

POLITECNICO DI TORINO

Master's Degree in Mechatronic Engineering



**Politecnico
di Torino**

Master's Degree Thesis

Development and implementation of an obstacle avoidance algorithm for an Unmanned Aerial Vehicle

Supervisors

Prof. ALESSANDRO RIZZO

Prof. MARINA INDRI

Dr. STEFANO PRIMATESTA

Adv. ALESSANDRO MANCINELLI

Candidate

DAVIDE GRAZIATO

December 2021

Abstract

The forthcoming of the fourth industrial revolution drives the implementation of autonomous systems and high collaborative robots inside the industrial environments. Flexibility and Adaptivity represent the keys to satisfy the increase of an always faster and low-cost market demands. The FIXIT project main objective is to provide an interactive support for the human operator in the industrial or logistic environment, fitting the requirements of the industry 4.0.

The aim of the thesis is to develop and implement a collision avoidance system for an UAV. The drone must not be intended as a single entity but must be able to cooperate with an AGV, which constitute the landing base. The UAV's architecture is based on an open-source flight controller unit (FCU), the Pixhawk model 2.4.8, a companion computer, the Jetson Nano, and the intel Realsense D435i depth camera as main sensor.

The Collision avoidance system designed is based on a path planner which implement an Informed-RRT* algorithm (Informed-Rapid-Exploring-Tree "Star") to produce obstacles-free paths and avoid collision in both known static and unknown dynamics environments. The point clouds produced by the depth-camera are used to extract information from the surroundings of the drone and subsequently, define a 3D occupancy map. In order to validate the obtained results, simulation in virtual generated environment using Gazebo and real-life tests are performed. Both the simulation and experimental results convincingly demonstrate how, the implementation of this strategy, allows the UAV to generate a safe path, preventing in this way any collision, and reaching the desired target position.

Acknowledgements

*Grazie a mia madre e a mio padre,
per avermi reso la persona che sono
ed avermi sempre spronato ad inseguire i miei sogni.
Grazie ad Elisa, sempre pronta a consigliarmi
e ad aiutarmi nei momenti difficili.
Infine un grazie a tutti i miei amici,
per avermi insegnato il vero significato della parola amicizia.*

Chapters Index

List of Tables	VI
List of Figures	VII
Acronyms	X
1 Introduction	2
2 Review of Collision Avoidance methodologies and techniques for Unmanned Aerial Vehicles	5
2.1 Path Planning Algorithms	7
2.2 Graph Traversal Planner Algorithms	9
2.2.1 Dijkstra's Algorithm	10
2.2.2 A* Algorithm	12
2.2.3 D* - Dynamic A* search	14
2.2.4 LPA*	15
2.2.5 RRT - Rapid Exploring Tree	16
2.2.6 RRT* - Rapid Exploring Tree Star	18
2.2.7 Informed-RRT*	21
2.3 Other Path Planning approaches	24
2.3.1 APF - Artificial Potential Field	24
2.3.2 VFH - Vector Field Histogram	26
2.3.3 Machine Learning Approaches	28
3 Hardware Implementation	29
3.1 Sensors for Collision Avoidance	29
3.2 Intel Realsense D435i	33
3.3 Companion Computer	36
3.3.1 Raspberry Pi 4	36
3.3.2 Nvidia Jetson Nano	37
3.4 Flight Controller - Pixhawk 2.4.8	38

3.5	Outdoor Flight Sensors	39
3.6	Indoor Flight Sensors	41
3.6.1	DWM1001C - Ultra-Wideband	41
3.6.2	VL53L1X Tof sensor	42
4	Software Implementation	43
4.1	Robotic Operating System - ROS	43
4.1.1	Nodes	43
4.1.2	Topics	44
4.1.3	Services	44
4.1.4	Messages	44
4.2	Gazebo	45
4.3	Mavros	45
4.3.1	PointCloud Library - PCL	46
4.3.2	OMPL - Open Motion Planning Library	47
4.3.3	FCL - Flexible Collision Library	48
4.4	Robot Localization in Indoor Environment	49
4.4.1	Visual Odometry - VO	49
4.4.2	IMU	50
4.4.3	UWB and VL53L1X Tof	50
4.4.4	Robot Localization Package	50
4.5	Octomap 3D occupancy Map	51
4.6	EKF - Ardupilot's Parameters tuning	53
4.7	Algorithm implementation	56
5	Virtual Simulation	61
5.1	SITL and Mavros implementation	61
5.2	Virtual Environments for Testing	63
5.3	Simulated Flight Analysis	65
6	Experimental tests and results	70
6.1	Outdoor Flight	70
6.2	Indoor Flight	75
7	Conclusions and future Developments	76

List of Tables

3.1	Sensors selection for the CAS	33
3.2	How the Resolution of the Camera modify the minimum depth . . .	34

List of Figures

1.1	FIXIT UAV developed by the Team	3
2.1	Main path planning algorithm for UAV	9
2.2	Simple graph example	10
2.3	Example of a Graph with Nodes and Vertices with relative cost	11
2.4	Graphical representation of the paths generated by RRT algorithm	18
2.5	Rewiring process in RRT* algorithm	19
2.6	Graphical representation of the paths generated by RRT* algorithm	20
2.7	Ellipsoid Sub-space used in Informed RRT* algorithm	21
2.8	Graphical comparison between RRT* and Informed-RRT*	22
2.9	Visualization of an application of Artificial Potential Field as path planner	25
2.10	Vector Field Histogram	27
3.1	Different types of sensors and approaches to the Collision Avoidance problem	30
3.2	Graphical representation of the Cameras FOV	34
3.3	Intel RealSense D435i model	35
3.4	Single Board Computer Raspberry Pi 4 (4GB)	36
3.5	Single Board Computer Nvidia Jetson Nano(4GB)	37
3.6	Pixhawk 2.4.8 Flight Controller	38
3.7	Taoglass ZED-F9P GPS	40
3.8	Decawave DWM1001 Development Board	41
3.9	Indoor Cage anchors configuration were the drone was tested	42
4.1	Visualization of the UAV in Gazebo from the CAD render	45
4.2	Filtering action on the Poincloud	47
4.3	Robot Localization scheme	51
4.4	Octrees used in order to build the 3D Octomap	52
4.5	3D Octomap representation of a Tree in the Gazebo virtual environment	52

4.6	Relative Position of the UAV in Outdoor condition, the Red Circle is a Radius of 10 centimeters.	55
4.7	Relative Position of the UAV in Outdoor condition	55
4.8	Topic and Messages published in the Planner part of the Algorithm	57
4.9	Drone object seen in Rviz as a box	58
4.10	Rosgraph of all the Nodes implemented	60
4.11	Reference Frames dependencies	60
5.1	SITL console with Mavros node launched	62
5.2	Two moving persons in front of the UAV during the flight	63
5.3	Map greatly populated of multiple obstacles	64
5.4	The CIM4.0's Digital Line replicated in Gazebo	64
5.5	UAV's relative altitude in time during the Simulation	65
5.6	Path and Waypoints during a mission	66
5.7	UAV's Yaw angle during a mission	67
5.8	UAV's Roll angle during a mission	68
5.9	UAV's Pitch angle during a mission	68
5.10	The Planner is able to generate trajectory through narrow passages	69
5.11	Re-planning process of a given initial path	69
6.1	UAV's relative Z position	71
6.2	UAV's relative XY position	72
6.3	Rviz showing the Octomap and Tf-frame in real-time	72
6.4	UAV's trajectory and waypoints during a flight test	73
6.5	Goal Point sent thanks to Rviz	74
6.6	Multiple GPS tests to examine the accuracy of the position estimation. In Red the actual path while in Blue the position obtained through the GPS data.	74

Acronyms

UAV

Unmanned Aerial Vehicle

AGV

Automated Guided Vehicle

SITL

Software in the Loop

CAS

Collision Avoidance System

FCU

Flight Controller Unit

CC

Companion Computer

ROS

Robot Operating System

UWB

Ultra-wideband

ToF

Time-of-Flight

GPS

Global Positioning System

RTK

Real-Time Kinematic

RRT

Rapid Exploring Tree

LPA*

Lifelong Planning A*

APF

Artificial Potential Field

VHF

Vector Field Histogram

VO

Visual Odometry

EKF

Extended Kalman Filter

IMU

Inertial Measurement Unit

OMPL

Open Motion Planning Library

PCL

Point Cloud Library

Chapter 1

Introduction

Drones are a valuable asset within industry and logistics and are now employed extensively to carry out inspections tasks, surveillance and mapping the terrain in harsh or hazardous environments.

To comprehend how they fit within an industrial ecosystem, it is possible to consider them as mobile sensors, able to complement all the emerging technology as IoT and Big Data. Not only that, the development and continuous increase of their power allows the movement of small objects and transport through different parts of a company.

The objective of the thesis is the development and the implementation of a collision avoidance algorithm for an autonomous unmanned aerial vehicle (UAV). Autonomous navigation cannot do without obstacle avoidance systems to meet the safety standards imposed in an industrial and logistical operating environment.

Thanks to CIM4.0, it was possible to work on the FIXIT project, which consists in a combination of an AMR and a UAV, aiming to simplify all maintenance operations. This idea is part of a bigger picture, where autonomous and collaborative robots constitute the basis for the growth of the Industry 4.0.

The initial UAV model used during the development phase was the DJI F450 quadcopter, an entry level open-source drone commercially available. Subsequently, thanks to the collaboration of several members of the FIXIT-team, the framework developed was exported over a custom-made drone model specifically designed for the project itself, which includes both a customize frame and power board (*figure 1.1*). As constraints during the development phase, all the software used had to be completely open-source. The deployment of the Pixhawk as flight controller unit (FCU) allows the implementation of two different open-source operating systems, Ardupilot and PX4. In this case, the first one was used due to the great customization provided and the integration with the ROS platform to control the drone autonomously. ROS represents the base framework which enables the communication between all the different functions in the developed algorithm, the

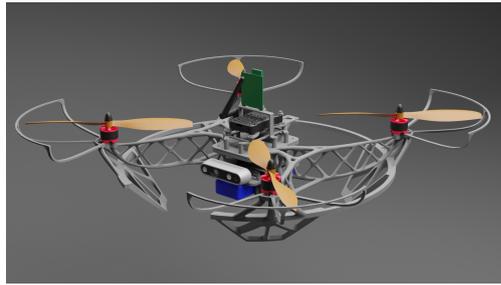


Figure 1.1: FIXIT UAV developed by the Team

FCU and the ground computer.

The challenges provided by this project were not only at software level but also at the hardware one. In order to improve initial performance, improvements were made to the electronics, batteries and on-board computer throughout the progress of the project, and these decisions are analyzed and described in detail in the following chapters. The choice of the most suitable sensors to achieve an effective obstacle tracking and maps generation, including an analysis on the sensors related to the drone's localisation for both indoor and outdoor environments. In this last part of the introduction, a schematic breakdown of the thesis is given with a small summary of the issues covered in each chapter:

- **Chapter 1:** Brief introduction on the FIXIT project and summary of the most important topics addressed during the development of the project.
- **Chapter 2:** In this first chapter the obstacle avoidance algorithm's state of art is analyzed. A number of possible solutions to the planning problem are described and their flaws highlighted. Subsequently, the one chosen for the thesis's application is reported and its basic functioning explained.
- **Chapter 3:** In this chapter all the hardware implemented in the UAV is listed and their basic functioning principles explained. Additionally, after a dissertation of the most widespread sensors for collision avoidance, the motivation for the one used in this thesis is given.
- **Chapter 4:** In the third chapter instead the analysis of all the software deployed in the creation of the script is listed. The different library and their configuration in order to work with the main collision avoidance algorithm are outlined.
- **Chapter 5:** Virtual simulation in order to determine the effectiveness of the implemented system. In particular, SITL and Gazebo are used to estimate the possible UAV's real behavior. The results of these simulations are examined and graphically represented.

- **Chapter 6:** Real-life testing of the algorithm and the drone's behavior during flight. In the same way as the previous chapter, the gathered data are graphically represented and analyzed to extrapolate useful information.
- **Chapter 7:** Conclusion and future developments are described. Possible improvements and considerations on the technology used.

Chapter 2

Review of Collision Avoidance methodologies and techniques for Unmanned Aerial Vehicles

In recent years the manufacturing and production industries are moving towards a gradually increasing automation of various operations and tasks – which will be performed by robots instead of human labors. In this class of automation robots both Unmanned Ground Vehicle (UGV) and Unmanned Aerial Vehicle (UAV) are included. While UGVs, whose mainly operate on the ground, have been widely used for manufacturing tasks like part-feeding and material handling, UAVs, which operate in the air, are emerging in various application domains such as surveillance, logistics, and search–rescue missions.

UAVs domain of applicability is not only limited to outdoor environments, UAVs can also be useful in indoor environments for manufacturing and services (hospitals, greenhouses, production companies, and nuclear power plant). Equipped with a camera sensor, UAV can be used for performing inspection and maintenance tasks in harsh environment, both visual and sensorial inspection. Equipped with a gripper, UAV can also be used for performing handling of light material in a manufacturing environment.

In addition to these applications, the employment of UAVs in indoor environment is also supported due to the superior freedom of maneuverability w.r.t. a ground vehicle, such as 360° inspection angle of an object in a three-dimensional space and moving in an empty upper-air space (which has been an idle area in the era of UGV).

Due to the limited area in indoor, there is less flexibility in dodging when a potential obstacle is found. Hence, flight during tasks execution must be scheduled in non-colliding paths. The risks of damage on both UAVs and resources is higher and human workers can be exposed to hazardous situations. Therefore, indoor UAV application has more constraints and precise controls to be taken into account. From these requirements, it is possible to deduce the key role represented by the path planning algorithm implemented in the UAVs. A valid path is generated by taking into account the mission constraints, vehicle characteristics, and the surrounding environment and combining all these elements with the mission task. The UAV class is important because different classes can have completely different dynamic and kinematic characteristics. The same flight path may be unreliable with another drone.

The basic concept of a Collision Avoidance System (CAS) involves monitoring the surroundings for any possible encounter, sense and detect possible threats and avoid them. In order to work efficiently and safely, a CAS is usually composed by five different key functions:

- **Sensing:** the UAV's ability to monitor the environment and collect appropriate current state information for any possible encounter, i.e. aircraft position, velocity and heading.
- **Detection:** system's ability to acquire the sensed data, process it and extract useful information discovering and managing incoming collision to the UAV.
- **Awareness:** generation of a map, graph or histogram containing the position of the obstacles found and the aircraft.
- **Escape trajectory:** based completely on the path planning algorithm implemented, it must be carefully chosen based on the environment and on the future tasks in which the UAV will be deployed.
- **Maneuver realizations:** function through which the moving command are sent to the flight controller to perform the required maneuvers in order to avoid the obstacles.

In the following part of the chapter an extensive review of the existing methodologies and techniques used in obstacles avoidance is performed. This analysis constitute a fundamental step in order to choose the best algorithm implementation for the case scenario of the project and subsequently the right sensors and hardware for the UAV.

2.1 Path Planning Algorithms

Collision avoidance systems are based on the implementation of a path planning algorithm. A path is specific route in the 3D space which a UAV can follow without colliding with obstacles. The route can be a smooth continuous curve generated by considering the dynamics of flying or, in the most simple and basic form a specific set of points through which the drone can move sequentially in a series of straight lines until reaching the end point. A planner needs to take in account the information about the physical environment, the position of the UAV and it's desired destination and extract a feasible and viable path using this data.

The development of path planning algorithms had been started by the attempt to solve the shortest path problem which is the problem of finding a path between two vertices (or nodes) in a graph such that the sum of the weights of its constituent edges is minimized. Initially focused in solving a 2D problem, as the technology progressed and the computational strength improved, were developed planners able to solve this problem in 3D space. The way this is achieved varies greatly based on the type of sensors and implementation, on how the space is mapped and how the walls and obstacles are considered, but generally the path planning problem boils down to represent the map as a weighted graph or occupancy grid that can be searched for a path between two vertices.

The problem of finding the most optimal path between two points in space minimizing the total cost path becomes extremely relevant for aircraft and drones. Drones and UAVs are usually small and can't carry to much weight in order to fly effectively, this constraint limits greatly the size of batteries that can be implemented on the aircraft. Planners able to generate paths closer to the optimum assume great relevance to fit the premise of conserve battery life in order to safely carry out the assigned mission.

Usually a Planner is composed by two main functions, with different purposes and based on different types of algorithm: a Global and a Local planner.

- **Global Path Planner:** A global path planner is a planner whose function is to generate a path between the Starting position and the Goal position. These kind of algorithms perform well in a known environment of which a map is already present. In a Static environment the performance of these algorithms are better than the local counterpart, but in presence of possible dynamic encounters possible collisions can happen.
- **Local Path Planner:** A local path planner is designed taking into account an unknown environment whereby dynamic obstacles can be present, the planner needs to be fast to elaborate the incoming data from the sensor and plan trajectory consequently.

In order for a planner to operate at full capacity, it must integrate these two functions simultaneously. Once the initial path produced by the global planner is defined and sent to the FCU through a list of waypoints, the local planner must intervene, if necessary, taking in account both the dynamic of the aircraft with its constraints and the information about the obstacles, by modifying the path consequently to avoid any collision.

So, to recalculate the path at a specific rate, the whole map is reduced to the surroundings of the aircraft and is updated as the vehicle is moving around. It is not possible to use the whole map because the sensors are unable to update the map in all regions, and a large number of cells would raise the computational cost, increasing the energy consumption and the possibility of a crash for the Companion Computer. Therefore, with the updated local map and the global waypoints, the local planning generates avoidance strategies for new static and dynamic obstacles and tries to match the trajectory as much as possible to the provided waypoints from the global planner.

In extreme synthesis, the standard implementation in a CAS is a Global path planner responsible for defining the initial route to the Goal point. Along it the Local planner operates in case an obstacle is found; when this happens, the local planner generates a new path avoiding the obstacle to continue the mission safely. In the **Figure 2.1** the main types of Path planners are represented in a scheme. It is possible to notice two different approaches, one for known environments and the second for unknown environments. The decision on which planner to choose requires a deep analysis on the operativity conditions of the drone and on the overall efficiency and computational cost due to the hardware used. In the following pages an extensive review of these methods is carried out.

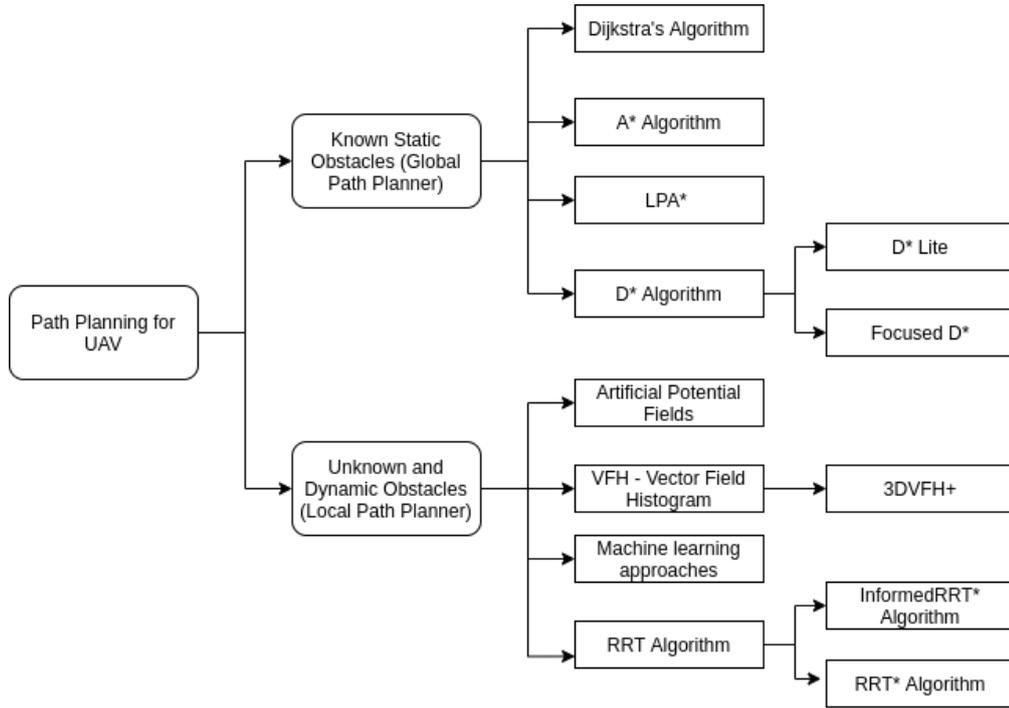


Figure 2.1: Main path planning algorithm for UAV

2.2 Graph Traversal Planner Algorithms

In order to understand how this kind of path-planning algorithms work is necessary to introduce some key-concepts about graph theory and the resolution of the shortest path problem.

In Graph theory a Vertex (or Node) is the fundamental unit of which graphs are formed: an undirected graph consists of a set of vertices and a set of edges (or Links) (unordered pairs of vertices), while a directed graph consists of a set of vertices and a set of arcs (ordered pairs of vertices).

From Graph's theory point of view, vertices are treated as featureless and indivisible objects, although they may have additional structure depending on the application. The two vertices forming an edge are said to be the endpoints of this edge, and the edge is said to be incident to the vertices. In the application of path planning the edges of a graph are usually associated with a given cost, which represents the distance of a given pair of nodes. The various algorithms, based on this methodologies, are distinguished by their ability to generate a path capable of connecting various nodes taking into account the cost of each vertex. How this is done sets the performance and efficiency of these systems by determining in which areas they are best suited for an autonomous implementation. In the subsequent

sections an extensive review of approaches of planner based on solving the graph search problem are analyzed.

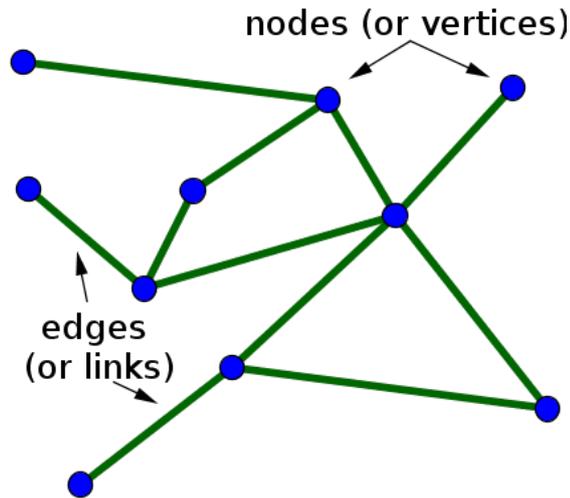


Figure 2.2: Simple graph example

2.2.1 Dijkstra's Algorithm

Dijkstra's Algorithm represents one of the first attempts to solve the shortest path problem. It was designed in 1956 by Edsger Dijkstra.

The algorithm achieves its goal by searching for all the possible paths between a starting position and the end of one path and ordering them by their cost. For a given source node, the algorithm finds the shortest path between that node and every other.

In the original implementation, the algorithm was mainly used to find a path between two points in the graph. But a common variant consists of fixing a single node as a "source node" and finding the shortest path from the source to all the other nodes in the graph, thus generating this way a shortest-path tree. Once the algorithm finds the shortest path between the source node and another node, that one is marked as "visited" and added to the path. The process continues until all nodes in the graph have been added to the path. This way, we have a path that connects the source node to all other nodes following the shortest path possible to reach each node. The algorithm also keeps track of the currently known shortest distance from each node to the source and can update these values if a shorter path is found.

Another application consists of stopping the algorithm immediately once the shortest path between a single node and a destination node has been determined, reducing this way the time spent in searching for all the possible paths. Dijkstra's

algorithm can work well as a basic path-finder algorithm for a whole set of nodes but, when used to search for the shortest path to one specific target, it can be extremely inefficient.

The main drawback for this type of algorithm resides in how it works. The exploration of large portions of the map, which are searched aimlessly and don't bring any contribution to finding the path to the goal point or improvements to the actual solution, increases the computational cost, time spent, and decreases the overall efficiency of the implementation. The reactivity of the drone is the main focus in order to obtain a reliable collision avoidance system, and the total time spent on finding the solution to the path problem needs to be low for re-planning operations in a dynamic environment. The use of this kind of algorithm for a UAV as a local planner has to be avoided due to the slow responsive time and large computational strain that reduces the autonomy of the aircraft.

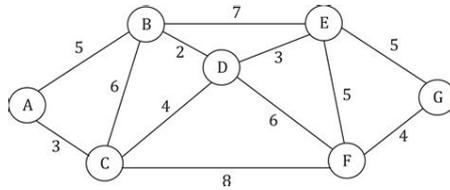


Figure 2.3: Example of a Graph with Nodes and Vertices with relative cost

2.2.2 A* Algorithm

A* Algorithm was developed and proposed by Hart, Nilsson and Raphael in 1968. It can be marked as an improvement on the Dijkstra's, due to the fact that it introduces a heuristic, consequently the algorithm gives a priority w.r.t. the distance to the goal point in addition to the length path. At a fundamental level, A* works as a Dijkstra's which tries to explore only the nodes relevant to the search of the path between two points.

Because of its simplicity, effectiveness and accuracy due to the heuristic A* algorithm is one of the most typical implementations in robotics to solve path-finding and graph traversal problems.

The A* is designed to follow the path which generates the lowest cost, to do that it preserves a sorted queue composed by all the different paths found that are useful when the robot has to change direction. In case an obstacle blocks the first direction, a new direction is found by calculating which of the new path minimizes the cost and shift to the lower path abandoning the previous one.

A* is based on the best-first search algorithm, this means it is a search algorithm which explores a graph by expanding the most promising node chosen according to a specified rule, in this case the least-cost path from two given points. At the base reasoning the algorithm tries to minimize the function:

$$f(x) = g(x) + h(x)$$

Where $f(x)$ is defined as "distance-plus-cost-heuristic" function, $g(x)$ the cost function and $h(x)$ is the heuristic function.

While $g(x)$ represents the cost from moving between the two points, the function $h(x)$ is an heuristic estimate of the distance to the goal point. The heuristic can be calculated in different ways, thus giving to the A* algorithm flexibility in its application, but the commonly employed heuristic for simplicity coincides with the straight-line distance to the goal. This heuristic function represents the shortest possible distance between two points

It is safe to assume that A* will always generate a valid path if possible, however the algorithm has no guarantee that the path found will be the optimal one. The generated path depends on different parameters. The resolution of the map used plays a major role, higher resolution will lead to have an higher number of vertices in the map, thus leading to better paths closer to the optimum but at the cost of longer computational time. Lower resolutions will produce paths way quicker, but at risk of producing paths way further from the optimum and closing off narrow passages, increasing in this way the energy consumption of the aircraft. Since the number of vertices in the map increases cubic to the increase of the resolution this is a major factor.

A lower resolution map will generate paths way quicker, but at risk of producing

paths way further from the optimum and closing off narrow passages that cannot be seen at too low resolution. Another obstacle that makes the path sub-optimal is the fact that A* operating in a square or cube grid is limited to moving in a set number of angles. This is solved in a variety of ways, usually by either checking for optimal parts in a small area at a time while the planner is running, or by post processing of the generated path.

In terms of performance, the main issue of the base A* algorithm is the fact that it is designed to solve the problem only once, to get an updated path it is necessary to run the algorithm continuously. This drawback may increase the computational load and overall slow down the process of path-finding in a highly dynamic environment due to the fact that many instances of the A* need to be executed. In large extent plain A* has been replaced by updated version of itself in order to avoid this problem.

The time complexity of this solution is linked to the chosen heuristic function, in the best case it has a polynomial trend if the search space is a tree, in the worst case it could be an exponential expansion of the nodes.

A* is an optimistic estimate of the cost of the path, in fact, the true cost of a path from the node to the goal will be at least as great as the estimate, and on the admissibility criteria which certified an optimal path thanks to an equal examination of all the nodes. However, there is also the possibility to modify the algorithm to find an approximated shortest path, in this way, it is possible to speed up the search at the expense of optimality by relaxing the admissibility criterion. Oftentimes we want to bound the relaxation criteria so that we can promise that the solution path is no worse than $(1 + \epsilon)$ times the optimal solution path.

2.2.3 D* - Dynamic A* search

D* was introduced in 1994 by Anthony Stentz, its behavior and basic functionality is directly derived by A*. It is still designed to find the minimum path in a graph but the main difference is that the procedure to search for the path starts from the goal point and go backward to the original point, choosing at each time step to less expensive arc until reaching the starting node.

The advancement w.r.t. the A* is that D* gives the possibility to update the path every time a change in the surrounding environment is found, as soon the cost arc is changed, an improved path is generated with a lower cost to avoid the obstacle. So, it is mandatory that the robot is able to update its map and re-plan optimal path each time a new information comes. For its qualities, the D* search algorithm, letting re-planning to occur incrementally and optimally in real time, is suitable for a partially known environment containing dynamic obstacles. However, this procedure of continuing updating and re-planning the paths is computationally onerous, compared to the A*, due to the fact that it tries to find a path not only from the goal, but also for all the nodes that are about as far from the target. For this reason, in some cases the A* has a higher efficiency, introducing the heuristic estimate of the distance between the start and the arrival, is able to limit the node that is analysed during the calculations.

In recent years, new iterations of the algorithm have been developed mainly expanding on the type of heuristic used in order to generate the path. Focused D* is an informed incremental heuristic search which tries to reduce the computational cost and speed up the process of path-planning. While in standard D* as in the A* algorithm the cost changes are propagated through the invalidated states without considering which expansions will benefit the robot at its current location, in this advanced implementation the search is focused so that new optimal paths are computed considering the robot's location w.r.t. the goal position. If, the robot then moves to a new location and its sensors discover another arc cost discrepancy the search should be focused on the robot's new location. Due to the advantages brought by Focused D*, this type of algorithm is replacing the standard D*.

Another variant is represented by Lite-D*, which is based on a combination of both incremental and heuristic search for its application. Similarly to the implementation of a LPA* algorithm, which is described in the following section, it repeatedly determines shortest paths between the current vertex of the robot and the goal vertex as the edge costs of a graph change while the robot moves towards the goal vertex it does not expand any vertices whose g-values were already equal to their respective goal distances. It does not make any assumptions about how the edge costs change, whether they go up or down, whether they change close to the current vertex of the robot or far away from it, or whether they change in the world or only because the robot revised its initial estimates. In this way the computational load

and the overall lightness of the algorithm is improved maintaining an efficiency equal to the standard D*.

2.2.4 LPA*

Lifelong Planning A* (LPA*) was first described by Sven Koenig and Maxim Likhachev in 2001 and it is an incremental version of A* which repeatedly tries to find shortest paths from a given start vertex to a given goal vertex while the edge costs of a graph change or vertices are added or deleted. This type of algorithm is implemented in those applications where the path-planning problem has to be solved in known finite graphs whose edge costs increase or decrease over time.

LPA* is basically an incremental version of A*, which can adapt to changes in the graph without recalculating the entire graph, given both a starting vertex $s_{\text{start}} \in S$ and a given goal $s_{\text{goal}} \in S$ the algorithm defines the function $g(s)$ (as heuristic) to denote the distance between them. All the known costs from the previous search are maintained and updated with the $g(s)$ function only when necessary. So, eventhough the first search has a behavior equal to that of a version of standard A*, expanding the same nodes in the same order and breaking ties in favor of vertices with smaller g-values, but many of the subsequent searches are potentially way faster because it reuses those parts of the previous search tree that are identical to the new one.

LPA* deals with this in the following way: when the occupation of a vertex changes, that is, a wall vertex is added or removed, the surrounding vertices are checked to see if the move distance from the start is consistent with those of the vertices that surround them. This means that vertices look at their neighbours to check if their data are up to date, or if they have been blocked off. For each vertex that was found to have changed the neighbours of that vertex is then checked, forming a wave that identifies all vertices whose data is now outdated, add them as potential targets for exploration again. So, only the outdated information are discarded, and the segments of the map which were explored stay that way for the next search, making the re-planning more efficient.

2.2.5 RRT - Rapid Exploring Tree

A different type of approach to the graph traversal problem is provided by the Rapid Exploring Tree algorithm family. While standard graph traversal (or graph search) is focused on visiting each vertex in a graph (checking and/or updating), the tree traversal (or tree search/walk) is a form of graph traversal which is focused on visiting (e.g. retrieving, updating, or deleting) each node in a tree data structure exactly once. Tree traversal is defined as a special case of the graph traversal problem.

RRTs were developed by Steven M. LaValle and James J. Kuffner Jr, they are particularly effective to handle problems with obstacles and differential constraints (nonholonomic and kinodynamic) and have been widely used in autonomous robotic motion planning. 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.

An RRT grows a tree rooted at the starting configuration by using random samples from the search space. As each sample is drawn, a connection is attempted between it and the nearest state in the tree. If the connection is feasible (passes entirely through free space and obeys any constraints), this results in the addition of the new state to the tree. With uniform sampling of the search space, the probability of expanding an existing state is proportional to the size of its Voronoi region. As the largest Voronoi regions belong to the states on the frontier of the search, this means that the tree preferentially expands towards large unsearched areas. On average, an RRT is constructed by iteratively breaking large Voronoi regions into smaller one.

The tree is constructed incrementally from samples drawn randomly from the search space, the obtained benefits are similar to those achieved by other successful randomized planning methods, and is inherently biased to grow towards large unsearched areas of the problem.

The basic reasoning of this type of algorithm family can be summarized by the following points:

- Start from a given point, which will be the *Source* of the tree for searching the surrounding space in order to define a path to the goal point.
- Randomly select a position in a given region. From current position and randomly selected point, find nearest node if present select that node as current node. Else create a new node at predefined step-size distance from current node.
- If the distance of a newly created node is less than a certain threshold to the destination node, we can end our search for the destination. And traverse

back from the last node to the source node.

- Structure of each node stores its immediate parent and list of immediate childrens. While travelling back from the destination we follow the parent node. Loop runs for some already defined number of iterations, controlling number of iteration more approximate path can be found, usually the greater is the number of iterations the more approximate to the ideal path would be.

This application is not restricted to a grid like the previous A* algorithm, which waste time in a direction-less search and probing every are frequently, instead RRT rapidly explores new portions of the state space and the distribution of vertex provided leads to a consistent behavior, RRT remains connected even if the number of edges is minimal.

Although the RRT algorithm can be considered stable and probabilistically complete under very general conditions, it is usually seen as a path planning module to incorporate with another existing planning system. Entire path planning algorithms can be constructed without requiring the ability to steer the system between two prescribed states, which greatly broadens the applicability of RRTs.

It is possible to demonstrate that, the cost of the best path generated by a standard implementation of RRT converges almost surely to a non-optimal value and the algorithm can be slow due to the fact that tries to search all the state space. For this reason, different improvement to the basic functioning of the algorithm were performed during the recent years, mainly focused on providing a faster convergence rate to the optimum, implementing different heuristic function to limit the area of search or combining multiple tree generated by the starting and goal point.

Algorithm 1: RRT Algorithm Pseudocode

```

1  $\mathcal{T}.init(x_{init});$  for  $k = 1$  to  $K$  do
2    $x_{rand} \leftarrow RANDOM\_STATE();$ 
3    $x_{near} \leftarrow NEAREST\_NEIGHBOR(x_{rand}, \mathcal{T});$ 
4    $u \leftarrow SELECT\_INPUT(x_{rand}, x_{near});$ 
5    $x_{new} \leftarrow NEW\_STATE(x_{near}, u, \Delta t);$ 
6    $\mathcal{T}.add\_vertex(x_{new});$ 
7    $\mathcal{T}.add\_edge(x_{near}, x_{new}, u);$ 
8 return  $\mathcal{T}$ 

```

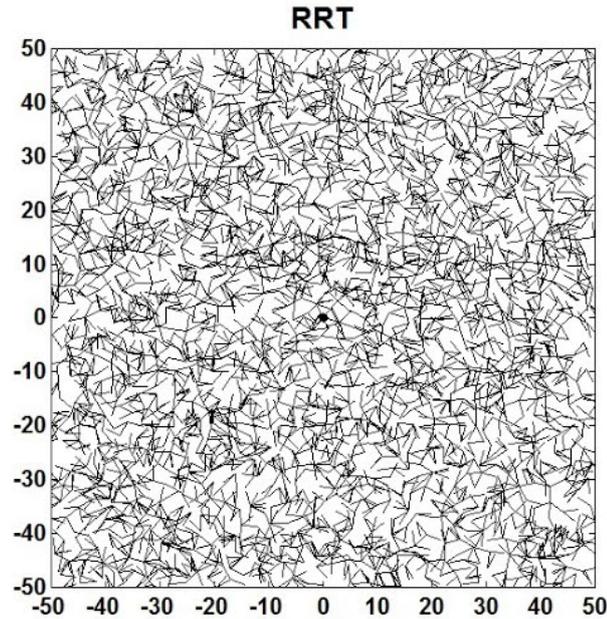


Figure 2.4: Graphical representation of the paths generated by RRT algorithm

2.2.6 RRT* - Rapid Exploring Tree Star

As mentioned in the previous section, improvements to the basic RRT algorithm have been made. The most famous variant is called RRT* (Rapid Exploring Tree-Star), which tries to obtain a path with a cost closer to the optimum as possible. While RRT* inherits all the properties of RRT and works similarly, it introduces two features called *near neighbor search* and *rewiring tree*.

As for the first function, with the neighbor search refers to the fact that to find the next node to be inserted in the tree, this choice takes place within a well-defined area, usually circular with a set radius.

The rewiring function instead, consists in changing the parent node if the cost of the previous connection can be decreased. As the number of iterations increases, RRT* keeps improving the solution and the path cost gradually comes close to the optimum, whereas RRT does not improve and maintains a jaggy-lines and sub-optimal path. Due to increased efficiency to get less jagged and shorter path, features of rewiring and neighbor search are being adapted in recent revisions of RRT*, therefore providing the basis for the development of new types of algorithms capable of improving the weaker aspects of it. In fact, while RRT* greatly improves the tree-based approach to the scheduling problem, because of its rewiring functions the convergence slow down if the space is large with higher number of dimensions. Another drawback consists in the uniform global sampling performed by the algorithm, this will provide asymptotically the optimal solution but at the cost of

slow time response in case of a dynamic obstacles or fast objects traversing the path.

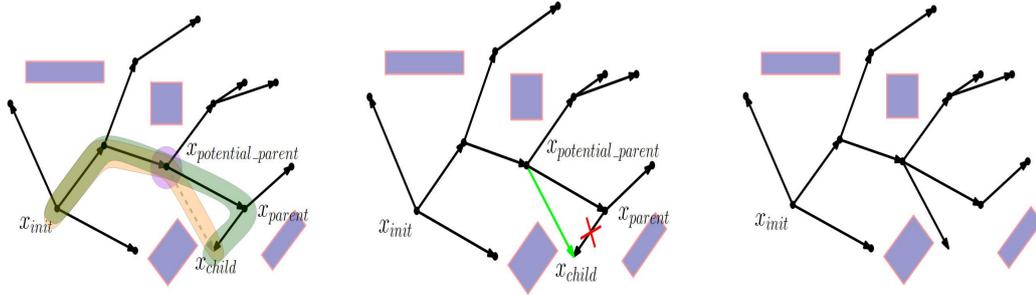


Figure 2.5: Rewiring process in RRT* algorithm

Algorithm 2: RRT* Algorithm Pseudocode

```

1  $\mathcal{T}.init(x_{init});$ 
2 for  $k = 1$  to  $K$  do
3    $x_{rand} \leftarrow RANDOM\_STATE(\chi);$ 
4    $x_{near} \leftarrow NEAREST\_NEIGHBOR(x_{rand}, \mathcal{T});$ 
5    $x_{new} \leftarrow EXTEND(x_{near}, x_{rand});$ 
6   if  $COLLISION\_FREE(x_{near}, x_{new})$  then
7      $\mathcal{T}.add\_vertex(x_{new});$ 
8      $\mathcal{T}.add\_edge(x_{near}, x_{new});$ 
9      $X_{near} \leftarrow nearest\_neighbors(\mathcal{T}, x_{new}, radius)$ 
10    for  $(x_{near}, X_{near})$  do
11       $rewire\_RRT * (x_{near}, x_{new});$ 
12 return  $\mathcal{T}$ 

```

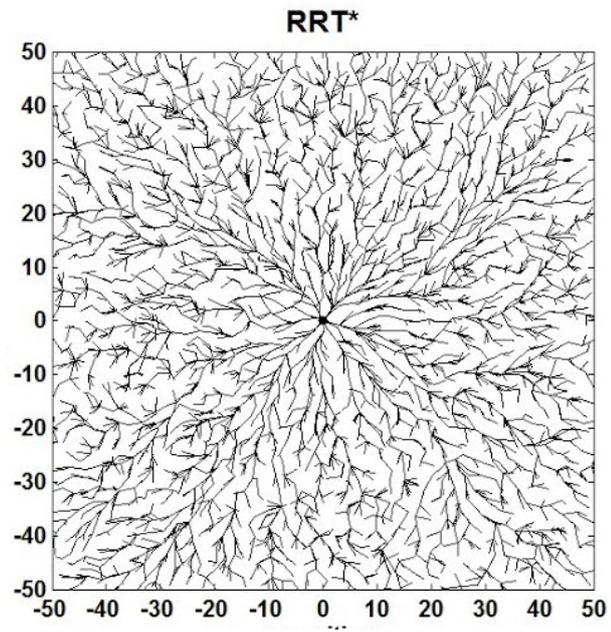


Figure 2.6: Graphical representation of the paths generated by RRT* algorithm

2.2.7 Informed-RRT*

As stated in previous section, the improvements on the RRT* algorithm are numerous and follow different strategies with the common aim of speeding up the process and improve the converging speed and reliability.

The Informed-RRT-Star algorithm was developed by Jonathan D. Gammell, Siddhartha S. Srinivasa, Timothy D. Barfoot and first published in 2014. The core of the algorithm resumes the basic functioning of the RRT* retaining the same probabilistic guarantees on completeness and optimality, but some changes have been made in order to significantly shorten the total search time. Furthermore, it can be shown experimentally that it outperforms RRT* in rate of convergence, final solution cost, and ability to find difficult passages while demonstrating less dependence on the state dimension and range of the planning problem.

Informed-RRT* behaves as RRT* until the first solution is found, after that the algorithm will start sampling only in an area shaped as an ellipsoid nearby the path (defined as $X_{\hat{f}}$). It can be shown that the optimal path resides inside this area. The subset $X_{\hat{f}}$ is where the algorithm will directly sample for new states, unlike path biasing no assumptions are made about the homotopy of the class of the optimum path and differently to the heuristic biasing it does not explore states that will not improve the solution.

Sampling only in the subset all the new states found are potentially improvements regardless from the size of $X_{\hat{f}}$ w.r.t. the initial X . This allows it to work effectively regardless of the size of the planning problem or the relative cost of the current solution to the theoretical minimum, unlike sample rejection and graph pruning methods.

The ellipse is built with both, the initial state, and the goal state as focal points, following the equation 2.1; the eccentricity is given by the ratio between actual best cost and the minimum one.

$$X_{\hat{f}} = \{x \in X \mid \|(x_{\text{start}} - x)\|_2 + \|(x - x_{\text{goal}})\|_2 \leq c_{\text{best}}\} \quad (2.1)$$

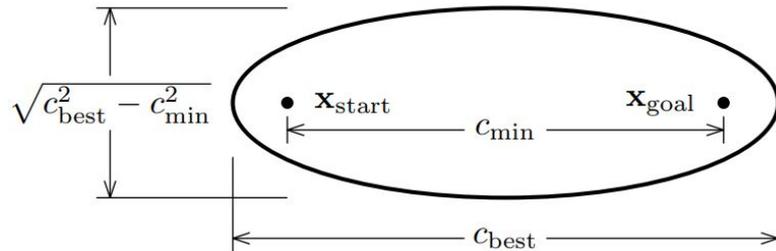


Figure 2.7: Ellipsoid Sub-space used in Informed RRT* algorithm

The creation of this subset implicitly balances both exploitation and exploration, no additional parameters or tuning are required. While the use of an heuristic may not always improve the search, in a real-world planning the practicality of this method is demonstrated; in situations where no additional information are provided by them, Informed RRT* is equal to RRT* but in any other case the improvements can't be neglected. In order to provide even better trajectories for the drone, in addition to the main algorithm, a Smoother can be implemented in combination to further reduce the search space.

As Informed RRT* uniformly samples the subset of the planning problem that can improve the solution, a rewiring radius can be calculated from the measure of this informed subset and the related vertices inside it. This updated radius reduces the amount of rewiring necessary and further improves the performance of Informed RRT*. At each iteration, the rewiring radius, r_{RRT^*} , must be large enough to guarantee almost-sure asymptotic convergence while being small enough to only generate a tractable number of rewiring candidates.

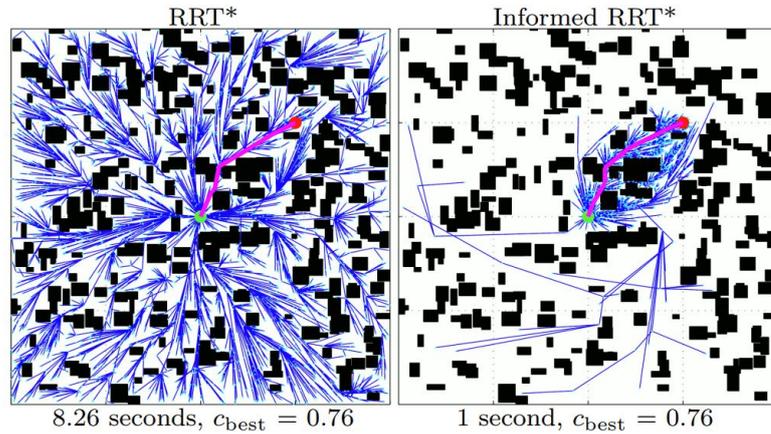


Figure 2.8: Graphical comparison between RRT* and Informed-RRT*

Algorithm 3: Informed RRT* Pseudocode given $(x_{\text{start}}, x_{\text{goal}})$

```

1  $V \leftarrow x_{\text{start}};$ 
2  $E \leftarrow \emptyset;$ 
3  $X_{\text{soln}} \leftarrow \emptyset;$ 
4  $\mathcal{T} = (V, E);$ 
5 for  $\forall x_{\text{near}} \in X_{\text{near}}$  do
6    $x_{\text{best}} \leftarrow \min_{x_{\text{soln}} \in X_{\text{soln}}} \text{Cost}(x_{\text{soln}});$ 
7    $x_{\text{rand}} \leftarrow \text{Sample}(x_{\text{start}}, x_{\text{goal}}, c_{\text{bests}});$ 
8    $x_{\text{nearest}} \leftarrow \text{Nearest}(\mathcal{T}, x_{\text{rand}});$ 
9    $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{rand}});$ 
10   $c_{\text{near}} \leftarrow \text{Cost}(x_{\text{near}});$ 
11   $c_{\text{new}} \leftarrow \text{Cost}(x_{\text{new}} + c \cdot \text{Line}(x_{\text{new}}, x_{\text{near}}));$ 
12  if  $c_{\text{new}} < c_{\text{near}}$  then
13    if  $\text{CollisionFree}(x_{\text{new}}, x_{\text{near}})$  then
14       $x_{\text{parent}} \leftarrow \text{Parent}(x_{\text{near}});$ 
15       $E \leftarrow E \setminus \{(x_{\text{parent}}, x_{\text{near}})\};$ 
16       $E \leftarrow E \cup \{(x_{\text{new}}, x_{\text{near}})\};$ 
17 if  $\text{InGoalRegion}(x_{\text{new}})$  then
18    $X_{\text{soln}} \leftarrow X_{\text{soln}} \cup \{x_{\text{new}}\};$ 
19 return  $(\mathcal{T});$ 
20

```

2.3 Other Path Planning approaches

All the methodologies analyzed in the previous sections are usually referred as global planner and useful in the definition of a set of waypoints to generate obstacles free trajectory for the drones given a starting and goal point.

However, when the aircraft found himself in a complete unknown environment without any previous information about the surrounding area the most part of these algorithm fail to generate a solution to the planning problem. For this kind of situations, local planner has been developed, while the basic concept is still that to provide a safe path for the UAV these planners have to be able to process the data coming from the sensors and elaborate escape trajectories in real-time.

Different type of approaches are subsequently analyzed in these last part of the chapter, which draw their inspiration from physical laws and image processing algorithm up to the most recent machine learning methods for UAVs.

2.3.1 APF - Artificial Potential Field

Artificial Potential Field (APF) is a methods first implemented in 2D application for an easy autonomous navigation implementation, then became very widespread for UAVs applications due to its properties and as a reactive collision avoidance.

At the base of this type of CAS, the navigation problem is expressed through physics concepts. Way-points and the Goal point are treated as attractive points with and attractive force while the obstacles generate repulsive one. Other types of forces are essentially arbitrary, such as potential energy; it is considered higher in points close to obstacles and lower in points near the way points. Therefore using simple electrostatic equations it is possible to define a safe trajectory; the one that has the lowest flux density becomes the new path for the UAV. Usually the Goal point express the lowest value possible in the potential field and so exerts the highest attractive force in order to move the UAV close to it.

Analysing the basic mathematical principles behind this method it is possible to define $q = (x, y)^T$ as the position of the drone in the state space. Through the equation 2.2 the potential field can be expressed as:

$$U(q) = U_{\text{att}}(q) + U_{\text{rep}}(q) \quad (2.2)$$

Where $U(q)$ is the APT, U_{att} the attractive field and U_{rep} the repulsive field. The attractive force is given by the negative of the gradient of the attractive field, while the repulsive force by the negative of the gradient of the repulsive field. The force moving the aircraft to the goal position can be expressed including the Euclidean Distance ρ from the goal point combining it with the gradient obtaining:

$$F_{\text{att}}(q) = -\nabla = \frac{1}{2}k_a\rho^2_{\text{goal}}(q) \quad (2.3)$$

However, there are drawbacks exist in APF such as: trap situation due to local minima or saddle points, oscillation in the presence of obstacles, no passage between closely spaced obstacles and oscillations in narrow passages. This can cause problems as the loss of control of the aircraft or collision threat. Another difficulty that can occur in practical application is that the dynamic limitations of the aircraft have to be considered. If this does not happen, the vehicle will not be able to fly the generated path. Moreover, considering the high importance of the availability of state information for this type of methods, any deficiency in this information may generate wrong field formation. It can cause aggressive control commands that may affect aircraft performances. Finally, there are other criticism of using a mathematical model to describe how the vehicle dynamics is affected by changes; for example, the method collapses all forces into one singular resulting force. It causes the loss of information about the obstacles location and consequently, even if it would be physically possible to traverse some difficult passages, it couldn't be possible in the real space.

To address some of these criticalities, other types of methods have been studied and developed as the vectorial field histogram.

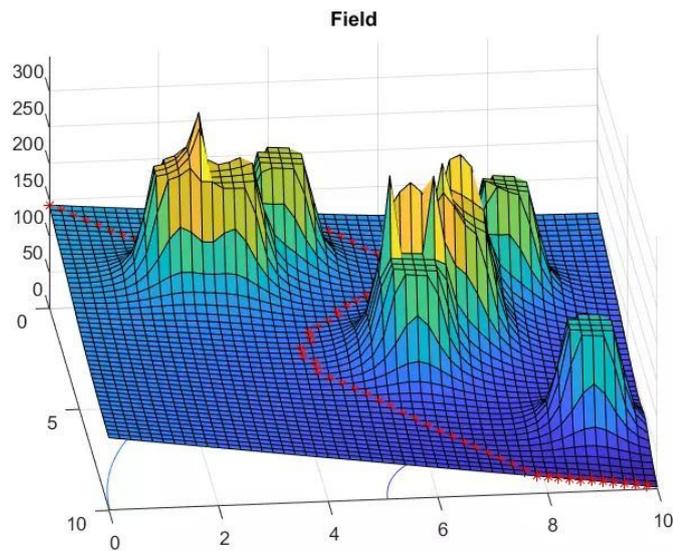


Figure 2.9: Visualization of an application of Artificial Potential Field as path planner

2.3.2 VFH - Vector Field Histogram

The Vector Field Histogram similarly to APF is a reactive real-time obstacle avoidance method. It was first proposed by Johann Borenstein and Yoram Koren in 1991 using a mobile robot. The VFH makes use of a 2-dimensional cartesian histogram grid considering it as the world model. The model is frequently updated by the data received by the sensors thanks to a two-stage reduction process. This process manages to compute the desired direction of motion first by reducing a constant size subset of the 2D histogram grid around the position of the robot in a one-dimensional polar histogram and then selecting the most suitable sector of the polar histogram. The process can be subdivided in three main steps:

- Creation of a bi-dimensional cartesian histogram grid representing the world around the vehicle with obstacles;
- Selection of an active window around the robot position of the 2D histogram grid and turn into a 1D polar histogram;
- Calculation of the steering angle and velocity from the 1D polar histogram resulting from an optimisation process.

The creation of the 2D Cartesian histogram is made by all the data incoming from the distance sensors on the aircraft and putting them into a grid map. This process is independent from the type of sensor used for the obstacle detection (laser, ultrasound, radar). For each range reading, the cell lying on the central axis and corresponding on a fixed distance d is incremented. The certainty value of the cells continuously updates during the vehicle motion.

The second phase transposes the bi-dimensional grid map in a one-dimensional structure. To better threat information about an obstacle and avoid high computational load, rather than process the whole grid map the active window concept is introduced. To restrict the 2D grid map, it is considered a constant dimensions area centered on the vehicle position; consequently, it moves with the vehicle and it represents a local area around it. The new grid is mapped in a one-dimensional structure called polar histogram.

Finally, during the last phase, the required direction to avoid any collision for the vehicle is evaluated. It is calculated by a given sector of the one-dimensional histogram in which the vehicle velocity is adapted according to the obstacle polar density. To choose the best sector of the active window to pass through it is necessary to analyse the polar histogram. To evaluate the best path to pass around the obstacle, is necessary to consider a various amount of factors, such as if the valley is large enough to permit the motion of the vehicle. If consecutive sectors are all defined as candidate valleys and considering other factors, like the alignment of the vehicle to the target, the difference between the current and the desired direction

and the difference between the previously selected direction and the new one, the new path is established.

The VFH overcomes some limitations of the potential field method. The influence of a bad sensor information is minimized, the absence of attractive and repulsive forces eliminates the problem of local minima of settle point. VFH can't be used as a global planner but only as a reactive local planner, it can't generate waypoints in its standard implementation but only provide the best direction for the drone to follow in order to avoid an incoming obstacle.

For the type of sensor deployed, a depth camera providing only 3D information of the world, the main drawback is the computing power required to generate and traslate individual points from a 3D image to a 2D histogram.

The 3DVFH+ is an extension of the basic vectorial histogram but, in this case, not only the dimension of the vehicle are considered, it will make a 2D polar histogram from an Octomap of the environment in this way merging both 3D and 2D information for the navigation of the robot.

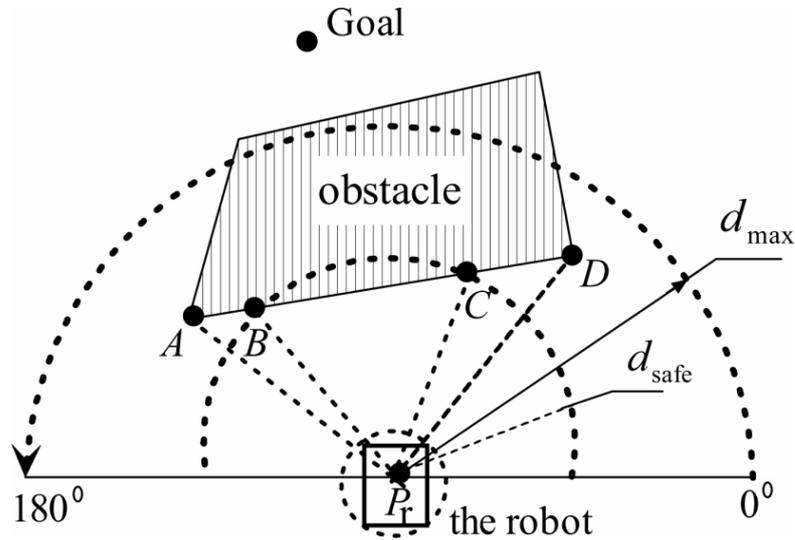


Figure 2.10: Vector Field Histogram

2.3.3 Machine Learning Approaches

Machine learning based approaches are gaining more popularity in recent years this is due to the advancement regarding the computing power of companion computers. In contrast to planners, these systems rely primarily on image analysis from sensors such as stereo cameras or mono cameras. Their purpose is not to generate a global or local trajectory, that because these systems are not using or constructing maps or graphs. Instead, after the training phase the algorithm learns a control policy that selects a steering direction as a function of the vision system's output.

In order to train the algorithm real camera images or consisting set of synthetic graphic images can be used. Labeling with ground-truth distances to the closest obstacle enables the recognition of it and to direct the drone in the direction with fewer obstacles or where the risk of a crash is lower.

The advantages of Reinforcement learnings and of policy search are then applied within a simulator that renders synthetic scenes. This learns a control policy that selects a steering direction as a function of the vision system's output. The great advantages are represented by the fact that the drone can be trained over virtual simulations and synthetic scenes in order to train the algorithm to recognize patterns of obstacles or to generate trajectories that can then be used in real-life applications.

Chapter 3

Hardware Implementation

The choice of the hardware parts represents a crucial step in the subsequent implementation of the collision avoidance system and in the drone's localization in space. Obtaining a stable and absolute position of it represents a challenging task and requires the use of different sensors. Without a system to obtain the location of the drone in a reliable way, any algorithm for the generation of a trajectory could not be implemented; an error in the position would lead the aircraft not to follow the points sent in the path and therefore in hazardous situations for workers and industrial machinery.

In this chapter, therefore, the emphasis is not only on sensors for localization purposes but also on the selection of the most suitable sensing elements for the chosen collision avoidance system.

3.1 Sensors for Collision Avoidance

As mentioned earlier, the choice of a sensor should be weighted by considering a set of arbitrary parameters chosen upstream. These criteria are mainly dictated by the tasks performed by the drone, maintenance, and the physical limitations of the vehicle. Not only that, but also the range of action of the device and its applicability in a purely industrial context limits the choice of usable sensors. In the subsequent list, these constraints are analyzed in more detail:

- The drone will be deployed in an indoor environment, mainly for maintenance purposes. The light conditions are considered optimal in most cases due to the indoor illumination. The sensor must, in any case, have a certain ability to perceive the environment, even in the absence of lighting.
- The range of action of the sensor has to be wide enough in order to ensure the effectiveness of the collision avoidance. The more the sensor is able to

perceive the surroundings of the drone, the larger will be the map generated, simplifying in this way the planning function.

- The weight of the sensor must not be too high to avoid overloading the aircraft, running the risk overheating the rotors of the drone or increasing the total energy consumption.
- The power consumption of the integrated sensor must be considered due to the fact that it could reduce the UAV's total time of flight and critically influence the missions.

Sensing technology can be subdivided into two major types based on their basic operation principles: active and passive sensors.

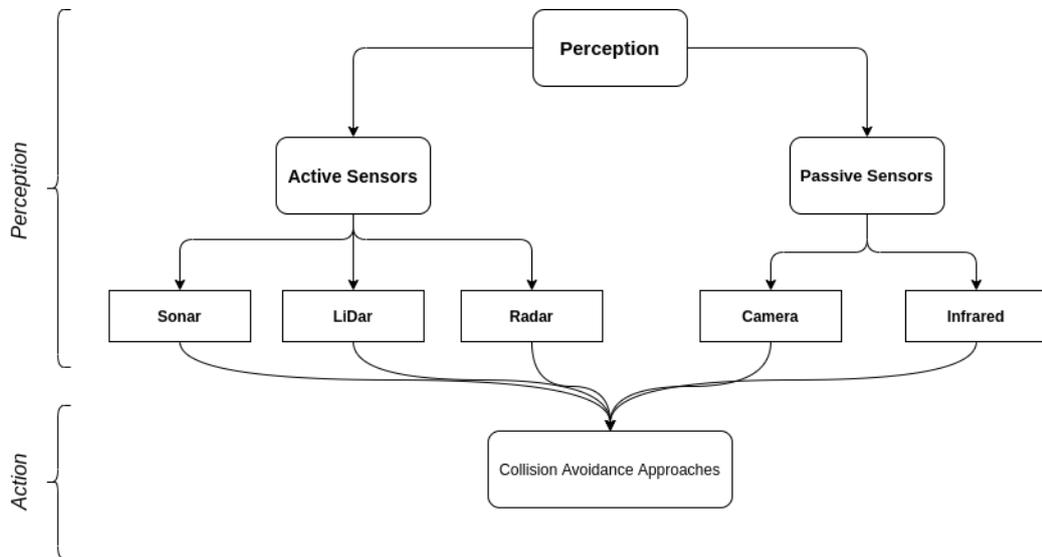


Figure 3.1: Different types of sensors and approaches to the Collision Avoidance problem

Active sensors all work using the same basic principle. An active sensor is composed by its own transmitter (source) and receiver (detector). A transmitter emits a signal (light wave, acoustic signal, electrical signal) which bounces on an object, the receiver then reads this reflected signal. Knowing the initial speed of the signal and the time it took to return to the receiver, it is possible to obtain information about the distance travelled.

Passive sensors are mainly designed to detect the energy discharged by the objects or the scenery under observation. Most part of these sensors are represented by optical, depth cameras and Infrared detectors.

Cameras

There are different types of cameras that operate on different wavelengths. Optical or visual sensors are cameras such as monoculars or stereo cameras that work by detecting the visible light. Thermal or infrared cameras have a longer wavelength and work with the infrared light. While the traditional cameras have poor performance in low lighting and bad weather conditions, IR cameras excel in such conditions.

All camera sensors rely heavily on image-processing algorithms in order to extract information from the chunks of data provided by the sensor.

Special consideration should be given to stereo sensors, cameras that have two sensors, spaced a small distance apart. A stereo camera takes the two images from these two sensors and compares them. Particular cameras that attempt to emulate human vision by using two cameras that fix the same scene at a fixed distance. The images from these cameras are then used for extraction and matching procedures in order to obtain a disparity map between them. Once the disparity map is obtained, it is possible to extract important features like depth and Point clouds that can be used for the construction of a depth-map or cost-map.

Lidar

Lidar sensors use laser beams to calculate the distance of objects around them and generate a clear image of their nearby features. The use of laser beams gives an upper notch over all other sensors that use the radio or sound waves. Lidar sensors usually come with an in-built software which is able to process the images at a great speed. This type of sensor can also work perfectly at night, ensuring no limitations in poorly illuminated environments.

Lidar technology, due to its physical properties and extremely high range w.r.t. to the other sensors is usually deployed to map out environments and large areas.

The basic principle of operation is extremely similar to that of sonar or radar. A laser beam is emitted from the sensor transmitter while the receiver receives the reflected one. Knowing the speed of light as a fixed value, it is possible to calculate the delay between the two beams and estimate the distance to the object. The use of a Lidar usually generates 2D maps and is not suitable for the generation of a 3D environment with a fast response time, where the use of a camera could be advantageous.

Time-of-Flight ToF

Time of flight sensors (ToF) and Lidar are essentially the same thing. They both use infrared light to calculate the depth of the surroundings. However, ToF is a camera that sends out a single laser pulse to get a reading of the depth of an

environment, whereas a Lidar scanner usually sends out multiple pulses to get an accurate reading of all the surrounding.

Therefore, they are not sensors used for map generation but to obtain information about the distance quickly from a given direction.

Sonar/Radar

Sonar and Radar sensors behave similarly to the ToF ones, their operating principle is capturing the bouncing waves and estimate the position of the obstacle by comparing them with an already known fixed speed.

Ultrasonic sensors (Sonar) provide a cheap and reliable means for obstacles detection. These sensors are amazingly accurate, though they may be thrown off by a sound absorbing obstacle, like a sponge. Application for collision avoidance systems specifically designed for drone already commercially exists. Usually, are composed by a ring of these sensors which enables a 360° cover from approaching obstacles. Another common application is for estimating the altitude of the aircraft or defining potential zone in Force-field approaches.

Furthermore, due to their structure supersonic sensors are susceptible to electrical noise, air turbulence and frame vibrations.

Event-based Cameras

Event-based cameras apply pixels that independently respond to change in brightness, each pixel records only when a threshold value is exceeded. The output is a sequence, an asynchronous stream of events. The main advantages over standard cameras are the very high dynamic range, no motion blur and extremely low latency. Due to the fact that the output of these sensors is not a series of frames, traditional vision algorithms cannot be applied. This type of camera sensors represents the future of camera application for collision avoidance systems, due to the fact that these sensors can be implemented with machine learning approaches to obtain high efficiency and safe systems which can be trained in a wide range of situations. The availability of event-based sensors is still short in the market, and only experimental models can be found.

After this brief introduction on the types of sensors most commonly used in the field of UAVs, a careful analysis has been conducted on the market to see the main available products. In the *table 3.1*, the main characteristics of different camera and Lidar sensors are reported. By keeping in mind the limitations listed above, the possibility of implementation and the technical features provided by the manufacturers, the Intel Realsense D435i camera has been picked as main sensor for the CAS of the drone.

Name	Dimensions(mm)	Weight(g)	Range(m)	FoV (H-V-D)
Intel RealSense D455	124x26x29	82	0.52 to 6	57°×86°x95°(±3°)
Intel RealSense D435i	90x25x25	72	0.28 to 3	57°×86°x95°(±3°)
Intel RealSense D435	90x25x25	72	0.28 to 3	57°×86°x95°(±3°)
Intel RealSense D415	99x20x23	72	0.45 to 3	41°×64°x72°(±3°)
Intel LiDar L515	61x26	95	0.25 to 9	70°x43°(±3°)
Intel RealSense T265	108x25x13	60	/	163°(±5°)
Azure Kinect	126x39x103	440	0.25 to 5.5	120°x120°
Velodyne VLP-16 Lite	103x72	590	up to 100	30°(±2°)x360°(±0.1°)
Mynt Eye S1030	165x31x30	184	0.5 to 18	76°x122°x146°
Mynt Eye D1000-120	165x31x30	190	0.3 to 10	58°x105°x121°
SilkyEvCam	30x30x36	40	/	/

Table 3.1: Sensors selection for the CAS

3.2 Intel Realsense D435i

Standard digital camera outputs images composed by a 2D grid of pixels. Each pixel has a defined value associated with it, which define the combination of primary color (RGB) for that specific pixel in a given time. On the contrary, a depth camera has an output image defined as RGBD, where each pixel shows a combination of 4 values; in addition to the 3 standard ones, the fourth expresses the "depth" corresponding to each pixel. Since the distance between the sensors is known, these comparisons give depth information. Stereo cameras work in a similar way to how we use two eyes for depth perception.

The Intel Realsense Camera works exactly in this way, in addition a Infrared sensor is added in order to improve the performance in low light condition and maintain a high level of details improving the generated data.

Beneficial to this type of implementation is the fact that there are no physical limits to how many camera sensors implements in a defined space, these camera don't interfere with each other like a pair of Lidar or ToF sensors would do. At the end of the analysis of sensors suitable for obstacle detection, only one of them has been selected. The most acceptable sensor has been chosen following an iterative analysis

between the main features of the sensors and the search for collision avoidance algorithms that supported them.

The Realsense D435i camera represents the best compromise, in terms of overall dimensions, weight, operative range and cost. The field of view is sufficiently wide to ensure optimum performance both at close and long range. In addition, the Realsense allows to apply a specific resolution to the images, in this way it is possible to directly act on the minimum radius of vision as reported in the schematics of the sensor. In addition, automatic flight missions will be programmed

Resolution	Minimum-Z Depth(mm)
1280x720	280
848x480	195
640x480	175
640x360	150

Table 3.2: How the Resolution of the Camera modify the minimum depth

to take UAVs always with the front facing the direction of flight, therefore it is not necessary to have a 360-degree field of view coverage. Thus, the camera will be integrated into the front of the vehicle.

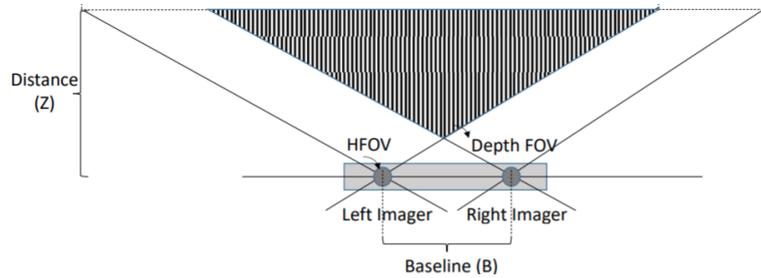


Figure 3.2: Graphical representation of the Cameras FOV



Figure 3.3: Intel RealSense D435i model

3.3 Companion Computer

The Companion Computer (CC) is nothing more than a single board computer (SBC) or a small computer, usually Linux-based, that can communicate directly with the flight controller. CCs applications can be very complex, before the advent of them, in order to integrate new feature on the aircraft it was necessary to modify directly the autopilot code, greatly increasing the difficulty of programming.

The development of suitable libraries allows an effective, fast and reliable communication via MAVLink protocol between the CC and the Flight controller. In this way, functionality like computer mediated flight paths, collision avoidance maneuvering or image processing algorithms are easily achievable; the only limit is represented by the computational power of the computer.

During the development of the drone model, two SBC were used, initially a Raspberry Pi model 4 but, due to the high workload of creating a three-dimensional map and streaming video, the software platform was moved to a Nvidia Jetson.

3.3.1 Raspberry Pi 4

The Raspberry Pi 4 is a single board computer commonly used in an extremely wide range of different applications due to the low price and high performance provided comparable to an entry level PC system. The Raspberry Pi 4 can be equipped with different sizes of RAM memory according to the required power, in this case 4GB.

The CPU is a high-performance 64-bit quad-core processor which enables hardware

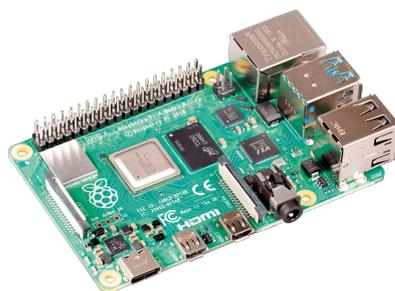


Figure 3.4: Single Board Computer Raspberry Pi 4 (4GB)

video decoding at up to 4K, the board has integrated dual-band 2.4/5.0 GHz wireless LAN, Bluetooth 5.0, Gigabit Ethernet and multiple USB 3.0 ports including I/O pin.

The extreme flexibility of this type of micro-computer was the reason of the first implementation of the hardware for the drone. The low power consumption and the great compatibility with all the other hardware parts like the Intel Realsense,

thanks to a large number of available open libraries, significantly speed up the first phases of testing.

Although the performance for a basic design were acceptable, the type of implementation that the UAV used, required a more powerful and computationally capable Companion computer. In fact, this card does not have enough power to allow extensive machine learning tasks and applications with a certain workload. In addition, due to the type of implementation used in this thesis, where a 3D occupancy map is deployed, to obtain reactivity and accuracy a more powerful companion computer should be used.

3.3.2 Nvidia Jetson Nano

NVIDIA Jetson Nano Developer Kit is a SBC designed with the purpose of being able to run multiple neural networks in parallel for various type of applications like image classification, object detection or segmentation.

Considering the computational power required by the drone's Collision Avoidance System, where it must be able to process the video stream, generate a 3D occupancy map, and resolve and generate the path for the aircraft, the choice of this board is more than necessary. Similar to the Raspberry Pi the power consumption is very small, the maximum power consumption is around 5W, all without compromising on the HW point of view being equipped with an ARM Cortex Quad-core processor in combination with 128 CUDA core Maxwell GPU and 4GB of RAM.

In addition, counting on a possible future development of the project that will



Figure 3.5: Single Board Computer Nvidia Jetson Nano(4GB)

include machine learning algorithms for the recognition of objects or people within the industrial environment, since the board is equipped with a discrete GPU, the operation of such technologies can only benefit.

The connection to the Jetson Nano is fully supported by the FCU, simply utilize

a Pixhawk's Telemetry port and directly attach it with to the Nvidia's I/O pins. The baudrate and the communication speed can be settled afterward thanks to the Ardupilot software.

3.4 Flight Controller - Pixhawk 2.4.8

The Flight Controller can be seen as the heart of the UAV and performs all the basic functions for the proper operation of the drone. While the computing power that these boards bring with them is not high as the one of the CC, they do enable the communication with a range of sensors that detect movement of the drone, as well as user commands. Using this data, then controls the speed of the motors to make the craft move as instructed.

In this case, the flight controller adopted was the Pixhawk version 2.4.8, an open-source autopilot developed by the collaboration of Holybro and PX4. This board can mount different versions of operating system, one dedicated entirely on the PX4 firmware and another one developed by the Ardupilot community. In addition, on the board multiple sensors are already present dedicated to provide all the basic data needed to localize and guide the aircraft : 3-axis gyroscope, 3-axis accelerometer in combination with a magnetometer and a barometer to gather data about the orientation, position and heading of the UAV.



Figure 3.6: Pixhawk 2.4.8 Flight Controller

3.5 Outdoor Flight Sensors

A functioning and safe CAS needs at its core an effective and reliable localization system, for this reason different sensors and solutions have been adopted also from a hardware point of view to locate the UAV based on the environments in which it can be used.

Focusing initially on drone flight in outdoor environments as a matter of greater simplicity of sensor implementation, several types of GPS technology were investigated. Since the Ardupilot firmware depends largely on the accuracy of GPS to determine the position of the drone in space and control its stability, with this notion in mind, sensor performance must be carefully analyzed to determine which to use within missions as the primary GPS sensor.

Not only the quality of the sensor, but also the weather conditions and the location chosen to accomplish the mission affect the quality of the signal. For these reasons in the outdoor tests we tried to maintain an outdoor location clear of large obstacles that could interfere with the reception of the signal and causing the multi-path effect and to test the flight of the UAV on sunny and cloudless days.

Poor information from the GPS greatly influences the quality of the flight and the precision of the drone to follow a described path, not only that, GPS glitches represents a severe problem for the EKF integrated in the Ardupilot software which can lead to crashes and loss of control of the vehicle.

The quality of the GPS signal is generally expressed through a set of parameters indicating the Dilution of Precision (DOP), respectively HDOP, VDOP, PDOP and TDOP (Horizontal, Vertical Position, Time), whose indicate the level of precision in each dimension of the receiver measurements. The lower these values, the greater the accuracy obtained by the GPS sensor in the real-time position estimate of the aircraft. For this reason, before taking any mission, it was mandatory to read these values and, if they were too high, change the starting point for more precise signal reception.

M8N GPS

The first type of GPS sensor used was the stock one given with the default configuration of the UAV. The NEO-M8 series of concurrent GNSS modules is built on the u-blox M8 GNSS engine and integrate a new digital compass model HMC5883L. The performance provided by this sensor were in line with the entry level GPS sensors on the market, the maximum number of satellites visible by it was around 10 giving a accuracy of the position between a five meters radius. Although the casing provided a good shielding from magnetic interference the ignition of the drone's engines could cause interference on the signal, especially for

the compass, during the most aggressive flight phases.

Beitian GPS

Another type of entry level GPS sensor deployed was the Beitian BN-880 which should provide better performances than the M8N GPS. It comes with the same compass sensor of the M8N GPS.

Although the ZED-F9P GPS was used as a main GPS sensor for the outdoor localization of the drone, the Beitian GPS was still being used as an external secondary compass for the drone. In fact, Ardupilot's platform allows you to use up to 3 compasses simultaneously, greatly improving the drone's attitude estimation. The communication is enabled thanks to an I2C splitter, where both the GPS and Sensors signal are injected to the Pixhawk platform. In order to obtain a working result, the baudrate of the communication was to be settled to the optimal value.

ZED-F9P RTK GPS

The ZED-F9P positioning module was the last GPS sensor implemented in the UAV design, providing the best results in term of localization and position hold. It features the new u-blox F9 receiver platform, a multi-band GNSS module with integrated u-blox multi-band RTK technology with the possibility to achieve accuracy in the range of centimeters. Despite that during our outdoor tests this feature was not used, still the flight performance and stability were guaranteed due to the fact that the number of satellites hooked up to 22.

The combination of this GPS sensor and the fusion between the compasses with the Beitian's one generates a stable attitude estimation and control of the yaw of the UAV.



Figure 3.7: Taoglass ZED-F9P GPS

3.6 Indoor Flight Sensors

Indoor flight adds complications regarding drone tracking, attitude management, and limitations on mission safety. For these reasons, different kinds of sensors are deployed in order to maintain stable performances close to the Outdoor scenario. An experimental implementation of Ultra-Wideband technology is approached and integrated in the flight stack in order to generate accurate coordinates to locate the drone in the Space even in location where the GPS signal is completely absent. The combination of this technology and a ToF sensor was used as substitute for flying the drone in indoor environment. Exploiting a *fake_gps* plugin integrated in the Mavros code, it was possible to force positioning information incoming from external sensors directly in to the Flight Controller.

3.6.1 DWM1001C - Ultra-Wideband

Ultra-wideband is a radio technology that is seeing increasing use in various sectors as target sensor data collection, precision locating and tracking applications due to its cost-effectiveness and accuracy. It has low energy consumption and suitable for short-range communication due to the high-bandwidth of the radio spectrum used (the frequency range classified as Ultra-wideband is between 3.1 and 10.6 GHz).

As its core, the functioning is similar to Bluetooth but with vastly improved



Figure 3.8: Decawave DWM1001 Development Board

precision (up to 5 centimeters) and effectiveness. UWB transmits data across short distances and precisely determines location by measuring how long it takes for a radio pulse to travel between devices.

In the thesis project the sensor used is a DWM1001C Qorvo directly attached to the center of the drone's frame (called Tag node) and connected to the Companion Computer exploiting a USB connection. The sensor is equipped with the DWM1001C UWB module directly soldered to the PCB which is capable of producing a position estimate by measuring the ranging from different Anchor nodes positioned in the room.

In order to achieve the most encouraging results and consistent, reliable tracking, it is crucial that the Anchor nodes are placed in precise geometric shapes and

their relative location has to be known with low uncertainties. For simplicity of application, usually a square shape is adopted, in our case scenario though a rectangular cell was designed by combining two square ones. In this way the environment can be subdivided in two separated cells improving the accuracy.

3.6.2 VL53L1X ToF sensor

The implementation of a Sonar, IR or ToF sensor was mandatory in order to generate stable altitude coordinates for indoor navigation. Luckily, the Pixhawk board comes with a wide predisposition for the deployment of many sensors of this type, both from a hardware and software perspective.

For this reason, the choice was mainly made based on the maximum range provided and the accuracy of the device. On the Frame of the UAV two VL53L1X ToF sensor are applied, one facing upward and one downward, in this way the height can be calculated both from the ceiling and from the floor according to different conditions. If the drone is found to be flying close to the ceiling where most of the lighting is present, the performance of the ToF would be greatly affected, in these cases the height is only provided by the downward facing sensor. On the contrary, in a standard flight situation both provide data for the height calculation to the EKF present in the Autopilot Firmware, guaranteeing a very high accuracy. The chosen model guarantees a maximum range up to 4 meters with a frequency of sampling of 50Hz.

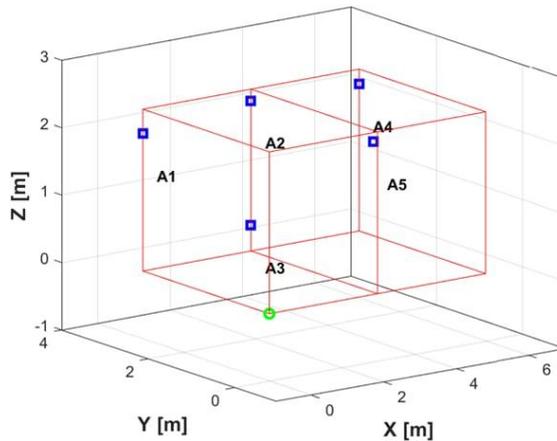


Figure 3.9: Indoor Cage anchors configuration were the drone was tested

Chapter 4

Software Implementation

On the software side, the requirement was to have a flexible and easily programmable system based on an open-source platform. For this reason, the operating system used by the companion computer is Linux Ubuntu version 18.04 in which ROS Melodic (Robotic Operating System) has been installed. In this chapter, an exhaustive analysis of all libraries and packages used in the writing and implementation of the CAS algorithm is conducted. In particular, in the last part, an in-depth description of the implemented code and its basic principles of functioning is given.

4.1 Robotic Operating System - ROS

Robot Operating System (ROS) is an open-source robotics meta-operating system. It provides all the services expected by an operating system as hardware abstraction, low-level control, implementation of commonly-used functionality, message-passing between processes and packages management. All the ROS processes can be represented as nodes in a graph structure, connected by edges called topics. These nodes can pass messages to one another communicating data thanks to these topics, call new nodes, provide specific services or retrieve information from a communal database called the parameter server.

ROS represents the heart of all the development and the implementation of the collision avoidance, providing a platform where all the incoming data from the different sensors, the flight controller and the camera are collected, shared and elaborated in order to run efficiently the algorithm.

4.1.1 Nodes

In ROS a node is a defined process that is able to perform computation. Nodes communicate with each other using messages passing via streaming topics, services

or Parameter server. Each node can send or get data from the another one using the publish/subscribe model. To manage this loosely-coupled environment, there is a Master-Node in ROS which is responsible for name registration and lookup for the rest of the system. Without the Master, nodes would not be able to find each other or exchange messages. Usually, in a robotic application multiple nodes are launched simultaneously, in this way each one of them can take care and control one specific task. This kind of system is represented by the Node-Graph, a tree-like structure containing all the active nodes and their links.

4.1.2 Topics

Topics are a core element of the ROS graph that act as a bus for nodes to exchange messages and are one of the main ways in which data are moved between nodes and therefore between different parts of the system. Topics have anonymous publish/subscribe semantics, which decouples the production of information from its consumption. In general, nodes are not aware of who they are communicating with. Instead, nodes that are interested in data subscribe to the relevant topic; nodes that generate data publish to the relevant topic. There can be multiple publishers and subscribers to a topic. Topics are intended for unidirectional, streaming communication. Each topic is strongly typed by the ROS message type used to publish to it and nodes can only receive messages with a matching type.

4.1.3 Services

Not all the communication in ROS follows the subscribe and publish paradigm. For all request and reply type of interactions, which are often required in a distributed system, the Service communication is used instead. This is performed by a pair of messages: one for the request and one for the reply. A providing ROS node offers a service under a string name, and a client calls the service by sending the request message and awaiting the reply. Client libraries usually present this interaction to the programmer as if it is a remote procedure call.

4.1.4 Messages

Nodes communicate with each other by publishing messages to topics. A message is a simple data structure, comprising typed fields. Standard primitive types (integer, floating point, Boolean, etc.) are supported, as are arrays of primitive types. Messages can include arbitrarily nested structures and arrays (much like C structures). Having matching type of messages and topics is a must in order to establish a communication between the ROS nodes.

4.2 Gazebo

Gazebo is an Open source 3D dynamic simulator, that has the ability to accurately and efficiently simulate robots in user-created virtual environments. Similar to a game engine, it offers an accurate physic simulation capacity with an high level of fidelity, a suite of already integrated sensors and different interfaces to analyze the simulated scenario.

With the possibility to simulate the robot systems, sensors and test a given algorithm in real-time it is an extremely useful tool to develop without risks or damaging the hardware of the machine. In addition, Gazebo is integrated with ROS, where it is possible to generate launch files able to pass directly relevant parameters and speed up the development.

For the scope of the thesis, the generation of different maps in the Gazebo environment enables the testing of the collision avoidance algorithm in every type of situation and obstacle behaviors. In this way, the tuning of the parameters for obtaining a working code becomes much easier.

To add a greater degree of realism, in all simulations a model of the actual UAV used was imported directly into the simulation via the CAD file, as shown in *Figure 4.1*

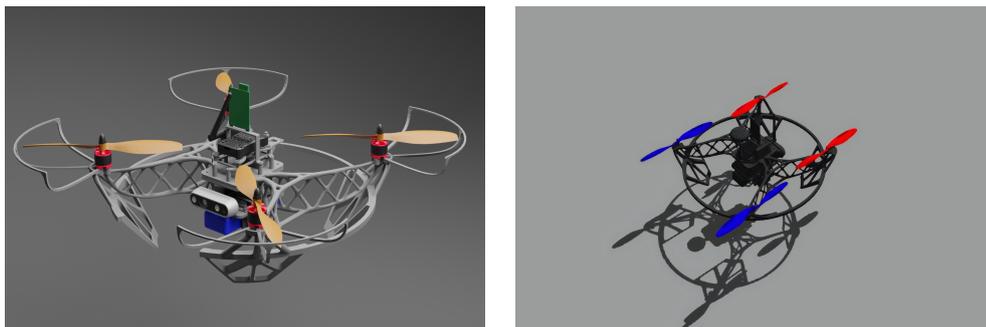


Figure 4.1: Visualization of the UAV in Gazebo from the CAD render

4.3 Mavros

Among the numerous packages deployed in the thesis development, Mavros node constitutes a key piece. The Mavros package enables MAVLink communication between the Companion Computer, which is running ROS, the MAVLink enabled Autopilot and the MAVLink enabled ground control station (GCS), in this case the PC connected to the network. This package provides communication driver for various autopilots with MAVLink communication protocol, directly translating the command sent using ROS in MAVLink in order to maneuver the UAV. Not only

that, the numerous plugins contained in the *Mavros_extra* package allow a huge versatility in the implementation of additional external peripherals to the classic Ardupilot firmware. This is the case for the *Fake_GPS* functionality which is used for the indoor navigation of the UAV.

Moreover, the node considerably simplify the creation of Reference Frames for estimating the position and the control of the aircraft, due to the fact that automatically translates Aerospace NED coordinates used by the FCU in ENU frames, the standard for ROS navigation.

4.3.1 PointCloud Library - PCL

The Intel RealSense camera is able to generate point cloud and depth images. To use this kind of data a specific library needs to be implemented. The Point Cloud Library (PCL) is a standalone, open source package for 2D/3D image and point cloud processing.

Although the camera sensor is reliable, it can still pick up interference or errors. These mistakes can accumulate and, once the mapping procedure is started, generate false information of the environment like non-existing obstacles or limit the perception of small openings. To obtain cleaner data, it is mandatory to filter out noises and perform images processing, for this tasks the PCL library is suited with different type of filters:

- Filtering a PointCloud using a PassThrough filter
- Downsampling a PointCloud using a VoxelGrid filter
- Removing sparse outliers using StatisticalOutlierRemoval
- Removing outliers using a Conditional or RadiusOutlier removal

The filtering procedure not only reduces the possibility of retaining false information, but increases the speed of the generation of the 3D occupancy map.

Before generating the Octomap, the incoming Pointcloud passes through a Statistical outliers removal, which removes all the sparse points that are spread over a fixed threshold, a PassThrough filter, that remove all the points which are out of a defined bounding box, and subsequently a Voxel Grid which merges multiple points in a single entity. After all of these operations the total number of points registered is reduced from the range of millions to thousands.

The result of all these passages can be seen in the *Figure 4.2* The PCL was fundamental in the image processing part required in order to obtain cleaner Occupancy map. In order to filter out the noise incoming from the camera and reduce the number of total point generated from the depth frame. PCL includes a set of different filters suited for the most common used: In this thesis application a

combination of outliers filter in order to remove sparse points from the PointCloud and a VoxelGrid application to greatly reduce and generate less point were used. In the subsequent images is possible to understand the effect of this application:



Figure 4.2: Filtering action on the Poincloud

4.3.2 OMPL - Open Motion Planning Library

The planning part of the algorithm is carried out through the OMPL library, which consists of many state-of-the-art sampling-based algorithms for planning. Unlike other libraries, OMPL does not contain any explicit code related to collision checks or visualization, thus all the planners are not tied to a particular collision checker or visualization front end. In this way, any algorithm can be easily integrated into systems that provide the additional needed components.

A sampling-based motion planning process computes a set of random robot configurations that are bound to a certain motion constraints, derived by the aerodynamics of the robot itself. Collision free paths are then computed connecting these samples. Sampling-based planning is a very powerful tool for planning in high-dimensional spaces or for system with complex dynamics.

It is useful to define key terms that allow to categorize standard objects used in writing the algorithm.

- *Work-space*: The physical space that the robot operates in. It is assumed that the boundary of the work-space represents an obstacle for the robot.
- *State space*: The parameter space for the robot. This space represents all possible configurations of the robot in the work-space. A single point is called a State.
- *Free state space*: A subset of the State Space in which each state corresponds to an obstacle free configuration of the robot embedded in the work-space.

- *Path*: A continuous mapping of states in the state space. A path is collision free if each element of the path is an element of the free state space.

Thus, to correctly implement the OMPL for geometric motion planning, it is essential to define and instantiate the State Space object for the robot, and provide the Start and Goal configuration in that space to define the motion planning query. The planner has to obtain the subset of the Free State Space to define all the possible configuration without any collision, in order to do so the additional library needs to be used. Any planner defined in the geometric name-space can then be employed to solve the motion planning query. OMPL provides implementations for several common states spaces, including R^n for Euclidean spaces and $SO(2)$ and $SO(3)$ for the space of rotations in 2D and 3D respectively.

In the motion planning literature, many variants of the classical planning algorithms are present, that change only one component of a planning algorithm in order to achieve benefits in certain instances, some of these also follow optimization objectives. Due to the open source nature of the library is easy to modify and highly customize the already present planners to derive new ones from the existing base code.

4.3.3 FCL - Flexible Collision Library

As stated before, the OMPL library does not provide any code for collision checking between the Robot and the State Space. For this reason the implementation of a Collision checking library was mandatory for the algorithm to work as intended. FCL is an open-source library able to perform different tasks, and achieve three types of proximity queries on a pair of geometric models composed of triangles.

- Collision detection: detecting whether the two models overlap, and optionally, all of the triangles that overlap.
- Distance computation: computing the minimum distance between a pair of models, i.e., the distance between the closest pair of points.
- Tolerance verification: determining whether two models are closer or farther than a tolerance distance.
- Contact information: for collision detection and continuous collision detection, the contact information (including contact Normals and contact points) can be returned optionally.
- Continuous collision detection: detecting whether the two moving models overlap during the movement, and optionally, the time of contact.

The main use of the FCL was to generate two separate objects: a Drone one, representing the dimension of the UAV, and a Map object, directly taking the information incoming from the Octomap node. Following this procedure it was possible to perform collision checking between them and obtain as an output if one was occurring in a define state space or not. The result of this operation is directly sent to the Planner to generate a collision-free path.

In particular, the great flexibility of this library enables to choose the shape and the dimensions of a defined object, e.g. the UAV, by doing so it is easy to transpose the algorithm to work with different drone models. Additionally, choosing the width and the length parameters accordingly the aircraft can be moved even in tight corridor and passages.

4.4 Robot Localization in Indoor Environment

Flying the UAV in an Indoor configuration results problematic for a series of reasons, the tighter tolerances for movements, GPS-less flight and more importantly possible electromagnetic interferences that greatly reduce the accuracy on the yaw-estimation and attitude estimate of the aircraft.

In order to solve these problems a different approach w.r.t the outdoor flight needed to be implemented. While, the position estimate could easily be solved by the UWB sensor implemented, to solve yaw estimation and attitude an additional EKF was added in order to fuse the data generated from the IMU sensors on the Pixhawk, the Visual Odometry provided by the Camera and the localization from the UWB.

In spite of the sensor fusion, in order to obtain a working system it is necessary to use some modifications on the covariance matrix, changing the weights based on the reliability and accuracy of the various sensors fused.

Another issue that arises from indoor use is due to the Autopilot itself and the functionality of the flight stack. Missing a GPS lock on the UAV disable the creation of local and global reference and the Mavros node is unable to send coordinates to a specific frame to navigate the drone. To solve this problematic it was mandatory to create a fictional reference frame with a defined odometry generated by the EKF implemented adopting the sensors fusion.

4.4.1 Visual Odometry - VO

Visual odometry refers to the process of extracting Odometry information, in particular the position and orientation of a robot, by analyzing the sequential camera images estimating the distance travelled. In this way, it is possible to define the movements made and localize the system without using a GPS signal.

This technique is broadly used in standard wheeled robots, but for mobile ones

with non-standard locomotion methods, such as aircraft, the accuracy obtained is extremely compromised. In addition, odometry used for navigation suffers from precision, since slip and slide or simply the vibration of the system may cause errors in the reading, thus generating a wrong position of the robot. Odometry readings become increasingly unreliable as these errors accumulate and compound over time.

For a Quadcopter the vibrations of the frame generate influence the image readings, not only that, the aircraft is not stationary in a fixed position in the space but keeps moving in the space with small movements. It is easy to understand why, this method can't be deployed alone but in conjunction with other localization systems the accuracy can be increased significantly.

4.4.2 IMU

Both the Intel RealSense camera and the Pixhawk board are provided with an IMU. While, the first one is not very precise, the latter can be used in the sensor fusion in conjunction to the VO to generate better odometry messages. In reality an arbitrary number of IMU sensors can be fused together to obtain better accuracy on the measurements. The data coming from the Pixhawk, in fact, are generated by the fusion of the two IMU soldered on the flight controller and very precise.

4.4.3 UWB and VL53L1X Tof

Thanks to the *fake_GPS* plugin provided by the Mavros it was possible to send the local position of the drone as a GPS message directly to the FCU. In the generated message, the X,Y coordinates sent were extremely precise thanks to the Kalman Filter implemented as a separate process on the system while, the Z one, was obtained directly from the ToF sensors. In this way, the message generated contained the best possible array of positions to achieve an accuracy in the scale of centimeters.

4.4.4 Robot Localization Package

After adding all the previous settings, using the Robot Localization package it was possible to deploy an EKF and proceed with the sensor fusion. The extended Kalman filter (EKF) is the nonlinear version of the Kalman filter which linearizes about an estimate of the current mean and covariance.

The first step is to provide the topics to be used as input for sensor fusion like the VO package, the UWB and Tof messages and the Mavros IMU messages. Subsequently, it is possible to proceed by directly modifying the covariance matrix of the filter, weighing which topics provide the most reliable measurements and

which should have a lower weight.

The output of the Robot Localization can be used directly to create a fictional odometry frame, stabilizing in this way the position of the UAV without requiring any GPS signal.

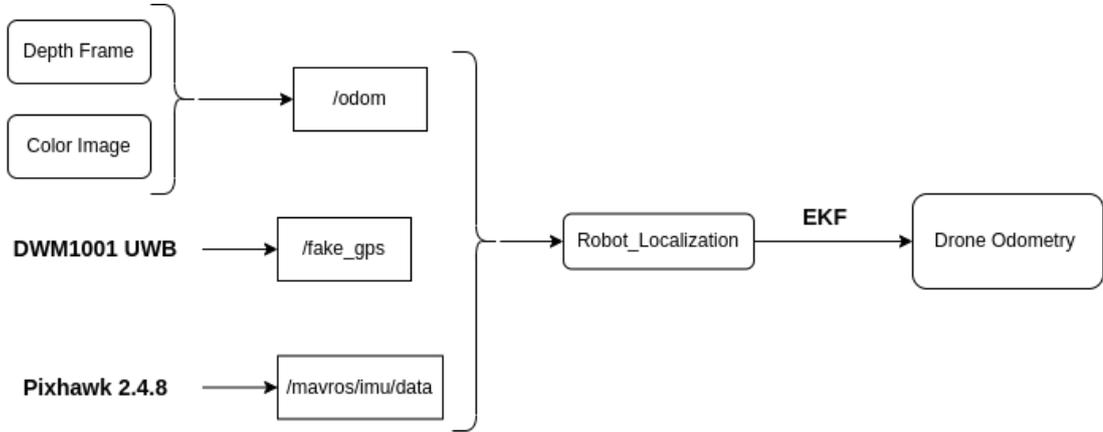


Figure 4.3: Robot Localization scheme

4.5 Octomap 3D occupancy Map

Once the point cloud has been filtered and the total number of points reduced, the generation of the map representing the drone’s surroundings can be started.

In order to perform the reconstruction, another open-source packet has been used. An Octomap representation is a three-dimensional model of the environment, providing a volumetric map of the space which can be used for a broad number of applications.

This kind of mapping approach is based on octrees, *figure 4.4*, and exploits a probabilistic occupancy estimation which can be set as needed, and it can represent both occupied and unoccupied spaces. Due to the use of Octrees this method implicitly provides a map compression which keeps the 3D models generated compact. An octree is a tree data structure in which each internal node has exactly eight children. Octrees are most often used to partition a three-dimensional space by recursively subdividing it into eight octants. This kind of approach is able to generate and update efficiently the map keeping memory requirement at a minimum. To further speed up the rendering of the map, the resolution of the Octomap can be chosen via a simple parameter, thus reducing the latency between the incoming data and the obstacle revelation. Furthermore, this framework is completely ROS integrated giving it a strong flexibility and interaction with the collision avoidance algorithm. To decrease the memory consumption a bounding box can be created

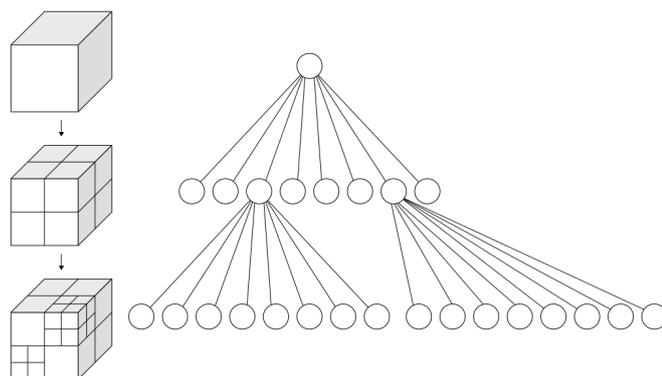


Figure 4.4: Octrees used in order to build the 3D Octomap

around the UAV's frame to clear at each cycle the 3D map. By doing so, the problem of dynamic obstacles generating a blurred trail during the mapping is completely eliminated and the map is always updated to the most recent time frame.

Due to the probabilistic nature, when generating a 3D map the measurements are affected by uncertainties typically produced by the range of the sensor, the reflection, dynamic obstacles or simply due to the noise. All these criticalities can be taken into account probabilistically, multiple uncertain measurements are fused together to form a robust estimation of the true state of the UAV's surroundings. In this way, even if multiple sensors are added in future the same package can be deployed just by doing a sensor fusion of all the incoming data.

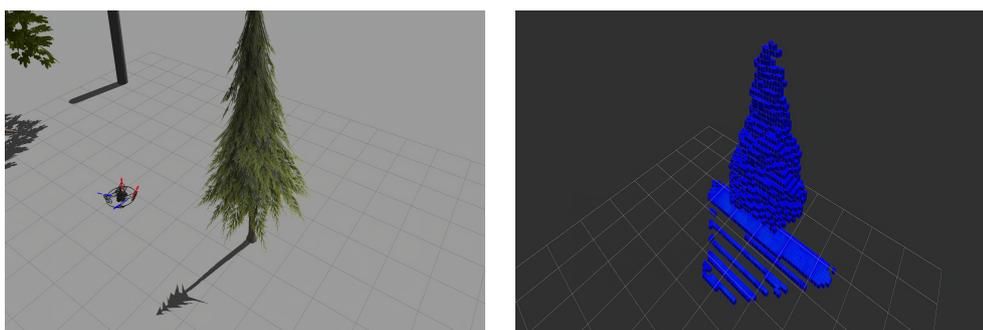


Figure 4.5: 3D Octomap representation of a Tree in the Gazebo virtual environment

4.6 EKF - Ardupilot's Parameters tuning

The latest changes are at the level of the Autopilot firmware parameters themselves. Both the Ardupilot and the PX4 firmware provide an Extended Kalman Filter (EKF) for Copter and Plane aircraft, which estimates the vehicle position, velocity and attitude based on the fusion of all the available sensors. Merging all the received measurements make possible to reject all that measures with significant errors. In this way, the system becomes less susceptible to faults that can affect one single sensor.

The filter in the Autopilot is deployed as multiple instances called *lanes*. Each one of them is able to provide an estimate, the more accurate of them becomes the primary lane and it is the one providing the drone's state estimate. The Ardupilot software enables a lane for each IMU sensor present. Additionally, each lane uses the primary instance of the Airspeed, Barometer, GPS and Magnetometer sensors. The primary sensor can be set as a parameter, but the FCU can later change it autonomously, even in-flight, in case of a driver-level fault. Affinity is a way for the EKF lanes to use non-primary sensors within any running lane. Statistically it can be shown that this provides a consistent way to use multiple high quality sensors and use lane-switching to select the lane which has best performing combination of sensors.

The Ardupilot's firmware provides multiple versions of EKF, simply called EKF2 and EKF3 which can be activated by modifying a single parameter. The different filters are due to the version of the firmware installed on the FCU, in this case using the Ardupilot 4.0.4 the UAV is limited to the EKF2, which is a 24-states Kalman Filter especially indicated for the flight of a Copter.

The advantages of this kind of filter are:

- It can run a separate EKF2 for each IMU making recovery from an IMU fault much more likely and is able to recover faster from bad sensor data.
- It provides slightly smoother output, in this way UAV's behavior is less sudden and abrupt in the maneuvers.
- It is an older implementation, more reliable with no bugs and with slightly less computing power required.

In order to stabilize the outdoor and indoor flight of the UAV some changes on the Ardupilot's parameters needed to be made. Using the APM-Planner Software the use of the Extended Kalman Filter type 2 was activated on the FCU, subsequently the EKF2 parameters have been tuned to provide a compromise between accuracy and robustness to sensor errors.

- `EK2_MAG_CAL`: This parameter is responsible for determine when the filter will use the 3-axis magnetometer fusion model to estimate both the earth

and the body frame fixed magnetic field states. This model can only be used when the external magnetic field environment is stable and without any major disturbances. In this case scenario, the activation was set after the Yaw reset and In-air. In this way the electromagnetic disturbances generated by the motors and the ESC controller was taken in account.

- `EK2_ALT_SOURCE`: This parameter impose which sensor use as a main source for the Altitude of the Aircraft. For both Indoor and Outdoor flight, due to the fact that the UAV was flew with low altitude to avoid any safety hazards, the main altitude source was set to the Range Finder (Tof) sensor.
- `EK2_POS_GATE`: This parameter controls the number of standard deviations applied to the GPS position measurement innovation consistency check. The implementation of the ZED-F9P GPS, with its extremely high accuracy, allows to set number of Standard Deviations very low to w.r.t the standard.
- `EK2_POSNE_NOISE`: This parameter sets the GPS horizontal position observation noise. Following the same reasoning as before, the implementation of a good GPS unit allows to lower this value to the minimum possible. Even in Indoor flight, due to the fact that the UWB is implemented instead of the GPS, this parameter can remain low.

To modify these parameters a trail-by-error procedure was adopted, although the precision of the sensors was known from the data-sheets in a real-life scenario the accuracy can be significantly different due to the various environmental conditions. Thus, after each change in the parameterization the impact was tested directly on the UAV to verify the effectiveness of it.

Using the implemented function of the Autopilot, which saves all the flight log file, it is possible to obtain a representation with numerical data directly coming from the FCU as output of the EKF. In particular, the focus in the localization to provide a better stability arises from this data set as shown in the *figure 4.7*, where the relative position extrapolated from the GPS localization (ENU coordinates) of the UAV resides inside a circle of radius 10 centimeters.

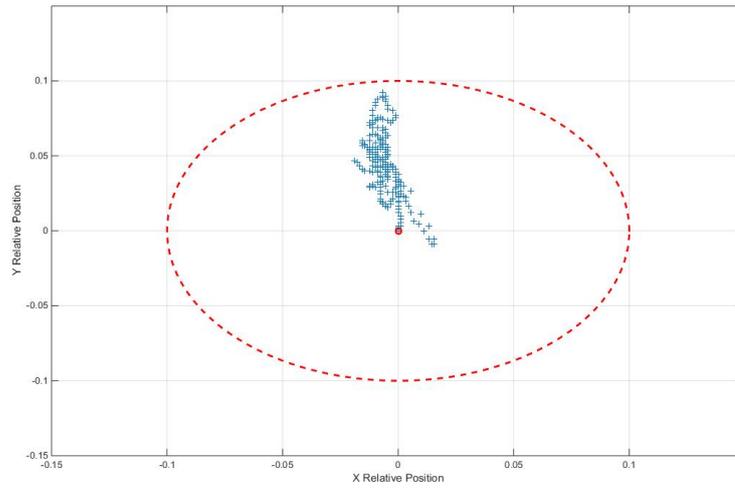


Figure 4.6: Relative Position of the UAV in Outdoor condition, the Red Circle is a Radius of 10 centimeters.

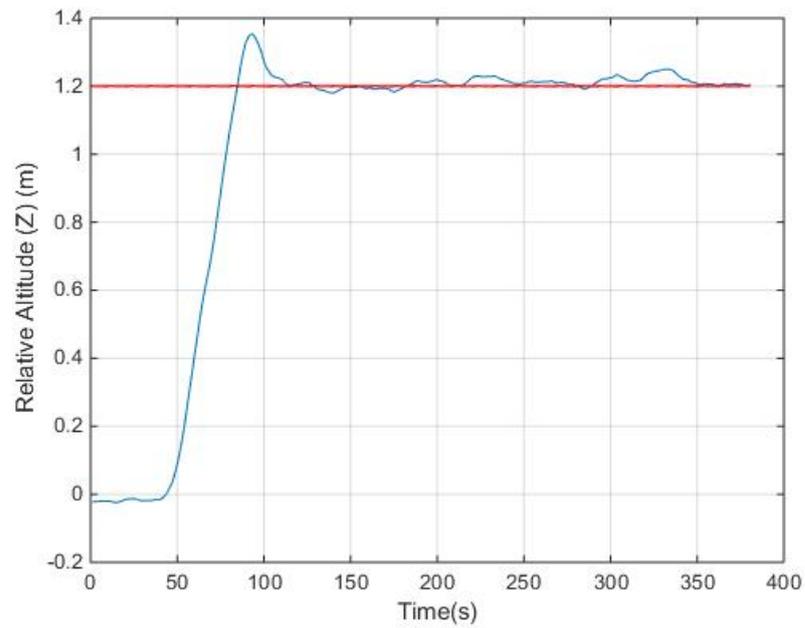


Figure 4.7: Relative Position of the UAV in Outdoor condition

4.7 Algorithm implementation

After all previous packages were installed within the ROS catkin workspace correctly, the implementation of the collision avoidance algorithm could begin. The algorithm can be subdivided into two parts, the first one composed of callback functions, in which the ROS topics are read and translated into useful variables for the code each time a new message get published, and the second one, where the resolution of the problem of path planning takes place and all the operations to ensure the validity of the route found and any eventual replanning.

In the following list the various callback functions are analyzed accordingly to their purpose:

- **Odometry Callback function:** this function reads the *nav_msgs::Odometry* directly generated by the Mavros topic */mavros/local_position/odom*. This kind of message contains both information about the odometry of the UAV and its relative position in the space. The information regarding the localization is saved and used to set the Starting point in the Planner function and to check if the actual location of the Drone corresponds to the Waypoint sent.
- **Goal Callback function:** the name of the function is self-explanatory, it reads the topic */move_base_simple/goal* which send both the coordinates and the desired attitude of a Point in the space. It is possible to publish in this topic easily through the GUI provided by the Rviz software. All the information are then saved and used to set the goal point in the planner function.
- **State Callback:** the State of the UAV corresponds on the modality in which the FCU is settled. Different states are available in the Autopilot firmware but, in order to remotely control the Drone and sending customize position it is mandatory to set the *GUIDED* mode. The role of this callback is sorely to read the information about the mode of the FCU and, if necessary, switch it to the right one.
- **Position Callback function:** the behavior of it similar to the Odometry one, it reads the actual position of the UAV in the Space without any information about the Odometry. This function is useful when, in indoor navigation condition, the Autopilot does not provide any information about the local position due to the lack of GPS signal. To read the coordinates is necessary to obtain the information directly from the *fake_GPS* plugin by reading the */mavros/fake_gps/mocap/tf* topic.
- **Octomap Callback function:** it is responsible to read the */octomap_binary* topic and extrapolate the informations necessary to construct and update constantly the 3D occupancy map. Once the datum containing the occupied

cells is saved it is stored inside the Map object used for the collision detection through the FCL library. The update rate has to be chosen carefully, due to the fact that these kind of messages represent a heavy computational load. A trade-off between constantly updating the map and the responsiveness has to be made.

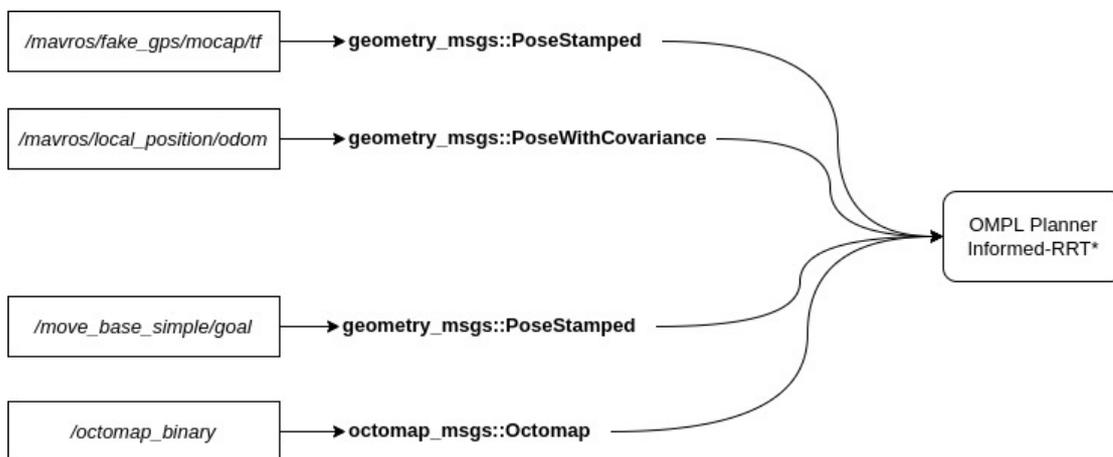


Figure 4.8: Topic and Messages published in the Planner part of the Algorithm

Once all the informations required to start the planning procedure are collected, the OMPL library is used to define the State Space in which the planner will have to find a path. The state space dimensions can be defined in different ways, it can follow the ones provided by the Octomap, by doing so the more the space is explored and mapped the larger will be the Space available for planning, or a more general approach in which the dimensions are fixed as a given parameter. In this application, the second approach was used to simplify and speed up the calculations. The 3D State space in which the UAV can move is defined as a Box with dimension 20x20x4 meters.

Subsequently, the Drone object is created in order to perform the validity check for each waypoints from the path. The easiest solution to define it without giving up to the kinematic constraints thus incurring to an over simplification, was to deploy the FCL library to generate a Box with precise length, width and Height close to the real dimension of the drone. To get a margin of error and compensate for a bad localization or vibration of the aircraft a safe margin has been established by increasing the real ones of some centimeters. In the algorithm then, the UAV is considered as a box moving in the Space with dimension equal to 80x80x50 centimeters. This can be easily seen by using the Rviz software to obtain a graphical representation like in the *figure 4.9*.

Once the information regarding the size of the state space and the coordinates of

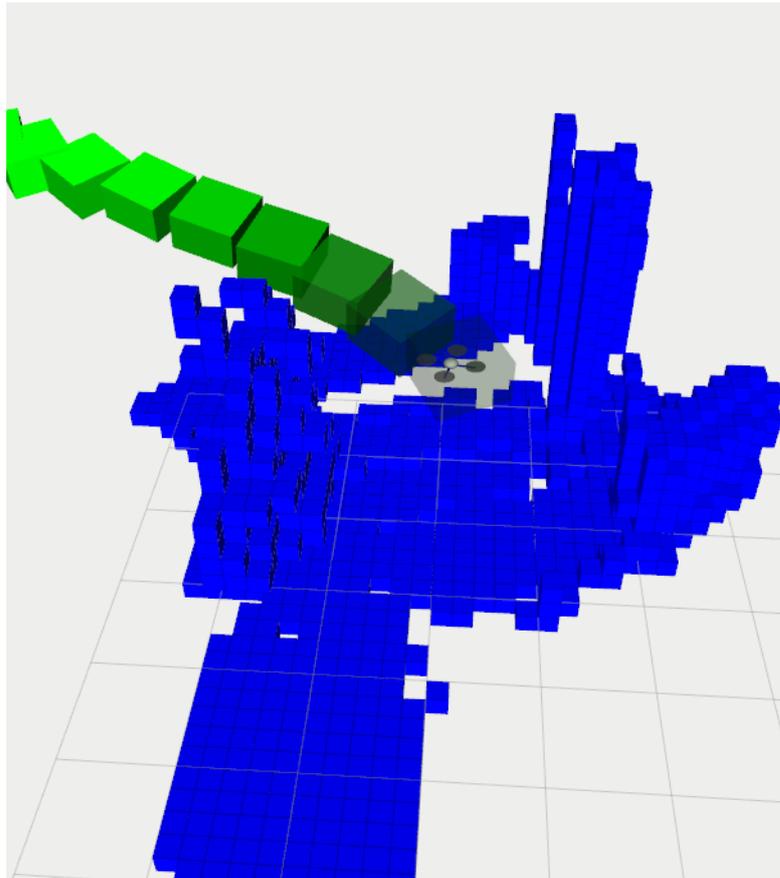


Figure 4.9: Drone object seen in Rviz as a box

Goal and Start points are received, the planner can be called through the OMPL library. In the definition of the object planner, it is mandatory to specify which kind of algorithm has to be used by it between the ones contained in the library. In addition, it is possible to modify the behavior by simply changing the standard parameters.

The code developed deploys the standard Informed-RRT* algorithm as a planner, the only changes have been made with a view to increasing the performance of the planner from a speed point of view. The main alterations applied are summarized below:

- *setRange*: this parameter greatly influences the run-time of the algorithm. It represents the maximum length of a motion to be added in the tree of motions. For this reason a proper value has to be set in order to obtain the best performance achievable. By trial-by-error the best value was hard-coded as 4 meters.

- *setKNearest*: it enables if the states are obtained by using a k-nearest search for rewiring. If disabled the speed of planning is greatly increased. For this code this function was disabled and the standard rewiring was utilized.
- *setRewireFactor*: this parameter controls the radius in which the rewiring action takes place. By definition, smaller rewiring neighbourhoods reduce the overall computational cost of rewiring at each iteration improving the performance of Informed RRT* while maintaining almost-sure asymptotic optimality. For this application the range was defined at 1.5 meter.
- *fixInvalidInputStates*: define a range where the goal and starting point can be redone.

Once the planner has resolved the path-problem, a *for* cycle is called in order to publish the waypoints that form the trajectory to the ROS topic `/mavros/setpoint_position/local` to control the UAV and make it maneuvers to the desired position.

To maintain a more precise control and prevent any conflict between the messages, one point at a time is sent. Before sending the next one the algorithm checks the UAV current position, if it is closer than a define euclidean distance then it will proceed to send the next waypoint, otherwise it will keep waiting for the drone to reach the previous one. This distance is fixed but can be changed by directly modifying the code. By experience, the one which provides the better stability of flight and security is 15 centimeters.

Additionally, each time the Octomap is updated or a waypoint is reached an auxiliary function is called from the FCL library which examine the validity of the generated path by searching for any collision between the object Drone and the occupancy map. If a collision is detected the *Replanning* function is called, in which the actual trajectory is erased and another instance of the Planner is called to generate an updated one that avoid the incoming obstacle.

The algorithm will run indefinitely and can even accept another Goal point while still proceeding to send all the trajectory for another one, this gives a certain amount of flexibility in the application of the code for scheduling UAV's missions. The resolution of the map plays a major role in the reactivity and in localization of any obstacles, this due to the onerous computational cost to generate a 3D environment. To work around this problem, it was lowered until each Cube of the Octomap represents in real-life a 10 centimeters block. This simplification affects the collision avoidance, due to the approximation, slight vibrations during the flight or the UAV not following correctly the trajectory not represent any problem due to the wide tolerances taken into account.

With the Rqt software is possible to visualize the ROS-Graph generated by launching all the required ROS-nodes in order for the algorithm to work and the reference frame dependencies (*figures 4.10, 4.11*).

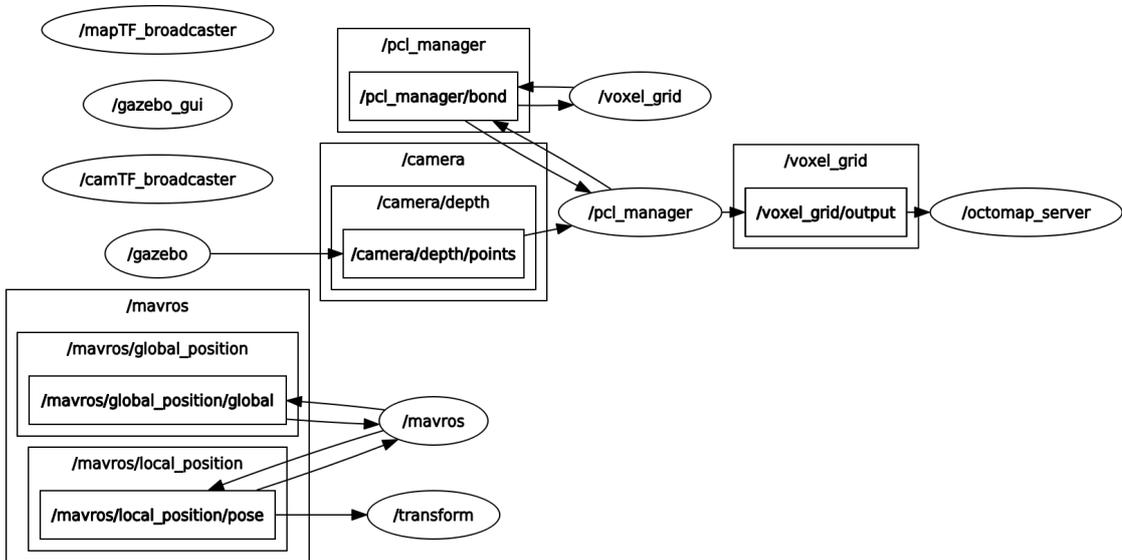


Figure 4.10: Rosgraph of all the Nodes implemented

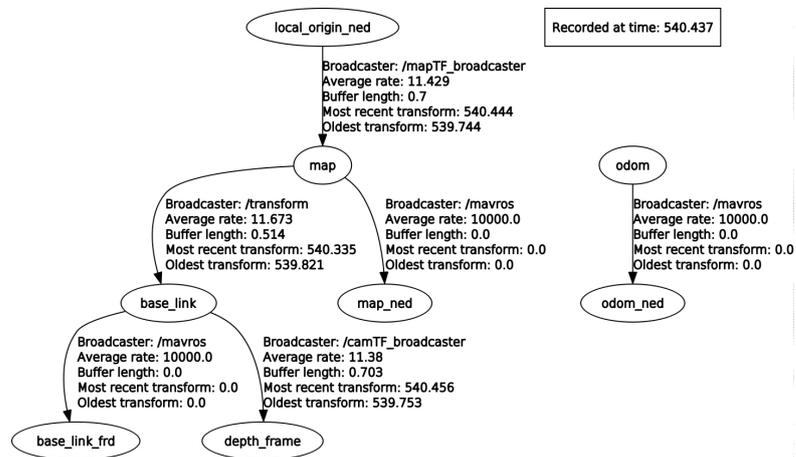


Figure 4.11: Reference Frames dependencies

Chapter 5

Virtual Simulation

The virtual simulation of the robot's behavior constitutes a critical step during the development phase. The main advantage is to be able to freely test the code and any changes without having to use directly the real UAV, avoiding in this way the exposure to possible security risks, accidents or damage to the components.

Thanks to Gazebo is possible to generate maps able to represent approximately the real world and its physic, but to obtain meaningful results from the simulation, it is mandatory to use a special software able to recreate the real FCU's real behavior. These kind of development platforms are called in short SITL, which means Software in the Loop. The Ardupilot community has already a working implementation of SITL software able to operate in combination with ROS and Gazebo to create a powerful tool in the development of open-source applications.

5.1 SITL and Mavros implementation

The SITL (software in the loop) is a simulator that allows to run Plane, Copter or Rover without using any hardware. It is built entirely on the autopilot software and allows to use Ardupilot on a computer virtually implementing all the flight stack behind the FCU. In this way it is possible to emulate the Pixhawk even though no hardware is used. When launched, SITL gather all the data incoming from the flight simulator, in our case Gazebo, elaborate them through the Flight Stack and returns a behavior close as possible to the real one.

In addition, SITL can be used together with the Mavros node, thus obtaining directly the control of the drone through simple ROS commands and the possibility to experiment and implement even complex codes exploiting the platform provided by ROS.

To start a simulation therefore, it is necessary to initialize the Gazebo world containing the UAV model, then SITL to start the virtual FCU and then connect

to it via the Mavros node. The advantage of using Mavros is that it is not necessary to express explicitly the various transformations between the reference frames used, that's because the links and transformation matrices are automatically generated by the node, greatly simplifying the design and testing phase of the code.

Once all connections have been successfully established, it is possible to launch any Ros-node to test its functionality with the drone and control the Ros-Topic and Ros-Messages exchanged using the Rviz (ROS-Visualizer) software.

Thanks to the Gazebo environment it is possible to introduce errors on the

The image shows two overlapping terminal windows. The top window, titled 'Console', displays MAVProxy status information including vehicle type (ArduCopter), GPS status, battery level (100%), and various sensor readings. It also shows MAVProxy messages such as 'AP:EKF2 IMU0 is using GPS' and 'Got COMMAND ACK: GET_HOME_POSITION:ACCEPTED'. The bottom window, titled 'Terminale', shows the execution of a script to launch the SITL environment, including commands for running MavProxy and connecting to the MAVROS node. The terminal output shows the successful connection and the MAVROS node entering the STABILIZE mode.

Figure 5.1: SITL console with Mavros node launched

measurements, the simulation can mirror in this way the same behavior in real-life application. In order to do so, some tweaks and modification on the parameters of the "virtual" drone have to be done:

- *'lib_gps_gazebo_plugin.so'*: setting the *Gpsnoise* parameter to *true* enables a Gaussian noise on the measurements of the GPS sensor in the simulation, in this way the localization of the UAV in the Gazebo environment becomes less precise and the drone might not follow with perfect accuracy the path sent by the planner.
- *SIM_WIND_SPD*: operating on these parameters present in SITL allows to generate a wind simulation on the drone flight, choosing intensity, direction and type of wind decay. These parameters greatly influence the type of response given by the flight stack.

After having carried out all the previous procedures, it is possible to send via Rviz the desired goal point and enter the final height required after which the process is

completely automated.

By reading the messages exchanged through the topics representing the UAV's position and the coordinates corresponding to the waypoints, the graphical representation of these data allowed to directly see the influences due to the disturbances introduced and the behavior of the script.

5.2 Virtual Environments for Testing

In order to test the collision avoidance's accuracy and reliability, multiple virtual environments were generated thanks to Gazebo. Starting with single obstacle map and subsequently moving toward more complex situation with fully automated dynamic obstacles in order to stress the performance of the replanning function and fine tuning the parameters of the code.

In the first map presented in the *figure 5.2*, although the surroundings seems empty, two people in movement with different speed are generated in front of the UAV. This kind of environment was specifically designed to test the ability of the Octomap to perceive the moving obstacles and set the best update speed for the mapping.

The second map in *figure 5.3*, was intended to test the planner's capability to

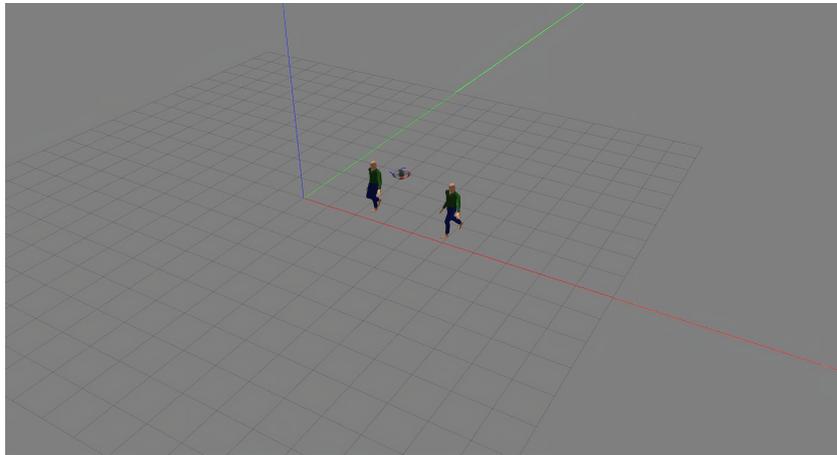


Figure 5.2: Two moving persons in front of the UAV during the flight

generate complex trajectory in a densely populated environment and the replanning function.

Finally, to add an additional grade of realism, a model of the CIM4.0's environment was created to test the UAV's indoor flight behavior (*figure 5.4*).

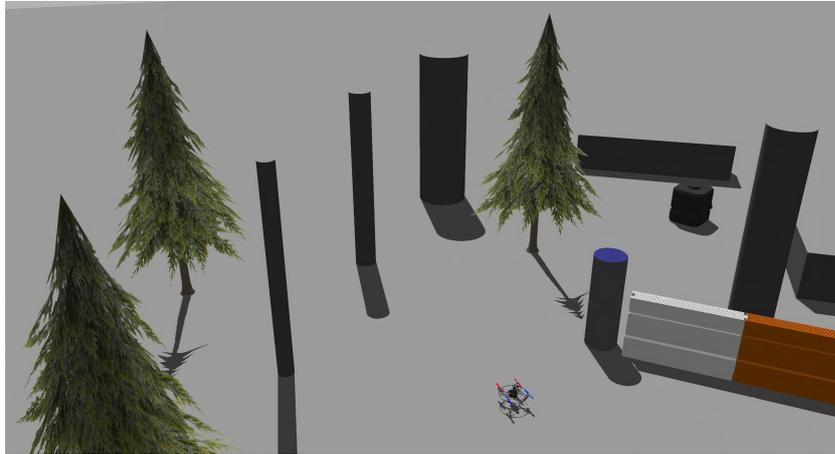


Figure 5.3: Map greatly populated of multiple obstacles



Figure 5.4: The CIM4.0's Digital Line replicated in Gazebo

5.3 Simulated Flight Analysis

The SITL simulation can support the generation of Data Flight logs file. These logs are stored and can be accessed and read using the standard APM planner or Mission Planner software. By doing so, all the data from a virtual mission can be retrieved and plotted using the implemented graphical tool.

However, despite this possibility, the application of software like MATLAB is recommended due to the greater completeness and flexibility for data analysis provided by it. For these reasons, a section of the collision avoidance algorithm is dedicated to generate a simple text file containing all the required data, acquired directly from the Mavros topics, to analyze the UAV's behavior.

In the following images, the relative coordinates of the UAV are represented. In this case, a simple mission was given setting the Goal Point around 15 meters forward the drone's starting position. The map used is the second one presented before, the one including different kind of obstacles, for this reason the resulting trajectory is not perfectly straight but presents some steering actions.

In the first *figure 5.5*, one must focus to the UAV's behavior in the early stages of flight. The TAKEOFF is one of the most crucial part of a flight, where the drone's motors goes full throttle in order to detach the aircraft from the ground. In the graph the bouncing behavior during the simulation is extremely similar to the one seen in real-life flight. This is generated by the "virtual" FCU, which tries to decelerate accordingly to the perceived UAV position and maintain a stable altitude. Still, after the achieved stability, the influence of the introduced noises on the GPS can be seen as the relative position contains small variations of around 2 centimeters that afflict the signal.

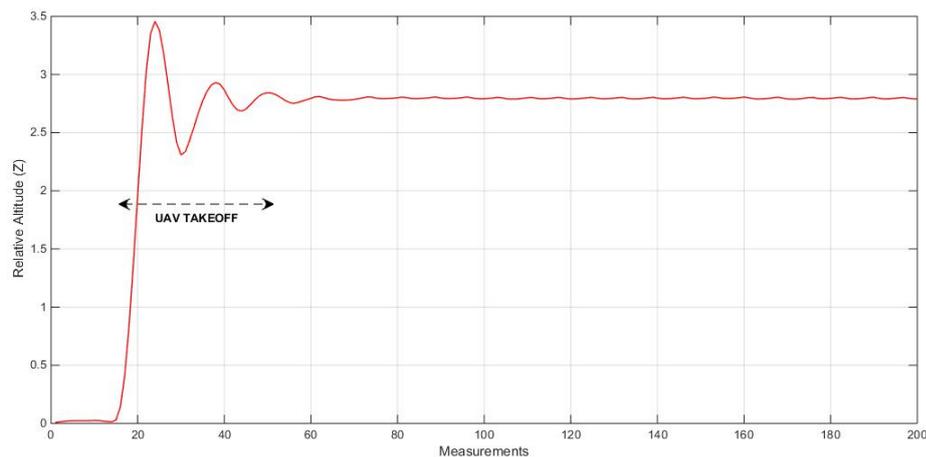


Figure 5.5: UAV's relative altitude in time during the Simulation

Of greater interest, in the *figure 5.6*, the actual influence of the added noise is clearly visible. In this image the UAV's relative position in the plane XY at each time frame is displayed. Additionally, all the waypoints generated by the script are replicated too. The steering direction that can be seen is due to the replanning function after the perception of an obstacle on the initial trajectory, for this reason the path is not perfectly straight.

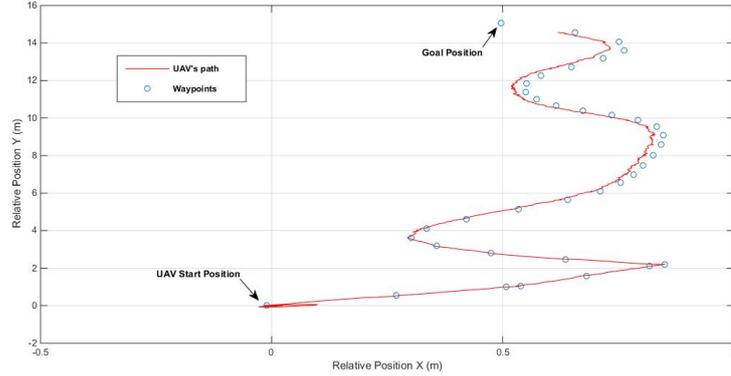


Figure 5.6: Path and Waypoints during a mission

The information are not only limited to the UAV's position, the attitude of the drone can be obtained by using the orientation quaternion (q) as odometry messages output and transforming it to the corresponding RPY angles. A simpler approach could be reading directly from the log files the angles information avoiding any possible error during the conversion.

Due to the simulation, the data obtained are almost noiseless and precise and no noteworthy behavior can be present. Focusing instead between the desired attitude and the actual one it is possible to notice how the FCU operates during the flight, sending a desired angle and compensating any possible discrepancy.

Focusing on the Yaw angle (*figure 5.7*), it is possible to see the steering action performed by drone in order to avoid a collision.

The UAV's heading changes multiple time in the first part of the graph with great speed, due to the fact that at each replanning the local position is reset with the initial heading. In reality the hardware remains stationary hovering while the planner is calculating the next path. Once the route is defined, in the second part of the graph, the behavior is more linear with small steering action. This is the case of the drone following the waypoints. Regarding instead, the Roll angle, few observations can be made. In fact, the navigation of the drone is so stable that small variations of it are ensured. This can be seen in the *figure 5.8*. Also in this case, the discrepancies between the desired Roll and the effective one are minimal thanks to the virtual simulation.

Of more interest, however, is the pitch angle's data, which can be seen in the *figure 5.9*. In this case, a sinusoidal behavior emerges from the graph. This small oscillations are generated from the subsequent accelerations and decelerations performed by the UAV while moving from a waypoint to another. While in the simulated environment this kind of maneuvering does not represent any danger, during the implementation in a real-life scenario it has to be taken into account. In a real mission, due to the potential environment's interferences, limiting the drone's speed of navigation and the acceleration directly from the Autopilot is a mandatory requirement to fly safely and without losing control of it.

To stress the system, difficult pathways are generated to see the ability to perceive trajectories within confined spaces or corridors and, eventually, react to incoming collisions. Thanks to the Gazebo environment, the overall process of generating 3D occupancy map is not suffering from loss of position or oscillation in the reference frame. The more precise is the process of stabilizing the frame thanks to the drone's localization the better will be the map produced. An higher accuracy allows to think about more dangerous missions with reduced maneuvering space. This is the case in the *figure 5.10*, where the UAV is capable of navigate a pathway inside a small corridor and avoid any collision.

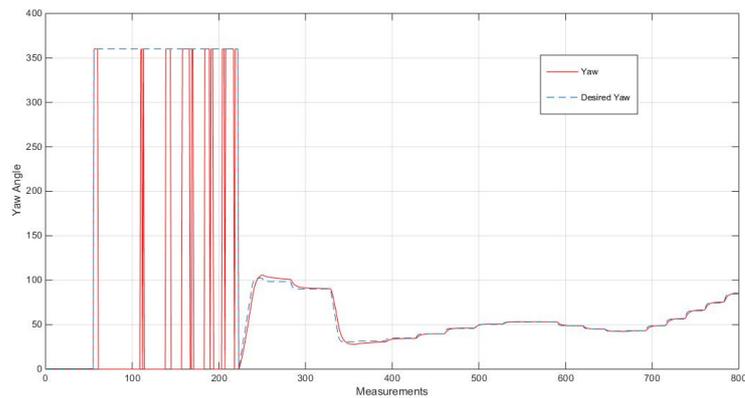


Figure 5.7: UAV's Yaw angle during a mission

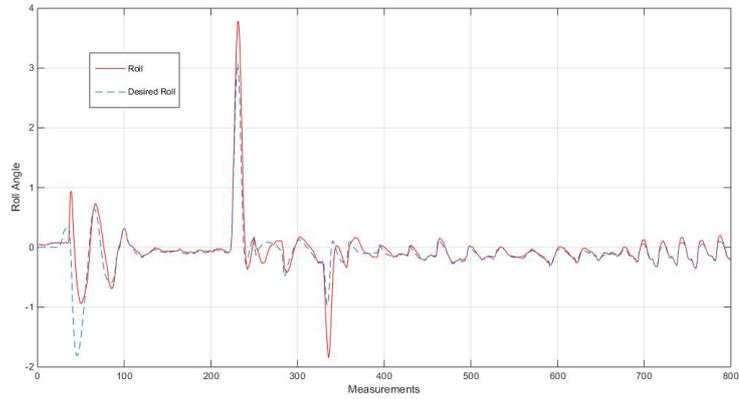


Figure 5.8: UAV's Roll angle during a mission

In the following sequence of figures (*figure 5.11*), instead, the replanning process of the trajectory is highlighted. As the map is generated and the presence of new obstacles reported to the planner, the waypoints sent are modified in order to operate evasive maneuvers.

Before the real-life implementation of the algorithm, as a final test, the study of the drone's behavior in an environment with multiple moving obstacles proved to be the most complex and challenging one. While the dynamism of the various obstacles is taken into account thanks to a continuous process of upgrading the map, the rate at which this occurs requires a fine tuning.

Too high an update rate could erase important features during the planning phase and not perceive an impending obstacle, it must be remembered that the creation of

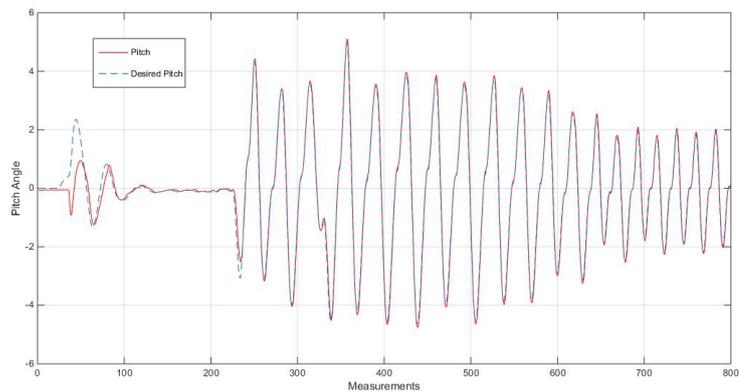


Figure 5.9: UAV's Pitch angle during a mission

occupancy map takes time because of the reduced computational power of the CC. Similarly, if the rate is too low, moving obstacles leave a trail that is treated as an obstacle, where, in reality, the space is clear. Therefore, lengthening or modifying the generated trajectory accordingly to a false information and increasing the cost. The rate was set in a range of possible values based on the required responsiveness, generally between 0.2Hz and 2Hz. Higher values may cause the ROS node to crash and the system to fail. The continuous map upgrade feature can be disabled in case the requirement is only to map the surroundings for other uses of the drone. Thus, simplifying the operation of the system.

Once all of these tests were successfully completed, it was possible to transfer all of the software architecture into the Jetson Nano and then proceed with real-life testing.

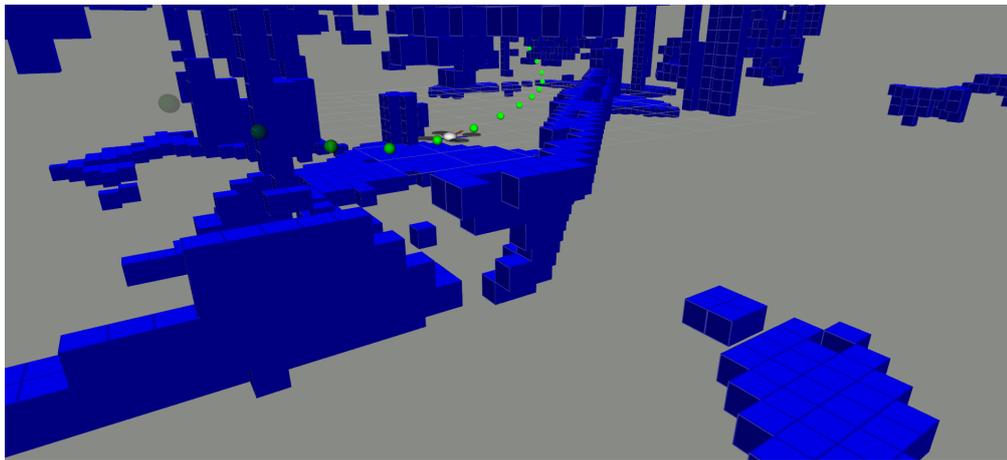


Figure 5.10: The Planner is able to generate trajectory through narrow passages

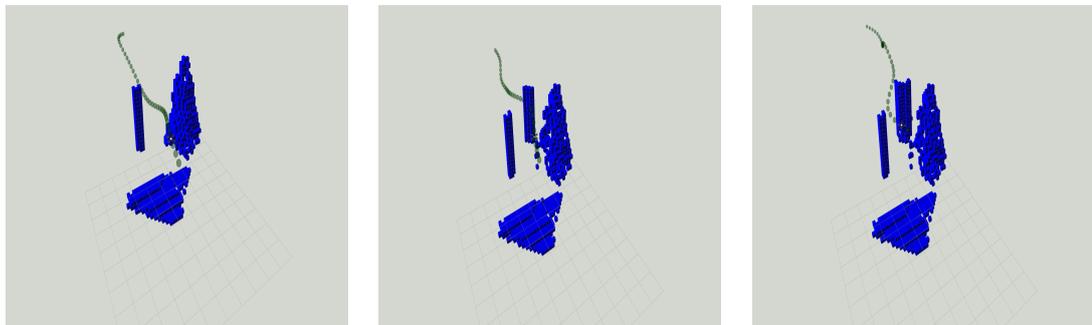


Figure 5.11: Re-planning process of a given initial path

Chapter 6

Experimental tests and results

In this last chapter, the analysis of the algorithm's real life testing and the drone's flight are carried out. Flight in real life corresponds to an increase in difficulties and issues due to noise and to an increase in errors and uncertainties. These issues become even greater for indoor flight where, the lack of GPS signal and tight spaces require increased accuracy. Safety measures should be taken such as activating the fail-safe mechanisms to safeguard the drone from damage due to accidental falls or impacts.

The testing procedures are performed in the two different environments in which the UAV will be deployed. For the indoor flight, a cage that includes UWB anchors while for the outdoor ones a wide open space in the CIM4.0's surroundings is used.

6.1 Outdoor Flight

Outdoor flying constitutes the starting point for subsequently moving towards indoor one. This flight mode is the most widespread and easily implemented in the UAV's industry. Thanks to the use of the GPS sensor, locating the drone with accuracy is not regarded as a concern. The wide operating spaces also increase the tolerance for interferences or possible manoeuvring errors.

Before actually using the algorithm for the collision avoidance, some tests to verify the drone's flight stability and its accuracy in holding a defined position have been performed.

Initially the connection via UART port between the companion computer (CC) and the flight controller (FCU) was established, then connecting through SSH to the CC from the Ground Computer (any PC connected to the same network) it was possible to initialize the Mavros node to control the drone remotely thanks to

ROS. Once all these operations are completed, the command to reach a set altitude is sent. In the following figures, the localization data obtained from the Log files are represented graphically.

In the *figure 6.1*, the UAV's relative altitude is shown. The drone's ability to maintain a predetermined altitude comes from the fusion of multiple sensors such as IMU, GPS and Barometer. In this case, the given altitude of 1.2 meter was set in order to detach from the ground the aircraft and subsequently submit a goal position using Rviz.

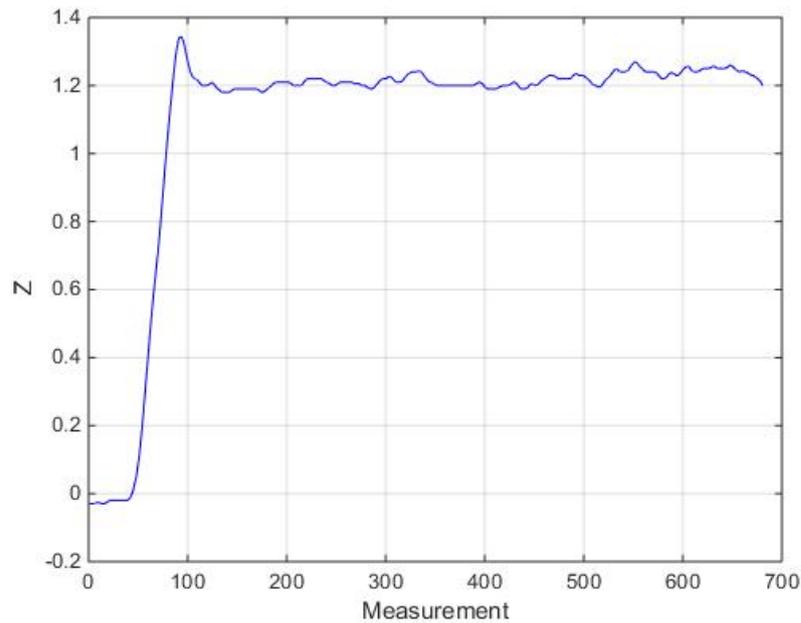


Figure 6.1: UAV's relative Z position

In this second image, *figure 6.2*, the relative longitude and latitude are reported. Although the drone in Loiter mode should automatically attempts to maintain the location, heading and altitude, it is possible to notice a shift caused by the initial overshoot during takeoff and the subsequent damping attempt performed. Despite these fluctuations, the aircraft stabilizes itself locking its position within a tolerance range.

Thanks to the various connections previously established, it was possible to use Rviz to read the ROS topics and visualize the different reference frames (Map, Drone) and the octomap generated in real-time.

To start a mission the only requirement is to send a 2D navigation goal using the software and subsequently, insert the desired final altitude in the Planner terminal opened on the ground PC. Once the UAV has detached itself from the ground and

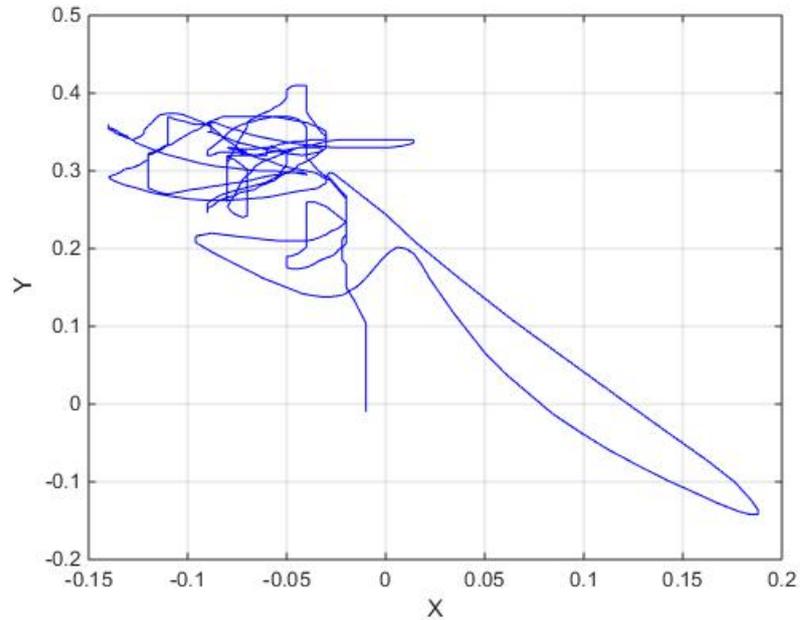


Figure 6.2: UAV's relative XY position

reached the desired altitude, a simple Goal position was given as shown in *figure 6.5*.

Although the waypoints were successfully generated by the planner, as shown in the graph containing the waypoints sent to the FCU *figure 6.4*. The drone's

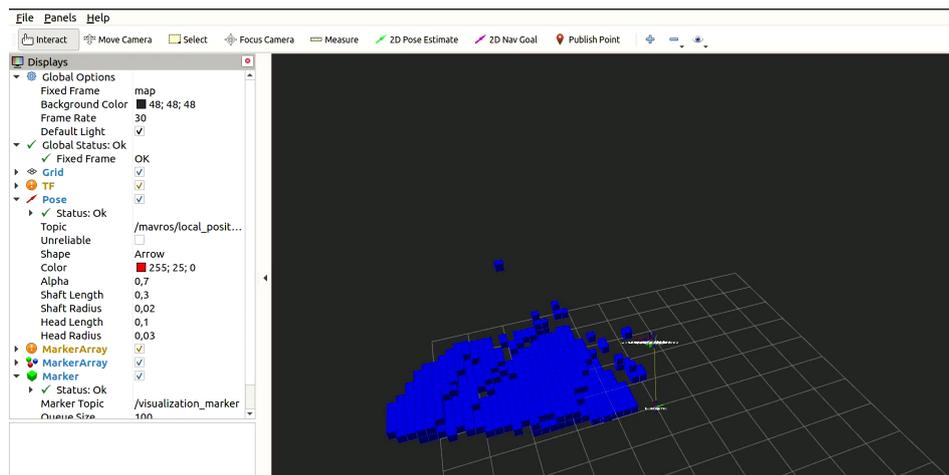


Figure 6.3: Rviz showing the Octomap and Tf-frame in real-time

high speed and its acceleration did not allow it to accurately follow the trajectory. Additionally, the unstable frame and weight of the battery have accentuated the effect of vibrations affecting the flight.

Unfortunately, the drone was unable to carry out the mission successfully due to a strong impact against an obstacle before arriving to the goal position. As a result of this damage, the practical tests were abandoned due to a series of problems on UAV's hardware side. Before proceeding with further tests, the platform must be stabilised and flight reliability improved.

In spite of this, it can be stated that the entire designed software structure is functional and the methodology fully applicable to the Pixhawk and ROS ecosystem. In fact, the algorithm still generated an occupancy map and a trajectory in response to the presence of an obstacle to prevent a collision. For this reason, by relying on the results obtained from the simulation, where the software applied is exactly the same as that on the UAV, it is possible, with a better drone model, to achieve the same results in real life.

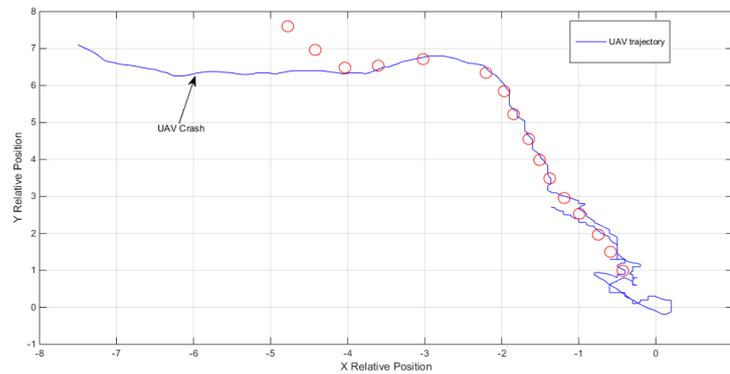


Figure 6.4: UAV's trajectory and waypoints during a flight test

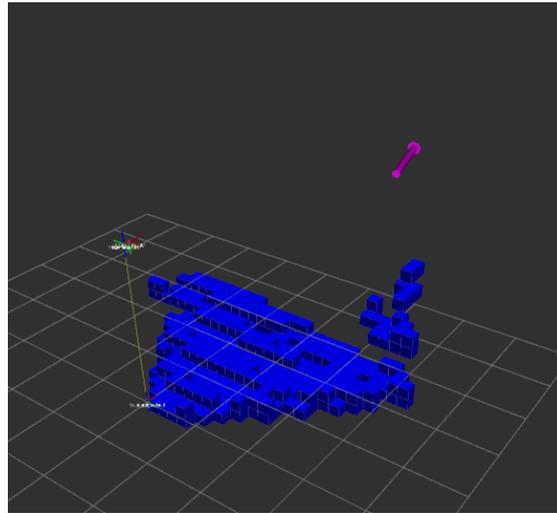


Figure 6.5: Goal Point sent thanks to Rviz

To test the accuracy of the GPS in following a given trajectory and the repeatability in traveling the same path over and over again, a given route was taken 20 consecutive times. As shown in the *figure 6.6*, the navigation's data showcases how the influence of vibration and the noises greatly impact the precision. For these reasons, the integration of a RTK-GPS in future should be considered in order to maintain a reliable UAV's localisation.

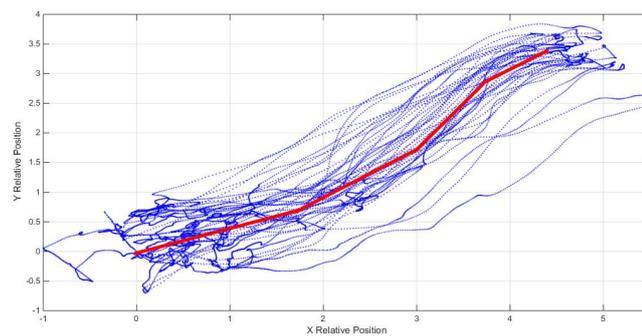


Figure 6.6: Multiple GPS tests to examine the accuracy of the position estimation. In Red the actual path while in Blue the position obtained through the GPS data.

6.2 Indoor Flight

Indoor flight presents considerable challenges, especially in terms of spatial localisation and determining the attitude of the drone.

In this condition no GPS signal is reliable, even if the correct number of satellites and a low HDOP are achieved, due to the multipathing effect, the signal is reflected to the antenna via walls, windows and other surfaces.

The implementation of UWB technology, exploiting several anchor nodes positioned within the test cage, solves this issue completely with a good level of accuracy. Thanks to the log files, it is possible to visualize the precision achieved in estimating the UAV's position.

Unfortunately, due to strong electromagnetic interference in the test room, some problems arise on how the magnetometer works. For this reason, the attitude and the heading of the UAV's estimate are not accurate for the indoor flight. An attempt to solve this problem was made by implementing a fictional reference frame generated by the sensor fusion of the UWB, Visual odometry and IMU.

Generally, flying the drone in LOITER mode indoors generates the same widespread issue present in drones without a good calibration. The UAV can't maintain a stable position in space but it starts moving in a circular motion, as the errors keep accumulating the radius increases until the aircraft is unable to maintain a stable flight and crashes.

Due to this issue, the testing of collision avoidance algorithm's behavior in indoor environment has not been extensively tested. The additional odometry generated by the sensor fusion can be used thanks to the *vision position* messages and subsequently sent to the flight controller. Disabling the Compass use in the EKF is mandatory to ensure that the filter does not implement any data coming from the compass.

This kind of implementation though generates a growth in complexity, due to the setting of covariance matrix parameters and filter sampling rate, all including possible sensor measurement errors. A fine tuning could provide a stable and reliable system to navigate in indoor environment.

A possible easier solution could be represented by the implementation of a Nooploop for Non-GPS Navigation. Still based on the UWB technology, this kind of system is specifically designed to work with the Ardupilot firmware in order to provide both a position and attitude estimate to substitute the GPS and Compass in indoor flight.

Chapter 7

Conclusions and future Developments

The aim of the thesis is to develop an algorithm capable of instantly recognising impending obstacles and reacting by generating safe trajectories for the drone to enable autonomous flight can be said succeeded.

Although the real-life flight attempts were affected by the poor stability of the UAV itself and the impossibility of carrying out extended tests, thanks to the use of the open-source platforms like SITL and Mavros, the resulting framework can be said to be fully functional for any drone capable of adopting the technologies listed in the thesis.

Software in the loop emulates exactly the behaviour of Pixhawk FCU, for this reason the methodology developed can be said to be perfectly applicable and functional. Additionally, the highly flexible parameterisation adopted allows the algorithm to be used on several UAVs with different sizes and dimensions, while still maintaining all the reliability and the functioning principles untouched.

In the future, the implementation of a more powerful on-board computer would also open up the possibility of considerably increasing the operating frequencies of the entire framework, therefore enabling an even more reactive system capable to cope to high-speed dynamic obstacles. Subsequent developments could integrate better FCU from the Pixhawk's family, such as the Cube Orange FCU, which provides better performance, stability and flight accuracy.

The ease of use of the system allows it to be utilized by unqualified personnel. With Rviz software, the user merely has to choose the end position with the mouse and enter the required height. While, the use of libraries such as OMPL allows extreme flexibility; the same methodology could be employed for a Rover, the only modification is to remove the third dimension in the definition of the path planning problem.

To expand the opportunities, the implementation of machine learning algorithms to recognise objects, people or any criticalities in a plant could bring this project much closer to the products currently on the market. In order to achieve this, the use of computers with a discrete GPU like the Nvidia Xavier or Intel Nuc could be considered as possible alternatives.

The development of indoor flight, on the other hand, presents considerable criticalities that could be resolved in numerous ways. The Ardupilot platform includes a number of products already available on the market and integrated in the autopilot's firmware which have not been tested in this thesis, but could considerably increase the stability of the drone. The use of camera solely designed for the visual odometry in combination with the one already present, like the Intel Realsense T265, the deployment of Vicon markers or the implementation of Optitrack systems; these are just some examples of possible solutions applicable.

Thanks to the use of ROS as middleware, the possibilities of interfacing the UAV with the AGV are manifold; cross-communication between the two platforms can vastly simplify all maintenance phases. In fact, the development of an autonomous landing does not require much effort, as the two systems are connected on the same local network and can communicate their respective positions in real-time and consequently locate each other in space.

The addition of UAV's custom-made power-board, allows the continuous recharging of the drone while it is attached to its base and not necessary to fly it; this feature can be fully exploited imaging a swarm of drones capable of performing maintenance and surveillance continuously 24 hours a day in a plant.

The framework developed can be fully converted to ROS2 and use a more recent version of Ubuntu with updated libraries and security patch. The project foreseen the use of voice control methodologies for the drone, as well the possibility of adding customised sensors according to the required application, such as thermal cameras or gimbal.

The future of the FIXIT project is in fact limited merely by the imagination of the end user, the platform offers incredible potential and flexibility to meet the most demanding requirements and it is more than capable of represent a glimpse of what will be the future of the industry 4.0.

Bibliography

- [1] Timothy D. Barfoot Jonathan D. Gammell Siddhartha S. Srinivasa. «Informed RRT*: Optimal Sampling-based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic». In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems (2014)*, pp. 2997–3004.
- [2] Jawad N. Yasin, Sherif A. S. Mohamed, Mohammad-Hashem Haghbayan, Jukka Heikkonen, Hannu Tenhunen, and Juha Plosila. «Unmanned Aerial Vehicles (UAVs): Collision Avoidance Systems and Approaches». In: *IEEE Access* 8 (2020), pp. 105139–105155. DOI: 10.1109/ACCESS.2020.3000064.
- [3] Jonathan D. Gammell, Siddhartha S. Srinivasa, and Timothy D. Barfoot. «Informed RRT*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic». In: *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems*. 2014, pp. 2997–3004. DOI: 10.1109/IROS.2014.6942976.
- [4] Juraj Oršulić, Robert Milijas, Ana Batinovic, Lovro Markovic, Antun Ivanovic, and Stjepan Bogdan. «Flying with Cartographer: Adapting the Cartographer 3D Graph SLAM Stack for UAV Navigation». In: *2021 Aerial Robotic Systems Physically Interacting with the Environment (AIRPHARO)*. 2021, pp. 1–7. DOI: 10.1109/AIRPHARO52252.2021.9571065.
- [5] Lorenz Meier, Petri Tanskanen, Friedrich Fraundorfer, and Marc Pollefeys. «PIXHAWK: A system for autonomous flight using onboard computer vision». In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 2992–2997. DOI: 10.1109/ICRA.2011.5980229.
- [6] Yang Yu, Wang Tingting, Chen Long, and Zhang Weiwei. «Stereo vision based obstacle avoidance strategy for quadcopter UAV». In: *2018 Chinese Control And Decision Conference (CCDC)*. 2018, pp. 490–494. DOI: 10.1109/CCDC.2018.8407182.
- [7] Claudio Sciortino and Adriano Fagiolini. «ROS/Gazebo-Based Simulation of Quadcopter Aircrafts». In: Sept. 2018, pp. 1–6. DOI: 10.1109/RTSI.2018.8548411.

- [8] António Raimundo, D. Peres, N. Santos, Pedro Sebastião, and Nuno Souto. «USING DISTANCE SENSORS TO PERFORM COLLISION AVOIDANCE MANEUVERS ON UAV APPLICATIONS». In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences XLII-2/W6* (Aug. 2017), pp. 303–309. DOI: 10.5194/isprs-archives-XLII-2-W6-303-2017.
- [9] J. Borenstein and Y. Koren. «Real-time obstacle avoidance for fast mobile robots in cluttered environments». In: *Proceedings., IEEE International Conference on Robotics and Automation*. 1990, 572–577 vol.1. DOI: 10.1109/ROBOT.1990.126042.
- [10] Igor Shapovalov, Victor Soloviev, Valeriy Finaev, Evgeny Kosenko, and Jury Zargaryan. «Research of graph-analytical methods for a vehicle motion planning». In: Oct. 2015, pp. 585–591. DOI: 10.1109/ICCAS.2015.7364986.
- [11] Chunxi Cheng, Qixin Sha, Bo He, and Guangliang Li. «Path planning and obstacle avoidance for AUV: A review». In: *Ocean Engineering* 235 (Sept. 2021), p. 109355. DOI: 10.1016/j.oceaneng.2021.109355.
- [12] Simon Vanneste, Ben Bellekens, and Maarten Weyn. «3DVFH+: Real-Time Three-Dimensional Obstacle Avoidance Using an Octomap». In: vol. 1319. July 2014.
- [13] Rethnaraj Rambabu, Muhammad Bahiki, and Syaril Azrad. «Relative position-based collision avoidance system for swarming UAVS using multi-sensor fusion». In: *Journal of Engineering and Applied Sciences* 10 (Nov. 2015).
- [14] Bo Wang. «Path Planning of Mobile Robot Based on A* Algorithm». In: *2021 IEEE International Conference on Electronic Technology, Communication and Information (ICETCI)*. 2021, pp. 524–528. DOI: 10.1109/ICETCI53161.2021.9563354.
- [15] Fahad Islam, Venkatraman Narayanan, and Maxim Likhachev. «Dynamic Multi-Heuristic A*». In: *2015 IEEE International Conference on Robotics and Automation (ICRA)*. 2015, pp. 2376–2382. DOI: 10.1109/ICRA.2015.7139515.
- [16] Chunyu Ju, Qinghua Luo, and Xiaozhen Yan. «Path Planning Using Artificial Potential Field Method And A-star Fusion Algorithm». In: *2020 Global Reliability and Prognostics and Health Management (PHM-Shanghai)*. 2020, pp. 1–7. DOI: 10.1109/PHM-Shanghai49105.2020.9280929.
- [17] Sertac Karaman, Matthew R. Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. «Anytime Motion Planning using the RRT*». In: *2011 IEEE International Conference on Robotics and Automation*. 2011, pp. 1478–1483. DOI: 10.1109/ICRA.2011.5980479.

- [18] Xiao-Huan Liu, De-Gan Zhang, Hao-Ran Yan, Yu-ya Cui, and Lu Chen. «A New Algorithm of the Best Path Selection Based on Machine Learning». In: *IEEE Access* 7 (2019), pp. 126913–126928. DOI: 10.1109/ACCESS.2019.2939423.
- [19] *Gazebo Simulator*. URL: <http://gazebo.org/>.
- [20] *ROS Melodic*. URL: <http://wiki.ros.org/melodic>.
- [21] *Ardupilot*. URL: <https://ardupilot.org/>.
- [22] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. «OctoMap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees». In: *Autonomous Robots* (2013). Software available at <https://octomap.github.io>. DOI: 10.1007/s10514-012-9321-0. URL: <https://octomap.github.io>.
- [23] Ioan A. Sucas, Mark Moll, and Lydia E. Kavraki. «The Open Motion Planning Library». In: *IEEE Robotics Automation Magazine* 19.4 (2012), pp. 72–82. DOI: 10.1109/MRA.2012.2205651.
- [24] Jia Pan, Sachin Chitta, and Dinesh Manocha. «FCL: A general purpose library for collision and proximity queries». In: *2012 IEEE International Conference on Robotics and Automation*. 2012, pp. 3859–3866. DOI: 10.1109/ICRA.2012.6225337.