10-15 minutes
- 400 g (.8 lb) payload capacity
- Battery: 3-cell 11.1 V 3.5 Ah lithium polymer with XT-60 type connector. Weight: 262 grams
- Propellers: (2) 10 x 4.7 normal-rotation, (2) 10 x 4.7 reverse-rotation
- Motors: AC 2830, 850 kV
- Telemetry/Control radios available in 915mHz or 433mHz
- Next generation 32-bit Pixhawk autopilot system with Cortex M4 processor
- uBlox GPS with integrated magnetometer

3.2 Basic concepts

The quadrotor is very well modeled with a four rotors in a cross configuration. This cross structure is quite thin and light, however it shows robustness by linking mechanically the motors. The front and the rear propellers rotate counter-clockwise, while the left and the right ones turn clockwise. This configuration of opposite pairs directions removes the need for a tail rotor (needed instead in the standard helicopter structure). Figure 3.2 shows the structure model in hovering condition, where all the propellers have the same speed.

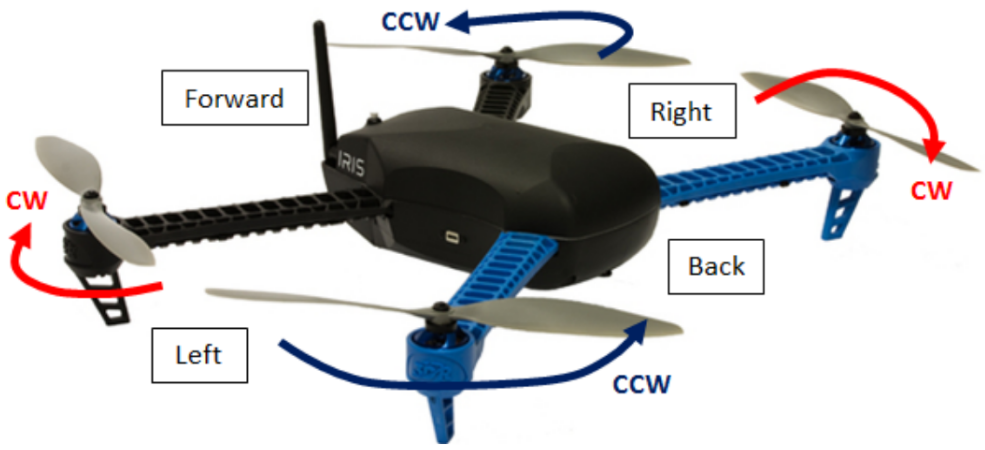


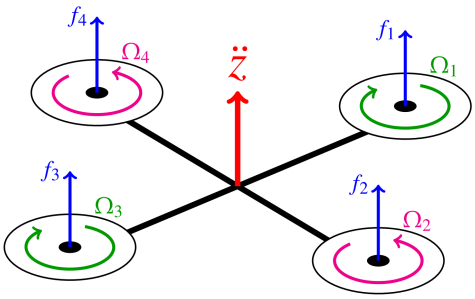
Figure 3.2: Illustration of Quadrotor Airframe in Cross Configuration.

3.3 Advantages and Disadvantages of Quadrotor/Multi-rotor Technology

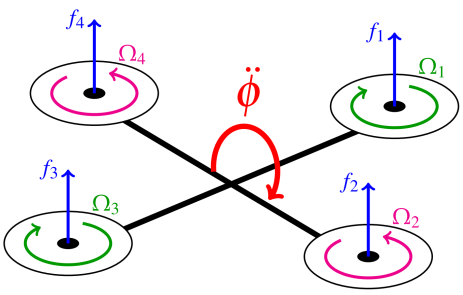
With the overview of the multi-rotor technology, a quick analysis of its advantages and disadvantages was summarized in Table 3.1.

Table 3.1: "Advantages and Disadvantages of Quadrotor."

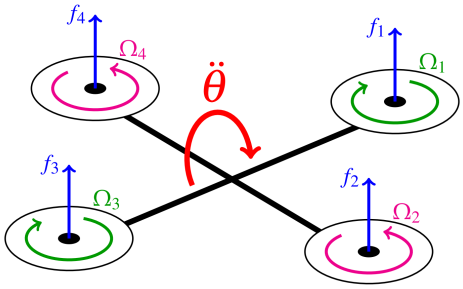
Advantage	Disadvantages
VTOL and Hovering Capabilities Unlike fixed wing UAV, the unique flight mechanism of the quadrotor allows it to perform VTOL and hovering in flight. These capabilities eliminate the need of a landing strip or a launch and recovery system.	**Short Battery Life** The battery life of most quadrotors is approximately 20 minutes and is constrained by the charge storage capability of Lithium battery; power density is the fundamental constrain. The short battery life reduces the mission duration of the quadrotor.
Agile Maneuverability By varying the rotational speed of its propellers, the quadrotor is able to generate thrust and moments to perform sharp turns during flight and hover in mid-flight. A fixed wing UAV in contrast makes turns with a larger turning radius.	**Under-actuated System** A quadrotor is an under-actuated system, where multiple actuators are used to perform 6 linear and angular control actions. Therefore, if the symmetry of the actuators action is damaged, it would either no longer be able to perform a maneuver or its control authority might be compromised.
Mechanically Simple Quadrotor uses propeller bladeswith fixedsymmetricalpitch propeller blades and consists of lesser mechanical components compared to conventional helicopters. Therefore, quadrotors are easy to maintain and cheaper to manufacture.	**Low Payload Capability** The payload limit of a medium sized quadrotor ($\sim$1.5kg) is typically between 0.8 to 1 lbs. Therefore, the equipment or load that can be carried by quadrotors is not substantial.



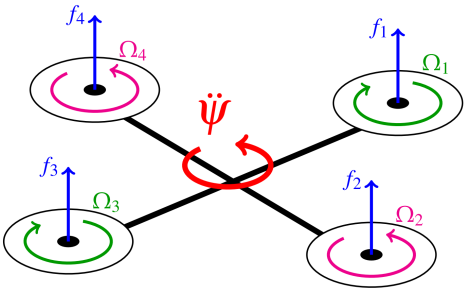
(a) Throttle movement.



(b) Roll movement.



(c) Pitch movement.



(d) Yaw movement.

3.4 Configuration Flight Mechanics

Based on the motors arrange described in Table 3.2, it is just needed to change the speed of one of the pair of motors as to cause motion in six degrees of freedom (DOF). This is the reason that allows the quadcopter to move in six DOF and be controlled just with four inputs

Table 3.2: "Motors configuration".

Motors	**Throttle**				**Roll**				**Pitch**				**Yaw**			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4

Throttle, Roll, Pitch and Yaw

Throttle Controls the motion of your drone up and down by speeding up or slowing down all of the propellers.

Roll Tilts the quadcopter to the left or the right by speeding up the rotors on one side of the quadcopter and slowing them down on the other side. By doing so, one side of the drone will sag, or tilt downward, causing the drone to strafe to the left or the right.

Pitch Tilts the quadcopter forward and backward in the same manner as rolling. By adjusting the pitch, the drone will sag down in the front causing it to go forward, or sag in the back causing it to go backwards.

Yaw Rotates the nose of your aircraft left to right. Quadcopter propellers do not tall spin in the same direction. If they did, the centrifugal force would cause them to just spin out of control. In order to combat that, diagonally opposing propellers spin in the same direction. This

picture details the directions in which propellers spin. To rotate your quadcopter the rotors that spin in the same direction will speed up to rotate the air craft to the left or the right.

4 Quadrotor model

When facing the quadrotors model identification problem, most researches consider dynamic models, as the kinematic model of a quadcopter simply represents the movement of a six degrees of freedom (DOF) point in the space. Thus, the main differences in the literature regarding the model correspond with the set of dynamics modelled, or the reference frame considered. During the development of this project, effects like the motor dynamics, or the asymmetry of lift, were not considered. This thesis will make use of the most common quadrotor dynamic model in the research field.

This chapter presents an overview of the different parts of the drone system used for the simulations of this research.

4.1 Reference systems

There are two coordinate systems, the local north, east, down (NED) frame E=xe, ye, ze for translation and the body frame B=xb, yb, zb for rotation. The quadrotor structure and the two coordinate systems are shown in Figure 4.1, where x,y,z represent the translational displacement in each direction, ϕ,θ,ψ (the Euler angles corresponding to roll, pitch and yaw), respectively [35] [36].

4.1.1 Attitude representation

One of the main problem in modelling and controlling a 3D system has always been the correct representation of the attitude of an object with respect to an inertial frame and a body fixed frame. This is actually a key issue in this work since the errors and so the control variables are computed based on it and the singularities of the different representations strongly influence the stability and robustness properties of the controlled system.

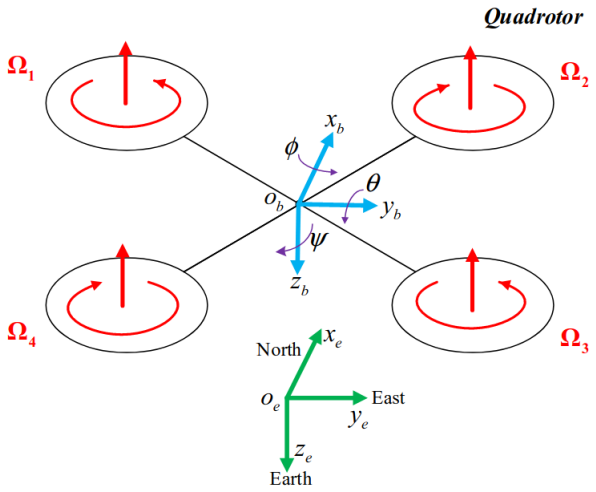


Figure 4.1: Schematic view of the quadrotor unmanned aerial vehicles.

4.1.2 Understanding Euler Angles

Euler angles provide a way to represent the 3D orientation of an object using a combination of three rotations about different axes. For convenience, used multiple coordinate frames to describe the orientation of the sensor, including the "inertial frame," the "vehicle-1 frame," the "vehicle-2 frame," and the "body frame." The inertial frame axes are Earth-fixed, and the body frame axes are aligned with the sensor. The vehicle-1 and vehicle-2 are intermediary frames used for convenience when illustrating the sequence of operations that take us from the inertial frame to the body frame of the sensor. It may seem unnecessarily complicated to use four different coordinate frames to describe the orientation of the sensor, but the motivation for doing, so will become clear as it was proceed.

For clarity, this application note assumes that the sensor is mounted to an aircraft. All examples and figures are given showing the changing orientation of the aircraft.

The Inertial Frame

With inertial reference system or Earth axes, it is named a coordinate system of three fixed axes that works as a fixed base for the representation of the position of a considered body or moving reference system. The origin of the system could be set everywhere, the intersection of the equator with the prime meridian and the mean sea level for example, it is completely arbitrary but once chosen it cannot vary. The displacement along one or more set of orthogonal axes attached to the origin describes the position of anything in this reference system. It is convenient to align the axes of the reference frame with the compass, one is aligned with the axis labelled North, one with the axis labelled East and the last with the normal to the surface generated by the previous two, pointing to the centre of the Earth and labelled Down. These three axes are mutually perpendicular by construction and when referring to them in the order N, E, D form a right-handed coordinate system.

The "Inertial frame" is an Earth-fixed set of axes that is used as an unmoving reference. Robotics' sensors use a common aeronautical inertial frame where the *x-axis* points north, the

y-axis points east, and the *z-axis* points up as shown below. It will call this a North-East-Down (NED) reference frame. Note that because the *z-axis* points down, altitude above ground is actually a negative quantity.

The sequence of rotations used to represent a given orientation is first yaw, then pitch, and finally roll.

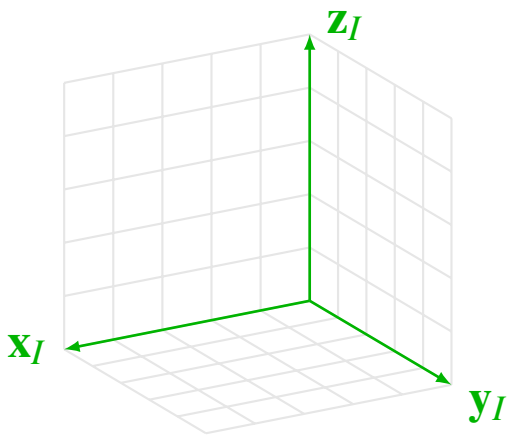


Figure 4.2: The Inertial Frame.

The Vehicle-1 Frame (Yaw Rotation)

As shown in Figure 4.2, *yaw* represents rotation about the inertial-frame $z-axis$ by an angle ψ. The *yaw* rotation produces a new coordinate frame where the $z-axis$ is aligned with the inertial frame and the x and y axes are rotated by the *yaw* angle ψ. We call this new coordinate frame the Vehicle-1 frame. The orientation of the vehicle-1 frame after *yaw* rotation is show in Figure 4.3. The Vehicle-1 frame axes are colored blue, while the inertial frame axes are green.

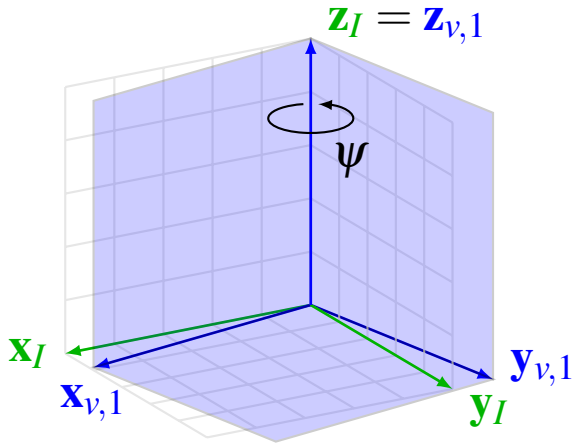


Figure 4.3: Yaw rotation into the Vehicle-1 Frame.

The transformation from F_v to F_{v1} is given by

$$p^{v1} = R_v^{v1}(\psi)p_v, \tag{4.1}$$

where rotation of a vector from the Inertial frame to the Vehicle-1 frame can be performed by multiplying the vector by the rotation matrix.

$$R_v^{v1}(\psi) = \begin{bmatrix} cos(\psi) & sin(\psi) & 0 \\ -sin(\psi) & cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

The Vehicle-2 Frame (Yaw and Pitch Rotation)

Pitch represents rotation about the Vehicle-1 frame $y-axis$ by an angle θ as shown in Figure 4.4. For clarity, the inertial-frame axes are not shown. The Vehicle-1 frame axes are shown in gray, and the Vehicle-2 axes are shown in red. It is important to note that pitch is NOT rotation about the Inertial-frame $y-axis$.

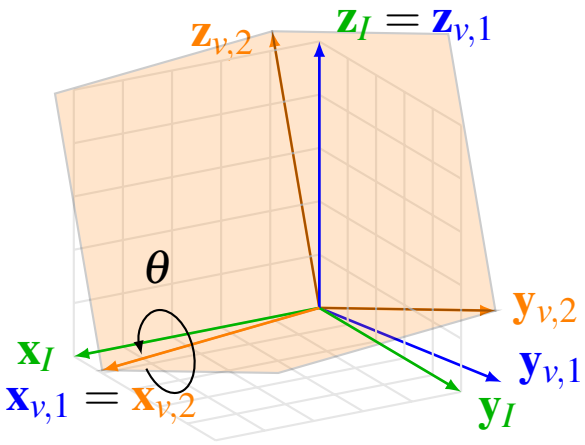


Figure 4.4: The Vehicle-2 Frame (Yaw and Pitch Rotation Applied).

The transformation from F_v^1 to F_{v2} is given by

$$p^{v2} = R_{v1}^{v2}(\theta)p_v^1, \tag{4.3}$$

The rotation matrix for moving from the inertial frame to the vehicle-2 frame consists simply of the yaw matrix multiplied by the pitch matrix:

$$R(\theta,\psi) = R(\theta)R(\psi) \tag{4.4}$$

where the rotation matrix for moving from the vehicle-1 frame to the vehicle-2 frame is given by

$$R_{v1}^{v2}(\theta) = \begin{bmatrix} cos(\theta) & 0 & -sin(\theta) \\ 0 & 1 & 0 \\ sin(\theta) & 0 & cos(\theta) \end{bmatrix} \tag{4.5}$$

The Body Frame (Yaw, Pitch, and Roll Rotation)

The body frame is the coordinate system that is aligned with the body of the sensor. On an aircraft, the body frame $x-axis$ typically points out the nose, the $y-axis$ points out the right side of the fuselage, and the $z-axis$ points out the bottom of the fuselage.

The body frame is obtained by performing a rotation by the angle ϕ around the vehicle-2 frame $x-axis$ as shown in Figure 4.5. For clarity, the inertial frame and vehicle-1 frame axes are not shown. The vehicle-2 frame axes are shown in gray, while the body-frame axes are shown in red.

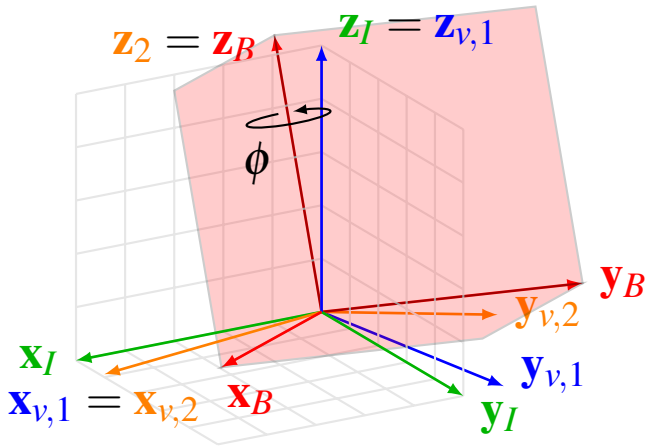


Figure 4.5: The Body Frame (Yaw, Pitch, and Roll applied).

The transformation from F_v^2 to F_b is given by

$$p^b = R_{v2}^b(\phi)p_v^2, \tag{4.6}$$

where the rotation matrix for moving from the Vehicle-2 frame to the body frame is given by

They are represented by:

$$R_{v2}^b(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\phi) & sin(\phi) \\ 0 & -sin(\phi) & cos(\phi) \end{bmatrix} \tag{4.7}$$

The complete rotation matrix for moving from the inertial frame to the body frame is given by

$$R(\phi,\theta,\psi) = R(\phi)R(\theta)R(\psi) \tag{4.8}$$

Performing the multiplication, and letting c represent cos and s represent sin, the complete rotation from the inertial frame to the body frame is given by

which results in:

$$R(\phi,\theta,\psi) = \begin{bmatrix} cos(\psi)cos(\theta) & sin(\psi)cos(\theta) & -sin(\theta) \\ cos(\psi)sin(\theta)sin(\phi) - sin(\psi)cos(\phi) & sin(\psi)sin(\theta)sin(\phi) + cos(\psi)cos(\phi) & cos(\theta)sin(\phi) \\ cos(\psi)sin(\theta)cos(\phi) + sin(\psi)sin(\phi) & sin(\psi)sin(\theta)sin(\phi) - cos(\psi)sin(\phi) & cos(\theta)cos(\phi) \end{bmatrix} \tag{4.9}$$

The rotation matrix for moving the opposite direction - from the body frame to the inertial frame - is given by

$$R(\phi,\theta,\psi) = R(\phi)R(\theta)R(\psi) \tag{4.10}$$

Performing the multiplication, the complete rotation from the body frame to the inertial frame is given by

$$R(\phi,\theta,\psi) = \begin{bmatrix} cos(\psi)cos(\theta) & cos(\psi)sin(\theta)sin(\phi) - sin(\psi)cos(\phi) & cos(\psi)sin(\theta)cos(\phi) + sin(\psi)sin(\theta) \\ sin(\psi)cos(\theta) & sin(\psi)sin(\theta)sin(\phi) + cos(\psi)cos(\phi) & sin(\psi)sin(\theta)sin(\phi) + cos(\psi)cos(\phi) \\ -sin(\theta) & cos(\theta)sin(\phi) & cos(\theta)cos(\phi) \end{bmatrix} \tag{4.11}$$

Note that all this does is reverse the order of operations and reverse the direction of rotation.

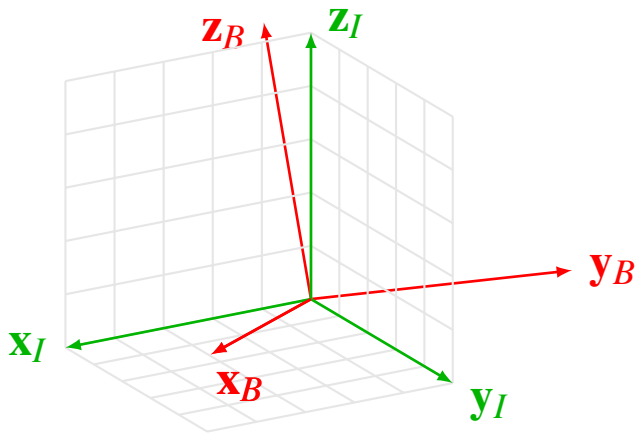


Figure 4.6: The rotation axis for moving from the Body Frame to the Inertial Frame.

4.2 Equations of Motion

Translational Kinematics

The state variables for velocity are in the body frame but the state variables for position are in the global frame. Therefore, it is necessary to define a rotation matrix to transform variables between the coordinate systems. The transformation between the global and body coordinate frames is described below.

$$x^b = R_G^b X^G = R(\phi)R(\theta)R(\psi)X^G \tag{4.12}$$

$$R_G^b = \begin{bmatrix} cos(\psi)cos(\theta) & sin(\psi)cos(\theta) & -sin(\theta) \\ cos(\psi)sin(\theta)sin(\phi) - sin(\psi)cos(\phi) & sin(\psi)sin(\theta)sin(\phi) + cos(\psi)cos(\phi) & cos(\theta)sin(\phi) \\ cos(\psi)sin(\theta)cos(\phi) + sin(\psi)sin(\phi) & sin(\psi)sin(\theta)cos(\phi) - cos(\psi)sin(\phi) & cos(\theta)cos(\phi) \end{bmatrix} \tag{4.13}$$

$$x^G = R_G^b X^G = R(\phi)^T R(\theta)^T R(\psi)^T X^G \tag{4.14}$$

$$R_b^G = \begin{bmatrix} cos(\psi)cos(\theta) & cos(\psi)sin(\theta)sin(\phi) - sin(\psi)cos(\phi) & cos(\psi)sin(\theta)cos(\phi) + sin(\psi)sin(\phi) \\ sin(\psi)cos(\theta) & sin(\psi)sin(\theta)sin(\phi) + cos(\psi)cos(\phi) & sin(\psi)sin(\theta)cos(\phi) - cos(\psi)sin(\phi) \\ -sin(\theta) & cos(\theta)sin(\phi) & cos(\theta)cos(\phi) \end{bmatrix} \tag{4.15}$$

Rotational Kinematics

Since the angular rates are defined in the body frame and the Euler angles are defined in intermediate coordinate frames, we can use the rotation matrix derived above to determine the relationship between the angular rates and the time derivatives of the Euler angles as shown below. The angular velocities are vectors pointing along each axis of rotation and are not equal to the time derivative of the Euler angles. The derivation below assumes that the time derivative of each Euler rate is small. See this page for more detailed information about Euler angles.

$$\omega = \begin{bmatrix} p \\ q \\ r \end{bmatrix} = R(\phi, \theta) \begin{bmatrix} 0 \\ 0 \\ \dot{\psi} \end{bmatrix} + R(\phi) \begin{bmatrix} 0 \\ \dot{\theta} \\ 0 \end{bmatrix} + \begin{bmatrix} \dot{\phi} \\ 0 \\ 0 \end{bmatrix} \tag{4.16}$$

$$S^T = \begin{bmatrix} 1 & sin(\phi)tan(\theta) & cos(\phi)tan(\theta) \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & \frac{sin(\phi)}{cos(\theta)} & \frac{cos(\phi)}{cos(\theta)} \end{bmatrix} \tag{4.17}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & sin(\phi)tan(\theta) & cos(\phi)tan(\theta) \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & \frac{sin(\phi)}{cos(\theta)} & \frac{cos(\phi)}{cos(\theta)} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{4.18}$$

If the Euler angles are assumed to be small (near 0), then the S matrix becomes the identity matrix and the angular rates are roughly equal to the time derivative of the Euler angles.

Translational Dynamics

The linear equations of motion are defined in the global reference frame. The acceleration of the quadrotor in the global frame is equal to the sum of force of gravity, the thrust force of the motors, and linear friction force resulting in drag. The thrust vector in the body frame is transformed into the global frame using the rotation matrix defined previously. The resultant force generated by the four motors in the global coordinate frame is defined below. Additionally, the resultant global drag force along each translational axis is defined below as the product of drag coefficients and linear velocities in each direction in the global frame. This is a simple model of fluid friction. More complex modeling would involve separate friction constants for each axis as well as modeling the friction in the body frame instead of the inertial frame.

m = mass of the quadrotor (kg)
g = acceleration due to gravity (m/s^2)
$\ddot{X}^G$ = accelerationin the global coordinate frame (m/s^2)
$\dot{X}^G$ = velocity in the global coordinate frame (m/s)
F_g = force of gravity in the global coordinate frame (N)
F_d = force of drag in the global coordinate frame (N)
F_T^G = total force of thrust in the global coordinate frame (N)
F_T^b = total force of thrust in the body coordinate frame (N)
R_b^G = body to global coordinate frame rotation matrix
F_i = thrust of each motor (N)
ω = motor angular velocity $(radians/sec)$
K_T = thrust coefficient $K_{dx} = K_{dy} = K_{dz}$ = drag coefficient

$$m\ddot{X}^G = F_g - F_T^G - F_d \tag{4.19}$$

$$\ddot{X}^G = \begin{bmatrix} \ddot{X}^G \\ \ddot{Y}^G \\ \ddot{Z}^G \end{bmatrix} \text{ and } F_g = \begin{bmatrix} 0 \\ 0 \\ mg \end{bmatrix} \tag{4.20}$$

$$F_d = \begin{bmatrix} K_{dx} & 0 & 0) \\ 0 & K_{dy} & 0 \\ 0 & 0 & K_{dz} \end{bmatrix} \begin{bmatrix} \dot{X}^G \\ \dot{Y}^G \\ \dot{Z}^G \end{bmatrix} \tag{4.21}$$

Rotational Dynamics

The rotational equations of motion are defined in the body frame so that the rotations can be computed about the quadrotors center and not the center of the global coordinate frame. Since we assumed the quadrotor to be symmetrical, the quadrotor moment of inertia matrix is symmetrical. These equations are comprised of three terms including aerodynamic effects and motor torques. First, the gyroscopic effect resulting from the rigid body rotation. Second, the motors produce a rolling, pitching, and yawing torque. The torque due to drag is shown to be proportional to the product of a drag constant and the square of the rotors angular velocity. This drag equation assumes that the quadrotor is operating in stable flight and that the propellers are

maintaining a constant thrust and not accelerating. This assumption results in the torque about the global z axis being equal to the torque due to drag. Third, the cross product which describes the gyroscopic effect resulting from the propeller rotation coupled with the body rotation. This is due to the fact that the rotors axis of rotation is itself moving with the angular velocity of the frame.

ω = rate of angular velocities $(radians/s)$
$\dot{\omega}$ = rate of angular accelerations $(radians/s)$
J_b = quadrotor body moment of inertia $(kg{\cdot}m^2)$
J_m = motor moment of inertia $(kg{\cdot}m^2)$
J_r = rotor moment of inertia $(kg{\cdot}m^2)$
J_x = quadrotor moment of inertia in x_b $(kg{\cdot}m^2)$
J_y = quadrotor moment of inertia in y_b $(kg{\cdot}m^2)$
J_z = quadrotor moment of inertia in z_b $(kg{\cdot}m^2)$
τ_m = body torques from motors $(N{\cdot}m)$
τ_g = gyroscopic effect torques $(N{\cdot}m)$
$\tau_{\phi,\theta,\psi}$ = body torques from motors $(N{\cdot}m)$
τ_d = drag torque $(N{\cdot}m)$
K_d = drag torque proportionality constant
R = propellor radius (m)
A = propellor cross section (m^2)
C_D = dimensionless coefficient
ρ = air coefficient (kg/m^3)
l = length of arm (m)

$$J_b\dot{\omega} = \tau_m - \tau_g - (\omega \times J_b\omega) \tag{4.22}$$

$$J_b = \begin{bmatrix} J_x & 0 & 0) \\ 0 & J_y & 0 \\ 0 & 0 & J_z \end{bmatrix};\ \dot{\omega} = \begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix}\ (\omega \times J_b\omega) = \begin{bmatrix} \dot{\theta}\dot{\psi}(J_z - J_y) \\ \dot{\psi}\dot{\phi}(J_x - J_z) \\ \dot{\theta}\dot{\phi}(J_y - J_z) \end{bmatrix} \tag{4.23}$$

$$J_m\dot{\omega} = \tau_m - \tau_\psi;\ \text{At the hover, } \dot{\omega} = 0;\ \text{so } \tau_m = \tau_D \tag{4.24}$$

$$\tau_D = \frac{1}{2}R_\rho C_D A(\omega_R)^2 = K_d\omega^2 \tag{4.25}$$

$$\tau_\psi = \tau_D = (-1)^{i+1}K_d\omega_i^2 = K_d(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2) \tag{4.26}$$

$$(\tau_{\psi,\theta}) = \sum r \times T; \tag{4.27}$$

$$(\tau_\phi) = lK_T(\omega_4^2 - \omega_2^2) \tag{4.28}$$

$$\tau_\phi = lK_T(\omega_4^2 - \omega_2^2) \tag{4.29}$$

$$\tau_\theta = lK_T(\omega_1^2 - \omega_3^2) \tag{4.30}$$

$$\tau_m = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} \begin{bmatrix} lK_T(\omega_4^2 - \omega_2^2) \\ lK_T(\omega_1^2 - \omega_3^2) \\ K_d(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2) \end{bmatrix} \tag{4.31}$$

$$\tau_g = \omega \times Gz \sum_{i=1}^{4} J_r \omega_i \tag{4.32}$$

$$\tau_g = \begin{bmatrix} \dot{\theta} \\ -\dot{\phi} \\ 0 \end{bmatrix} J_r \sum_{i=1}^{4} \omega_i = \begin{bmatrix} J_r\dot{\theta}(\omega_1 - \omega_2 + \omega_3 - \omega_4) \\ -J_r\dot{\phi}(\omega_1 - \omega_2 + \omega_3 - \omega_4) \\ 0 \end{bmatrix} \tag{4.33}$$

Equations of Motion

Solving for the quadrotors global translational acceleration and body frame angular accelerations leads to the following equations. It is used the global frame for translational position since that is the same coordinate frame as our GPS sensor. Similarly, the body frame is chosen for attitude since the IMU (accelerometer, magnetometer, and gyroscope) also make measurements in the body frame. These equations are both non-linear and highly coupled. They can be simplified using reasonable assumptions in order to design and implement stable control systems.

$$\begin{bmatrix} \dot{X}^G \\ \dot{Y}^G \\ \dot{Z}^G \end{bmatrix} = \begin{bmatrix} cos(\psi)cos(\theta) \\ cos(\psi)sin(\theta)sin(\phi) - sin(\psi)cos(\phi) \\ cos(\psi)sin(\theta)cos(\phi) + sin(\psi)sin(\phi) \end{bmatrix}$$
$$\begin{bmatrix} sin(\psi)cos(\theta) & -sin(\theta) \\ sin(\psi)sin(\theta)sin(\phi) + cos(\psi)cos(\phi) & cos(\theta)sin(\phi) \\ sin(\psi)sin(\theta)cos(\phi) - cos(\psi)sin(\phi) & cos(\theta)cos(\phi) \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{4.34}$$

$$\begin{bmatrix} \ddot{X}^G \\ \ddot{Y}^G \\ \ddot{Z}^G \end{bmatrix} = \begin{bmatrix} \frac{1}{m}(-[cos(\phi)cos(\psi)sin(\theta) + sin(\phi)sin(\psi)]F_T^b - K_{dx}\dot{X}^G \\ \frac{1}{m}(-[cos(\phi)sin(\psi)sin(\theta) - cos(\psi)sin(\phi)]F_T^b - K_{dy}\dot{Y}^G \\ \frac{1}{m}(-[cos(\phi)cos(\theta)]F_T^b - K_{dz}\dot{Z}^G) + g \end{bmatrix} \tag{4.35}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & sin(\phi)tan(\theta) & cos(\phi)tan(\theta) \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & \frac{sin(\phi)}{cos(\theta)} & \frac{cos(\phi)}{cos(\theta)} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{4.36}$$

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = \begin{bmatrix} \frac{1}{J_x}[(J_y - J_z)qr - J_rq(\omega_1 - \omega_2 + \omega_3 - \omega_4) + lK_T(\omega_4^2 - \omega_2^2)] \\ \frac{1}{J_y}[(J_z - J_x)pr - J_rp(\omega_1 - \omega_2 + \omega_3 - \omega_4) + lK_T(\omega_1^2 - \omega_3^2)] \\ \frac{1}{J_z}[(J_x - J_y)pq - K_d(\omega_1^2 - \omega_2^2 + \omega_3^2 - \omega_4^2)] \end{bmatrix} \tag{4.37}$$

Quaternion math

A quaternion is a hyper complex number of rank 4, which can be represented as follow

$$\mathbf{q} = \begin{bmatrix} q_0 & q_1 & q_2 & q_3 \end{bmatrix}^T \tag{4.38}$$

The quaternion units from q_1 to q_3 are called the vector part of the quaternion, while q_0 is the scalar part [37]. Multiplication of two quaternions $\mathbf{p}$ and $\mathbf{q}$, is being performed by the *Kronecker product*, denoted as $\otimes$. If $\mathbf{p}$ represents one rotation and $\mathbf{q}$ represents another rotation, then $\mathbf{p} \otimes \mathbf{q}$ represents the combined rotation.

$$\mathbf{p} \otimes \mathbf{q} = \begin{bmatrix} p_0q_0 - p_1q_1 - p_2q_2 - p_3q_3 \\ p_0q_1 + p_1q_0 + p_2q_3 - p_3q_2 \\ p_0q_2 - p_1q_3 + p_2q_0 + p_3q_1 \\ p_0q_3 + p_1q_2 - p_2q_1 + p_3q_0 \end{bmatrix} \tag{4.39}$$

$$= Q(\mathbf{p})\mathbf{q} = \begin{bmatrix} p_0 & -p_1 & -p_2 & -p_3 \\ p_1 & p_0 & -p_3 & p_2 \\ p_2 & p_3 & p_0 & -p_1 \\ p_3 & -p2 & p_1 & p_0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix} \tag{4.40}$$

$$= \bar{Q}(\mathbf{q})\mathbf{p} = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \\ q_1 & q_0 & q_3 & -q_2 \\ q_2 & -q_3 & q_0 & q_1 \\ q_3 & q_2 & -q_1 & q_0 \end{bmatrix} \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} \tag{4.41}$$

The norm of a quaternion is defined as

$$||\mathbf{q}|| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} \tag{4.42}$$

If the norm of the quaternion is equal to 1, then the quaternion is called *unit quaternion*. The complex conjugate of a quaternion has the same definition as normal complex numbers.

$$\mathbf{q}^* = \begin{bmatrix} q_0 & -q_1 & -q_2 & -q_3 \end{bmatrix}^T \tag{4.43}$$

The inverse of a quaternion is defined as a normal inverse of a complex number.

$$\mathbf{q}^{-1} = \frac{\mathbf{q}^*}{||\mathbf{q}||^2} \tag{4.44}$$

The time derivative of the unit quaternion is the vector of *quaternion rates* [38]. It requires some algebraic manipulation but is important to notice that the quaternion rates, $\dot{\mathbf{q}}$, are related to the angular velocity $\omega = \begin{bmatrix} \omega_x & \omega_y & \omega_z \end{bmatrix}^T$. It can be represented in two ways:

- as in equation (4.45) in case that the angular velocity is in the global frame (subscript G)

$$\dot{\mathbf{q}}_{\omega_G}(\mathbf{q}, \omega_W) = \frac{1}{2}\mathbf{q} \otimes \begin{bmatrix} 0 \\ \omega_G \end{bmatrix} = \frac{1}{2}Q(\mathbf{q}) \begin{bmatrix} 0 \\ \omega_G \end{bmatrix} \tag{4.45}$$

- as in equation (4.46) if the angular velocity vector is in the body frame of reference (subscript B).

$$\dot{\mathbf{q}}_{\omega_B}(\mathbf{q}, \omega_B) = \frac{1}{2}\begin{bmatrix} 0 \\ \omega_B \end{bmatrix} \otimes \mathbf{q} = \frac{1}{2}\bar{Q}(\mathbf{q})\begin{bmatrix} 0 \\ \omega_B \end{bmatrix} \tag{4.46}$$

A unit quaternion can be used also as a rotation operator, however the transformation requires both the quaternion and its conjugate, as show in equation (4.47). This rotates the vector $\mathbf{v}$ from the world frame to the body frame represented by $\mathbf{q}$.

$$\omega = \mathbf{q} \otimes \begin{bmatrix} 0 \\ \mathbf{v} \end{bmatrix} \otimes \mathbf{q}^* \tag{4.47}$$

Unit quaternion can be use also to represents *rotation matrices*. Consider a vector $\mathbf{v}_G$ in the global frame. If $\mathbf{v}_B$ is the same vector in the body coordinates, then the following relations hold

$$\begin{bmatrix} 0 \\ \mathbf{v}_B \end{bmatrix} = \mathbf{q} \cdot \begin{bmatrix} 0 \\ \mathbf{v}_G \end{bmatrix} \cdot \mathbf{q}^* \tag{4.48}$$

$$= \bar{Q}(\mathbf{q})^T Q(\mathbf{q}) \begin{bmatrix} 0 \\ \mathbf{v}_G \end{bmatrix} \tag{4.49}$$

$$= \begin{bmatrix} 1 & \mathbf{0}^T \\ \mathbf{0} & R_{\mathbf{q}}(\mathbf{q}) \end{bmatrix} \begin{bmatrix} 0 \\ \mathbf{v}_G \end{bmatrix} \tag{4.50}$$

where

$$R_{\mathbf{q}}(\mathbf{q}) = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \tag{4.51}$$

That is,

$$\mathbf{v}_B = R_{\mathbf{q}}(\mathbf{q})\mathbf{v}_G \tag{4.52}$$

$$\mathbf{v}_G = R_{\mathbf{q}}(\mathbf{q})^T \mathbf{v}_B \tag{4.53}$$

Just as with rotation matrices, sequences of rotations are represented by products of quaternions. That is, for unit quaternions $\mathbf{q}$ and $\mathbf{p}$, it holds that

$$R_{\mathbf{q}}(\mathbf{q} \cdot \mathbf{p}) = R_{\mathbf{q}}(\mathbf{q}) R_{\mathbf{q}}(\mathbf{p}) \tag{4.54}$$

Finally, for representing quaternion rotations in a more intuitive manner, the conversion from Euler angles (roll ϕ, pitch θ and yaw ψ) to quaternion and vice versa can be performed by utilizing the following two equations respectively.

$$q = \begin{bmatrix} \cos(\phi/2)\cos(\theta/2)\cos(\psi/2) + \sin(\phi/2)\sin(\theta/2)\sin(\psi/2) \\ \sin(\phi/2)\cos(\theta/2)\cos(\psi/2) - \cos(\phi/2)\sin(\theta/2)\sin(\psi/2) \\ \cos(\phi/2)\sin(\theta/2)\cos(\psi/2) + \sin(\phi/2)\cos(\theta/2)\sin(\psi/2) \\ \cos(\phi/2)\cos(\theta/2)\sin(\psi/2) - \sin(\phi/2)\sin(\theta/2)\cos(\psi/2) \end{bmatrix} \tag{4.55}$$

$$\begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} = \begin{bmatrix} \text{atan2}(2(q_0q_1 + q_2q_3), q_0^2 - q_1^2 - q_2^2 + q_3^2) \\ \text{asin}(2(q_0q_2 - q_3q_1)) \\ \text{atan2}(2(q_0q_3 + q_1q_2), q_0^2 + q_1^2 - q_2^2 - q_3^2) \end{bmatrix} \tag{4.56}$$

The nonlinear model

The state variables for the quadrotor model are the following:

The inertial position $P = [p_n\ p_e\ h]^T$ is measured with respect to the inertial frame F_i, where p_n is the north position; p_e is the east position; and h is the height (z_i direction).

p_n = the inertial (north) position of the quadrotor along i^i in F_i,
p_e = the inertial (east) position of the quadrotor along j^i in F_i,
h = the altitude of the aircraft measured alongk_i in F_i.

The linear velocity $V = [u\ v\ w]^T$, is measured with respect to the body frame F_b, where u is the x_b-component; v is the y_b-component; and w is the z_b-component.

u = the body frame velocity measured along i_b in F_b,
v = the body frame velocity measured along j_b in F_b,
w = the body frame velocity measured along k_b in F_b.

The angular orientation is represented by quaternions and Euler angles. The quaternion $Q_v^b = [q_0\ q_1\ q_2\ q_3]^T$ represents the orientation of the body frame with respect to the inertial frame.

The angular orientation is represented by the Euler angles. The Euler angles are $\Sigma = [\phi\ \theta\ \psi]^T$, where ϕ is the *roll* angle, θ is the *pitch* angle and ψ is the *yaw* angle.

ϕ = the *roll* angle defined with respect to F_{v2} ,
θ = the *pitch* angle defined with respect to F_{v1},
ψ = the *yaw* angle defined with respect to F_v.

The angular velocity $\Omega = [p\ q\ r]^T$ is measured with respect to F_b, where p is the x_b-component, or *roll* rate, q is the y_b-component, or *pitch* rate, and r is the z_b-component,or *yaw* rate.

p = the *roll* rate measured along i_b in F_b,

q = the pitch rate measured along $j_b$in F_b,
r=the yaw rate measured along k_b in F_b.

The following state vector will be used for representing the motion of the multi-rotor:

$$\begin{bmatrix} \dot{p}_n \\ \dot{p}_e \\ \dot{h} \end{bmatrix} = R_b^v \begin{bmatrix} u \\ v \\ w \end{bmatrix} \tag{4.57}$$

$$\begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \end{bmatrix} = \begin{bmatrix} rvqw \\ pwru \\ qupv \end{bmatrix} + R_v^b \begin{bmatrix} 0 \\ 0 \\ g \end{bmatrix} + \frac{1}{m} \begin{bmatrix} 0 \\ 0 \\ -F \end{bmatrix} \tag{4.58}$$

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & sin(\phi)tan(\theta) & cos(\phi)tan(\theta) \\ 0 & cos(\phi) & -sin(\phi) \\ 0 & \frac{sin(\phi)}{cos(\theta)} & \frac{cos(\phi)}{cos(\theta)} \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{4.59}$$

$$\begin{bmatrix} \dot{q}_0 \\ \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \frac{1}{2} \begin{bmatrix} -q_1 & -q_2 & -q_3 \\ q_0 & -q_3 & q_2 \\ q_3 & q_0 & -q_1 \\ -q_2 & q_1 & q_0 \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{4.60}$$

$$\begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} qr(J_y - J_z)/J_x \\ pr(J_z - J_x)/J_y \\ pq(J_x - J_y)/J_z \end{bmatrix} + \begin{bmatrix} \tau_\phi/J_x \\ \tau_\theta/J_x \\ \tau_\psi/J_x \end{bmatrix} \tag{4.61}$$

Equation 4.57 is the Translation Kinematics, equation 4.58 is the Translation Dynamics, equations 4.59 and 4.60 represent the Rotation Kinematics in terms of Euler angles and quaternions, respectively, and equation 4.61 is the Rotation Dynamics. The above equations can be obtained by considering the physical principles of a quadrotor frame and then applying the Newton-Euler or the Lagrangian methods [39]. In Equations 4.57 to 4.61, m is the mass of the quadrotor, g is the acceleration of the gravity, and J_x, J_y, and J_z are the x_b, y_b, and z_b axis moments of inertia of the airframe, respectively. Finally, R^b is the rotation matrix from frame F_v to frame F_b and $R_v^b = (R_b^v)^T$ is the inverse rotation matrix. In the expanded version of the paper, we present the rotation matrices in terms of quaternions and Euler angles and also the unraveled dynamic equations.

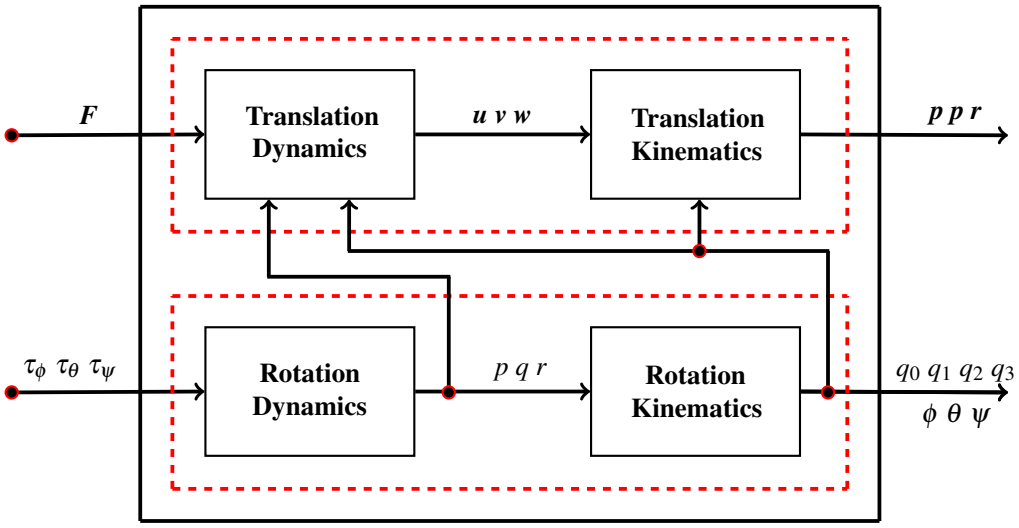


Figure 4.7: Model Block Diagram.

Figure 4.7 shows a block diagram for the nonlinear model of a quadrotor, indicating the dataflow of the variables along the blocks.

4.3 Control strategy

The goal is to have complete position control of the quadrotor. To achieve this multiple control loopshave to be implemented.

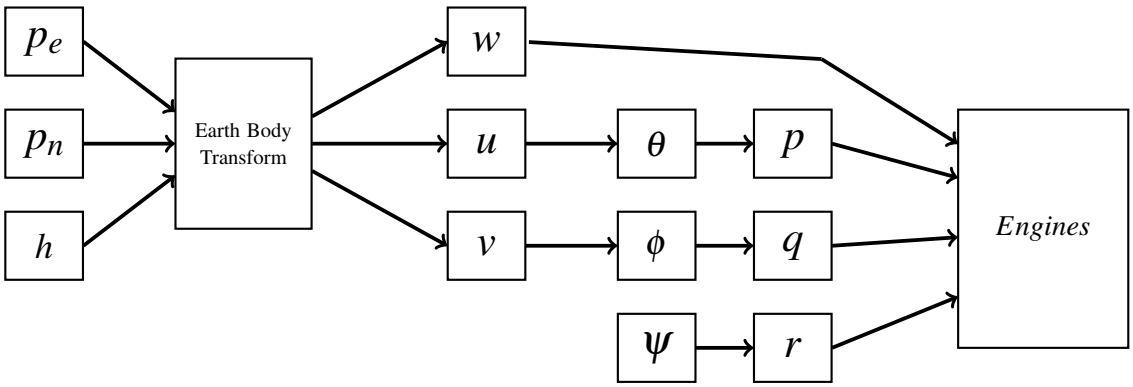


Figure 4.8: Control loops schematic.

Picture showing different levels of control loops. From inner loops to outer loops: angular rates control, attitude control, body-fixed frame speeds control and earth-fixed frame position control.

In the Figure 4.8 each block represents one control loop with the controlled variable written inside. The design of the nested loops was chosen because of following reasons:

- The quadrotor system contains twelve integrators (this can be seen from the linear state equations in the subsection 4.3) which are introducing phase lag. To counter this derivative action creating phase lead needs to be introduced. As mentioned before the D parts of the PIDs are only acting as normal proportional gains thus introducing no phase lead at all. The only controllable astatic system by the simple P controller is system with only one integrator. Hence the control loops are decomposed in the manner to have only one integrator in them.
- Nested control loops provide more precise control of the system. It is very very convenient toput limitations on some control signal (e.g. pitch and roll angle limit), which is easily achievedusing the nested loops.
- Nested loops provide more flexibility. In a case when lower level control is needed (e.g.velocities, attitude, ...) it is very easy to disconnect the outer loops and leave only the appropriate inner loops.

The disadvantage of many control loops each controlling only one state is slower response time compared to controller controlling more state simultaneously.

The nonlinear model expressed in terms of Euler angles is:

$$\begin{cases} \dot{p}_n = w(sin\phi sin\psi + cos\phi cos\psi sin\theta) + \\ \quad - v(cos\phi sin\psi - cos\psi sin\phi sin\theta) + ucos\psi cos\theta \\ \dot{p}_e = v(cos\phi cos\psi + sin\phi sin\psi sin\theta) + \\ \quad - w(cos\psi sin\phi - cos\phi sin\psi sin\theta) + ucos\theta sin\psi \\ \dot{h} = wcos\phi cos\theta - usin\theta + vcos\theta sin\phi \\ \dot{u} = rv - qw - gsin\theta + \frac{f_{wx}}{m} \\ \dot{v} = pw - ru + gsin\phi cos\theta + \frac{f_{wy}}{m} \\ \dot{w} = qu - pv + gcos\theta cos\phi + \frac{F}{m} \\ \dot{\phi} = p + rcos\phi tan\theta + qsin\phi tan\theta \\ \dot{\theta} = qcos\phi - rsin\phi \\ \dot{\psi} = r\frac{cos\phi}{cos\theta} + q\frac{sin\phi}{cos\theta} \\ \dot{p} = \frac{\tau_\phi}{J_x} + rq\frac{J_y - J_z}{Jx} \\ \dot{q} = \frac{\tau_\theta}{J_y} + pr\frac{J_z - J_x}{Jy} \\ \dot{r} = \frac{\tau_\psi}{J_z} + pq\frac{J_x - J_y}{Jz} \end{cases} \tag{4.62}$$

And the model expressed in terms of quaternions is:

$$\begin{cases} \dot{p}_n = u(2q_0^2+2q_1^2-1) - v(2q_0q_3-2q_1q_2) + w(2q_0q_2+2q_1q_3) \\ \dot{p}_e = v(2q_0^2+2q_2^2-1) + u(2q_0q_3+2q_1q_2) - w(2q_0q_1-2q_2q_3) \\ \dot{h} = w(2q_0^2+2q_3^2-1) - u(2q_0q_2-2q_1q_3) + v(2q_0q_1+2q_2q_3) \\ \dot{u} = rv - qw - g(2q_0q_2-2q_1q_3) \\ \dot{v} = pw - ru + g(2q_0q_1+2q_2q_3) \\ \dot{w} = qu - pv + g(2q_0^2+2q_3^2-1) - \frac{F}{m} \\ \dot{q}_0 = -\frac{q_1p}{2} - \frac{q_2q}{2} - \frac{q_3r}{2} \\ \dot{q}_1 = \frac{q_0p}{2} - \frac{q_3q}{2} + \frac{q_2r}{2} \\ \dot{q}_2 = \frac{q_3p}{2} + \frac{q_0q}{2} - \frac{q_1r}{2} \\ \dot{q}_3 = \frac{q_1q}{2} - \frac{q_2p}{2} + \frac{q_0r}{2} \\ \dot{p} = \frac{\tau_\phi}{J_x} + rq\frac{J_y-J_z}{Jx} \\ \dot{q} = \frac{\tau_\theta}{J_y} + pr\frac{J_z-J_x}{Jy} \\ \dot{r} = \frac{\tau_\psi}{J_z} + pq\frac{J_x-J_y}{Jz} \end{cases} \tag{4.63}$$

Notice that while the Euler angles model is characterized trigonometric functions and singularities, the quaternion model is polynomial and singularity free.

External forces and moments

External forces and moments are generated by gravity, aerodynamics and propulsion. In first approximation, the expression of the force and axial momentum generated by each motor depends on the square of the propeller angular speed via aerodynamic coefficients. Considering k_T and k_R respectively force and axial momentum coefficients and l the distance between rotors axes, applied forces and moments on quadrotor can be modelled as in

$$\begin{cases} F_z = \sum_{n=1}^{4} F_i = \sum_{n=1}^{4} k_T \omega_i^2 = 4k_T \omega_i^2 \\ M_x = (\omega_2^2 - \omega_4^2) k_T l \\ M_y = (\omega_3^2 - \omega_1^2) k_T l \\ M_z = \sum_{n=1}^{4} M_i(-1)^{i+1} = \sum_{n=1}^{4} \omega_i^2 k_R (-1)^{i+1} \end{cases} \tag{4.64}$$

In matrix notation:

$$\begin{Bmatrix} F \\ M_x \\ M_y \\ M_z \end{Bmatrix} = \begin{bmatrix} k_T & k_T & k_T & k_T \\ 0 & k_T l & 0 & -k_T l \\ -k_T l & 0 & k_T l & 0 \\ k_R & -k_R & k_R & -k_R \end{bmatrix} \begin{Bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{Bmatrix} \tag{4.65}$$

Gravitational force acts on the center of gravity and its defined in the body-fixed frame by:

$$\underline{F_g} = \begin{Bmatrix} F_g x \\ F_g y \\ F_g z \end{Bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\phi & -sin\phi \\ 0 & sin\phi & cos\phi \end{bmatrix} \begin{bmatrix} cos\theta & 0 & sin\theta \\ 0 & 1 & 0 \\ -sin\theta & 0 & cos\theta \end{bmatrix} \begin{Bmatrix} 0 \\ 0 \\ mg \end{Bmatrix} = \begin{Bmatrix} -mgsin\theta \\ mgcos\theta sin\phi \\ mgcos\theta cos\phi \end{Bmatrix} \tag{4.66}$$

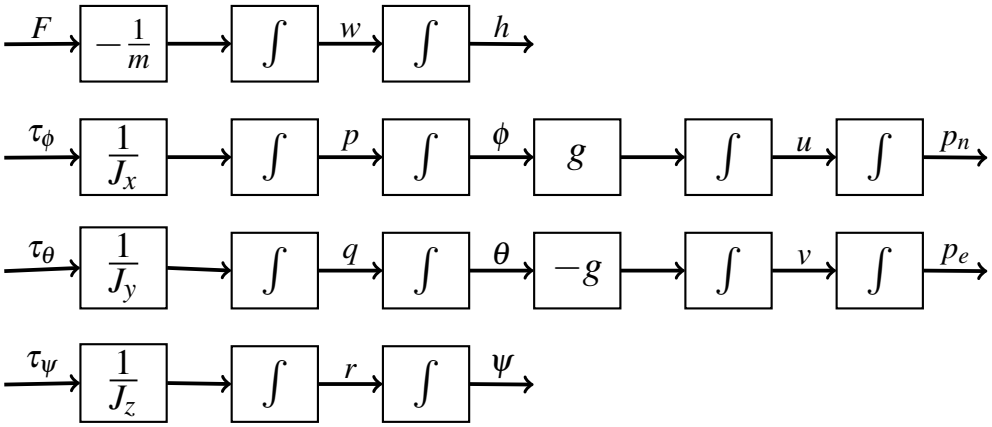


Figure 4.9: Linear Model Block Diagram (Euler angles).

As for the aerodynamic forces and moments, drag is opposite to the speed vector and depends on the dynamic pressure and the shape of the structure. The aerodynamic drag can be considered focused on the center of gravity of the quadrotor and it depends on C_d and S , related to the quadrotor geometry. However in a first approximation it can be neglected if low speed manoeuvres are requested to the vehicle. This is a conservative assumption due to the damping nature of aerodynamic forces and moments:

$$\underline{D} = -0.5\rho|V|^2 SC_d\underline{v} \tag{4.67}$$

Linearization Design

In order to perform control design and reachability analysis we use two approaches for the linearization of the quadrotor nonlinear equations. First, here is presented the linearization around an equilibrium point, corresponding to the quadrotor being at a static hovering position in 3D space.

The basic hypothesis are that $u_0 = v_0 = w_0 = 0\ [m/s]$ and $p_0 = q_0 = r_0 = 0\ [rad/s]$, and the following relations come from the analysis of the nonlinear equations: $\phi_0 = \theta_0 = \psi_0 = 0\ rad$, $F_0 = mg\ [N]$, $\tau_\phi = \tau_\theta = \tau_\psi = 0\ [N \cdot m]$, and $(p_{n0}, p_{e0}, h_0) \in \mathbb{R}^3\ [m]$. The linearized model can be written as the following simple set of linear equations3:

$$\begin{array}{llll} \dot{p}_n = u & \dot{u} = -g\theta & \dot{\phi} = p & \dot{p} = \tau_\phi / J_x \\ \dot{p}_e = v & \dot{v} = g\phi & \dot{\theta} = q & \dot{q} = \tau_\theta / J_y \\ \dot{h} = w & \dot{w} = -F/m & \dot{\psi} = r & \dot{r} = \tau_\psi / J_z \end{array} \tag{4.68}$$

The corresponding model block diagram is depicted in Figure 4.9.

The previous linearized model is useful for control design, because of its simplicity. The decoupling of the variables is exploited in the controller design, in order to make nested control architecture. Notice that the rotational part of the nonlinear model presented in [41] is equivalent to the rotational part of the linear model in equation 4.68. The relevant difference in the translational parts of both models is that the linear velocities in [41] are expressed in terms of inertial frame coordinates.

When applying the reachability analysis to investigate the stability and the performance of the closed loop system, we need a richer model that could better represent the actual system behavior. One option could be to use the nonlinear model. Notice that this nonlinear model has also uncertainties and unmodelled dynamics, like lift and drag forces [42], and would contribute with some of the possible traces of the state trajectory. Therefore, can be use a piece wise model for the quadrotor to be applied in the reachability analysis [43].

PX4 attitude control module

Before designing the controller for angular rates control, it is necessary to understand how the current attitude module on the Pixhawk is structured. The attitude module is comprised of two loops: Proportional (P) controller loop for angular error and a Proportional Derivative (PD) controller loop for angular rate error; at the time of writing this thesis, the angular rate error was controlled by a PD controller. In Figure 4.10 below, the P controller and PD controllers are used in the attitude and rates controller blocks respectively.

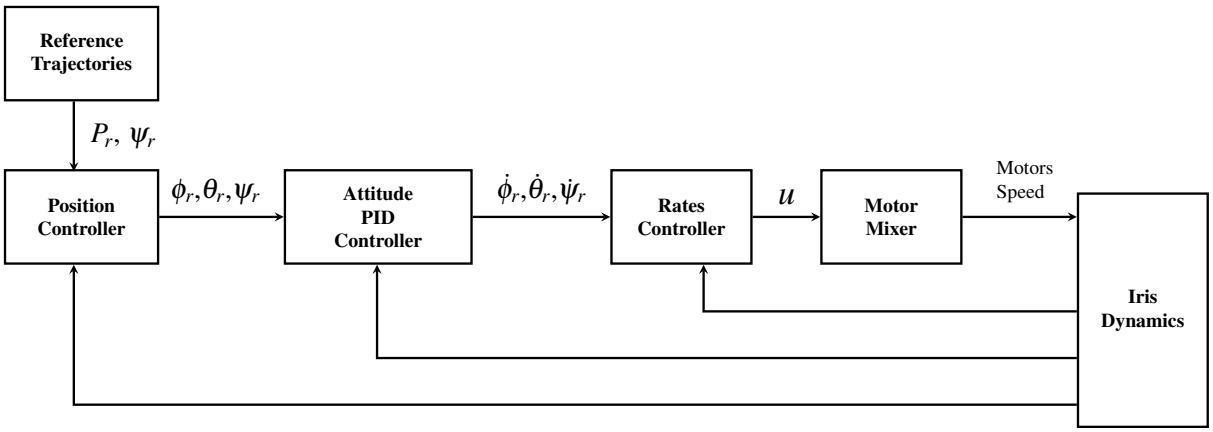


Figure 4.10: Block Diagram of Quadrotor Control Model.

Controller Design

The architecture of the attitude controller is shown in Figure 4.11. Weve proposed a nested control architecture. The inner loops are the angular rates and the vertical speed controllers that act directly on the thrust F and the torques τ_ϕ, τ_θ and τ_ψ . In an outer loop, the vertical position and the angular orientation controllers receive the references to the height and orientation, and transform them into references for the inner loop controllers. The proposed nested control architecture permits the development of each control loop in arelatively independent way, being each controller responsible for a different aspect of the overall attitude control. It is inspired on both the decoupling of the linear model and the common practice of aircraft and missile control design [44].

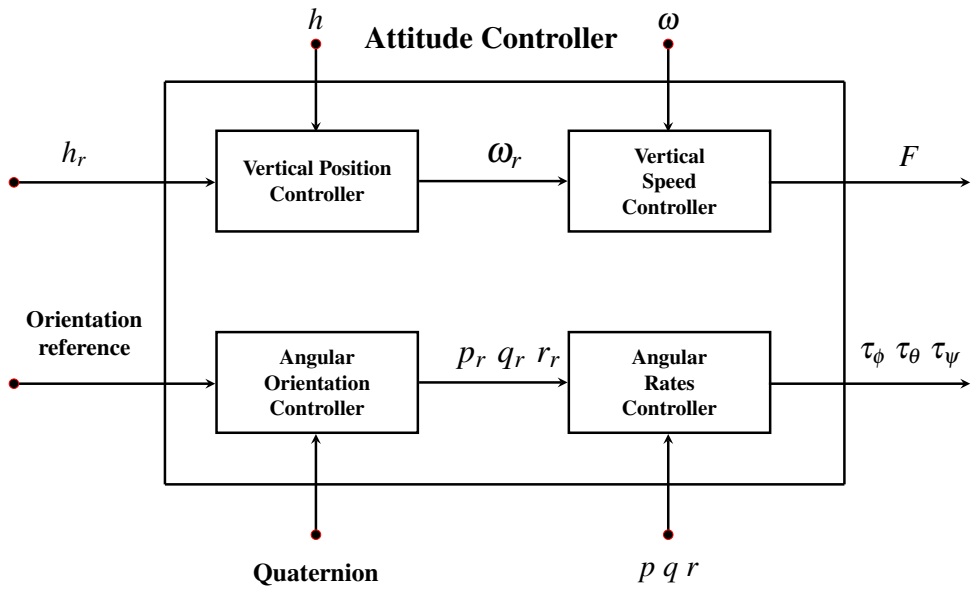


Figure 4.11: Attitude Controller Architecture.

Parameter Tuning

The control structureof the quadrotor is divided into two loops, i.e., inner loopand outer loop. The block diagram of such a structure isschematically shown in Figure 4.11. In the outer loop, the position vector P_r and *yaw* angle ψ_r are chosen as the reference signals. In the inner loop, the reference signal (ϕ,θ,ψ) comes from the outer loop. Moreover, the fourinputs generated by the two loops are converted into theangular speeds of motors.

Trajectory Generator Block

he trajectory generator block computes the desired flight trajectory that needs to be followed by the quadrotor and generates the set of desired positions (i.e., x_d, y_d and z_d) and desired Euler angles (i.e. ϕ_d,θ_d and ψ_d) in the inertial coordinate frame for that trajectory. If a Remote Control (RC) is used to generate flight command during an actual flight, the trajectory generator block will output the desired Euler angles or their rates to the IRIS quadrotor; the choice depends on the control architecture. The desired positions and Euler angles are used as inputs for the controller block to stabilize the IRIS in flight during the change in the IRIS attitude.

Controller Block

The control of the quadrotors position and attitude is accomplished by the design of the feedback controller and the method was well documented in [39] and [4]. As mentioned in the previous chapters, the quadrotor is an under-actuated system. Therefore, to move forward in the x direction, the quadrotor must first change its attitude by pitching downwards to generate a horizontal force from the propellers thrusts, while maintaining its altitude. Similarly, in order to move laterally in the y direction, the quadrotor must change its attitude by rolling to the right or

left while maintaining its altitude. Therefore, the control equations of the quadrotors position and attitude channels are shown in equations 4.69 and 4.70.

1. Attitude Control Equations:

$$\phi_d = K_{1,y}[R_I^b(y_d - y) - K_{2,y}\dot{y}_b] \tag{4.69}$$

$$\theta_d = K_{1,x}[R_I^b(x_d - x) - K_{2,x}\dot{x}_b] \tag{4.70}$$

where: R_b^I is the rotation matrix that transforms the quadrotors position from the inertia to vehicle coordinate frame,
K_1,x is the proportional gain term for the position in the x direction,
K_2,x is the derivative gain term for the velocity in the x direction,
K_1,y is the proportional gain term for the position in the y direction,
K_2,x is the derivative gain term for the velocity in the y direction.

2. Forces and Moments Control Equations:

$$\tau_\phi = K_{P,roll}(\phi_d - \phi_m) + K_{I,roll}(\phi_d - \phi_m) + K_{D,roll}(\dot{\phi}_d - \dot{\phi}_m) \tag{4.71}$$

$$\tau_\theta = K_{P,roll}(\theta_d - \theta_m) + K_{I,roll}(\theta_d - \theta_m) + K_{D,roll}(\dot{\theta}_d - \dot{\theta}_m) \tag{4.72}$$

$$\tau_\psi = K_{P,roll}(\psi_d - \psi_m) + K_{I,roll}(\psi_d - \psi_m) + K_{D,roll}(\dot{\psi}_d - \dot{\psi}_m) \tag{4.73}$$

$$T = K_{P,z}(z_d - z_m) + K_{I,z}(z_d - z_m) + K_{D,z}(\dot{z}_d - \dot{z}_m) \tag{4.74}$$

$$\omega_0 = \sqrt{\left(\frac{mg}{4K_T}\right)} \tag{4.75}$$

where:
$K_P,_{roll,pitch,yaw}$ is the proportional gain term for the roll, pitch or yaw angles,
$K_I,_{roll,pitch,yaw}$ is the Integral gain term for the roll, pitch or yaw angles,
$K_D,_{roll,pitch,yaw}$ is the derivative gain term for the roll, pitch or yaw rates,
$K_{P,z}$ is the proportional gain term for the position along the z-axis,
$K_{I,z}$ is the integral gain term for the position along the z-axis,
$K_{D,z}$ is the derivative gain term for the velocity along the z-axis,
ω is the motor rotational speed required to generate a thrust that is equal to the weight of the quadrotor.

Motor Mixer Block

The motor mixer block computes the required angular speed of each propeller in order to generate the thrust force and moments (i.e., T_b, τ_ϕ,τ_θ,τ_ψ) to perform a maneuver or changing the quadrotors attitude in flight. The thrust and moments of the quadrotor are directly proportional to the square of the propellers angular speed (i.e., ω_i^2). The proportional relationship between the thrust and moments of the quadrotor and each propellers angular speed is shown in equation 4.76, represented by the matrix, M. Therefore, to determine the angular speed required for each propeller in order to generate the thrust force and moment, the inverse of M is first derived and multiplied to the *thrust*, *roll*, *pitch* and *yaw* moment vector as shown in the equation 4.77. The calculated angular speed for each propeller will be used as an input for the quadrotor dynamics block.

$$\begin{bmatrix} T_b \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} b & b & b & b \\ -bl & 0 & bl & 0 \\ 0 & -bl & 0 & bl \\ -d & d & -d & d \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_1^2 \\ \omega_1^2 \\ \omega_1^2 \end{bmatrix} = \begin{bmatrix} M \end{bmatrix} \begin{bmatrix} \omega_1^2 \\ \omega_1^2 \\ \omega_1^2 \\ \omega_1^2 \end{bmatrix} \tag{4.76}$$

$$\begin{bmatrix} \omega_1^2 \\ \omega_1^2 \\ \omega_1^2 \\ \omega_1^2 \end{bmatrix} = \begin{bmatrix} M \end{bmatrix}^{-1} \begin{bmatrix} T_b \\ \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} \tag{4.77}$$

IRIS Quadrotor Dynamics Block

The IRIS dynamics block consists of the 6DOF. The required angular speed for each propeller is provided by the motor mixer block to the IRIS dynamics block as an input and the body response is measured by the sensors onboard the IRIS as the measured position, Euler angles and their rates. The measured responses are fed back to the controller block and used to determine the error signals for each of the position and Euler angle channel.

4.4 WGS 84

The earth has an irregular spheroid-like shape. The natural coordinate reference system for geographic data is longitude/latitude. This is an angular system. The latitude ϕ of a point is the angle between the equatorial plane and the line that passes through a point and the center of the Earth. Longitude λ is the angle from a reference meridian (lines of constant longitude) to a meridian that passes through the point.

Obviously we cannot actually measure these angles. But we can estimate them. To do so, you need a model of the shape of the earth. Such a model is called a datum. The simplest datums are a spheroid (a sphere that is flattened at the poles and bulges at the equator). More complex datums allow for more variation in the earths shape. The most commonly used datum is called WGS84 (World Geodesic System 1984). This is very similar to NAD83 (The North American Datum of 1983). Other, local datums exist to more precisely record locations for a single country or region.

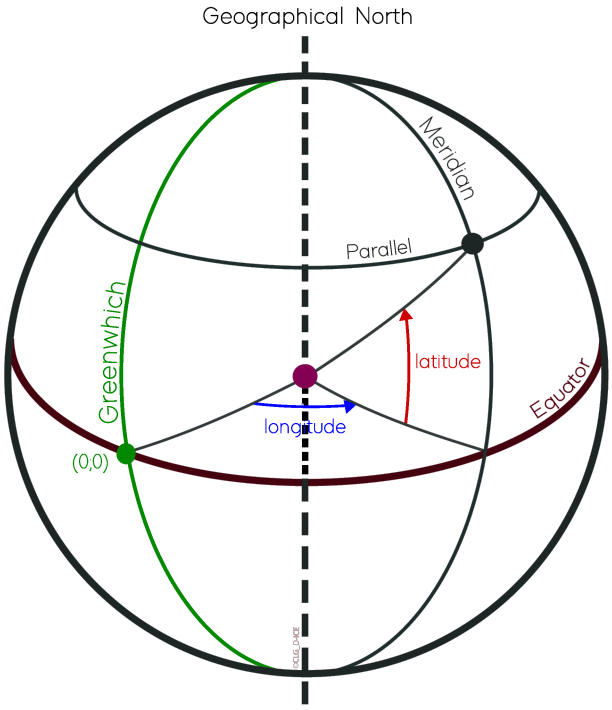


Figure 4.12: World Geodetic System.

So the basic way to record a location is a coordinate pair in degrees and a reference datum. (Sometimes people say that their coordinates are in WGS84. That is meaningless; but they typically mean to say that they are longitude/latitude relative to the WGS84 like Figure 4.12 datum).

UTM

The *UTM* system provides a set of 2-dimensional Cartesian coordinate frames. Referring to any possible reference ellipsoid, the Earth is segmented into 60 zones, each being a 6band of longitude. In each zone a secant transverse Mercator projection is applied. For details and formulas see e.g.[40].

Although, strictly speaking latitude bands are not part of **UTM**, but rather defined in the Military Grid Reference System (**MGRS**), they are commonly used. Each *UTM* (longitude) zone is divided into 20 latitude bands. Each latitude band is 8 high, and is identified by letters startingfrom "C" at 80S, following the alphabet until "X", without the letters "I" and "O", because of their similarity to 1 and 0 respectively.

The **UTM** tiles over Europe are illustrated in Figure 4.13 on the facing page.

Figure 4.13: UTM Grid.

5 Autonomous Flight System of UAV through Global and Local Path Generation

In this chapter, the method used in order to accomplish the goal of this project is presented, explaining all the steps necessary to reproduce this work, for this reason it was proposed and chosen a Global and Local path planning and trajectory system for autonomous UAV's flight. The whole system was built based on the ROS robot operating system. The UAV's embedded computer detects obstacles using 2-D Lidar and generates real-time Local and Global paths based on VFH and RRT* respectively. In addition, it gives the ground station computer receives the obstacle information, generates a 2-D SLAM map, transmits the destination point to the embedded computer, and controls the status of the unmanned aerial vehicle. The autonomous flight system of the proposed unmanned aerial vehicle was verified through simulation flight in 3-D space.

5.1 Research tools: hardware and software

In order to accomplish the implementation phase of this research, some hardware and software tools and frameworks have been necessary. The hardware tools used for this purpose are the following:

- *IMU:* IMU (Inertial Measurement Unit) measures and reports on the crafts velocity, orientation and gravitational forces, using a combination of accelerometers, gyroscopes, and magnetometers.
- *LiDAR sensor:* LiDAR (Light Detection And Ranging) is used for obstacle detection and distance measurements from the ground. It is the sensor used by the UAV to obtain data from the environment. The set of collected data is called pointcloud, which is a cloud of virtual points created using the distance information obtained by the LiDAR by casting laser rays in the environment.

- *Autopilot PX4:* An autopilot is the bridge between the commands given to the UAV for its motion and the electric signals that arrives to themotors and allow the UAV to fly.
- *on-board computer:* Quadcopter with a 3DR Pixhawk Flight Controller, equipped with a Raspberry Pi.

For what concerns instead the software part, many frameworks and tools have been used for different reasons. Their functions are summarized here:

- *ROS:* This is the framework used in this project for the development and programming of the UAV both from the simulation and in the real unit. It provides the necessary services for the robots, such as the communication infrastructure with the sensors and the low-level control, and it provides also and easy integration with simulator engines, allowing to develop and test the algorithms in a simulation and directly import them in the real robot.
- *Gazebo:* The simulator for robotics used in this project. Its easy integration with ROS and its fidelity in the physic engine are the reason of this choice.
- *PX4 firmware:* The firmware integrated in the PX4 autopilot, which offers numerous features for the control and the monitoring of the vehicle.
- *Octomap and Rviz:* Octomap is the framework used in this project to create a 3D map of the environment which tells the UAV the occupied and free space for the exploration. Rviz is the software used for the visualization of the Octomap and of other features.

More details about the hardware and the software used in this project can be found in the chapter 6.

5.2 Planning Under Differential Constraints

In this section there can be both Global (obstacles) and Local (differential) constraints on the continuous state spaces that arise in motion planning. Dynamical systems are also considered, which yields state spaces that include both position and velocity information (this coincides with the notion of astate spacein control theory or a phase space in physics and differential equations).

The entire network system was constructed by publishing and subscribing communication between program nodes that necessary for data and commands elaborate between the ground station PC and the embedded computer of the UAV using ROS Network.

The autonomous flight system of the UAV in this section performs global route flight by generating a global route based on the HECTOR SLAM map when flying a local route when detecting a nearby obstacle and when inputting a destination route from a ground station. Global path regenerate global path based on a new map after evasive maneuver using local path generation when detecting a nearby obstacle on the global path movement path. In this case, the movement control command is generated as a local route according to the movement vector in consideration of the speed of the unmanned aerial vehicle in both the local and global routes.

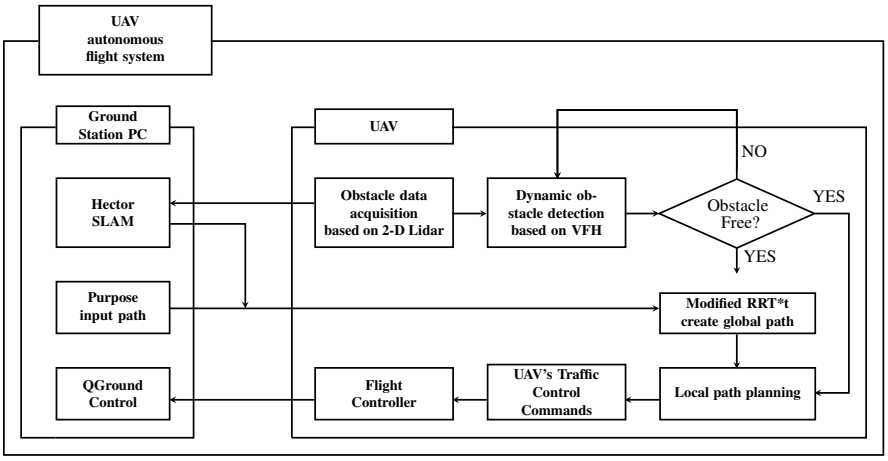


Figure 5.1: Block diagram of the autonomous flight system.

The block diagram of the autonomous flight system for performing the entire mission of the unmanned aerial vehicle is shown in Figure 5.1

To study how the collision avoidance algorithm interacts with the system during automatic missions, the entire system has been simulated in an adequate testing environment. Experimental tests have been made firstly on simulations especially for security reasons and also because of the need to find an easy to deploy solution that was close to real UAV behaviours. For these reasons, a SITL approach has been preferred. It would allow to perform simulations, implementations and optimization of the collision avoidance algorithm without taking any risk. In order to reproduce a 3D simulated environment, a suitable framework has been firstly chosen: Robot Operating System (ROS) is an open source library tool box generally utilized to develop robot-based applications. It allows to operate with different environment simulators. For this thesis, Gazebo has been chosen. It is an open source 3D simulation environment that allows to simulate multi-robot behaviours on indoor and outdoor applications. These two softwares allow to test the collision avoidance algorithm, providing realistic scenarios with the real physics simulation. Their implementation works as an external simulator for PX4 autopilot firmware allowing to simulate several activities, such as read sensor data from the virtual-created UAV, communicate between user inputs and the vehicle model, GPS information and 3D UAV physical stability simulations with user command responses.

5.2.1 Local Path Planning

A local path was created based on the Vector Field Histogram (VFH) algorithm using the surrounding obstacle data acquired through the 2-D LiDAR mounted on the unmanned aerial vehicle. The VFH algorithm is an algorithm that has already been developed and is being operated for robot route planning, and is used to avoid local path for unexpected obstacles approaching the unmanned aerial vehicle. The VFH-based local path generation algorithm generates a local avoidance path by generating a real-time histogram of the surrounding obstacle data acquired through 2-D LiDAR.

Obstacle detection using 2-D Lidar

In this work a 2-D LiDAR capable of 360° measurement is used. The angle of the obstacle obtained by θ, distance r, and (equation 5.1) below. The relative position p_x, p_y of the obstacle was calculated.

$$[p_x, p_y] = [rcos\theta, rsin\theta] \tag{5.1}$$

HECTOR SLAM

The HECTOR SLAM algorithm used in this project is a 2-D map generation algorithm through simultaneous localization using the relative positions of the unmanned aerial vehicle and the detected obstacle. In Figure 5.2 for HECTOR SLAM map creation shows the TF structure of a man-made aircraft and a 2-D lidar.

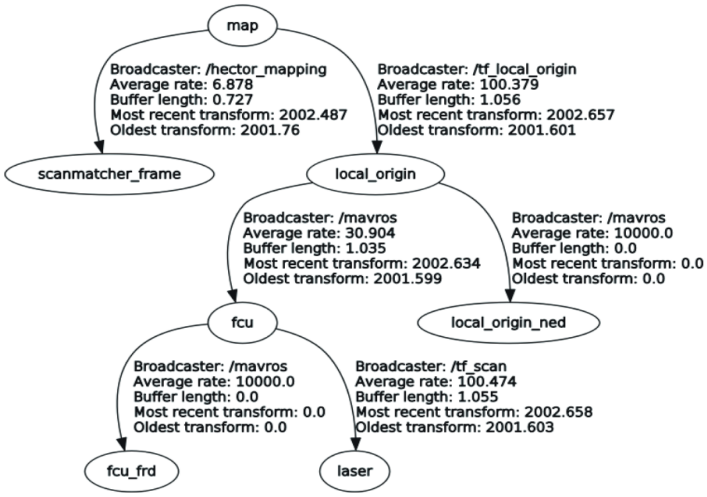
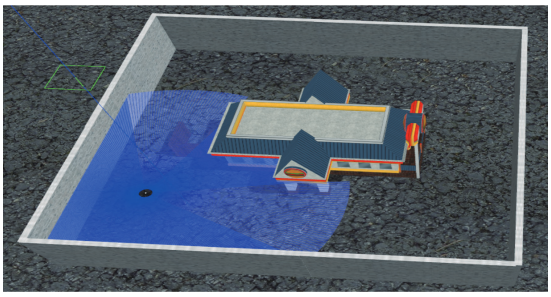
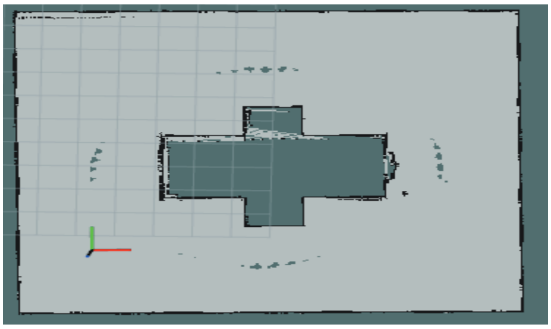


Figure 5.2: UAV and 2-D Lidar's TF Structure

Figure 5.3-(a) shows an unmanned aerial vehicle equipped with a 2-D LiDAR and a simulator of an obstacle environment, Figure 5.3-(b) shows the HECTOR SLAM map using Rviz, a 3-D visualization tool.



(a) Gazebo Simulation.



(b) Rviz 3-D Visualization Tool.

Figure 5.3: HECTOR SLAM.

VFH Algorithm

The Vector Field Histogram (VFH) method, introduced by Borenstein and Korem, [13], is a real-time obstacle avoidance method that permits the detection of unknown obstacles and avoids collisions while simultaneously steering the mobile robot towards the target.

The original paper on VFH, [13], presents a good summary of the method as follows: the VFH method uses a two-dimensional Cartesian histogram grid as a world model. This world model is updated continuously with range data sampled by on-board range sensors. The VFH method subsequently employs a two-stage data-reduction process in order to compute the desired control commands for the vehicle. In the first stage, a constant size subset of the 2D histogram grid considered around the robots momentary location, is reduced to a one-dimensional polar histogram. Each sector in the polar histogram contains a value representing the polar obstacle density in that direction. In the second stage, the algorithm selects the most suitable sector from among all polar histogram sectors with a low polar obstacle density, and the steering of the robot is aligned with that direction.

The three main steps of implementation of the VFH method are summarized:

Step 1 Builds a 2D Cartesian histogram grid of obstacle representation;

Step 2 From the previous 2D histogram grid, considers an active window around the robot, and filters that 2D active grid onto a 1D polar histogram;

Step 3 Calculates the steering angle and the velocity controls from the 1D polar histogram, as a result of an optimization procedure.

Step 1 of VHF

Step 1 of VHF generates a 2D Cartesian coordinate from each range sensor measurement and increments that position in a 2D Cartesian histogram grid map C. The construction of this grid maps does not depend on the specific sensor used for obstacle detection, in particular ultrasound and laser range sensors may be used.

For ultrasound sensors and for each range reading, the cell that lies on the acoustic axis and corresponds to the measured distance d is incremented like in Figure 5.4, increasing the certainty value (CV) of the occupancy of that cell. This is the Histogrammic in Motion Mapping (HIMM) algorithm presented in [13].
(Figute 5.4) illustrates the cells certainty value update along the movement of arobot equipped with ultrasonic sensors. It is clear that, for each range reading,only one cell is updated. The HIMM provides a histogramic pseudo-probabilitydistribution by continuous and rapid sampling the sensors while the robot is moving. This is different from the certainty grid maps proposed by Moravec and Elfes in [45], [46], that project a probability profile onto all the cells affected by a range reading. For an ultrasound sensor these cells are contained in a cone that corresponds to the main lobe of the sensor radiation diagram.

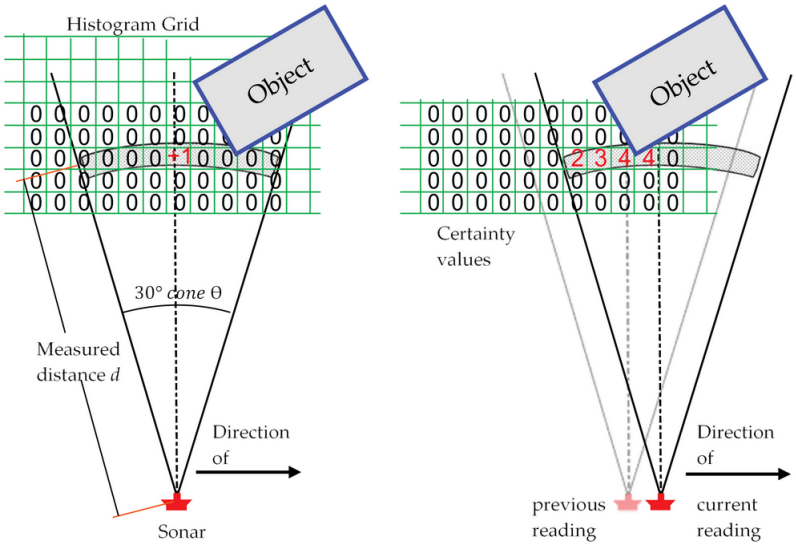


Figure 5.4: Construction of the 2D Histogram grid map. Reproduced from [13]

By storing information about how obstacles are distributed around the UAV, the vector direction of the surrounding obstacles can be known, and a real-time histogram is formed for obstacles close to an arbitrary distance to the UAV. The formed histogram is used to generate local avoidance routes. Weights are given according to the distance of the sector with obstacles to the 360 direction of the unmanned aerial vehicle. Which sector of the pole histogram the obstacle belongs to the following (equation 5.2) is calculate as.

$$\phi = INT(\frac{\theta}{s}) \tag{5.2}$$

ϕ is the index value of the sector, and s is the resolution of one sector. When a sector is determined, the obstacle weight m_ϕ for each sector is

$$m_\phi = \beta(\alpha - r_{min})(\alpha > 0, \beta > 0) \tag{5.3}$$

In (equation 5.3), α is the maximum measurement range of the 2-D Lidar, β is the weight value. α and β are set so that r becomes 0 when m_ϕ is the maximum. r_{min} is the minimum obstacle distance within the sector section.

The VFH algorithm divides the 360° omnidirectional direction into partial sectors using 2-D Lidar. By storing information about how obstacles are distributed around the unmanned aerial vehicle, it is possible to know the vector direction of the distributed obstacles in the vicinity, and a real-time histogram is formed for obstacles close to an arbitrary distance to the unmanned aerial vehicle. The formed histogram is used to generate local avoidance routes. Sec with obstacles for the 360° direction of the unmanned aerial vehicle.

Set the threshold of the unmovable proximity obstacle avoidance range of the unmanned aerial vehicle, and if the sector value is greater than the threshold, it is designated as an immovable area.

The VFH-based local path generation algorithm flies along the proximity obstacle avoidance path, and during flight along the global path, it is set as the vector sum of the avoidance path

vector and the global path vector.

Step 2 of VHF

The next step of VFH, maps the 2D Cartesian histogram grid map C onto a 1D structure. To preserve and isolate the information about the local obstacle information rather than using the entire grid map C, the 2D grid used in this step is restricted to a window of C, called the active window, denoted by C^*, with constant dimensions, centered on the Vehicle Central Point (VCP) and that, consequently, moves with the robot. C^* represents a local map of the environment around the robot.

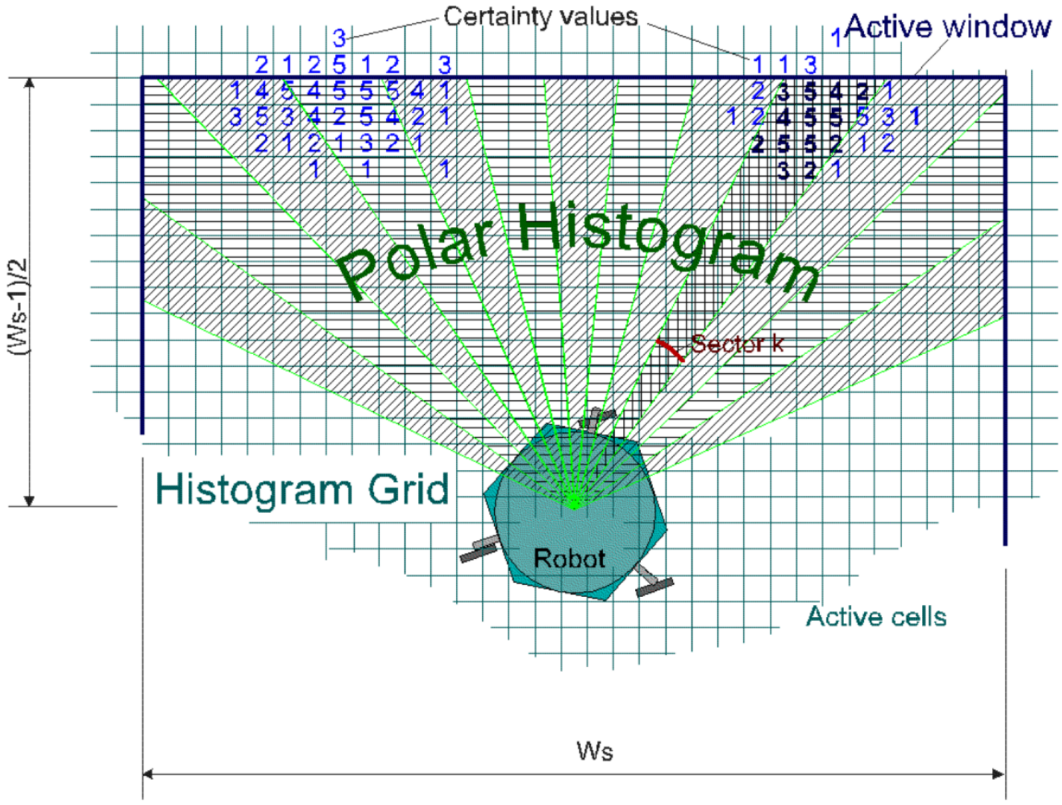


Figure 5.5: Mapping of active cells onto the polar histogram. Reproduced from [13]

The active grid C^* is mapped onto a 1D structure known as a polar histogram, H, that comprises n angular sections each with width α. Figure 5.5 illustrates the cell occupancy of C^*, the active window around the robot, and represents the angular sectors considered for the evaluation of the 1D polar histogram.

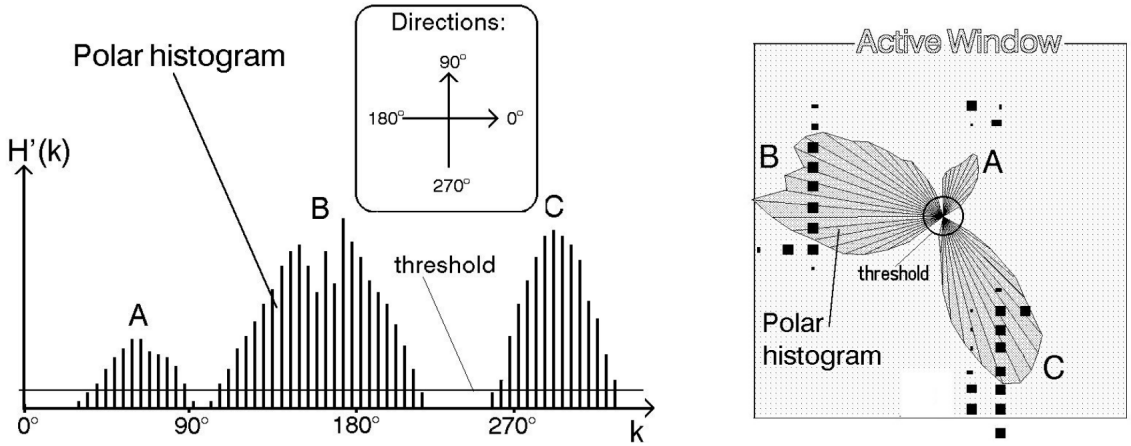


Figure 5.6: a) 1D polar histogram of obstacle occupancy around the robot. b) Polar histogram shown in polar form overlapped with C^*. Both Reproduced from [13]

The x-axis of the polar histogram represents the angular sector where the obstacles were found and the y-axis represents the probability P that there is really an obstacle in that direction based on the occupancy grids cell values of C^*. Figure 5.6-a) represents the 1D polar histogram with obstacle density for a situation where the robot has three obstacles, A, B and C in its close vicinity. In Figure 5.6-b) the previously obtained 1D histogram is shown in polar form overlapped with the referred obstacles.

Step 3 of VHF

The step 3 of VFH evaluates the required steering direction for the robot that corresponds to a given sector of the 1D histogram and, simultaneously, adapts the robot velocity according to the obstacle polar density.

A typical polar histogram contains "peaks" or sectors with a high polar density and "valleys", i.e., sectors with low polar density. To evaluate the steering direction towards the target goal, all openings, i.e., valleys large enough for the vehicleto pass through, are identified as candidate valleys. Valleys are classified as wide or narrow according to the number of sectors below the threshold. If the number of consecutive sectors below the threshold is greater than S_{max} the valley is wide, while it is classified as narrow if that number is less than S_{max}. The parameter S_{max} is set according to the robot dimensions and kinematic constraints.

From the set of wide valleys, VHF chooses the one that minimizes a cost function that accounts for:

- the alignment of the robot to the target,
- the difference between the robot current direction and the goal direction,
- the difference between the robot previously selected direction and the new robot direction.

If the threshold is set too high, the robot may be too close to an obstacle, and moving too quickly in order to prevent a collision. On the other hand, if set too low, VHF can miss some valid candidate valleys. As stated in [13] the VHF needs a fine-tuned threshold only for the

most challenging applications, for example, travel at high speed and in densely cluttered environments. Under less demanding conditions the system performs well even with an imprecisely set threshold. One way to optimize performance is to use an adaptive threshold. Besides the robot steering direction, the VFH also sets the robot speed according to the obstacle polar density, [13].

Creation of the Local Map

Local Map is a grid map that records obstacles around the robot centered on the robot. The size of the grid map is 33 x 33 cells, and each cell has a size of 10 cm x 10 cm.

Creation of the Polar histogram

Create an obstacle vector from each cell $C^*_{i,j}$ in the active area of the grid map. The direction $(\beta_{i,j})$ and magnitude $(m_{i,j})$ of this vector are calculated as follows.

$$(\beta_{i,j}) = tan^{-1}(\frac{y_j - y_0}{x_i - x_0}) \tag{5.4}$$

$$(m_{i,j}) = tan^{-1}(\frac{y_j - y_0}{x_i - x_0}) \tag{5.5}$$

$$(\beta_{i,j}) = (C^*_{i,j})^2(a - bd_{ij}) \tag{5.6}$$

$$(d_{i,j}) = \sqrt{(x_i - x_0)^2 + (y_j - y_0)^2} \tag{5.7}$$

a, b - positive constants;
$C^*_{i,j}$ - certainty value of active cell (i, j)
x_0, y_0 - current coordinates of the VCP(Vehicle Center Point);
s x_i, y_j - coordinates of active cell (i, j)
a, b is set to satisfy $a - bd_{max} = 0$
Calculate which sector the obstacle belongs to on the polar histogram.

$$k = INT\left(\frac{\beta_{i,j}}{\alpha}\right) \tag{5.8}$$

where α is the resolution of one sector. In the program, $\alpha = 1$ was applied. In each sector k, the Polar Obstacle Density (POD) is calculated as follows.

$$h_k = max \rightarrow (m_{i,j}, h_k) \tag{5.9}$$

Smoothed Polar Histogram

The polar histogram is smoothed by considering the obstacle information of each adjacent sector.

$$h'_k = \frac{h_{k-l} + 2h_{k-l+1} + \cdots + lh_k + \cdots + 2h_{k+l-1} + h_{k+l}}{2l+1} \tag{5.10}$$

$h_k^{'}$ becomes the Smoothed Polar Obstacle Density. In the program, it is applied as $l = 25$

Find Steering Direction

Find the closest valley to the robot's destination direction in the smoothed polar histogram. There can be two cases here.

This case is wider or narrower than the maximum width s_{max} of the valley we set. (Here, $s_{max} = 90^{\circ}$ is set.)

In the valley, when the sector close to the robot's destination is k_n and the far sector is k_f, if $\left|k_n - k_f > s_{max}\right|$, the far sector is limited to $k_f = k_n \pm s_{max}$. Steering direction is calculated as follows.

$$\theta = (\frac{k_n + k_f}{2})\alpha \tag{5.11}$$

Speed Control

The angular velocity of the robot is calculated in proportion to the difference between the robot's current direction (θ_R) and the direction the robot wants to go (θ).

$$\Omega = K_{\Omega}(\theta_R \ominus \theta) \tag{5.12}$$

Here, K_{Ω} is a proportionality constant, and the operator calculates the shortest angular difference between the two angles to have a value between $[-180^{\circ} \sim 180^{\circ}]$.

The forward speed of the robot is limited by obstacles in front of the robot. That is, it follows the value $h_c^{'}$ of the sector of the smoothed polar obstacle density in the direction the robot wants to proceed (Steering direction; θ).

$$V' = V_{max}(1 - \frac{h_c^{''}}{h_m}) \tag{5.13}$$

$$h_c^{''} = min(h_c^{'}, h_m) \tag{5.14}$$

where mh is an empirically determined constant. However, $h_c^{''} < h_m$ must be satisfied. The forward speed is once again limited by the angular velocity Ω of the robot.

$$V = V'(1 - \frac{\Omega}{\Omega_{max}}) + V_{min} \tag{5.15}$$

Ω_{max} - Maximum angular velocity of the robot
V_{min} - Minimum forward speed of the robot
V_{max} - Maximum forward speed of the robot

Advantages and Drawbacks

The Vector Field Histogram overcomes some of the limitations exhibited by the potential field methods. In fact, the influence of bad sensor measurements is minimized because sensorial data is averaged out onto an histogram grid that is further processed. Moreover, instability in travelling down a corridor, present when using the potential field method, is eliminated because

the polar histogram varies only slightly between readings. In the VFH there is no repulsive nor attractive forces and thus the robot cannot be trapped in a local minima, because VFH only tries to drive the robot through the best possible valley, regardless if it leads away from the target. Enhanced versions of the VHF method are presented in [20] and [21].

5.2.2 Global Path Planning

RRT* Algorithm

This subsection briefly introduce motion planning using the RRT* algorithm to build the background for understanding RRT.

The Rapidly-exploring Random Tree algorithm, based on incremental sampling, efficiently computes motion plans. Although the RRT algorithm quickly produces candidate feasible solutions, it tends to converge to a solution that is far from optimal.

In addition, RRT* proposed based on the map generated by HECTOR SLAM (simultaneous localization and mapping), a 2-D map generation algorithm and simultaneous localization using the relative position of the unmanned aerial vehicle and the acquired obstacle data.
Let x define the configuration space in which x_{obs} is the obstacle region, $x_{free} = x/x_{obs}$ is the obstacle-free region and x_{goal} is the goal region.
RRT* works to find out an input $u : [0 : T] \in U$ that yields a feasible path $x(t) \in x_{free}$ that starts from $x(0) = x - initial$ to $x(T) = goal$ following the system constraints. While finding this solution, RRT* maintains a tree $T = (V, E)$ of vertices V sampled from the obstacle-free state space x_{free} and edges E that connect these vertices together. This algorithm makes use of a set of procedures which are explained as below :

Sampling: It randomly samples a state $z_{rand} \in x_{free}$ from the obstacle-free configuration space.

Distance: This function returns the cost of the path between two states assuming the region between them is obstacle free. The cost is in terms of Euclidean distance.

Nearest Neighbor: The function Nearest (T, z_{rand}) returns the nearest node from $= (V, E)$ to z_{rand} in terms of the cost determined by the distance function.

Steer: The function Steer $(z_{rand}, z_{nearest})$ solves for a control input $u[0, T]$ that drives the system from $x(0) = z_{rand}$ to $x(T) = z_{nearest}$ along the path $x : [0, T] \rightarrow x$ giving z_{new} at a distance Δq from $z_{nearest}$ towards z_{rand} where Δq is the incremental distance.

Collision Check: The function $obstaclefree(x)$ determines whether a path $x : [0, T]$ lies in the obstacle-free region x_{free} for all $t = 0$ to $t = T$.

Near-by Vertices: The function $Near(T, z_{rand}, n)$ returns the nearby neighboring nodes that lie in a ball of volume $((logn/n))$ around z_{rand}, where β is a constant that depends on the planner.

Insert node: The function $Insertnode(z_{parent}, z_{new}, T)$ adds a node z_{new} to V in the tree $T = (V, E)$ and connects it to an already existing node z_{parent} as its parent, and adds this edge to E.

A cost is assigned to z_{new} which is equal to the cost of its parent plus the Euclidean cost returned by the Distance function between z_{new} and its parent z_{parent}.

Rewire: The function $Rewire(T, z_{near}, z_{min}, z_{new})$ checks if the cost to the nodes in z_{near} is less through z_{new} as compared to their older costs. If it is for a particular node, its parent z_{parent} is changed to z_{new}.

Pseudocode describing RRT* is shown in Algorithm 1.

$T \leftarrow InitializeTree();$
$T \leftarrow InsertNode(0, z_{init}, T);$
for $i = 0$ *to* $i = N$ **do**
    $z_{rand} \leftarrow Sample(i);$
    $z_{nearest} \leftarrow Nearest(T, z_{rand});$
    $(x_{new}, u_{new}, T_{new}) \leftarrow Steer(z_{nearest}, z_{rand})$
    if $Obstaclefree(x_{new})$ **then**
        $z_{near} \leftarrow Near(T, z_{new}, |V|);$
        $z_{min} \leftarrow Chooseparent(z_{near}, z_{nearest}, z_{new}, x_{new};);$
        $T \leftarrow InsertNode(z_{min}, z_{new}, T);$
        $T \leftarrow Rewire(T, z_{near}, z_{min}, z_{new});$
        return T

At first, a sample z_{rand} is placed randomly in the configuration space x_{free}. Then, the nearest node $z_{nearest}$ to z_{rand} is checked for in the entire configuration space. A node z_{new} is placed at a distance Δq from the nearest node $z_{nearest}$ in the line of direction of z_{rand}. Then, the trajectory path x_{new} is checked if it is free of obstacles. If the trajectory is obstacle free then a ball of radius $\beta(logn/n)$ around z_{new} is checked for near nodes z_{near}. Among this set of nodes, the node that gives the least cost from the starting point to z_{new} through itself is selected as the parent of z_{new}. Once the parent is selected, rewiring takes place. The costs of all the nodes inside this ball around z_{new} is calculated through z_{new}. If this cost is less than the previous cost for any node then that particular node is disconnected from its old parent and is connected to z_{new} as its parent.

RRT* is a landmark sampling based algorithm to approach an optimal solution ensuring asymptotic optimality, apart from probabilistic completeness, as opposed to its predecessor RRT (and its various other improved versions). Although it tends to approach an optimal solution but it has been proven mathematically that it reaches the said solution in infinite time [47].

The previously developed RRT* is an algorithm that finds the movement path from the initial state of the robot to the target state.

Path Optimization

Once RRT* gives an initial path, the nodes in the path x: $[z_{init}, z_{goal}] \rightarrow x$ that are visible to each other are directly connected. An iterative process starts from z_{goal} and moves towards z_{init} checking for direct connections with successive parents of each node until the collision free condition fails. By the end of this process, no more directly connectable nodes are present. Hence the path is optimized based on the concept of the Triangular Inequality as illustrated in Figure 5.7.

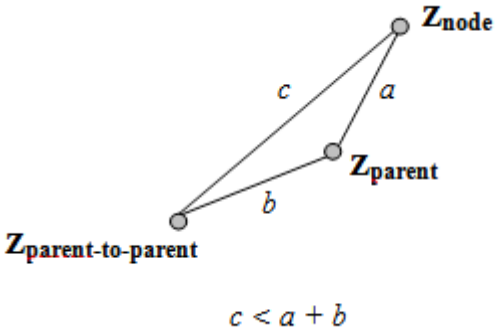


Figure 5.7: Path Optimization based on Triangular Inequality.

1: **while do**$(!\,visible\ nodes)$;
2: $z_{vc} = z_{goal}$;
3: **while do**$(!\,z_{init})$;
4: **if** $Obstacle free(z_{vc}, z_{vc-parent-to-parent})$ **then**
5: $z_{vc-parent} = z_{vc-parent-to-parent}$;
6: $z_{vc} = z_{cv-parent}$;

According to the Triangular Inequality, c is always less than the sum of a and b, and hence always gives a shorter path. The proposed path optimization is elaborated in Algorithm 3, where, z_{vc} is a visibilty check node, which is used for notation in the psuedocode.

The number of nodes present in this path are thus reduced, as compared to the original path found by RRT*. These nodes are termed as Beacons $(z_{beacons})$, which form the basis for intelligent sampling.

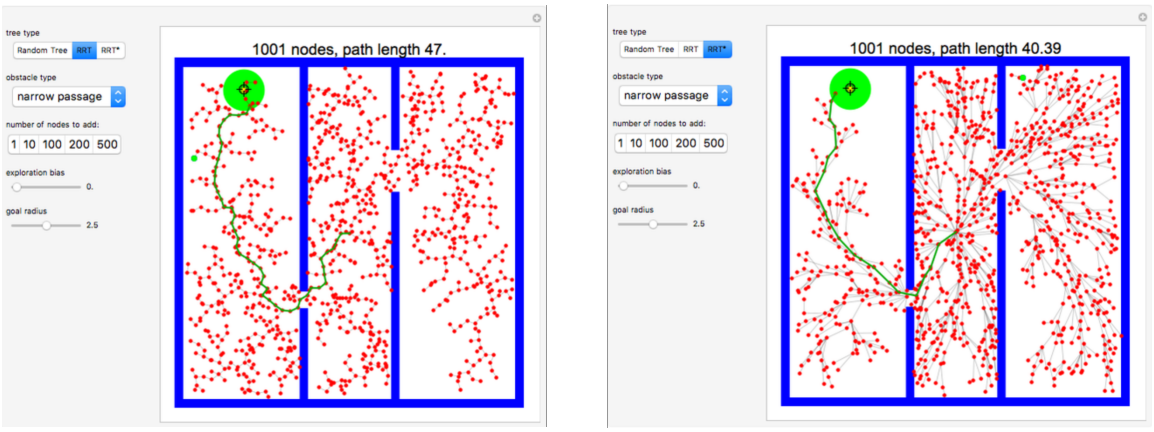
Every time a new RRT* path with a shorter cost is found, it is optimized by the path optimization technique to give a relatively better path. The cost of this optimized path is compared with the cost of the previous optimized path. If the cost is better, then the nodes that are present in this path are selected as beacons $z_{beacons}$, for intelligent sampling.

At each visibility check between two nodes, the collision free check is required. For this purpose, an interpolation technique is utilized in the proposed method, which works by constructing every point on the line (by connecting all the nodes together), while making sure that the newly added points lie in the free configuration space. This method of visibility check does not need explicit information about obstacles as required in other collision checking methods, [48]. Hence, the proposed method of interpolation for collision free checking is independent of the shape of the obstacles and is computationally less expensive.

Intelligent Sampling

The idea behind intelligent sampling is to approach optimality by generating the nodes as close as possible to the obstacle vertices following the underlining idea of visibility graph technique. However, the visibility graph techniques require complex environmental modeling and explicit information about obstacles, [49]. Furthermore the basic visibility graph method may not reach a solution in environments containing obstacles with complex geometries (concave, polygonal, circular etc). Some solutions in this regard have been presented like Reduced Visibility Graphs [50], which improve the efficiency of visibility graphs. Whereas, Generalised Visibility Graphs

[50], are an extension of the basic technique that form paths around intricate complex geometries. However, in doing so, the computational complexity for Generalised Visibility Graphs is increased to a great extent as it find solution for all the complex gemotroies present in the configuration space.
Once the initial path has been found, intelligent sampling starts with a certain number of samples being directly spawned in a ball of radius $R_{beacons}$ centered at $z_{beacons}$. The sampling is biased towards these beacons because they provide useful clues regarding the position of obstacle vertices (or periphery in the case of circular obstacles). Therefore, these beacons need to be surrounded by maximum nodes to optimize the path at these turns. This feature forces the proposed algorithm to reach the optimal solution in less number of iterations, as compared to RRT*, as later demonstrated in the experimental results section of this paper.
As the algorithm iterates an optimized path is calculated, when ever a new RRT* path is found.. The cost of this new optimized path is compared with the previous optimized path. If the cost is smaller, new beacons, $z_{beacons}$, are generated resulting in the formation of new biasing points. These newly formed beacons are closer to the vertices. This process continues until the required iterations are completed.
Although the obstacles are not being explicitly defined by keeping the beneficial property of sampling based algorithms intact; the proposed algorithm finds a way to spawn the tree nearer to the vertices by using intelligent guessing and biasing beacons, which eventually leads towards an optimal/near-optimal path $x : [z_{init}, z_{goal}] \rightarrow$ optimal x. This path also has a very few number of samples. Hence, the RRT* algorithm works to provide a much better solution at a faster rate of convergence. The proposed algorithm also finds a simpler path for the mobile robots to follow in any kind of environment due to the less number of waypoints.



(a) RRT. (b) RRT*.

Figure 5.8: Comparison of the intelligent sampling-based path planning in obstacle environments.

Figure 5.8 RRT* is an optimized version of RRT. When the number of nodes approaches infinity, the RRT* algorithm will deliver the shortest possible path to the goal. While realistically unfeasible, this statement suggests that the algorithm does work to develop a shortest path. The basic principle of RRT* is the same as RRT, but two key additions to the algorithm result in significantly different results.

6 System Development

This chapter is divided in two parts: an introduction with a brief explanation of the information about the hardware and the software used to complete this project is presented.

6.1 Software

With all this information to be the basics for understanding the implementation, software suites were selected that will be the major role in achieving the thesis research goals.

6.1.1 ROS and Gazebo

As for computers, even robots need an operating system to work, and for this project the chosen one was ROS. ROS (Robot operating system) is an open-source framework for development and programming of robots, which provides the necessary services for the robot, including hardware abstraction, low-level device control, message passing between processes, package management and so forth. The version of ROS used for this project is ROS Melodic.
The ROS runtime is based on a peer-to-peer network of processes (quite independent of each other), which communicate using the ROS infrastructure.This infrastructure implements different kinds of communication, such assynchronous communication over services or asynchronous streaming of data over topics.
Instead, in order to simulate the system, the robotic simulator chosen was Gazebo. Gazebo is a 3D dynamic open-source simulator which can efficiently simulate different kinds of robots in complex environments. This simulator has a lot in common with game engines (indeed it offers a 3D visualization very similar to them), but offering a much higher fidelity for the physics engine, as well as for the simulation of the sensors involved. Indeed, a key feature which makes Gazebo great for robotics simulations is the fact that it uses multiple high-performance physics engines.
To obtain the integration of ROS and Gazebo, a set of ROS packages *(named gazebo_ros_pkgs)*

are used. These packages provide the necessary interfaces to simulate a robot in Gazebo using the ROS infrastructures.

6.1.2 Autopilot PX4 and MAVROS

PX4 is a great platform to implement a UAV system based on a opensource autopilot. One of the great features of PX4 is that we can run a SITL simulation(Software in the loop simulation) to simulate your flight on simulation. This is useful as we can check new mission or control algorithms before actually flying the quadrotor and possibly damaging it.

The "brain" of the UAV is called autopilot. The concept derives from the autopilots used in airplanes, which allow the pilot to fly without continuous hands-on commands of the airplane. In the case of UAVs, instead, it consists of a firmware which is a collection of guidance, navigation and control algorithms running on the controller hardware, which allow the UAV for example to be stable during the flight or to understand how to run the motors in order to get to a particular target position.
The autopilot used in this project is PX4, a powerful open-source autopilot by Pixhawk. This autopilot offers many useful features, such as the possibility to be used with different vehicles frames/types (multicopters, fixed wing aircrafts, VTOLs, etc.) and also many flight modes and safety features useful for the good end of the flights.
Then, a ground control station called QGroundControl is present in order to obtain real-time information from the UAV during the flight, for example about the status of the vehicle (armed, disarmed, flight modes, etc.). More-over, it is used also to setup the vehicle and to change different flight parameters.
In order to communicate with the ROS framework and/or with the Gazebo Simulator, PX4 uses the MAVROS node. MAVROS is the official supported bridge between ROS and the MAVLink protocol, which is a lightweight messaging protocol for communication with UAVs and between the on-board UAV components.MAVLink uses a publish-subscribe model for data streams (using topics) and a point-to-point pattern for sub-protocols such as the mission protocol or parameter protocol. MAVLink messages are defined in XML files, then MAVLink uses these files to generate MAVLink libraries for the respective programming language used. Then, the UAV and the ground control station (or in the case of the simulation the Gazebo simulator) use these libraries to communicate with each other.

6.1.3 Octomap and Rviz

In order to create the 3D model of the map in real-time, the algorithm exploits a framework called Octomap [51].
Robotic systems usually works with noise of the measurements and with variable environments, hence in these cases a probabilistic interpretation of the occupancy of the volume is needed in order to avoid possible fatal errors. At the same time, a discrete interpretation (meaning that the volume elements are classified as occupied, free or unmapped) is convenient and allows to reduce the computational load.
Octomap combines both these approaches. It uses a tree-based approach and performs a probabilistic occupancy estimation of the area to map. The tree-based approach is based on octree [52], which is a hierarchical structure for spatial subdivision in three dimensions. Each node of

this structure represents a small cubic volume of the space, and these cubes are called voxels. Each voxel is divided recursively in eight until the minimum voxel size is reached, which is actually the resolution of the octree.

6.2 QGroundControl

QGroundControl (QGC) is an intuitive and powerful ground control station (GCS) for UAVs. The primary goal of QGC is ease of use for both first time and professional users. It provides full flight control and mission planning for any MAVLink enabled drone, and vehicle setup for both PX4 and ArduPilot powered UAVs. Instructions for using QGroundControl are provided in the User Manual.

Figure 6.1 shows a screen capture of QGroundControl, a well-known GCS software that can run on both personal computers and mobile devices. The screen capture shows the drone s position and orientation in a map, its followed path, some telemetry data, and a navigation instrumentation panel in the manner of a virtual cockpit.

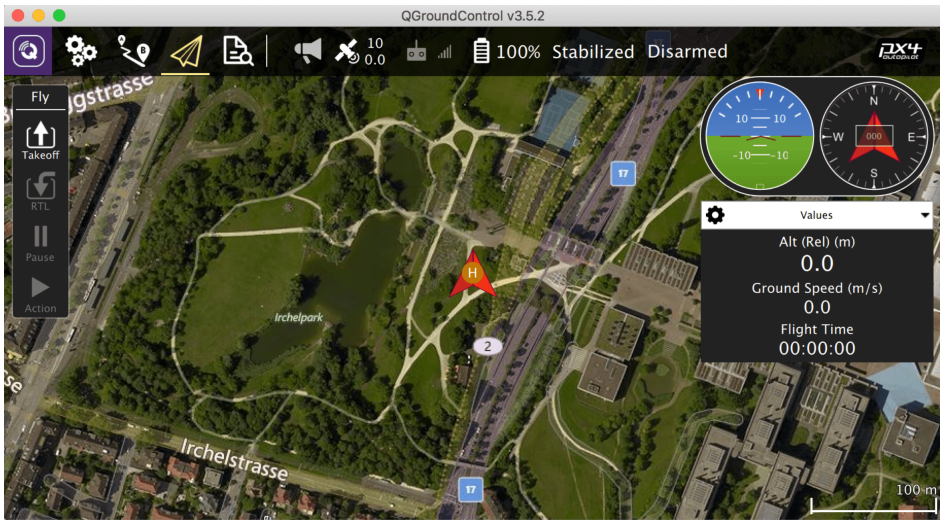


Figure 6.1: QGroundControl interface.

6.3 Hardware

This section provides information about Hardware options. The data below is sorted by manufacturer and product name.

6.3.1 The PIXHAWK and PX4 UAV System

Architecture The PixHawk flight controller is open-hardware microprocessor designed to implement autopilot solutions, and integrates two boards, PX4FMU (flight management unit) and PX4IO (input/output). Among the integrated sensors, it contains:

- L3GD20H gyroscope at 760 Hz

- LSM303D magnetometer at 100 Hz
- MPU-6000 accelerometer at 1000 Hz
- MS5611 barometer at 1000 Hz

PX4 implements data processing methods for navigation to estimate the vehicle attitude and cinematics:

a) Direction Cosine Matrix

Takes as input the triaxial accelerometers and gyroscopes data to obtain the attitude, usually represented as Direction Cosine Matrix (DCM) or the orientation parameters of the vehicle (roll, pitch, yaw angles).

b) Inertial Navigation System (INS)

This algorithm calculates the trajectories and corrections that allows the vehicle to move between single points using the DCM data, integrated to compute the vehicle attitude with high frequency.

c) Extended Kalman Filter (EKF)

The Px4 system counts with several Extended Kalman Filter algorithms to fuse sensor data in a function that exploits the specific noise and accuracy characterization of each sensor to estimate the vehicle navigation parameters. There are three different modes of EKF operation.

EKF1 : Only use the DCM for attitude control and the Inertial navigation for position prediction (dead reckoning);

EKF2 : Use the GPS for 3D velocity and position. The GPS altitude can be used if barometer data is very noisy;

EKF3 : If there is no GPS, it can use optical flow to estimate 3D velocity and position;

This work explores the possibility of extending data processing by include data from a LiDAR sensor in order to implement obstacle avoidance as a function working in parallel with the flight management unit, sending appropriate commands when a dangerous situation is detected.

Possibility of SIL and HIL design. Pixhawk supports SIL and HIL simulation using Gazebo [53].

Gazebo allows you to debug navigation and object traversal algorithms in the PixHawk flight controller without using any real device. In addition, Gazebo offers various models of real autonomous vehicles, which saves simulation time. If Gazebo does not offer it, it will be necessary to use the time and resources for correct physical modeling , so that the modeling conditions are as close to reality as possible.

UAVs that fly in unknown envinroments are required to be capable of obstacle detection and path planning to navigate a path free of obstacles. Seeing as GPS is not a valid option for indoor loitering, the quadrotor is equipped with LiDAR, an exteroceptive sensor, which is the input device for reading envinronment to maintain position.

The system developed in this word integrates the IRIS model of the real UAV previously use to execute missions. From this point, the proposal develops a 2D LiDAR sensor simulation that is mounted on the top to evaluate its capacity for detecting obstacles.

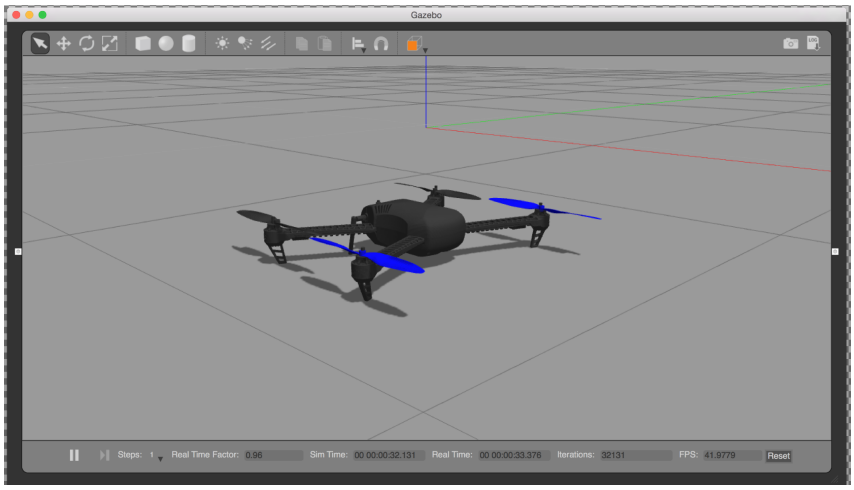


Figure 6.2: Model implementation in Gazebo.

In order to validate the 2D LIDAR model, some static trials have been performed with the vehicle in a fixed position on ground and mode than one obstacles.

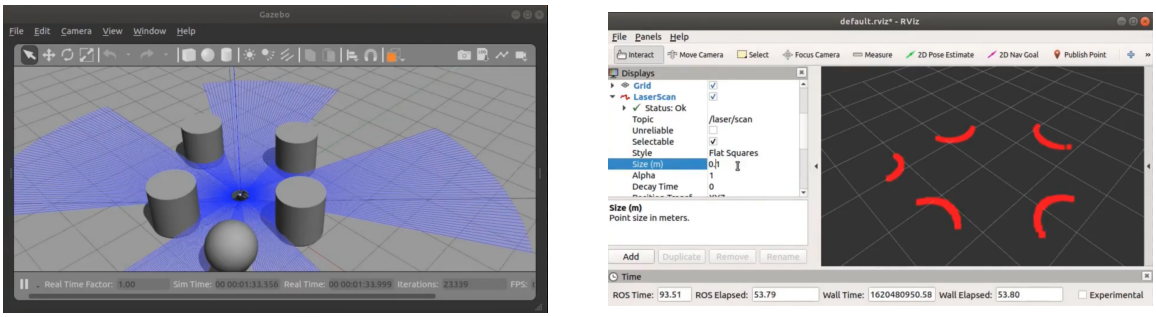


Figure 6.3: 2D LiDAR simulation world on Gazebo and RViz with obstacles.

Figure 6.3 shows the sample method to configure IRIS 2D LiDAR and visualize laser scan data in RViz. Simulation and multicopter firmware are based on Gazebo and PX4 SITL platform.

7 Simulation and Experiments

7.1 Testing Environment and Simulations

This section shows results of the experiments carried out to evaluate RRT* and VHF algorithms, next presents the results of simulations carried out to test the developed collision avoidance system. The VFH algorithm has been evaluated for its requirements: it must avoid obstacles in a 3D environment and perform these calculations in real time. The selected obstacle detection sensor has been modeled in order to be tested and verify its effective performance in this specific application.

The 2D LiDAR model implemented in Gazebo is connected with PX4 and sends simulated data (distances and angles). Tests are defined as a mission without obstacle and then the same mission with obstacles, putting the drone in several situations where it must turn in one or more objects.

7.1.1 Simulation stategy

Scenario 1 - Without obstacles

The first series of simulations developed are carried out in an obstacle-free environment. The most important factor is to check the collision avoidance system - to make sure that it works. In fact, it has been determined to come into action by detecting obstacles in the sensor's field of view. When no obstacle is detected, the collision avoidance algorithm must not redefine the new path, and therefore the planned mission must be fully executed in accordance with the trajectory determined during the planning phase of the mission.

The purpose of this simulation is therefore to observe the realization of the mission without any change of trajectory made by the collision avoidance system.

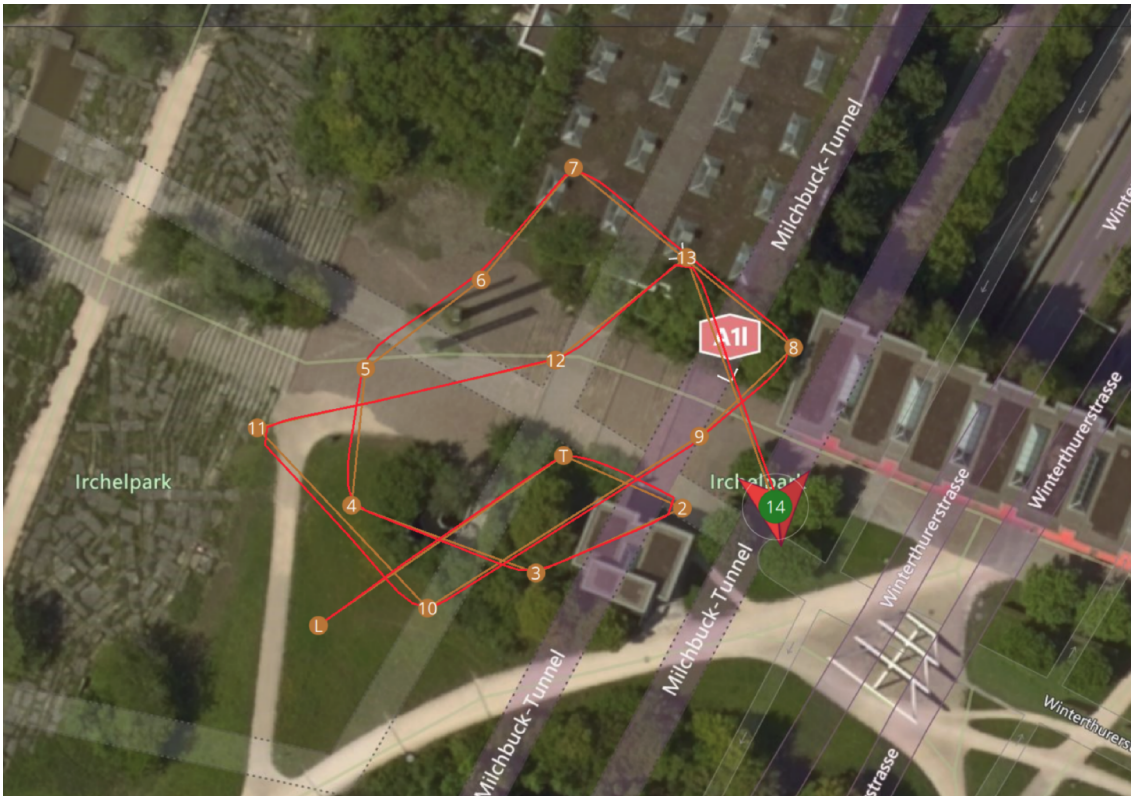


Figure 7.1: Scenario 1 - Mission planning and flight visualization on QGC

The (Figure 7.1) shows the QGC screen during the flight. The mission has been designed by inserting several waypoints connected by the yellow line.

The second set of simulations has been carried out in the environment with obstacles. It show that the algorithm is capable of effectively avoiding obstacles given the type of obstacles that are expected to be encountered in an outdoor delivery scenario.

Then, the waypoints positioning has been defined to simulate a monitoring mission, waypoints plan has been uploaded by the QGC, according to the project's provisions. For this reason, the UAV flight was planned along a non-linear trajectory in order to test the dynamics of the drone in three-dimensional space during flight.

The figure shows the QGC screen during flight. The mission was developed by adding multiple waypoints connected by a yellow line.

After that, the mission is loaded and ready for execution if the UAV is set to Mission flight mode or Offboard flight mode. Different parameters such as altitude and flight speed can be set for each waypoints.At the competition, the locations with the stationary obstacles were given before take-off. The developed path planning successfully found a path which avoided all obstacles and achieved all necessary waypoints.

Scenario 2 - With obstacles

Figure above shows the route designed in the mission planning phase (yellow) and the route actually taken by the UAV (red) thanks to the use of the active collision avoidance system. Figure above shows the route designed in the mission planning phase (yellow) and the route

actually taken by the UAV (red) thanks to the use of the active collision avoidance system. So, the basic principle of obstacle avoidance is to detect obstacles and figure out the distances between the UAV and obstacles. When an obstacle is getting closer, the UAV is supposed to avoid or turn around under the instructions of obstacle avoidance module as depicted in Figure 7.2. The basic principle of obstacle avoidance is to detect obstacles and figure out the distances between the UAV and obstacles. When an obstacle is getting closer, the UAV is supposed to avoid or turn around under the instructions of obstacle avoidance module as depicted in Figure 7.2.

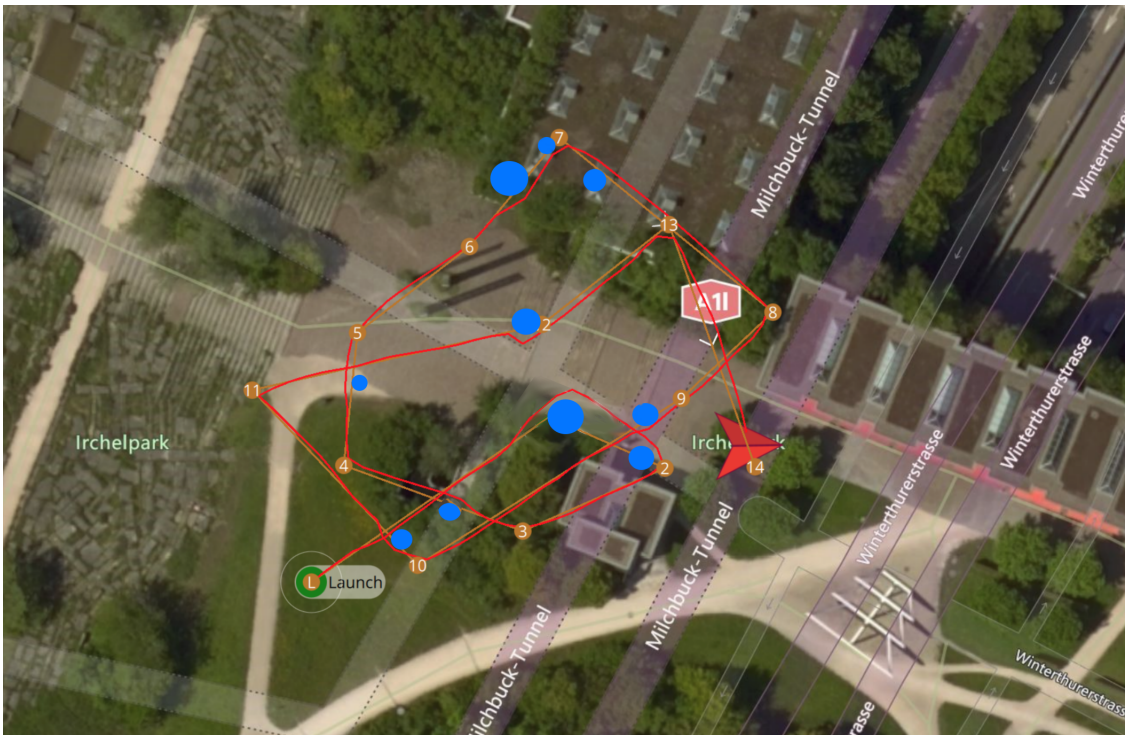


Figure 7.2: Scenario 2 results - Obstacle avoidance trajectory variations

Visualization of path planning of quadrotor from one point to other is shown below. Gazebo and Rviz visualizations are shown below. In Rviz, green one is orginal trajectory and red one is the jerk minimal trajectory. The planning time is restricted to just 5 seconds. When more time is allowed, the path obtained gets shorter and more smooth as well. The testing is done through a node which plots the waypoints, obstacles and the current pose of UAV on RVIZ for examining the accuracy of the algorithm.

In addition, the unmanned aerial vehicle autonomous flight system verified in the Gazebo simulation was verified through actual outdoor flight. Figure 7.3 shows an unmanned aerial vehicle equipped with a 2-D LiDAR and a simulator of an obstacle environment.

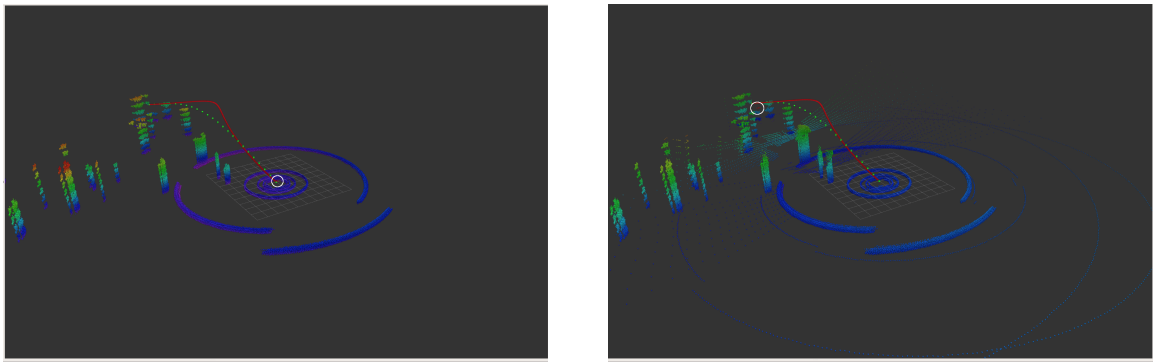


Figure 7.3: 2D LiDAR simulation world on Gazebo and RViz with obstacles.

Figure 7.3 show the global path created using the RRT* algorithm of the local path (green line) by the new proximity fixed obstacle. Subfigures (a) serve as a starting point and (b) appear to indicate the target conditions under which the drone was reached. Then Figure shows the regenerated global path (red line) according to the local avoidance path planning flight.

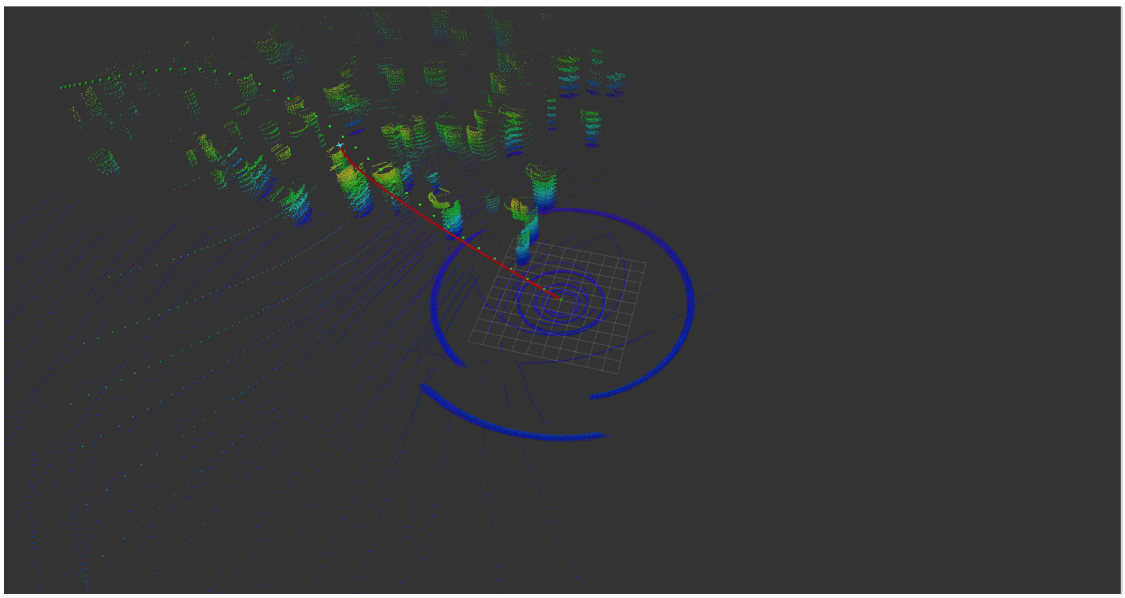


Figure 7.4: Local Avoidance Path Generation and Global Avoidance Path Regeneration by Proximity Obstacle

Figure 7.5 shows the HECTOR SLAM map using Rviz, a 3-D visualization tool.

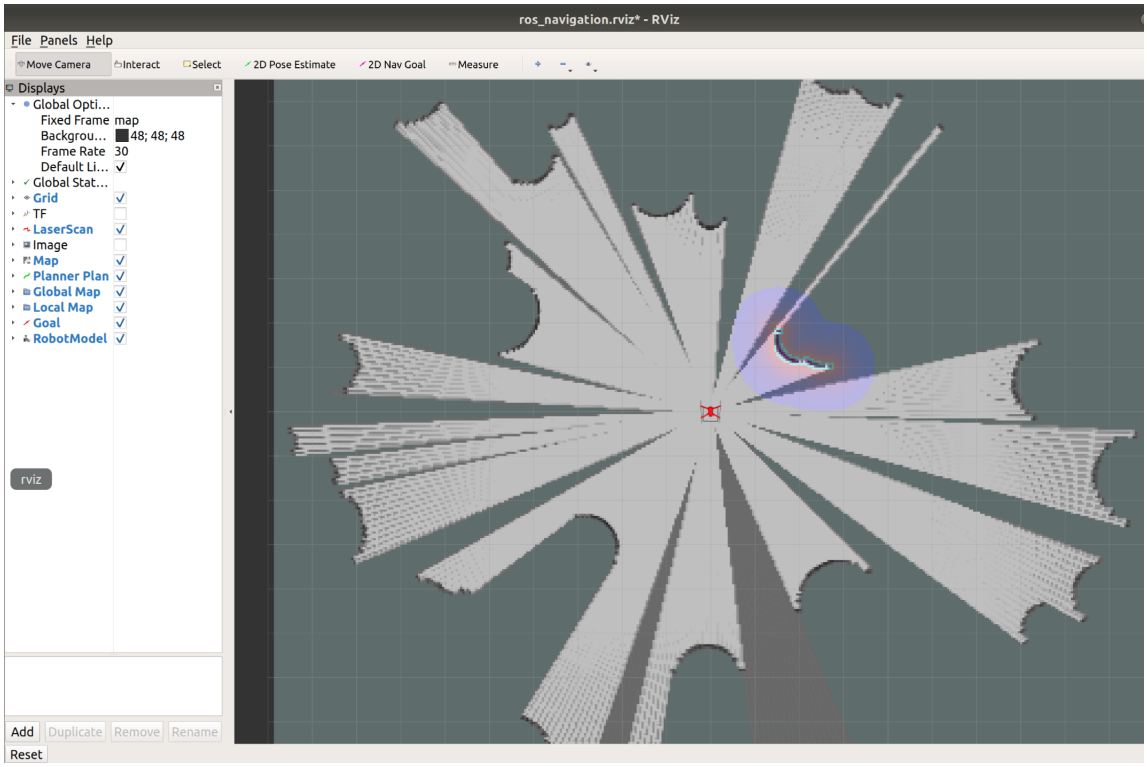


Figure 7.5: Rviz 3-D Visualization Tool.

Raspberry Pi receives real-time location and status of Pixhawk flight controller using MAVROS-based communication protocol, creates VFH local route and RRT* global path plannig using the acquired obstacle information, and transmits movement control signals accordingly do.

A real-time histogram was generated with obstacle information obtained through 2-D Lidar, and a global path using RRT* was created based on the VFH-based local path and the 2-D map generated using HECTOR SLAM.

Setting Goal in the RVIZ Window

First, estimate the initial Pose i.e locating the real robot location with respect to the Map. This can be set in the RVIZ window itself using the 2D Pose Estimate and pointing and dragging the arrow in the current robots location and orientation.

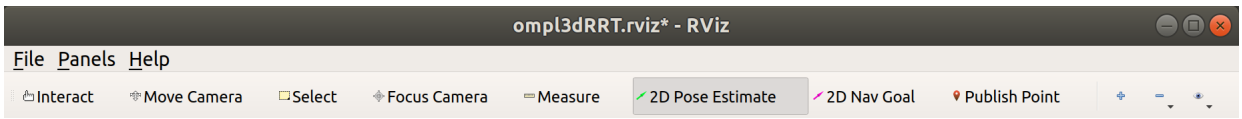


Figure 7.6: 2D Estimate Tab

A GOAL point can be set in the RVIZ window itself using the 2D Nav Goal option which will be available in the top window tab. This allows you to set a goal point in the map within the RVIZ environment, then the robot automatically performs the path planning and starts to move in its path.

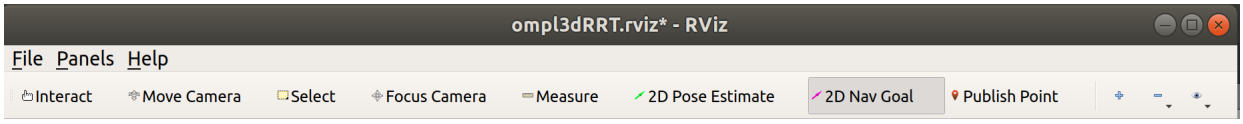


Figure 7.7: 2D Nav Goal Tab

It is able to utilize ROS node commands such as rostopic echo and rostopic info to understand how different navigation sensors could be displayed in the interface, depicted in (Figure 7.8)

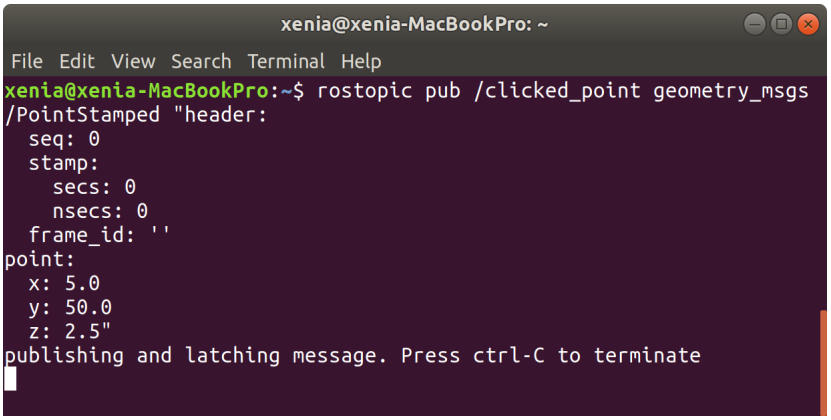


Figure 7.8: Rostopic info and echo demonstration

7.1.2 Result Discussion and Analysis

7.1.3 Discussion

The two step planning approach of global planning and local planning was presented. Tests are defined as a mission without obstacle and then the same mission with obstacles, putting the drone in several situations where it must. As a result of the comparative experiment, it was confirmed that an efficient global path was generated with less computation and distance cost to generate the global path. First, the proposed autonomous flight system was verified through the 3-D space GAZEBO simulator, and finally, the actual flight was verified by installing an outdoor artificial obstacle. By simultaneously operating VFH and RRT*, it was confirmed that autonomous flight is possible even when the entire terrain of the terrain is not input.

Although the RRT* and VFH algorithms were successfully planned paths to avoid stationary obstacles, many additional features can be added to improve obstacle avoidance. Some of these ideas are presented here:

- Plan straight line or filleted paths, to improve waypoint accuracy. To account for straight line path transitions, obstacles could be inflated significantly.
- Flight boundaries can be handled dynamically rather than as stationary obstacles. Despite the fact that the developed algorithm planned trajectories to bypass the flight boundaries, the UAV flew out of the bounds during the competition due to an error following the trajectory. One solution to this problem is a dynamic vector field approach that takes the UAV away from the boundaries.

- Functionality can be added to the algorithm to avoid dynamic obstacles.

8 Conclusion

This thesis was devoted to the development and simulation of a path planning system for a drone capable of performing static obstacle detection and collision avoidance during accomplishment of autonomous missions. Obstacle avoidance and path planning are essential for the UAV to safely and quickly get to the target location without collision. Besides, UAV navigation requires a global or local 3D map of the environment. Extra dimension means greater computation and storage consumption. So there is a great challenge when a UAV is navigating in a large scale environment for a long time.

Several design steps were performed to obtain a complete system that could be tested in a simulation environment. The adopted solution is based on the RRT* (Rapidly-exploring Random Tree star) and the VFH (Vector Field Histogram) algorithms to provide the global and local paths, respectively with performance suitable for the specific application. While 2D-LiDAR a avoidance was performed so as to avoid the obstacle using proposed method. In this thesis we also studied various state of art techniques of obstacle avoidance for UAV and fundamentals, however, more experimental research is needed.

The chosen method conveyed the search for an algorithm that would allow the method to be applied on a hardware and on a simulated system. Propose real-time path planning algorithm, with ability to detect and avoiding static obstacle. 2-D LiDAR is used to detected obstacles in real-time to generate the histogram for local path planning and a SLAM map used for global avoidance path generation trajectory to reach the target point. The algorithmic tools presented in this project show that path planning and obstacle avoidance research techniques have reached a level of maturity that allow their can transfer onto real platforms. Using the Gazebo as a simulator based on ROS platform, it was shown that proximity obstacle avoidance based on the chosen algorithms. In the future, due to the characteristics of the unmanned aerial vehicle flying in 3-D space, it is expected that research will be needed to create a 3-D global route based on the 3-D map and generate local and global path to fly.

Hence the result of this research was based on this new approach for obstacle detection and the output was gathered as a proof of concept to show that this is very simple and effective

approach in the field of obstacle avoidance in UAVs.

Future work can be done to improve stationary obstacle avoidance and to avoid dynamic obstacles.

Bibliography

[1] **William W. Greenwood, Jerome P. Lynch, and Dimitrios Zekkos**. Applications of uavs in civil infrastructure. *Journal of Infrastructure Systems, 2019*. 12

[2] **S. Bouabdallah**. Design and control of quadrotors with application to autonomous flying. *M.S. thesis, 2007*. 24

[3] **W. R. Beard**. Quadrotor dynamics and control. *M.S. thesis, Dept. Sci and Eng., EPFL, 2007*. 24

[4] **P. Corke**. Robotics, Vision and Control. *Springer, 2013*. 24, 50

[5] **A. G. Sidea, R. Y. Brogaard, N. A. Andersen and O. Ravn**. General model and control of an n rotor helicopter. *J. Phys.: Conf., 2014*. 24

[6] **T. Bresciani**. Modelling, identification and control of a quadrotor helicopter. *M.S. thesis, Dept. Automatic Control, Lund Univ, 2008*. 24

[7] **A. R. Partovi, H. Lin, G. Cai, B. Chen, and A. Z. Y. Kevin**. Development of a cross style quadrotor. *AIAA Guidance, Navigation, and Control Conference, 2012*. 24

[8] **Dashkevich, A.; Rosokha, S.; Vorontsova, D,**. Simulation Tool for the Drone Trajectory Planning Based on Genetic Algorithm Approach. *Proceedings of the 2020 IEEE KhPI Week on Advanced Technology, 2020*. 18

[9] **Tomas Lozano-Perez, M. Wesley**. An algorithm for planning collision-free paths among polyhedral obstacles. *Computer Science, 1979*. 19

[10] **Jean-Daniel Boissonnat, Camille Wormser, Mariette Yvinec**. Curved Voronoi diagrams. *Effective Computational Geometry for Curves and Surfaces, 2007*. 19

[11] **N. H. Sleumer and N. Tschichold-Grman**. Exact Cell Decomposition of Arrangements Used for Path Planning in Robotics. *Technical Reports, 1999*. 19

[12] **S. Ge, Y. Cui**. Dynamic Motion Planning for Mobile Robots Using Potential Field Method. *Computer Science, 2002*. 19

[13] **Johann Borenstein, Yoram Koren**. The Vector Field Histogram - Fast Obstacle Avoidance For Mobile Robots. *IEEE Transactions on Robotics and Automation, 1991*. 5, 19, 21, 59, 60, 61, 62, 63

[14] **S. LaValle**. Rapidly-exploring random trees : a new tool for path planning *The annual research report,1998*. 20

[15] **Oussama Khatib**. Artificial Inteligence Laboratory. *Stanford University, 1983*. 20

[16] **F. Janabi-Sharifi, D. Vinke**. Integration of the artificial potential field approach with simulated annealing for robot path planning. *Proceedings of 8th IEEE International Symposium on Intelligent Control, 1993*. 20

[17] **Park, M.G., Lee, M.C.**. A new technique to escape local minimum in artificial potential field based path planning. *KSME International Journal, 2003*. 20

[18] **Mahdi Fakoor; Amirreza Kosari; Mohsen Jafarzadeh**. Revision on fuzzy artificial potential field for humanoid robot path planning in unknown environment. *International Journal of Advanced Mechatronic Systems, 2005*. 20

[19] **Park, Min and Jeon, Jae and Lee, Min Cheol**. Obstacle avoidance for mobile robots using artificial potential field approach with simulated annealing. *Industrial Electronics, 2001*. 21

[20] **Iwan Ulrich, Johann Borenstein**. VFH+: reliable obstacle avoidance for fast mobile robots. *IEEE International Conference on Robotics and Automation, 1991*. 21, 65

[21] **Iwan Ulrich, Johann Borenstein**. VFH*: Local Obstacle Avoidance with Look-Ahead Verification. *IEEE International Conference on Robotics and Automation, 2000*. 21, 65

[22] **Edsger W. Dijkstra**. A note on two problems in connexion with graphs. *Numerische Mathematik, 1959*. 21

[23] **Mehlhorn and P. Sanders**. Algorithms and Data Structures. *Computer Science Software Engineering, 2008*. 21

[24] **Daniel Delling**. Engineering route planning algorithms. *Algorithmics of large and complex networks, 2009*. 22

[25] **W. Zeng, R. Church**. Revision on fuzzy artificial potential field for humanoid robot path planning in unknown environment. *International Journal of Advanced Mechatronic Systems, 2005*. 22

[26] **A. Stentz**. The Focussed D* Algorithm for Real-Time Replanning. *Computer Science, 1995*. 22

[27] **Hart, P., Nilsson, N. and Raphael, B.**. A Formal Basis for the Heuristic Determination of Minimum Cost Paths. *Systems Science and Cybernetics, 1968*. 22

[28] **Sven Koenig, Maxim Likhachev, David Furcy**. Lifelong Planning A*. *Artificial Intelligence, 2004*. 22, 23

[29] **Mohammadreza Radmanes, Manish Kumar, Paul H. Guentert and Mohammad Sarim**. Overview of Path-Planning and Obstacle Avoidance Algorithms for UAVs: A Comparative Study. *Unmanned Systems, 2018*. 22

[30] **Sven Koenig, Maxim Likhachev**. D* Lite. *Unmanned Systems, 2003*. 22

[31] **S. LaValle**. Rapidly-exploring random trees : a new tool for path planning. *The annual research report, 1998*. 23

[32] **Steven M. Lavalle, James Kuffner**. Rapidly-Exploring Random Trees: Progress and Prospects. *Algorithmic and computational robotics: New directions, 2000*. 23

[33] **Hasan, Osman**. RRT*-Smart: A rapid convergence implementation of RRT*. *International Journal of Advanced Robotic Systems, 2011*. 23

[34] **Fahad Islam**. Rrt-smart: Rapid convergence implementation of RRT* towards optimal solution. *IEEE International Conference on Mechatronics and Automation (ICMA), 2012*. 23

[35] **Yi, K.; Gu, F.; Yang, L.; He, Y.; Han, J.**. Sliding model control for a quadrotor slung load system. *Proceedingsof the Chinese Control Conference, 2017*. 31

[36] **Guo, Y.; Jiang, B.; Zhang, Y.**. A novel robust attitude control for quadrotor aircraft subject to actuator faultsand wind gusts. *IEEE/CAA J. Autom. Sin., 2018*. 31

[37] **E. Fresk, and G. Nikolakopoulos**. Full Quaternion Based Attitude Control for a Quadrotor. *European Control Conference, 2013*. 41

[38] **J. Diebel**. Representing Attitude: Euler Angles, Unit Quaternions, and Rotation Vectors. *European Control Conference, 2013*. 41

[39] **R. W. Beard**. Quadrotor dynamics and control. *Brigham Young University, 2008*. 44, 50

[40] **J. W. Hager, J. F. Behensky, and B. W. Drew**. The universal grids: Universal transverse-mercator (utm) and universal polar stereographic (ups). *Defence mapping Agency Hydrographic/Topographic Center Washington DC, 1989*. 53

[41] **S. Kaynama and C. J. Tomlin**. Benchmark: Flight envelope protection in autonomous quadro-tors. *Defence mapping Agency Hydrographic/Topographic Center Washington DC, 1989*. 48

[42] **R. Leishman, J. Macdonald, R. Beard, and T. McLain**. Quadrotors and accelerometers: Stateestimation with an improved dynamic model *Control Systems, IEEE, 2014*. 49

[43] **G. Frehse, C. Le Guernic, A. Donz e, S. Cotton**. G. Frehse, C. Le Guernic, A. Donz e, S. Cotton *Springer, 2011*. 49

[44] **J. H. Blackelock**. Automatic Control of Aircraft and Missiles, 2nd ed. *John Wiley and Sons, 1991*. 49

[45] **H. Moravec and A. Elfes**. High resolution maps from wide angle sonar. *Proceedings of the IEEE International Conference on Robotics and Automation, 1985*. 59

[46] **A. Elfes**. Sonar-based real-world mapping and navigation. *IEEE Journal of Robotics and Automation, 1987*. 59

[47] **S. Karaman and E. Frazzoli**. Sampling-based Algorithms for Optimal Motion Planning. *International Journal of Robotics Research, 2011*. 66

[48] **T. Bialkowski, S. Karaman, and E. Frazzoli**. Massively Parallelizing the RRT and the RRT*. *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2011*. 67

[49] **I. Petrovic and M. Brezak**. A visibility graph based method for path planning in dynamic environments. *Electronics and Microelectronics (MIPRO), 2011*. 67

[50] **Latombe, J.C.**. Robot Motion Planning. *Springer, 1990*. 67, 68

[51] **Hornung, Armin**. OctoMap: an efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots, 2013*. 70

[52] **D. Meaghe**. Geometric modeling using octree encoding *Computer Graphics and Image Processing, 1982*. 70

[53] **L. Meier**. PX4: A node-based multithreaded open source robotics framework for deeply embedded platforms. *IEEE International Conference on Robotics and Automation (ICRA), 2015*. 72