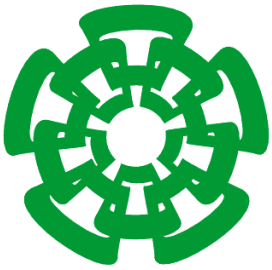


Centro de Investigación y de Estudios Avanzados
del Instituto Politécnico Nacional
Unidad Guadalajara

# Navegación Autónoma de Sistemas Multi-Agentes en Ambientes Desconocidos

Tesis que presenta:

**Leobardo Emmanuel Campos Macías**

para obtener el grado de:

**Doctor en Ciencias**

en la especialidad de:

**Ingeniería Eléctrica**

Directores de Tesis

**Dr. Antonio Ramírez Treviño**
**Dr. David Gómez Gutiérrez**

CINVESTAV del IPN Unidad Guadalajara, Guadalajara, Jalisco, Diciembre de 2021.

# Navegación Autónoma de Sistemas Multi-Agentes en Ambientes Desconocidos

**Tesis de Doctorado en Ciencias**
**Ingeniería Eléctrica**

Por:
**Leobardo Emmanuel Campos Macías**
Maestro en Ciencias de la Ingeniería Eléctrica
Centro de Investigación y de Estudios Avanzados del
Instituto Politécnico Nacional 2013-2015

Becario de Conacyt Expediente no. 543862

Directores de Tesis
**Dr. Antonio Ramírez Treviño**
**Dr. David Gómez Gutiérrez**

CINVESTAV del IPN Unidad Guadalajara, Diciembre de 2021.

Centro de Investigación y de Estudios Avanzados

del Instituto Politécnico Nacional

Unidad Guadalajara

# Multi-Agent Autonomous Navigation in Unknown Cluttered Environments

A thesis presented by:
**Leobardo Emmanuel Campos Macías**

to obtain the degree of:
**Doctor in Science**

in the subject of:
**Electrical Engineering**

Thesis Advisors:
**Dr. Antonio Ramírez Treviño**
**Dr. David Gómez Gutiérrez**

**CINVESTAV del IPN Unidad Guadalajara, December 2021**

# Multi-Agent Autonomous Navigation in Unknown Cluttered Environments

**Doctor of Science Thesis**
**In Electrical Engineering**

By:
**Leobardo Emmanuel Campos Macías**
Master of Science in Electrical Engineering
Centro de Investigación y de Estudios Avanzados del Instituto Politécnico Nacional 2013-2015

Scholarship granted by CONACyT no. 543862

Thesis Advisors:
**Dr. Antonio Ramírez Treviño**
**Dr. David Gómez Gutiérrez**

**CINVESTAV del IPN Unidad Guadalajara, December, 2021.**

Dedicada a mis padres, a mi hermana, a mis abuelos y abuelas, y a mi prometida.

# Acknowledgments

Quisiera agradecer principalmente a mi mamá Rosa María Jimenez, a mi papá Leobardo Campos y a mi hermana Karla Campos por su apoyo incondicional durante toda mi vida y en todas las decisiones, este trabajo es más de ellos que mío.

A mis abuelos, en especial a mi abuelo Jesús Macías, por todas esas tardes que pasamos juntos en su taller reparando electrodomésticos, iniciando así mi pasión en la ingeniería. A toda mi demás familia por aportar lo necesario en los momentos adecuados.

A mi prometida Linda Valencia por su soporte emocional, comprensión y amor durante todo este viaje.

A mis asesores y mentores Dr. David Gómez y Dr. Antonio Treviño, por sus enseñanzas y correcciones a pesar de lo aferrado que puedo ser.

A mis sinodales por las correcciones enriquecedoras de este trabajo.

A todas las personas y profesores del Cinvestav, en especial a mis compañeros de maestría generación 2013.

A todos mis amigos, Alonso, Ricardo, Iram, Macedo, David, Vaca, Mitchel y Cobian por seguir conmigo a pesar de mis desfiguros.

A todos mis compañeros y excompañeros de Intel Guadalajara, en especial, Rafael de la Guardia, Rodrigo Aldana, Anthony Guzman y Ignacio Vilchis.

A Intel por permitirme los componentes ideales para desarrollar este trabajo, tiempo, dinero y materiales, ya que sin eso la mitad de esta tesis no se habría logrado.

Finalmente, a CONACyT por otorgarme la oportunidad de completar esta línea de investigación.

# Contents

# Resumen

El sistema de navegación autónoma de un vehículo aéreo no tripulado (UAV, por sus siglas en inglés) se compone de tres algoritmos principales: a) generación de una representación del entorno, b) prevención de colisiones y c) planificación y seguimiento de trayectorias. Los UAVs capaces de navegar autónomamente son grandes y cuentan con plataformas computacionales y sensores sofisticados que le permiten procesar grandes cantidades de información en poco tiempo. Sus aplicaciones se enfocan en espacios abiertos, por ejemplo, en tareas de vigilancia y agricultura. En tareas donde los espacios son estrechos, por ejemplo, búsqueda y rescate o inspección de estructuras industriales, este tipo de UAVs no pueden ser utilizados y se debe optar por el uso de micro-vehículos aéreos no tripulados (MAVs por sus siglas en inglés).

Por su tamaño reducido, las plataformas computacionales con las que cuenta un MAV son de bajo desempeño, pero sus necesidades de navegación autónoma siguen siendo las mismas que las de los UAVs, por lo que los sistemas de navegación de los UAVs no pueden ser utilizados, requiriendo el diseño de nuevos algoritmos que componen los sistemas de navegación de estos nuevos vehículos.

Para dotar a los MAVs con navegación autónoma, en este trabajo proponemos un sistema de navegación donde sus componentes han sido rediseñados para ser ejecutados en plataformas de bajo rendimiento. Estos nuevos algoritmos permiten alcanzar o incluso mejorar el desempeño de los que actualmente se utilizan en los UAVs. El nuevo algoritmo de generación de una representación del entorno se basa en obtener una partición volumétrica heterogénea del espacio y representarlo en memoria en una estructura tipo árbol octal (octree por denominación en inglés) que permite almacenarlo en tamaños de memoria reducidos, dotado con una novedosa función de acceso directo al árbol (hash) que permite de manera muy eficiente determinar si una región del espacio está libre. El siguiente algoritmo genera las trayectorias resolviendo colisiones y se tomando en cuenta las restricciones físicas del MAV y su dinámica, evitando el uso de algoritmos de optimización que son computacionalmente costosos. Adicionalmente, la propuesta se extiende para tomar en cuenta sistemas con muchos MAVs, permitiendo asignar prioridades.

El resultado es una propuesta de un sistema de navegación muy eficiente. Los experimentos muestran que dicho sistema obtiene una alta tasa de éxito, es decir que se genera una trayectoria de la posición actual del MAV a su posición final. Para mostrar la efectividad de nuestra propuesta, comparamos el sistema propuesto contra otros reportados en la literatura, usando ambientes simulados y reales (benchmarks) diseñados para evaluar a este tipo de algoritmos. Los resultados, tanto en simulación como con MAVs reales, muestran que la propuesta mejora a las ya reportadas en la literatura. Los resultados muestran que la propuesta implementa un sistema de navegación autónoma que se puede ejecutar en plataformas de cómputo y sensado limitadas y en entornos desconocidos.

# Abstract

The autonomous navigation system of an unmanned aerial vehicle (UAV) is typically composed of three main algorithms: a) mapping, b) collision avoidance, and c) trajectory planning. The UAVs endowed with autonomous navigation are usually large and equipped with high-end computers and sensors processing a vast amount of information a very short time. Their applications are focused on large outdoor spaces such as surveillance and agriculture. However, for tasks that must be carried out in narrow space volumes, such as search and rescue operations and industrial inspections, these types of UAVs are not suitable. In this later case, the micro air vehicles (MAVs) are the must adequate vehicles.

Due to their small size, the computing platforms and the available sensors on MAVs are limited, nevertheless their autonomous navigation requirements are still the same as the UAVs. Therefore, the existing navigation systems used in UAVs cannot be implemented in these small platforms, requiring further research on developing new algorithms to endow MAVs with the same autonomous navigation capabilities.

This work is concerned on the developing of these algorithms, proposing a framework for autonomous navigation of MAVs, where the algorithms have been developed from scratch to ensure the required performance during their execution on platforms with limited sensing and computing capabilities. The proposed novel algorithms dramatically improve the performance of those ones currently used in UAVs. The first proposed algorithm computes a heterogeneous partition of the work space, representing it in an octree structure. It allows to save a lot of memory space. To avoid collisions, the second introduced algorithm leverages a novel hash function that allows a direct access to the octree to determine when a space region is obstacle free, saving computational time and retrieving enough information for the collision avoidance at further path planning stages. Finally, based on the information provided by previous algorithm and the MAV dynamics and restrictions, the proposed trajectory planning algorithm efficiently computes a connected tree to the required final MAV position from the current one. It exploits a stochastic sampling-based method to reduce the computational load, saving computational time and memory. Additionally, we extend our framework by considering systems with many UAVs, allowing the assignment of priorities.

The final framework implements an efficient autonomous navigation system. Furthermore, the experiments show that this framework has a high success rate. Our proposal and others reported in the literature were tested in simulated and real environments, used as benchmarks in this area. The results show that our proposal performance better over the other ones, in many aspects, for example our approach presents no collisions, and it is executed in less time using less memory when compared with other approaches. The experiments conducted in this thesis represent the implementation of autonomous navigation in a computationally constrained platform in unknown cluttered environments demonstrated to date.

# Chapter 1

# Introduction

Cooperative multi-robot systems have recently received much attention due to advances in computing power, onboard sensing capabilities, and communication efficiency. As a result, they are an attractive solution for tasks that have the potential of being solved more efficiently than with a single robot. A clear example is in a search and rescue scenario, where multiple robots could cover a more significant space leading to finding the objective in a shorter time. Other applications for multi-agent systems are the inspection of industrial and civil structures (Jordan et al., 2018) and aerial cinematography (Bonatti, Zhang, Choudhury, Wang, & Scherer, 2020).

Due to their lightweight, small size, cost, and high maneuverability, multi-rotor Micro Aerial Vehicles (MAVs) allow easy access to environments to which no humans or other vehicles can't get access. Unfortunately, scenarios with limited-long range communications restrict MAVs from being remotely controlled by humans. Moreover, restricted computing power, GPS-denied scenarios, battery endurance, and limited onboard sensing capabilities make autonomous navigation of MAVs in these environments a challenging problem.

Enabling multiple MAVs to navigate in complex unknown environments autonomously requires essential capabilities, such as map building, trajectory planning, and cooperative collision avoidance, which must be performed onboard.

Although plenty of literature proposes methodologies for these capabilities, the extension to platforms with reduced computing power and limited sensing capabilities is not straightforward. In particular, the autonomous navigation of a group of MAVs, of less than one kilogram, to reach a set of goals in unknown cluttered environments, avoiding collisions with the environment and with each other, remains an open problem. To navigate narrow spaces in an agile way, small and lightweight platforms are desirable.

Even though there are several similar ongoing efforts in the robotics community, with a single (Gao, Wu, Lin, & Shen, 2018; Usenko, Von Stumberg, Pangercic, & Cremers, 2017) or groups of MAVs (Zhou, Wen, et al., 2021), the existing approaches present several drawbacks, such as producing dynamically unfeasible trajectories, getting stuck in pockets, handling only low obstacle density, moving outside the field of view, and requiring 360 sensing range, poor success rate, as we demonstrate later in Section 5.1.

This thesis aims to enable a group of MAVs of less than one kilogram with limited communication and controlled in a distributed manner to navigate safely autonomously and reach a set of given goals in unknown cluttered environments avoiding collisions with the environment and each other while fulfilling dynamic constraints with onboard sensing and computing capabilities, focusing on obtaining a high success rate, significantly superior than with existing approaches.

A high-level, distributed, or centralized algorithm that assigns goals to the MAVs is not the focus of this work. However, the framework proposed in this thesis is a solid base to solve applications, such as exploration or area covering. Therefore, these interesting extensions are suggested as future works.

## 1.1 State of the Art

The works of Shen, Michael, and Kumar (2011) and Fraundorfer et al. (2012) are among the earliest demonstrating autonomous MAV navigation. They showed that MAVs could follow waypoints in the currently estimated map, avoid obstacles, and complete exploration tasks in cluttered environments. Nonetheless, those early works fell short of demonstrating full autonomy since the former lacked onboard planning while the latter relied on offline mapping.

Advances in computing hardware and software, availability of robust yet lightweight sensors, such as depth cameras, and integrated vision-based localization modules, coupled with algorithmic developments, have recently enabled precise, aggressive maneuvers and navigation of MAVs in unknown environments, like the methods proposed by Thrun, Burgard, and Fox (2006); Tsardoulias, Iliakopoulou, Kargakos, and Petrou (2016), and by Cadena et al. (2016). Nevertheless, a framework combining elements from these fields to warrant fully autonomous navigation in general environments for compute-constrained MAVs is still missing since it has not been demonstrated to date.

We contrast our contributions with some related works. In particular, we focus first on contributions that present autonomous navigation for unknown environments, with onboard mapping, planning, and trajectory generation/execution. Next, we extend the discussion to multi-agent collision avoidance. Then, given the growing interest in deep learning methods in MAVs, we also summarize the state-of-the-art in this area. Finally, we also include a brief review of the platforms used to carry out the experiments reported in the selected methods, particularly the computing power and sensors used.

In this literature review, we contrast the different approaches of each method regarding mapping, path planning, trajectory generation, multiagent coordination, and computing platform used in the implementation. This qualitative comparison, exhibiting the advantages and disadvantages of each contribution, is summarized in Table 1.1. A detailed discussion of the table, along with the multiagent coordination discussion, is presented next.

### 1.1.1 Mapping

Recent work has advocated the use of 3D voxel grids to represent the environment. The authors in Cieslewski, Kaufmann, and Scaramuzza (2017a) use a regular 3D voxel grid map implementation, while the authors in Usenko et al. (2017) propose a robot-centric grid map implemented as a fixed-size three-dimensional circular buffer.

These works solve the exploration at high speeds by selecting data structures optimized for fast lookup and measurement insertion operations, at the cost of limiting the space that can be mapped. In contrast, as used by the authors in Oleynikova, Taylor, Siegwart, and Nieto (2018), spatial hash tables provide a sparse representation that can be dynamically expanded with constant time insertion and lookup.

Octrees map representation first proposed by Hornung, Wurm, Bennewitz, Stachniss, and Burgard (2013), provides a complete hierarchical data structure which, while sacrificing raw speed in single queries, can be advantageous for planning, as shown by Vespa et al. (2018). Linear octrees, as proposed by Gargantini (1982), trade-off reduced memory usage for speed by storing only leaf nodes instead of all tree nodes while still preserving the hierarchical structure of regular octrees. Combining these two features makes this a compelling data structure for navigation in unknown cluttered environments, where frequent replanning is necessary. Therefore, in our approach, we choose to implement a linear octree to represent our map, with a voxel hashing approach similar to the one used by Nießner, Zollhöfer, Izadi, and Stamminger (2013).

Various authors implement additional structures on top of the base map. Among the most popular are distance transformers, such as TSDF, EDT, ESDF. Those transformers are used to facilitate collision checking and to allow fast querying of distance information to nearest obstacles, as used by Gao et al. (2018); Han, Gao, Zhou, and Shen (2019); Han et al. (2019); Oleynikova, Taylor, Millane, Siegwart, and Nieto (2018); Usenko et al. (2017), and Lin et al. (2018). While those approaches are beneficial for planning, computation of such structures is costly and requires specialized graphics hardware or powerful CPUs to achieve real-time performance.

Some works use different data structures to represent local versus global space to reduce computation and memory requirements. In Mohta et al. (2018) the authors used a hybrid map formed by combining a local robot-centric 3D map with a 2D global map to escape dead ends. In Collins (2019) the local map is represented as a KD-Tree with a sliding voxel map, while a sparse topological graph represents the whole environment that has been explored. The latter is used to help the robot escape pockets by adding edges and nodes to the graph to capture potential pathways that the robot has passed but not taken. Finally, in Florence, Carter, Ware, and Tedrake (2018) a method to use a 2D global map to guide local exploration based on uncertainty-aware proximity queries for planning without any prior discretization of the data is proposed. While potentially useful, in more constrained platforms, hybrid approaches introduce additional software complexity, which in our case was not necessary.

### 1.1.2 Path Planning

The works of Papachristos, Khattak, and Alexis (2017); Selin, Tiger, Duberg, Heintz, and Jensfelt (2019), and Cieslewski, Kaufmann, and Scaramuzza (2017b) focus on pure exploration. In particular, Papachristos et al. (2017) builds a Rapidly-exploring random tree (RRT) with cost proportional to the number of new voxels to find the next goal and a second RRT to find a path to the selected viewpoint that minimizes the robot's pose and landmarks uncertainty. The work from Selin et al. (2019) insert nodes with high potential information gain in an RRT to use as planning targets when the agent has explored everything in its nearby surroundings. Instead of planning trajectories, the authors in Cieslewski et al. (2017b) use a reactive mode that generates instantaneous velocity commands towards frontiers.

The work of Oleynikova, Taylor, Siegwart, and Nieto (2018) also considers exploration but adds a goal-reaching objective to address the problem of getting stuck in local minima explicitly. They select the next goal by drawing the global goal with some probability. Otherwise, they set a local goal by finding an unoccupied point in the field of view, maximizing an approximate exploration gain.

The works in Gao et al. (2018); Mohta et al. (2018), and Lin et al. (2018) use the output of the planning stage to set up an optimization problem to produce safe, dynamically feasible trajectories. They all attempt to generate a safe corridor to constrain the optimization but differ significantly in their approach. In particular, in Gao et al. (2018) the use of a fast marching method in a velocity field based on ESDF to obtain a time-indexed path is proposed, which is expanded in a second stage to include the free cubes centered on each node along the path. The authors in Mohta et al. (2018) uses an A* to plan a path in a hybrid graph that links the voxels in both a local 3D map and a 2D global information map. A path modification step then shifts the original path away from obstacles using a bisector plane that intersects with connected polyhedra through each waypoint. Finally, the work of Lin et al. (2018) uses a global offline pathfinder to generate a guiding path. During navigation, the target for the local planner is selected near the guiding path, and A* is used to search a minimum distance path to it.

Unlike the other methods in our comparison, which generate plans based on maps, in Florence et al. (2018), the authors propose to use locally a map-less stochastic motion planner for choosing among a finite 3D motion primitive library, with global guidance provided by a 2D A* planner.

Regardless of the approach to generate the path, a critical aspect for collision-free navigation in unknown environments is imposing constraints on the motion plan to navigate within the current field of view towards the next waypoint. Our method incorporates this idea which was also used by the methods in Cieslewski et al. (2017a); Oleynikova, Taylor, Siegwart, and Nieto (2018), and Florence et al. (2018). In contrast, the method in Gao et al. (2018) assumed that the sensors onboard the robot cover an entire spherical region centered at the robot, which is hard to achieve in practice. Similar to the technique used in Papachristos et al. (2017), and Selin et al. (2019), we construct rapidly exploring random trees (RRTs) in the 3D space represented by the entire map. Furthermore, we also use an informed sampling approach similar to the method proposed in Lin et al. (2018) to speed up the search. To obtain further performance gains, we have implemented an efficient collision checking algorithm taking advantage of the structure provided by our map.

### 1.1.3 Trajectory Generation

The algorithm presented by Gao et al. (2018) proposes to bound the entire trajectory within the free space and its derivatives within the feasible space with hard constraints; they pose a constrained convex quadratic program (QP) using a Bernstein basis to represent the trajectory as piecewise Bezier curves. For each piece of the trajectory, the optimization solves a time scale factor and the control points of the curves minimizing the integral of the jerk. Similarly, the authors in Mohta et al. (2018) formulate a QP for trajectory generation where in addition to the usual constraints on velocity, acceleration, and jerk, they add a term in the cost function proportional to the square of the distance between the trajectory and the line segments of the modified path. The trajectory is represented as a 7th order polynomial

spline. For allocating time to each spline segment, which is critical to the feasibility of the QP and the quality of the resultant trajectory, they use the times obtained by fitting a trapezoidal velocity profile through the segments.

In contrast, the authors in Usenko et al. (2017) pose a local replanning problem as the optimization of a cost function formed by a term penalizing position and velocity deviations at the end of the trajectory, a collision cost, an integral cost over the squared derivatives, and a soft limit on the norm of time derivatives over the trajectory. The local trajectory is represented by a uniform quintic B-spline, which simplifies the computation of the cost terms, and the problem is solved using a nonlinear optimization package. Another approach is given by Lin et al. (2018). In this work, a nonlinear optimization problem is formulated using 8th order piecewise polynomials to represent the trajectory. They iterate over two optimization problems. The first problem iterates with only a collision cost. The second problem adds a smoothness cost on the fourth derivative of the position and a dynamical penalty on velocity and acceleration.

The method from Cieslewski et al. (2017b) uses a reactive mode to generate instantaneous velocity commands based on currently observed frontiers. For frontiers at the detection range of the depth sensor, the desired velocity will be the maximum velocity and point towards the unknown volume. In contrast, for frontiers closer to the robot, the desired velocity will be lower. On the other hand, the algorithm posed in Florence et al. (2018) search in a finite library for a 3D motion primitive which maximizes the Euclidean progress to the global goal, considering collision probabilities for entire paths in environments with only convex obstacles.

An important aspect of navigation in the field of view is ensuring that the camera axis is aligned with the direction of motion. Oleynikova, Taylor, Siegwart, and Nieto (2018), and Cieslewski et al. (2017a) attempted to address this restriction by implementing a velocity tracking yaw approach, but they don't guarantee that the generated yaw trajectories will meet angular dynamical constraints. Additionally, linear motion dynamical constraints of the robot must be handled explicitly to guarantee that it will remain in safe regions. In particular, the works of Cieslewski et al. (2017a); Selin et al. (2019), and Lin et al. (2018) do not handle dynamical constraints explicitly. Moreover, the methods proposed by Usenko et al. (2017) and Lin et al. (2018) solve optimization problems with soft limits, which generate smooth trajectories but may still not be dynamically feasible. Finally, in the algorithms proposed in Florence et al. (2018); Gao et al. (2018); Oleynikova, Taylor, Siegwart, and Nieto (2018); Papachristos et al. (2017), and Mohta et al. (2018) a constrained optimization problem is solved to handle positional dynamical constraints explicitly. However, solving optimization problems online is computationally expensive, even in the convex case. In our approach, the dynamical constraints are incorporated without the need for an optimization-based solution.

### 1.1.4 Multi-agent Collision Avoidance

Trajectory planning for multi-agent coexistence navigation has been a problem widely studied, and a large number of proposals can be found in the state of art. All these proposals can be grouped into two sets depending on whether the coordination is centralized or decentralized. Or, if there is communication between the agents or a state prediction module for neighboring agents is considered to get the robot's intentions. For this work, we assume communication within a limited radius between the agents and the algorithm is computed in a decentralized manner.

Optimal Reciprocal Collision Avoidance (ORCA), proposed by Van Den Berg, Guy, Lin, and Manocha (2011), is among the earliest works that achieved the coexistence of many agents in a shared workspace acting independently without communication with other robots. The method is based on the so-called velocity obstacles, the set of all relative velocities between two agents that will result in a collision at some horizon time. However, the agents oscillated at specific configurations or were trapped in a local minimum induced by the velocity projections. Snape, Van Den Berg, Guy, and Manocha (2011) further solved the oscillation problem. The main contribution is that they explicitly consider that the other robots also sense their surroundings and change their trajectories accordingly when projecting the velocities, in an algorithm called Reciprocal Velocity Obstacles (RVO). Nonetheless, the extension to 3D space is not trivial, and the lack of a global plan is still present.

Shome, Solovey, Dobson, Halperin, and Bekris (2020) proposed a method for coupled multi-robot problems, which provides desirable path-quality guarantees with computing efficiency, named dRRT*. They first build a roadmap for each robot and implicitly search these structures' tensor product in the composite space identifying the conditions for convergence to optimal paths using come heuristics. The reported metrics indicate obtaining high-quality paths quickly while escalating to a higher number of robots, where various alternatives fail. While this method prevents the local minimum problem and can be directly applied to 3D spaces, the dynamics of the robots are not considered. Not considering the dynamics of the robot and only their positions lead to the circumnavigation of agents for their avoidance.

Among the algorithms that consider the robot's dynamical model and the physical constraints are the methods proposed by Chen, Cutler, and How (2015) and by Luis and Schoellig (2019). Both approaches pose an optimization problem that creates a trajectory for the agents given a cost function and predefined conditions. Thus, obstacle avoidance is considered as a restriction in the optimization problem, causing that the static obstacles and surrounding agents to be modeled with a linear mathematical function. The former formulates the problem in a quadratic programming manner, while the latter uses model predictive control. We demonstrate later that both methods escalate poorly respecting the number of agents and the quantity of static obstacles.

Finally, the closest framework to this work is the proposal of Zhou, Wang, et al. (2021). They introduce a decentralized spatial-temporal trajectory planning based on an unconstrained control effort minimizer (MINCO). Their report ensures high-quality local planning for each agent subject to any constraint from the swarm coordination or safety requirements in cluttered environments. The local trajectory generation is formulated as an unconstrained optimization problem triggered by a decentralized asynchronous mechanism. The overall proposal extends their previous method proposed by Zhou, Wen, et al. (2021). They developed the problem under a gradient-based local planning framework, where collision avoidance is achieved by formulating the collision risk as a penalty of a nonlinear optimization problem.

We believe that the method from Zhou, Wang, et al. (2021) is an excellent candidate for comparison against our multiple agent extension; however, as of the writing date of this document, the code of the algorithm has not been released.

### 1.1.5 Deep Learning Approaches

Recently, deep learning methods have been developed for autonomous navigation of MAVs, either as end-to-end approaches such as the work by Li et al. (2018) and by Bonatti, Madaan, Vineet, Scherer, and Kapoor (2020) or in combination with traditional planning methods like the methods of Kaufmann et al. (2018) and by Kaufmann et al. (2019). These approaches have shown to be effective under structured environments, such as in drone racing, but face many challenges for navigation in unknown cluttered environments, mainly due to the difficulty to collect real-world data for learning robust behaviors.

Toward addressing this challenge, Loquercio, Maqueda, Del-Blanco, and Scaramuzza (2018) proposed a convolutional neural network for collision-free navigation through the streets of a city, using data collected by cars and bicycles in an urban environment. However, since navigation is not goal-oriented, no map is created. Also, the navigation is performed only in a two-dimensional space. Therefore, additional mapping, localization, and planning stages would need to be added for MAV navigation in general unknown cluttered environments.

Among the most popular methods for multiagent coexistence is the algorithm proposed by Riviere, Hönig, Yue, and Chung (2020). They present a Global-to-Local Autonomy Synthesis (GLAS) policy generation for multi-robot planning in a distributed fashion. Using a neural network architecture consisting of 5 feed-forward modules, they generate probably safe trajectories for the agents, taking into account the states of the surrounding agents and the static obstacles within a radius. In the end, to guarantee collision avoidance, the trajectories are rectified by a control barrier function. Also, considering a portion of the map leads to getting stuck in pockets scenarios if global planning is not considered, but codifying the map efficiently for neural networks is still an open problem. Serra-Gómez, Brito, Zhu, Chung, and Alonso-Mora (2020) presents another exciting work. They propose an efficient communication method for learning when and with whom to communicate in multi-robot scenarios. Using Multi-Agent Reinforcement Learning (MARL) to learn a policy offline that minimizes the communication the agents perform collision avoidance.

While we believe that machine learning will play an increasingly important role, further developments in hardware, software, and algorithms are still needed before they catch up with state of the art in autonomous MAVs.

### 1.1.6 Platform Deployment

Regarding the characteristics of the hardware in which the method was implemented, the authors of Florence et al. (2018); Gao, Wu, Gao, and Shen (2019); Mohta et al. (2018); Oleynikova, Taylor, Siegwart, and Nieto (2018); Papachristos et al. (2017); Usenko et al. (2017), and Lin et al. (2018) used high-end Intel i7 processor. On the other hand, the algorithm of Gao et al. (2019) and by Lin et al. (2018) used an additional Tx1 Nvidia GPU.

Cieslewski et al. (2017a) used a quad-core CPU at 1.7 GHz for onboard state estimation but did not provide details on how mapping was implemented. Selin et al. (2019) reported using a motion capture system for tracking MAV pose, and Collins (2019) only presented simulation results showing that their complete framework can run at under 20ms but did not specify the characteristics of the processor used. In contrast, we only used a low-power Intel Atom processor to run everything, including state estimation, flight control, mapping, planning, trajectory generation, and agent coordination.

Oleynikova, Taylor, Siegwart, and Nieto (2018) and Papachristos et al. (2017) replan at 4 Hz and 20 Hz, respectively. Both Gao et al. (2019) and Lin et al. (2018) replan at 10 Hz, limited by the map update rate. Florence et al. (2018) perform local obstacle avoidance at the depth image rate (60 Hz) while the global frequency was limited to 2 Hz. In contrast, we were able to plan and replan trajectories at the camera frame rate (33Hz).

The most common sensor setup for mapping onboard MAVs are stereo cameras, for example, as used by Cieslewski et al. (2017a); Papachristos et al. (2017); Usenko et al. (2017) and Oleynikova, Taylor, Siegwart, and Nieto (2018). We also used a stereo camera in this work. Lidar sensors are the most popular alternative to stereo cameras for dense mapping. They are accurate and perhaps less computationally expensive to run in real-time on MAV hardware but are more expensive, heavier, and bulkier, especially for 3D sensing. Mohta et al. (2018) used a 2D lidar mounted on a nodding gimbal. The platform used by Florence et al. (2018) had a similar sensor, but it was only used for state estimation and to aid obstacle perception.

In terms of state estimation, visual-inertial odometry (VIO) is the general approach, for example, as used by Papachristos et al. (2017) and Oleynikova, Taylor, Siegwart, and Nieto (2018). Mohta et al. (2018) used a stereo camera for VIO but also fused measurements from a downward pointing lidar. Florence et al. (2018) used a 2D lidar plus a downward-facing single-point LIDAR for state estimation, fusing, and matching measurements against a prior-built map. Gao et al. (2018); Lin et al. (2018) used a monocular fish-eye camera and an IMU both for mapping and for state estimation.

| | Map Generation | | Path Planning | | Trajectory Generation | | Computing Resources |
|---|---|---|---|---|---|---|---|
| | **Type** | **Nav Objective** | **Type** | **Nav in FoV** | **Type** | **Feasible Dynamics** | |
| Cieslewski et al. (2017b) | Octomap | Exploration | Frontier Based | Yes | Direct Velocity Control | No | N/A |
| Papachristos et al. (2017) | Octomap | Exploration | NBVP | No | Direct Velocity Control | No | 3.4 GHz i7 dual-core |
| Selin et al. (2019) | Octomap | Exploration | NBVP + Frontier Based | No | Direct Velocity Control | No | N/A |
| Oleynikova, Taylor, Siegwart, and Nieto (2018) | Voxel Hashing TSDF & ESDF | Goal | NBVP + Distance Goal Minimization | Yes | Quadratic Optimization | Position | 2.4 GHz i7 dual-core @ 4Hz |
| Gao et al. (2018) | Regular ESDF Grid Map | Goal | Fast Marching in Distance Field | No | Quadratic Optimization | Position | 3.00 GHz i7 Nvidia TX1 @ 10Hz |
| Florence et al. (2018) | Search over views | Goal | 2D A* | Yes | MPC | Position Yaw | i7 dual-core @ 200 - 250Hz |
| Usenko et al. (2017) | Egocentric Grid | Goal | Offline RRT* | No | Uniform B-Splines | Position | 2.1 GHz i7 dual-core |
| Mohta et al. (2018) | 3D local map and 2D global map | Goal | A* in Hybrid Graph | No | RILS + Path Modification + QP | Position | 3.40 GHz Intel i7-5557U @ 3Hz |
| Lin et al. (2018) | TSDF | Goal | Sampling-Based Informed A* | No | Quadratic Optimization | No | Intel i7-5500U Nvidia TX1 @ 4Hz |
| Collins (2019) | KD Tree + Sliding Voxel Map | Goal | Graph Search | Yes | Motion Primitives | Position Yaw | N/A |
| Ours | Linear Octree | Goal | Informed RRT Connect | Yes | Contouring based | Position Yaw | 1.6 GHz Atom x7-Z8750 @ 33Hz |

Table 1.1: Qualitative comparison of state-of-the-art methods for single agent navigation in unknown environments.

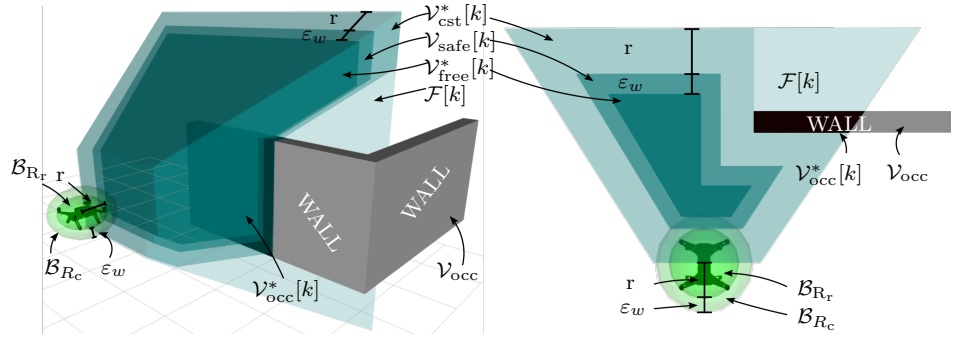



Figure 1.1: Illustration of the different regions used. On the right, a top view of a cross section, parallel to the ground, at the height of the agent.

## 1.2 Problem Definition

The following notation is introduced first to formulate the problem that this work solves. For that, we define the different regions illustrated in Figure 1.1.

Let $\mathcal{B}_{R_\text{c}}(p)$ be a ball in $\mathbb{R}^3$ of radius $\boldsymbol{R_\text{c}} = R_\text{r} + \varepsilon_w$ centered at a point $p$, where $R_\text{r}$ is the radius of the smallest ball containing the robot and $\varepsilon_w$ is a safety distance. Continuous-time is represented by $t$ while $k$ represents discrete-time. The time interval $[t_0, t_1]$ is represented by the subscript $_{[t_0:t_1]}$. The frame rate of the 3D sensor is denoted as $f_\text{cam}$, and the field of view at the $k$-th frame, which is assumed to be a frustum, is denoted by $\mathcal{F}[k]$. The work space space is a bounded 3D cubic volume $\mathcal{V} \subset \mathbb{R}^3$.

The obstacle region is denoted by $\mathcal{V}_\text{occ} \subset \mathcal{V}$ which characterizes the environment. Portions of $\mathcal{V}_\text{occ}$ are sensed to generate a point cloud based on a probabilistic occupancy model that later is added to volume $\mathcal{V}^*_\text{occ}[k]$. Moreover, when a portion of $\mathcal{V}^*_\text{occ}[k-1]$ is sensed to be free at the moment of the $k$-th frame, it is removed from $\mathcal{V}^*_\text{occ}[k]$. Thus, $\mathcal{V}^*_\text{occ}[k]$ represents the volume that is assumed to be occupied at the moment that the $k$-th frame is taken, using an asterisk to distinguish it from the actual occupied volume $\mathcal{V}_\text{occ}$.

We define $\mathcal{V}^*_\text{cst}[k]$ as the volume swept by all the rays originating at the sensor and terminating either at a detected point or at the border of the $\mathcal{F}[k]$. To guarantee safety, we want the robot to always move inside $\mathcal{V}^*_\text{cst}[k]$. Therefore, is important to distinguish between $\mathcal{V}_\text{safe}[k]$ and $\mathcal{V}^*_\text{free}[k]$, namely the regions obtained by erosion of $\mathcal{V}^*_\text{cst}[k]$ with respect to spheres of radius $R_\text{r}$ and $\boldsymbol{R_\text{c}}$, respectively. If we design our trajectory generation algorithm to produce trajectories inside $\mathcal{V}_\text{free}[k]$ we can be sure that they will be collision-free and that the robot will move inside $\mathcal{V}^*_\text{cst}[k]$ provided the maximum tracking error is less than $\varepsilon_w$. In other words, $\mathcal{V}^*_\text{free}[k]$ is the space we need to use for trajectory planning.

A voxel $v$ represents the space contained in a cubic volume, obtained by recursively dividing $\mathcal{V}$ into eight equally sized volumes. The smallest voxel edge length is $r$.

The position, linear velocity, linear acceleration, linear jerk and linear snap are denoted by $p, v, a, j, \zeta \in \mathbb{R}^3$, respectively. The yaw orientation and its derivatives, angular velocity and angular acceleration are denoted by $\psi, \omega, \alpha \in \mathbb{R}$, respectively. A trajectory is defined as the vector $[p(t)^T,\ \psi(t)]^T$ containing a time-dependent sequence of positions and yaw orientations, where $[\bullet]^T$ is the transpose operator.

The augmented state for the trajectory is defined as

$$x(t) = [p(t)^T, \psi(t), v(t)^T, \omega(t), a(t)^T, j(t)^T]^T.$$

The set of feasible states is defined as

$$\mathcal{X}_{\text{safe}}[k] = \left\{ x \ : \ \begin{array}{c} p \in \mathcal{V}_{\text{safe}}[k], \\ \|v\|_\infty \leq V_{\max}, \ \|a\|_\infty \leq A_{\max}, \\ \|j\|_\infty \leq J_{\max}, \ |\omega| \leq \Omega_{\max}, \end{array} \right\}.$$

That is, the set of points inside the space free of collisions such that the dynamical constraints given by a maximum velocity, acceleration, jerk and angular velocity are satisfied.

Consider a set $\mathcal{A} = \{1, \ldots, N\}$ of $N \in \mathbb{N}_{\geq 1}$ agents in the workspace,

$$x^i(t) = [p^i(t)^T, \psi^i(t), v^i(t)^T, \omega^i(t), a^i(t)^T, j^i(t)^T]^T,$$

where $i \in \{1, \ldots, N\}$, denotes the state of the $i$-th agent. We consider that the relative positions and orientations between the agents in $\mathcal{A}$ are known or that they share a common reference frame. Also, we assume a limited communication between the agents, i.e., there exits a communication radius $\boldsymbol{R}_{\text{o}}$ in which the agents broadcast information.

**Problem 1.** *Consider the set $\mathcal{A}$ conformed by $N \in \mathbb{N}_{\geq 1}$ holonomic robots, where each robot is modeled as a ball $\mathcal{B}^i_{R_{\text{c}}}$ with dynamics described by*

$$\dot{p}^i(t) = v^i(t), \quad \dot{v}^i(t) = a^i(t), \quad \dot{a}^i(t) = j^i(t), \quad \dot{j}^i(t) = \zeta^i(t),$$

$$\dot{\psi}^i(t) = \omega^i(t), \quad \dot{\omega}^i(t) = \alpha^i(t).$$

*Given initial and final positions $p^i_s$, $p^i_f$ for each agent such that*

$$p^i_s, p^i_f \in \{p^i \in \mathcal{V} : \mathcal{B}^i_{R_{\text{c}}}(p^i) \cap \mathcal{V}_{\text{occ}} = \emptyset\}$$

*and*

$$\left\|p^{\mathbf{i}}_s - p^{\mathbf{j}}_s\right\| \geq 2\boldsymbol{R}_{\mathbf{c}} \quad \forall \mathbf{i}, \mathbf{j} \in \mathcal{A}, \mathbf{i} \neq \mathbf{j},$$

$$\left\|p^{\mathbf{i}}_f - p^{\mathbf{j}}_f\right\| \geq 2\boldsymbol{R}_{\mathbf{c}} \quad \forall \mathbf{i}, \mathbf{j} \in \mathcal{A}, \mathbf{i} \neq \mathbf{j},$$

*and initial and final yaws $\psi^i_s, \psi^i_f$, respectively, navigate from $x^i(t_s) = [p^{iT}_s, \psi^i_s, 0, \cdots, 0]^T$ to $x^i(t^i_f) = [p^{iT}_f, \psi^i_f, 0, \cdots, 0]^T$ such that*

$$x^i(t) \in \mathcal{X}_{safe}[k] \quad \forall i \in \mathcal{A}, \quad k = \lfloor (t \ - \ t_i) f_{cam} \rfloor,$$

*and*

$$\left\|p^{\mathbf{i}}(t) - p^{\mathbf{j}}(t)\right\| \geq 2\boldsymbol{R}_{\mathbf{c}} \quad \forall \mathbf{i}, \mathbf{j} \in \mathcal{A}, \mathbf{i} \neq \mathbf{j},$$

$\forall t \in [t_s, T_f]$, *with* $T_f = max\left(\left\{t^i_f : i \in \mathcal{A}\right\}\right)$ *a finite-time.*

## 1.3 Contributions

- A computationally efficient method for mapping the environment using lightweight and power-efficient sensors.
- An $O(n)$ time complexity algorithm for collision detection.
- A stochastic technique to generate safe trajectories to a goal.
- A contouring method to obtain the necessary dynamics to track the generated path while fulfilling the physical constraints of the robot.
- An extension of the RRT method that avoids sampling in high-order dimensions or solving complex optimization problems compared with the kinodynamic-RRT.
- A methodology for neighbor coexistence with limited communication that prevents circumnavigation and local minima attraction.
- Exhaustive comparison with state of the art, both for the case of a single agent and for multiple, demonstrating superior performance, especially in challenging scenarios.
- Implementation of the method on a platform less than one kilogram with high computation restrictions and limited sensing capabilites.
- Open-source code available at https://github.com/IntelLabs/autonomousmavs
- An accepted manuscript, JCR Q1: Leobardo Campos-Macías, Rodrigo Aldana-López, Rafael de la Guardia, José I. Parra-Vilchis, David Gómez-Gutiérrez. *Autonomous Navigation of MAVs in Unknown Cluttered Environments*, Journal of Field Robotics, 2020.

## 1.4 Thesis Organization

The essential components that a mobile robot needs to be autonomous are explained in Chapter 2, altogether with the basic algorithms for localization and depth estimation using camera sensors. Also, we include definitions of primitives used for our motion planning algorithm. The blocks that form our framework for single robot navigation in unknown cluttered environments are described in Chapter 3. We first illustrate our mapping method that allows efficient collision detection. Then, we detail our extension of the RRT algorithm, which guarantees a solution that fulfills the dynamic constraints for the robot, if one exists. Finally, we include extensive evaluations of our method against the state-of-the-art.

Then, in Chapter 4, we extend our algorithm to consider the intention of neighboring robots, preventing the circumnavigation to spend time. Instead, the robot with lower priority reduces its speed or waits for the robot with high priority to pass. Finally, we conclude this chapter with a comparison against popular methods of multi-robot trajectory planning.

The results of this work are presented in Chapter 5. We start making different benchmarks using the complete framework. We used as baseline two main scenarios; a forest environment and a maze scenario. The results show a high success rate compared with the chosen methods. At the end of this chapter, we evidence the implementation of the framework in a small platform of less than one kilogram. Finally, Chapter 6 concludes with the learned lessons of this work and the future research lines to continue with our work.

# Chapter 2

# Preliminaries

Given a desired goal position or series of positions and partial or null knowledge about its environment, navigation encompasses the ability of the robot to act based on its knowledge and sensor values to reach its goal positions as efficiently and as reliably as possible. The focus of this chapter is to explain the necessary components to solve this navigation problem. In addition, this chapter also includes the theory required to understand the advances proposed by this work that allowed to extend the state-of-the-art algorithms to mobile platforms with low computation power and limited sensing, considering cooperation between robotic agents.

Basic software engineering concepts embrace software modularity, and the same motivations apply to mobile robot applications. However, in mobile robotics, modularity is even more important because the mobile robot hardware or its physical environmental parameters can change substantially. For example, an ultrasonic sensor could be replaced by a depth camera. Or the robot could navigate in a scenario where the obstacles are dynamic, and the algorithms for obstacle avoidance would have to be changed to handle moving obstacles. Thus, part of the robot's competence should change without causing a chain of side effects that forces to revisit the functioning of other robot competencies. For example, the trajectory generation module should remain without changes independently of the sensors attached to the robot that detect obstacles.

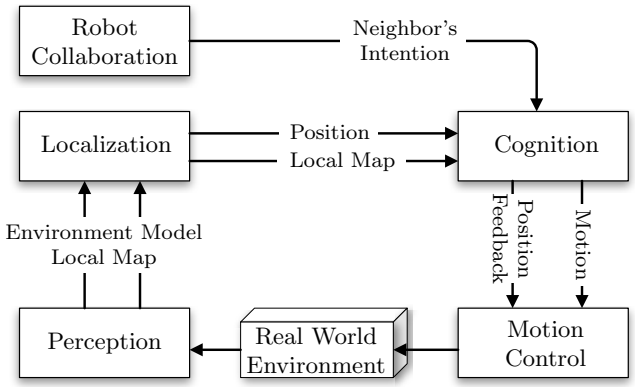



Figure 2.1: Basic architecture for mobile navigation with cooperation among agents.

Moreover, the demands of the algorithms of a mobile robot distinguish between real-time and non-real-time operation. For example, the low-level controller needs a higher execution rate to reject disturbances than the update rate of the map representation that depends upon the sensors' refresh frequency.

For the reasons above, we propose decomposing the navigation problem in the modules shown in Figure 2.1, which is an extension of the architecture suggested by Siegwart, Nourbakhsh, and Scaramuzza (2011), to consider the interaction with different mobile robots.

Section 2.1, Perception, begins with a discussion of available sensors focused on the aerial platforms used in this work, concluding that vision-based sensors are the most suitable for this type of robotic platform. But perception is more than sensing. Perception is also the interpretation of sensed data in meaningful ways. Therefore, Section 2.1 also includes how the images acquired by the vision sensors are used to localize the robot giving the basic framework to perform visual odometry and detect the obstacles in the scenario in point clouds. Finally, the section concludes with a brief explanation of the Octree map representation, which is the foundation of the map used in this work.

Cognition generally represents the purposeful decision-making and execution that a system utilizes to achieve its highest order goals. However, in a mobile robot, the specific aspect of cognition directly linked to full mobility is navigation competence. Thus, Section 2.2 discusses the critical competence required for mobile robot navigation: motion planning. A summary of different motion planning methods is discussed first, concluding that sampling-based algorithms better suit aerial platforms with limited computing power. Finally, the section describes the primitive functions used to propose a motion planning algorithm that cooperates with neighboring agents and avoids static obstacles.

## 2.1 Perception

A critical task of a mobile robot is to acquire knowledge about its environment to be autonomous. Using various sensors and then extracting meaningful information from those measurements, the robot can make sense of its surroundings. This section presents the necessary preliminaries to extract data from the robot's sensors to create a map representation and locate itself in it so that the robot can navigate through it.

This work focuses on using air vehicles for autonomous navigation; for that reason, the weight and energy consumption of the sensors must be taken into consideration since this directly impacts the flight time of a MAV, reducing the possibilities to a limited set of sensors. According to Siegwart et al. (2011), the standard sensors used in robotics can be classified using two important functional axes; proprioceptive or exteroceptive and passive or active. Proprioceptive sensors measure values internal to the system, and exteroceptive sensors acquire information from the robot's environment. Moreover, passive sensors measure ambient environmental energy entering the sensor while active sensors emit energy into the environment, then measure the environmental reaction.

Among the most useful exteroceptive sensors for air vehicle navigation, we can find sensors that obtain the robot's orientation about a fixed reference frame, classified as passive sensors, such as gyroscopes. Ground-based beacons that provide local-

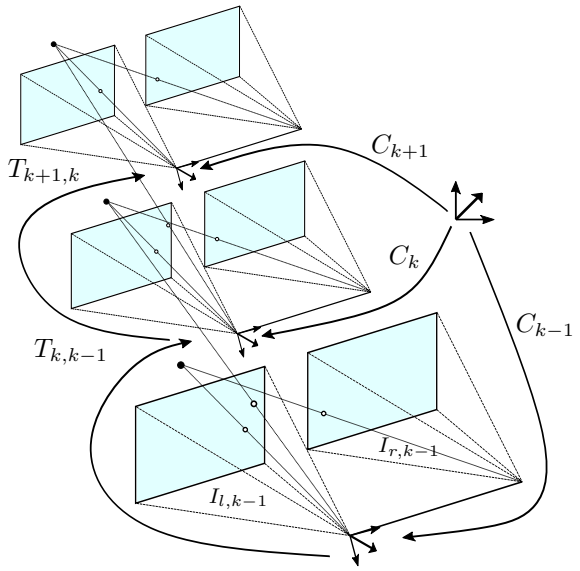



Figure 2.2: Graphic representation of the Visual Odometry (VO) problem. With $I_{l,k-1}$ and $I_{r,k-1}$, the left and right images at time $k-1$, compute the transformation $T_{k,k-1}$ from camera pose $C_{k-1}$ to $C_k$. Then, the concatenation of the transformations $T_{0:K}$ will return the location of camera $C_K$ respecting the initial pose $C_0$.

ization in a fixed reference frame are classified as active sensors, such as GPS and RF beacons. Active ranging sensors that measure reflectivity, time-of-flight, and geometric triangulation are classified as active sensors, such as laser rangefinder. Motion or speed sensors that obtain the speed relative to fixed or moving objects are classified as active sensors, such as radars. And the vision-based sensors that receive visual ranging and perform whole-image analysis are classified as passive sensors, such as CMOS cameras.

Passive sensors are preferred over active ones to save energy and weight; thus, the vision-based sensors become the more suitable air vehicle navigation sensors. Furthermore, specific assumptions about the environment can not be made; therefore, relying on a clean GPS signal for the localization is inadequate since a search and rescue scenario is tackled in this work that is known to be a GPS-denied environment.

As explained in the subsection 2.1.1, vision-based sensors can obtain the necessary information so that the agent can locate itself in the environment and obtain the obstacles position estimation in a point cloud form to create a map representation.

### 2.1.1 Making Sense of Sensors

This subsection first presents a method for robot localization and then a procedure for point cloud obstacle generation based on the processing of frames acquired from the vision-based sensors. The first method, visual odometry (VO), calculates the egomotion of the robot using only the input of multiple cameras attached to it. The second takes two or more images and builds a 3D model of the scene by finding matching pixels in the images and converting their 2D positions to 3D depths.


Figure 2.3: Main components of Visual Odometry algorithm.

According to Scaramuzza and Fraundorfer (2011), the VO problem is described as an agent moving through an environment that takes images with a rigidly attached camera at any time $k$. There are $I_{l,k}$ and $I_{r,k}$, the left and right images, for the case of stereo cameras at every time instant $k$. Also, the camera coordinate frame $C_k$ at time $k$ is assumed to be the agent's coordinate frame. This setup is shown in Figure 2.2

Two camera positions $C_{k-1}$ and $C_k$ at times $k-1$ and $k$ respectively, are related by the homogeneous transformation $T_{k,k-1} \in \mathbb{R}^{4\times4}$:

$$T_{k,k-1} = \begin{bmatrix} R_{k,k-1} & t_{k,k-1} \\ 0 & 1 \end{bmatrix},$$

where $R_{k,k-1} \in SO(3)$ is the rotation matrix and $t_{k,k-1} \in \mathbb{R}^{3\times1}$, then, $T_{0:K} = \{T_{0,1} \cdots T_{K,K-1}\}$ contains the subsequent motions.
The main task in VO is to compute the relative transformations $T_{k,k-1}$ from the left and right images and then concatenate these transformations to recover the full trajectory $C_{0:K}$ of the camera. The typical components used to solve this problem are shown in Figure 2.3. For every image at time $k$, the first step consists of detecting features (block `Feature Detection` in Figure 2.3) independently in both images and then matching them based on some similarity metrics; various techniques can be found in the work of Rosten and Drummond (2006), Lowe (2004), and Bay, Tuytelaars, and Van Gool (2006).

Then, these features are matched (block `Feature Matching` in Figure 2.3) with those from the previous frames, $k-1$, and then they are tracked in the following images using a local search technique, such as Euclidean distance-based correlation as shown by Lowe (2004). The next step consists of computing the relative motion $T_{k,k-1}$ between the time instants $k-1$ and $k$ (block `Motion Estimation` in Figure 2.3), i.e., the camera motion between the current and the previous images. Depending on the type of features, the transformation $T_{k,k-1}$ between a sequence pair of images $I_{l,k-1}$, $I_{r,k-1}$, and $I_{l,k}$, $I_{r,k}$ can be computed from the corresponding two sets of features. Diverse methods are described in detail in the tutorial from Scaramuzza and Fraundorfer (2011). The camera pose $C_k$ is then computed by concatenation of $T_{k,k-1}$ with the previous pose. Finally, an iterative refinement (block `Bundle Adjustment` in Figure 2.3) can be done over the last frames to obtain a more accurate estimate of the local trajectory, as proposed by Triggs, McLauchlan, Hartley, and Fitzgibbon (1999). As reported by Scaramuzza and Fraundorfer (2011), VO provides more accurate trajectory estimates with relative position error ranging from 0.1 to 2% compared with other localization methods.

Building a 3D model of the scene obstacles based on stereo matching is used in this work, given that the VO algorithm does most of the computation required to perform stereo matching. The purpose is to generate a sparse or dense depth map that assigns relative depths to pixels in the input images. Szeliski (2010) provides a method based on the principle that the amount of horizontal motion or disparity is inversely proportional to the distance from the camera. But, automatically measuring this disparity by establishing dense and accurate inter-image correspondences is a challenging task and remains an open computer vision problem Poggi, Tosi, Batsos, Mordohai, and Mattoccia (2020).

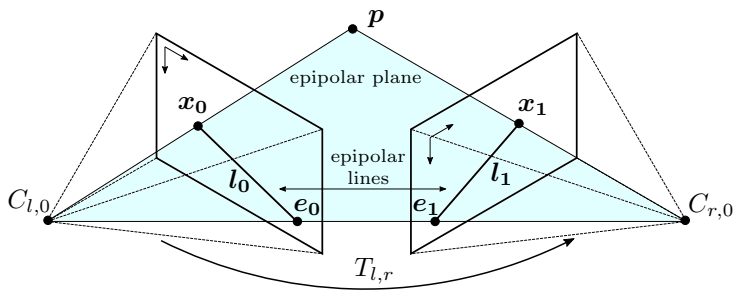



Figure 2.4: A line bounded at one end by the point of interest $p$ and at the other end by the center of the camera $C_{l,0}$ and passes through feature $x_0$. Another line constructed in similar form using feature $x_1$ and camera center $C_{r,0}$ with a line from the center of both cameras builds the epipolar plane. Thus, the epipolar line $l_0$ is bounded by the feature $x_0$ and the epipole $e_0$. Epipolar line $l_1$ is constructed in similarly using epipole $e_1$.

For completeness of this subsection, we briefly describe the method proposed in Szeliski (2010), while any other manner that generates obstacles point clouds is suitable for the framework proposed in the chapter 3, including another kind of sensors, e.g., lidars.

First, the correspondence between the pixels images is computed by using, for example, the methods described by the blocks of feature detection and matching from Figure 2.3, but considering that the transformation $T_{l,r}$, between $I_{l,k}$ and $I_{r,k}$ images is known. $T_{l,r}$ is also called the essential matrix. Then, the triangulation of those features is performed based on the epipolar constraint, which determines the line on which the corresponding features points lie in the other image (Figure 2.4) and is used to generate a standard rectified image, i.e., a projection of both images in one image plane. The entire methodology to obtain the epipolar lines is described in detail in Szeliski (2010). This leads to an inverse relationship between 3D depths $Z$ and disparities $d$

$$d = f\frac{B}{Z}$$

where $f$ is the focal length (measured in pixels), $B$ is the baseline, and

$$p_{l,x} = p_{r,x} + d\left(p_{r,x}, p_{r,y}\right) \quad p_{l,y} = p_{r,y}$$

describes the relationship between corresponding pixel coordinates in the left and right images, in which the disparity map $d\left(p_{r,x}, p_{r,y}\right)$ is obtained. After rectification, the similarity of pixels at corresponding locations $(p_{r,x}, p_{r,y})$ and $(p_{l,x}, p_{l,y})$ $= (p_{r,x} + d, p_{r,y})$ can be compared and be stored in a disparity space image (DSI) $C\left(p_{r,x}, p_{r,y}, d\right)$ that directly generates a 3D point cloud representation for the current scene.

### 2.1.2 Map Representation

Choosing a representation of the environment in which the robot moves entirely depends on the robot's possible positions and the available sensors. An air vehicle to autonomously navigate environments requires a probabilistic representation of free, occupied, and unmapped volumes with efficiency for runtime and memory usage.

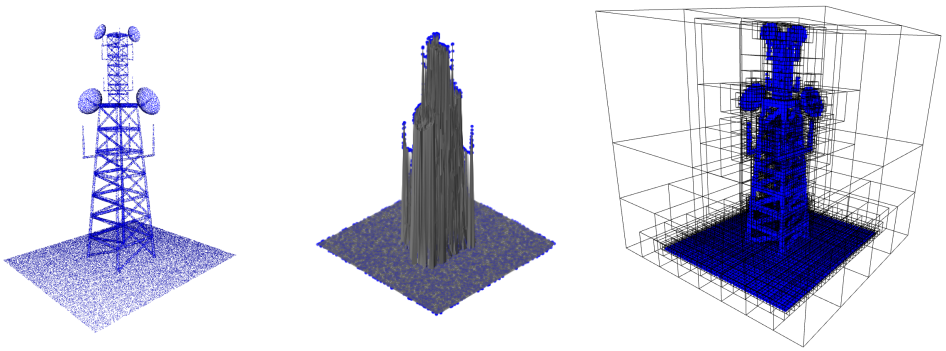


Figure 2.5: A comparison between different 3D representations of a radio tower. At the left a point cloud, in the middle an elevation map, and at the right a volumetric representation.

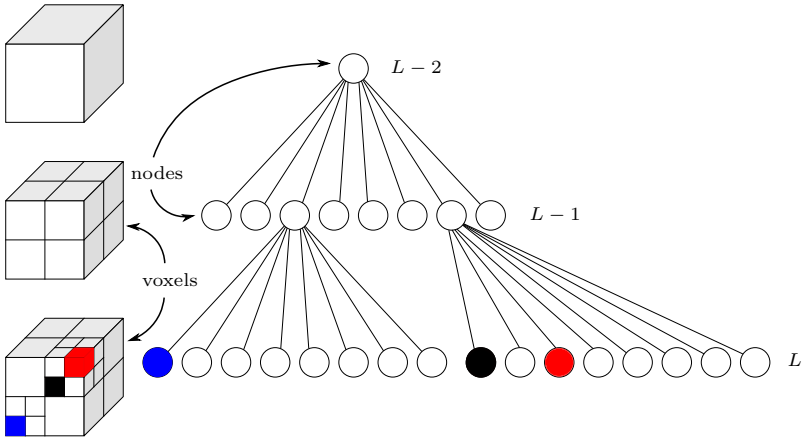



Figure 2.6: Octree representation of 3D volume. At the highest level $L-2$, all the space is enclosed in the root node subdivided into eight children upon maximum depth $L$. Any node (voxel) may have a property representing the value of the volume; the colors red, blue, and black represent occupied, free and unknown space.

Different approaches have been proposed to model the 3D environments (volumes). Figure 2.5 illustrates the most common techniques used in robotics; point clouds, elevation and multi-level surface maps presented by Kweon, Hebert, Krotkov, and Kanade (1989) and by Triebel, Pfaff, and Burgard (2006), and volumetric representation introduced in Hornung et al. (2013). Using the point cloud generated by the sensors directly provides a granular representation of the environment; however, it requires large amounts of memory to store it and, for navigation purposes, contains redundant information. Moreover, this representation lacks the possibility of categorizing the free and unmapped volumes. On the other hand, elevation maps and multi-level surface maps reduce the memory consumption but do not represent unmapped volumes either; besides that, they cannot represent the 3D volume properly. As shown in the middle of Figure 2.5, the spaces between the tubes are marked as occupied, and therefore a flying robot could not fly through disabling essential tasks such as inspection.

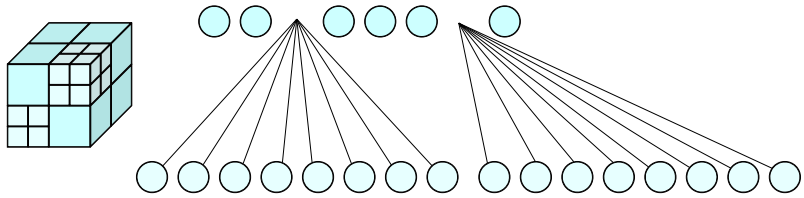


Figure 2.7: Linear Octree: the children are divided on the fly, i.e., when a new property of a voxel must be assigned. Also, only the parent nodes that don't have children (high cyan) are stored in memory alongside with the children generated (low cyan).

A popular approach to modeling environments in 3D is to use a grid of cubic volumes of equal size, called voxels, to discretize the mapped area. This approach uses a tree-based representation that depends on the volume and resolution to improve the memory usage respecting previous methods.

An octree is a tree data structure for spatial subdivision of 3D space, in which each internal node has exactly eight children. Each node in an octree represents the space in a cubic volume, usually called a voxel, as shown in Figure 2.6. Each node is divided into eight children recursively until reaching a maximum depth determining the represented volume's resolution. In addition, the nodes (voxels) can store a feature of the volumetric space represented, such as occupied, free, unknown, or other high-level properties, e.g., velocity, colors.

Octrees require an overhead compared to a fixed-size 3D grid due to the tree structure. However, according to Hornung et al. (2013), a single random query can be performed with a complexity of $\boldsymbol{O}(L) = \boldsymbol{O}(\log n)$ on a tree data structure containing $n$ nodes with a tree depth of $L$.

As stated by Shu and Kankanhalli (1994), a linear octree further decreases memory usage. The main difference compared to the usual octree is that the divisions of the entire volume are not generated a priori; instead, these partitions are generated on the fly, e.g., when some obstacle has been observed, as exemplified by Figure 2.7. Also, parent nodes are no longer kept in memory. In summary, a linear octree is a midpoint between an octree representation and a fixed-size 3D grid.

## 2.2 Motion Planning

According to Karaman and Frazzoli (2011), in simple words, a motion planning algorithm returns a sequence of inputs like control signals to drive an agent from a given starting state to a final given destination while avoiding some predefined obstacles, provided with a description of the robot dynamics as well as a model of the environment. Furthermore, if it terminates in finite time with a valid solution when there is one, it's said to be complete.

The motion planning problem was formally described in the work of Schwartz and Sharir (1983) for the first time, and it's known to be a complex problem to solve from a computational perspective. Typically, the algorithms that solve it are classified depending on its environment modeling, i.e., the representation of the obstacles and the space. We can identify mainly three strategies.

Cell decomposition methods divide the space into connected geometric regions called cells, discriminating between free and occupied objects. The idea is to find a connectivity graph that connects the initial state with the final destination. Thus, any graph search algorithm can be used, such as Depth-first search, Breadth-first search, or A star. The main problem lies with the representation of the obstacles; the better represented they are, the smaller the cells should be, inducing a complex graph with exhaustive computation for search algorithms. On the other hand, bigger cells reduce complexity in the graph, leading to free areas falling into an occupied cell.

The second category is to model the robot's navigation space mathematically. An example of this method is creating a field or gradient across the robot's map that directs the robot to the goal position. The given goal induces an attractive force while the obstacles are modeled as repulsive forces; thus, the robot follows the resultant gradient towards the goal. The main problem with these methods is that the robot can be trapped in a local minimum, in the gradient induced by the location of the obstacles, e.g., in a four-room scenario, the walls will generate a repulsive force that creates a local minimum at the center trapping the robot.

In contrast to previous methods, which need an explicit representation of the obstacles to construct a solution, the roadmap-based algorithms identify a set of routes within the free space. A type of those algorithms is the sampling-based techniques that avoid such obstacle representation. Sampling-based algorithms use a collision checking module instead of explicitly representing the scenario. This module helps to connect a set of sampled points providing feasible trajectories in the obstacle-free space to build a graph (roadmap). Finally, the roadmap constructs a solution from the given initial state to the desired final state.

One of the first sampling-based methods proposed is the Probabilistic Road Maps (PRMs) presented by Kavraki, Svestka, Latombe, and Overmars (1996). The PRMs first construct a graph representing collision-free trajectories and then compute the shortest path connecting the initial state with a final state through that roadmap. Then, Kuffner and LaValle (2000) proposed the Rapidly-exploring Random Trees (RRTs), which is known as the online counterpart of the PRMs since the roadmap is built at the same time that the path is computed. Moreover, since both algorithms are incremental, a maximum number of samples is avoided and they return a solution as soon as the set of trajectories built by the algorithm is rich enough, enabling online implementations. The RRTs algorithms are probabilistically complete (Kuffner and LaValle (2000)), with an exponential rate of decay for the probability of failure (Frazzoli, Dahleh, and Feron (2002)). The RRTs and their variants were the base for developing the algorithms proposed for multi-agent motion planning.

### 2.2.1 Rapidly Random Exploring Trees

The path planning problem is defined in the same fashion as Karaman and Frazzoli (2011) to describe the RRT algorithm adequately.

Let $\chi = (0,1)^d$ be the configuration space, where $d \in \mathbb{N}$, $d \geq 2$. Let $\chi_{\text{obs}}$ be the obstacle region, such that $\chi \setminus \chi_{\text{obs}}$ is an open set, and denote the obstacle-free space as $\chi_{\text{free}} = \text{cl}\,(\chi \setminus \chi_{\text{obs}})$, where $\text{cl}\,(\cdot)$ represents the closure of a set. The initial condition $x_{\text{init}}$ is an element of $\chi_{\text{free}}$, and the goal region $\chi_{\text{goal}}$ is an open subset of $\chi_{\text{free}}$. A path planning problem is defined by a triplet $(\chi_{\text{free}}, x_{\text{init}}, \chi_{\text{goal}})$.

**Definition 2.** *A function $\sigma : [0,1] \to \mathbb{R}^d$ of bounded variation is called a*

- Path*, if it is continuous;*
- Collision – free path*, if it is a path, and $\sigma(\tau) \in \chi_{\text{free}} \forall \tau \in [0,1]$;*
- Feasible path*, if it is a collision-free path, $\sigma(0) = x_{\text{init}}$, and $\sigma(1) \in \text{cl}(\chi_{\text{goal}})$.*

**Problem 3.** *Given a path planning problem $(\chi_{\text{free}}, x_{\text{init}}, \chi_{\text{goal}})$, find a feasible path $\sigma : [0,1] \to \chi_{\text{free}}$ such that $\sigma(0) = x_{\text{init}}$ and $\sigma(1) \in \text{cl}(\chi_{\text{goal}})$, if one exists. If no such path exists, report failure.*

We also introduce the primitive procedures that Karaman and Frazzoli (2011) used to explain the RRT algorithm and its variants.

**Sampling:** Let $\texttt{Sample} : w \mapsto \{\texttt{Sample}_i(w)\}_{i \in \mathbb{N}_0} \subset \chi$ be a map from an integer-valued random variable to sequences of points in $\chi$, such that the random variables $\texttt{Sample}_i$, $i \in \mathbb{N}_0$, are independent and identically distributed (i.i.d.). Moreover, let $\texttt{SampleFree} : w \mapsto \{\texttt{SampleFree}_i(w)\}_{i \in \mathbb{N}_0} \subset \chi_{\text{free}}$ that returns sequences of i.i.d. samples from $\chi_{\text{free}}$. Those samples are assumed to be drawn from a uniform distribution.

**Nearest Neighbor:** Given a graph $G = (V, E)$, where $V \subset \chi$, a point $x \in \chi$, the function $\texttt{Nearest} : (G, x) \mapsto v \in V$ returns the vertex in $V$ that is *closest* to $x$ in terms of a given distance function. For the case of the Euclidean distance

$$\texttt{Nearest}\,(G = (V, E), x) := \text{argmin}_{v \in V} \|x - v\| .$$

**Near Vertices:** Given a graph $G = (V, E)$, where $V \subset \chi$, a point $x \in \chi$, and a positive real number $r \in \mathbb{R}_{>0}$, the function $\texttt{Near} : (G, x, r) \mapsto V' \subseteq V$ returns the vertices in $V$ that are contained in a ball $\mathcal{B}$ of radius $r$ centered at $x$, i.e.,

$$\texttt{Near}\,(G = (V, E), x, r) := \{v \in V : v \in \mathcal{B}_{x,r}\} .$$

**Steering:** Given two points $x, y \in \chi$, the function $\texttt{Steer} : (x, y) \mapsto z$ returns a point $z \in \chi$ such that $z$ is *closer* to $y$ than $x$ is.

**Collision Test:** Given two points $x, x' \in \chi$ , the Boolean function $\texttt{CollisionFree}(x, x')$ returns `True` if the line segment between $x$ and $x'$ lies in $\chi_{\text{free}}$, i.e., $[x, x'] \subset \chi_{\text{free}}$, and `False` otherwise.

**Rapidly-exploring Random Trees (RRT):** The Rapidly-exploring Random Tree algorithm incrementally builds a directed tree of feasible paths rooted at the initial condition. In Algorithm 1, an outline is shown; it is initialized with a graph with the initial state as a single vertex without edges. At each iteration, a point $x_{\text{rand}} \in \chi_{\text{free}}$ is sampled. Then, the nearest vertex $v \in V$ in the tree is tried to connect to the new sample. If the connection succeeds, $x_{\text{new}}$ is added to the vertex set, and $(v, x_{\text{new}})$ is added to the edge set.
In most applications, the quality of the solution returned by a motion planning algorithm is important. For example, one may be interested in solution paths of minimum cost concerning a given cost functional, such as the length of a path or the time required to execute it. Let $\Sigma$ denote the set of all paths, and $\Sigma_{\text{free}}$ the set of all collision-free paths. The optimality problem of path planning asks for finding a feasible path with minimum cost:

**Problem 4.** *Given a path planning problem $(\chi_{\text{free}}, x_{\text{init}}, \chi_{\text{goal}})$ and a cost function $c : \Sigma \to \mathbb{R}_{\geq 0}$, find a feasible path $\sigma^*$ such that $c(\sigma^*) = \min\{c(\sigma) : \sigma \text{ is feasible}\}$. If no such path exists, report failure.*

**Algorithm 1:** RRT

**input :** $V \leftarrow \{x_{\text{init}}\}$, $E \leftarrow \varnothing$
**output:** $G = (V, E)$

**for** $i = 1, \cdots, n$ **do**
  $x_{\text{rand}} \leftarrow \text{SampleFree}_i$
  $x_{\text{nearest}} \leftarrow \text{Nearest}(G = (V, E), x_{\text{rand}})$
  $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{rand}})$
  **if** $\text{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $V \leftarrow V \{x_{\text{new}}\}, E \leftarrow E \cup \{(x_{\text{nearest}}, x_{\text{new}})\}$

This work focused on the asymptotically optimal and computationally efficient version of the RRT proposed by Karaman and Frazzoli (2011). The optimal RRT (RRT*) avoids cycles by removing redundant edges that are not part of the shortest path from the root of the tree (i.e., the initial state) to a vertex. We further introduce the definition of new primitives to explain the complete algorithm.

**Line:** Given two points $x_1, x_2 \in \mathbb{R}^d$, let $\texttt{Line}\,(x_1, x_2) : [0, s] \rightarrow \chi$ denote the straight-line path from $x_1$ to $x_2$, where $s$ is the arc length.

**Parent:** Given a tree $G = (V, E)$, let $\texttt{Line} : V \rightarrow V$ be a function that maps a vertex $v \in V$ to the unique vertex $u \in V$ such that $(u, v) \in E$. By convention, if $v_0 \in V$ is the root vertex of $G$, $\texttt{Line}(v_0) = v_0$.

**Cost:** Let $\texttt{Cost} : V \rightarrow \mathbb{R}_{\geq 0}$ be a function that maps a vertex $v \in V$ to the cost of the unique path from the root of the tree to $v$. It is assumed an additive cost function, so that $\texttt{Cost}(v) = \texttt{Cost}(\texttt{Parent}(v)) + c(\texttt{Line}(\texttt{Parent}(v), v))$. By convention, if $v_0 \in V$ is the root vertex of $G$, then $\texttt{Cost}(v_0) = 0$.

**Optimal Rapidly-exploring Random Trees (RRT*):** Algorithm 2 shows the RRT* method, which adds points to the vertex set $V$ the same way as RRT. It also considers connections from the new vertex $x_{\text{new}}$ to vertices in $X_{\text{near}}$, i.e., other vertices that are within distance $r(\#V) = \min\left\{\gamma_{RRT*}\left(\log(\#V)/\#V\right)^{1/d}, \eta\right\}$ from $x_{\text{new}}$, where $\#(\cdot)$ represents the cardinality of a set, $\eta$ is a constant that depends on the definition of the local steering function, and $\gamma_{RRT*} = 2(1+1/d)^{1/d}(\mu(\chi_{\text{free}})/\zeta_d)^{1/d}$, $d$ is the dimension of the space $\chi$, $\mu(\chi_{\text{free}})$ denotes the Lebesgue measure (i.e., volume) of the obstacle-free space, and $\zeta_d$ is the volume of the unit ball in the d-dimensional Euclidean space.

However, not all possible connections result in new edges being inserted in the edge set $E$. Only the edges that can be connected to $x_{\text{new}}$ from $X_{\text{near}}$ along a path with minimum cost and the edges created from $x_{\text{new}}$ to vertices in $X_{\text{near}}$, if the path has lower cost compared with the path through the current parent. In this case, the edge linking the vertex to its current parent is deleted to maintain the tree structure.

The primitives and the RRT* are used to develop the trajectory planning algorithms for single and multiple agents. The described benefits, such as avoiding the discretization of the scenario, the possibility of escaping local minimums, and the generation of asymptotically optimal paths, make the RRT* an excellent candidate to start with to improve our framework.

**Algorithm 2:** RRT*

**input :** $V \leftarrow \{x_{\text{init}}\}$, $E \leftarrow \varnothing$
**output:** $G = (V, E)$

**for** $i = 1, \cdots, n$ **do**
  $x_{\text{rand}} \leftarrow \text{SampleFree}_i$
  $x_{\text{nearest}} \leftarrow \text{Nearest}(G = (V, E), x_{\text{rand}})$
  $x_{\text{new}} \leftarrow \text{Steer}(x_{\text{nearest}}, x_{\text{rand}})$
  **if** $\text{ObstacleFree}(x_{\text{nearest}}, x_{\text{new}})$ **then**
    $X_{\text{near}} \leftarrow \text{Near}\left(G = (V, E), x_{\text{new}}, \min\left\{\gamma_{RRT*}\left(\log(\#V)/\#V\right)^{1/d}, \eta\right\}\right)$
    $V \leftarrow V \cup \{x_{\text{new}}\}$
    $x_{\min} \leftarrow x_{\text{nearest}}$
    $c_{\min} \leftarrow \text{Cost}(x_{\text{nearest}}) + c\left(\text{Line}(x_{\text{nearest}}, x_{\text{new}})\right)$
    **foreach** $x_{\text{near}} \in X_{\text{near}}$ **do**
      **if** $\text{CollisionFree}(x_{\text{near}}, x_{\text{new}}) \wedge$
      $\text{Cost}(x_{\text{near}}) + c\left(\text{Line}(x_{\text{near}}, x_{\text{new}})\right) < c_{\min}$ **then**
        $x_{\min} \leftarrow x_{\text{near}}$
        $c_{\min} \leftarrow \text{Cost}(x_{\text{near}}) + c\left(\text{Line}(x_{\text{near}}, x_{\text{new}})\right)$
    $E \leftarrow E \cup \{(x_{\min}, x_{\text{new}})\}$
    **foreach** $x_{\text{near}} \in X_{\text{near}}$ **do**
      **if** $\text{CollisionFree}(x_{\text{new}}, x_{\text{near}}) \wedge$
      $\text{Cost}(x_{\text{new}}) + c\left(\text{Line}(x_{\text{new}}, x_{\text{near}})\right) < \text{Cost}(x_{\text{near}})$ **then**
        $x_{\text{parent}} \leftarrow \text{Parent}(x_{\text{near}})$
        $E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$

# Chapter 3

# Reaching a Goal in Unknown Environments

This chapter focuses on the problem of a single agent navigating in an unknown environment towards a given goal while fulfilling the dynamical constraints of the robot. Thus, a framework is proposed divided into two core components: mapping and motion generation.

First, the unknown environment is mapped based on depth images alongside the camera's pose. The state of the map, which is implemented as a linear octree, represents the probability of occupancy of the cells, corresponding to the occupied space $\mathcal{V}^*_{\text{occ}}[k]$. Note that at any time $k$, the map contents inside $\mathcal{V}^*_{\text{occ}}[k]$ may change due to measurement errors at the current frame or sensor drift errors accumulated over time. When a voxel previously marked as occupied is observed to be free, $\mathcal{V}^*_{\text{occ}}[k]$ is updated. For global planning, the space that has not been observed is assumed to be free.

Then, at the motion planning stage, the following action is generated by creating a trajectory inside the collision-free space, using a variation of the RRT algorithm. In our implementation, the root of the tree (current robot's position) is initialized with a set of random points inside $\mathcal{V}^*_{\text{free}}[k]$ obtained from the last sensor scan. In this way, the next position is guaranteed to be contained in $\mathcal{V}^*_{\text{free}}[k]$.

The variation of the RRT algorithm here proposed is based on a method for trajectory generation in such a way that drives the robot from its current state to the next planned action from the previous stage, taking into account the robot's dynamical constraints. Crucially, the yaw's trajectory is generated so that the robot always moves inside $\mathcal{F}[k]$ because the forward-looking camera is aligned with the direction of motion.

## 3.1 Map Representation

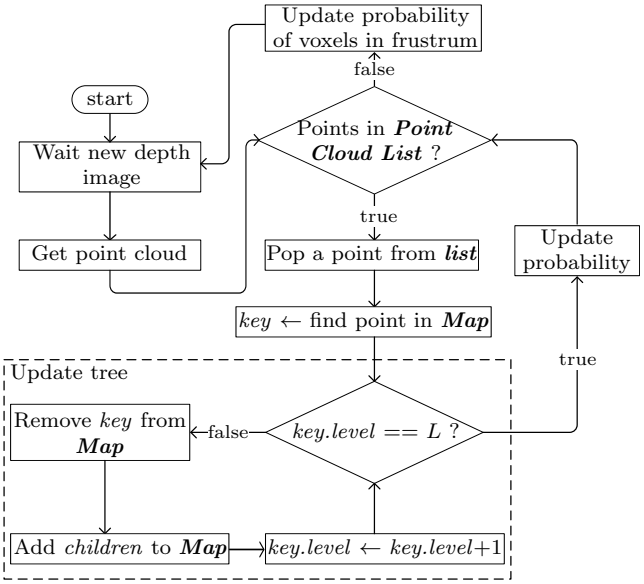



Figure 3.1: Flow diagram of the mapping algorithm.

The map creation is illustrated in Fig. 3.1. The `Map` is implemented as a linear octree with $L$ levels and with axes-aligned bounding boxes. When a new disparity image is received, a point cloud list is created by reprojecting occupied pixels to 3D-space in the world coordinate system (block `Get point cloud` in Figure 3.1). For each point (block `Points in Point Cloud List ?` in Figure 3.1), a key is generated formed by a level code indicating the depth $l$ in the octree, i.e. `key.level` $= l$, and a spatial code computed by

$$\texttt{key.spatial}(x, y, z, l) = \texttt{cat}\left\{\bar{x}, \bar{y}, \bar{z}\right\} \tag{3.1}$$

where

$$\bar{x} = \left\lfloor \frac{x}{r2^{L-l}} \right\rfloor, \quad \bar{y} = \left\lfloor \frac{y}{r2^{L-l}} \right\rfloor, \quad \bar{z} = \left\lfloor \frac{z}{r2^{L-l}} \right\rfloor,$$

$\texttt{cat}\{\bullet, \bullet, \bullet\}$ concatenates its arguments represented in $m$ digits and $\lfloor \bullet \rfloor$ represents the floor operation. To find the key corresponding to the current location of the point in the map, equation (3.1) is used to generate different keys from the root level $l = 0$ up to $l = L$ until the key is found in `Map` (block `find point in Map` in Figure 3.1). As is common in octrees, `Map` insertion is logarithmic in the size of the map, which can be dynamically expanded as needed at practically no cost. For each level $l$, if the obtained key does not correspond to one in $l$, the key is removed from `Map` and the space represented by this voxel is subdivided into eight equal volume subspaces and their centers are used to generate new keys that are added to the `Map` (block `Add children in Map` in Figure 3.1). Then, `key.level` is incremented. This process (block `Update tree` in Figure 3.1) is repeated until `key.level` is equal to $L - 1$. Finally, when `key.level` is equal to $L$ a counter linked to the corresponding voxel is incremented.

### 3.1.1 Probabilistic occupancy model.

In addition to the underlying data structure, another critical aspect is how to update the map's state based on measurement data, taking the sensor characteristics into

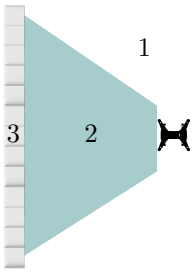



Figure 3.2: An illustration for the laser scanner model provides a probability for the voxel to be occupied given a sensor scan. Voxels at (1) have a probability equal to the prior, at (2) have a low probability and at (3) have a high probability.

account. Stereo cameras are ideal sensors for MAVs because of their lightweight, power efficiency, and good accuracy within their domain of operation. As described in detail by Pohl, Dorodnicov, and Achtelik (2019), depth sensors for MAV navigation can be tuned to provide depth values with high confidence at the cost of losing some valid depth data in some instances. For example, scenes where there are structures with repetitive content, may affect stereo feature matching in neighboring areas. Different methods have been proposed for integrating individual range measurements into an occupancy grid. In this work, we process the point clouds obtained from a depth camera, first by grouping points into voxels similar to the method proposed in Oleynikova, Taylor, Fehr, Siegwart, and Nieto (2017). Then, the goal is to update the individual voxels state, which is either free $c$ or occupied $\bar{c}$, i.e., the posterior $p(c|z_{1:t})$, where $c$ represents the current value of the voxel given the sensor observation $z_t$. As explained by Thrun et al. (2006), this is achieved by making the standard assumption of measurement independence and using a Bayesian filter

$$p(c|z_{1:t}) = \frac{p(z_t|c, z_{1:t-1})p(c|z_{1:t-1})}{p(z_t|z_{1:t-1})}.$$

By making the Markov assumption that $p(z_t|c, z_{1:t-1}) = p(z_t|c)$ and using the Bayes' rule, we get

$$p(c|z_{1:t}) = \frac{p(c|z_t)p(z_t)}{p(c)} \frac{p(c|z_{1:t-1})}{p(z_t|z_{1:t-1})}$$

where $p(c|z_t)$ is known as the *inverse sensor model*. Using the same approach, an update rule for $p(\bar{c}|z_{1:t})$ can be derived and used to divide both update rules

$$\frac{p(c|z_{1:t})}{p(\bar{c}|z_{1:t})} = \frac{p(c|z_t)}{p(c)} \frac{p(\bar{c})}{p(\bar{c}|z_t)} \frac{p(c|z_{1:t-1})}{p(\bar{c}|z_{1:t-1})}.$$

Finally, we take the log of this update rule to find the log odds of the belief the voxel is occupied over the belief it isn't filled

$$\begin{aligned} \log\frac{p(c|z_{1:t})}{p(\bar{c}|z_{1:t})} &= \log\frac{p(c|z_t)}{p(c)} \frac{p(\bar{c})}{p(\bar{c}|z_t)} \frac{p(c|z_{1:t-1})}{p(\bar{c}|z_{1:t-1})} \\ l_t &= \log\frac{p(c|z_t)}{p(\bar{c}|z_t)} + \log\frac{p(\bar{c})}{p(c)} + l_{t-1}. \end{aligned}$$

We need to know the inverse sensor model $p(c|z_t)$ and the prior $p(c)$ for this update rule. The rest are the complements of these two terms. In the work of Thrun et al. (2006), is included a detailed derivation of the rule mentioned above.

The inverse sensor model specifies a distribution over the state variable as a function of the measurement $z_t$. In this work, we used a laser scanner model, as exemplified in Figure 3.2. We define the probability $p(c|z_t)$ as

$$p(c|z_t) = \begin{cases} n\Upsilon(\gamma), & \text{if } n > 0, \\ \frac{1}{2} + \frac{1}{2}\text{Erf}\left(\frac{d\gamma}{\sqrt{2}(\sigma f/b)\gamma^2}\right), & \text{if } d\gamma < 0, \\ P_{\text{free}}, & \text{otherwise.} \end{cases}$$

The first condition is applied to all voxels that were observed in the last scan, where $n$ represents the number of points mapped into each voxel and $\Upsilon(\gamma)$ is a function of the distance $\gamma$ from the camera to the voxel representing the inverse of the maximum number of points that can be mapped into the voxel. The latter is obtained by back-projecting the voxel volume to the camera frame and counting the number of pixels. This can be precomputed and stored as a table by discretizing $\gamma$ (block `Update Probability` in Figure 3.1). The second condition applies to unobserved voxels that are intersected by a ray from the camera to one of the observed voxels. The formula approximates the in-ray probabilities corresponding to the stereo camera model from Schauwecker and Zell (2014), where $d\gamma$ represents the difference between the distance to the voxel and the distance to the end point of the ray and the camera parameters $\sigma$, $b$ and $f$ are the disparity standard deviation, the baseline and the focal length, respectively. The third condition applies to the remaining voxels inside the frustum, with a constant value $P_{\text{free}}$ (block `Update Probability of Voxels in Frustrum` in Figure 3.1).

In comparison, frameworks such as Octomap use ray casting for updating the occupancy probability of all voxels along with the rays. To reduce the amount of computation, the grouped raycasting approach in Oleynikova et al. (2017) applies ray casting only once on the weighted mean of all points within each voxel. This approach can be improved as shown in Kwon, Kim, An, and Yoon (2019) by grouping all the rays sharing the same set of cells into a so-called super ray, which allows updating the map with a single traversal on those cells, and by eliminating unnecessary work by culling out cells having already saturated occupancy probabilities to fully free states. An alternative to ray casting is space carving, where projection and interpolation rather than rasterization are used to carve out space considered likely to be unoccupied Klingensmith, Dryanovski, Srinivasa, and Xiao (2015). Instead of iterating over rays, the idea is to iterate over voxel cells in the visual field of view and, by projecting them into the depth image, to determine if they should be marked as free based on their distance from nearby sensor hits.

Our approach is similar in that we also iterate over voxels in the frustum. However, it's less conservative since we consider the distance and the relative position of free and occupied voxels. This can be done efficiently by using a hash table indexed by the orientation of occupied voxels to store the endpoint of rays. Determining the in-ray probability of a query voxel is reduced to computing its hash and searching the table, which can be done in constant time.

### 3.1.2 Collision testing during planning

Besides point cloud processing, collision detection during planning is the bottleneck in our framework. The essential operation of the RRT-based algorithms is the random sampling of the map space in search of new points that can be joined to an existing node in the search tree via a straight line, without intersecting occupied voxels in the map. Our algorithm for collision detection takes advantage of the octree structure for efficiency, and it is presented in Algorithm 3. It receives the `line`

**Algorithm 3:** Detect Collision

**input :** *key*, *line*
**output:** *collide*

```
1  if key.level == L then
2      v ← GetVoxel(key)
3      if Intersection(line, v) and v ⊆ V*_occ[k] then
4          return true
5      else
6          return false
7  else
8      children ← GetChildren(key)
9      foreach child in children do
10         if DetectCollision(child, line) then
11             return true
12     return false
```

to be tested for collision and the `key` of a voxel containing both ends of the `line`, i.e., the voxel that contains the entire line is recursively found by generating the keys for both points, each time with a lower level until the keys are the same. This algorithm tests recursively for collisions in all levels starting from `key.level` up to $L$.

Even though a flat grid representation of the map, i.e., only saving observed voxels at maximum discretization, would be faster for map creation, our linear octree implementation presents a significant advantage for collision checking between points during path planning. Notice that the linear octree is similar to a sparse grid representation with an additional block `Update tree` (see Figure 3.1). Also, notice that all points inserted into the map must be inside $\mathcal{F}[k]$ and that typically there will be some overlap between $\mathcal{F}[k]$ and $\mathcal{F}[k+1]$ as the robot needs to update its map frequently during navigation. Therefore, since the method `Update tree` is only applied to points in $\mathcal{F}[k+1] \setminus \mathcal{F}[k]$, and the worst case in terms of computation cost would be expected to happen only at the start when the first frame is processed. Nonetheless, in the worst case, that is, when 100% of the points in $\mathcal{F}[k]$ are added to the map, the penalty in computation time to implement our linear octree has been found experimentally to be only a tiny fraction of complete perception pipeline.
Figure 3.3 presents a benchmark comparison between Algorithm 3, which exploits our map representation, and the naive way of collision checking on the same map by testing every voxel between the two points defining a line. Fig. 3.3 shows the average computing time for 10000 lines with lengths between 5 to 400 voxels in different random forest environments. In contrast with the naive method, which has a computation time that grows close to linearly with the size of the line, the computation time remains nearly flat with our map representation, and Algorithm 3 mainly depends on the depth $L$ of the linear octree. Note that the average length between two random points in this map was 165 voxels.

## 3.2 Motion Generation

The proposed motion planning algorithm is described in the flow diagram of Figure 3.4 and illustrated in Figure 3.5. At the beginning, the robot has a starting state $x_{t_c} = x_s$, where the notation $x_{t_c}$ indicates the state $x(t_c)$, a goal position $p_f$ and has no information about the map as illustrated in Figure 3.5a. When a new depth image from the sensor arrives, it is converted to a point cloud and processed for

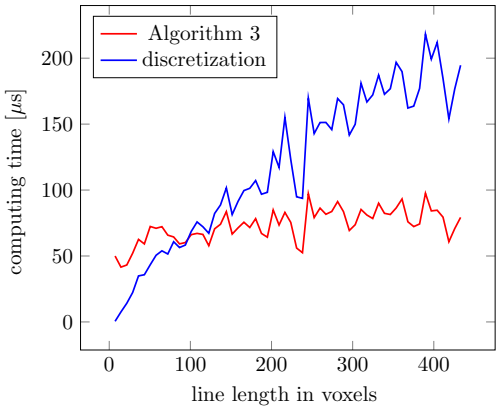



Figure 3.3: Benchmark of collision detection using Algorithm 3 against a discretization-based collision detection

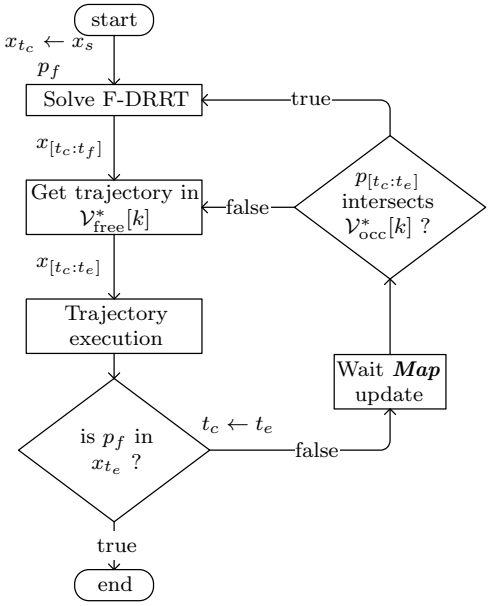



Figure 3.4: Flow diagram of the motion planning algorithm for reaching a goal in unknown environments.

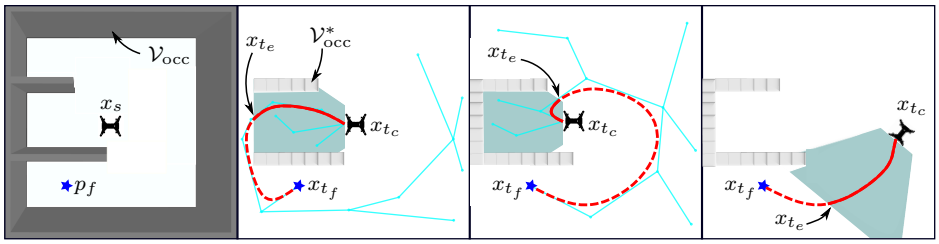



Figure 3.5: Illustration of the motion planning method for reaching a goal in unknown environments: (a) Unknown environment and goal. (b) Initial proposed trajectory using the current map. (c) Replanned trajectory after updating the map. (d) Final trajectory towards the goal.

map storage as described in Section 3.1. The map representation, the current state $x_{t_c}$, and the goal position $p_f$ serve as the input for the trajectory planning algorithm.

In our approach, the robot always has to move in its current field of view, i.e., inside $\mathcal{V}^*_{\text{free}}[k]$, to ensure collision-free navigation; in all cases, this is ensured using our stochastic trajectory planner.

Thus, at the beginning of the algorithm, we proceed to solve the motion planning problem using a variation of the RRT algorithm (block `Solve F-DRRT` in Figure 3.4). The solution to this problem is a sequence of time-indexed states $x_{[t_c:t_f]} = [x_{t_c}, \cdots, x_{t_f}]$ connecting through the navigation space the current state $x_{t_c}$ to the goal $x_{t_f} = [p_f, \cdots, 0]$, as illustrated in Figure 3.5b. Note that the agent is considered to reach the final position at rest, but any desired state can be chosen. We use the notation $x_{[t_c:t_f]}$ to emphasize that the state trajectory $x(t)$ is obtained in the time interval $[t_c, t_f]$.

By design, a subset of states $x_{[t_c:t_e]}$ from $x_{[t_c:t_f]}$ is in $\mathcal{V}^*_{\text{free}}[k]$, with $x_{t_e}$ being the state at time $t_e$ whose position intersects with the region $\mathcal{V}^*_{\text{free}}$, i.e., these states are conformed by the portion of the corresponding positions that are inside the current $\mathcal{V}^*_{\text{free}}[k]$. The states $x_{[t_c:t_e]}$ are extracted from the complete trajectory (block `Get trajectory in` $\mathcal{V}^*_{\text{free}}[k]$ in Figure 3.4) and executed by the robot, i.e., the states $x_{[t_c:t_e]}$ act as the input to the low-level trajectory tracking controller (block `Trajectory execution` in Figure 3.4).

The motion planning algorithm ends when the goal position $p_f$ is included in $x_{[t_c:t_e]}$, this is when $x_{t_e} = [p_f, \cdots, 0]$ (decision block `is` $p_f$ `in` $x_{t_e}$ `?` in Figure 3.4). Otherwise, the final time $t_e$ of the portion of states from the complete trajectory is updated as the current time $t_c \leftarrow t_e$.

Then, suppose that the current plan is still valid because none of the corresponding positions $p_{[t_c:t_e]}$ of the next subset of states $x_{[t_c:t_e]}$ are in $\mathcal{V}^*_{\text{occ}}[k]$ (decision block $p_{[t_c:t_e]}$ `intersects` $\mathcal{V}^*_{\text{occ}}[k]$ in Figure 3.4). In that case, the states $x_{[t_c:t_e]}$ are extracted from the complete trajectory for robot execution, and $t_c$ is updated as before. If, on the other hand, the plan is invalid, a new plan is needed to find a new solution from $x_{t_c}$ to $p_f$ in the navigation space as shown in Figure 3.5c. This process continues, exploring new regions until the goal is reached, as illustrated in Figure 3.5d.

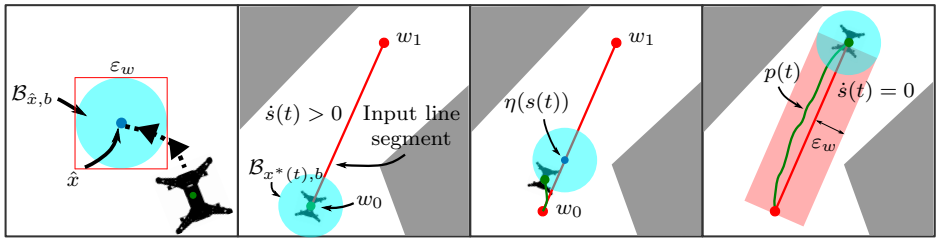



Figure 3.6: Illustration of the trajectory generation method: (a) Representation of the 2D invariant set enclosed by $\varepsilon_w$ boundaries. (b) Initial conditions for the trajectory generation algorithm. (c) Movement of $\eta$ along the line segment. (d) Final trajectory.

The following subsections explain our variation of the RRT algorithm named Fast Dynamic Rapidly Exploring Random Trees (F-DRRT). Instead of generating random samples for the complete state and trying to connect them with the tree by propagating a control input signal, as proposed by Webb and van den Berg (2013), our method first generates random samples only in the position, ensuring that they are admissible, i.e., collision-free. Then, the new line segment created by the connection with the tree is used to generate a time-dependent trajectory considering the dynamic model of the robot using a polynomial control input signal. This significantly reduces the search space compared with typical kinodynamic approaches, leading to a faster solution.

First, in subsection 3.2.1, we introduce a method for contouring a line segment formed by two positions in the space using the dynamic model of the robot for fulfilling its physical constraints. Then, in subsection 3.2.2, we use this result to describe the complete trajectory planning algorithm ensuring that the robot always moves in its field of view.

### 3.2.1 Fulfilling Dynamic Constraints

The proposed trajectory generation algorithm described in this subsection takes as input a pair of positions $w_0$ and $w_1$, and the admissible initial yaw $\psi_0$, velocities, accelerations and jerks. The output is a reference signal for the trajectory tracking controller, which is guaranteed to be dynamically feasible and ensures that the MAV remains within the space that is considered safe for navigation. The pair of positions are used as the start and end of an input line segment. To ensure that the robot will always move in its field of view, the final yaw $\psi_1$ is taken as

$$\cos\psi_1 = \frac{\boldsymbol{w}\cdot\boldsymbol{w_x}}{\|\boldsymbol{w}\|\times\|\boldsymbol{w_x}\|},$$

with $\boldsymbol{w} = w_1 - w_0$ and $\boldsymbol{w_x}$ the unitary vector along $x$ axis.

This algorithm aims to traverse the line segment while guaranteeing that the dynamic constraints of the MAV are satisfied and maintaining an a priori defined maximum separation, denoted by $\varepsilon_w$, from the line, including also a maximum deviation $\varepsilon_\psi$ from the desired orientation $\psi_1$ at any position in the trajectory. The former constraint guarantees that the position $p(t)$ remains in $\mathcal{V}_{\text{safe}}[k]$, with $k = \lfloor (t - t_i) f_{\text{cam}} \rfloor$. The latter ensures that during navigation, the next point $w_1$ will be inside the robot's field of view from the last point $w_0$ when $w_1$ is reached.

Our method consists of two parts. The first part is a virtual control for the robot's motion that induces an exponentially stable equilibrium state $\hat{x}$. The second part is a strategy to move the equilibrium state along the input line segment such that the state $x(t)$ always satisfies the constraints $\varepsilon_w$ and $\varepsilon_\psi$ and the restrictions in velocity and acceleration and jerk.

Figure 3.6 illustrates how the algorithm works in a simplified 2D case. Figure 3.6a shows the set of admissible states of the robot around a desired set point $\hat{x}$, which are given by

$$\mathcal{B}_{\hat{x},b} = \left\{ x \ : \ \begin{array}{l} \|p|_\infty \leq \varepsilon_w, \ |\psi| \leq \varepsilon_\psi \\ \|v\|_\infty \leq V_{\max}, \ \|a\|_\infty \leq A_{\max}, \\ \|j\|_\infty \leq J_{\max}, \ |\omega| \leq \Omega_{\max}, \end{array} \right\},$$

where

$$b = \min\{\varepsilon_w, \varepsilon_\psi, V_{\max}, \Omega_{\max}, A_{\max}, J_{\max}\}. \tag{3.2}$$

Note that in reality this is a 14-dimensional region (4 dimensions for each of the 3 spatial coordinates and 2 dimensions for yaw). However for illustrative purposes $\mathcal{B}_{\hat{x},b}$ is depicted as a 2-dimensional ball in Figure 3.6.

In Figure 3.6b, the red line is the input line segment, and the green point is the position of the robot at the current time. Initially, the equilibrium point $x^*$ is selected as $x^* = [\eta^T, \psi_0, 0, \ldots, 0]^T$, where $\eta$ is a point along the input line segment, with initial condition $\eta = w_0$ and $\psi_0$ the initial orientation of the robot. We parametrize $\eta$ with a continuous, positive, and non-decreasing function $s(t)$ to control the progress of the equilibrium state along the line. Figure 3.6c shows in green the trajectory traced by the robot as it follows $\eta(s(t))$. The condition $\dot{s}(t) > 0$ implies that the equilibrium is moving forward along the line, the dynamic of $s$ is chosen such that the trajectory $x(t)$ remains in $\mathcal{B}_{x^*(t),b}$.

The trajectory of the robot depicted by the green line will remain inside the pink region $\forall t \geq 0$, and will eventually reach the final position corresponding to the end of the line segment $w_1$ as shown in Figure 3.6d. The next theorem demonstrates a correct selection of $\dot{s}$ based on a exponentially stabilizing control law for maintaining $x(t) \in \mathcal{B}_{x^*(t),b}$.

**Theorem 5.** *Let $\dot{x} = f(x, u)$ be the state equation of the robot, with the state $x(t) = [p(t)^T, \psi(t), v(t)^T, \omega(t), a(t)^T, j(t)^T]^T$, and $u \in \mathbb{R}^m$ the control input, and suppose that $u = \upsilon(z)$ is a control law, such that $z = 0$ is a globally exponentially stable equilibrium of the system*

$$\dot{z} = f(z + \hat{x}, \upsilon(z))$$

*with a Lyapunov function $V(z)$ satisfying:*

$$c_1 \|z\|^2 \leq V(z) \leq c_2 \|z\|^2 \tag{3.3}$$

$$\frac{\partial V}{\partial z} f(z + \hat{x}, \upsilon(z)) \leq -c_3 \|z\|^2 \tag{3.4}$$

$$\left\| \frac{\partial V}{\partial z} \right\| \leq c_4 \|z\|, \tag{3.5}$$

*with known constants $c_1$, $c_2$, $c_3$ and $c_4$, and $\hat{x} = [\hat{p}^T, \hat{\psi}, \hat{v}^T, \hat{\omega}, \hat{a}^T, \hat{j}^T]^T$ a set point with $\hat{\omega} = 0$, and $\hat{v} = \hat{a} = \hat{j} = 0$. Moreover, let $\eta(s(t)) = s(t)w_1 + (1 - s(t))w_0$, represent the parametrization of the line segment connecting points $w_0$ and $w_1$,*

$\eta_\psi(s(t)) = s(t)\psi_1 + (1-s(t))\psi_0$, *with* $s \in [0,1]$. *If* $s(0) = 0$ *and* $x(0) \in \mathcal{B}_{x^*(t),b}$, *then with the control law* $u = \upsilon(x - x^*(t))$, *where*

$$x^*(t) = [\eta(s(t)), \eta_\psi(s(t)), 0, \ldots, 0]^T$$

*and with the speed at which the line is progressed given by*

$$\dot{s} = \begin{cases} \theta \frac{\nu_{\max}}{\|\boldsymbol{\eta}\|} & \text{if} \quad s \in [0,1) \\ 0 & \quad s = 1. \end{cases}$$

*where* $0 < \theta < 1$, $\boldsymbol{\eta} = [w_1 - w_0, \psi_1 - \psi_0, 0, \ldots, 0]^T$ *and* $\nu_{\max} = b\frac{c_3}{c_4}\sqrt{\frac{c_1}{c_2}}$, *where* $b$ *is given by* (3.2), *then* $x(t)$ *will remain inside in a ball* $\mathcal{B}_{x^*(t),b} \forall t \geq 0$, *exponentially reaching* $[w_1, \psi_1, 0, \ldots, 0]^T$.

*Proof.* Let $e(t) = x(t) - x^*(t)$, then the dynamics of the closed loop system under the proposed control is given by the perturbed system:

$$\dot{e} = f(e + x^*(t), \upsilon(e)) - \dot{x}^*(t). \tag{3.6}$$

The proof follows by showing, similar as in (Khalil, 2002, Lemma 9.2), that the perturbed system is ultimately bounded with ultimately bound $b$. Thus, if $\|e(0)\| \leq b$, then for all $t$, $\|e(t)\| \leq b$, i.e. $e(t) \in \mathcal{B}_{0,b}$, for all $t$. To this end, we first consider the case when $s \in [0,1)$. Notice that

$$\begin{aligned} \dot{x}^*(t) &= \dot{s}\frac{\partial}{\partial s}[\eta(s(t)), \eta_\psi(s(t)), 0, \ldots, 0]^T \\ &= \dot{s}[w_1 - w_0, \psi_1 - \psi_0, 0, \ldots, 0]^T \\ &= \dot{s}\boldsymbol{\eta} \end{aligned}$$

with $\|\dot{x}^*(t)\|$ satisfying

$$\|\dot{x}^*(t)\| =\leq \theta\nu_{\max} \tag{3.7}$$

Hence taking the derivative of $V(e)$ along the trajectories of the closed loop system (3.6) yields

$$\dot{V}(e) = \frac{\partial V}{\partial e} f(e + x^*(t), \upsilon(e)) - \frac{\partial V}{\partial e}\dot{x}^*(t)$$

It follows by inequality (3.4) that

$$\begin{aligned} \dot{V}(e) &\leq -c_3 \|e\|^2 + \left\| \frac{\partial V}{\partial e}\dot{x}^*(t) \right\| \\ &\leq -c_3 \|e\|^2 + \left\| \frac{\partial V}{\partial e} \right\| \|\dot{x}^*(t)\| \end{aligned}$$

Thus, by the inequalities (3.5) and (3.7)

$$\begin{aligned} \dot{V}(e) &\leq -c_3 \|e\|^2 + c_4\theta\nu_{\max} \|e\| \\ &= -(1-\theta)c_3 \|e\|^2 - \theta c_3 \|e\|^2 + c_4\theta\nu_{\max} \|e\| \\ &\leq -(1-\theta)c_3 \|e\|^2, \quad \forall \|e\| \geq \mu := \nu_{\max}\frac{c_4}{c_3}. \end{aligned} \tag{3.8}$$

Consider the set $\Omega_\mu(e) = \{e : V(e) \leq c_2\mu^2\}$, it follows from the left inequality of (3.3) that

$$c_1\|e\|^2 \leq V(e) \leq c_2\mu^2, \quad \text{if} \quad \|e(t)\| \leq \mu\sqrt{\frac{c_2}{c_1}} = \nu_{\max}\frac{c_4}{c_3}\sqrt{\frac{c_2}{c_1}} = b$$

Thus, $\mathcal{B}_{0,\mu} \subseteq \Omega_\mu(e)$. Since, according to (3.8), $\dot{V}(e)$ is negative outside $\Omega_\mu(e)$, then $\Omega_\mu(e)$ is a positivily invariant set. Therefore, if $e(0) \in \mathcal{B}_{0,b}$, $e(t)$ remains in $\mathcal{B}_{0,b}$ for all $t$, which implies that $x(t)$ remains in $\mathcal{B}_{x^*(t),b}$. Since $\dot{s}(t)) > 0$, $s(t)$ is an increasing function. Thus, $s = 1$ and $\dot{s} = 0$ after a finite time, and

$$x^*(t) = \hat{x} = [w_1, \psi_1, 0, \ldots, 0]^T,$$

with $\dot{x}^*(t) = 0$. Thus, the error dynamics (3.6) becomes

$$\dot{e} = f(e + x^*(t), \upsilon(e)) \tag{3.9}$$

which by assumption its origin is exponentially stable, which completes the proof. □

According to Mellinger and Kumar (2011), the model of the MAV is differentially flat; thus, it can be treated as a free particle, which implies that the dynamics for the position and the yaw can be considered decoupled from each other. The yaw angle is modeled using a second-order chain of integrators for the design and analysis in this section. To each component of the position, we associate a fourth-order chain of integrators. The output of the chain of integrators is given as a reference trajectory to be tracked by a trajectory tracking control.

In this context the MAV model treated in this analysis, $\dot{x}(t) = f(x(t), u(t))$ is considered linear time-invariant, composed of decoupled single-input single-output systems, were the state vector elements are connected to each other through a chained integration procedure, i.e., the state variables are the integral of the subsequent state variables, while the last state variable is equal to the integral of the control input, known as a Brunovsky Canonical Form. Consequently, the dynamic of the motion of the MAV is given by $\dot{x} = Ax + Bu$. Hence, a controller $u(t) = \upsilon(z) = -Kz$ such that $A - BK$ is Hurwitz, is such that

$$\begin{aligned} \dot{z} &= Ax + Bu \\ &= (A - BK)z + A\hat{x}. \end{aligned} \tag{3.10}$$

Notte that by the Brunovsky Canonical form of $A$ and the definition of $\hat{x}$, $A\hat{x} = 0$. Moreover, by(Khalil, 2002, Theorem 4.5) the origin of (3.10) is globally asymptotically stable. Also, by (Khalil, 2002, Theorem 4.6), for any given positive definite symmetric matrix $Q$ there exist a positive symmetric matrix $P$ satisfying

$$P(A - BK) + (A - BK)^T P = -Q.$$

Therefore, the controller $u(t) = \upsilon(z) = -Kz$ and the Lyapunov function $V(z) = z^T P z$ satisfies the conditions of Theorem 5, where equations (3.3)–(3.5) become

$$\begin{aligned} \lambda_{\min}(P) \left\|e\right\|_2^2 \leq V(e) \leq \lambda_{\max}(P) \left\|e\right\|_2^2 \\ \frac{\partial V}{\partial e} Ae = -e^T Qe \leq -\lambda_{\min}(Q) \left\|e\right\|_2^2 \\ \left\| \frac{\partial V}{\partial e} \right\|_2 = \left\|2e^T P\right\|_2 \leq 2 \left\|P\right\|_2 \left\|e\right\|_2 = 2\lambda_{\max}(P) \left\|e\right\|_2, \end{aligned}$$

Theorem 5 is generalized for any dynamical system that satisfies the established conditions; thus, any robot can be selected to generate its states while outlining a given segment of the line and fulfilling its physical constraints.

The algorithm described is similar to the contouring control method given by Liniger, Domahidi, and Morari (2015), in that a path is traversed by continuously making a trade-off between separation from the path and how fast the path is traversed. However, unlike existing contouring methods, our approach doesn't require solving an optimization problem online. It is also similar to LQR-Trees proposed by Tedrake, Manchester, Tobenkin, and Roberts (2010), in that Lyapunov analysis, is used to induce "funnels" along the path. However, our method is significantly more straightforward since, according to Tedrake et al. (2010), LQR-Trees are computationally equivalent to Kinodynamic planning in $\mathbb{R}^{14}$. Our method, on the other hand, uses an offline stage of LQR virtual control design and Lyapunov analysis to compute the region $\mathcal{B}_{\hat{x},b}$, while online it only needs to numerically integrate the differential equations for position, yaw and $s(t)$. This exploits the availability of a collision-free path in $\mathbb{R}^3$, which can be generated efficiently using fast exploration algorithms as described in the following subsection.

### 3.2.2 F-DRRT: Fast Dynamic Rapidly Exploring Random Trees

**Algorithm 4:** F-DRRT

**input :** $V \leftarrow \{x_{tc}\}$, $E \leftarrow \varnothing$
**output:** $G = (V, E)$

1 **for** $i = 1, \cdots, n$ **do**
2 $\quad p_{\text{rand}} \leftarrow \text{SampleFree}_i$
3 $\quad x_{\text{nearest}} \leftarrow \text{Nearest}(G = (V, E), p_{\text{rand}})$
4 $\quad$ **if** $\text{CollisionFree}(p_{\text{nearest}}, p_{\text{new}})$ **then**
5 $\quad\quad x_{\text{new}} \leftarrow \text{Propagate}(x_{\text{nearest}}, p_{\text{rand}})$
6 $\quad\quad X_{\text{near}} \leftarrow \text{Near}\left(G = (V, E), x_{\text{new}}, \min\left\{\gamma_{RRT*}\left(\log(\#V)/\#V\right)^{1/d}\right\}\right)$
7 $\quad\quad V \leftarrow V \cup \{x_{\text{new}}\}$
8 $\quad\quad x_{\min} \leftarrow x_{\text{nearest}}$
9 $\quad\quad c_{\min} \leftarrow \text{Cost}(x_{\text{nearest}}) + c\left(\text{Propagate}(x_{\text{nearest}}, p_{\text{new}})\right)$
10 $\quad\quad$ **foreach** $x_{\text{near}} \in X_{\text{near}}$ **do**
11 $\quad\quad\quad$ **if** $\text{CollisionFree}(p_{\text{near}}, p_{\text{new}}) \wedge$
12 $\quad\quad\quad \text{Cost}(x_{\text{near}}) + c\left(\text{Propagate}(x_{\text{near}}, p_{\text{new}})\right) < c_{\min}$ **then**
13 $\quad\quad\quad\quad x_{\min} \leftarrow x_{\text{near}}$
14 $\quad\quad\quad\quad c_{\min} \leftarrow \text{Cost}(x_{\text{near}}) + c\left(\text{Steer}(x_{\text{near}}, x_{\text{new}})\right)$
15 $\quad\quad E \leftarrow E \cup \{(x_{\min}, x_{\text{new}})\}$
16 $\quad\quad$ **foreach** $x_{\text{near}} \in X_{\text{near}}$ **do**
17 $\quad\quad\quad$ **if** $\text{CollisionFree}(p_{\text{new}}, p_{\text{near}}) \wedge$
18 $\quad\quad\quad \text{Cost}(x_{\text{new}}) + c\left(\text{Propagate}(x_{\text{new}}, p_{\text{near}})\right) < \text{Cost}(x_{\text{near}})$ **then**
19 $\quad\quad\quad\quad x_{\text{parent}} \leftarrow \text{Parent}(x_{\text{near}})$
20 $\quad\quad\quad\quad E \leftarrow (E \setminus \{(x_{\text{parent}}, x_{\text{near}})\}) \cup \{(x_{\text{new}}, x_{\text{near}})\}$

The pseudo-code of the implementation of the F-DRRT is shown in Algorithm 4. The secret sauce is that we divide the procedure into two parts; first, an admissible vertex is generated in the Euclidean space, then the states are propagated through the generated vertex, using the method explained in subsection 3.2.1.

Compared with the original version of the algorithm proposed by Karaman and Frazzoli (2011), the main differences are that first; we sample a point in the Euclidean space instead of sampling the entire states (line 2). Then, we find the nearest edge in the tree, returning the final condition of the states in that edge (line 3). In

our implementation, we use the Euclidean distance as a metric to select the closest edges. Still, any other metric can be used, such as functions depending on time or arrival angle. Next, the sampled point and the edge coordinates are used to review collisions respecting the current map $\mathcal{V}^*_{\text{occ}}$ (line 4). If it results obstacle-free, the states are propagated from the last condition in the closest edge upon the sampled point using the method described in subsection 3.2.1 (line 5). Also, the cost from the nearest edge to the sampled state is computed using the Euclidean distance (line 9); however, any other cost function can be used, e.g., minimize the time or maximize the velocity. Finally, the edges that lie within a radius are selected to be reconnected if a lower cost is found.

To ensure that the robot movement always occurs within its field of view, edges are connected to the root edge $x_{tc}$ only if the angle between the x-y plane and the sampled position respecting $x_{tc}$ is less than the camera's vertical field of view angle. This condition prevents that the MAV rises vertically. Note that no such requirement is necessary for the horizontal angle since if an edge is connected outside the field of view, the MAV will rotate on its yaw and then move towards the sampled point in the direction of movement.

As stated by diagram 3.4, the portion of states $x_{[t_c:t_e]}$ is selected to be executed, which in this case is to remain in the same position and only rotate. Once this command is completed, and the map is updated, the states' next portion is validated.

Given the characteristics of our motion method, it is natural to compare against kinodynamic planning algorithms since these algorithms provide an optimal solution while the dynamic constraints are satisfied. However, although these algorithms provide an optimal solution respecting a cost function, they are time-consuming, so it is essential to demonstrate the trade-off between the computing time for generating a trajectory and its optimality.

Since this thesis focuses on navigation between multiple agents in an unknown environment, our main interest is to generate a motion plan as fast as possible to avoid collision among other agents or static obstacles. Thus, we compare the quality and success rate of our generated trajectories depending on different timeouts.

We further compare our method against three techniques: the kinodynamic algorithm proposed by Webb and van den Berg (2013), the decoupled incremental sequential convex programming procedure proposed by Chen et al. (2015), and the hybrid method based on sampling path planing and quadratic programming (QP) suggested by Campos-Macías, Gómez-Gutiérrez, Aldana-López, de la Guardia, and Parra-Vilchis (2017).

Webb and van den Berg (2013) proposes an incremental sampling-based approach for asymptotically optimal motion planning for robots with linear differential constraints. They use a fixed-final-state-free-final-time controller for connecting a pair of states with a cost function that trades between the duration of a trajectory and the expended control effort. Chen et al. (2015) presents a path planning algorithm based on sequential convex programming (SCP) that finds locally optimal trajectories. Their method first generates a trajectory that connects the starting and final positions by solving a quadratic problem; then, it checks if the trajectory is collision-free at each instant of time. If not, it adds a new restriction at that sampling time and solves the convex problem again. This process iterates until a feasible solution is found or until a maximum number of iterations is reached. Finally, Campos-Macías et al. (2017) introduces a fusion of sampling-based techniques and model-based op-

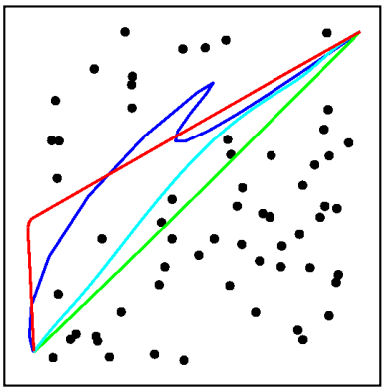


Figure 3.7: Top view of the simulated forest environment, together with the traversed paths. Blue- Webb and van den Berg (2013); Green- Chen et al. (2015); Cyan- Campos-Macías et al. (2017); and Red-Ours.

timization via quadratic programming. The former is used to efficiently generate an obstacle-free path, while the latter considers the robot dynamical constraints to generate a time-dependent trajectory.

The proposed algorithm was implemented using the MatLab language since the algorithms chosen were encoded in that same language. The simulated experiments were performed in several Poisson forest scenarios, generated using the method proposed by Karaman and Frazzoli (2012), in a space of 20m $\times$ 20m $\times$ 20m. The trees, numbering between 10 and 100, were limited to a radius of thirty centimeters and a constant height of 20m. More than 150 simulated forests were created for each tree density and then applied to each of the algorithms with timeouts ranging from 1 to 60 seconds. The start and goal positions of the robot were chosen in opposite corners to maximize the distance traveled. Figure 3.7 shows this scenario with 60 trees and the generated trajectories for all the algorithms.

In the benchmark, we analyzed the overall performance of the generated trajectories in each algorithm: success rate, total trajectory length, and average velocity. Also, we include the average computation time for each algorithm to obtain the first trajectory since some methods iterate for the optimal solution. In addition, to avoid inducing a bias in the benchmark, only the scenarios that the four algorithms solved were selected to generate the statistics of computation time, path length, and average velocity; nonetheless, every trial counted to obtain the success rate.

We use the same parameters as reported by Webb and van den Berg (2013), with boundaries from -5 to 5 for the states and the control input. Still, to obtain a success rate as high as possible, we decrease the order of the system using the jerk as a control input, and only the translational states were considered. For the method of Chen et al. (2015), some parameters had to be tuned to obtain a high success rate, starting from the configuration reported. Therefore, we selected a final trajectory time of 7 seconds, a time step duration of 0.04 seconds, and an acceleration boundary of $5m/s^2$. In the algorithm of Campos-Macías et al. (2017), a set of possible parameters were reported; we selected the best balance between computational time and success rate, with an extension parameter of 0.1 meters and a maximum acceleration of $5m/s^2$. Finally, for our method, we only used the translational part, selected a $\dot{s} = \frac{10}{\|\boldsymbol{\eta}\|}$, and also used the jerk as control input with

| Time Out [s] | Trees | Success Rate [%] | | | | Avg. Comp. Time [s] | | | | Avg. Path Lenght [m] | | | | Avg. Max Velocity [m/s] | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 10 | 60 | 0 | 39 | **94** | - | - | - | - | - | - | - | - | - | - | - | - |
| | 30 | 30 | 0 | 16 | **85** | - | - | - | - | - | - | - | - | - | - | - | - |
| | 60 | 11 | 0 | 4 | **71** | - | - | - | - | - | - | - | - | - | - | - | - |
| | 100 | 0 | 0 | 0 | **42** | - | - | - | - | - | - | - | - | - | - | - | - |
| 10 | 10 | 97 | 98 | 42 | **100** | 7.35 | 1.47 | 4.88 | **0.07** | 32.91 | 33.48 | **31.42** | 33.82 | **4.69** | 0.34 | 1.90 | 0.50 |
| | 30 | 80 | 90 | 21 | **93** | 8.43 | 1.86 | 5.87 | **0.16** | **32.77** | 33.89 | 32.85 | 34.38 | **4.67** | 0.33 | 1.86 | 0.48 |
| | 60 | 41 | 74 | 4 | **74** | 9.36 | 5.80 | 7.67 | **0.80** | 32.72 | 34.61 | **28.12** | 33.88 | **4.66** | 0.32 | 1.71 | 0.48 |
| | 100 | 14 | 24 | 0 | **66** | - | - | - | - | - | - | - | - | - | - | - | - |
| 20 | 10 | 98 | 99 | 55 | **100** | 12.31 | 1.49 | 6.67 | **0.07** | **32.87** | 33.36 | 33.49 | 34.72 | **4.68** | 0.34 | 1.88 | 0.47 |
| | 30 | 90 | 91 | 24 | **95** | 15.12 | 4.16 | 8.01 | **0.15** | **32.80** | 34.01 | 33.71 | 35.26 | **4.68** | 0.33 | 1.83 | 0.46 |
| | 60 | 66 | 72 | 8 | **78** | 17.70 | 5.40 | 9.65 | **0.85** | 32.82 | 35.69 | **30.20** | 36.92 | **4.68** | 0.32 | 1.77 | 0.46 |
| | 100 | 20 | 66 | 0 | **69** | - | - | - | - | - | - | - | - | - | - | - | - |
| 40 | 10 | 97 | **100** | 55 | **100** | 14.12 | 1.50 | 11.36 | **0.09** | **32.97** | 33.49 | 33.55 | 34.67 | **4.70** | 0.34 | 1.87 | 0.47 |
| | 30 | 96 | **98** | 36 | **98** | 16.37 | 6.94 | 11.81 | **0.29** | 32.97 | 33.97 | **32.80** | 36.18 | **4.70** | 0.33 | 1.82 | 0.45 |
| | 60 | 76 | 83 | 9 | **84** | 21.72 | 10.47 | 15.69 | **0.91** | **32.85** | 35.29 | 35.35 | 39.06 | **4.68** | 0.34 | 1.60 | 0.41 |
| | 100 | 35 | **71** | 1 | 70 | 27.09 | 12.68 | 39.17 | **1.45** | **32.82** | 36.16 | 33.99 | 45.51 | **4.66** | 0.35 | 1.65 | 0.37 |
| 60 | 10 | 99 | **100** | 61 | **100** | 14.48 | 2.98 | 16.56 | **0.10** | **32.96** | 33.36 | 35.96 | 35.92 | **4.70** | 0.34 | 1.83 | 0.45 |
| | 30 | 98 | **99** | 40 | **99** | 21.47 | 7.76 | 14.99 | **0.55** | **32.94** | 33.80 | 33.87 | 35.74 | **4.70** | 0.33 | 1.79 | 0.46 |
| | 60 | 80 | 89 | 12 | **90** | 30.51 | 12.04 | 23.17 | **0.97** | 32.77 | 35.02 | **31.60** | 36.16 | **4.67** | 0.34 | 1.85 | 0.47 |
| | 100 | 38 | 74 | 4 | **77** | 47.30 | 23.94 | 48.87 | **1.76** | **33.26** | 36.63 | 34.20 | 37.00 | **4.74** | 0.36 | 1.78 | 0.44 |

Table 3.1: Qualitative comparison results. Blue- Webb and van den Berg (2013); Green- Chen et al. (2015); Cyan- Campos-Macías et al. (2017); and Red- Ours.

gains obtained by minimizing the contouring error while maximizing the velocity:

$$j(t) = -12.0e(t) - 21.0v(t) - 10.0a(t)$$

The results of this benchmark are presented in Table 3.1, where it can be noticed that our method outperforms in success rate, especially with higher tree density and lower time outs, which was the primary goal of our algorithm. On the other hand, the Campos-Macías et al. (2017) method almost achieves the same level of success rate because our algorithm and theirs are based on computing first an RRT and use it as a base to generate a feasible trajectory. The difference is that Campos-Macías et al. (2017) must solve a QP problem, which is slow compared to ours, that only solves a chain of integrators. Therefore, we can observe that it does not obtain a solution in very short timeouts. Thus, although it has been shown that quadratic methods can be solved in allowable times, it is still greater than just performing system integration.

The main competitor to our algorithm is the method proposed by Webb and van den Berg (2013), since in general terms, they sample a possible configuration and try to connect it to the RRT tree, similar as in our proposed algorithm. In their case, they make the connection using an optimal control function, arguing asymptotic optimality. These methods are well suited for lower dimensional state space, but as this state-space increases, the connection becomes more difficult to do. Our approach shows superior performance in the success rate time out relation. Also, in the Table 3.1, we can observe that the algorithm of Webb and van den Berg (2013) outperformed in average path length. Nonetheless, this can be adjusted by increasing the sampling time in our method to obtain better path lengths since it is based on an RRT* with Euclidean distance for the cost function.

Finally, we decided to add the algorithm by Chen et al. (2015) to the comparison since this algorithm provides an optimal solution according to some criteria. In this case, we select the same that their results; to optimize the acceleration. In the Table 3.1, we can observe that their success rate is significantly higher than the method by Webb and van den Berg (2013) but still lower than our proposal; this is because, on each iteration, they have to solve a QP. However, the technique by Chen et al. (2015) outperforms in average velocity for all the experiments. The average velocity in our method can be improved by selecting another control input

signal that maximizes the invariant set induced by the Lyapunov function allowing to choose a more prominent boundary for $\mathcal{B}_{\hat{x},b}$, so the contouring to the line segments be faster.

# Chapter 4

# Multi-agent Navigation

The extension of the single MAV case to multiple agents navigation is described in detail in this chapter. This method assumes that all agents have a standard reference frame or that the transformations between their local reference frames are known. Also, the agents have a limited communication radius by which the desired trajectories are shared.

First, the logic to generate new references for the low-level controller is introduced. This logic takes into account the latest updates in the map and the intentions of neighboring agents. Then, we detail the key ingredient to achieve the multiagent navigation, which is an extension of the stochastic trajectory planning algorithm for the single-agent case. The extension of the F-DRRT avoids that the agents circumnavigate to make time for the agents' coordination and collision avoidance. Instead, the agents reduce their speed or stop to allow neighbors to pass without losing the capability of the RRT to solve scenarios like pockets and maintaining the benefits of a sampling-based asymptotically optimal method.

The method here described is based on the ranking of the agents to decide which agent has to stop and which can continue with its planned trajectory, i.e., each of the agents has a hierarchy over the others. Therefore, this chapter includes a brief analysis of the different forms for ranking selection and how this impacts our method's performance.

Finally, we evaluate our algorithm in cluttered scenarios with a high density of agents against the most popular methods for multiagent trajectory planning.

## 4.1 Collision Avoidance

The proposed motion planning algorithm that considers shared trajectories of $N$ agents is shown in the diagram flow of Figure 4.1 and illustrated in Figure 4.2. This algorithm extends the motion planning method for the single-agent case, explained with detail in Section 3.2. In summary, we consider that each MAV has a starting state $x^i_{t_c} = x^i_s$, where the notation $x^i_{t_c}$ indicates the state $x^i(t_c)$, and for this case, the superscript $^i$ will denote the $i$-th agent. Also, each of them has a goal position $p^i_f$ and has no information about the map or the other agents, as illustrated in Figure 4.2a. Like our previous approach, the algorithm waits for a new update in the map and then solves a motion planning method that returns a set of states $x^i_{[t_c:t_f]}$ that connects the initial state with the final goal. Then, the low-level controller executes the portion of the states $x^i_{[t_c:t_e]}$ that lies inside the current field of view.

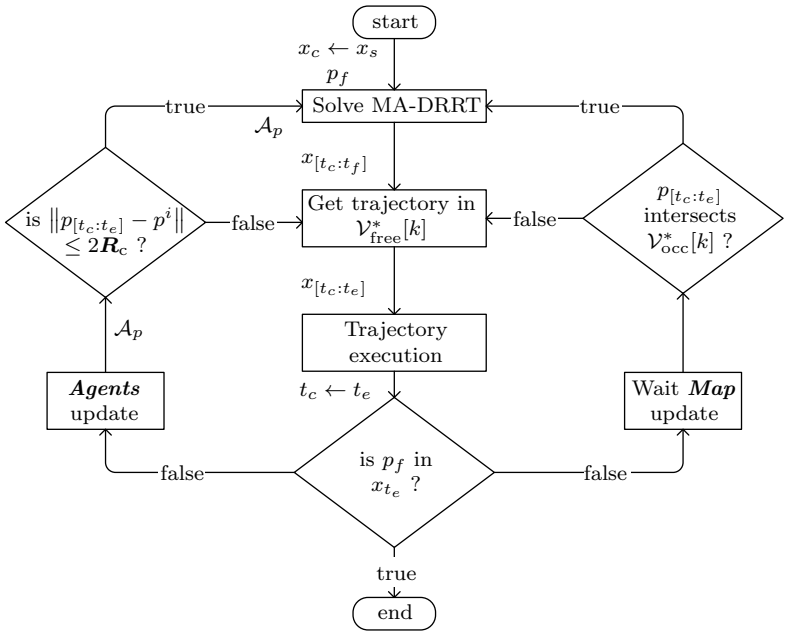



Figure 4.1: Flow diagram of the multiple agents motion planning algorithm for reaching goals in unknown environments with neighboring agents $p^i$.

When a new update to the map is computed, the next portion of states inside the new field of view is reviewed for collisions in the map, which were the only obstacles considered in the previous method. If no collision is detected, then this portion is also executed by the low-level controller. On the other hand, a new trajectory is generated using our modified version of the stochastic RRT planner if a collision is detected. This process is repeated until the goal is achieved, which means that $p_f$ is in $x_{t_e}$.

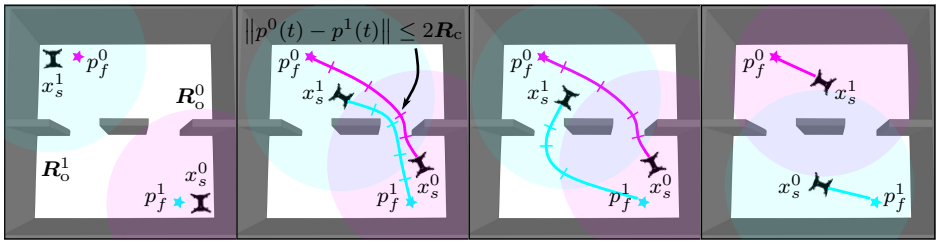
In addition, the algorithm described in the diagram flow of Figure 4.1 also considers that there is communication and cooperation between surrounding agents, i.e., a portion of the intended trajectory from the neighbors

$$\mathcal{A}_p = \left\{ p^i_{[t_c:t_e]} : i \in \mathcal{A} \setminus \{i_{\text{current}}\} \right\}$$

is broadcasted and received within a limited communication radius $\boldsymbol{R}_o$. Furthermore, we contemplate that each agent has a hierarchical level and computes its trajectory independently, meaning that our trajectory planning algorithm is decentralized. Thus, the agent $i_{\text{current}}$ with the lowest hierarchy level will constantly generate a new trajectory if a collision with one of its neighbours is detected, considering the highest ranking of the agents' intention and the current state of the map $\mathcal{V}^*_{\text{occ}}[k]$.

When a new trajectory arrives (block `Agents Update` in Figure 4.1), its time-dependent position $p^i_{[t_c:t_e]}$ is compared against the current position intention $p_{[t_c:t_e]}$. To make this section more readable, we will avoid using the superscript $^i_{\text{current}}$ when we refer to the current agent.

Suppose that the Euclidean distance of this comparison at some time instant $t$ is lower than a prescribed collision radius $\boldsymbol{R}_c$ as exemplified in Figure 4.2b (decision




(a) (b) (c) (d)

Figure 4.2: Illustration of the motion planning method for reaching a goal in unknown environments with agent collision avoidance: (a) Unknown environment with two agents, goals, and communication radius. (b) Initial proposed trajectories using the current map, showing a future collision. (c) Replanned trajectory after considering the intention of the agent with the highest hierarchy. (d) Final trajectories towards their goals.

block $\|p_{[t_c:t_e]} - p^i\| \leq 2\boldsymbol{R}_c$ ? in Figure 4.1). In that case, the agent with the lower hierarchy must create a new trajectory taking into account the intention of the neighboring agents $\mathcal{A}_p$ and the current state of the map $\mathcal{V}^*_{\text{occ}}[k]$. In the other case, the agent continues with the original plan.

To create a plan that considers the trajectories of neighboring agents and static obstacles, we propose the Multiple Agents-Dynamic Rapidly Exploring Random Trees (MA-DRRT) which extends our previous proposal, F-DRRT explained in Subsection 3.2.2. Thus, the agent with lower priority generates a trajectory that is collision-free respecting static obstacles $\mathcal{V}^*_{\text{occ}}[k]$ and avoids the surrounding agents, as exemplified in Figure 4.2c. Note that it is of paramount importance to avoid other agents by considering their position respecting time, since otherwise agents can occlude paths if they are considered static obstacles by the trajectory algorithm, leading to unfeasible solutions when the scene has at least one solution.

The method in Figure 4.1 is repeated until all agents achieve their final positions, as illustrated in Figure 4.2d, meaning that each agent explores the scene by updating their map representation and upgrading their planned trajectory if needed. They also consider the received intention of surrounding agents with a higher hierarchy.

To achieve that, an agent must include the positions in time of the agents within a communication radius and come up with a trajectory that avoids them; in this work, we propose the algorithm MA-RRT. The MA-RRT algorithm modifies the `Propagate` function in Algorithm 4, which is how the robot's dynamic system contours a line segment.

The previous function $\mathtt{x}_{\text{new}} \leftarrow$ `Propagate(`$\mathtt{x}_{\text{nearest}}$`,` $\mathtt{p}_{\text{rand}}$`)` is replaced in all the places where it appears with `valid,` $\mathtt{x}_{\text{new}} \leftarrow$ `Propagate(`$\mathtt{x}_{\text{nearest}}$`,` $\mathtt{p}_{\text{rand}}$`,` $\mathcal{A}_p$`)`. Thus, instead of receiving the initial state and a random position $p_{\text{rand}}$, the new proposal also gets the neighboring agents' positions $\mathcal{A}_p$. In addition, the function returns the state propagation along the vertex $x_{\text{new}}$, together with its viability, `valid`.

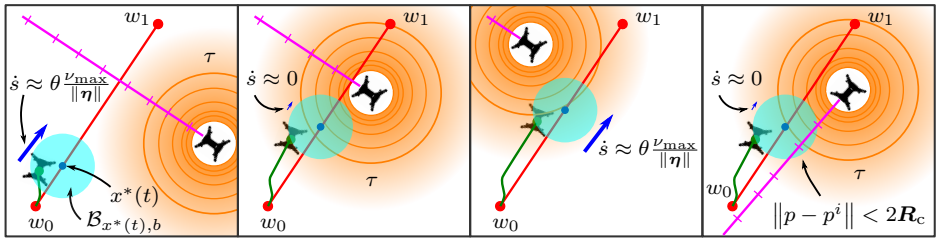



Figure 4.3: Illustration of the trajectory generation method with agent collision avoidance: (a) Representation of the 2D line segment contouring and $\tau$ distance-based function effect. (b) The speed of the equilibrium state $x^*(t)$ is almost zero, given the influence of $\tau$ in $\dot{s}$. (c) Speed of $\dot{s}$ is the selected boundary since a low impact of $\tau$ due to the relative distance between agents. (d) Special case; with $\dot{s}$ zero, the collision cannot be avoided with respect to the trajectory of a higher hierarchy agent.

In particular, in line 5 of Algorithm 4, if the `Propagate` function returns `false`, the random sample $p_{\text{rand}}$ is discarded. Meaning that a collision occurred with an agent's trajectory at some time of the new vertex generated (line segment), and therefore is not a valid sequence of states to be wired. Otherwise, the algorithm continues and uses the propagated states $x_{\text{new}}$.

Finally, at the rewiring stage in lines 9, 12, and 18 of Algorithm 4, the output of the `Propagate` function is used to calculate the cost, which is done if the reconnection is valid, i.e., if the propagate function returns `true`. Otherwise, reconnection is not considered. Like the function `CollisionFree`, if a collision exists, the new vertex cannot be generated even with a lower cost. In this way, the asymptotic optimality respecting the cost function selected of the generated paths is still conserved but considering the surrounding agents' trajectories.

Note that this does not mean that a globally optimal trajectory will be obtained respecting the cost function. The control input signal is not necessarily optimal, and confining the propagation to an invariant set dragged along the vertex neither ensures optimality. Further analysis of Theorem 5 can be made to demonstrate optimality. Moreover, for the single-agent case, the cost function was selected as the Euclidean distance between the initial and final positions of the line; in this particular case, the properties of the optimal RRT are conserved. For the case of multiple agents, this guarantee is not ensured.

The new trajectory generation algorithm is a modified version of the method described in Subsection 3.2.1. It uses the surrounding agent's intentions $\mathcal{A}_p$ as an input to generate a sequence of states $x_{\text{new}}$ that prevents a collision. If this is not possible, then an invalid flag is returned. The method uses the input state $x_{\text{nearest}}$ and the input position $p_{\text{rand}}$ to define $w_0$ and $w_1$, the start and end of a line segment, and the initial conditions for the yaw, velocity, acceleration and jerk. To ensure that the robot will always move in its field of view, the final yaw $\psi_1$ is taken as

$$\cos\psi_1 = \frac{\boldsymbol{w} \cdot \boldsymbol{w_x}}{\|\boldsymbol{w}\| \times \|\boldsymbol{w_x}\|},$$

with $\boldsymbol{w} = w_1 - w_0$ and $\boldsymbol{w_x}$ the unitary vector along $x$ axis.

Like the previous method, it ensures that the line segment is contoured with a maximum separation $\varepsilon_w$ from the line while satisfying the dynamic constraints of the MAV, including a prescribed deviation error $\varepsilon_\psi$ from the desired orientation $\psi_1$. The difference lies in the strategy of moving the equilibrium state $x^*(t)$ induced by the control signal along the input line segment to consider the agents' trajectories.

We assume that $\eta(s(t))$ is a point along the input line segment and that the control signal used for the single-agent case guarantees to have a set $\mathcal{B}_{\hat{x},b}$. We also parametrize $\eta$ with a continuous, positive, and non-decreasing function $s(t)$, with condition $\dot{s}(t) > 0$, and by Theorem 5, $x(t)$ exponentially reaches $[w_1, \psi_1, 0, \ldots, 0]^T$.

Figure 4.3 illustrates how this modified version of the algorithm works for a simplified 2D case. $\mathcal{B}_{\hat{x},b}$ is depicted as a 2-dimensional ball for illustrative purposes, and the magnitude of $\dot{s}(t)$ is represented with a blue arrow. Figure 4.3a shows the agent $i_{\text{current}}$ inside its invariant set with equilibrium moving state $x^*(t)$ and $\dot{s}(t) = \theta \frac{\nu_{\max}}{\|\boldsymbol{\eta}\|}$, where the green point is the robot's position at the current time, and the red line is the input line segment. The magenta line outlines the time-indexed position $p^i_{[t_c:t_e]}$ of a neighbor agent $i$. Finally, the orange gradient around the position $p^i(t)$ pictures the influence of a distance-dependent function $\tau\left(p(t), p^i(t)\right)$ to the value of $\dot{s}(t)$.

The key factor for avoiding surrounding agents is the distance-dependent function $\tau\left(p(t), p^i(t)\right) \to [0, 1]$. Thus, the new imposed dynamic for $\dot{s}$ is:

$$\dot{s}(t) = \theta \frac{\nu_{\max}}{\|\boldsymbol{\eta}\|} \tau\left(p(t), p^i(t)\right),$$

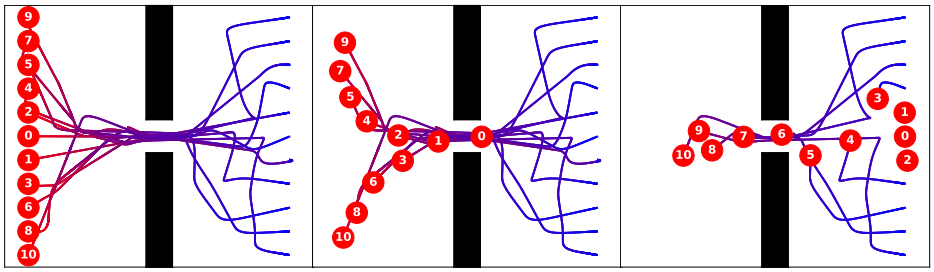
$$0 \leq \dot{s}(t) \leq \theta \frac{\nu_{\max}}{\|\boldsymbol{\eta}\|}.$$

A typical selection for $\tau$ is a function such that as the Euclidean distance between agents decreases, the evaluation of $\tau$ tends to zero. On the other hand, as the distance increases, $\tau$ goes to one. Furthermore, when the relative distance between the agents is equal to the collision radius $\boldsymbol{R}_{\text{c}}$, $\tau$ is zero. A function that fulfills those conditions is:

$$\tau\left(p(t), p^i(t)\right) = \begin{cases} 0.0 & \left\|p(t) - p^i(t)\right\| < 2\boldsymbol{R}_{\text{c}} \\ \min\left(1.0, \gamma\left(\left\|p(t) - p^i(t)\right\| - \boldsymbol{R}_{\text{c}}\right)^2\right) & \text{otherwise} \end{cases},$$

where $\gamma$ is a positive hyperparameter to decide the vanishing effect of $\tau$. In other words, when the relative distance between the agents decreases, since $\tau$ multiplies $\dot{s}$, the speed with which the equilibrium state $x^*(t)$ moves along the line segment is reduced, implying that the agent reduces its velocity. This effect is exemplified in Figure 4.3b, where $\tau$ has a more considerable influence on the speed of $s(t)$, small blue arrow. When the relative distance increases, $\tau$ tends to one, making $\dot{s}$ grow to its original value and recovering the speed of the equilibrium point $x^*(t)$, as shown in Figure 4.3c.

A special case occurs when the collision cannot be avoided, even if $\dot{s}$ is equal to zero, i.e., even if the agent stops. These cases arise for example when the agent with the highest priority moves in the opposite direction respecting the current agent's movement, as illustrated in Figure 4.3d. When this happens, the `Propagate` function simply returns an invalid state so that the vertex can be discarded and a new random position can be sampled.



(a) (b) (c)

Figure 4.4: Illustration of a narrow corridor scenario with eleven agents with a diameter of 0.5m in 2D view of a 3D map of 6m x 6m x 1m. (a) Shows the complete trajectories for each agent, the gradient represents the position respecting the time, red $t = 0$ and blue $t = t_f$. (b) Exposes the hierarchy selection for the agents, where the lower number represents the higher ranking. (c) Depicts the agents reaching their destinations with all conflicts solved.

In this way, collision avoidance among agents is performed at the path planning stage. Considering their positions respecting the time helps to generate better trajectories and significantly raises the success rate in contrast to considering the trajectories of agents as static obstacles. Furthermore, $\tau$'s effect prevents the agents from circumnavigating to wait for other agents while passing.

As explained before, the method considers the effect of the relative time dependant distance of the agent with the higher priority over the lower. In other words, the agent with the lower ranking has to reduce his velocity or change his route to avoid the collision respecting the higher priority agent. Thus, choosing the priorities is of paramount importance since this directly impacts the algorithm's performance.

Figure 4.4a shows a scenario with eleven agents and a narrow passage. The goal is to navigate through the corridor up to the opposite position respecting the start. Also, a lower number represents a higher priority. These types of scenarios highlight the importance of a correct selection of priorities. In this experiment, the ranking was selected as a function of the distance between the start and the goal; the shorter, the higher the hierarchy. This ranking is better than choosing the opposite; the higher the length, the higher order. Figure 4.4b shows how the agent with the highest priority is the first to pass through the corridor without obstructing any other agent. In contrast, selecting the agents from the corners with a higher ranking will block all the other agents in the middle, causing them to wait and resulting in a poor solution. Finally, Figure 4.4c shows the agents reaching their goal.

In this work, we propose to use the Euclidean distance between the start and the goal as a factor for choosing the ranking of the agents; the shorter distance, the higher hierarchy. A better method would be that each agent plan a trajectory considering only static obstacles and select the hierarchy as a function of the cost, the lower cost, the higher rank. Nonetheless, the final result of this work is to enable the agents to navigate through unknown scenarios; thus, using this approach is not suitable given that the absolute cost path is unknown.

Finally, we compare our method against two techniques: the distributed model predictive control method (DMPC) of Luis and Schoellig (2019) and the multiagent version of the decoupled incremental sequential convex programming procedure pro-

posed by Chen et al. (2015). The algorithm of Luis and Schoellig (2019) introduced an on-demand collision avoidance strategy for a modified version of the DMPC applied to multiagent point-to-point transitions. They discretize the translational dynamic equations and impose motion constraints. Also, they defined a boundary for each agent as an ellipsoid elongated along the vertical axis to capture the downwash effect of the agents' propellers. The method starts with each agent simultaneously computing a new input sequence over a predefined horizon and sharing it with their neighbors. Then, future collisions are reviewed using the latest predicted states and build the optimization problem, including state and actuation constraints and collision constraints only if required. This process is repeated until all the agents reach their destination.

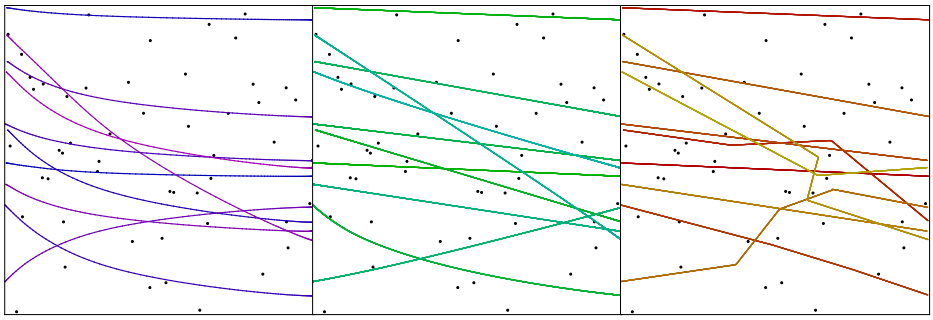
On the other hand, Chen et al. (2015) presented a trajectory planning method based on sequential convex programming (SCP) that finds locally optimal trajectories. In summary, an optimization problem is formulated in a quadratic programming (QP) fashion upon a cost function and linearized restrictions representing the map boundaries and the dynamic constraints. The method proposed to optimize the acceleration; thus, the solution returns a sequence of accelerations that connects the start and goal positions. Then, at each time step, collision with static obstacles or other agents is reviewed. If a collision is found, it is included as a new constraint in the QP problem. This process iterates until a feasible solution is found or until a maximum number of iterations is reached.

Another popular technique for multiagent coexistence is the Optimal Reciprocal Collision Avoidance (ORCA) method, proposed originally by Van Den Berg et al. (2011). While demonstrations with hundreds of agents have been shown, they usually performed them in static-obstacle-free scenarios since the technique is based on a projection of the velocity of surrounding agents imposing something similar as a field force vector, producing the same behavior of stuck in pockets. For those reasons it is not included in the comparison.

The evaluation of our algorithm was performed similarly to the single case of Chapter 3. Also, our complete method was programmed using the MatLab language. Again, the scenarios chosen were a set of Poisson forest environments generated using the technique proposed by Karaman and Frazzoli (2012). For this case, the size of the maps was $50\text{m} \times 50\text{m} \times 1\text{m}$ with a tree density varying from 0 to 50 with a limited radius of thirty centimeters and a constant height of 1m. For each tree density, twenty-five experiments were realized and applied to each algorithm with agent density ranging from 5 to 25 agents. In addition, the agents' start and goal positions were randomly sampled from one extreme of the map to the opposite to maximize the distance traveled and the conflicts between them. Figure 4.5 shows this scenario with 50 trees and the trajectories of 10 agents for the three algorithms.

In this comparison, we evaluate the quality of the solutions for the generated trajectories: success rate, average trajectory length, and maximum time to reach all of the goals. As we did in the comparison of Chapter 3, to avoid inducing a bias in the benchmark, only the scenarios that the three algorithms solved were selected to generate the statistics of path length and maximum time; nonetheless, every trial counted to obtain the success rate.

We used the jerk as a control input signal to increase the success rate for both algorithms chosen, and only the translational part of the model was considered. For the method proposed by Luis and Schoellig (2019), we tuned the horizon length to 30 samples with a time step of 0.01 seconds. The order for the super ellipsoid



(a) (b) (c)

Figure 4.5: Top view of the simulated forest environment, together with the traversed paths for ten agents, the gradients in the colors represent a different agent. (a) Luis and Schoellig (2019); (b) Chen et al. (2015); (c) Ours.

| **Agents** | **Trees** | **Success Rate [%]** | | | **Avg. Path Lenght [m]** | | | **Avg. Max Arrival Time [s]** | | |
|---|---|---|---|---|---|---|---|---|---|---|
| 5 | 0 | 88 | 68 | **100** | 106.24 | **104.11** | 105.59 | 37.62 | **37.51** | 308.47 |
| | 25 | 84 | 64 | **100** | 107.95 | **105.84** | 109.73 | 43.00 | **42.87** | 321.97 |
| | 50 | 72 | 40 | **100** | 106.13 | **104.12** | 106.04 | **62.02** | **60.02** | 312.46 |
| 10 | 0 | 56 | 16 | **100** | 106.28 | **104.12** | 106.52 | 528.00 | 425.00 | **316.20** |
| | 25 | 48 | 8 | **100** | 110.38 | **107.63** | 111.21 | 555.87 | 501.20 | **346.20** |
| | 50 | 28 | 4 | **100** | 107.03 | **104.97** | 109.28 | 587.00 | 499.98 | **373.93** |
| 25 | 0 | 24 | 0 | **100** | - | - | - | - | - | - |
| | 25 | 4 | 0 | **100** | - | - | - | - | - | - |
| | 50 | 0 | 0 | **100** | - | - | - | - | - | - |

Table 4.1: Qualitative comparison results. Purple- Luis and Schoellig (2019); Green- Chen et al. (2015); and Red- Ours.

constraint was 2 with $[1.0, 1.0, 1.5]$ as a diagonal matrix. The workspace boundaries were the same as the map size, and the maximum acceleration was $5m/s^2$. For the algorithm proposed by Chen et al. (2015), we select a final time randomly between 30 and 800 seconds with a time step of 0.04 seconds and an acceleration boundary of $5m/s^2$. We also use the ellipsoid shape for collision avoidance between agents, as proposed initially by the authors, formed by the diagonal matrix $[1.0, 1.0, 2.0]$. The incremental sequential programming method depends heavily on the selection of the final time. An incorrect choice leads to an infeasible solution. We found that a random election between those values increased the algorithm's success rate compared to a fixed value.

Finally, for our method, we only used the translational part, selected a $\dot{s} = \frac{10}{\|\boldsymbol{\eta}\|}$, and also used the jerk as control input with control law:

$$j(t) = -12.0e(t) - 21.0v(t) - 10.0a(t)$$

and $\tau$ function:

$$\tau\left(p(t), p^i(t)\right) = \begin{cases} 0.0 & \left\|p(t) - p^i(t)\right\| < 0.5 \\ \min\left(1.0, 3.0\left(\left\|p(t) - p^i(t)\right\| - 0.5\right)^2\right) & \text{otherwise} \end{cases}.$$

The results of the comparison are shown in Table 4.1. It can be noticed that our proposal solved all the experiments for all the cases taken into account, outperforming at the success rate metric. Moreover, the success rate falls rapidly in the

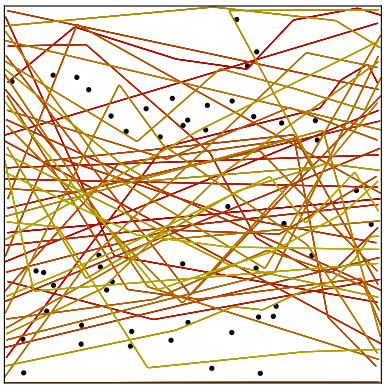


Figure 4.6: Top view of the simulated forest environment for a 75 trees density, together with the traversed paths for 50 agents, the gradients in the colors represent a different agent.

method of Chen et al. (2015) because the agents and static obstacles are treated in the same form, i.e., as a constraint at that specific time step for the quadratic problem. The selection of the final time for the trajectories must be the same for all agents; this increases the chances of failure and the method's dependence on specific scenarios with a limited number of agents.

In the case of 25 agents and 50 trees, none of the compared algorithms solved any scenario. The method of Luis and Schoellig (2019) uses a finite time horizon to compute trajectories collision-free; when the static obstacles and the agents increase, that time horizon must be different, so the algorithm can anticipate and avoid the collisions.

The average path length was similar for three of the algorithms; in particular, our algorithm took advantage of the asymptotic optimality given by the RRT*.

Like the single case benchmark, our algorithm lacks at finding trajectories at high speeds, inducing a more prominent time for the agents to reach their goals. Nonetheless, we can observe how the time increases slightly as the number of agents increases. At the same time, the path length maintains similarity between experiments, showing the effect of the $\tau$ function preventing the agents from circumnavigating.

We further evaluate our algorithm behavior with a set of 25 experiments in a scenario of the same size but with a density of 75 trees of fifty centimeters and 50 agents. The resultant trajectories are shown in Figure 4.6. The average path length for this case was 30m, with an average maximum time to reach the goals of 10secs.

# Chapter 5

# Results

We compared exhaustively the performance of our proposed method. In this chapter, we present the results obtained. First, we used a single agent to get quantitative metrics about the computation time of our algorithms. Second, we also include some statistics about the quality of the resultant trajectories, such as velocity or trajectory length.

We used three main scenarios as a testbed for the algorithms; forest environments, a maze, and two floor-up goal scenarios. Finally, we included a comparison with two important state-of-the-art algorithms. Then, this chapter includes the extension of the single case to multiple drone navigation, using the same testbed scenarios against one state-of-the-art algorithm.

Finally, this chapter concludes with the evidence of the real-time implementation of our framework in a commercial drone of less than one kilogram.

## 5.1 Algorithm Benchmark

This section presents different performance benchmarks on simulated MAV platforms for the single-agent case and his extension to multiple drone navigation. First, we show a quantitative comparison with the state-of-the-art on simulated Poisson forest scenarios. Then, we demonstrate results in a complex maze scenario to gain further insight into the system's qualitative behavior.

The simulation experiments for all the methods in the benchmark for the single-agent case were performed on an Intel Core i7-5557U @ 3.1 GHz, 8 GB of RAM with Ubuntu 16.04-LTS operating system. On the other hand, the experiments were executed for multiple drone navigation on an Intel Core i7-10710U @ 1.10GHz, 16 GB of RAM with Ubuntu 20.04-LTS operating system.

The proposed algorithm was implemented using the C++ language with the ROS Kinetic and ROS Melodic frameworks (Quigley et al., 2009) for the single-agent case and multiple drone navigation. The environments were simulated using Gazebo alongside Rotor-S MAV (Furrer, Burri, Achtelik, & Siegwart, 2016) and RealSense simulation packages. The simulated robots were selected as the Hummingbird, available in Rotor-S, with a RealSense depth camera.
Finally, Table 5.1 shows the hyperparameters used by our algorithm for all benchmark experiments.

| **Map Generation** | |
|---|---|
| max depth distance | 3.5m |
| voxel radius | 0.15m |
| max map size | 30m |
| frame size | $640 \times 480$ pixels |
| horizontal FoV | 70deg |
| vertical FoV | 46deg |
| **Path Planning** | |
| robot radius ($r$) | 0.27m |
| RRT bounds | $[-30, -30, -30] \times [30, 30, 30]$m |

| **Trajectory Generation** | |
|---|---|
| $\varepsilon_w$ | 0.10m |
| $\varepsilon_\psi$ | 3.0deg |
| $V_{\max}$ | 2.5m/s |
| $A_{\max}$ | 1.0m/s$^2$ |
| $J_{\max}$ | 2.0m/s$^3$ |
| $\Omega_{\max}$ | 0.2rad/$s$ |
| $s(t)$ | $-13.0e - 33.0v - 31.0a - 10j$ |
| $\alpha(t)$ | $-6.0\psi - 5.0\omega$ |

Table 5.1: Hyperparameters used by our algorithm in benchmark simulation.

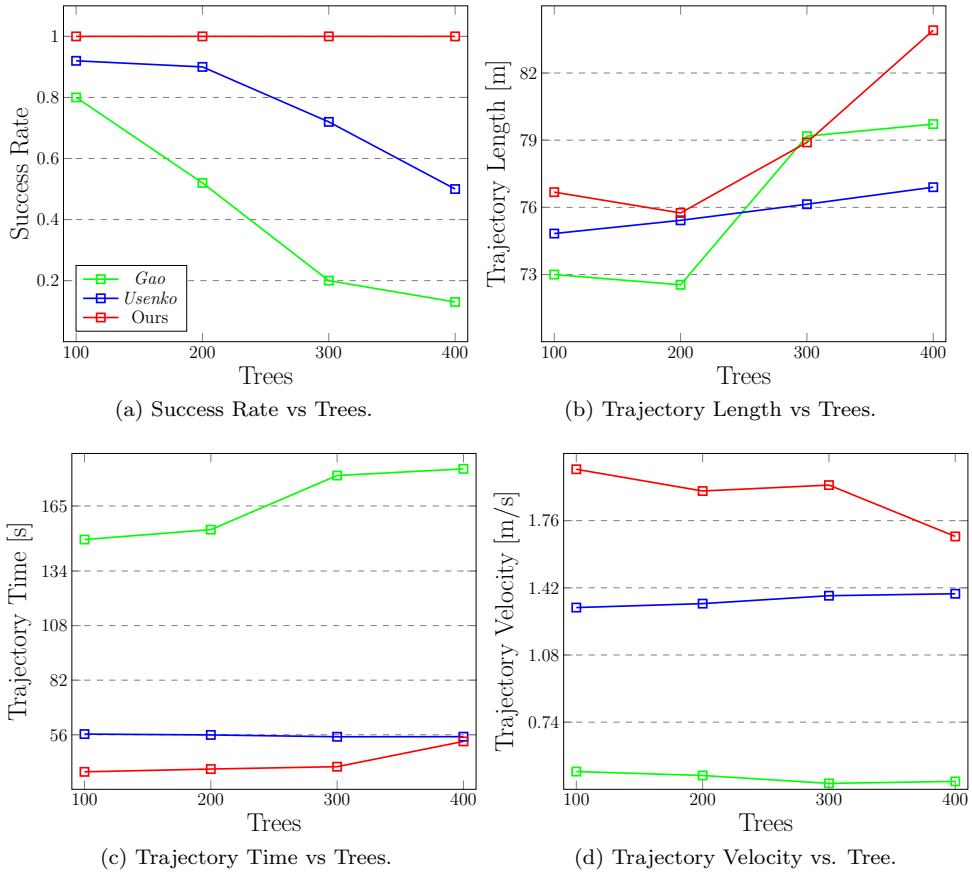
### 5.1.1 Single Agent

This subsection focuses specifically on a single agent reaching a goal in an unknown environment problem. Unfortunately, from the contributions reviewed in Table 1.1 dealing with this problem, only the works by Gao et al. (2018) and by Usenko et al. (2017) have made their code available. For this reason, we limit our comparison to only these two frameworks.

The simulated experiments were performed in several Poisson forest scenarios, generated using the method proposed by Karaman and Frazzoli (2012), in a space of 50m × 50m × 2m. The trees, numbering between 100 and 400, were limited to a height of two meters and a radius of twenty centimeters. More than 20 simulated forests were created for each tree density and then applied to each of the three algorithms running on the simulated MAV. The start and goal positions of the robot were chosen in opposite corners to maximize the distance traveled. Two sets of metrics were analyzed in the benchmark. The first set relates to the computation time of each algorithm, distinguishing between map generation time and motion generation time. The second set describes the overall performance of the generated trajectories in each algorithm: success rate, total trajectory length, average velocity, and time to reach the goal. The results show significant differences in computation time and perhaps more critical differences in key performance metrics such as success rate and average velocity.

Some parameters had to be tuned to obtain a success rate as high as possible in our simulated forest with 400 trees, starting from the configuration reported in each paper. The method, as reported by Gao et al. (2018), assumes a 360-degrees horizontal FoV and ten meters sensing range. For this reason, in the experiments, for (Gao et al., 2018), the observable space was set to cover a horizontal FoV of 360-degrees by using six RealSense cameras with a depth limit of 7m on the multirotor MAV. The maximum velocity of the robot was set to 0.9m/s, the maximum acceleration to 0.9m/s$^2$ and the check horizon and stop horizon parameters were set to 15.0m and 3.0m, respectively.

For the implementation of the algorithm from Usenko et al. (2017), the best success rate in our experiments was obtained by setting a maximum velocity of 1.5m/s and a maximum acceleration of 0.7m/s$^2$ for the robot. In addition, the distance threshold, number of optimization points, and the $dt$ parameters of the algorithm were set to 0.5m, 5 and 0.5s, respectively. The results of this benchmark are presented in Figure 5.1 and Figure 5.2.




(a) Success Rate vs Trees.

(b) Trajectory Length vs Trees.

(c) Trajectory Time vs Trees.

(d) Trajectory Velocity vs. Tree.

Figure 5.1: Algorithm comparison among the methods in (Gao et al., 2018), (Usenko et al., 2017), and the proposed approach on different forest environments varying on number of trees.

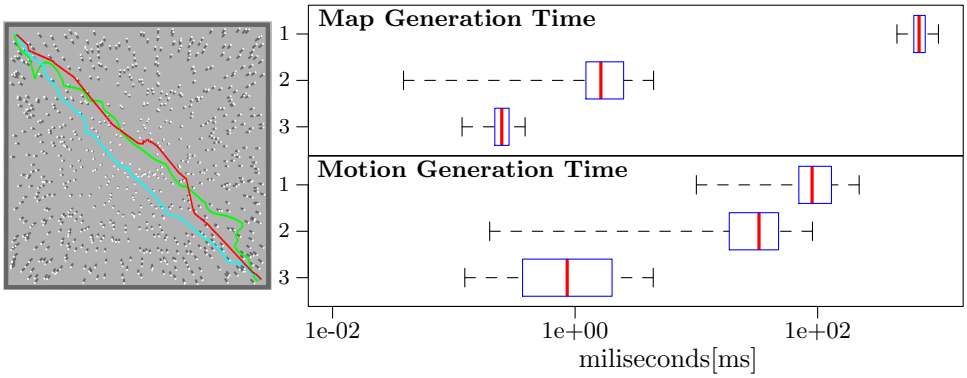



Figure 5.2: Qualitative comparison against (Gao et al., 2018) and (Usenko et al., 2017). **On the left.** Top view of the simulated forest environment, together with the traversed paths. Cyan- Usenko et al. (2017); Green- (Gao et al., 2018) and Red- Ours. **On the right.** Boxplot, on a logarithmic scale, of the computed time of each algorithm for map generation (Section 3.1) and motion generation (Section 3.2). (1) (Gao et al., 2018); (2) Usenko et al. (2017); and (3) Ours.

Regarding the performance of the algorithms on the task of navigating in a random forest, the results are presented in Figure 5.1 where it can be noticed that our method outperforms (Gao et al., 2018) and (Usenko et al., 2017) in success rate, especially as the tree density increases. Looking at the graphs for trajectory length and velocity, the increase in length and the corresponding decrease in speed suggest that our method could change direction and find alternative paths when required, considering the relatively short-range and narrow field of view of the perception system.

The other methods don't vary, especially in velocity, but their success rate drops dramatically. In our opinion, one of the main reasons for the low performance is that dynamic constraints in yaw and acceleration are not taken into account in the trajectory generation leading to two types of failures.

On the one hand, collisions occur, either because the robot is moving too fast and is unable to stop before a detected obstacle or because it changes direction too quickly and is unable to align the FoV with the velocity before running into unobserved obstacles. On the other hand, the robot became unstable when the controller tried to follow trajectories requiring large accelerations in some cases. In the particular case of (Gao et al., 2018), a type of failure which occurred frequently was that the non-linear optimization was unable to find a solution given the dynamic restrictions imposed. The robot would freeze in its current position whenever this happened, and the simulation had to be aborted.

In terms of computation, our algorithm outperforms Gao et al. (2018) and Usenko et al. (2017) in the metrics shown in Figure 5.2, with an average motion generation time of 3.37ms against 103.2ms obtained by Gao et al. (2018) and 35.5ms obtained by Usenko et al. (2017). This highlights the advantage of not having to solve a quadratic optimization problem online. In terms of average mapping time our algorithm took 0.256ms against 700.7ms obtained by (Gao et al., 2018) and 2.035ms obtained by (Usenko et al., 2017)

The average computation time for map generation is measured from when a new depth image is ready until the map update is complete. The motion generation time is computed from the time a replan is requested following a map update until the reference trajectory for the controller is ready. The results in Figure 5.2 were obtained by running each algorithm in ten random forest scenarios with the maximum density. The reason the algorithm from (Gao et al., 2018) is much slower than the others is that it was originally implemented in a powerful GPU while our tests were on a CPU.

A second scenario in which the algorithms were tested is in the maze environment shown in Figure 5.3. This scenario is challenging as a successful completion requires escaping *pockets* and maneuvering in very tight spaces. Hence, navigating inside the field of view becomes of paramount importance. A typical path execution using our algorithm is illustrated in Figure 5.3. Notice the loops in the trajectory where the MAV encountered pockets. Rather than stopping entirely and then turning around while hovering in place, the MAV was able to turn around while continuing to make forward progress safely by navigating inside the field of view all the time.

We did our best to complete this scenario with the algorithms (Gao et al., 2018; Usenko et al., 2017) but unfortunately, none of our attempts were successful. In the case of the method given by Usenko et al. (2017), we believe that failure to take into account yaw dynamical constraints of the system in the planning algorithm

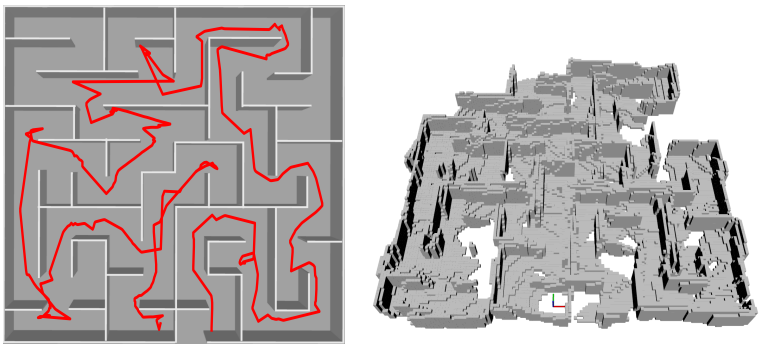


Figure 5.3: Navigation in an unknown maze environment. **On the left.** In red, the generated trajectory of our algorithm to reach the goal. The start and goal positions are given in blue and red, respectively. **On the right.** The map created while finding the goal.

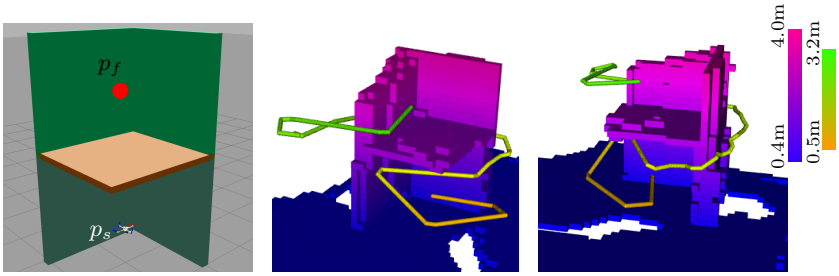
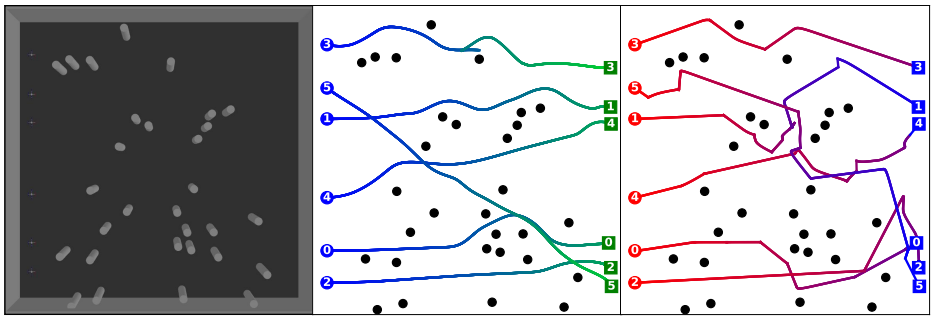



Figure 5.4: Goal at 3.5 meters on top of the drone. The left image is the Gazebo environment. The center and right images shows different views of the map and trajectory created by our algorithm

was the leading cause for the navigation always ending in collisions with the maze walls. In the case of the method given by Gao et al. (2018), it had the advantage of six cameras covering the full 360-degrees of the horizontal field of view, and we carefully tuned the algorithm parameters, including the polynomial order in the optimizer. Nonetheless, the maze environment could not be completed because, in every experiment, the optimizer could not find feasible solutions even after many re-plannings.

This environment demonstrates the advantage of the trajectory generation algorithm proposed here. As the results indicate, compared to polynomial trajectories, our method has the advantage of navigating through tight corners, such as doorways in an indoor environment.

To demonstrate the importance of navigation consistently inside the FoV, we include a third challenging scenario that involved a vertical climb in reaching a goal from a starting position near the ground and with a significant obstruction in the middle. We only tested our method in this simulation. This scenario is shown in Figure 5.4.



(a) (b) (c)

Figure 5.5: The top view of the simulated forest environment with 30 trees and the traversed paths for six agents, along with their priority. The gradients in the colors represent the positions respecting the time, circles, and squares, their starts, and goals. (a) Initial conditions of the experiment; (b) Zhou, Wen, et al. (2021); (c) Ours.

The MAV was initially facing two vertical walls at a short distance from it, and it had to move up to reach the goal with a significant obstruction placed directly on top of it. Figure 5.4 shows two different views of the resulting trajectory found by our algorithm to reach the goal after successfully navigating away from the obstacles. This scenario once again illustrates the importance of navigation inside the field of view.

### 5.1.2 Multiple Agents

The performance of the extension to multiple agents navigation in unknown cluttered environments is evaluated in this subsection. Moreover, the work of Zhou, Wen, et al. (2021) is the closest approach to our method that provided open-source code; for that reason, we also compare against them.

Zhou, Wen, et al. (2021) presented a decentralized and asynchronous systematic solution for multi-robot autonomous navigation. The main contribution is a gradient-based planning system where collision avoidance is formulated as a penalty in a nonlinear optimization problem, including a lightweight topological trajectory generation method for escaping local minima. They also detect the neighboring agents in the depth images for relative location correction and remove them from their map.

We continue using the base scenario used along with this thesis to evaluate the performance of the different algorithms, Poisson forest scenarios, generated using the method proposed by Karaman and Frazzoli (2012). For this case, we used maps of 15m×15m×1m with a tree density varying from 30 to 100 with a limited radius of twenty centimeters and a constant height of 1m.

Twenty experiments were realized and applied to both algorithms with an agent density ranging from 3 to 10 for each tree density. In addition, the agents' start and goal positions were randomly sampled from one extreme of the map to the opposite to maximize the distance traveled and the conflicts between them. Figure 5.5 shows this scenario with 30 trees and the trajectories of 6 agents for the algorithms.

| Agents | Trees | Success Rate [%] | | Avg. Path Lenght [m] | | Avg. Max Time [s] | |
|---|---|---|---|---|---|---|---|
| 3 | 30 | 90 | **100** | **17.21** | 26.73 | **16.35** | 25.04 |
| | 60 | 45 | **100** | **18.02** | 41.91 | **15.91** | 45.76 |
| | 100 | 27 | **100** | **21.52** | 70.08 | **19.91** | 139.71 |
| 6 | 30 | 63 | **100** | **16.39** | 29.79 | **15.25** | 27.41 |
| | 60 | 0 | **100** | - | - | - | - |
| | 100 | 0 | **100** | - | - | - | - |
| 10 | 30 | 0 | **100** | - | - | - | - |
| | 60 | 0 | **100** | - | - | - | - |
| | 100 | 0 | **100** | - | - | - | - |

Table 5.2: Qualitative comparison results. Blue- Zhou, Wen, et al. (2021); and Red- Ours.

In this comparison, we evaluate the quality of the solutions for the generated trajectories: success rate, average trajectory length, and maximum time to reach all of the goals. As we did in all comparisons, to avoid inducing a bias in the benchmark, only the scenarios that the algorithms solved were selected to generate the statistics of path length and maximum time; nonetheless, every trial counted to obtain the success rate.

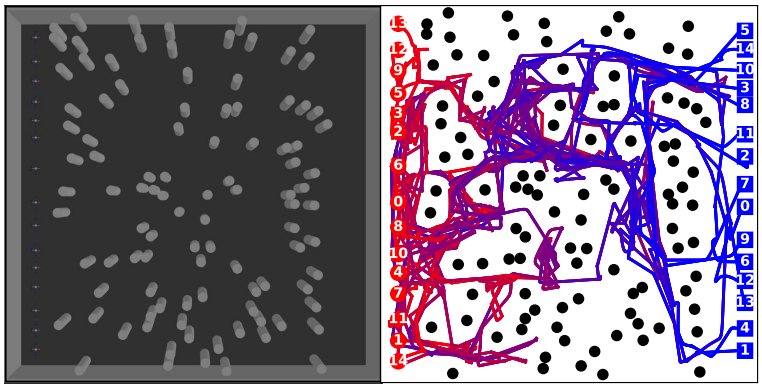
We tuned some parameters of our method and the method of Zhou, Wen, et al. (2021) to obtain a success rate as high as possible. For the algorithm of Zhou, Wen, et al. (2021), we started from the parameters already present in their code when we consulted it. We used the same RealSense cameras for all agents with an image resolution of 640 x 480 pixels, a map size of 42m, 30m, and 1m for the $x$, $y$, and $z$ boundaries with a resolution of 0.1m and obstacles inflation of 0.2m. The available setup included a module for odometry simulation. We removed it and used the ground truth provided by Gazebo since odometry performance is out of the scope of this evaluation. A maximum velocity and acceleration of 2.0m/s and 6.0m/s$^2$ with a planning horizon of 7.5m. $\lambda_s = 1.0$, $\lambda_c = \lambda_w = \lambda_t = 0.5$, $\lambda_d = 0.1$. Re-planning is triggered every second, or a collision is predicted.

For our method, we use the exact parameters of Table 5.1. We assumed an agent radius $\boldsymbol{R_c}$ of 0.27m; thus our proposed $\tau$ function was:

$$\tau\left(p(t), p^i(t)\right) = \begin{cases} 0.0 & \left\|p(t) - p^i(t)\right\| < 0.54 \\ \min\left(1.0, 10.0\left(\left\|p(t) - p^i(t)\right\| - 0.54\right)^2\right) & \text{otherwise} \end{cases},$$

with a communication radius $\boldsymbol{R}_\text{o}$ of 5.0m on intervals of 0.5s and 20.0s of trajectory length. Since the two methods are hierarchical scheme-based, we set the same priority of the agents for both proposals to keep the benchmark as impartial as possible.

The results of the comparison are shown in Table 5.2. Our method outperforms the success rate by solving 20/20 experiments for all tree densities and varying agents, while Zhou, Wen, et al. (2021) success rate falls to zero as the number of trees or agents increases. Their method is founded on the solution of their optimization method, which finds a safe trajectory that satisfies a distance depending on a pair position-velocity generation. Thus, a pair position-velocity generation that surrounds the obstacles becomes essential for obstacle collision avoidance. We believe that this technique, deformation of the trajectory upon a pair position-velocity, is suitable for maps where the size of the obstacles is known or constant and sufficient



(a) (b)

Figure 5.6: The top view of the simulated forest environment with 100 trees and the traversed paths for 15 agents, along with their priority. The gradients in the colors represent the positions respecting the time, circles, and squares, their starts, and goals. (a) Initial conditions of the experiment; (b) Ours.

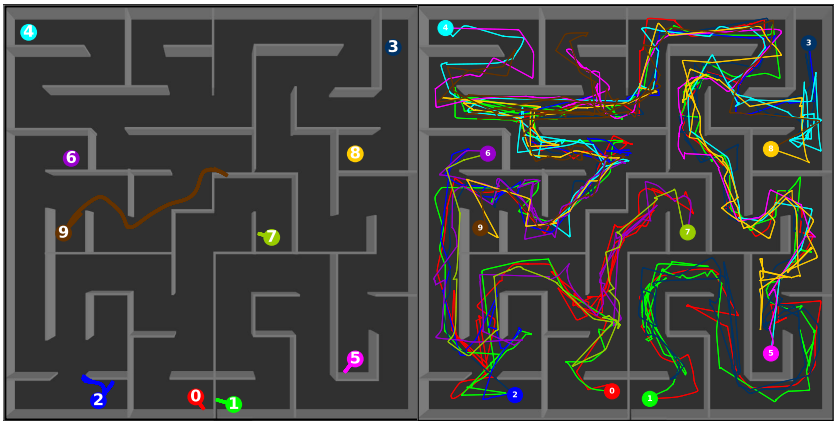
space between them exists, which is the case of the scenarios reported in their paper.

Nonetheless, even though the trees' radius is the same in our scenarios, the space between them is randomly generated. This caused that the trajectory's deformation could not to be computed in time, leading the agents to collide with the obstacles. Also, the non-linear optimization problem did not find a valid solution in repeated scenarios with high tree density.

Since Zhou, Wen, et al. (2021) started with a straight line between the start and goal positions and deformed it depending on the sensed obstacles, they outperformed the average trajectory length to reach the goals. Nevertheless, we could increase the boundary time for our anytime MA-RRT and get solutions closer to the optimal. Finally, our method took more time for all agents to reach their goals influenced by two factors; the input signal $j(t)$ could be improved to generate a less restrictive velocity condition and by the $\tau$ function's effect that induces a speed reduction.

To demonstrate the capability of our algorithm to handle highly cluttered environments together with a large density of agents, we further evaluate it in twenty random forest scenarios of 15 agents with a radius of 0.27m and 100 trees of 0.2m radius with a height of 1.0m in a 15m×15m×1m size map. The scenario and the resultant trajectories are shown in Figure 5.6. Note that because of the radius of agents considered, the maximum number of agents that fulfill Problem 1, i.e., starting and final positions didn't overlap, were 15 agents. For this case, only our method was available to solve 20/20 experiments. The average position length was 86.38m, with an average maximum time to reach the goals of 551.82s.

The second scenario in which we tested the algorithms was the maze environment shown in Figure 5.7. As mentioned in the single-agent analyses, these scenarios are challenging because the algorithm needs the ability to escape *pockets*, keep in memory the explored map, and avoid infinite loops at the discovery stage. Moreover, by adding multiple agent navigation to the maze, the coordination of the robots is of paramount importance to successfully achieve their destinations. For this case, we set the drones to swap positions by pairs, i.e., the agent identified with *zero*



(a) (b)

Figure 5.7: Multiple drone navigation in an unknown maze environment. The color represents the trajectory of each one of the ten agents. The goal for the agents is to swap positions with a pairing agent. 0-1, 2-3, 4-5, 6-7, 8-9. (a) Zhou, Wen, et al. (2021); (b) Ours.

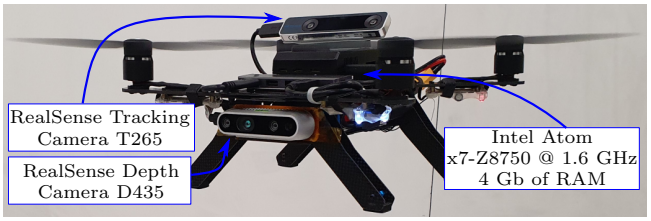



Figure 5.8: Intel Aero Ready-To-Fly drone system setup used for real time experiments

label must achieve the initial position of the drone with *one* label as a goal, and the *one* agent has to move to the initial position of the *zero* agent. We selected this configuration to increase the conflicts between robots and exhaustively evaluate our algorithm. A typical paths execution using our algorithm is illustrated in Figure 5.7b, where each color represents a different agent.

We also perform the same evaluation to the algorithm of Zhou, Wen, et al. (2021), with the same map and the same starting and final configurations. The resultant trajectories are shown in Figure 5.7a. Unfortunately, none of our attempts were successful, despite that we did our best while tunning their hyperparameters. As we mentioned before, creating a trajectory between the start and goal positions may result in infeasible trajectories. This is the case of the *zero* agent going to the *one* agent position, where a deformed straight line with a pair position-velocity is insufficient.

## 5.2 Real-World Experiments

In this section, the real-world results of our proposal are shown. We first describe the platform characteristics used for all the experiments, illustrated in Figure 5.8. Then, we describe the software architecture used to implement our setup entirely onboard.

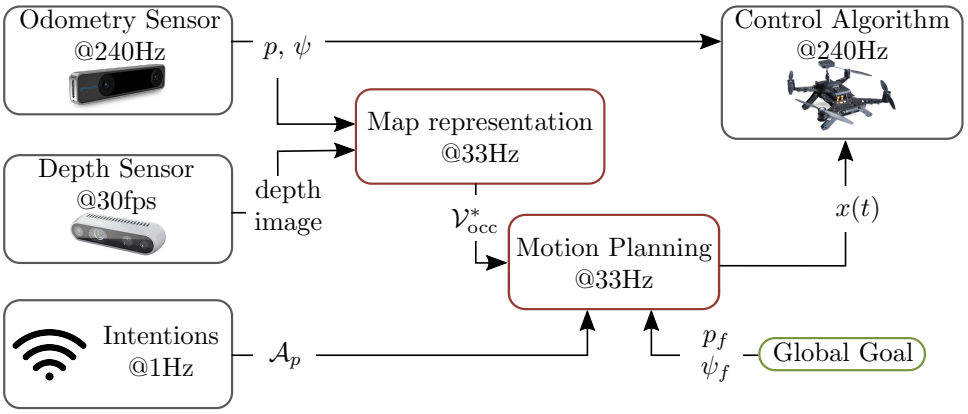



Figure 5.9: Illustration of the system architecture implemented on Intel Aero Ready to Fly Drone.

| **Map Generation** | | **Trajectory Generation** | |
|---|---|---|---|
| max depth distance | 3.0m | $\varepsilon_w$ | 0.15m |
| voxel radius | 0.2m | $\varepsilon_\psi$ | 2.5deg |
| max map size | 25m | $V_{\max}$ | 0.35m/s |
| frame size | $640 \times 480$ pixels | $A_{\max}$ | $0.078\text{m/s}^2$ |
| horizontal FoV | 85.2deg | $J_{\max}$ | $0.98\text{m/s}^3$ |
| vertical FoV | 58deg | $\Omega_{\max}$ | $0.2\text{rad}/s$ |
| **Path Planning** | | $s(t)$ | $-13.0e - 33.0v - 31.0a - 10j$ |
| | | $\alpha(t)$ | $-6.0\psi - 5.0\omega$ |
| robot radius ($r$) | 0.45m | | |
| RRT bounds | $[-25, -25, -25] \times [25, 25, 25]$m | | |

Table 5.3: Hyperparameters used by our algorithm in real-world experiments.

We implemented the framework on an Intel Aero Ready-To-Fly drone kit[1]. The kit comes with an Intel Atom x7-Z8750 @ 1.6 GHz processor and 4 Gb of RAM. We replaced all the firmware, installed Ubuntu 16.04-LTS operating system with ROS Kinetic, and added an Intel RealSense Tracking Camera T265 [2] for visual-inertial odometry (which has built-in loop closures capability) and an Intel RealSense Depth Camera D435[3] as our sensor for mapping and obstacle avoidance.

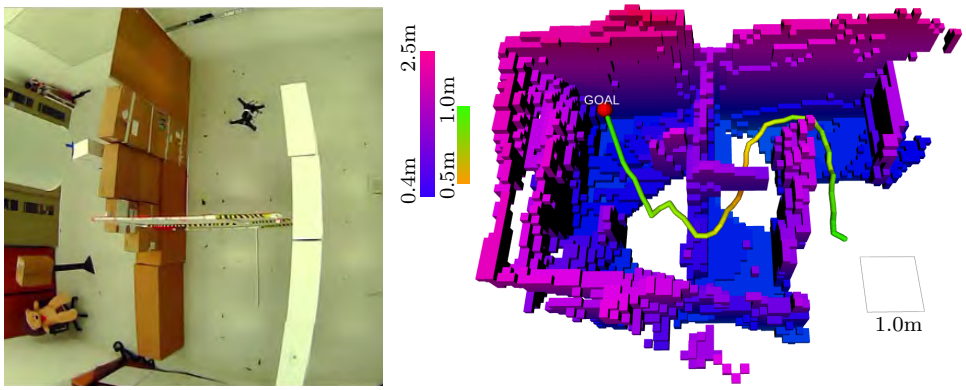
The software architecture is illustrated in Figure 5.9. The entire system runs entirely on the Atom processor onboard the MAV. Mapping and motion planning algorithms run at the camera rate of 33Hz. In addition, a custom nonlinear controller and sensor fusion algorithms run at 240Hz to enable smooth and precise trajectory tracking.

Finally, Table 5.3 lists all the parameters used for all the real-world experiments.

### 5.2.1 Single Agent

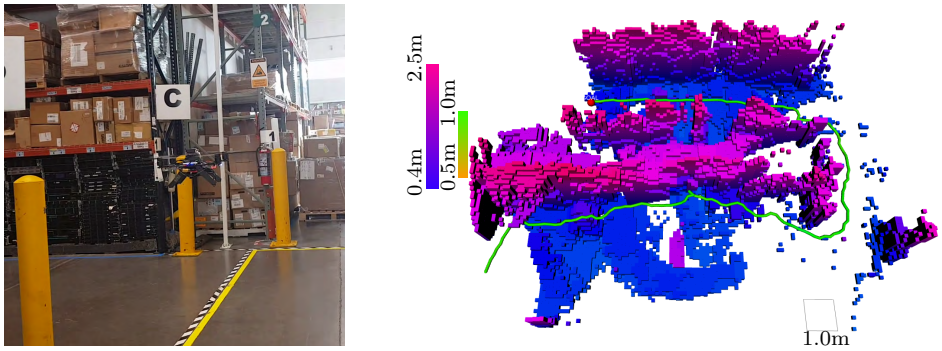
To illustrate the effectiveness of a single agent navigating in real unknown scenarios, experiments were performed in the different static and dynamic environments shown in Figures 5.10-5.13. A video of the different tests can be found in https://youtu.be/79IFfQfvXLE.

[1] https://www.intel.com/content/www/us/en/products/drones/aero-ready-to-fly.html
[2] https://www.intelrealsense.com/tracking-camera-t265/
[3] https://www.intelrealsense.com/depth-camera-d435/




(a) (b)

Figure 5.10: A 3D maze real-world experiment: (a) a picture of the robot in the 3d maze scenario performing the experiment (b) the map representation generated together with the generated trajectory.

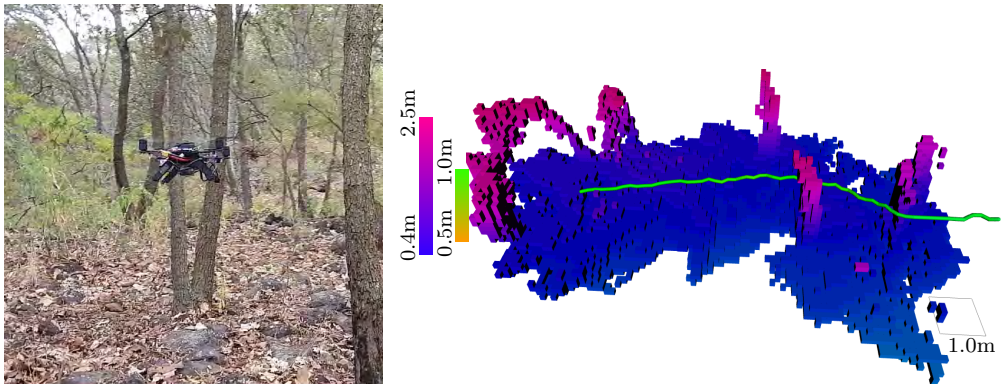



(a) (b)

Figure 5.11: A Warehouse real-world experiment: (a) a picture of the robot in the warehouse scenario performing the experiment (b) the map representation generated together with the generated trajectory.

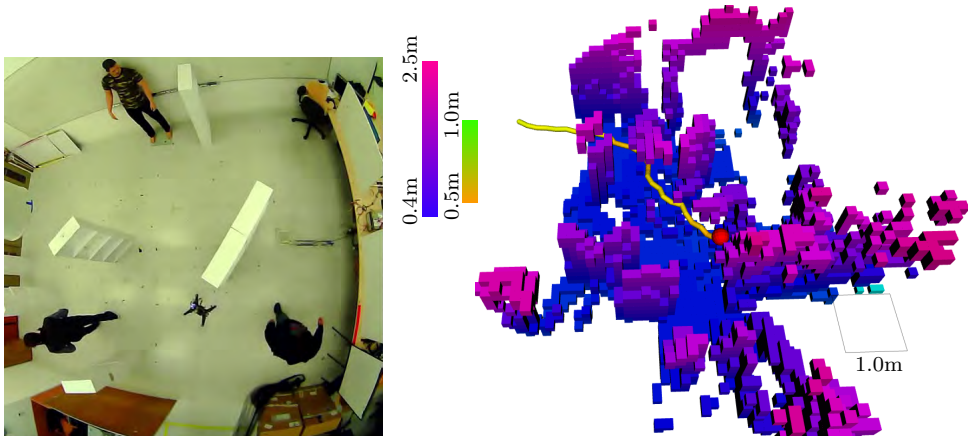
With the configuration of Table 5.3, the average computation time for map generation was 3.07ms, measured from the time a new depth image is ready until the map update is complete. For motion generation, from the time a replan is requested following a map update until the reference trajectory for the controller is ready was measured to be 5.1ms on average. The reference trajectory includes the states from the current position to the last waypoint inside the frustum.

The first scenario, shown in Figure 5.10, is a 3D maze, where the quadcopter had to navigate from its initial position in one corner of the maze area to the opposite corner. This scenario is interesting since the robot had to navigate through narrow passages, around tight blind corners, and through windows at different heights. The total distance traveled, the flight time, and the average velocity in this experiment was 9.60m, 62.2s, and 0.15m/s, respectively.




(a) (b)

Figure 5.12: A Forest real-world experiment: (a) a picture of the robot in the forest scenario performing the experiment (b) the map representation generated together with the generated trajectory.




(a) (b)

Figure 5.13: An artificial forest experiment with walking humans: (a) a picture of the robot in the forest scenario performing the experiment (b) the map representation generated together with the generated trajectory.

The second scenario, shown in Figure 5.11, is an industrial warehouse. This scenario was chosen because it presents a potential real-world application for autonomous MAVs, e.g., capturing the inventory and locating out-of-place items. The total distance traveled, the flight time, and the average velocity in this experiment were 28.62m, 160.2s, and 0.17m/s, respectively.

The third scenario, shown in Figure 5.12, is an outdoors navigation in a forest environment, which was chosen because it represents an unstructured and natural scenario with irregular obstacles. As in all the other examples in this section, the drone was provided with a manually defined goal specified as an offset along each coordinate. The total distance traveled, the flight time, and the average velocity in this experiment was 11.58m, 57.5s, and 0.20m/s, respectively. It should be noted that only in this experiment, the tracking module T265 was pointing downward to ensure that enough features were captured.

Finally, the fourth scenario, shown in Figure 5.13 is a dynamic cluttered environment with people walking inside a lab area. This experiment exhibits the importance of keeping an up-to-date occupied space since, without this feature, the dynamic obstacles would create "virtual" walls blocking the paths to the goal. It is essential to highlight that, from the methods described in Table 1.1 only our framework has been demonstrated in a dynamic environment. The total distance traveled, the flight time, and the average velocity in this experiment was 4.81m, 26.4s, and 0.18m/s, respectively.

Some instants in the video show a delay in map creation. We have carefully measured the time for creating the map, and we can attribute this delay to visualization issues. We believe that depicted trajectories are faster to render than all the map voxels, creating the visualization delay.

# Chapter 6

# Conclusions

We proposed a complete framework that allows multiple robots to navigate in unknown environments autonomously. In more detail, the framework enables robots to plan and execute trajectories in cluttered environments while a map is simultaneously generated. Furthermore, the robots can use the map to escape pockets and reach target locations in complex 3D environments. The methodology used for the coordination of multiple robots allowed to save time and energy as it avoided the circumnavigation of robots to waste time; instead, a distance-dependent function was used to vary the speed of the robots achieving more intelligent coordination.

Our analytical and numerical results show that the trajectories generated by the framework are safe since they take into account both field of view restrictions as well as dynamical constraints of the robot. Moreover, we demonstrated a superior success rate compared with state-of-the-art achieving autonomous navigation in more challenging scenarios than those proposed so far.

The framework was implemented in a commercial multirotor MAV kit of less than one kilogram and was demonstrated to work real-time, at the camera frame rate of 33Hz, without a discrete GPU or a high-end CPU.

## 6.1 Discussions

Increasing average velocity is beneficial because it potentially reduces the flight time needed for completing a mission. However, in our case, increasing speed without changing the maximum acceleration requires increasing the safety margin $\varepsilon_w$. Consequently, it effectively reduces the space $\mathcal{V}^*_{\text{free}}[k]$ for path planning and may result in the MAV being unable to navigate in highly cluttered environments. In our current implementation, the value of $\varepsilon_w$ is fixed and needs to be manually specified. Making $\varepsilon_w$ adaptable based on the voxel discretization level in the current position of the MAV could allow it to move faster in wide-open spaces while maintaining safety when near obstacles.

Another way to increase speed would be to increase the maximum acceleration without changing $\varepsilon_w$ since it would allow following closely along the specified path even when it requires significant changes in direction. However, the maximum acceleration is limited by the physical characteristics of the platform. In addition, the perception and computation capabilities limit the reaction times and hence the maximum accelerations that can be safely applied. The problem in our case in terms of

perception was that sudden movements could result in more significant uncertainty on the visual odometry sensor due to loss of features and more significant errors on the measurements from the IMU and temporal misalignment between the signals due to latency. Moreover, large velocities also require increasing the field of view and especially the sensor range. But, increasing the sensor range is problematic, especially when using depth cameras, for two main reasons; first, the uncertainty increases quadratically with distance (Schauwecker, 2014), and second because it requires more computation resources to process the point cloud and the voxels inside $\mathcal{V}^*_{\text{free}}[k]$.

A key feature of our framework is the ability to generate dynamically feasible trajectories without solving computationally expensive optimization problems online. This feature allowed our experimental MAV to navigate safely in some highly cluttered environments while reacting to neighboring agents efficiently. Nevertheless, the method is over-conservative because it constrains the MAV to move at low velocities ( 0.2 m/s) even in open spaces far from obstacles. The reason is that the same restriction is applied for all the states in the system; this is, the error in the position contouring should correspond to the maximum velocity, which can be improved in two senses. On the one hand, a different selection of control gains can be chosen to impose a more suitable invariant set. On the other hand, the stability proof can be extended to use a different boundary per state. Note that this restriction was not a problem in the simulations where VIO sensors are unaffected by sudden movements. Moreover, the MAV responds immediately to control inputs without the latency of any kind.

In our current implementation, the map size is predetermined and fixed before the start of the mission. Thus, the map is limited by the available memory and the budget assigned to map updates in terms of computation. Increasing the size of the map dynamically while simple to implement would require adding levels in our linear octree hierarchy, which impacts the processing time for map updates and map queries, in particular for collision detection. Also, in our current implementation, reducing the size of the map, either by eliminating unoccupied voxels or by forgetting sections of the map, is not possible. Methods for compressing the map while maintaining topological information, for example, as suggested by Collins (2019) would need to be explored. Another approach to save memory could be to increase the radius of the voxel for the maximum discretization level $L$. However, this would reduce the space available for navigation.

When navigating in dynamic environments, adding semantic information into the map, which is impossible in our current implementation, would be helpful. The most straightforward approach would be distinguishing between static and dynamic objects, as done in dynamic occupancy grid maps used in autonomous driving. The semantic difference could help track dynamic objects and predict their trajectories in path planning to obtain more efficient trajectories and increase safety around people or not rely on the drone to drone communication for coordination. For instance, a timestamp may be added to the dynamic voxels, causing the probability of occupied decay as a function of time. Adding this feature could improve benchmarks results. For example, in some experiments with cluttered environments and a high density of agents, we observed that some spaces remained occupied when the neighbors moved because the sensor view never pointed to that region, avoiding updating the map and staying as occupied space in the drone memory map.

Another difficulty is that the final goal must be specified beforehand as an offset to the robot's starting position. In some cases, it could be blocked, for example, by a dynamic object, as in the experiment in Figure 5.13. In this case, the MAV would not move until the path is cleared or it runs out of battery. Having a strategy to abort the mission and return home would be helpful in such cases.

Our framework assumed a global frame for all the MAVs, which is not straightforward in practice since each robot has its reference frame provided by its VIO algorithm. However, we solved this by manually measuring the distances between the drones and initializing them with the same orientation as possible. A more elegant solution could be sharing their images and trying to compute online the transformation between them. Moreover, we considered only the local map of each agent to compute a trajectory to the goal. The naive solution of broadcasting the map to other agents is not adopted in this work. It is not a realistic solution due to the bandwidth of the MAVs. Instead, we propose as future work to use a coder-decoder approach to compress the map information. Finally, the hierarchy assignation remains an open problem in this work since we merely adopted the best heuristic for the scenarios we used. We propose using a different method as future work, for instance, a deep learning-based methodology, to obtain better performance in more generalized scenarios. Furthermore, this can be extended to designate the goals dynamically to reduce the conflicts between agents and improve desired metrics, such as arrival time.

As far as we know, the Intel Aero Ready to Fly was the only available open platform for drones under 1Kg with enough computing power for building an autonomous MAV. That is, the software and hardware can easily be modified. In our case, the original Yocto distribution was replaced with Ubuntu and ROS Kinetic for faster algorithm development. We also changed the original R200 RealSense camera with the more recent and better-performing D435 depth sensor. We also added a T265 tracking module with minimum effort, thanks to already available Linux drivers and removed the GPS module.

An issue with the Aero platform is that the onboard IMU is not mechanically isolated from the frame, and the motors added noise. A similar problem affected the tracking module. For that reason, we experimented with adding vibration-absorbing gel between the board and the frame to reduce the problem, achieving lower vibration levels. The same gel was also used to isolate the tracking module from the structure. It's important to note that during our early development on the Aero platform, we experimented with other VIO algorithms such as SVO, proposed by Forster, Zhang, Gassner, Werlberger, and Scaramuzza (2017) and VINS, proposed by Qin, Li, and Shen (2018) which were also affected by this problem. The T265 is a module that reduces computation on the central processor. It can easily be mechanically decoupled from the frame using damping material, making it a desirable sensor in MAVs. However, due to weight and power limitations in our Aero MAV platform, the maximum flight time was limited to less than 20 minutes.

The maximum velocity in our experiments was also low because of the reasons explained above. Therefore, drift from the VIO sensor was not a significant issue that we experienced. Nevertheless, we point out that in our experience, the T265 sensor used in our platform provides low closed-loop drift most of the time. In terms of sensors, another problem was the sensitivity of the depth camera to changes in illumination, requiring tedious manual adjustment of the camera configuration in each environment and making compromises between detection accuracy and density.

Overall we can say that the secret sauce which allowed us to achieve our goal of fully autonomous navigation in a compute-constrained MAV was the tight integration between the three main components of our framework, namely mapping, planning, and coordination between agents—for instance, bootstrapping the stochastic planner reusing information from the map updates and efficient collision detection leveraging the data structures judiciously.

# References


Bay, H., Tuytelaars, T., & Van Gool, L. (2006). Surf: Speeded up robust features. In *European conference on computer vision* (pp. 404–417).

Bonatti, R., Madaan, R., Vineet, V., Scherer, S., & Kapoor, A. (2020). Learning visuomotor policies for aerial navigation using cross-modal representations. In *2020 ieee/rsj international conference on intelligent robots and systems (iros)* (pp. 1637–1644).

Bonatti, R., Zhang, Y., Choudhury, S., Wang, W., & Scherer, S. (2020, 8). Autonomous Drone Cinematographer: Using Artistic Principles to Create Smooth, Safe, Occlusion-Free Trajectories for Aerial Filming. In *Proceedings of the 2018 international symposium on experimental robotics* (pp. 119–129). Retrieved from http://arxiv.org/abs/1808.09563

Cadena, C., Carlone, L., Carrillo, H., Latif, Y., Scaramuzza, D., Neira, J., ... Leonard, J. J. (2016, 12). Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, *32*(6), 1309–1332. Retrieved from http://ieeexplore.ieee.org/document/7747236/

Campos-Macías, L., Gómez-Gutiérrez, D., Aldana-López, R., de la Guardia, R., & Parra-Vilchis, J. I. (2017). A hybrid method for online trajectory planning of mobile robots in cluttered environments. *IEEE Robotics and Automation Letters*, *2*(2), 935-942.

Chen, Y., Cutler, M., & How, J. P. (2015). Decoupled multiagent path planning via incremental sequential convex programming. In *2015 ieee international conference on robotics and automation (icra)* (p. 5954-5961).

Cieslewski, T., Kaufmann, E., & Scaramuzza, D. (2017a, 9). Rapid exploration with multi-rotors: A frontier selection method for high speed flight. In *Ieee international conference on intelligent robots and systems* (Vol. 2017-Septe, pp. 2135–2142). IEEE. Retrieved from http://ieeexplore.ieee.org/document/8206030/

Cieslewski, T., Kaufmann, E., & Scaramuzza, D. (2017b, 9). Rapid exploration with multi-rotors: A frontier selection method for high speed flight. In *2017 ieee/rsj international conference on intelligent robots and systems (iros)* (pp. 2135–2142). IEEE. Retrieved from http://ieeexplore.ieee.org/document/8206030/

Collins, M. (2019). *Efficient Planning for High-Speed MAV Flight in Unknown Environments Using Sparse Topological Graphs* (Doctoral dissertation, Carnegie Mellon University). Retrieved from https://www.semanticscholar.org/paper/Efficient-Planning-for-High-Speed-MAV-Flight-in-Collins/e21044a3ccbb49a7fc1c52ff842510c13f526a1b

Florence, P. R., Carter, J., Ware, J., & Tedrake, R. (2018, 2). NanoMap: Fast, Uncertainty-Aware Proximity Queries with Lazy Search over Local 3D Data. *Proceedings - IEEE International Conference on Robotics and Automation*, 7631–7638. Retrieved from http://arxiv.org/abs/1802.09076

Forster, C., Zhang, Z., Gassner, M., Werlberger, M., & Scaramuzza, D. (2017).

SVO: Semidirect Visual Odometry for Monocular and Multicamera Systems. *IEEE Transactions on Robotics*, *33*(2), 249–265.

Fraundorfer, F., Heng, L., Honegger, D., Lee, G. H., Meier, L., Tanskanen, P., & Pollefeys, M. (2012, 10). Vision-based autonomous mapping and exploration using a quadrotor MAV. In *Ieee international conference on intelligent robots and systems* (pp. 4557–4564). IEEE. Retrieved from http://ieeexplore.ieee.org/document/6385934/

Frazzoli, E., Dahleh, M. A., & Feron, E. (2002). Real-time motion planning for agile autonomous vehicles. *Journal of guidance, control, and dynamics*, *25*(1), 116–129.

Furrer, F., Burri, M., Achtelik, M., & Siegwart, R. (2016). RotorS—A modular gazebo MAV simulator framework. In *Studies in computational intelligence* (Vol. 625, pp. 595–625). Springer, Cham. Retrieved from http://link.springer.com/10.1007/978-3-319-26054-9_23

Gao, F., Wu, W., Gao, W., & Shen, S. (2019, 12). Flying on point clouds: Online trajectory generation and autonomous navigation for quadrotors in cluttered environments. *Journal of Field Robotics*, *36*(4), 710–733. Retrieved from http://doi.wiley.com/10.1002/rob.21842

Gao, F., Wu, W., Lin, Y., & Shen, S. (2018, 5). Online Safe Trajectory Generation for Quadrotors Using Fast Marching Method and Bernstein Basis Polynomial. In *Proceedings - ieee international conference on robotics and automation* (pp. 344–351). IEEE. Retrieved from https://ieeexplore.ieee.org/document/8462878/

Gargantini, I. (1982, 12). Linear octrees for fast processing of three-dimensional objects. *Computer Graphics and Image Processing*, *20*(4), 365–374. Retrieved from https://www.sciencedirect.com/science/article/pii/0146664X82900582

Han, L., Gao, F., Zhou, B., & Shen, S. (2019, 3). FIESTA: Fast Incremental Euclidean Distance Fields for Online Motion Planning of Aerial Robots. *IEEE International Conference on Intelligent Robots and Systems*, 4423–4430. Retrieved from http://arxiv.org/abs/1903.02144

Hornung, A., Wurm, K. M., Bennewitz, M., Stachniss, C., & Burgard, W. (2013). Octomap: An efficient probabilistic 3d mapping framework based on octrees. *Autonomous robots*, *34*(3), 189–206.

Jordan, S., Moore, J., Hovet, S., Box, J., Perry, J., Kirsche, K., ... Tse, Z. T. H. (2018, 2). State-of-the-art technologies for UAV inspections. *IET Radar, Sonar and Navigation*, *12*(2), 151–164. Retrieved from https://digital-library.theiet.org/content/journals/10.1049/iet-rsn.2017.0251

Karaman, S., & Frazzoli, E. (2011, 6). Sampling-based algorithms for optimal motion planning. *International Journal of Robotics Research*, *30*(7), 846–894. Retrieved from http://journals.sagepub.com/doi/10.1177/0278364911406761

Karaman, S., & Frazzoli, E. (2012, 5). High-speed flight in an ergodic forest. In *Proceedings - ieee international conference on robotics and automation* (pp. 2899–2906). IEEE. Retrieved from http://ieeexplore.ieee.org/document/6225235/

Kaufmann, E., Gehrig, M., Foehn, P., Ranftl, R., Dosovitskiy, A., Koltun, V., & Scaramuzza, D. (2019, 5). Beauty and the beast: Optimal methods meet learning for drone racing. In *Proceedings - ieee international conference on robotics and automation* (Vol. 2019-May, pp. 690–696). Institute of Electrical and Electronics Engineers Inc.

Kaufmann, E., Loquercio, A., Ranftl, R., Dosovitskiy, A., Koltun, V., & Scaramuzza, D. (2018). Deep drone racing: Learning agile flight in dynamic envi-

ronments. In *Conference on robot learning* (pp. 133–145).

Kavraki, L. E., Svestka, P., Latombe, J.-C., & Overmars, M. H. (1996). Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, *12*(4), 566–580.

Khalil, H. K. (2002). Nonlinear systems third edition. *Patience Hall*.

Klingensmith, M., Dryanovski, I., Srinivasa, S. S., & Xiao, J. (2015, 7). CHISEL: Real time large scale 3D reconstruction onboard a mobile device using spatially-hashed signed distance fields. In *Robotics: Science and systems* (Vol. 11). Robotics: Science and Systems Foundation. Retrieved from http://www.roboticsproceedings.org/rss11/p40.pdf

Kuffner, J. J., & LaValle, S. M. (2000). Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 icra. millennium conference. ieee international conference on robotics and automation. symposia proceedings (cat. no. 00ch37065)* (Vol. 2, pp. 995–1001).

Kweon, I.-S., Hebert, M., Krotkov, E., & Kanade, T. (1989). Terrain mapping for a roving planetary explorer. In *Ieee international conference on robotics and automation* (pp. 997–1002).

Kwon, Y., Kim, D., An, I., & Yoon, S. E. (2019). Super Rays and Culling Region for Real-Time Updates on Grid-Based Occupancy Maps. *IEEE Transactions on Robotics*, *35*(2), 482–497. Retrieved from https://ieeexplore.ieee.org/document/8625443/

Li, G., Müller, M., Casser, V., Smith, N., Michels, D. L., & Ghanem, B. (2018). Oil: Observational imitation learning. *arXiv preprint arXiv:1803.01129*.

Lin, Y., Gao, F., Qin, T., Gao, W., Liu, T., Wu, W., ... Shen, S. (2018, 1). Autonomous aerial navigation using monocular visual-inertial fusion. *Journal of Field Robotics*, *35*(1), 23–51. Retrieved from http://doi.wiley.com/10.1002/rob.21732

Liniger, A., Domahidi, A., & Morari, M. (2015, 9). Optimization-based autonomous racing of 1:43 scale RC cars. *Optimal Control Applications and Methods*, *36*(5), 628–647. Retrieved from http://doi.wiley.com/10.1002/oca.2123

Loquercio, A., Maqueda, A. I., Del-Blanco, C. R., & Scaramuzza, D. (2018, 4). DroNet: Learning to Fly by Driving. *IEEE Robotics and Automation Letters*, *3*(2), 1088–1095.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, *60*(2), 91–110.

Luis, C. E., & Schoellig, A. P. (2019). Trajectory generation for multiagent point-to-point transitions via distributed model predictive control. *IEEE Robotics and Automation Letters*, *4*(2), 375–382.

Mellinger, D., & Kumar, V. (2011). Minimum snap trajectory generation and control for quadrotors. *Proceedings - IEEE International Conference on Robotics and Automation*, 2520–2525.

Mohta, K., Watterson, M., Mulgaonkar, Y., Liu, S., Qu, C., Makineni, A., ... Kumar, V. (2018, 1). Fast, autonomous flight in GPS-denied and cluttered environments. *Journal of Field Robotics*, *35*(1), 101–120. Retrieved from http://doi.wiley.com/10.1002/rob.21774

Nießner, M., Zollhöfer, M., Izadi, S., & Stamminger, M. (2013, 11). Real-time 3D reconstruction at scale using voxel hashing. *ACM Transactions on Graphics*, *32*(6), 1–11. Retrieved from http://dl.acm.org/citation.cfm?doid=2508363.2508374

Oleynikova, H., Taylor, Z., Fehr, M., Siegwart, R., & Nieto, J. (2017, 9). Voxblox: Incremental 3D Euclidean Signed Distance Fields for on-board MAV planning. In *Ieee international conference on intelligent robots and systems* (Vol. 2017-Septe, pp. 1366–1373). IEEE. Retrieved from http://ieeexplore.ieee

.org/document/8202315/

Oleynikova, H., Taylor, Z., Millane, A., Siegwart, R., & Nieto, J. (2018, 12). A Complete System for Vision-Based Micro-Aerial Vehicle Mapping, Planning, and Flight in Cluttered Environments. *arxiv*. Retrieved from https://arxiv.org/abs/1812.03892

Oleynikova, H., Taylor, Z., Siegwart, R., & Nieto, J. (2018). Safe Local Exploration for Replanning in Cluttered Unknown Environments for Microaerial Vehicles. *IEEE Robotics and Automation Letters*, *3*(3), 1474–1481. Retrieved from http://arxiv.org/abs/1710.00604

Papachristos, C., Khattak, S., & Alexis, K. (2017). Uncertainty-aware receding horizon exploration and mapping using aerial robots. In *Proceedings - ieee international conference on robotics and automation* (pp. 4568–4575).

Poggi, M., Tosi, F., Batsos, K., Mordohai, P., & Mattoccia, S. (2020). On the synergies between machine learning and stereo: a survey. *arXiv preprint arXiv:2004.08566*.

Pohl, D., Dorodnicov, S., & Achtelik, M. (2019, 9). Depth map improvements for stereo-based depth cameras on drones. In *Proceedings of the 2019 federated conference on computer science and information systems, fedcsis 2019* (pp. 341–348). Institute of Electrical and Electronics Engineers Inc.

Qin, T., Li, P., & Shen, S. (2018). VINS-Mono: A Robust and Versatile Monocular Visual-Inertial State Estimator. *IEEE Transactions on Robotics*, *34*(4), 1004–1020.

Quigley, M., Gerkey, B., Conley, K., Faust, J., Foote, T., Leibs, J., ... Ng, A. (2009). ROS: an open-source Robot Operating System. In *Proc. of the ieee intl. conf. on robotics and automation (icra) workshop on open source robotics.*

Riviere, B., Hönig, W., Yue, Y., & Chung, S.-J. (2020). Glas: Global-to-local safe autonomy synthesis for multi-robot motion planning with end-to-end learning. *IEEE Robotics and Automation Letters*, *5*(3), 4249–4256.

Rosten, E., & Drummond, T. (2006). Machine learning for high-speed corner detection. In *European conference on computer vision* (pp. 430–443).

Scaramuzza, D., & Fraundorfer, F. (2011). Visual odometry [tutorial]. *IEEE robotics & automation magazine*, *18*(4), 80–92.

Schauwecker, K. (2014). *Stereo Vision for Autonomous Micro Aerial Vehicles* (Tech. Rep.). Retrieved from https://publikationen.uni-tuebingen.de/xmlui/handle/10900/55173

Schauwecker, K., & Zell, A. (2014, 5). Robust and efficient volumetric occupancy mapping with an application to stereo vision. In *Proceedings - ieee international conference on robotics and automation* (pp. 6102–6107). IEEE. Retrieved from http://ieeexplore.ieee.org/document/6907758/

Schwartz, J. T., & Sharir, M. (1983). On the "piano movers'" problem i. the case of a two-dimensional rigid polygonal body moving amidst polygonal barriers. *Communications on pure and applied mathematics*, *36*(3), 345–398.

Selin, M., Tiger, M., Duberg, D., Heintz, F., & Jensfelt, P. (2019, 4). Efficient autonomous exploration planning of large-scale 3-D environments. *IEEE Robotics and Automation Letters*, *4*(2), 1699–1706. Retrieved from https://ieeexplore.ieee.org/document/8633925/

Serra-Gómez, A., Brito, B., Zhu, H., Chung, J. J., & Alonso-Mora, J. (2020). With whom to communicate: learning efficient communication for multi-robot collision avoidance. In *2020 ieee/rsj international conference on intelligent robots and systems (iros)* (pp. 11770–11776).

Shen, S., Michael, N., & Kumar, V. (2011, 5). Autonomous multi-floor indoor navigation with a computationally constrained MAV. In *Proceedings - ieee international conference on robotics and automation* (pp. 20–25). IEEE. Retrieved from http://ieeexplore.ieee.org/document/5980357/

Shome, R., Solovey, K., Dobson, A., Halperin, D., & Bekris, K. E. (2020). drrt*: Scalable and informed asymptotically-optimal multi-robot motion planning. *Autonomous Robots*, *44*(3), 443–467.

Shu, R., & Kankanhalli, M. S. (1994). Efficient linear octree generation from voxels. *Image and vision computing*, *12*(5), 297–303.

Siegwart, R., Nourbakhsh, I. R., & Scaramuzza, D. (2011). *Introduction to autonomous mobile robots*. MIT press.

Snape, J., Van Den Berg, J., Guy, S. J., & Manocha, D. (2011). The hybrid reciprocal velocity obstacle. *IEEE Transactions on Robotics*, *27*(4), 696–706.

Szeliski, R. (2010). *Computer vision: algorithms and applications*. Springer Science & Business Media.

Tedrake, R., Manchester, I. R., Tobenkin, M., & Roberts, J. W. (2010, 7). LQR-trees: Feedback motion planning via sums-of-squares verification. *International Journal of Robotics Research*, *29*(8), 1038–1052. Retrieved from http://journals.sagepub.com/doi/10.1177/0278364910369189

Thrun, S., Burgard, W., & Fox, D. (2006). *Probabilistic robotics*. Massachusetts Institute of Technology, USA. Retrieved from https://mitpress.mit.edu/books/probabilistic-robotics

Triebel, R., Pfaff, P., & Burgard, W. (2006). Multi-level surface maps for outdoor terrain mapping and loop closing. In *2006 ieee/rsj international conference on intelligent robots and systems* (pp. 2276–2282).

Triggs, B., McLauchlan, P. F., Hartley, R. I., & Fitzgibbon, A. W. (1999). Bundle adjustment—a modern synthesis. In *International workshop on vision algorithms* (pp. 298–372).

Tsardoulias, E. G., Iliakopoulou, A., Kargakos, A., & Petrou, L. (2016, 12). A Review of Global Path Planning Methods for Occupancy Grid Maps Regardless of Obstacle Density. *Journal of Intelligent and Robotic Systems: Theory and Applications*, *84*(1-4), 829–858. Retrieved from http://link.springer.com/10.1007/s10846-016-0362-z

Usenko, V., Von Stumberg, L., Pangercic, A., & Cremers, D. (2017, 3). Real-time trajectory replanning for MAVs using uniform B-splines and a 3D circular buffer. In *Ieee international conference on intelligent robots and systems* (Vol. 2017-Septe, pp. 215–222). Retrieved from http://arxiv.org/abs/1703.01416

Van Den Berg, J., Guy, S. J., Lin, M., & Manocha, D. (2011). Reciprocal n-body collision avoidance. In *Robotics research* (pp. 3–19). Springer.

Vespa, E., Nikolov, N., Grimm, M., Nardi, L., Kelly, P. H., & Leutenegger, S. (2018, 4). Efficient Octree-Based Volumetric SLAM Supporting Signed-Distance and Occupancy Mapping. *IEEE Robotics and Automation Letters*, *3*(2), 1144–1151. Retrieved from http://ieeexplore.ieee.org/document/8255617/

Webb, D. J., & van den Berg, J. (2013). Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics. In *2013 ieee international conference on robotics and automation* (p. 5054-5061).

Zhou, X., Wang, Z., Wen, X., Zhu, J., Xu, C., & Gao, F. (2021). Decentralized spatial-temporal trajectory planning for multicopter swarms. *arXiv preprint arXiv:2106.12481*.

Zhou, X., Wen, X., Zhu, J., Zhou, H., Xu, C., & Gao, F. (2021). Ego-swarm: A fully autonomous and decentralized quadrotor swarm system in cluttered environments. *arXiv preprint arXiv:2011.04183*.