POLITECNICO DI TORINO
Repository ISTITUZIONALE

Autonomous Navigation for Unmanned Aerial Systems - Visual Perception and Motion Planning

Original
Autonomous Navigation for Unmanned Aerial Systems - Visual Perception and Motion Planning / Osman, OSMAN ABDALLA SIDAHMED. - (2022 Sep 01), pp. 1-166.

Availability:
This version is available at: 11583/2971114 since: 2022-09-08T15:20:32Z

Publisher:
Politecnico di Torino

Published
DOI:

Terms of use:
openAccess

This article is made available under terms and conditions as specified in the corresponding bibliographic description in the repository

Publisher copyright

(Article begins on next page)

18 October 2022

Doctoral Dissertation
Doctoral Program in Electronics and Telecommunications (DET) (33^{th}cycle)

Autonomous Navigation for Unmanned Aerial Systems

Visual Perception and Motion Planning

By

Osman Abdalla Sidahmed Osman

******

Supervisor(s):
Prof. Alessandro Rizzo, Supervisor
Prof. Marcello Chiaberge, Co-Supervisor

Doctoral Examination Committee:
Prof. Elisa Capello
Prof. Marina Indri
Prof. Lucia Pallottino
Prof. Domenico Prattichizzo (Reviewer)
Prof. Kimon Valavanis (Reviewer)

Politecnico di Torino
2022

Declaration

I hereby declare that, the contents and organization of this dissertation constitute my own original work and does not compromise in any way the rights of third parties, including those relating to the security of personal data.

Osman Abdalla Sidahmed Osman
2022

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the Graduate School of Politecnico di Torino (ScuDo).

Acknowledgements

Foremost, I would like to express my sincere gratitude to Prof. Alessandro Rizzo and Prof. Marcello Chiaberge, my supervisors, for their continuous support, patient guidance and motivation. Their guidance and critical judgment was keeping the thesis schedule on track and on target.
I am particularly grateful to Dr. Stefano Primatesta for his valuable technical support and their help in conducting the research. His experience and constructive recommendations made it possible to cover a wide range of issues and applications. My grateful thanks also goes to the the colleagues at PIC4SeR and the Complex Systems Laboratory for the support and friendly environment. It made the research and the PhD period both joyful and rich.
I would like also to express my heartfelt thanks to my family and my friends whom were the eminent source of motivation through these studies.

Abstract

Unmanned Aerial Systems (UAS) have attracted a great deal of attention in recent years. UAS Autonomous Navigation is often split into four main tasks: Perception, Localization, Motion Planning and Motion Control.

This Ph.D. dissertation focuses on two of those tasks: Perception and Motion Planning, specifically in the context of the UAS.

The first part of the thesis focuses on perception. We investigate the design characteristics that influence the quality of the 3D environment modelling generated by a vision system on-board a UAS, including the camera design and configuration in addition to the flight plan parameters (speed and altitude). The 3D environment model richness and accuracy were used as quality indicators. We present design scheme highlighting a method to navigate the trade-offs during the design phase, grouping the design factors into requirement parameters, selection parameters and configuration, and we further map the design factors' inter-dependencies.

Moreover, we further analyse the effect of geo-tags uncertainty on the quality of the 3D environment model. ASONY ILCE-QX1L camera with 20.1MP was utilized on-board a UAS capturing images while descending and ascending in altitude range of 60 - 20 m. We evaluate the use of two different GPS data sets as initials for the camera extrinsic parameters in order to determine their accuracy. The design trade-offs between the various camera parameters and the flight plan under specific requirements like the final object resolution, the UAS speed and the required images' overlap is evaluated. The inter-dependencies and relations between these parameters and the number of quality matches are mapped and a structured computation and trade-off workflow was recommended for terrestrial mapping applications.

Finally, an industrial solution for a challenging object recognition, localization and assembly supervision is implemented and evaluated. Using an FPGA programmable industrial camera, the developed solution enables the detection and localization of the target parts, and supervision over the of the parts' feeding, conveying and heuristic orientation manipulation system. The vision system is further integrated with a FANUC industrial robotic arm, which is used to perform the pick-and place operations. The final solution is successfully implemented in a manufacturing facility resulting in processing time of less than 200ms between image capture and the declaration of the found parts, while complying with the reliability, robustness and processing time defined by the application. The solution developed can be useful for UAS perception, it enables fast object detection and localization with minimal computational cost using images processing techniques.

The second part of the thesis focuses on Motion Planning. We develop a novel kinodynamic sampling-based motion planning algorithm called MP-RRT$^{\#}$, which builds on the existing RRT$^{\#}$ by augmenting it with a Model Predictive Control method used to compute the optimal trajectory for UAS. The use of the MPC ensures the feasibility and applicability of the resulting trajectory, as both the obstacles constraints (which restrict the feasible states to the free space) and the vehicle constraints (which limit the control input constraints) are taken into consideration by the MPC during the design process. Similar to other RRT-based algorithms, MP-RRT$^{\#}$ explores the map constructing an asymptotically optimal graph. In each iteration the graph is extended with a new vertex in the reference state of the UAS. Then, a forward simulation is performed using a Model Predictive Control strategy to evaluate the motion between two adjacent vertices, and a trajectory in the state space is computed. As a result, the MP-RRT$^{\#}$ algorithm eventually generates a feasible trajectory for the UAS satisfying dynamic constraints. Simulation results obtained with a simulated drone controlled with the PX4 autopilot corroborate the validity of the MP-RRT$^{\#}$ approach.

Contents

List of Figures **ix**

List of Tables **xii**

1 Introduction **1**

1.1 Trends in Robotics 1

1.2 Autonomous Navigation 5

1.2.1 Perception and Localization 7

1.2.2 Motion Planning and Control 10

1.3 Objective 15

1.4 Outline 16

I Perception **17**

2 3D Environment Modelling **19**

2.1 Introduction to Visual Perception 21

2.2 UAS Vision System for 3D Environment Modelling 24

3 Visual Perception for Mobile Robots **27**

3.1 Visual Perception Design Process 27

3.1.1 Operational parameters' configuration 29

3.2 Use of Additional Data Sets 29
3.2.1 Quality Indicators 30
3.2.2 Results and Discussion 31
3.2.3 Conclusion 33
3.3 Industrial Visual Perception 34
3.3.1 Industrial Visual Perception Solution 35
3.3.2 Calibration 44
3.3.3 Robot Interaction 45
3.3.4 Robustness Evaluation 47
3.3.5 Feeder Control 49
3.3.6 Remarks 51

II Motion Planning 52

4 Motion Planning State-of-Art 54

4.1 Background 54
4.2 Methods for Motion Planning 57
4.2.1 Roadmaps 59
4.2.2 Heuristic Methods 63
4.2.3 Evolution Methods 67
4.2.4 Sampling-Based Methods 69
4.3 RRT-based Algorithms 79
4.3.1 Problem Formulation 79
4.3.2 Basic RRT Algorithm 80
4.3.3 RRT* Algorithm 81
4.3.4 RRT# Algorithm 86
4.4 Motion Planning Constraints 88

5 The MP-RRT# Algorithm **92**

5.1 Problem Definition 92

5.1.1 UAS Model 92

5.1.2 UAS Model Lineraization and Discretization 97

5.1.3 Problem Statement 99

5.2 The MP-RRT# strategy 100

5.2.1 Model Predictive Control 106

5.3 Technological Tools 110

5.3.1 OMPL 110

6 Experiment and Results **114**

6.1 Experiment 114

6.1.1 MPC Optimization Object 115

6.2 Results 122

6.2.1 Implementation 122

6.2.2 Simulation results 124

7 Discussion and Conclusion **133**

7.1 Conclusions 133

7.1.1 Perception 133

7.1.2 Motion Planning 137

References **139**

List of Figures

1.1 Generic Robots Classification 3

2.1 Structure from Motion Workflow 24

2.2 Vision System Design parameters inter-dependencies and the way they affect the final 3D reconstruction through affecting the number of quality matched features 26

3.1 (a) Focal length deviation from the nominal value, the two data sets with the same large uncertainty bounds (standard) gives the same calibration results which is also equal to the calibration without any additional data. (b) Distribution of the residual of the estimated camera positions deviation from the initial data 32

3.2 (a) Number of 3D points reconstructed with each additional data set as a measure of the richness of the resulting 3D model. (b) Deviation of the Ground Control Point in each 3D reconstructed model as a measure of the accuracy of the environment model. 32

3.3 Topics investigated and their place in the SfM framework . . . 34

3.4 Algorithm Top Level Structure. 38

3.5 Image processing: binary filter and noise cancellation 39

3.6 Full/Limited search Decision Flowchart 40

3.7 Saving/Latching ready parts' list 41

3.8 Full Search 42
3.9 Limited Search 42
3.10 Camera-Robot Calibration 45
3.11 In/OUT FOV monitoring 46
3.12 Robustness Evaluation 48
3.13 Feeder control logic flowchart. 50

5.1 Example of graphs constructed with MP-RRT#. The graph $\mathcal{G}^{\mathcal{Y}}$ consists of vertices (in black) and edges (in blue) in the reference state. Instead, the graph $\mathcal{G}^{\mathcal{X}}$ consists of trajectories (in magenta) obtained through evaluating the edges of $\mathcal{G}^{\mathcal{Y}}$ using the MPC strategy. An edge of $\mathcal{G}^{\mathcal{Y}}$ is labeled as invalid if its corresponding trajectory in $\mathcal{G}^{\mathcal{X}}$ crosses an obstacle. 105
5.2 Example of reference trajectory computed using Dubins curves and connecting two adjacent vertices. The green line is the reference trajectory, whereas magenta arrows are the state trajectory computed using MPC. 109
5.3 Open Motion Planning Library (OMPL) object-oriented structure 113

6.1 The roll and pitch control inputs computed by MPC to follow the trajectory of Figure 5.2. 125
6.2 The construction of the exploration tree using the MP-RRT# algorithm. The start and target positions are in green and in red, respectively. The graph $\mathcal{G}^{\mathcal{Y}}$ in the reference space is colored in blue, while the computed path obtained from the graph $\mathcal{G}^{\mathcal{X}}$ in the state space is colored in magenta. In (a), the graph consists of 10 vertices rooted from the start pose finding an initial solution in the map with a cost (i.e. the path length) of 66.44 m. In (b), the graph with 20 vertices, in which the solution is improved with a cost of 45.09 m. In (c), the graph consists of 60 vertices, but the solution is not improved. In (d), the graph has 100 vertices obtaining a solution with cost 38.70 m. 126

6.3 The average cost of the solution path against the number of vertices in the MP-RRT# algorithm. The average cost is computed running the algorithm 50 times in the same scenario of Figure 6.2. 127

6.4 The average tracking error for 20 trajectories running the same scenario of Figure 6.2. 129

6.5 Trajectories computed with the MP-RRT# by constructing a graph with 400 vertices. The start and target positions are in green and in red, respectively. 130

6.6 Example of trajectory computed with the MP-RRT# by constructing a graph with 400 vertices. The start and target positions are in green and in red, respectively. 131

6.7 In (a), the trajectory computed with the MP-RRT# algorithm by constructing a graph of 100 vertices. In (b) the computed trajectory is executed by the PX4 autopilot in a simulation. . . 132

7.1 Vision System Design parameters inter-dependencies and the way they affect the final 3D reconstruction through affecting the number of quality matched features 135

7.2 Topics investigated and their place in the SfM framework . . . 136

List of Tables

3.1 Typical Camera and Flight Design Flow 28

6.1 Parameters used for the UAS model. 123

6.2 Trajectory tracking performance indices collected over 20 trajectories. 128

List of Algorithms

1 RRT 81
2 RRT* 84
3 Parent 84
4 Rewire 85
5 Body of the RRT# Algorithm 89
6 Extend Procedure 89
7 Replan Procedure 90
8 The MP-RRT# algorithm 102
9 The Extend procedure 103
10 The FindParent procedure 104
11 The Replan procedure 105

Listings

6.1 initializeParameters() method 117

6.2 Computation of the integral of the system matrices through an incremental approach 118

6.3 Cost computation using the linear approximation of the Euclidean Distance 121

6.4 2D linear distance computation 122

Chapter 1

Introduction

1.1 Trends in Robotics

The advancement of robotic technology has occurred throughout the years since its inception, and robots are now an essential component of a wide range of industries and businesses across the world. Automotive manufacturers, appliance makers, food and beverage processing, distribution and warehousing, healthcare, law enforcement, and security services are just a few examples of organizations that sell, operate, and develop robots.

The adoption of modern day robots can be witnessed in many operations from weed control in agriculture [1] to securing vacant properties [2] to manufacturing vehicles [3], to state a few operations among others. Notably, the use of industrial robots is becoming ubiquitous in a wide range of applications, including material handling, welding, assembly of parts and products, spray painting and dispensing of coatings, and packaging of materials and commodities. Whereas in sectors like nursing homes, hospitals, and hospitality, some robots are working alongside humans [4]. The complexity of some applications, on the other hand, has necessitated the development of new technologies that substantially improve the interaction between robots and humans even more. For example, [5] showed that robotic surgeries can assist in complex procedures while incurring less risk than human surgeons.
It is expected that these advancements in robotics will only continue to grow

in the next decade. In recognition of the way that our population is becoming older and that the number of wage workers is becoming a smaller percentage of our overall population, it is evident that robots will be expected to fill the void in our society in the near future. Robots in the industrial sector, and to a greater extent, robots in the service sector, have the potential to bridge this gap in the years ahead.

In order to fully understand the robots' capabilities that are both defining the challenges and opportunities in this growth trend, it is necessary to categorize the robots in a logical manner. A Robot, as defined by ISO 8373 [6], is an *"actuated mechanism, programmable in two or more axes, with a degree of autonomy, moving within its environment, to perform intended tasks"*. This broad definition reflects the reality that robots have grown to include a diverse range of technologies that are being used in an ever-expanding number of applications. However, robots are often classified into: industrial robots and service robots, which are distinguished by the tasks they perform and the market requirements for which they are developed. For their part, in research and development, robots are principally differentiated based on the capabilities they possess as well as the environment in which they work. Figure 1.1 illustrates such a classification, which can be described as follows:

1. *Stationary robots:* The vast majority of industrial robotic manipulators have a permanent base and operate in environments that have been specifically designed for robots to execute particular repetitive tasks in a consistent manner. Typically, they are capable of handling objects or carry out activities such as welding, painting, assembling, and machining, just to highlight a few examples. As sensors and human-robot interface technologies advance, these stationary robots are increasingly being deployed in less controlled environments, such as high-precision surgery, to assist surgeons in performing precise operations.

2. *Mobile robots:* They must be able to explore and carry out tasks in broad, ill-defined, and unknown environments which are not specifically designed for robots and may contain unforeseen moving or static entities.

For performing a task, mobile robots must be capable of gathering information from their environment, perceiving the information gathered, and taking actions (including their own mobility) to complete the task. We can further divide mobile robots into two types:

- *Remote-controlled robots:* They operate exclusively under the direct supervision of an operator. That makes them less demanding in terms of perception and intelligence.
- *Autonomous robots (Unmanned Systems):* They perform the tasks on their own, without the assistance of external human operators, in an environment that is unknown, partly known, or constantly changing. They can be distinguished from other robots by their ability to move autonomously, with enough intelligence to react and make decisions based on the perception they receive from the environment. Autonomous mobile robots (also known as Unmanned Systems) can be further categorised by the environment in which they operate:
  - Air – aerial robots are usually called Unmanned Aerial Vehicles (UAVs);
  - Water – underwater robots are referred to as Autonomous Underwater Vehicles (AUVs);
  - Land – waling, rolling, climbing robots; Unmanned Ground Vehicles (UGVs).

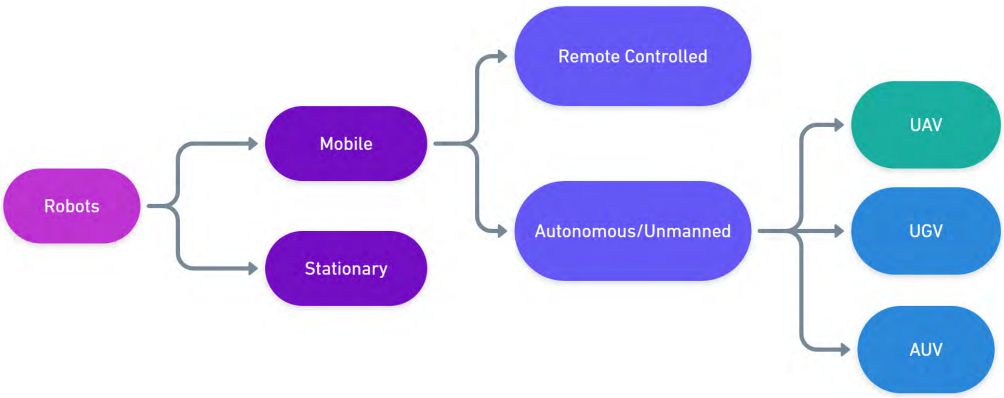


Fig. 1.1 Generic Robots Classification

The previous categorization is often used in research and development because it helps to differentiate between the technological requirements of the many robotic applications that are being studied. As a matter of fact, stationary robots are usually mounted to a permanent base on the ground, allowing them to infer their position using their inner state representation, while mobile robots must infer their location dynamically from sensory input.
A comprehensive framework for mobile robots is also necessary in order to coordinate all of the subsystems of sensing and perception, motion planning, and control [7].
Mobile robotics is now one of the most rapidly growing areas of scientific study, and it is anticipated to continue to grow in the future, propelling the whole field of robotics into new markets and applications as a result.
Mobile robots have the potential to replace humans in a wide range of fields, including surveillance, planetary exploration, emergency rescue operations, reconnaissance, industrial automation, construction, museum guides, personal services, transportation, and medical care, among a variety of other industrial and nonindustrial applications.

Throughout the robotics community, Unmanned Aerial Systems (UASs) have attracted a great deal of attention in recent years. Photographic and cinematographic operations, precision agriculture, power line and structural inspection, surveying, infrared and multi-spectral imaging as well as natural disaster recovery are just a few of the applications for the Unmanned Aerial Systems that are becoming more commonplace.
As noted by [8], aerial robots in most of their applications are expected to have some degree of autonomy in order to self-navigate in challenging environments where human operators may be at danger or unable to complete the job at hand. Fortunately, in recent decades, advances in computer vision, artificial intelligence, and micro electro-mechanical sensors (MEMs) have made the control, estimation, and perception of motion all more dependable and applicable. As a result, it has become possible to develop a reliable level of autonomy matching the required robustness and effectiveness of motion planning in real-world scenarios. A growing number of academics are starting to focus on higher-level tasks such as navigation and motion planning in unmanned aircraft systems (UAS) as a consequence of this trend, according

to [9]. However, UAS have demonstrated extra challenges in developing autonomous navigation skills when compared to other autonomous vehicles on land or in water. The particular nature of UAS as resource-constrained systems in terms of energy, computational capacity, and payload constraints, as well as the limited hardware-set available onboard, poses many challenges for autonomous skills' development.
In particular, unmanned aerial systems have a variety of challenging attributes in common, including non-trivial dynamics, three-dimensional environments, shocks, and uncertainty in state information. These special characteristics of UAS make it essential to develop autonomous solutions and motion planning methods that adhere to more rigorous standards as compared to the requirements for other kinds of mobile robotic systems.

1.2 Autonomous Navigation

In the area of robotics, some of the most challenging open research problems are those involving autonomous navigation. Research on mobile robot navigation systems, in general, involves developing appropriate systems for gathering adequate sensory information, using that information to plan the robot's motion along a suitable trajectory, and tracking the planned trajectory safely within its environment without colliding with anything. However, Autonomous Navigation development is often segmented into four building blocks: perception, localization, planning, and motion control.

1. *Perception* is the set of skills that enables the mobile robot to acquire knowledge about its work environment and itself. Hence, the key components of a perception system are essentially sensory data processing and data representation (environment modeling). This is generally achieved through a set of comprehensive sensors that acquire information from the robot's environment, and subsequently implementing various sensor fusion techniques in order to construct an environment model that describes the robot surroundings. The environment model must fulfill the navigational requirements of the intended application in terms of the model richness, accuracy and update rate. All those

requirements reflects on further requirements on the type of sensor set used and algorithmic solutions employed to compute the environment model.

2. *Localization* refers to the ability of the robot to determine its position with respect to the environment. This skill is tightly connected to the perception and environment modelling skill previously described. However, in addition to the exteroceptive sensory information, localization makes use of various interoceptive sensors such as IMU's and odometery.

3. *Planning* refers to the ability of the robot to decide how to act to achieve its goals. This skill relies more than the rest on what can be define as "intelligence" in general terms; i.e. the ability to reason an optimal decision given limited knowledge about the environment and the robot internal state. However, for autonomous navigation this takes a more specific definition since the goal to achieve is to navigate through the environment respecting some defined criteria such as obstacle avoidance, optimizing the energy or distance or controls or time to reach destination or any other application-specific criterion. Hence, planning in autonomous navigation refers to the process of making purposeful decisions in order to bring the vehicle from a start location to a goal location while avoiding obstacles and optimizing over predefined performance criteria.

4. *Motion Control* refers to the robot's ability to execute the planned navigational strategy or actions that have been generated in the planning stage. Motion Control is tightly coupled with Motion Planning and their specific interaction/integration is governed by the nature of the application. In applications where the environment is dynamic, motion planning and control are tightly coupled in a cyclic manner in which the motion control feeds back some knowledge to the planning of the next processing cycle. Whereas in the applications where the robot is navigating through a static environment, motion planning can be

highly independent from the motion control. And between those two scenarios various degrees of environment characteristics and application requirements defines different degrees of integration between the motion planning and motion control.

1.2.1 Perception and Localization

Robotic perception is related to many applications in robotics such as object detection, environment modelling, scene understanding, human/pedestrian detection, activity recognition, semantic classification, object modeling, among others. The fusion of vision sensors with IMU's and odometery data to provide a better robotic localization has led to interesting breakthrough in obstacle avoidance and autonomous navigation applications. However, as pointed out recently by Sünderhauf et al. [10], robotic perception (also designated robotic vision in [10]) differs from traditional computer vision perception in the sense that, in robotics, the outputs of a perception system will result in decisions and actions in the real world. Therefore, perception is a very important part of a complex, embodied, active, and goal-driven robotic system. As exemplified by Sünderhauf et al. [10], robotic perception has to translate images (or scans, or point-clouds) into actions, whereas most computer vision applications take images and translate the outputs into information. Hence, in particular, Autonomous Navigation requires both a precise and robust environment perception and localization. Representing the environment model with a map, the robot will be able to detect free spaces, obstacles, and different semantics and signs which will all be the bases on which the motion planning will be performed.

Among the numerous approaches used in environment modelling for mobile robotics, and for autonomous robotic-vehicles, the most influential approach is the occupancy grid mapping [11]. This 2D mapping is still used in many mobile platforms due to its efficiency, probabilistic framework, and fast implementation. Although many approaches use 2D-based representations to model the real world, presently 2.5D and 3D representation models are becoming more common. The main reasons for using higher dimensional representations are essentially twofold: (1) robots are demanded to navigate

and make decisions in higher complex environments where 2D representations are insufficient; This is particularly true for UAS. (2) current 3D sensor technologies are affordable and reliable, and therefore 3D environment representations became attainable.

Currently used sensors' sets includes conventional optical cameras, Light Detection and Ranging (LiDAR) and Red,Green,Blue-Depth (RGB-D) cameras. The later two (LIDAR and RGB-D) are scanners in the sense that they scan the scene sequentially using IR or Laser beam to obtain the depth information in addition to the 2D plain images. Although they directly measure the depth information (which is missing in conventional cameras) they suffer form other shortcomings when it comes to deploying them on board the UAS, in terms of payload and power consumption.

Particularly in the recent years, with the introduction of the Microsoft Kinect, many projects have been under development, such as those from the Microsoft KinectFusion and MIT [12] where the RGB-D sensory was used. However, RGB-D sensors like Microsoft Kinect are heavy, demanding in energy terms and not suitable for outdoor environments. The depth data is normally acquired using IR (infrared) sensors which the sunlight heavily interferes with their measurements. On the other hand, the LIDAR sensors are heavy and costy that deploying them on board a drone is not feasible. In addition to this they normally suffer from unavoidable severe occlusions. Hence although many modern high definition 3D environment modelling solutions employs RGB-D cameras or LIDAR, conventional cameras comes more handy when considered for UAS.

Small cameras have become widespread throughout industry and household devices. They are cheap, light weight and can provide high quality and intense data as a great source of information. In particular, monocular vision techniques are attractive because they require only a single, lightweight camera which is readily available and easy to deploy on board. Nevertheless, the challenge of retrieving the missing depth measurements remains the core objective of the vision algorithms to be used. For acceptable final results the

employed algorithms should be computationally efficient in a post processing offline scenario at least.

One of the technologies used are the Embedded optical flow techniques which rely on hardware to compute the inter-frame changes between images to extract depth information. These techniques have worked well on UAS, demonstrating autonomous takeoff, landing [13] and obstacle avoidance [14]. This technology has been successful for aircraft flight control and is now available commercially. Embedded optical flow, however, is limited in its resolution, providing only general guidance about obstacles. For more sophisticated functionalities like the 3D environment modelling, we must look beyond embedded optical flow techniques for solutions that provide greater resolution.

3D point clouds are playing a central role in robotics perception in general and more specifically in Simultaneous Localization and Mapping (SLAM) and augmented reality. SLAM is the process by which a robotic system constructs a map of the environment using different kinds of sensors while estimating its own position in the environment simultaneously. SLAM has been widely studied and applied with various kinds of sensors and for a multitude of robotic platforms. Although various SLAM algorithms are considered mature, such approaches are still very dependent on the platform, on the environment, and on the parameters that have to be tuned.

The fundamental idea behind the SLAM is the use of landmark correlations to improve the solution, and the implementation of an Extended Kalman Filter (EKF) to reduces the uncertainties. Using a probabilistic model, the EKF guarantees convergence and the consistency of the map. While relying on data associations, SLAM solutions can perform loop closure to reduce the uncertainties of every pose of the map. This makes the algorithms very sensitive to data association errors, which can originate either from the uncertainty of the sensors used in the estimation or from the performance of landmark recognition technique. Current state-of-the-art approaches solve this MAP problem thanks to optimization techniques such

as Bundle-Adjustment (BA). Most recently, state-of-the-art Bundle Adjustment, Structure-from-Motion methods have been the standard platform for 3D environment modelling in post processing solutions [15]. The Bundler, the Clustering Views for Multi-view Stereo (CMVS) and the Patch-based Multi-View Stereo (PMVS) packages, first published at the University of Washington, provide an offline solution for creating dense 3D point clouds given a set of unordered overlapping images [16]. Bundle Adjustment and Structure-from-Motion are used to perform dense reconstruction given a large set of unordered images with multiple views of the same scene, compute the correspondences between the overlapping portions in those images and then compute sparse point cloud, followed by a dense reconstruction of the image set, generating a dense 3D point cloud. They have produced compelling results from large sets of unordered images collected from the Internet. These same tools can be utilized and applied to the images from the drones to generate 3D reconstruction of target objects or terrains.

Given all these techniques each with their advantages and shortcomings, further studies on the effects of the various parameters affecting the quality of the resulting 3D environment model is needed in order to inform the perception system design choices.

1.2.2 Motion Planning and Control

As part of the navigation system, the motion planning and control problem is defined as the computation of the control inputs capable of driving the robot from an initial state to a target state satisfying the vehicle kinematic and dynamic constraints, as well as avoiding obstacles and other forbidden zones [17]. In addition to being complicated and computationally expensive, these problems are compounded when the robot is expected to operate in a range of demanding environments, which is the case when developing a generic motion planner. In particular, avoiding obstacles is a significant challenge to overcome. Therefore, significant number of techniques were developed for obstacle avoidance [18], [19], [20].

For the most practical complete solutions, the motion planning and control problem is typically composed of a motion planner that works in conjunction with an on-board controller that follows the planned path. For example, the authors in [21] present a framework for path planning and tracking for an autonomous vehicle collision avoidance system where a 3D hazardous potential field is used to plan a real-time collision-free trajectory, then, the planned trajectory tracking problem was formulated as an optimal problem using the MMPC technique. The authors in [22] propose a two-stage approach where path planning is computed by leveraging the Rapidly-exploring Random Tree (RRT) algorithm, associated with a Linear Quadratic Regulator (LQR) controller for the tracking of the resulting reference trajectory.
Similarly, in [23] the RRT (Rapidly-exploring Random Tree) algorithm is used to compute an initial trajectory in a 3D environment, then the planned trajectory is post-optimized in order to satisfy the state constraints using MPC resulting in a feasible trajectory plan. However, none of the two-stage methods described above can ensure that the computed trajectory is dynamically feasible.

Alternatively, a classical approach to motion planning and control partitions the problem into two phases: in the first phase, a continuous collision-free path is generated, and in the second phase, a low-cost trajectory is constructed along the previously established path while taking into account dynamic constraints. However because of the potential that the cost functions being incompatible between the two-staged optimization, this method may result in an infeasible or inefficient trajectories. For example, minimizing the Euclidean length in the first phase may result in a continuous collision-free path that is incompatible with the dynamic constraints of the second phase. Specially in scenarios when motion planning is supposed to navigate obstacles in a time-varying environment, this becomes even more evident. As a result, it would be advantageous to include both the kinematic and dynamic constraints at the same time, rather than separately, while constructing the trajectory during the planning phase.

In general terms, the motion planning algorithm is expected to optimize the computed path in terms of an execution model (such as energy, time,

distance, the amount of braking or the amount of turning), while respecting robot kinematic and dynamic constraints, as well as avoiding both static and dynamic obstacles of the environment, in order to achieve the best possible performance. Existing literature has a multitude of methods that are tailored to specific scenarios involving a particular set of constraints. Various methods to mobile robot motion planning have been thoroughly studied in recent years:

The roadmap path-planning method: Before searching for a feasible trajectory, the continuous environment model (search space) is transformed into a discrete map. Although these methods are computationally efficient for lower-dimensional spaces, the discretization of the search space renders them expensive for higher-dimensional spaces. Therefore, due to the high computational cost of these approaches, techniques such as Cell Decomposition methods [24][25], Delaunay Triangulations [26], and Dynamic Graph Search methods [27][28] are only suitable for low-dimensional spaces [29].

The Artificial Potential Field method: creates a virtual potential field in the search space with repelling fields representing the obstacles and the attracting field representing the goal configuration. Using techniques such as hill climbing and gradient descent, it is plausible to navigate the potential field formed by the forces acting on the configuration space. However, The disadvantage of APF, as discussed in [30], is that it is vulnerable to drifting into a local minima [31], which may cause the robot to get trapped oscillating between obstacles, where the resultant force of the robot is zero and the desired configuration cannot be achieved. Furthermore, in order to represent dynamic obstacles, it is necessary to make substantial changes to the potential field method.

The Dijkstra's method evaluates the vertices in a graph iteratively to find the shortest path.However, despite the fact that it is well-known to be very quick and computationally simple, it loses time for performing a "blind" search.

The A Search Algorithm* Although similar to Dijkstra's algorithm, the A* Search Algorithm is distinguished by the fact that it directs its search towards the most promising configurations leveraging weighted graphs. Because of the implementation of weighted graphs, there are many significant benefits, such as the capacity to find solutions in a relatively short period of time. Furthermore, A* has the capability of including time, energy consumption, and safety as measurements. Conversely, the A* Search Algorithm does not always provide the shortest path since it heuristically computes the cost of navigating from a specific node on the grid to a final destination.

The Genetic Algorithm (GA) The Genetic Algorithm (GA) is an optimization technique based on genetic algorithms. It is a mix of the survival of the fittest and a random data generator, and it is used to find the best solution. However, although this approach is effective for trajectory planning in scenarios where the obstacles are static, it is inadequate of taking into account the robot's dynamic constraints or the existence of dynamic obstacles in the surrounding environment.

The Probabilistic RoadMap (PRM) It is possible to tackle motion planning problems with more than four dimensions by using Probabilistic RoadMap (PRM), which is a multiple-query planner. Although PRM is capable of handling multiple queries, it requires a large number of connections in order to operate effectively. When dealing with single queries that have specific start and goal configurations, PRM may not be the most efficient planner available.

Recently, it has been demonstrated that kinodynamic motion planning algorithms may be used to satisfy both the kinematic and the dynamic requirements of the motion planning model at the same time [32]. In particular, the use of incremental sampling-based planners to solve the kinodynamic motion planning problem as a two-point boundary value problem in the dynamic state space of a robot system was shown to be effective. To address motion planning problems, sampling-based planners such as the Rapidly Exploring Random Tree (RRT) are used even in high-dimensional environments [33].

LaValle and Kuffner proposed the first sampling-based kinodynamic algorithm in [34], in which an RRT algorithm samples the control input of the vehicle and, then, predicts its corresponding motion in order to construct a tree of trajectories in the state space.
As a consequence, the computed trajectory satisfies by design the constraints imposed by the vehicle dynamics and can be readily implemented by the vehicle with relative ease. Since then, several sampling-based planners have been developed by broadening and refining such a method,such as [35] and [36] among several others.

The authors in [37] proposed the RRT* algorithm, one of the most widely used sampling-based approaches today.
RRT* constitutes an improvement of the original RRT algorithm towards the attainment of a near optimal solution. Further improvement has been made to the RRT* algorithm for specific applications such as real-time path planning [38], anytime planning [39], multi-agent planning [40], and others [41].
RRT* has improved the path quality of the original RRT through employing two new major mechanisms: rewiring and best neighbor search. RRT* incrementally adds new connections to the existing tree whenever a new sample is generated. Such a rewiring procedure gives RRT* the chance to gradually improve its path-cost, asymptotically approaching the lowest-cost path as the number of iterations increases.
However, this rewiring is performed on a local basis, preventing a global propagation of the changes in the graph and the consequent optimization at the global level. An improvement is constituted by RRT#, which generates a guaranteed asymptotically optimal graph that always contains the lowest-cost path [42].

Another common concern in sampling-based approaches is related to the random sampling of the control space, instead of the random sampling of the reference space of the robot [43].
While sampling in the reference space always generates feasible trajectories, sampling in the control space may often result in the selection of control inputs that can lead to infeasible trajectories, due to the presence of dynamic constraints.

This typically yields longer execution times and an inefficient management of the algorithm. To address this problem, the authors in [43] proposed the *closed-loop* RRT (CL-RRT), in which the samples are drawn from the reference space instead of the control space.
The sampled reference is then used to compute a trajectory using the closed-loop model of the robot.

Hence, a more efficient kinodynamic motion planning strategy based on RRT-class of planners is a promising improvement that would be suitable for UAS in particular. The desired kinodynamic strategy should guarantee the feasibility of the trajectory for each sample, and uses as minimal samples as possible to build the feasible trajectory by sampling in the reference space instead of the control space.

1.3 Objective

This research focuses on the *Perception* and the *Motion Planning* of Unmanned Aircraft Systems for Autonomous Navigation purposes.

In perception the objective is segmented into two parts. The first one deals with the vision sensory system design choices. The objective here is to analyse the various design parameters that affect the quality of the 3D environment model generated from a vision system. This will include the camera design and configuration parameters in addition to the flight plan parameters (speed and altitude). In addition to analysing the effect of geo-tags uncertainty on the quality of the 3D environment model. Whereas, the second part deals with the algorithmic solution for an industrial setup where a challenging object recognition and localization is considered. The objective in this part is to develop an industrial solution that complies with the reliability, robustness and processing time defined by the application.

In motion planning the objective is to investigate the motion planning strategies currently available in the literature with the objective of developing a novel kinodynamic sampling-based motion planning algorithm capable

of effectively addressing motion planning problems under differential constraints. The algorithm must be applicable to the commercially available Unmanned Aircraft Systems (UAS) equipped with a professional autopilot. The resulting trajectory computed must be at least a near-optimal trajectory that respects both the kinematic and dynamic constraints of the UAS.

1.4 Outline

This Ph.D. dissertation is split into two parts.

The first part deals with Perception, and includes two chapters. Chapter 2 investigates the UAS on-board vision system design characteristics that influence the quality of the 3D environment modelling. Whereas in Chapter 3 we further extend the analysis to the design trade-offs between the various camera parameters and the flight plan under specific requirements like the final object resolution, the UAS speed and the required images' overlap.

The second part presents the work done in Motion Planning, which is further split into three chapters (4, 5 & 6). Chapter 4 covers the literature review and background about motion planning methods used for autonomous navigation. Whereas Chapter 5 presents the problem definition and the proposed motion planning solution (MP-RRT$^{\#}$). Chapter 6 details the experiment and discuss the simulation results of the MP-RRT$^{\#}$ proposed motion planning solution.

Finally in Chapter 7 presents the research conclusion for both parts and highlights some prospective future work.

Part I

Perception

Chapter 2

3D Environment Modelling

The set of capabilities reputed as perception enables robotic systems to comprehend their surroundings and internal dynamics. In particular, accurate and reliable environment sensing and localization are essential for autonomous navigation. Using a map to represent the environment model, the robot will be able to recognize open areas, obstacles, and diverse semantics and signs, all of which could serve as the groundwork for undertaking motion planning. The resultant environment model must, in terms of model richness, accuracy, and update rate, fulfill the navigational requirements of the target application. All of those requirements have an impact on subsequent specifications for the kind of sensor set and algorithmic techniques employed to obtain the environment model. This is typically performed using a combination of sensors to collect information from the robot's environment, followed by the utilization of advanced sensor fusion techniques to produce an environment model that depicts the robot's surroundings. Particularly, the integration of vision sensors with IMUs and odometery data to improve robotic localization has delivered intriguing headway in obstacle avoidance and autonomous navigation applications. The algorithms used in such applications should be computationally efficient at the very least in a post-processing offline scenario.

Out of the various sensors used for perception, small cameras have become widespread throughout industry and household devices. They are cheap, light weight and can provide high quality and intense data as a great source of information. In particular, monocular vision techniques are attractive

because they require only a single, lightweight camera which is readily available and easy to deploy on board. Nevertheless, the challenge of retrieving the missing depth measurements remains the core objective of the vision algorithms to be used.

In this chapter, we present an introduction to visual perception covering the state-of-the-art, and we explore the design parameters of the mobile robot visual-perception system, as well as their impact on the quality of the resultant 3D environment model in terms of richness and accuracy. We investigate the 3D environment model and aerial mapping solutions based on Structure-from-Motion (SfM) and bundle adjustment were identified. Despite the fact that this application is a post-processing visual-perception application, the choice of this application was motivated by the goal of investigating the trade-offs in vision system design that would have an impact on the resulting environment model in terms of accuracy and richness, as well as the overall quality of the resulting environment model. Because they are more algorithm reliant than sensory system dependent, the computational cost and real-time features were not taken into account. This application especially provides us with the opportunity to investigate the trade-offs involved in vision system design and their impact on the quality of the perception model. Precision flight planning and precise equipment selection/configuration (camera, optics, and imaging parameters' settings) are required for 3D environment modeling and aerial mapping solutions based on the Structure-from-Motion (SfM) and bundle adjustment techniques (number of images and their overlap percentage).

More specifically, we investigate the design trade-offs between the camera and flight plan attributes, as well as the impact of these trade-offs on image quality, as well as the quantity and quality of matched features that are subsequently used in the 3D environment model of aerial mapping solutions based on the Structure-from-Motion (SfM) and bundle adjustment techniques. Section 2.2 addresses the design phase trade-offs between the camera selection/configuration parameters and the flight plan parameters for the UAS 3D environment modelling applications. The camera selection/-configuration and flight planning dependencies are presented in a structured

map facilitating the trade-offs between the various design parameters (camera sensor size, optics focal length, flight altitude, overlap percentage and UAS speed) during the design process.

2.1 Introduction to Visual Perception

For an autonomous mobile robot to successfully perform the navigation task, it must know its position relative to the position of its goal. Furthermore, it must take into account the hazards of the external environment and readjust its operations in order to increase the probability of reaching its destination. Specifically, an autonomous mobile robot must have the ability to recognize and represent its environment in an environment model that encompasses key entities such as: the target to be reached, the location of static obstacles, the ongoing and prospective location of moving obstacles, recognition of various extracted features, and classification of the collected data according to its semantic interpretation, in addition to the vehicle's present state (position, speed, and so on).
Perception is the expression used to designate the problem of extracting a contextual comprehension of one's surrounding. Perception of the surrounding environment encompasses data acquisition, which would be attained through the use of exteroceptive sensors with a wide range of capabilities and attributes, and data processing in order to generate the knowledge required to produce an environment model, which will be used to plan and carry out behaviour in the future. The kind of model and the design of the sensory subsystem are determined by the application and the configuration of the vehicle. For a mobile robot to be equipped with the adequate sensory subsystem capable of supporting a rich perception adequate for navigation planning whilst still complementing its design and intended uses, a holistic and diverse collection of multiple sensors is often utilized. In this framework, the fusion of complementary information supplied by multiple sensors is a fundamental problem to be addressed.

Nonetheless, vision systems standout among the sensory subsystems that are deployed in autonomous navigation strategies. The integration of vi-

sion sensors with inertial measurement units (IMUs) and odometery data to enhance robotic localization has ultimately results in some significant breakthroughs in obstacle avoidance and autonomous navigation systems. With imaging sensors, mobile robots can develop a greater understanding of their surroundings through images and videos, which have been shown to be particularly effective due to a number of advantages such as their capabilities to provide distinctive viewing angles, high resolution data, and extensive coverage data about the surrounding environment. Those qualities are useful in a wide variety of applications because they provide efficient, quick, and cost-effective solutions. For example, following Hurricane Katrina, unmanned aerial vehicles (UAVs) were dispatched to look for individuals or groups who were being trapped by flooding substantially more swiftly than emergency personnel who went out in rowboats. Following the massive earthquake that devastated Japan in 2011, unmanned aerial vehicles (UAVs) were employed to perform reconnaissance and assess environmental parameters at the devastated Fukushima Nuclear Power Plant, which was too dangerous for humans to approach. An unmanned aerial system fitted with vision systems can also be used to inspect power plants and other sites for vulnerabilities that might constitute a threat to the environment. Vision-enabled UAS have also become more widespread in agriculture, where they allow for more accurate crop management, which can increase yield whilst saving producers millions of dollars in time and resources. Farmers are also using UAS to control for pests and pathogens, oversee their crops, and inspect for symptoms of dryness and disease, all at a reduced cost than they could normally spend. Those and other emerging applications are made possible by breakthroughs in vision-enabled unmanned aerial systems technology, which makes use of state-of-the-art image sensory and computer vision algorithms for perception, as well as environment modeling, to provide a more realistic view of the world. Among the most important vision-enabling technologies is 3D environment modeling, which has the capability of providing the spatial distribution of data and is thus very significant. Besides being effective for navigation, digital 3D models of urbanized areas are suitable for a variety of activities, including urban planning, virtual reality, and entertainment.

Particularly for autonomous navigation systems, 3D point clouds are playing an important part in the perception of mobile robots, as is the case with the Simultaneous Localization and Mapping (SLAM) system (SLAM). The overall topic of Video-Based Navigation (VBN) includes space-applications, which are attempting to extract such breakthroughs and tailor them for space applications such as autonomous landing. VBN may be used to improve the accuracy of the landing operation during the Entry, Descent, and Landing (EDL) phase of space modules (for example, space-crafts). For example, in the historic space research missions Spirit, Opportunity, and Curiosity, the descending trajectory and landing site were pre-computed and calculated with a maximum landing point accuracy of 20 kilometers. It is possible to lower the landing ellipse through using VBN-assisted EDL technologies in their development, hence boosting the accuracy of the landing.

In general, computer vision solutions may be thought of as a spectrum, with the VBN and obstacle avoidance at one end and the post-processing 3D model reconstruction at the other end of the spectrum. Navigation and obstacle avoidance are examples of solutions that are designed and optimized for computational efficiency, real-time responsiveness, and reliability. There are no stringent requirements for a detailed representation of the environment because the primary goal is to provide sufficient situational awareness for localization and environment modeling. However, when it comes to post processing (3D environment model), the solutions are built and tuned to enable a thorough, rich, and exact representation of the environment. This takes more time and computational resources and is not as efficient as the first method. Throughout this chapter, the section **??** will investigate the design parameters of the mobile robot visual-perception system, as well as their impact on the quality of the resultant perception model in terms of richness and accuracy. This will be investigated by exploring the use of 3D environment modeling in the UAS environment. Whereas section 3.3, on the other hand, will further develop a computer vision solution for a manufacturing line in an industrial setting. The performance requirements in this application are the solution computational cost, dependability, and reaction time.

2.2 UAS Vision System for 3D Environment Modelling

Structure from Motion and Bundle adjustment techniques are used in a broad range of applications, including photogrammetric survey, the automated reconstruction of virtual reality models from video sequences, and the detection of camera motion, among others. The process of the SfM and the bundle adjustment framework is shown in Figure 2.1 in which the multi-view photos' set – which is the sole mandatory input – is utilized to extract the distinguishing features and match them across the whole collection of images. It is from this list of matching characteristics that all subsequent computational blocks are constructed (camera calibration and point cloud generation). According to the amount of features recovered from each frame as well as the overlap between images, the number of excellent quality matched features is determined. This encourages us to describe the design phase in terms of two parameters: *extractable number of features per image and the overlap between the images*.

Fig. 2.1 Structure from Motion Workflow

Camera selection and Configuration

The images acquisition system characteristics can be grouped into the three sets:

- The Optics: the focal length and the lens Modulation Transfer Function (MTF) which describes the performance of the optical system.
- The sensor: the sensor size and the Pixel size.

- The operational configuration: the focus, shutter speed and exposure time, ISO and the aperture settings.

The real object resolution and the contrast level of the photographs are the two most important specifications for the images' set that determine their overall quality. However, among other characteristics, the actual object resolution of the photos is dependent on both the camera object and the height at which the photographs are taken. The camera object resolution is also dependent on the sensor resolution and the optical quality of the camera lens (MTF).
In this case, selecting the sensor for the needed camera object resolution and then selecting optics -by evaluating their MTF- that can maintain that sensor resolution while providing acceptable contract performance is a sound strategy. However, The object resolution of the generated photos, on the other hand, is not solely dependent on the camera's object resolution, but is also directly influenced by the flight altitude and the camera field of view (FOV). Those factors are also interdependent and are influenced by the amount of overlap required between the photos and the velocity of the UAV. Hence, there is no explicit way to quantify or choose such parameters in the design, whereas it is typical to have limitations on the flight plan (altitude range, UAV speed and overlap), as well as on the imaging system (altitude range, imaging system overlap, focal length and sensor size). Figure 7.1 depicts the inter-relatedness between the various factors and groups them into three categories: requirement parameters, selection parameters, and configuration parameters. Figure 7.1 scheme may be utilized to reason about the many trade-offs involved in the design phase.

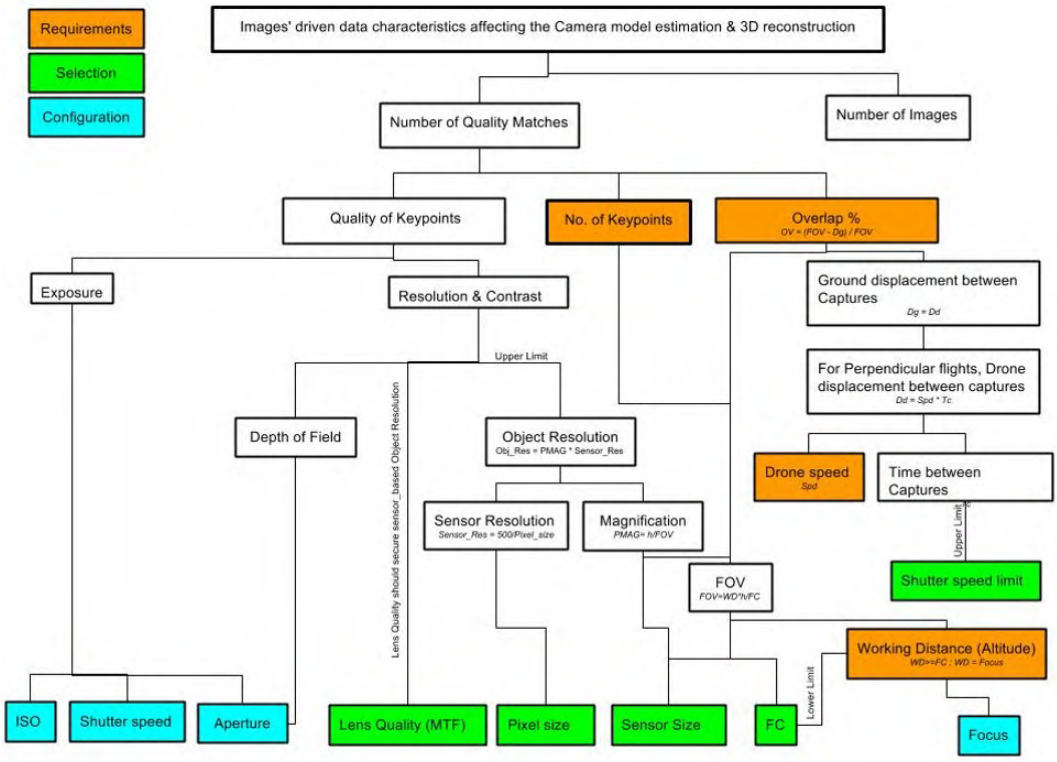


Fig. 2.2 Vision System Design parameters inter-dependencies and the way they affect the final 3D reconstruction through affecting the number of quality matched features

Chapter 3

Visual Perception for Mobile Robots

The UAS visual perception system can be seen as two processes: visual data collection, and the transformation of the collected data to useful information. In the first two sections of this chapter we evaluate the design aspects of the first process (visual data collection), while in the third section we develop a solution for object recognition and localization (transforming data into information). The first two sections deals with the UAS vision system design and configuration including the sensory system design and the use of additional GPS data to further improve the quality of the data extracted from the vision system. Then, the third section presents the object recognition and localization solution for an industrial setup. Although the developed solution was implemented in an industrial setup instead of a UAS, nevertheless, the solution presented is built with UAS limitations in mind. Limited computational power and the reliability and response time requirements.

3.1 Visual Perception Design Process

The following design route is recommended for a typical design process in which criteria such as the object resolution (Obj_{res}), UAV speed ($UAVsp$), time between captures (T_c), and overlap percentage ($Ov_{\%}$):

1. In order to determine the desired field of view(FOV), one must take into consideration the UAV speed, duration between captures, and overlap percentage as follows:

$$FOV = \frac{UAV_{sp} \times T_c}{100 - Ov_{\%}} \tag{3.1}$$

2. The sensor and pixel sizes should be carefully selected from the commercially available options in order to achieve the necessary object resolution and the previously calculated field of view:

$$\frac{Obj_{res}}{FOV} = \frac{Pixel_{size}}{Sensor_{hor}}$$

3. Make a conscious choice from among the commercially available focal lengths for the optics in order to maintain a suitable flying altitude while still respecting the sensor specifications and the calculated FOV that have been defined previously:

$$\frac{Sensor_{hor}}{FOV} = \frac{F_c}{WD}$$

Table 3.1 depicts a typical design parameters computed with the above described scenario:

Requirements	Computed parameters	Selection
Time between Captures = 12 s	FOV_{min} = 3.6 m	FOV = 4m
Object Resolution = 2 mm UAV speed = 3m/s Object	Maximum Sensor ratio: $\frac{Pixel\_size}{Sensor\_hor}$ = 0.5 x 10^{-3}	$Sensor_{hor}$ = 5.6 mm $Pixel_{size}$ = 2.2 um Actual ratio =0.4* 10^{-3}
overlap% = 90%	$\frac{F_c}{WD}$ =1.4 x 10^{-3}	F_c = 25 mm WD_{max} = 17.8 m

Table 3.1 Typical Camera and Flight Design Flow

3.1.1 Operational parameters' configuration

Changing one of these factors (aperture, shutter speed, or ISO), it will have a direct effect on the magnitude of detected light intensity at each given pixel under identical lighting circumstances. The aperture has an influence on the Depth of Field (DOF) and the exposure, but has no effect on the Field of View (*FOV*), (AFOV), or the effective focal length. The shutter speed has an impact on the exposure time and the capability to trace rapidly moving objects, and consequenty has an impact on the designed maximum velocity of the unmanned aerial vehicle. When changing the aperture for purposes of depth of field (DOF) or the shutter speed for purposes of flight speed and moving objects in the scene, the ISO can be used as a compensation variable to modify the sensitivity of the sensor cells in order to manipulate the contrast for a given shutter and aperture settings. The configuration guidelines for defining these settings that have been learned via experience are listed below:

- Focus: should be set for the distance between the drone and the highest point in the terrain.
- Aperture: should be set to have enough depth of field, equal to or greater than the distance between the highest point in the terrain and its base (ground). The best corresponding f-stop value can be extracted from the MTF curves of the lens or by trial before the actual flight.
- ISO: should be set to automatic allowing the software can adjust for the lighting conditions by altering the pixels' sensitivity.

3.2 Use of Additional Data Sets

This research was previously initiated by evaluating the performances of real-time vision-based navigation and autonomous landing algorithms that were used for the validation of the Mars lander mission at the Vision-based Terrain Navigation Facility in Thales Alenia Space Italy. During the Entry, Descending, and Landing phases of the mission, an unmanned aerial vehicle (UAS) flying over a scaled Mars environment is utilized to locate safe landing

locations autonomously and in real time. During a subsequent stage, an outside landing scenario was to be performed in order to determine the impact of extra data (such as GPS data) on the resultant 3D point cloud and, ultimately, the 3D environment model.
Despite the fact that nadir and oblique flight plans are the most typically employed in "earth" applications, certain requirements and methodologies from this space application may be applied in traditional mapping applications. This includes the evaluation of the use of any extra information about the environment to be recreated, such as geo-localization data or any other geometrical limitations. An on-board camera (SONY ILCE-QX1L camera with 20.1MP) was utilized to record photographs while descending and ascending in the altitude range of the flight test (vertical flight) on an agricultural terrain in order to mimic a landing situation (vertical flight) on an agricultural terrain (60 - 20 m).

When the UAV was flying, two GPS data sets were utilized to record the geotags of the UAV when the images were captured (one conventional GPS used by the UAV autopilot and another L1/L2 GPS corrected using PPK) to ensure that each picture was geotagged.

Each of these two data sets was then used with the same images set as inputs to the PIX4Dmapper (a professional 3D environment modelling software) as initial guess for the camera extrinsics with various uncertainties to investigate the effect of two GPS data sources “Geotags” with various uncertainties on the CAMERA CALIBRATION and resulting 3D environment modelling.

3.2.1 Quality Indicators

In order to assess the overall quality of each 3D environment model, various indications were created for both evaluating the Camera Calibration results and judging the 3D environment model.
Camera Calibration:
Focal length deviation from nominal value: variation in focal length is used as a quality control measure for the calibration results because the focal length

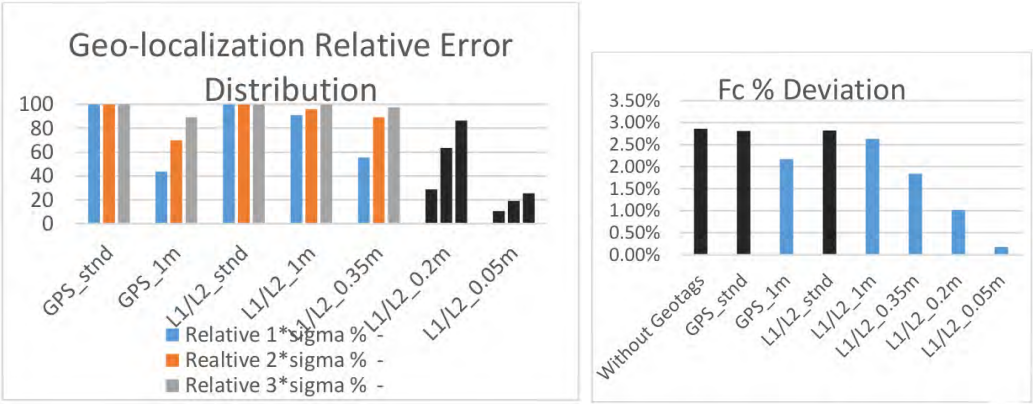
nominal value is well-known to the manufacturer and changes in focal length under typical working circumstances are assumed to be restricted. When doing an adequate camera calibration, it is assumed that the focal length will stay within a 5 percent range of its nominal value.
3D environment model:

- Number of densified 3D points.
- Ground Check Points deviation: the discrepancy between the calculated coordinates of a reference ground point and the actual coordinates of that ground point.

3.2.2 Results and Discussion

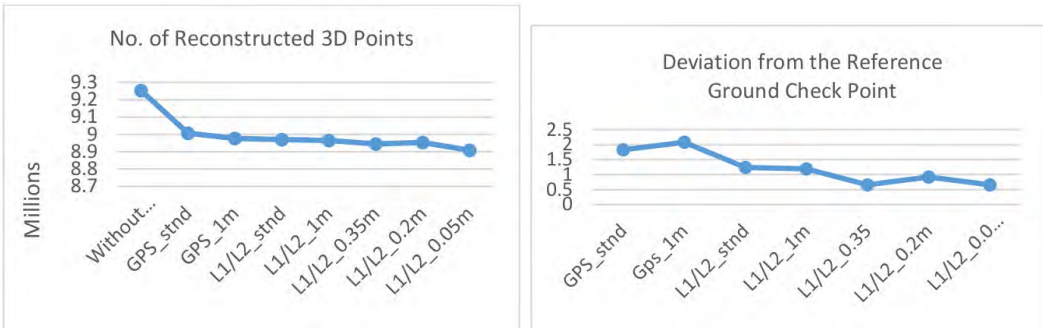
Figure 3.1(a) depicts the geo-localization relative error distribution, which is used to assess the validity of the calibration results. The results show that the calibration estimates from the L1/L2 GPS with 20cm & 5cm uncertainties resulted in a poor coverage of the error which indicates that those uncertainties are stricter than what is compatible with the images.

Figure 3.1(b) shows that although the L1/L2 GPS is intrinsically more precise, when used with standard uncertainty (L1/L2_stnd) produces the same results as the conventional GPS (GPS_stnd). This is because when using larger uncertainties, the software optimization algorithms will have way more confidence on the image driven data (matched features) than on the Geotags, which will result in having almost the same quality of the project eventually for conventional GPS, the L1/L2 GPS and even without Geotags.



(a) (b)

Fig. 3.1 (a) Focal length deviation from the nominal value, the two data sets with the same large uncertainty bounds (standard) gives the same calibration results which is also equal to the calibration without any additional data. (b) Distribution of the residual of the estimated camera positions deviation from the initial data



(a) (b)

Fig. 3.2 (a) Number of 3D points reconstructed with each additional data set as a measure of the richness of the resulting 3D model. (b) Deviation of the Ground Control Point in each 3D reconstructed model as a measure of the accuracy of the environment model.

Figure 3.2(a) indicates that the 3D points reconstructed without geo-tags were noticeably higher than the reconstruction with any additional localization data sets. This shows that the use of localization data leads to mark some matched features as inconsistent, but due to the good images' quality the number of reconstructed 3D points was sufficiently high in all cases. We notice also that the use of different GPS data sets results in a limited difference in the number of reconstructed 3D points, which is reduced with

stricter uncertainty bounds.

Figure 3.2(b) indicates that the Ground check points' deviation from the 3D model was significantly less for the L1/L2 GPS with respect to the conventional GPS. Novatel with 35cm uncertainty had the best overall results for both camera location estimation and 3D reconstruction location referenced to the ground check points.

3.2.3 Conclusion

Using the 3D environment model richness and accuracy as quality indicators, we realized that they are a function of the quality of the camera calibration – which is a self-calibration process in the SfM framework (3.3) – and the amount of quality matched features. When it comes to the camera self-calibration process, the quality of the initial values and the data set utilized have a significant impact since it is a non-linear optimization problem (which is the matched features in this case). In order to assess their contribution to the total 3D quality indicators previously specified, we evaluated the use of two different GPS data sets as initials for the camera location (extrinsic parameters) in order to determine their accuracy.
The comparison of the resulting 3D environment models revealed the use of additional data sets will have a limited effect when it is used with a stated uncertainty bounds higher than the actual ones. The use of smaller uncertainty limits on the other hand results in inferior camera calibration and, thus, worse reconstruction outcomes.
This emphasizes the need of determining the real uncertainty limits of any extra data sets, which are often not the same as the nominal ones but are rather impacted by system factors such as camera capture/GPS synchronization. The design trade-offs between the various camera parameters and the flight plan under specific requirements like the final object resolution, the UAS speed and the required images' overlap was addressed. The interdependencies and relations between these parameters and the number of quality matches was mapped and a structured computation and trade-off workflow was recommended for terrestrial mapping applications.

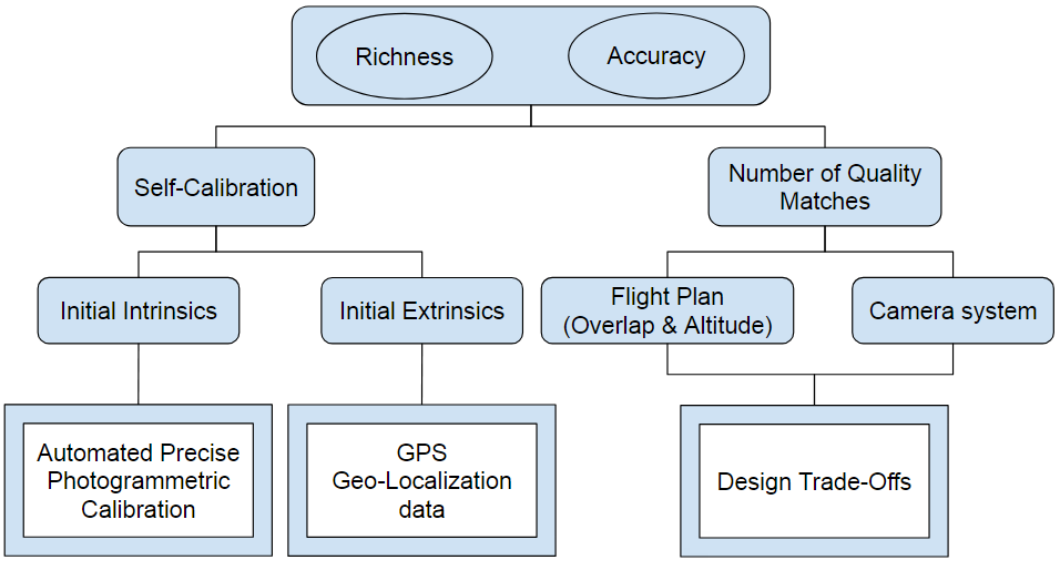


Fig. 3.3 Topics investigated and their place in the SfM framework

3.3 Industrial Visual Perception

In this part, we will investigate and design a visual perception solution for an industrial robot that will be used in a manufacturing process. When used in an industrial setting, more than often, the visual perception algorithm is expected to be integrated with an industrial robotic arm that performs some pick and place operations. The objective is to develop a robust detection and localization algorithm using only visual sensory. The developed solution must also have computational cost, reliability and response time relevant to UAS application scenarios. This means that the computer vision algorithm should be robust, reliable to work in continuous mode and with frame throughput that is way less than the robotic arm assembly cycle-time. The application setup includes the following systems:

- Vision System: a programmable camera that serves as the primary controller utilized to detect the pieces that are in the proper orientation to be handled by the robotic arm, as well as to provide the coordinates of the pieces to the robot for picking. When none of the pieces are available, the camera should decide whether to signal a feeder with the adequate command in order to further increase the parts' density

on the conveyor, or whether to signal the conveyor with the adequate command in order to shake the existing parts in order to change their orientation, whichever is more best suited. This feature places the vision system at the core of the solution, not only identifying and localizing the components, but also determining why there are no parts available for assembly at the time of recognition (do we need to add more parts or shake the existing parts or both).

- Feed system: when it is instructed by the vision system to deliver more pieces or to start shaking the conveyor (flip) in a preconfigured mode. The feed system is capable of dispensing different amounts and shaking the conveyor in different configuration modes depending on the command it receives from the vision system. It is constantly transmitting its current state to the camera for control purposes.

- Robot: commanded by the vision system to pick ready pieces. The robotic arm is also needed to continually transmit its status (whether it is inside or outside of the camera's field of view) to the vision system for control purposes.

The solution should be dependable, resilient to variations in lighting conditions and minor dimension variations in the pieces of interest. And most importantly, it should be multiple times faster than the robotic arm assembly operation in order to avoid increasing the total production cycle-time in the manufacturing process.

3.3.1 Industrial Visual Perception Solution

Using edge-based pattern training, the suggested solution is implemented. The algorithm in the training phase is manually tweaked to identify the edges of each piece, with the defining edges then being used to train a sparse search algorithm based on these edges' patterns . In this case, the camera is a 5MP programmable camera with a global shutter and a frame rate of 16

frames per second. The camera optical settings (FC, FOV, focus, and depth of field) are set in order to get the maximum sensor resolution (magnification ratio and focus for the conveyor area). The algorithm is described in further detail below.

Solution WorkFlow

According to a minimal success factor criteria, each raw acquired image is utilized to search for/recognize up to 20 pieces. When the system is first starsd, this pattern recognition function will be performed in FULL-SEARCH mode, which implies that it will process the whole frame of the raw image being processed. Once successfully identified, pieces that have been found in a region of interest, those pieces are passed on to the other four particular recognition processes, where they are designated as completely recognized parts. These four distinct recognition functions are as follows:

- Front/Back orientation check: to differentiate between the head and tail of the part;
- Up/Down orientation check: i.e. the part is not upside down with respect to the default picking configuration;
- Gripper Clearance check: i.e. the gripper can reach the parts without collision;
- Robot Reachability Check: i.e. the part is within the robot working area.

Parts successfully passing all those recognition functions are then ranked according to their success-rate and the best will be selected. The selected part origin coordinates in the camera frame is then processed to compute the picking point coordinates in the world frame through relative transformation and camera/world frame calibration functions.

If the list of successfully recognized parts remains non-zero for a predefined number of samples (robustness threshold), then the selected part will be considered as robustly found and the picking coordinates will be sent to the robot whenever the robot is ready. On the other hand, if the list

of successfully recognized parts remains zero for a pre-defined number of samples (feeder threshold), then a properly defined command (depending on the number and distribution of parts in the shaker bed) will be send to the feeder to either dispense more parts or perform a specific shake operation.

The remaining parts in the successfully recognized parts list are then latched to the next sample time. As long as this list has more than a single part, the next sample operation will run the raw pattern recognition function in LIMITED-SEARCH mode. In this mode, pattern is searched for only in selected areas around the previously saved parts' list instead of searching the full frame image. This way we perform FULL-SEARCH only once every while and in between we perform LIMITED-SEARCH which is way faster working on less image area. This eventually has a significant effect on reducing the mean cycle time of the overall operation.

The Scenario described above can be logically segmented into the following functional parts:

- Image Processing: Images filtration to be used in the different recognition functions giving some degree of robustness against dust (dynamic noise) and lighting conditions.
- Full Search: Part Recognition in Full image
- Limited Search: Part Recognition in small regions around previously found parts.
- Calibration: Camera to World frame transformation of target parts.
- Robustness evaluation: Evaluate the part Robustly-Found and Robustly-Not-Found state before using it for either picking or feeder command respectively.
- Feeder Control: Sending the proper Shake/Dispense command to maintain reasonable number of parts in the camera FOV, monitoring the Feeder status.
- Robot Interaction: Sending coordinates of target parts and monitor the Robot state (IN/OUT camera FOV).

Fig 3.4 depicts a top-level structure of the workflow:

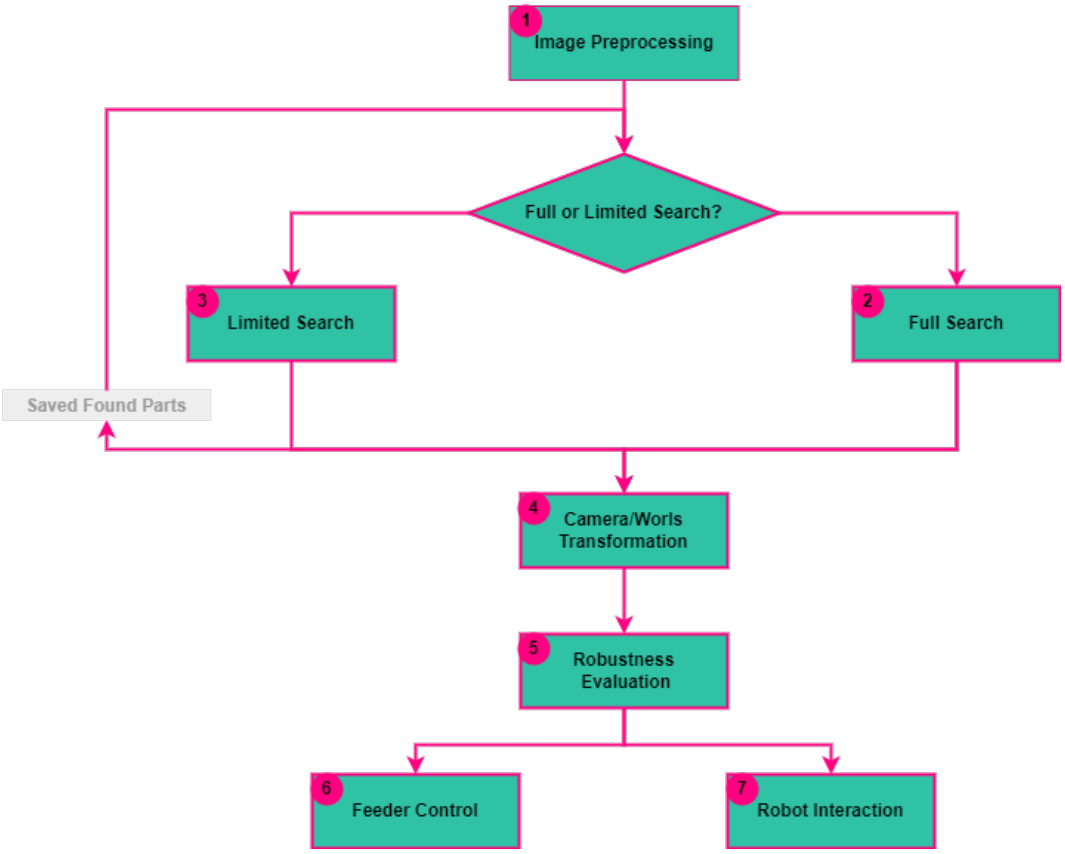


Fig. 3.4 Algorithm Top Level Structure.

Image Processing

As described previously, captured images are used through several recognition functions which collectively will decide the best recognized part. The final quality of those pattern recognition functions depends greatly on the quality of the images they process. To this end various combinations of filters were implemented to furnish different images suitable for every specific pattern recognition function. Filters' types and configurations depend on several factors which can be categorized into static factors and dynamic factors. The static factors include the geometry and material reflection of the part to be recognized. Whereas the dynamic factors include the variability of lighting conditions and the image dynamic background noise resulting from the d debris accumulation in the Feeder shaker bed through time. The

dynamic background noise was by far the most dominant factor affecting the image quality and consequently the part recognition hit rate. Hence, filters' combinations were repeatedly tested and configured to give satisfactory performance in extreme background noise conditions -kindly refer to the test videos.

In addition to the raw captured image, another filtered image was configured by implementing a binary filter followed by a noise cancellation (close) filter, 3.5. The resulting image is a high contrast noise free image that produces satisfactory results in Front/Back orientation check, Up/Down orientation check and Gripper Clearance check operations.

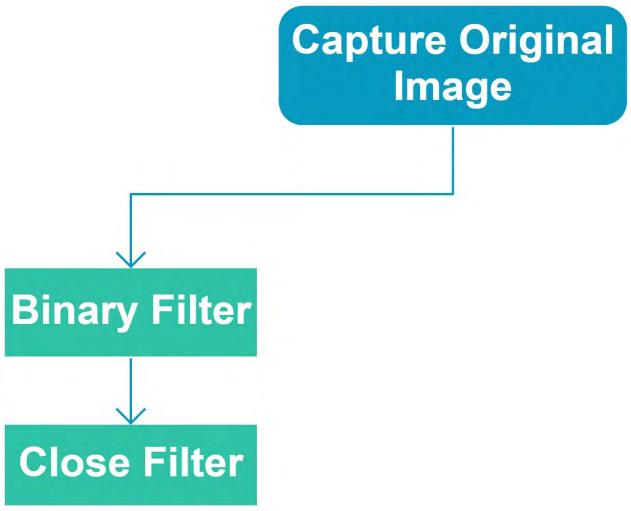


Fig. 3.5 Image processing: binary filter and noise cancellation

Full/Limited Search Decision

The decision whether to process the full image FULL-SEARCH or LIMITED-SEARCH on specific areas depends on the number of found parts in the previous sample. Starting from initial conditions where no parts were found previously, FULL-SEARCH will be enabled. If the FULL-SEARCH finds only one ready part, then this part shall be picked leaving empty list and hence the next sample shall be processed in FULL-SEARCH mode too. However, if

the FULL-SEARCH finds more than a single ready part, then the best part shall be picked, and the rest of the list shall be saved/latched to be used by the LIMITED-SEARCH mode in the next sample. This logic is depicted in Fig 3.6 flowchart. The final outcome of this logic is to enable one of the two modes and disable the other one (Full/Limited) which shall be used consequently by the rest of the application.

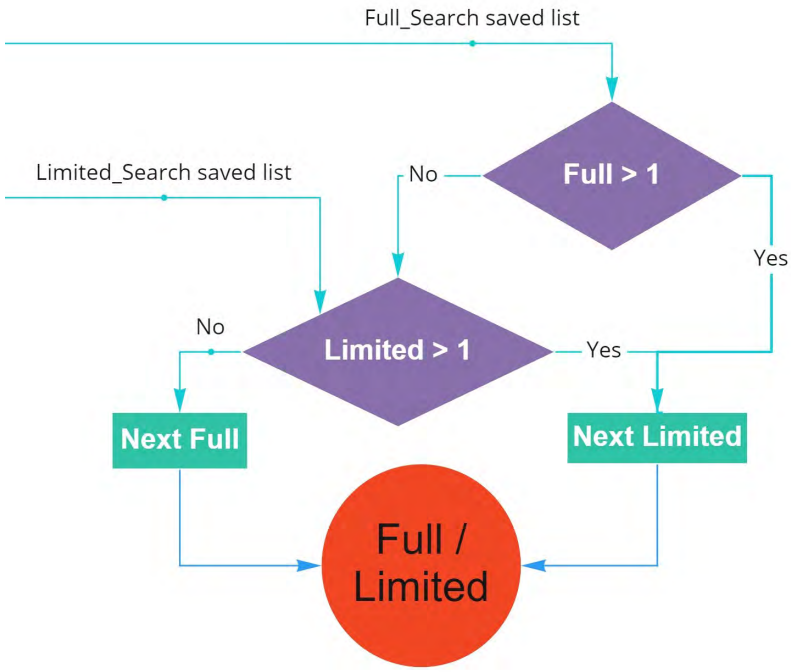


Fig. 3.6 Full/Limited search Decision Flowchart

Both search modes have the same structure the only difference is that the LIMITED-SEARCH mode limits the search area while the FULL-SEARCH mode process the full camera frame. This difference practically translates to that in FULL-SEARCH mode we will use a single pattern recognition function whereas in LIMITED-SEARCH we will use 20 pattern recognition functions each for a small area around a previously found part. The rest of the recognition steps (Front/Back orientation check, Up/Down orientation check, Gripper Clearance check operations and Robot Reachability Check) are identical. For the LIMITED-SEARCH to work it needs a previously saved list of ready parts. This list is provided by the previous sample whether it

was LIMITED-SEARCH or FULL-SEARCH. Fig 3.7 describes the process of saving/latching the ready parts' list for the next sample, whereas Fig 3.8 and Fig 3.9 describes the FULL-SEARCH and LIMITED-SEARCH overall recognition logic respectively.

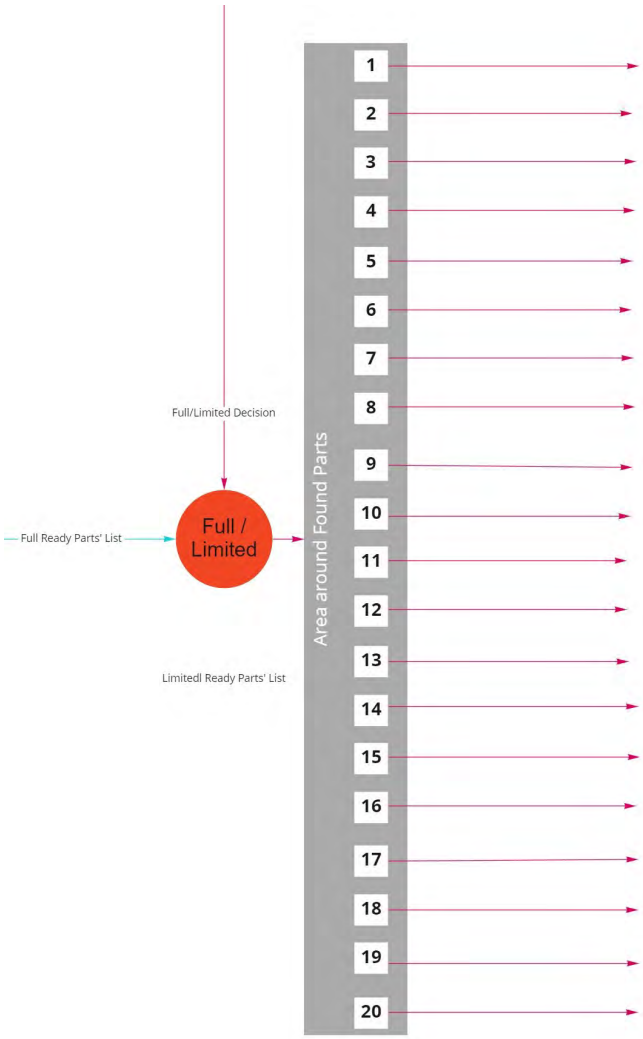


Fig. 3.7 Saving/Latching ready parts' list

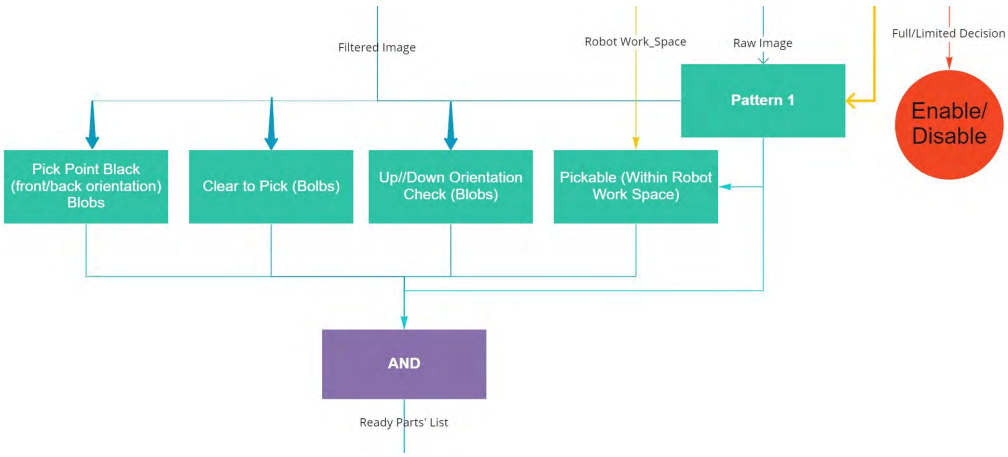


Fig. 3.8 Full Search

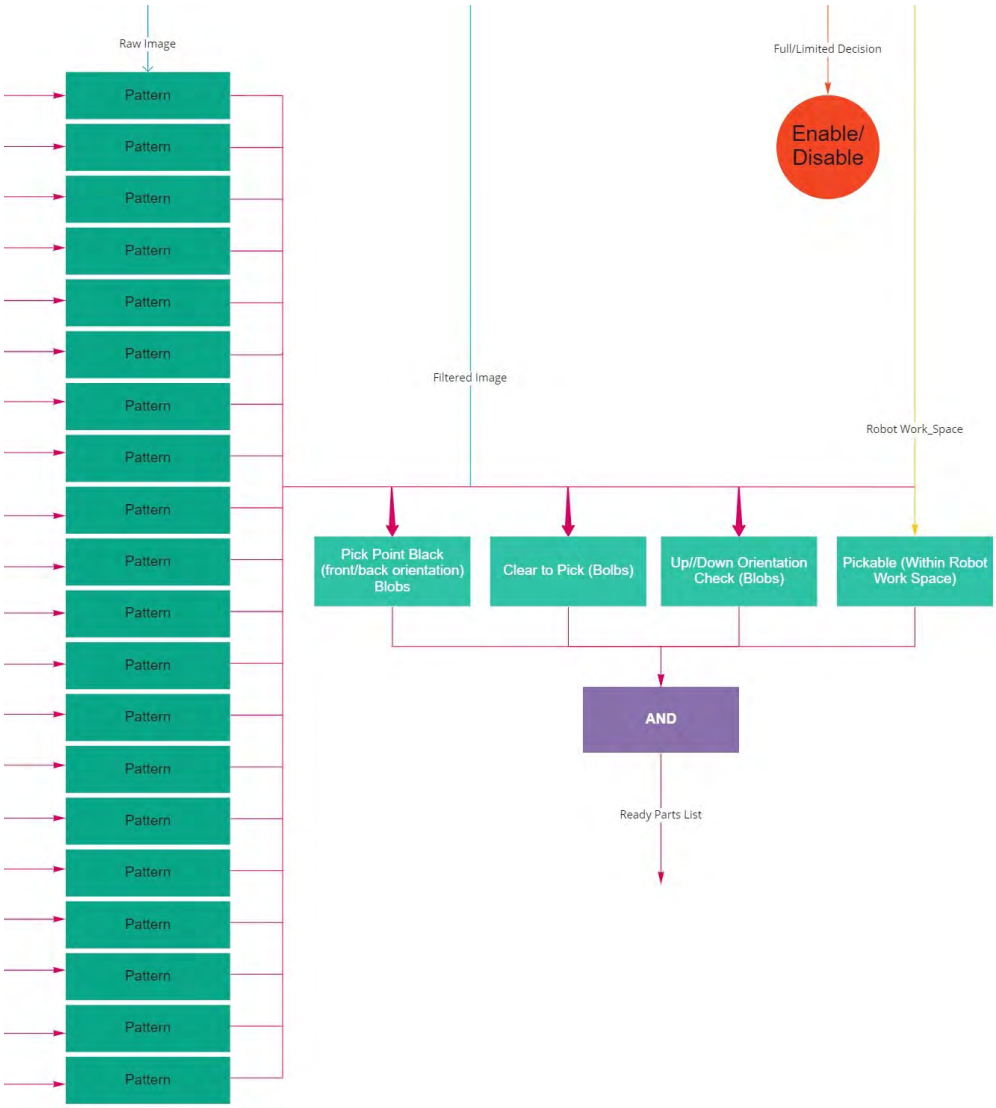


Fig. 3.9 Limited Search

Full Pattern

To compensate for the camera optics prospective, some scale variance was tolerated (95% – 120%). The number of recognized parts were further improved by tolerating 8 overlap between recognized parts.

Robot Reachability

Parts found by the Full Pattern recognition function are evaluated against the robot workspace. The Full Pattern function delivers the pattern origin for every successfully recognized part. This origin is then transformed using fixed user-defined angle and distance to obtain the picking point. The resulting picking point is evaluated if it lies within the robot work space which is user-defined too.

Up/Down Orientation

The feature used to differentiate between the up versus down orientation is the square hole in the middle of the part. In the binary filtered image this square appears black for the down orientation due to the shadow it makes whereas it appears white for up orientation. To check the square region only, the origin coordinates given by the Full Pattern Function Is used to build a square area of interest (through fixed transformation). This square area includes the square hole at the centre of the part. A blob is used to evaluate each square area to check if it is black (indicating down orientation) or white (indicating up orientation). The shadow might appear partially due to the filter operations, so further blob features (elongation) are used to make the final judgement about the orientation.

Gripper Clearance

The gripper approaches the target part while open and closes only when it has reached the part already, Hence, the robot needs a clearance around the part to be picked. This clearance (shape and size) is user-defined for the specific part and gripper. The result is a geometry around the picking point

that should be clear of obstacles. In the application a gripping clearance area is drawn around ever found part and a blob is used to check that this area is clear.

Front/Back Orientation

This part is meant to determine the head from tail of the part because sometimes the part is recognized but the angle orientation is missed by 180 degrees which result in wrong picking coordinates sent to the robot. A circle area around each picking point is drawn and further evaluated using a blob to make sure it contains the head.
For a part to be included in the Ready-Parts-List it should pass all the above-mentioned steps (AND function). If any of the previous steps failed (down orientation, back orientation, out of robot workspace, no enough gripper clearance) then the part found by the Full-Pattern function will be discarded.

3.3.2 Calibration

Depending on the recognition mode (FULL-SEARCH or LIMITED-SEARCH) the calibration process selects the best part from the Ready-Parts'-List of the corresponding recognition mode. This part is defined by its origin in in the camera frame, the calibration process was implemented to first compute the picking point coordinates given the origin coordinates and then calibrate the picking point coordinates from the camera frame to the world frame. The resulting coordinates are the ones that shall be sent to the robot eventually. The calibration process depends on a user-defined saved calibration object that correlates the camera coordinates to the robot ''world" coordinates. Fig 3.10 shows the calibration process flowchart.

The remaining segments of the application are three sections which are inter-related, i.e. Robustness evaluation, Feeder Control and Robot Interaction. Their operations are inter-related because the interaction with the Robot and the Feeder depends simultaneously on the robustness evaluation

in addition to their states which are monitored continuously.

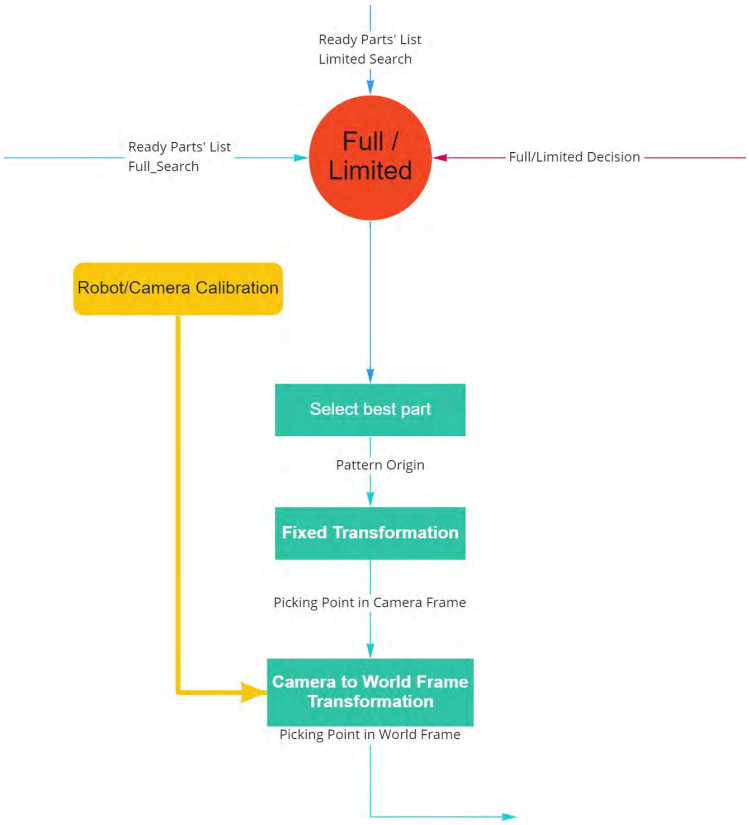


Fig. 3.10 Camera-Robot Calibration

3.3.3 Robot Interaction

The picking point coordinates computed by the calibration step is sent to the robot if and only if the following conditions were met simultaneously:

1. Robot has made the transition from IN to OUT state; i.e. it made the transition from inside the camera FOV to out of the FOV. Here the transition is used instead of simply the OUT state to ensure that the robot gets the coordinates of the part only once. In early trials the condition used was the robot being in OUT state (state not transition) and the result was that the robot received the same coordinates more

than once, which drove the robot back to the same point where he already picked the part previously.

2. A part has been Robustly Found; this is a result from the Robustness evaluation segment. This condition ensures that the part found is steady and not still moving or vibrating. The condition is crucial because each part type requires a certain amount of time before it settles with no apparent vibration after a shake operation. Without this condition the application might find a part which is still on the move or vibrating.

The actual robot interface is configured as a TCP/IP device with a String+Carriage_Return message format. Two functions (write & read) are used to send coordinates and receive robot state respectively. The write function takes the picking point world coordinates $(x, y, angle)$ and formulate a string with a comma separator. On the other hand, the Robot is programmed to continuously send the IN state whenever it is inside the camera FOV and send OUT otherwise.

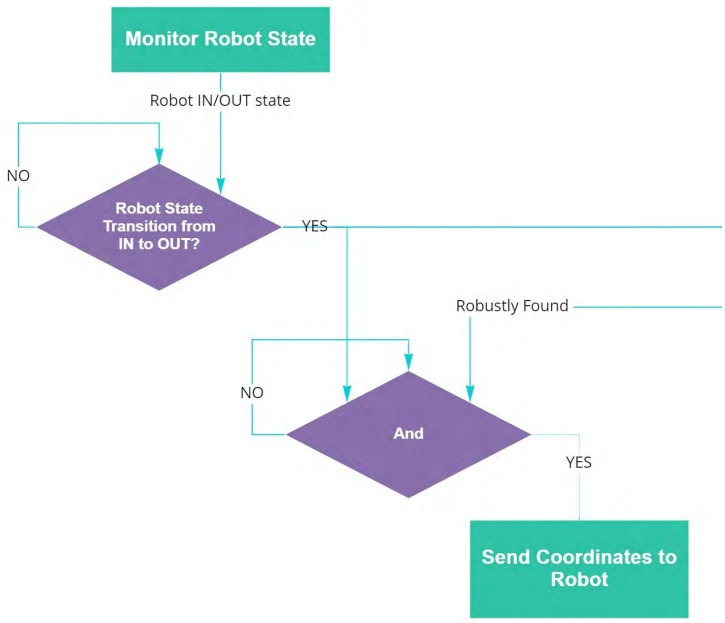


Fig. 3.11 In/OUT FOV monitoring

3.3.4 Robustness Evaluation

This segment of the application was implemented to increase the overall application reliability. The entire part recognition result is assessed as being trustworthy based on the reliability. That is to say when the application indicates that it did recognize a part, or it cannot find any part, those statements should be reliable and trustworthy since they will be used to either command the robot for picking or to command the feeder for a shake of dispense operation with specific configuration. Poor reliability will lead to more feeder operations that the optimum and less parts picked by the robot which eventually translated to longer mean cycle time.
Robustness is measured by counting in a history timeline (3 previous samples and the current sample) the number of samples where we found parts and the number of sample where we did not find any part. To work with a timeline, a script object is more efficient than normal cell logic and latching operations. So, the robustness evaluation logic was implemented using a script object that takes the following inputs:

1. From Robot Interaction segment: Signal indicating the robot IN-OUT state transition.
2. From Full & Limited Search segments: the total number of found parts.
3. From feeder control segment: feeder ready signal.

The final outputs of the script object are the following:

1. Robustly Found signal: used as a condition (among others) to send the coordinates to the robot.
2. Robustly Not Found signal: used as a condition (among others) to command the feeder in the feeder control segment.

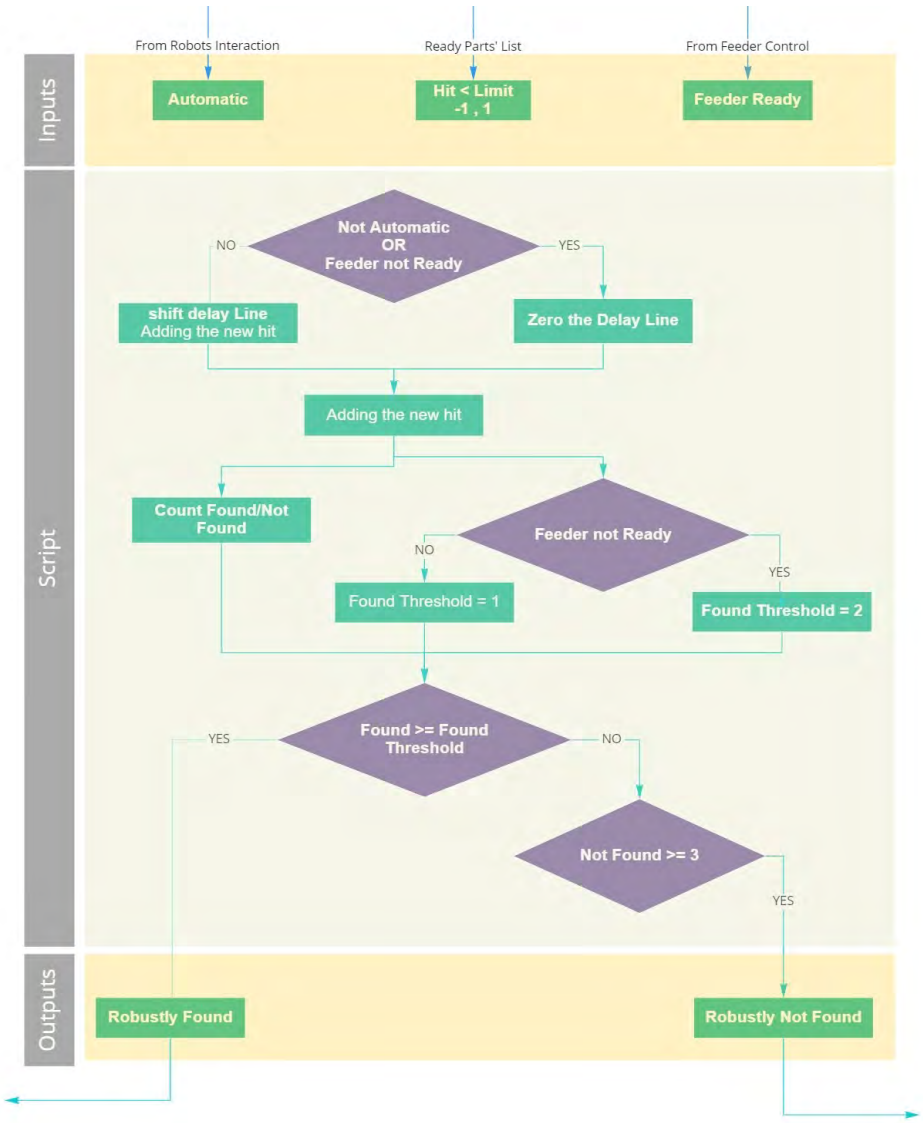


Fig. 3.12 Robustness Evaluation

As illustrated In Fig 3.12, the found threshold used depends whether the feeder is active or not (shake or dispense operation). If the feeder is active (not ready) then we set a higher threshold, i.e. the part should be found in more samples because when the feeder is active the parts are moving/vibrating. On the contrary, if the feeder is not active (ready) then we set a lower threshold because the parts are assumed already settled.

3.3.5 Feeder Control

The feeder controller main function is to maintain suitable parts' density in the feeder bed, not too much that its not possible to recognize individual parts nor too few that requires dispense operations more often. The feeder has three main operations (Flip, Flip forward, Flip backward & Dispense). Each of those operations has two parameters to be set defining the operation interval and intensity. To this end the image was divided into two segments (Back & Front). This way we can better address the frequent situation where a dispense operation accumulates much parts in the back of the feeder bed with very few parts in the front segment. In such case, knowing the parts' density of the two segments separately, the feeder controller will be able to request a feeder forward flip operation to spread the parts more evenly across the feeder bed. Similarly, knowing the parts' density of the two segments, any other in-homogeneous distribution of parts shall drive the feeder controller to request the suitable feeder operation. The parts' density is measured practically by the image histogram. Hence, for each segment we have a histogram measured value and user-defined desired range.
Comparing a segment (Front or Back) measured value to its desired range can give one of three states (Higher, Within, Lower). The feeder command request depends on the resulting combination of comparison results of the two segments. Each combination was assigned to a proper feeder operation (flip, Dispense...).
The above detailed logic describes the way the application selects the feeder operation request. However, this feeder operation request will be communicated to the feeder under some specific conditions that ensures the proper functioning of the feeder and not interruption of robot picking operations. This feeder operation request shall be communicated to the feeder if and only if the following conditions are satisfied:

1. The robot is out of the FOV (Automatic).

2. Enough time delay has elapsed since the last command sent to the feeder; without this condition consecutive feeder commands will physically overlap causing unpredictable feeder operations like violent flips causing the parts to fly out of the working area. This is practically

implemented through monitoring the feeder state, when the feeder receives a command a timer is set off. Only when this timer has elapsed, and the feeder motor drives are back to idle this condition is set.

3. Robustly Not Found signal from the robustness evaluation segment.

The feeder physical interface is a serial port through which it receives and echoes every message received and some specific codes describing the state of its two motor drives. Through those echo & state messages, the feeder state is continuously monitored to ensure that the commands sent were successfully received and executed. Fig 3.13 describes the feeder controller logic.

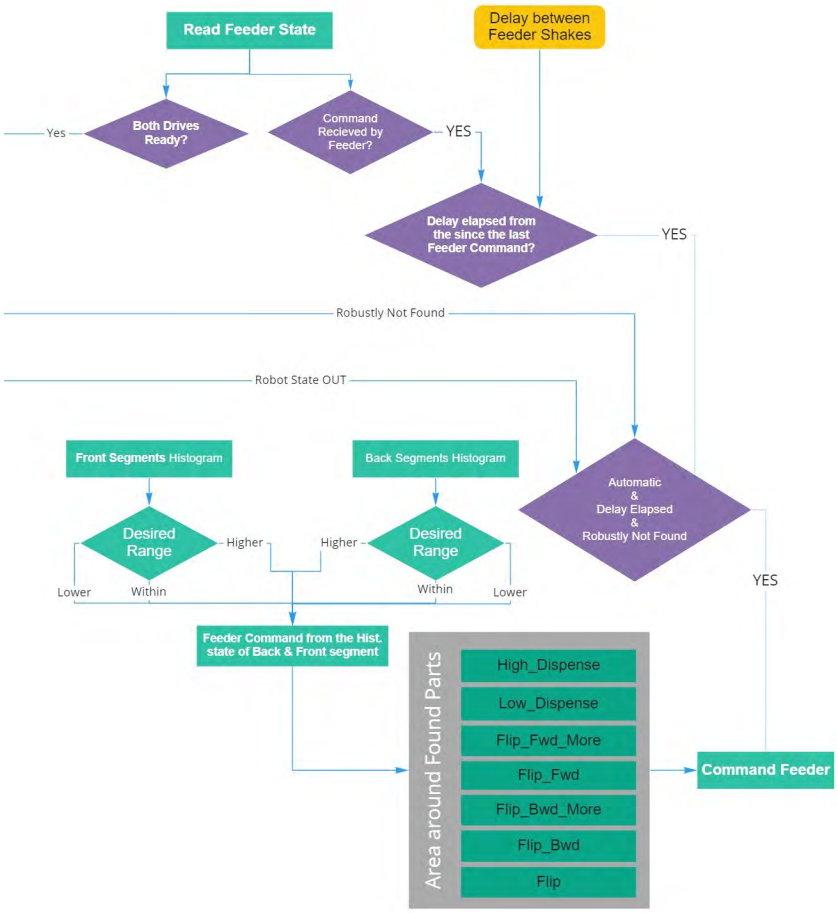


Fig. 3.13 Feeder control logic flowchart.

3.3.6 Remarks

In this application we have used state-of-the-art image processing techniques to implement pattern recognition. The core innovation in this application was the manipulation of those image processing techniques in order to respect the application requirements in terms of cycle time, precision and robustness of recognition. The staging of the pattern recognition (Full Search Limited Search and robustness evaluation) was crucial in meeting those application requirements.

Part II

Motion Planning

Chapter 4

Motion Planning State-of-Art

This chapter presents a broad literature review for motion planning methods used for autonomous navigation.

4.1 Background

Unmanned aerial vehicles (UAVs) have recently attracted the public's attention due to their potential for use in a variety of applications. However, in order for UAVs to be effective in unstructured environments, they ought to be able to operate independently and with high availability and reliability, while also demonstrating great flexibility in order to adapt to changing circumstances. They are not, however, insurmountable, even when faced with such strict standards.
Technology advances in a number of relevant fields, such as optimal control methods, machine learning and efficient computing capabilities, have not only made it possible to develop cutting-edge capabilities for improved performance, but they have also created new opportunities for developing autonomous capabilities for a wide range of unmanned systems, such as UAS.

Unmanned vehicles can be divided into several categories based on the environment in which they operate, including unmanned aerial vehicles

(UAVs), unmanned ground vehicles (UGVs), unmanned surface vehicles (USVs), and autonomous underwater vehicles.

Unmanned aerial vehicles (UAS) are best described in the aviation industry as aerial aircraft that do not need human navigation or control but have a high level of operational efficiency, allowing them to operate in hazardous environments where personnel would otherwise be at risk of injury or death. As a result of these characteristics, they are becoming increasingly popular for a wide range of applications such as remote sensing, search and rescue, security and surveillance, precision agriculture (including precision agronomy), infrastructure inspection and urban planning, space exploration, bomb detection, and even recreational activities, such as drones, to name a few. Unmanned aerial systems (UAS) have significant hurdles in developing autonomous navigation skills as compared to other autonomous vehicles on land or in water. Due to the unique characteristics of unmanned aerial systems (UASs), which are resource-constrained systems in terms of energy consumption, computational capacity, and payload limits, as well as the restricted hardware-set available onboard, a number of challenges must be overcome. Unmanned aerial systems (UAS) also include a range of characteristics, such as nonlinear dynamics, three-dimensional environments, shocks and errors in state information, to mention a few examples. Hence, a more stringent set of criteria must be met in the development of autonomous solutions and motion planning techniques than are necessary for typical mobile robots or manipulators.
According to [8], as a result of the rising use of unmanned aerial systems (UAS), resource constraints, and other inherent issues in the UAS, several associated academic fields have experienced an increase in popularity, with autonomous flying leading the way.

Intelligent solutions in a variety of domains, including perception, localization and mapping, motion planning and control, and navigation, are required to address the challenging problem of autonomous flying. Automatic motion planning, in particular, continues to be one of the most difficult difficulties that autonomous robots must overcome presently. The topic of motion planning is well-known in a variety of areas, including robotics [33][17], assembly maintenance [44], computer animation [45], computer-aided surgery [46] and manufacturing [47] is that of motion planning.

A motion planner's overall goal is to enable users to describe operations using high-level languages and have the robot turn those descriptions into a collection of low-level motion primitives, or feedback controllers, that are necessary for the job to be successfully completed. The ability of a robot to change configurations while avoiding impediments is critical for the correct operation of any robot, whether it is an arm or a mobile robot. This is the most fundamental function performed by all robots. This subject areas, however, been enlivened by the introduction of new challenges such as uncertainty, an enormous number of entities, and the dynamic behavior of the entities themselves. Motion planning, among other things, includes the planning of movements among obstacles as well as the synchronization of movement with other mobile robots, among other things.
For instance,when it comes to autonomous cars, motion planning entails considerably more than simply the identification of objects in the road. People's intuitive interaction with their environment, as well as the sort of operational intelligence necessary for sensing and motion planning, are challenging to mimic in a computer program, according to [17]. Conversely, motion planning can be defined as *"the computation of the control input required to drive a vehicle from a starting state to an end state while complying with the kinematics and dynamics constraints of the vehicle and avoiding obstacles or driving through otherwise prohibited areas."* As a result, motion planning may be thought of as a challenge that addresses the three problems listed below.

- The best way to move from one configuration to another while minimizing the length of the path, the amount of energy expenditure (or whatever measure may be of relevance to the application in question);
- On the path, keep clear from obstacles and restricted areas.;
- Find the path as rapidly as possible with the least amount of computing effort.

Furthermore, the motion planning problem can be contextualised in different environments dealing with obstacles of various characteristics, as follows:

- Motion planning in a known environment with static obstacles;

- Motion planning in a known environment with dynamic obstacles;
- Motion planning in an unknown or partially known environment with static obstacles;
- Motion planning in an unknown or partially known environment with dynamic obstacles.

4.2 Methods for Motion Planning

First and foremost, in order to navigate through the motion planning approaches available in the literature, we must establish certain nomenclature that will be used throughout, notably the robot configuration and the configuration space.

- **Robot configuration**: The configuration of a robot system is a complete specification of the position of every point of that system.
- **Configuration space**: The configuration space, also known as the C-space, of a robot system is defined as the space containing all potential configurations of the system.

// In recent years, a number of different approaches to mobile robot motion planning have been researched and tested. In the literature, there are many types of motion planning techniques from which to choose. For starters, as numerous authors have pointed out, separating them into off-line and online categories is a useful overall distinction to make.

- Offline path planning is defined as follows: Creating a plan in advance, based on a known model of the environment, and then passing it on to an executor is referred to as offline route planning. In some instances, if thorough information on both static and moving impediments is available, this strategy may be utilized efficiently to overcome such hurdles. This approach creates a complete path from the robot's starting

point to its ultimate destination before the robot starts to move, allowing the robot to arrive at its goal faster. For example, service robots, autonomous guided vehicles, and other similar devices in which the acquired environment map is not susceptible to change are examples of off-line route planning.

- On-line path planning: Whenever the planner produces the plan gradually while the robot executes, the planning environment is said to be "on-line." In this case, the planner may be sensor-based, which means that it will be able to combine sensing, computing, and action into a single bundle of functionality. This type of navigation is used to travel in scenarios when the environment is entirely unknown, courses have been planned, and the robot is moving at the same time.
  Its course changes as it travels as a result of changes in the environment detected by sensors in the environment, which cause it to modify its path. The most fundamental kind of online planning starts in offline mode and transforms to online mode when new changes in the status of obstacles are detected, as described above. As examples of path planning in online context, [48] and [49] provide the examples of reconnaissance robots and planet exploration, respectively. As a consequence of technology advances, many formerly offline techniques are now as On-line planners.

A significant number of researchers have created techniques that have been tested in a range of circumstances with both dynamic and static limitations in trying to address the motion planning problem.
When Nils John Nilsson presented a mobile robot system with basic motion planning capabilities in the late 1960s, it marked the beginning of the field. He was also the first person to use the visibility graph approach in the world. In contrast, while it became a research topic in the 1970s, it was only in the 1980s that considerable active development began, owing to the introduction of computers in the workplace.
There has been a huge increase in the number of techniques created during the last two decades. When dealing with the high dimensionality of configuration space C-space, which was a source of difficulties for academics in the 1990s, planners resorted to randomization to help them cope.

The methodology and algorithms for current motion planning may be divided into four categories: classical approaches, probabilistic approaches, heuristic approaches, and evolutionary approaches. Classical approaches are those that are based on mathematical formulas, while probabilistic approaches are those that are based on probability.

4.2.1 Roadmaps

Roadmaps are a generic term that refers to the creation of a graph in the configuration space that specifies how different configurations are connected to one another rather than a specific path planning method in and of itself. As a general rule, the initial step in any roadmap path-planning method is to transform a continuous environmental model or search space into a discrete map that is appropriate for the path-planning technique that has been selected. The discretization of the search space makes the method computationally costly for higher-dimensional spaces since it increases the number of possibilities.
A road map is made up of a sequence of paths, each of which connects the starting points to the destination without colliding into any obstacles. A usable trajectory may be identified if all of the configurations of a roadmap are connected together, as well as the start and goal configuration being connected to the roadmap. Furthermore, a roadmap may be used to answer a variety of questions concerning the search space if they are needed. Among them, indeed is the shortest path between source and destination included in the visual graph or the roadmap. Thus, the path used is often longer than necessary, but it is also more secure. The visibility graph and the Voronoi diagram are the two most well-known types of road maps [50] [51], respectively. To deal with optimality problems, Bhattacharya and Gavrilova [52] suggested utilizing a Voronoi diagram to provide a safe clearing path. Furthermore, techniques such as Cell Decomposition methods [24][25], Delaunay Triangulations [26], and Dynamic Graph Search methods [27][28] are only suitable to low-dimensional spaces [29] due to the high computational cost of these approaches. The negative effects of discretization also affected algorithms such as [53][54][55], which combine the set of allowed

movements with graph search techniques to create state lattices, resulting in a reduction in performance.

Artificial Potential Fields (APF)

The Artificial Potential Field [56] is a well-known path planning method that was originally suggested by O. Khatib in 1986 and has since gained widespread acceptance. They are a family of path planning algorithms that rely on attracting and repellant forces to steer an object across as it moves through the configuration space.

A potential function is a differentiable real-valued function. The value of a potential function may be thought of as energy, and the gradient of a potential function can be thought of as force. The gradient is a vector defined as the following equation:

$$\nabla U(q) = \begin{bmatrix} \frac{\partial U}{\partial x_1}(q) \\ \vdots \\ \frac{\partial U}{\partial x_n}(q) \end{bmatrix} U : \mathbb{R}_m \to \mathbb{R} \tag{4.1}$$

This principle is based on the idea of creating a virtual potential field in the configuration space that repels obstacles while simultaneously attracting the robot to the desired configuration. By using methods such as hill climbing and gradient decline, it is possible to evaluate the potential field produced by the forces acting on the configuration space. An artificial intelligence robot is driven by a potential function, much like a particle moving through a gradient vector field.
In order to find a collision-free path, the robot must first follow the downward gradient of the staged potential function in the descending direction of the potential function in order to bypass the obstacles and move from its starting location to its destination location under the influence of these two forces. When the robot reaches the point q where $U(q) = 0$, it comes to a complete stop.
The most critical points in U are the points where q is a maximum, minimum, or saddle point. This method has been extensively used because because

of its simple structure, high computational efficiency and ability to regulate processes in real time.

Some navigation issues may be solved by designing the potential field in such a manner that following the gradient always results in the robot reaching the objective. This is known as gradient following. Nevertheless, it has the drawback of being susceptible to falling into a local minimum point [31], where the resulting force of the robot is zero and the desired configuration cannot be attained. As described in [57], the robot may get stuck by oscillating between obstacles. This technique also does not work well in environments with small passageways, as previously stated.
As an alternative, in the event that computing such a potential field is difficult or impossible, researchers instead employ one that is simple to calculate but may have the undesired characteristic of local minima, which are places where the robot becomes “stuck". In this instance, researchers also might simply utilize the prospective field to direct a search-based planner, which is a straightforward solution.
Many scholars have proposed solutions to this problem, such as introducing virtual target points [58] or instructing robots to move randomly [59], employing the simulated annealing algorithm [60], employing the adding extra control force method [61], incorporating the genetic algorithm into the artificial potential field method [62], incorporating the gain factor [63], incorporating a virtual obstacle concept [64], or incorporating a virtual obstacle concept [65]. Jinseok Lee developed an internal state model to solve the local minimum challenge at a low computing cost [66], which is considered to be a breakthrough in the field.
Zhang Tao and associates [67] presented an enhanced wall-following method as well as path memory. The vector potential function was suggested by Anugrah K and associates [68]. Ya-Chun Chang and associates [69] integrated the Artificial Potential Field technique with the Voronoi diagram method to enhance the movement quality of mobile robots. Rahman suggested that each agent be assigned a fixed path, thus eliminating the need to change the trajectory [70].
Qinzhao Wang developed a technique of gravity field rotation and virtual obstacle filling [71], which was later used by other researchers. In his paper [72], Chen JinXin proposed a repulsion deflection model. It is also essen-

tial to note that integrating the time-based element of the problem with a potential approach is not straightforward since a force does not take into consideration if an obstacle is moving. As a result, the trajectory of a moving obstacle cannot be predicted using a force, which means that a potential field technique may not be feasible or may need significant modifications.
However, despite the fact that these drawbacks make the development of a potential field algorithm very unappealing, there are certain benefits to doing so, as shown in [73]. As an example, take into account the fact that these techniques generate smooth, curved trajectories on their own, which may be desired at a later stage.

Cell Decomposition

These methods are founded on the concept of splitting the configuration space into simple cells, which are then connected to one another in order to create a roadmap of the configuration space.
In this way, precise cell decomposition structures represent free space by combining simple regions known as cells into a more complex structure. The shared boundaries of cells often have a physical significance, such as a change in the proximity of the nearest obstacle or a change in the line of sight to surrounding obstacles, among other things. If two cells have a shared border, they are said to be neighboring.
It is named after the adjacency graph, which is a kind of data structure that stores the adjacency connections between cells. A node corresponds to a cell, and an edge links nodes of neighboring cells.
Assuming that the decomposition has been calculated, path planning using a cell decomposition is often done in two stages: first, the planner identifies the cells that include the start and goal, and then the planner searches for a path inside the adjacency graph, which is generally done in two steps.
It should be noted that the adjacency graph may also be used as a map of the available free space. As a result, mapping may be accomplished by building the adjacency graph in small steps over time.

Cell decomposition, on the other hand, differ from other techniques in that they may be utilized to obtain complete coverage of the target area. A coverage path planner is an algorithm that calculates a path that passes an

effector (such as a robot or a detector) across all of the locations in a free area. Given the simplicity of each cell's construction, basic movements such as back-and-forth forming operations may be used to cover each cell; once the robot visits each cell, coverage is complete. In other words, coverage may be simplified to the task of locating an exhaustive walk across the adjacency network of nodes.
Sensor-based coverage is accomplished by concurrently covering an unknown area and building the space's adjacency graph. Another essential characteristic of this family of algorithms is that they ensure completeness, which implies that if a solution exists, it will always be discovered.

Despite this significant benefit, cell decomposition methods are associated with a significant number of drawbacks as well.
For instance, they will get increasingly more difficult to execute as the number of dimensions of the problem increases, which may cause difficulties in the future if somehow the problem is broadened.
The second point to mention is that cell decomposition methods may take a long time to process since the whole configuration space is indexed, which may not always be required.

The grid method is a popular cell decomposition method that was used for the compilation of the environment map, among other things. The more difficult it is to define the grid's size, the smaller the grid's size must be, and the more precise the environmental representation, the smaller the grid size must be.
A new molecular decomposition strategy is developed when obstacles, goals, sensor platform and field of vision (FOV) are provided as limited and closed subsets related to Euclidean workspace [50], [74].
Some of the issues posed in this method need the use of a large amount of memory for the analysis of the environment, resulting in significant computational complexity.

4.2.2 Heuristic Methods

Previous motion planning techniques relied on an explicit representation of the geometry of an obstacle-free environment, which was not always attain-

able or practical. As a result, as the size of the configuration space expands, these planners become unfeasible.
However, although heuristic planning methods have only recently gained popularity in contrast to conventional approaches, they are very important owing to the fact that they are based on human-like behavior and therefore learn as they go.
Various path-planning applications are studied using heuristic motion planning techniques, which use a range of various methodologies for generating samples (collision-free configurations of the robot) and connecting the samples with trajectories in order to find solutions.
The Dijkstra algorithm and the A* algorithm are the two most often used algorithms. Both algorithms may be thought of as special types of dynamic programming [75].

Dijkstra algorithm:

Although its not a heuristic method, the Dijkstra's algorithm is the base behind the A* algorithm. Hence we will start by discussing the Dijkstra algorithm before we discuss the first heuristic method A*. Dijkstra algorithm is one of the most historical algorithms for determining the shortest path between vertices in a network, and it is still in use today.
It was proposed in 1956 by the computer scientist Edsger W. Dijkstra, who devised an algorithm to discover very short paths between the nodes in a graph that may represent a road map. This method is capable of finding a path based on the cost of edges, which may be thought of as a "cost-to-go" from a starting node to a destination node. The algorithm's goal is to find the shortest path between any two vertices in a network, regardless of their location.
Dijkstra's algorithm, given a collection of vertices, first reports the vertex with the smallest distance from the preceding vertex (i.e. the lowest cost for connecting the current vertex to another vertex). It is feasible to evaluate the shortest-path-to-take in order to reach the target vertex in this manner.
Iteratively repeating the following stages until all vertices have been visited once a starting vertex has been defined is how the method works:

- Visit the vertex that is the least distant from the starting vertex and has not been visited yet;
- Examine the vertices that are the unvisited neighbours of the current vertex;
- Calculate the distance (cost) of each neighbour from the starting vertex;
- Keep a record of the distance (cost) that each neighbor has to go from the starting vertex.;
- For each of the newly updated distances, update the preceding vertex to reflect the new distances;
- Mark the current vertex as visited;
- Proceed to the next unvisited vertex in the list.

However, despite the fact that it is very quick and computationally simple, Dijkstra's algorithm wastes time by conducting a "blind" search and performing needless computations.
Furthermore, Dijkstra's algorithm has the capability of returning the global optimum solution with regard to a quantifiable variable, such as length or another cost variable. The minimization of multivariable functions is, on the other hand, expected in a same situations. This method has been updated to account for these limitations by using certain heuristics to make the process more efficient.

Algorithm A*

The A* Search Algorithm is one of the most often utilized techniques in path finding and graph traversals because of its simplicity and ability to find solutions in a very short amount of time. When this algorithm is run, it searches for all possible paths that lead to the goal and evaluates which path incurs the least amount of cost (i.e., minimal time, minimal distance traveled, and so on).
The method selects the path or pathways that are most likely to result in a quick solution out of all of the potential alternatives.

The method works via the use of weighted graphs: starting from the first graph node, construct a path tree that begins from that node, observing potential paths one by one, until it reaches the point where one of its paths ends up ending up at the targeted node. For the most part, A * is comparable to Dijkstra's capabilities with the exception that it directs its search towards the most promising states and can save a considerable amount of time in the computation process. Consider a square grid with a large number of obstacles, ideally, the optimum path from a starting cell to a goal cell should be planned in the shortest amount of time.
The A* Search Algorithm selects a node x at each computational step by calculating the cost f(x), which is equal to the sum of two functions, $g(x)$ and $h(x)$. At the end of each iteration step, the node/cell with the lowest $f(x)$ value is selected. The values of $g(x)$ and $h(x)$ are defined as follows:

- $g(x)$: calculates the cost of moving from a starting point to a particular cell on the grid by following the path that was created to get there. This is the same cost as is calculated in the Dijkstra's Algorithm.
- $h(x)$: The estimated cost moving from a particular node on the grid to a final destination. In order to ensure that the optimal solution in the graph is ultimately reached, the heuristic must be admissible, which means that it must never overstate the cost of advancing toward the target node.

For this method, the most significant benefit is in its ability to accept, modify, or add another distance to the distances used as a metrics. This enables a broad variety of changes to be made to this fundamental concept, allowing for the inclusion of time, energy usage, and safety in function $f(x)$.
Furthermore, due of the heuristic that has been established, it is possible to take into consideration global information [76]. It is not necessarily the shortest path that is found by using the A* Search Algorithm since it largely depends on heuristics/approximations to compute $h(x)$. As previously stated in [77], this method provides a solution, which is a series of vertices that must be followed in order to navigate from a start point to a target position.

In the end, all of the methods mentioned above have been successfully applied to the new robot systems that have emerged on the platform in the last several years, such as space rovers and humanoid robots.
However, the previously discussed techniques are often restricted to a feasible local optimal solution, which may be considerably smaller than the global solution in terms of size and complexity. Furthermore, when the environment is dynamic, the task becomes much more difficult to implement. Because of these drawbacks, such techniques are ineffective in complex situations.

4.2.3 Evolution Methods

The evolutionary process and methodological approach have developed over the last two decades, and "evolution algorithms" have been created as a result of these developments.
These algorithms have been extensively utilized in a variety of applications, including path planning and the management of complicated and dynamic obstacle hazards in the environment. As a result, many kinds of evolutionary methods have emerged, including:

Genetic Algorithm (GA)

The Genetic Algorithm (GA) is an optimization method that is based on genetic selection and genetic engineering. The search algorithm they're employing to come up with solutions to search problems is a hybrid of the survival of the fittest and a random data generator. After generating a random population, crossover, mutation, and selection are used to create feasible paths in GA [50], [78], [79],[80].
In order to reach the goal in a complex environment, chromosomes of varying lengths are needed to do the task. Ismail et al. [78] presented binary coding of various lengths of GA, in which the gene encodes both the direction of movement and the distance traveled by the organism during its movement. [79] created a path-planning algorithm based on genetics that has been adopted by populations, even those overcoming obstacles (and illegal paths also). Following that, the penalty function is assessed using an erroneous path sequence of procedures, which increases the computational burden as

well as the execution time of the algorithm.
For their part, Chaymaa et al. [80] used a genetic algorithm to obtain an optimum reaction that was controlled by the best available information. The cost of each ideal response is calculated, and the most cost-effective one is selected for the next phase. This process will be continued until the expenses are lowered to a manageable level.

Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) is another evolutionary technique that is widely utilized in path planning. Using the simplicity of a social program as a starting point, the method's idea aspires to simulate the unexpected movements of a flock of birds or a school of fish.
Several years of study into animal dynamics has resulted in the possibility of harnessing this behavior as a tool for success in a variety of fields. The PSO has the advantage of being simpler to use than the GA, as well as having a smaller number of repair parameters [7], [50], [81],[82], [83] compared to the GA. It has been proposed by Nasrollahy and Javadi [81] to use a PSO-based dynamic area system in which populations are created via defective paths and their validity is checked using a penalty function.
Gong et al. [84] also presented a model that coupled a multi-objective PSO with a mutation operator that was similar to that of genetic algorithms.
The wrong paths were created using the mutation operator in [82], while in [83], variables length were used to construct the improper pathways depending on the number of vertices of the polygonal obstacles. The binary PSO is used in conjunction with a genetic-like mutation operator to achieve direction optimization.

Ant Colony Optimization (ACO)

In 1992, Marco Dorigo proposed the above algorithm, which is a member of the colonial family of algorithms, into the swarm intelligence approach, which was then adopted by other researchers.
First, using the ants' actions as a guide, determine the shortest path from their current location to the food supply using a first method called seek

the best path in the graph (or search the best path in the graph). It has been shown that the ACO may be used to identify a global optimal path from a set of sub-optimal pathways [85] [86].
Using the ACO and Artificial Potential Field (APF), Mei et al. [85] presented a novel hybrid method for dynamic environments that combines the ACO with the APF. For the planning of the global path, the ACO has been employed, and the APF has been used for directing the robot. It is possible to fulfill the obstacle avoidance in both a global optimum and real-time manner using this hybrid method.
Nevertheless, the ACO's primary problems are the difficulties in obtaining fast solution convergence and the difficulty in applying to complicated maps, both of which are very big. As a result, Lee et al. [86] enhanced ACO and presented it using a potential field technique in order to get a fast convergence solution by changing ACO control settings in order to obtain a quick convergence solution. The advanced ACO updates the vector of the position vector using a conventional pheromone selectivity (a chemical produced by the ants) as opposed to the more advanced ACO.

4.2.4 Sampling-Based Methods

The development of sampling-based planners took place at a period when many complexity findings for the path-planning issue were known to be valid.
Reif [87] demonstrated that the generalized mover's problem, in which the robot is composed of a collection of polyhedra that are freely connected together at different vertices, is PSPACE-hard. Additional research into exact path-planning methods for the generalized mover's issue led to the development of an algorithm by Schwartz and Sharir [88] that was double exponential in the number of degrees of freedom of the robot. This method is based on a cylindrical algebraic decomposition of semi-algebraic descriptions of the configuration space [89].
Recently published work in real algebraic geometry has shown that the method is singly exponential [90]. Canny's method [91], which constructs a roadmap in the robot's configuration space, is also singly exponential in the number of degrees of freedom that the robot has.

In addition, Canny's work shown that the generalized mover's issue is PSPACE-complete [92], [93],[94],[95],[92],[96]. In path-planning research, the difficulty of path-planning algorithms for the generalized mover's issue has driven a number of recent advancements.
There were several of these, including the search for sub-classes of the problem for which complete polynomial-time algorithms existed (e.g., [97],[98]), the development of methods that approximated the free configuration space (e.g., [99],[100],[101][102]), heuristic planners (e.g., [103]), potential-field methods (e.g., [59],[104]), and the early sampling-based planners ((e.g., [104],[105],[106],[107], [108], [109],[110]).

However, because sampling-based methods only deal with configuration-to-configuration connections, they are easily adapted to higher dimensions. Because the amount of work required is proportional to the number of configurations that are added, sampling-based methods have the advantage of being simple to adapt to higher dimensions. This is in contrast to, for example, cell decomposition techniques or grid-based approaches, where the complexity increases exponentially with the number of dimensions used in the calculation.
When it comes to configuration-to-configuration connection, it's up to the local planner to decide whether or not a path can be traversed within the constraints of of the application, such as obstacles or differential constraints. Local planners often rely on a linear interpolation between two configurations, but their capabilities may be extended to cope with any kind of interpolation as well. When a local planner is included in the motion planning algorithm, it has the advantage of not requiring adaptation when the problem's dimensionality changes.
Because the local planner can simply be replaced with one that is capable of dealing with the new dimensionality without changing the algorithm's overall performance. Compared to the other classes of path planning algorithms, sampling-based approaches perform much better in the overall evaluation process.

In order to get the desired performance, sample-based methods make use of a graph or tree that is expanded by attempting to connect and add randomly chosen configurations to it in order to achieve it. The obvious implication of this is that sampling-based methods do not guarantee that

data is reliable and complete in the first place.
Nevertheless, it is possible to demonstrate that sampling-based techniques are probabilistically complete as the number of additional configurations approaches infinity. If at least one viable solution exists, the probability of finding that solution will converge to one as the number of new configurations approaches infinity. Despite the fact that they are not exact, many of these methods are demonstrated to be probabilistically complete in their application.
They will come up with a solution that has a fair probability of success, provided they have enough time. Because of advances in computational capacity in mobile robots, such algorithms have become widely used.

To find solutions to path-planning problems, sampling-based techniques use various ways for generating samples (i.e., collision-free configurations of the robot), connecting the samples with paths, and connecting the paths with the samples. There is a variation among these planning algorithms in terms of the heuristics they use to govern where and how the tree is grown. In certain algorithms, two trees are grown: one from the starting and one from the end.
Comparatively, while roadmap algorithms begin by sampling states and building a road map of the whole environment, which is then used to search for paths that connect a new configuration, expanding tree methods begin by building a tree of states that are connected by valid movements.
To create a collision-free path, the motion planner uses the random sampling technique, which performs random sampling in the free configuration space of the robot, then builds the connection graph from sample points, and then searches for the collision-free path using graph searching. For the most part, sampling-based Methods may be broadly divided into two categories: the Probabilistic Roadmap Method PRM [111] and the Rapidly Exploring Random Tree RRT [112].

Probabilistic RoadMap (PRM)

The probabilistic roadmap (PRM) (Kavraki et al., 1996) [113] is a well-known sampling-based motion planning technique that has shown the great potential of sampling-based approaches. It is possible to manage numerous

inquiries using this technique by building a roadmap in the state space, which means that there are not just one but several starting locations from which the queries are issued.
PRM takes full advantage of the fact that it is inexpensive to evaluate whether a single robot configuration is in Q free mode or not. PRM generates a roadmap (graph) in Q free by uniformly coarse sampling the nodes of the roadmap during what is known as the preprocessing phase, which is where the nodes of the roadmap are obtained. An additional node is only added to the graph if the sampled configuration does not fall inside the obstacle space; this is done in a stepwise manner.
In addition, it conducts a very fine sampling to get the roadmap edges, which are free paths between node configurations, from which it may derive the roadmap. During the second phase, each pair of nodes sampled is connected together to create the edges of the graph, which is then reported.
Planning queries may be addressed after the roadmap has been created by connecting the user-defined starting and goal configurations to the roadmap and solving the path-planning problem that has been presented. The solution is a network of straight segments, or arcs, which join in the nodes. Initial node sampling in PRM was carried out using a uniform random distribution to ensure that all nodes were sampled equally. Basic PRM is the name given to this planner.
Random sampling was shown to be very effective for a broad range of issues [114], [113], [115] and to guarantee the probabilistic completeness of the planner [113], [116]. In [114] it was also shown that random sampling is just a baseline sampling for PRM and that many alternative sampling methods are beneficial and bound to be efficient for various planning problems, as shown by the analysis of the planner. These sampling methods now vary from deterministic sampling schemes such as quasi-random sampling and sampling on a grid, which are used to identify regions of significance that are difficult to evaluate throughout the course of computations, to probabilistic sampling schemes such as importance sampling. PRM was originally intended to be used as a multiple-query planner.
PRM is modified when it is used to answer a single question: the initial and goal configurations are added to the roadmap nodes, and the roadmap is constructed gradually and only halted when the query at hand can be

answered successfully. The effectiveness of this method may be attributed to the fact that it is resilient in high-dimensional search fields. This method was the first to tackle motion planning problems with more than four dimensions, making it a significant milestone. However, because of its multiple query nature, PRM was originally intended for holonomic environments and requires thousands of connections between the nodes to function properly. As a result, it was necessary to develop a new class of sampling-based motion planning algorithms. Furthermore, PRM may not be the most efficient planner when dealing with single queries. The Rapidly-exploring Random Tree planner (RRT) [117], [118] is found to be a sampling-based planner that is remarkably successful in terms of planning efficiency. These planners have shown outstanding experimental performance.

Rapidly-exploring Random Tree (RRT)

Rapidly-exploring Random Trees (also known as Rapid Random Trees) are a class of probabilistic path planning that is designed to search non-convex high-dimensional spaces as efficiently as possible. RRT was developed by Steven M. LaValle and James Kuffner [117] to operate in non-holonomic environments, and it is still in use today [111].
In his book Planning Algorithms [33], LaValle provides a detailed presentation of the RRT method, demonstrating the method's potential in a wide range of applications and modifications. A single-query planning algorithm, RRTs, was introduced as a way of efficiently covering the space between q_{init} and q_{goal} [117], [118], [34], [119]. There is only one starting point for a Rapidly-Exploring Random Tree (RRT), and previous queries are not taken into account when searching the state space of the tree.
A single-query algorithm attempts to answer each query by starting from the inital configuration. As a result, RRTs are constructed incrementally in such a way that the expected distance between a randomly chosen point and the tree is reduced as quickly as possible. When it comes to the fundamental RRT algorithm, its incremental and single-query nature was specifically designed to be capable of incorporating differential constraints as well as managing non-holonomic environments. So RRT planners were initially developed for kinodynamic motion planning, where a single tree is built.

Although they are useful for kinodynamic planning problems, their applicability is much broader. When it comes to path planning problems involving obstacles, differential constraints, as well as robot kinematic and dynamic constraints, RRTs are particularly well suited. They can be thought of as a technique for generating open-loop trajectories for nonlinear systems with state constraints that can be applied to any nonlinear system.

Another important notion to note is that a random sequence generated by the RRT method is almost certainly dense. When used to its maximum extent, RRT will densely cover the state space; as a result, the class of RRT is sometimes referred to as Rapidly-Exploring Dense Tree (RDT).
When the number of samples drawn approaches infinity, the trees that are constructed resemble space-filling curves in their appearance. RRT, on the other hand, is comprised of shorter paths, and there is a path from each node in the tree to the starting, or root, node of the tree.

The fundamental concept of RRT is to explore the configuration space in a non-greedy manner in order to construct a graph that connects the start and goal configurations. In the motion planning problem, RRT-based algorithms build a space-filling tree by obtaining a sample from the space in order to identify a goal while taking into account the constraints of the problem.
Specifically, this is accomplished by beginning with the start configuration and continuously inserting a randomly selected configuration within a specified distance from the closest configuration in the graph, after which an attempts are made to connect the two configurations together. Eventually, as the algorithm proceeds through the configuration space, it will approach the goal with a high probability of being successful.
When this occurs, the process of adding nodes is halted, and the graph may be explored in search of the solution.The fundamental concept of RRT is to explore the configuration space in a non-greedy manner in order to construct a graph that connects the start and goal configurations.
In the motion planning problem, RRT-based algorithms build a space-filling tree by obtaining a sample from the space in order to identify a goal while taking into account the constraints of the problem. Specifically, this is accomplished by beginning with the start configuration and continuously inserting a randomly selected configuration within a specified distance from the closest configuration in the graph, after which an attempts are made to connect

the two configurations together.
Eventually, as the algorithm proceeds through the configuration space, it will approach the goal with a high probability of being successful. When this occurs, the process of adding nodes is halted, and the graph may be explored in search of the solution.

Despite the fact that the RRT algorithm has been demonstrated to be probabilistically complete [118], an RRT alone is commonly insufficient to solve a planning problem and find an optimal solution. This component may therefore be seen as an add-on that can be used in the development of a wide range of planning algorithms [120]. The use of additional algorithms for smoothing the result trajectory or other enhancements to the solver's performance is highly recommended to improve its convergence to the goal (for example, optimizing random sample generation to generate biased samples while taking into account the goal's configuration or another meaningful heuristic). Therefore, the development of RRT-class variations has been accelerating since 2010, and the class is increasingly being used in robotic tasks.

As we will see in section 4.3 on algorithms, there are a number of different modifications that can be made to this algorithm, resulting in a number of different variants that attempt to improve on some of the shortcomings that we will encounter along the way as well.

Comparing PRM to RRT

The most significant distinction between RRT and PRM is that RRT employs non-greedy exploration, which sacrifices speed for a greater likelihood of adding a new node to the graph. PRM establishes a direct connection to the new candidate node, while RRT takes a more conservative approach. With the introduction of this characteristic comes the introduction of a new notion known as Voronoi Bias, which may be defined as the likelihood of exploring a previously unknown region of space being proportionate to the size of the chunk of space in question.
This can be visualized by imagining a graph and dividing all of the space according to a Voronoi diagram using the nodes of that graph as its seeds.

Due to the fact that each cell has precisely one node and therefore bigger cells are less explored, it may be argued that a larger cell in this figure represents an undiscovered area of space.
A cell's size divided by the entire space determines the chance of selecting a random configuration from inside it, thus cells that have been less examined have a greater probability of containing the next random configuration and, as a result, have a higher probability of being explored. A Voronoi bias is the term used to describe this phenomenon.
Furthermore, since RRTs are created using non-greedy exploration, the graph of an RRT usually includes shorter edges than the graph of a roadmap created with PRM. Considering that in certain instances longer trajectories with fewer edges are favored than shorter ones, this may be considered a disadvantage of RRT.

Another distinction is that PRM usually connects to a newly added node several times. Because PRM is a multiple-shot method, which means that a roadmap may be utilized many times because it is not committed to a particular combination of start and goal configurations, as opposed to RRT, which is a single-shot approach.
In addition to the fact that RRT attempts to link the start and goal configurations, another benefit of RRT over PRM is the fact that RRT may be utilized by adding a bias towards the goal in the sampling of the new randomly selected nodes, or even growing two graphs towards each other [121].
However, this is more difficult with PRM since a roadmap is, by definition, a multi-shot method, which cannot be skewed towards a goal because a multi-shot technique does not take a goal configuration into account. Having a goal bias is a heuristic that may be used to decrease the amount of time it takes to find a solution [122].
Despite the fact that both RRT and PRM have shown the ability to deal with greater dimensionality, RRT has the benefit of being a single-shot method, which implies that the graph may be steered in one direction. Since the graph of most RRTs is a tree, this is already true, but it becomes much more essential when applying RRT to kinodynamic or other time-involved path planning problems [123], [124],[34], even though PRM has been used for these types of problems as well [125].

Another significant feature of sampling-based planners is their ability to attain a certain level of completeness in their solutions.
For a planner to be considered complete, it must always provide appropriate response to the path-planning problem within an asymptotically limited time. Due to the enormous combinatorial complexity of complete planners, they cannot be implemented in reality for robots with more than three degrees of freedom in the real world. However, there are somewhat weak, but still significant, versions of completeness, which can be described as follows: if a feasible solution path exists, the planner will ultimately find it.
This kind of completeness is referred to as probabilistic completeness for those sample-based planner where the sampling is done in a heuristic manner.
However, if the sampling is deterministic, such as quasirandom or sampling on a grid, this kind of completeness is referred to as resolution completeness with regard to the sample resolution. The Randomized Path Planner (RPP) [104], [126], one of the first sampling-based planners, was proven to be probabilistic complete, establishing a precedent for sampling-based techniques in general. It has also been shown that PRM is probabilistically complete [127], [114],[128], [129], [130],[131], as we as RRT planners.

In general, we may list the following characteristics of RRT that have made it appealing for a range of motion planning problems, as follows:

- The RRT method almost-sure will converge to a suboptimal path.
- RRT is probabilistically complete under the assumptions of the general path-planning problem formulation. In other words, the chance of finding a solution increases progressively as time goes on.
- The RRT algorithm does have a few parameters and heuristics to work with.
- In the case of a node $v \in V$, there is always a path between it and x_init, since it is required that $x_init \in V$ and that each node has only one parent.
- RRTs are quickly expanding into previously unknown areas of X_free.

- The distribution of V tends to converge towards the sampling distribution, resulting in a basically complete filling of the search space with nodes.

The first property of RRT tells us that it will almost certainly converge to a suboptimal path. Despite the fact that the RRT algorithm is simple to implement, explores its space quickly, and is capable of dealing with high-dimensional environments, it is not without its shortcomings. RRT is suboptimal, which can be translated into the assertion that the generated graph does not improve on the previously computed solution. As a result, with the various existing constraints represented in the search space, it is possible that the RRT method will become stuck with the first solution that is computed.
However, despite the fact that several algorithms have been proposed to guide the growth of an RRT towards finding an optimal path, using heuristics [122] or simply by exploiting the Voronoi diagram of the search, the quality of the solution found has fallen short, particularly when an optimal path to the goal is required.
The problem is primarily caused by RRT's failure to take into account the costs of the path. However, due to the rapid exploration of the state space and the uniform sampling distribution, the paths in RRT are very close to the true shortest paths, resulting in the identification of a suboptimal solution at some point in the process.
Furthermore, because the state space is explored uniformly, it is possible that the system will reach the goal at a slow (or premature) pace. Furthermore, with each iteration, the set of nodes grows in size, which means that the search for the nearest neighbour becomes increasingly time-consuming and expensive. Numerous solutions, such as the use of an alternative data structure (a Kdtree [132]), rather than a naive implementation, have been proposed, including the use of Kdtree [132].
Additionally, excellent results have been obtained using an approximate nearest neighbour search method ([133]; [134]; [112]) instead of taking a methodical approach According to [122], the growth of an RRT graph is guided by the evaluation of the path to a node with the least amount of cost, as well as the use of a quality measure that is defined for each node.
Additionally, instead of computing only the nearest neighbour from V, the

k-nearest neighbours can be used to provide a more accurate evaluation of the neighbours. At the end of the day, even after the path costs are taken into account and even after the guidance produces better results, this RRT variant retains its inherent probabilistic sub-optimality.

These characteristics, particularly the probabilistic completeness, are what allows for viable planning, which has been used in a variety of applications throughout the years [33]. Some examples of this include a spacecraft operating in a simulated world that must travel over hazardous terrain while also taking gravity into consideration [119]. Because of all of the characteristics listed above, RRT is not limited to the field of robotics or path planning; it can also be used as a modeling tool for Diffusion Limited Aggregation (DLA) [135]; [119], in which particles perform what appears to be a random walk, resulting in a tree that is similar to the tree generated by RRT.
RRT shown positive prospects in a broad range of challenges, including robotic arms, pianos in motion, and satellite docking maneuvers, among others [119]. RRT was shown to be asymptotically suboptimal despite these findings [136], and it was found to be unsuitable for certain situations where an optimum solution was required. As a consequence, additional development of the fundamental RRT technique was proposed, which resulted in the introduction of RRT* and RRT$^\#$ [112],[137].

4.3 RRT-based Algorithms

It is the intention of this section to discuss the algorithms of the RRT class, as well as some of their variants and the characteristics that they exhibit. RRT, RRT*, and RRT$^\#$ are the three primary variations of the RRT class that are discussed. However, we need to formally define the problem of path-planning first.

4.3.1 Problem Formulation

A motion planning problem for RRT, RRT*, and RRT$^\#$ is often characterized by the following characteristics: the following description:

1. The state space ,or search space or configuration space, has the definition $X \subseteq \mathbb{R}^d$, where d is the dimension of the state space. X is made up of two parts: X_{free}, which represents the search area that is free of obstacles, and X_{obs} which represents the search area that is populated with the obstacle.

2. The starting configuration from which the RRT grows the graph is represented by x_{init}, and the goal region is represented by X_{goal}, which both must be found inside the X_{free} set. Although the goal configuration may consist of a single configuration, it is preferable to add a little region surrounding the goal in the simulation.

3. $\epsilon > 0$ denotes the predefined step-size, which corresponds to the maximum distance between any existing node and a newly connected one

4. The maximum number of iterations for an RRT is denoted by the letter K.

given the above definitions, the path planning algorithm aims to solve the following problem:

- find a path σ^* starting from $\sigma(0) = x_{init}$ ending in $\sigma(s) \in x_{goal}$, with minimum cost c between x_{init} and x_{goal} such that $c(\sigma^*) = min\left\{c(\sigma)|\sigma \in X_{free}\right\}$.

4.3.2 Basic RRT Algorithm

To summarize, the RRT algorithm's main goal is to build a space-filling tree by incrementally biasing the search towards empty spaces. Additionally, with each iteration, the algorithm attempts to add potential new nodes in the graph/tree moving away from the nodes that have already been added to V in previous iterations. In the beginning, a specified point in the configuration space (x_{init}) is added to V, and the set of edges E is empty due to the fact that only one node has been evaluated so far (x_{init}). The *Sample* function is used to sample a point x_{rand} from a uniform distribution. Hence, this randomly generated sample is used to search for the nearest node $x_{nearest}$ in the V with respect to x_{rand}.

Finally, a new node x_{new} is identifed by reducing the distance between x_{rand} and $x_{nearest}$ to its shortest distance respecting the predefined step-size ϵ. In other words, x_{new} is the new node created by executing one step with a distance less or equal to ϵ in the direction of x_{rand}, and the result is a new potential node x_{new}. If the global constraints are satisfied, then the new node x_{new} is added to V and a new edge from the nearest neighbour to the new node is added to E. If, however, the global constraints are not satisfied, then the new node x_{new} is removed from V and the new edge from the nearest neighbour to the new node is consequently removed from E.

Algorithm 1: RRT

```
Input: initial node x_init, max nodes K ← ℕ
V ← {x_init}; E ← ∅;
for i = 0 to K do
    x_rand ← Sample(i);
    x_nearest ← Nearest(V, x_rand);
    x_new ← Steer(x_nearest, x_new);
    if ObstacleFree(x_nearest, x_new) then
        V ← V ∪ {x_new};
        E ← E ∪ {(x_nearest, x_new)};
    end
end
return (V, E)
```

4.3.3 RRT* Algorithm

RRT (Rapidly Exploring Random Tree) is a method that is similar to RRT in that it rapidly explores a random tree. RRT* is a path-planning algorithm that connects two robot configurations (stat and goal) through form a tree that navigates the free configuration space X_{free} by progressively adding randomly chosen robot configurations (nodes) from the obstacle-free set X_{free}. The RRT* algorithm operates in a similar way as the RRT algorithm in that it searches the configuration space gradually, with a bias for traversing open spaces. RRT*, however, differs from the standard RRT in that it is an incremental sample-based approach that splits the path planning problem into two phases: the exploration phase and the exploitation phase.

Consequently, an initial path is found very quickly, and the path is then optimized as the execution continues [37], resulting in a rapid optimal path realization. It is because of this that the RRT* algorithm has an advantage over the RRT algorithm in that it has a high chance of ultimately arriving at an optimal solution [112].
This is accomplished via the application of a cost function in the assessment of path segments between any two nodes in the tree. Specifically, in order to improve the rate of convergence of the RRT algorithm, RRT* was proposed [112]. This method is based on the RRG 11 algorithm [137], which is a graph-based algorithm with the feature of asymptotic optimality, which RRT* inherits.
The introduction of the path cost function into the RRT* algorithm, which results in significantly improved performance than the original RRT method, may be ascribed to this improvement. The cost of a path is taken into consideration in order to choose the node with the lowest cost from x_{init} among the k-nearest neighbours defined by a region with radius r centered around the node in question. To compare path costs, the idea behind RRT* is to construct an array of surrounding nodes near the new node that are built on the basis of RRT, that is, walking through these surrounding nodes to determine whether or not a better path exists and, if so, replacing the existing path with the better path in order to improve the existing search tree. Indeed, new sample nodes are not connected to the closest node, but rather to the least expensive of the nodes in the network. The cost of the connection, specified by a cost function c(t), is calculated for each additional node that is added. This way, each node in the graph is labeled with a cost that is proportional to the distance between it and the parent node. Thus, with each new connection, the overall cost of the graph (i.e. the total of each node's cost) is updated, and the graph is considered updated.
Furthermore, the RRT* method also takes into consideration collision avoidance as criterion to decide wether to add a new node to the existing tree or not. Hence, adding a new robot configurations to the growing tree is only done if the connection to the parent node is collision-free and with a minimal cost.
To implement this logic, RRT* introduces two new procedures that examine the cost of the path. In the case of a new node, the *Parent* procedure selects the

appropriate parent from within a specified radius, and the *Rewire* procedure restructures the tree in order to identify less expensive paths that take into account the new node. Eventually, after a predefined maximum solve time or after a predefined number of nodes are sampled, the algorithm terminates delivering a graph that connects the start and goal configurations with the lowest cost path, as determined by the cost function c.

The RRT* algorithm, as a result, behaves in a similar manner to the RRT algorithm in the sense that it generates its tree in a similar manner to RRT; at each iteration, the values assigned to the variables x_{rand}, $x_{nearest}$, and x_{new} are assigned in the same manner as it has been discussed previously for RRT only this time considering the cost function. The RRT* algorithm, therefore, contains three novel processes that enable it to achieve its superior performance:

- The Near function: A function that computes a set of nodes $V' \subseteq V$ containing the closest neighbours within a region of radius r centered around the node x. In order to ensure the almost-sure convergence to an optimal solution characteristic of the algorithm, the radius r is computes according to the following equation:

$$r_i = min\left\{ (\frac{\gamma}{\zeta}\frac{log(n)}{n})^{\frac{1}{d}}, \eta \right\} \tag{4.2}$$

  Where η is a predefined maximum radius, n is the length of V , ζ_d is the volume of the neighborhood region of radius, $\gamma > 0$ a constant, such that

$$\gamma \geq 2^d(1 + \frac{1}{d})\mu(X_{free}) \tag{4.3}$$

  where mu denotes the size of the free space available in the configuration space, whereas r_i can be fixed with a predefined constant at the expense of the asymptotic optimality.

- The Parent function: A function used to select the parent node which has the lowest cost-to-go from the new candidate node x_{new}, from the set of neighboring node.

- The Rewire function: A function used to remove the discarded nodes which were found to have higher than the minimum cost.And consequently adds the winning candidate node to the tree

Algorithm 2: RRT*

```
Input: initial node x_init, max nodes K ← ℕ
V ← {x_init}; E ← ∅;
for i = 0 to K do
    x_rand ← Sample(i)
    x_nearest ← Nearest(V, x_rand)
    x_new ← Steer(x_nearest, x_new)
    V ← V ∪ {x_new}
    if ObstacleFree(x_nearest, x_new) then
        X_near ← Near(V, x_new)
        X_nearest ← Parent(X_near, x_nearest, x_new)
        E ← E ∪ {(x_nearest, x_new)}
        E ← Rewire(X_near, E, x_new)
    end
end
return (V, E)
```

Algorithm 3: Parent

```
Input: X_near, x_nearest, x_new
for x_near ∈ X_near do
    if ObstacleFree(x_near, x_new) then
        c' ← x_near.cost + c(Line(x_near, x_new))
        if c' < x_new.cost then
            x_nearest ← x_near
return x_nearest
```

When the new randomly generated node x_{new} is available, the Near function computes the neighboring subset the contains the closest nodes in the existing tree V to the values of the x_{new}. After which, a parent is selected from among this neighborhood subset by computing the node x_{near} that is at a distance bounded by the predefined size-step and having the minimum path cost to the new node x_{new}. Afterwards, the tree is rewired in order to identify a possibly cheaper path towards x_{new}. This is accomplished by

Algorithm 4: Rewire

```
1: Input: X_near, E, x_new
2: for x_near ∈ X_near do
3:     if ObstacleFree(x_near, x_new) and x_near.cost > x_new.cost +
        c(Line(x_near, x_new)) then
4:         x_parent ← x_near.parent
5:         E ← E \ {(x_parent, x_near)}
6:         E ← E ∪ {(x_new, x_near)}
7: return E
```

analyzing the present cost of the path to x_{near} and comparing it with the cost of the path to x_{near} in the event that it would be connect through x_{new}. It turns out that the Parent and Rewiring functions gradually improve the path cost of RRT* as the number of nodes in the tree grows, resulting in asymptotic optimality.

Properties of RRT*

With the introduction of the previously discussed functions, the RRT* algorithm adds the following properties on top of the properties it inherts from the basic RRt algorithm mentions earlier:

- RRT* asymptotically converges to an optimal solution
- RRT* retains the same time requiremets (computational complexity) of the basic RRT; the generation of the tree in both algorithms costs O(n log n) while the path query part of the algorithm costs (Karaman & Frazzoli, 2010) and (Karaman & Frazzoli, 2011).

In order to evaluate the asymptotic optimality of the RRT*, Professor Emilio Frazzoli and colleagues [137] utilized the algorithm to park an autonomous forklift in front of a truck while the vehicle avoided obstacles. Because of an obstruction along the track, the forklift had to circle around the truck in order to locate a suitable parking space.
In this planning problem, the RRT successfuly found a path that was suboptimal. In order to improve the solution reaching the optimality, RRT* was

implemented, which in turn corrected the loop that RRT generated in order to locate a shorter path to park the forklift. Hence, The RRT's jumbled and chaotic pathways were adjusted by the team implementing RRT*.

Limitations: Despite its advantages of the basic RRT algorithm, the RRT* suffers a number of limitations nevertheless. The most obvious one is the can only reach an optimal solution after an infinite number of iterations ideally. The means that, in practice, the RRT* will converge to the optimal solution in an infinite time. Nevertheless, because RRT* continues to search the whole state space, the rate of convergence is still slow.
There are several variants of RRT*, each of which attempts to improve on the convergence rate by looking for shorter paths utilizing techniques like the triangle inequality and intelligent sampling. In some variants, some nodes are employed as beacons, and the growth is skewed even more towards the goal configuration. However, such improvements comes always at the expense of additional memory requirements.

4.3.4 RRT# Algorithm

RRT# was proposed by Arslan et al. [138] as a modified version of the incremental Rapidly-exploring Random Graph (RRG) that guarantees the generation of a globally optimum graph in the search space.
For the underlying graph, a spanning tree with a root at the starting node is generated, that guarantees all the information needed to identify the the best possible path connecting the starting node to the goal.This information set includes the cost-to-come values for a subset of vertices that are potential candidates for the optimal path, including the goal.

Such informative graph gives rise to the main advantages of the RR# over other variants of RRT. Given this informative graph, the algorithm is able to provide the shortest path possible at every iteration given the nodes that have been identified so far.
The fact that this kind of data is available on top of the network also allows the algorithm to classify the current vertices according to their probability of becoming a component of an optimal path at some point in the future.
As a result, such classification may be useful in increasing the convergence

rate by prioritizing the assessment of the vertices with the highest probability first during the exploitation phase of each iteration. The algorithm starts with the exploration phase which implements the extension process of the RRG algorithm, followed by the exploitation task, which implements the Gauss-Seidel version of the BellmanFord algorithm.

Exploration: A random sample is selected from the obstacle free set X_{free} at this step, which is then utilized to expand the current graph.

Exploitation: This step involves re-calculating the cost-to-come values of the existing vertices in order to reflect the information from the newly added vertex on the whole graph at the same time. In addition, the priority queue is updated with the most promising vertices. RRT* The RRT$^\#$ algorithm ensures that the maximal amount of information available at each iteration is used by repeatedly performing the exploration and exploitation stages for each new sample. One of the most significant differences between RRT$^\#$ and RRT* is that, in the RRT* algorithm, all vertices in the tree are computed based on their cost-to-come value, whereas in the RRT$^\#$ algorithm, the vertices are classified into various types based on their estimated cost-to-come value. The RRT$^\#$ algorithm convergence rate was significantly improved as a result of the innovation of calculating the cost-to-come value one step ahead of the current iteration. Based on the values of its $(\mathrm{g}(v), \mathrm{lmc}(v))$ pair, each vertex v is categorized into one of four categories in the RRT$^\#$ algorithm. These are as follows:

- stationary with finite key value $(\mathrm{g}(v) < \infty, \mathrm{lmc}(v) < \infty$ and $\mathrm{g}(v) = \mathrm{lmc}(v))$;
- stationary with infinite key value $(\mathrm{g}(v) = \infty, \mathrm{lmc}(v) = \infty)$;
- nonstationary with finite key value $(\mathrm{g}(v) < \infty, \mathrm{lmc}(v) < \infty$ and $\mathrm{g}(v) \neq \mathrm{lmc}(v))$;
- nonstationary with infinite g-value and finite lmc-value $(\mathrm{g}(v) = \infty, \mathrm{lmc}(v) < \infty)$.

This categorization is used in the Replan method in order to iteratively update the graph.

When compared to the other RRT-variants, particularly the previously presented RRT* algorithm, the RRT$^\#$ method, as given in 1, has a similar overall structure to the RRT$^\#$ algorithm. Starting from an initial configuration x_{init}, the algorithm incrementally expands the graph, as described in 2, navigating the obstacle-free space X_{free} through randomly sampling x_{rand} from the configuration space. Once this is done, the whole existing graph is updated in order to optimize for cost-to-come of all the nodes in the graph while taking into consideration the extention made by the Extend function. It is important to note that the steps mentioned above form a single iteration, which will be continued until the specified number of iterations has been reached or the desired goal configuration has been achieved. The resultant graph will include the information about the lowest cost path information for the promising vertices and v^*_{goal}.

The*Extend* procedure given in 2 is responsible for extending the existing graph in the direction of the newly sampled x_{rand}. The extension computes $x_{nearest}$ denoting the nearest node in graph to x_{rand}, then defines x_{new} with a maximum predefined distance from $x_{nearest}$ along the path to x_{rand}. Upon identifying the set of the neighboring node to x_{new}, the parent node is selected to attain the minimum cost-to-come. In the end of this procedure, the new vertex is decided to be inserted in the priority queue or not based on its stationarity in the UpdateQueue procedure.

Following that, the Replan process, described in 3, updates the graph by propagating the g-value of the selected vertex to all of its neighbors, utilizing the list of promising vertices in the priority queue.

4.4 Motion Planning Constraints

Path planning techniques may be divided into two categories based on the nature of the constraints: kinematic planners and kinodynamic planners. The kinematic planners take into account only the Kinematic Constraints of the rigid bodies while creating their models [139]. These may represent joint constraints of a (mechanical) system, or they could be obstacles in the system. The kinodynamic class, on the other hand, takes into account the dynamic constraints imposed on the rigid bodies. In most cases, they are second order

Algorithm 5: Body of the RRT$^{\#}$ Algorithm

```
RRT#(x_init, χ_goal, χ)
    V ← {x_init}; E ← ∅;
    G ← (V, E)
    for k = 1 to N do
        x_rand ← Sample(k);
        G ← Extend(G, x_rand);
        Replan(G, χ_goal);
    (V, E) ← G; E' ← ∅;
    foreach x ∈ V do
        E' ← E' ∪ {(parent(x), x)}
return T(V, E)
```

Algorithm 6: Extend Procedure

```
Extend(G, x)
    (V, E) ← G; E' ← ∅;
    x_nearest ← Nearest(G, x);
    x_new ← Steer(x_nearest, x);
    if ObstacleFree(x_nearest, x) then
        Initialize(x_new, x_nearest);
        χ_near ← Near(G, x_new, |V|);
        foreach x_near ∈ χ_near do
            if ObstacleFree(x_near, x_new) then
                if lmc(x_new) > g(x_near) + c(x_near, x_new) then
                    lmc(x_new) = g(x_near) + c(x_near, x_new);
                    parent(x_new) = x_near;
                E' ← E' ∪ {(x_near, x_new), (x_new, x_near)};
        V ← V ∪ {x_new};
        E ← E ∪ E';
        UpdateQueue(x_new);
return G ← (V, E)
```

Algorithm 7: Replan Procedure

```
1:  Replan(G, χ_goal)
2:      while q.findmin() ≺ Key(v*_goal) do
3:          x = q.findmin();
4:          g(x) = lmc(x);
5:          q.delete(x);
6:          foreach s ∈ succ(G, x) do
7:              if lmc(s) > g(x) + c(x,s) then
8:                  lmc(s)= g(x) + c(x,s);
9:                  parent(s) = x;
10:                 UpdateQueue(s);
```

differential constraints operating on the body, such as the torques, velocities, or accelerations [33].

Typically, the kinematic constraints are classified into two types: holonomic constraints and non-holonomic constraints.
A holonomic kinematic constraint is one that can be described by: $f(q_i, t) = 0$, where q_i represents the spatial coordinates of the point, and t denotes the time. Whereas, any other kinematic constraint that is not possible to put in the previous form is said to be non-holonomic. These non-holonomic restrictions may result from the velocities of the rigid bodies, but they could also result from limitations on the state space of the rigid bodies.
For the most part, issues described inside non-holonomic systems are those in which mechanical or differential restrictions must be taken into account. More importantly for our motion planning purposes is the fact that the kinematic approach disregards forces that cause the inhibition of motion.

On the other hand, the kinodynamic method is more comprehensive than the kinematic approach in that it incorporates kinematic constraints as well as the dynamic constraints which might represent dynamic obstacles or intrinsic constraints on the vehicle dynamics.
However, when three or more dimensions are taken into consideration, the class of kinodynamic problems has been demonstrated to be NP-hard [32]. For high dimentional problems with non-holonomic constraints, algorithms that attempt to approximate solutions, such as PRM, does not give satisfac-

tory results. Despite this, there have been methods that use PRM to address issues in non-holonomic systems that have been successful [140], [141], [142].

In order to overcome these challenges, kinodynamic motion planning algorithms may be used to fulfill both the kinematic and the dynamic requirements of the motion planning model model [32]. Such kinodynamic algorithms are typically formulated as a two-point boundary value problem in the dynamic state space of the robot system. To that end, a significant number of kinodynamic methods make use of different basic curves to establish a reference path, including Bezier curves [143], harmonic potential fields [144], and learning approaches [145] to name a few. Several techniques of determining the shortest path between configuration pairs for wheeled vehicles are presented in [33]. There are many different techniques for finding optimum paths for a basic vehicle model, including Dubins curves and the Dubins car, which are both named after Dubins. [146] extend this model even further by include all three dimensions in three dimensions in three dimensions, allowing them to discover the optimum paths for a basic aircraft model.

One of these primitive curves is the Dubins curves defines by Dubins [147]. Dubins curve defines the shortest path connecting two configurations of a wheeled vehicle with constant velocity model through a combination of three primitive motions S, L and R.

The S motion drives the car straight forwards, while the L and R motions turns the car left and right, respectively. Hence, the shortest paths can be formulated as a set of specific sequences of these primitive motions. The possible sequences are: LRL, RLR, LSL, RSR, LSR, RSL

Chapter 5

The MP-RRT$^{\#}$ Algorithm

In this chapter we present the motion planning problem definition and the proposed motion planning solution (MP-RRT$^{\#}$).

5.1 Problem Definition

As stated in the section 1.3, the study performed in this chapter aims to develop an algorithm for kinodynamic motion planning for an Unmanned Aerial System (UAS). As we go through this Chapter, we will establish the UAS model, then provide the technical tools that will be utilized to construct the suggested motion planner, and lastly we will describe the proposed solution in more detail.

5.1.1 UAS Model

Unmanned Aerial Vehicle (UAV) systems have been introduced to the market in recent years, and there is a vast variety of options available. Fixed-wing aircraft vehicles and multi-rotor aircraft vehicles are the two most common types of aviation vehicles on the road today. Fixed-wing unmanned aerial vehicles (UAVs) often have longer flight endurance capabilities and can cover large areas in a single trip, according to the [148]. While multi-rotor aircraft are more expensive than fixed-wing aircraft, they offer a number of

advantages over them, the most significant of which being the ability to take off and land vertically. Furthermore, the multi-rotor has the capability of hovering, similar to that of a helicopter. More importantly, due to the large number of rotors on it, it is capable of supporting far larger weight loads when compared to its own weight [149]. A tiny-sized multi-rotor is also nimble, highly maneuverable, and naturally more stable due to the design of numerous rotors with counter rotating props, which reduce the need for a tail rotor and results in the aircraft being intrinsically more stable due to its small size. As a consequence of these advantages, a growing number of practical applications for the multi-rotor have emerged, and the multi-rotor has risen to the status of a key research focus. Hence, the UAS vehicle that will be considered in this study will be a multi-rotor type.

UAS Sysytem Model

We apply the same method as in [150] to describe the vehicle location and speed with respect to the world fixed inertial frame $\mathbb{W}$. Further we define a body fixed frame B attached to the UAS CoG. Both reference frames are of the right-handed kind. In particular, the location of the UAS Center of Gravity CoG in the inertial frame $P_B \in R^3$, the vehicle velocity in the inertial frame v, the vehicle orientation $R_{WB} \in SO(3)$, and the body angular rate ω are used to characterize the vehicle configuration. We define the velocity vector as the derivative of the position vector at a particular point in time along the model's trajectory. According to geometrical principles, given a point in three-dimensional space, the following equation may be defined:

$$P_B = \begin{pmatrix} x_B \\ y_B \\ z_B \end{pmatrix} = v_{AB} \cdot t + P_A = \begin{pmatrix} v_{AB,x} \\ v_{AB,y} \\ v_{AB,z} \end{pmatrix} \cdot t + \begin{pmatrix} x_A \\ y_A \\ z_A \end{pmatrix} \tag{5.1}$$

A particular representation of the Euler angles, ardan angles convention, will be used to describe the orientation of the UAS in space. Keeping in mind

that the vehicle is specified in a three-dimensional right-handed reference frame, the rotations may be designated as follows:

- *Roll ϕ:* Rotation around the vehicle longitudinal axis;
- *Pitch θ:* Rotation around the vehicle transverse axis;
- *Yaw ψ:* Rotation around the vehicle vertical axis passing through its center of mass.

It is necessary to perform rotations in order for information to be appropriately translated from the inertial world frame to the vehicle body frame B. From a geometric standpoint, these transformations are done by pre-multiplying the drone body frame orientations by the rotation matrices before to applying the transformations that are in concern.

The following are the most often seen UAS maneuvers:

- *Hovering:* When an unmanned aerial system (UAS) hovers at a certain altitude, it is referring to the condition of stationary flight at a specific height, suggesting that the drone maintains a constant altitude without altering its angle of attack. In order to do this, all four rotors must be activated to spin at the same speed, with each rotor compensating for a fourth of the drone's total weight.
- *Move Up and Move Down:* It is necessary to change the spins of all four rotors in order to raise or reduce the altitude of a UAS; raising the rotational speeds of the four rotors while in an ascension maneuver and reducing them while in a descent maneuver are examples of such variations.
- *Roll:* When the aircraft's rotors are rotated at different speeds on each side of its lateral axis, it is possible to perform roll maneuvers. For the drone yaw to be unaltered, the UAS rotors on the same axis must be synced so that whenever the speed of one drone rotor is increased in order to perform the needed maneuver, the speed of the opposing drone rotor is reduced.

- *Pitch:* The pitch maneuver is conducted in the same manner as the roll maneuver, with the exception that the rotational speeds of the rotors are raised or lowered along the longitudinal axis of the aircraft.
- *Yaw:* A pair of rotors that are opposite each other's rotational speeds are raised in order to conduct this move, whereas the rotary speeds of the two other rotors are reduced. Hence, a rotation about the vertical axis occurs as a consequence of the entire imbalance of moments present.

As a consequence, the propellers are responsible for the vast majority of the forces acting on the vehicle. Each propeller generates thrust that is proportional to the square of the propeller rotation speed, as well as angular moment, in a proportionate manner. When we examine at the i_{th} propeller, we can denote the generated thrust and moment thrust using the variables $F_{T,i}$ and M_i as follows:

$$F_{T,i} = k_n n_i^2 e_z \tag{5.2}$$

$$M_i = (-1)^{i-1} k_m F_{T,i} \tag{5.3}$$

where n_i denotes the i_{th} rotor velocity, k_n and k_m denote positive constants, and e_z denotes a unit vector pointing in the direction of z.

To account for certain factors which becomes significant in some scenarios, it is necessary to represent additional force which impact the UAS dynamics. Specific examples of these impacts include the flapping of the blades and the resulting drag as a consequence of this. The sum of these aerodynamic factors has a different force acting as a damping force on the UAS, which we need take into consideration while developing our model of the UAS. Typically, these forces are combined into a single lumped drag coefficient, denoted by the symbol k_D. As a result, the aerodynamic force $F_{aero,i}$ is:

$$F_{aero,i} = f_{T,i} K_{grag} R_{WB}^T v \tag{5.4}$$

where $K_{drag} = diag(k_D, k_D, 0)$ and $f_{T,i}$ is the z component of the i_{th} thrust force.

In addition to the forces acting on the UAS, it is plausible to simulate the velocity of the vehicle using the equations shown below:

$$P^{\cdot} = v \tag{5.5}$$

$$v^{\cdot} = \frac{1}{m}\left(R_{WB}\sum_{i=0}^{N_r} F_{T,i} - R_{WB}\sum_{i=0}^{N_r} F_{aero,i} + F_{ext}\right) + \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} \tag{5.6}$$

$$R_{WB}^{\cdot} = R_{WB}[\omega] \tag{5.7}$$

$$J\omega^{\cdot} = -\omega \times J\omega + A \begin{bmatrix} n_1^2 \\ ... \\ n_{N_r}^2 \end{bmatrix} \tag{5.8}$$

m denotes the mass of the vehicle. F_{ext} denotes the external aerodynamic forces operating on the vehicle. The inertia matrix is denoted by J, while the allocation matrix is denoted by A, and the number of propellers is denoted by N_r.

Typically, while building the higher-level attitude controller, it is critical to take into consideration the inner loop system of the attitude model in order to achieve better control quality when following a predefined trajectory. In accordance with the [150], the inner-loop attitude dynamics are then expressed as follows for our purposes:

$$\phi^{\cdot} = \frac{1}{T_\phi}(k_\phi \phi_{cmd} - \phi) \tag{5.9}$$

$$\theta^{\cdot} = \frac{1}{T_\theta}(k_\theta \theta_{cmd} - \theta) \tag{5.10}$$

$$\psi^{\cdot} = \psi_{cmd}^{\cdot} \tag{5.11}$$

where k_ϕ denotes the roll angle gain, k_θ denotes the pitch angel gain. T_θ and T_ϕ are the time constants for the pitch and the roll respectively.

ϕ_{cmd} and θ_{cmd} are the commanded roll and pitch angles and $\dot{\psi}_{cmd}$ is commanded angular velocity of the vehicle heading.

5.1.2 UAS Model Lineraization and Discretization

We linearize the dynamic motion model around its hovering condition as in [150], in which tiny fluctuations of the attitude angle $(\psi = 0)$ are assumed and the vehicle heading is aligned with the x-axis of the mult-irotor inertial frame. We define the following state vector:

$$x = \begin{bmatrix} p^T & v^T & {}^{\mathbb{W}}\phi & {}^{\mathbb{W}}\theta \end{bmatrix}^T \tag{5.12}$$

where p is the position vector of the UAS in the three-dimensional space, v is the velocity vector, ${}^{\mathbb{W}}\phi$ and ${}^{\mathbb{W}}\theta$ are the roll and pitch angles in the inertial frame $\mathbb{W}$.

Whereas We define the following control vector:

$$u = \begin{bmatrix} {}^{\mathbb{W}}\phi_d & {}^{\mathbb{W}}\theta_d & T \end{bmatrix}^T \tag{5.13}$$

Where ${}^{\mathbb{W}}\phi_d$ and ${}^{\mathbb{W}}\theta_d$ are the roll and pitch control commands in the inertial frame and T is the thrust control command, which presume that this can be accomplished instantly since the dynamics of the motors are normally fairly rapid.

However, the transformation between attitude angles and heading free attitude angles may be calculated using:

$$\begin{pmatrix} \phi \\ \theta \end{pmatrix} = \begin{pmatrix} cos\psi & sin\psi \\ -sin\psi & cos\psi \end{pmatrix} \begin{pmatrix} \phi^W \\ \theta^W \end{pmatrix} \tag{5.14}$$

Where I_ϕ, I_θ are the roll and pitch angles which we denote in inertial frame to get rid of the vehicle heading ψ from the model.

It is possible to reform the pose and velocity equations in the following way after discretizing the system:

$$P(t+1) = v(t) \cdot T_s + P(t) \tag{5.15}$$

where $P(t+1)$ and $P(t)$ are the positions at time $t+1$ and t, $v(t)$ is the vehicle speed at time t, and T_s is the sampling time.

The following linear state-space model is produced after linearization and discretization of the model, while also neglecting exterior forces, which are represented by the wind forces F_{ext}. The model is linearized and discretized as follows:

$$\dot{x}(t) = A_c x(t) + B_c u(t) \tag{5.16}$$

where A_c is the state matrix in continuous time

$$A_c = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & -a_x & 0 & 0 & g & 0 \\ 0 & 0 & 0 & 0 & -a_y & 0 & 0 & -g \\ 0 & 0 & 0 & 0 & 0 & -a_z & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau_\phi} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau_\theta} \end{bmatrix} \tag{5.17}$$

B_c is the input matrix in continuous time

$$B_c = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \\ \frac{k_\phi}{\tau_\phi} & 0 & 0 \\ 0 & \frac{k_\theta}{\tau_\theta} & 0 \end{bmatrix} \tag{5.18}$$

where a_x, a_y and a_z are the drag coefficients, g is the gravity acceleration, τ_ϕ is the roll time constant, τ_θ is the pitch time constant, k_ϕ is the roll gain and k_θ is the pitch gain.

Since we will be utilizing a controller that acts in discrete time, we have discretized the UAS model as follows:

$$A = e^{A_c T_s}, \tag{5.19}$$

$$B = \int_0^{T_s} e^{A_c d\tau} d\tau B_c, \tag{5.20}$$

where T_s is the sampling time.

5.1.3 Problem Statement

For a generic UAS dynamic model described by 5.16, where $\boldsymbol{x}(t) \in \mathbb{R}^{n_x}$ is the system state with dimension n_x, and $\boldsymbol{u}(t) \in \mathbb{R}^{n_u}$ is the control input with dimension n_u. Both states and control inputs should respect specific constraints. Specifically, the vehicle state must belong to the *free state space* $\mathcal{X}_{\text{free}} = \mathcal{X} \setminus \mathcal{X}_{\text{obs}}$, in order to navigate through an obstacle-free trajectory. This constraint is expressed by

$$\boldsymbol{x}(t) \in \mathcal{X}_{\text{free}}, \tag{5.21}$$

where $\mathcal{X}$ is the state space and $\mathcal{X}_{\text{obs}}$ is the space occupied by obstacles. Moreover, the system input is constrained as

$$\boldsymbol{u}(t) \in \mathcal{U}, \tag{5.22}$$

where $\mathcal{U}$ is the space of the admissible inputs (roll, pitch and thrust commands) in order to consider the vehicle specifications.

Given the initial state of the UAS $\boldsymbol{x}_0 = \boldsymbol{x}(0)$ at time $t = 0$ and the target state defined by the goal region $\mathcal{X}_{\text{goal}} \subset \mathbb{R}^{n_x}$, the aim of the motion planning problem is to compute an optimal state trajectory $\overline{\boldsymbol{x}}^* : [0, t_f] \in \mathcal{X}_{\text{free}}$ and an optimal control input sequence $\overline{\boldsymbol{u}}^* : [0, t_f] \in \mathcal{U}$ over a finite time horizon from 0 to t_f able to drive the vehicle from the initial state $\boldsymbol{x}(0) = \boldsymbol{x}_0$ to a final state within the goal region $\boldsymbol{x}(t_f) \in \mathcal{X}_{\text{goal}}$.
$\overline{\boldsymbol{x}}^*$ and $\overline{\boldsymbol{u}}^*$ are computed minimizing a cost function $\text{Cost}(\cdot)$ while satisfying the constraints imposed by Equations (5.21) and (5.22). Hence, the optimal motion is the solution of the following problem

$$\begin{aligned} & \overline{\boldsymbol{x}}^*, \overline{\boldsymbol{u}}^* = \arg\min \ \text{Cost}(\boldsymbol{x}(t), \boldsymbol{u}(t)) \\ \text{subject to} \quad & \boldsymbol{x}(0) = \boldsymbol{x}_0 \\ & \boldsymbol{x}(t_f) = \boldsymbol{x}_{\text{goal}} \in \mathcal{X}_{\text{goal}} \\ & \boldsymbol{x}(t) \in \mathcal{X}_{\text{free}}, \ \forall t \in \left[0, t_f\right] \\ & \boldsymbol{u}(t) \in \mathcal{U}, \ \forall t \in \left[0, t_f\right]. \end{aligned} \tag{5.23}$$

5.2 The MP-RRT# strategy

This section introduces the proposed MP-RRT# algorithm (*Model Predictive Rapidly-exploring Random Tree "sharp"*). The core planner upon which the proposed algorithm is built is the RRT# [42]. The MP-RRT# improves on the RRT# by incorporating an MPC strategy to compute a near-optimal trajectory for a UAS while taking into account dynamic and kinematic constraints and avoiding obstacles. The RRT# is a particular version of the Rapidly-exploring Random Graph (RRG) that guarantees that the graph in the search space is the most optimum graph in the world.

Like other kinodynamic RRT-based algorithms, our MP-RRT# method explores the search space by building an incremental network that is rooted at the start of the search procedure. Specifically, the MP-RRT# generates two graphs simultaneously:

1. $\mathcal{G}^{\mathcal{Y}}$ in the reference space $\mathcal{Y}$: consists of vertices and edges in the reference space $\mathcal{Y}$. It is constructed incrementally by sampling vertices and growing the graph to uniformly explore the reference space. This is equivalent to any other graph generated to other algorithms of the RRT family.

2. $\mathcal{G}^{\mathcal{X}}$ in the state space $\mathcal{X}$: consists of a graph of trajectories computed through MPC. $\mathcal{G}^{\mathcal{X}}$ is built concurrently with $\mathcal{G}^{\mathcal{Y}}$ and, practically, it is used to evaluate the motion between vertices of $\mathcal{G}^{\mathcal{Y}}$, generating a graph of feasible trajectories in the state space.

The main pseudocode of MP-RRT# is defined in Algorithm 8. The inputs of the algorithm are the initial state x_0, the goal region $\mathcal{X}_{\text{goal}}$, the reference space $\mathcal{Y}$ and the state space $\mathcal{X}$ in which the motion planning searches for a feasible solution.
First, both graphs $\mathcal{G}^{\mathcal{Y}}$ and $\mathcal{G}^{\mathcal{X}}$ are initialized (from lines 2 to 4). In particular, the initial vertex in the reference state is defined using the initial state x_0 (line 3). In fact, we assume that the reference space $\mathcal{Y}$ is a subset of the state space $\mathcal{X}$ and, as a consequence, an element $r \in \mathcal{Y}$ can be derived from a state $x \in \mathcal{X}$. Then, the iterative procedure of the construction of the graph starts and continues until a certain number N of vertices are sampled and added to the graph (lines 5 to 8). Specifically, a vertex r_{rand} is randomly sampled (line 6) and both graphs $\mathcal{G}^{\mathcal{X}}$ and $\mathcal{G}^{\mathcal{Y}}$ are extended by adding the new vertex (line 7). The Replan() function propagates this update on the graphs (line 8). Both the Extend() and Replan() functions are detailed in Algorithms 9 and 11, respectively. Finally, the branch $\mathcal{T}^{\mathcal{X}}$ connecting the initial and the target states is extracted from the graph $\mathcal{G}^{\mathcal{X}}$ (line 9) and returned as the solution of the algorithm.

The Extend procedure is a crucial element for the proposed approach; it is responsible for the expansion of both graphs by adding a new vertex, after which the cost of the state trajectory is computed using MPC. This procedure

Algorithm 8: The MP-RRT# algorithm

```
MP-RRT#(x_0, X_goal, Y, X)
    G^X ← {x_0};
    r_0 ← x_0;
    G^Y ← {r_0};
    for i = 0 to N do
        r_rand ← Sample();
        G^X, G^Y ← Extend(G^X, G^Y, r_rand);
        Replan(G^X, G^Y);
    T^X ← SpanningTree(G^X);
    return T^X
```

is detailed in Algorithm 9. Initially, the new vertex r is connected to the nearest vertex r_{nearest} in the graph $\mathcal{G}^{\mathcal{Y}}$ (line 2). Then, the Nearest() function finds the vertex with the minimum Euclidean distance from r.
Hence, the states x_{nearest} and x are defined from r_{nearest} and r, respectively (lines 3 and 4). The ComputeTrajectory() function (line 5) uses MPC to compute the optimal state trajectory $\overline{\text{x}}$ moving from x_{nearest} to x. Then, if the computed trajectory is valid, i.e. it does not collide with obstacles, the *cost-to-come* of vertex r, denoted by $g(r)$ is computed by adding the cost at the previous vertex to the cost of the trajectory $\overline{x}$, denoted by $c(\overline{x})$ (line 7).
In line 8 all the neighbor vertices of r are added to the neighbor set $\mathcal{N}$ and, then, the vertex r is included in the neighbor set of its neighbors (lines 8 to 10).

The Near() function selects the M-nearest vertices as defined in [37]. Specifically, the number M of neighbors evaluated is defined as

$$M = e(1 + 1/d)\log|V|, \tag{5.24}$$

where d is the dimension of the reference space $\mathcal{Y}$, and the notation $|V|$ defines the cardinality of the set of vertices, i.e. the number of vertices in the graph $\mathcal{G}^{\mathcal{Y}}$. According to [37], Equation (5.24) ensures the asymptotic optimality of the algorithm.

The FindParent() function searches for the neighbor vertex of r that provides the minimum cost-to-come $g()$ including the vertex r to the graph $\mathcal{G}^{\mathcal{Y}}$

and, similarly, the corresponding state x to the graph $\mathcal{G}^{\mathcal{X}}$ (line 11). Then, the vertex r is included in the priority queue q (line 12) used in the Replan() to propagate any updated cost in the graph $\mathcal{G}^{\mathcal{Y}}$.

The FindParent() procedure is detailed in Algorithm 10. For each near vertex of r, the state trajectory from x and x_{near} is computed to select the best parent vertex (from lines 2 to 9). Then, the selected r_{near} is defined as parent of r (line 8) and, similarly, x_{near} is defined as parent of x (line 9).

The priority queue has a crucial role in the RRT# algorithm [42] because it is a queue of vertices that is evaluated in the Replan() procedure to propagate any update on the graph. Vertices of the queue are ordered based on their cost $f(r)$ from the highest to the lowest. Specifically, the cost $f(r)$ is the estimated cost to reach the goal passing through the vertex r, inspired by the well-known cost function define in the A* algorithm [151]:

$$f(r) = g(r) + \hat{h}(r). \tag{5.25}$$

Function $g(r)$ represents the *cost-to-come* at the vertex r, i.e. the cost of moving between the start vertex r_0 and r, with $g(r_0) = 0$. Function $\hat{h}(r)$ is the estimated *cost-to-go* to reach the goal state, with $\hat{h}(r_{\text{goal}}) = 0$.

Algorithm 9: The Extend procedure

```
1:  Extend(G^X, G^Y, r)
2:      r_nearest ← Nearest(G^Y, r);
3:      x_nearest ← r_nearest;
4:      x ← r;
5:      x̄ ← ComputeTrajectory(x_nearest, x);
6:      if isTrajectoryValid(x̄) then
7:          g(r) ← g(r_nearest) + c(x̄);
8:      N(r) ← Near(G^Y, r);
9:      foreach r_near ∈ N(r) do
10:         N(r_near) ← N(r_near) ∪ {r};
11:     FindParent(r, x);
12:     UpdateQueue(r);
13:     return G^X, G^Y
```

In particular, the Replan() procedure is detailed in Algorithm 11. This procedure is based on an iterative loop that updates only promising vertices

Algorithm 10: The FindParent procedure

```
FindParent(r, x)
    foreach r_near ∈ N(r) do
        x_near ← r_near;
        x̄ ← ComputeTrajectory(x_near, x);
        if isTrajectoryValid(x̄) then
            if g(r_near) + c(x̄) < g(r) then
                g(r) = g(r) + c(x̄);
                P(r) = r_near;
                P(x) = x_near;
```

(lines 2 to 15), i.e., vertices that can improve the current solution in the graph. Specifically, the set of promising vertices $V_{\text{prom}} \subset V$ contains vertices inside the relevant region $\mathcal{Y}_{\text{rel}} \in \mathcal{Y}$

$$\mathcal{Y}_{\text{rel}} = \{r \in \mathcal{Y}_{\text{free}} : f(r) < g(r^*_{\text{goal}})\}, \tag{5.26}$$

with r^*_{goal} is the vertex in the goal region with the minimum cost-to-come. Notably, the heuristic cost $\hat{h}(r)$ used to compute $f(r)$ must be admissible, i.e., it should not overestimate the cost-to-go, discarding vertices that would lead to the optimal solution. The evaluation of promising vertices is essential to avoid the propagation toward vertices that cannot improve the current solution, speeding up the algorithm. The first element of the queue is selected (line 3) and removed from q (line 5). Then, the procedure verifies if the current vertex can improve the cost-to-come of its neighbors (lines 6 to 15) as a new parent vertex. This is verified by computing the cost-to-come of the resulting state trajectory of moving from x to x_{nbh}. Similar to Algorithm 10, line 10 checks if the neighbor vertex $r_{\text{nbh}} \in N$ is a promising vertex and, in line 11, if r can be the new parent vertex of r_{nbh}. If this condition occurs, the vertex r_{nbh} is included in q to be evaluated in the Replan() procedure. In particular, r_{nbh} is defined in the reference space, while x_{nbh} is the corresponding state in the state space.

Figure 5.1 shows a simple example of the proposed strategy, where the graph $\mathcal{G}^{\mathcal{Y}}$ in blue is constructed in the reference state, while the corresponding graph $\mathcal{G}^{\mathcal{X}}$ in the state space is colored in magenta.

Algorithm 11: The Replan procedure

```
1:  Replan(G^X, G^Y)
2:      while f(q.top()) ≺ g(r*_goal) do
3:          r = q.top();
4:          x ← r;
5:          q.pop();
6:          foreach r_nbh ∈ N(r) do
7:              x_nbh ← r_nbh;
8:              x̄ ← ComputeTrajectory(x, x_nbh);
9:              if isTrajectoryValid(x̄) then
10:                 if g(r) + c(x̄) + ĥ(r_nbh) < g(r*_goal) then
11:                     if g(r) + c(x̄) < g(r_nbh) then
12:                         g(r_nbh) = g(r) + r(x̄);
13:                         P(r_nbh) = r;
14:                         P(x_nbh) = x;
15:                         UpdateQueue(r_nbh);
```

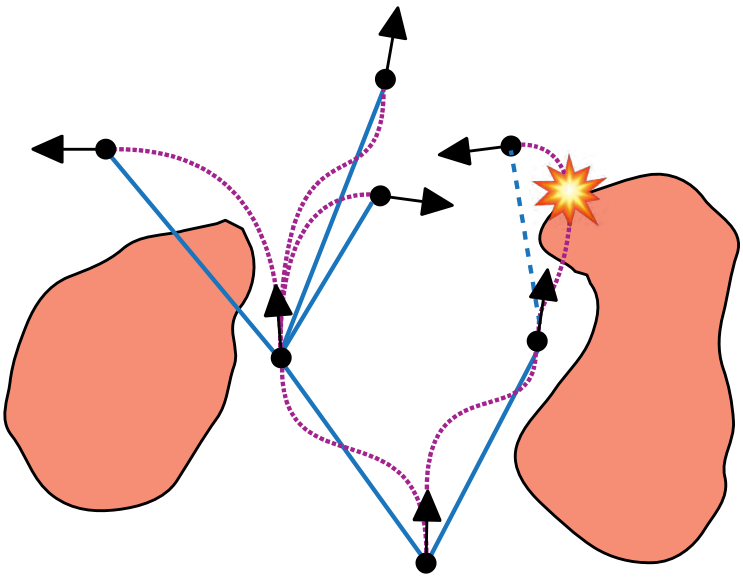


Fig. 5.1 Example of graphs constructed with MP-RRT#. The graph $\mathcal{G}^{\mathcal{Y}}$ consists of vertices (in black) and edges (in blue) in the reference state. Instead, the graph $\mathcal{G}^{\mathcal{X}}$ consists of trajectories (in magenta) obtained through evaluating the edges of $\mathcal{G}^{\mathcal{Y}}$ using the MPC strategy. An edge of $\mathcal{G}^{\mathcal{Y}}$ is labeled as invalid if its corresponding trajectory in $\mathcal{G}^{\mathcal{X}}$ crosses an obstacle.

As can be observed in Figure 5.1, collisions with obstacles are accounted for by graph $\mathcal{G}^{\mathcal{X}}$. If a trajectory in $\mathcal{G}^{\mathcal{X}}$ enters the obstacle space, the corresponding edge in $\mathcal{G}^{\mathcal{Y}}$ will not be included in the resulting graph. On the contrary, as can be observed in Figure 5.1, even if an edge in $\mathcal{G}^{\mathcal{Y}}$ crosses an obstacle, it is not discarded if its corresponding trajectory in $\mathcal{G}^{\mathcal{X}}$ does not collide with obstacles. This choice is motivated by the fact that, in general, the reference state has generally a lower dimension than the dimension of the vehicle state. As a consequence, it is more efficient to generate a graph in the reference space than in the state space.

5.2.1 Model Predictive Control

The cost of tracking the reference path is quantified using a LQR controller in recently published research, such as in [152]. Local linearization of the system dynamics was done in order to use linear quadratic regulation (LQR). Trajectory tracking within sample-based motion planners is typically performed for the linearized dynamics even though the actual UAS dynamics are known to be nonlinear. This is because when the system dynamics are linear, it is possible to solve for the optimal value function efficiently in closed-form. Furthermore, it is possible to apply this approach to non-linear systems by linearizing the nonlinear process dynamics about an operating point.

By using RRT instead of RRT$*$, the proposed sampling-based planner gets over the local rewiring limitations of RRT$*$ and, as suggested in [43], obtains its samples from the reference space rather than the control space. The proposed technique then uses MPC [153] to execute a forward simulation to generate a state trajectory and the appropriate control input to track the input sample of the closed-loop system, which is a reference r in the reference space $\mathcal{Y}$.

The proposed planner utilizes the linear MPC strategy to track the reference trajectory rather than the LQR, which optimizes across the entire time window (horizon), because the MPC can be implemented incrementally, that

is, updated through each step while still keeping in view a horizon with a configurable number of future steps sampled from the reference trajectory.

MP-RRT# use MPC to calculate the optimum state trajectory between each newly added vertex and its neighboring vertices and, to compute the cost of such a trajectory in order to gradually build the final trajectory graph $\mathcal{G}^{\mathcal{X}} \in \mathcal{X}$. Based on the UAS model previously defined, in this work we implement a Linear Model Predictive Control inspired by [150]. Specifically, the MPC searches for an optimal trajectory by optimizing the cost function

$$\begin{aligned} &J(\overline{x},\overline{u}) = \\ &\Big(\sum_{\mathrm{k}=0}^{H_\mathrm{p}-1} (x_\mathrm{k} - x_{\mathrm{ref,k}})^T Q_\mathrm{x} (x_\mathrm{k} - x_{\mathrm{ref,k}}) \\ &+ (u_\mathrm{k} - u_{\mathrm{k}-1})^T R_\Delta (u_\mathrm{k} - u_{\mathrm{k}-1}) \Big) \\ &+ (x_{\mathrm{H_p}} - x_{\mathrm{ref,H_p}})^T Q_{\mathrm{final}} (x_{\mathrm{H_p}} - x_{\mathrm{ref,H_p}}), \end{aligned} \tag{5.27}$$

where H_p is the prediction horizon.
The control input vector is defined as $\overline{u} = \begin{bmatrix} u_0 & u_1 & \dots & u_{\mathrm{H_p}} \end{bmatrix}^T$, with $u_\mathrm{k} \in \mathbb{R}^3$, for $k = 0, \dots, H_\mathrm{p} - 1$.
The vehicle state vector is $\overline{x} = \begin{bmatrix} x_0 & x_1 & \dots & x_{\mathrm{H_p}} \end{bmatrix}^T$, with $x_\mathrm{k} \in \mathbb{R}^8$, for $k = 0, \dots, H_\mathrm{p}$.
The reference state setpoint vector is defined as $\overline{x}_{\mathrm{ref}} = \begin{bmatrix} x_{\mathrm{ref},0} & x_{\mathrm{ref},1} & \dots & x_{\mathrm{ref,H_p}} \end{bmatrix}^T$, with $x_{\mathrm{ref,k}} \in \mathbb{R}^8$, for $k = 0, \dots, H_\mathrm{p}$.

However, the MPC weighting matrices Q_x, $R_\blacksquare$ and Q_{final} are positive semidefinite matrices indicating the penalty matrix on the state error, the penalty matrix on the variation of the control input, and the terminal cost matrix on the last state error, respectively.

In order to compute Q_{final}, it is necessary to iteratively solve an appropriate Algebraic Riccati Equation [154] which is implemented as follows: The computation of Q_{final} is carried out by iteratively solving a suitable Algebraic Riccati Equation [154]. Hence, the following convex optimization problem is

solved:

$$\overline{x}^*, \overline{u}^* = \min_{U,X} \quad J(\overline{x}, \overline{u}) \tag{5.28}$$

$$\text{subject to} \quad x_{\mathrm{k}+1} = Ax_{\mathrm{k}} + Bu_{\mathrm{k}} \tag{5.29}$$

$$u_k \in \mathcal{U} \tag{5.30}$$

$$x_0 = x(t_0) \tag{5.31}$$

It is necessary to have a reference trajectory x_{ref} in order to solve the optimization issue posed by Equation (5.28). The reference trajectory is implemented in this study using Dubins curve as defined in [147].

Dubins curves are the shortest route between two points in a two-dimensional space when the curvature is constant. Because our solution will be implemented in two-dimensional space, this approach is a great match for our research. Nevertheless, due to recent developments in three-dimensional space [155] and variable radius curvature [156], Dubins curves are now possible to be utilized in more complicated situations as well.

Assuming fixed vehicle speed and two-dimentional position vector $\begin{bmatrix} p_x & p_y & p_\beta \end{bmatrix}$, The differential equation of Dubins curves can be defined as follows:

$$\dot{p}_x = cos(p_\beta), \tag{5.32}$$

$$\dot{p}_y = sin(p_\beta), \tag{5.33}$$

$$\dot{p}_\beta = u_c, \tag{5.34}$$

As shown by [147], the maximum nubmer of combinations to express the shortest path between two poses can have no more than three primitive moves. Hence, only three values of u_c are defined $u_c \in \{-1, 0, 1\}$. The value $u_c = 0$ describes a straight motion (S), $u_c = -1$ describes a right (R) turn, and $u_c = 1$ describes a left (L) turn, thus obtaining six possible curves:

$$\{LRL, RLR, LSL, LSR, RSL, RSR\} \tag{5.35}$$

When compared to other primitive curves used to find the shortest path between two configurations, and for our purposes of building the graph

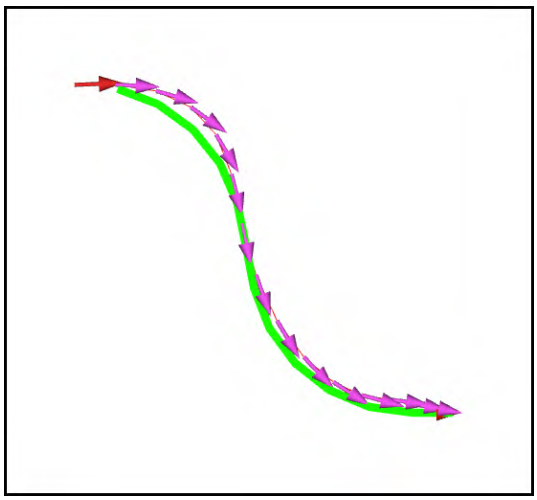


Fig. 5.2 Example of reference trajectory computed using Dubins curves and connecting two adjacent vertices. The green line is the reference trajectory, whereas magenta arrows are the state trajectory computed using MPC.

iteratively from curve segments extending the existing path with a new vertex, Dubins curves are a suitable solution to achieve flyable paths because they are forward-only curves, whereas other curves such as Reeds-Shepp require backward-motion [33]. Motion planning is typically influenced by the Dubins curvature radius, with various curvature radii resulting in varied planned trajectories. Curvature radius should be set to represent the smallest amount of curvature that the vehicle is capable of executing while still complying with kinodynamic restrictions. In this case, we are looking at a multicopter with a curvature radius of zero, which is theoretically possible (i.e., it can rotate around its axis in place). However, since we are assuming a continuous nonzero cruising velocity in the motion planning, we must provide a lower limit for the curvature radius.

The optimization problem of Equation (5.28) is solved following the reference trajectory. Then, in accordance with the MPC philosophy, only the first control input is applied and the optimization is solved iteratively. Figure 5.2 shows an example of reference trajectory generated using Dubins curves and followed through MPC.

5.3 Technological Tools

5.3.1 OMPL

Using the Open Motion Planning Library (OMPL), which is an abstract representation for all of the basic ideas in motion planning, it is possible to construct motion planning implementations. State space representation, control space representation, state validity, sampling, goal representations, and planners can all be implemented as part of the OMPL representation framework. OMPL is adaptable and may be utilized with a wide variety of robotic systems, including industrial robots. Towards this rationale, the library makes no specific reference of the geometry of a given workspace or of a particular robot that functions inside it. The fact that in OMPL there are so many file formats, data structures, and other ways of representation accessible for robotic systems was done on purpose in order to accomodate the enormous number of options available in the robotic field. As a result, in order for the robot to work as intended in the OMPL framework, one must choose a computational representation for the robot and implement an explicit state validity/collision detection method. In the OMPL programming language, there is no such fixed predefined or default collision checker. However, with only a limited number of assumptions, the library is capable of planning for a huge number of different systems while being lightweight and portable.

OMPL sampling-based motion planners are composed of a series of modules that all interact with one another to manage the motion planning problems in a coordinated manner. There are a few essential parts that must be used in order to handle a planning query across many sampling-based motion planners: a sampler to compute valid configurations of the robot, a state validity checker to rapidly assess a specific robot configuration, and a local planner to interconnect samples along a collision-free path. The most of of these components are furnished by OMPL in classes with the same names as the components they represent. The notions used in OMPL's classifications are comparable to those used in traditional sampling-based motion planners, and include the following:

- *StateSampler*: The *StateSampler* class of OMPL is modular to be configured to fit the vast majority of state space arrangements. It provides uniform and Gaussian sampling algorithms, as well as a multitude of other functionalities. For application scenarios where a combination of functionalities is required, such as sampling in Euclidean spaces, rotation spaces for 2D and 3D orientations, the *CompoundStateSampler* comes handy in defining the specific sampling method. Moreover, *StateValidityChecker* is used by the *ValidStateSampler* to identifying valid state space configurations.

- *NearestNeighbors*: This abstract class can be utilized to define a common interface to the planners for the purpose of implementing a nearest neighbor search among samples in the state space. Among the nearest neighbor search algorithms available to the core library are the Geometric Near-neighbor Access Tree and linear searches, and some others. It is also possible to make use of an external data structure and to supply an implementation of that data structure to the core library, as an alternative.

- *StateValidityChecker*: When evaluating a single state, the *StateValidity-Checker* will determine whether or not the configuration overlaps with any obstacle and whether or not it complies with the robot's constraint needs before proceeding. Given that this structure is a crucial aspect of sampling-based motion planning, it is vital for the user to provide the planner with a callback to such a function in order for the planner to check that all configurations are feasible for the robot.

- *MotionValidator*: When a robot moves across two configurations, the *MotionValidator* class evalutates the validity of the planned motion. For the *MotionValidator*, high-level criteria include the capacity to identify whether or not a move between two states is collision-free and whether or not the motion conforms with all of the robot's motion constraints.

- *OptimizationObjective*: motion planners are also required to optimize the motion against some performance criterion. These performance criterion is formulated in a cost function within this *OptimizationObjective* class. In order to optimize the motion strategy, the *OptimizationObjective* class offers an abstract interface to the processes and cost function formulation that planners would ultimately employ in order to optimize the strategy. Many cost functions are accessible in the library, including an optimization of the route length *PathLengthOptimizationObjective*.

- *ProblemDefinition*: The *ProblemDefinition* object specifies the query to be utilized for motion planning. Individual instances of this class include information about the robot's start state, goal configuration, and optimization target, if any, as well as the robot's start state and goal configuration. There may be a single configuration or a region that includes a certain state or territory that would be the goal. The solutions to motion planning inquiries that are provided may also be obtained via the usage of the *MotionPlanning* class.

OMPL's object-oriented structure is one of its most significant advantages; as a consequence, it is possible to inherit components that are already in existence while also creating additional components. As a result, it is not required to identify each and every object that is part of the OMPL structure since, for ordinary motion inquiries, the most of of the objects can be used in the their standard configuration.
In practice, the standard configurations of the large majority of these classes are sufficient for the greater part of fundamental planning purposes in the enormous circumstances.

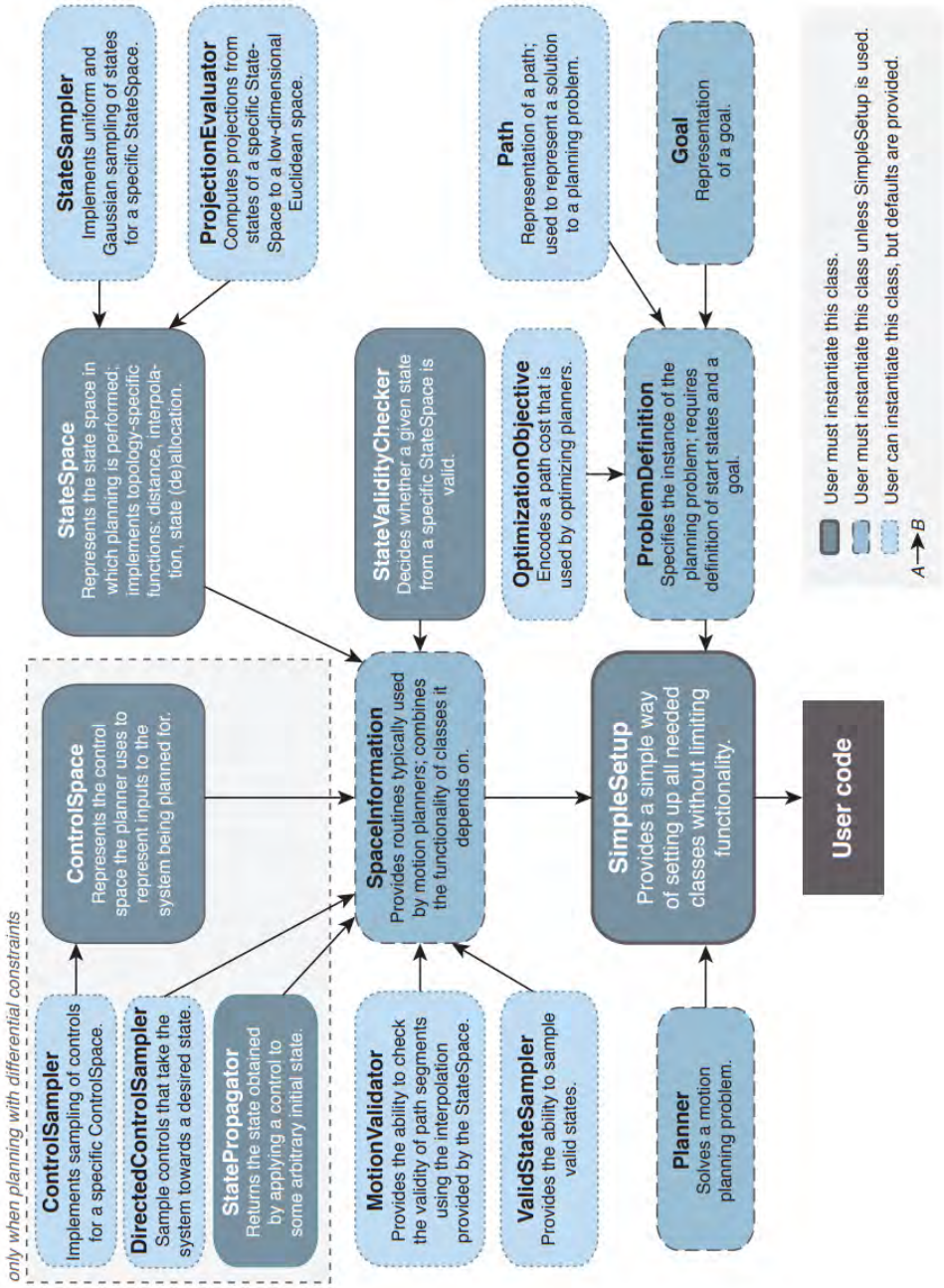


Fig. 5.3 Open Motion Planning Library (OMPL) object-oriented structure

Chapter 6

Experiment and Results

This chapter details the experiment setup and discusses the simulation results of the proposed MP-RRT$^{\#}$ motion planning solution.

6.1 Experiment

Here we will detail specifically the ComputeTrajectory() function that is used three times in every iteration, first to compute the trajectory and its cost of moving from the $x_{nearest}$ to the the newly sample x (line 5 in 9), then to evaluate the cost of the trajectory from every element in the near set x_{near} to the sample x during the FindParent() function (line 4), and finally again used to evaluate the trajectory from the x_{new} to each of the near set x_{near} in the Replan() function in order to check if rewiring the new sample as a parent of a neighbor node would improve the cost-to-come. The computed cost of the trajectory between any two nodes passed to the ComputeTrajectory() function uses the MPC controller that computes the trajectory respecting the constraints on the state and control input.

6.1.1 MPC Optimization Object

The trajectory cost calculation, which takes use of the MPC logic, is a unique aspect of the proposed method that distinguishes it from others previously given in the literature. It is necessary to construct a class called MPCOptimizationObjective in order to complete this task. It is set as the default class for performing the RRT$^{\#}$ optimization through the OMPL default setOptimizationObjective function, which is called by the setOptimizationObjective function. It contains some methods that are required for effectively solving the MPC optimization problem and for actually producing the recommended trajectory. In the instance that a motion query is submitted, OMPL immediately triggers two methods for determining how much the motion will cost. All of these methods should be available in the optimization class, including the stateCost() function, which is used to calculate the cost of a state, and the motionCost() method which is used to calculate the cost of transitioning from an initial state to a final state.

This motionCost() method of the OMPL MPCOptimizationObject class corresponds to the function named ComputeTrajectory() in the algorithm previously mentioned. This function is used to compute the actual trajectory while taking into account the obstacles constraints and the control input constraints, and it is responsible for reporting the actual trajectory between the two nodes that have been sent to the function as inputs, as well as the cost of the computed trajectory. The outputs of this function are then used to either *FindParent* or *Replan* the nearby nodes to the newly sampled node, depending on which phase of the algorithm invoked the function in the first place.

The MPC problem is defined by inspiring to [150], where a MPC-based trajectory tracking is constructed using CVXGEN. CVXGEN is an abbreviation for Code Generation for ConveX Optimization. It is an online tool for producing quick custom code for convex optimization problems that are modest and QP-representable.

The CVXGEN [157] interface is used to define an optimization problem using a simple and powerful language. This interface enables for the automated generation of library-free C code for a customized, high-speed solver which can be downloaded straight from the CVXgen website. CVXGEN offers solutions that are between twelve and one thousand times faster than the solutions produced by the most popular optimizers when dealing with the identical optimization challenges [158].

Assuming a disturbance free system model (i.e. $d(t) = 0 \forall t \in [0, H_P]$), the optimization problem can be defined as:

$$min_{U,X}(\sum_{k=0}^{H_P-1}(x_k - x_{ref,k})^T Q_x (x_k - x_{ref,k}) + (u_k - u_{k-1})^T R_\Delta (u_k - u_{k-1})) + \\ + (x_P - x_{ref,H_P})^T Q_{final} (x_P - x_{ref,H_P})$$

Subject to

$$x_{k+1} = Ax_k + Bu_k \\ u_k \in U \\ x_0 = x(t_0)$$

where H_P is the prediction horizon, U is the input space $U = [u_0\ u_1\ ...\ u_{H_P-1}]^T$ with $u_k \in R^3$ for $k = [0,\ ...,\ H_P - 1]$, X is the state space $X_{ref} = [x_0\ x_1\ ...\ x_{H_P}]^T$ with $x_k \in R^8$ for $k = [0,\ ...,\ H_P]$, Q_x is the penalty on the state error, R_Δ is a penalty on the control change rate and Q_{final} is the terminal state error penalty. The computation of the terminal cost matrix Q_{final} is done by solving the Algebraic Riccati Equation iteratively [154].

OMPL's motionCost() method takes as input two states then instantly convert them in ROS Pose2D format after initalizaing the MPC parameters.

initializeParameters()

The initializeParameters method is used in order to appropriately populate the UAS model matrices, the optimization function matrices, and the refer-

ence state vectors. Once the MPCOptimizationObjective class is created, this method is invoked to complete the process. Since the initialization phase takes significant time, and because the MPC tool configurations are constant for the duration of the RRT# graph creation phase, the intialization of parameters is only applied once within the MPCOptimizationObjective class constructor, rather than several times iteratively.
initializeParameters incorporates a ROS node handler that enables the function to read parameters from a launch file, which is useful for debugging. The read parameters are then stored into global variables. Although the OMPL class and MPCOptimizationObjective are meant to sample states in a variety of state spaces, the reference trajectory is only derived for a $x - y$ plan for our testing purposes, with the z position and z speed presumed to be equal to zero. Next we describe how the state matrix A and input matrix B are populated:

```
Eigen::Matrix<double, StateSize, StateSize> Q;
Eigen::Matrix<double, StateSize, StateSize> Q_final;
Eigen::Matrix<double, InputSize, InputSize> R;
Eigen::Matrix<double, InputSize, InputSize> R_delta;

const double kGravity = 9.8066;

drag_coefficients_.push_back(0.01);
drag_coefficients_.push_back(0.01);
drag_coefficients_.push_back(0.0);

A_continuous_time(0, 3) = 1;
A_continuous_time(1, 4) = 1;
A_continuous_time(2, 5) = 1;
A_continuous_time(3, 3) = -drag_coefficients_.at(0);
A_continuous_time(3, 7) = kGravity;
A_continuous_time(4, 4) = -drag_coefficients_.at(1);
A_continuous_time(4, 6) = -kGravity;
A_continuous_time(5, 5) = -drag_coefficients_.at(2);
A_continuous_time(6, 6) = -1.0 / roll_time_constant_;
A_continuous_time(7, 7) = -1.0 / pitch_time_constant_;

B_continuous_time(5, 2) = 1.0;
B_continuous_time(6, 0) = roll_gain / roll_time_constant_;
```

```
B_continuous_time(7, 1) = pitch_gain / pitch_time_constant_;

model_A_ =(prediction_sampling_time_ *A_continuous_time).exp();

Eigen::MatrixXd integral_exp_A;

integral_exp_A = Eigen::MatrixXd::Zero(StateSize, StateSize);
const int count_integral_A = 100;

for (int i = 0; i < count_integral_A; i++)
{
  integral_exp_A+= (A_continuous_time *
   prediction_sampling_time_ *i / count_integral_A).exp()*
  prediction_sampling_time_ / count_integral_A
}
```

Listing 6.1 initializeParameters() method

For computing B matrix in discrete time, it is necessary to compute the integral through an incremental approach.
The MPC Objective function matrices are initialize as follows:

```
Q.setZero();
Q_final.setZero();
R.setZero();
R_delta.setZero();

Q.block(0, 0, 3, 3) = q_position_.asDiagonal();
Q.block(3, 3, 3, 3) = q_velocity_.asDiagonal();
Q.block(6, 6, 2, 2) = q_attitude_.asDiagonal();

R = r_command_.asDiagonal();

R_delta = r_delta_command_.asDiagonal();

//Compute terminal cost
Q_final = Q;
for (int i = 0; i < 1000; i++)
{
  Eigen::MatrixXd temp = (model_B_.transpose() * Q_final *
   model_B_ + R);
```

```
  Q_final = model_A_.transpose() * Q_final * model_A_ - (
    model_A_.transpose() * Q_final * model_B_)* temp.inverse()*
    (model_B_.transpose() * Q_final * model_A_) + Q;
}
```

Listing 6.2 Computation of the integral of the system matrices through an incremental approach

Motion Cost Computation

Some heuristics are used to early discard poor samples. The main heuristic applied controls for the perpendicular of the initial pose yaw angle, it its in the opposite direction then the sample is marked with infinite cost and another sample is retrieved. The same is true for the final sampled poses with orientations that are more than sixty degrees off from the line connecting the initial and final postures, as well as for the first sampled poses. These heuristics helps to eliminate non-promising samples and, as a result, to avoid wasting time computing unnecessary MPC problem solutions, given that it is well known that initial and final poses with opposite directions can result in poor cost trajectories at the start and end of the process. This is implemented in function motionCostHeuristic2D inside the MPCOptimizationObjective class.

The output of this function is a Boolean parameter which is set to true whenever a goal pose does not satisfy pre-configured heuristic checks, and the cost computation is immediately stopped, returning an infinite cost. Alternatively, if the function return false, the MPC solver, the input limits, the initial position, and the reference states are set, and the optimization problem solution is calculated utilising the MPC solver and the reference states.

MPC Solver Configuration

The setSolver function initialize the MPC with the default configuration, however the solver generated by the CVXGEN tool can be customized. Of much interest is the settings.eps settings which is used to configure the

solver duality gap for returning the MPC problem solution. The solver will not declare a problem converged unless it is verified that the duality gap is confined by settings.eps. More specificaly, the solver will not declare a problem converged until the norm of the equality and inequality residuals are both less than the value specified by settings.resid_tol. If settings.eps and settings.resid_tol are not achieved, settings.max_iters is used to specify the maximum number of iterations the MPC solver is permitted to perform before delivering a solution. The MPC solver's solution processing may be significantly expedited by appropriately fine tuning the configuration settings. For the purpose of this study, it has been chosen that the specifications for these parameters would be retained. The setLimits function establishes the boundaries of the input values. Limits are established in accordance with the [150].

setReferenceLine To construct a reference trajectory $x_{ref,k}$ with $k \in [0,\ H_P]$ linking the two nodes in question, this function is utilized. Afterwards, the MPC logic works on following this reference trajectory while also taking into consideration the free space limits as well as the control input constraints. The reference trajectory is generated using the Dubins curves described in 5.2.1. Nevertheless, since this trajectory will be utilized as a reference by the MPC in the future, we sample/interpolate 12 points along the trajectory and preserve them as the reference set-points that will be used by the MPC later on in the process. It assumes constant speed module of the UAS, as well as the sampling time, in order to sample/interpolate points along the trajectory that are compatible with the discrete time model and the digital controller, as well as the sampling time.

MPCSolver The MPC logic takes as input the whole set of setpoints sampled along the reference trajectory of the Dubins curve and solves the optimzation problem specified in 5.27 for the entire length of the 12 set-points. However, instead of preserving the entire trajectory generated by the MPC method, only the first optimized state $x_1 \in R^8$ is preserved as the current starting point, hence reducing the amount of data stored. The MPC then proceeds through the process of solving for the other setpoints recursively, utilizing the initial answer as a starting point for each subsequent solution. This mimics the online MPC behavior in which the controller utilises the

measured value of its location at each sample time to generate the solution for the remaining horizon.

Cost Upon the computation of the MPC trajectory which was tracking the Dubins curve reference trajectory, this actual trajectory cost is then computed and reported back along the trajectory. The cost can be measured in whatever metrics of interest to the specific application at hand. However, for our study purposes, we used the simple Euclidean Distance along the computed MPC trajectory. Its however worth noting that the distance computed is the linear distance between the sample points along the trajectory, although the actual trajectory might be somewhat more curved. However, since we are making all the computations with the temporal resolution of the system sampling time, and given the constant speed of the vehicle, the linear distance computer is a fair approximation of the actual distance.
Moreover, since the cost computed will be used to compare the trajectory between two nodes against other trajectories,then given that all the costs are computed the same way, the linear approximation of the Euclidean Distance has no effect on the comparison among the trajectories.

```
double ompl::base::MPCOptimizationObjective::computeCost1(Eigen
    ::MatrixXd matrix_x,Eigen::MatrixXd matrix_x_ref,Eigen::
    MatrixXd matrix_u, int it) const
{
  double c1=0.0;
  geometry_msgs::Pose2D pose1,pose2;
  for (size_t i = 1; i < it; i++) {
    pose1.x=matrix_x(i-1,0);
    pose1.y=matrix_x(i-1,1);
    pose2.x=matrix_x(i,0);
    pose2.y=matrix_x(i,1);
    //distance Cost
    c1+=computeDistance2D(pose1,pose2);
  }
  ROS_ERROR( "size of ref = %f", matrix_x.rows());
  return(c1);
}
```

Listing 6.3 Cost computation using the linear approximation of the Euclidean Distance

```
double ompl::base::MPCOptimizationObjective::computeDistance2D(
    geometry_msgs::Pose2D pose1, geometry_msgs::Pose2D pose2)
    const
{
  float x1=pose1.x;
  float y1=pose1.y;
  float x2=pose2.x;
  float y2=pose2.y;

  float d = sqrt(pow(x2 - x1, 2) + pow(y2 - y1, 2));
  return(d);
}
```

Listing 6.4 2D linear distance computation

6.2 Results

6.2.1 Implementation

The proposed strategy is implemented in C++ using the Robot Operating System (ROS) [159] framework and using the Open Motion Planning Library (OMPL) [160], which provides many state-of-the-art sampling-based algorithms and many additional functionalities to facilitate the development of new algorithms.

The MP-RRT$^{\#}$ algorithm is implemented considering a two-dimensional space, i.e., flying at a fixed altitude. Specifically, the Special Euclidean Group $SE(2)$ is used, in which each admissible configuration is a pose in the two-dimensional space free to translate and rotate.
Hence, each reference sampled by the algorithm in the reference space $\mathcal{Y}$ consists of three parameters, i.e., two defining the position of the UAS and a third defining its orientation, corresponding to the flight direction. Each time the MP-RRT$^{\#}$ algorithm evaluates the motion between two states, a reference trajectory is computed using Dubins curves and the MPC computes the optimal state trajectory and control input to track it.

Parameter	Value
a_x	0.01
a_y	0.01
a_z	0
k_ϕ	0.9
k_θ	0.9
τ_ϕ	0.250 s
τ_θ	0.255 s

Table 6.1 Parameters used for the UAS model.

The motion-cost of the trajectory is computed considering the path length of the resulting trajectory:

$$c(\overline{x}, \overline{u}) = \sum_{i=1}^{M} \|x_{\mathrm{i}} - x_{\mathrm{i}-1}\|_2, \tag{6.1}$$

with $x_{\mathrm{i}} \in \overline{x}$, and M is the size of the trajectory. On the other hand, the cost-to-go $\hat{h}(r)$ is computed as the distance of the Dubins curve between the vertex r and the goal region $\mathcal{X}_{\mathrm{ref}}$.

The optimization problem of the MPC is solved using CVXGEN [157], a tool for code generation for convex optimization. CVXGEN can be used to generate fast custom code for small, QP-representable convex optimization problems. The mathematical problem is translated into a high speed solver that is twelve-to thousand-times faster than other popular optimizers [157]. Hence, the linear model of the UAS and the Linear MPC problem of Equations (5.27) and (5.28) are included and solved with CVXGEN.

Experimental tests are performed considering the multicopter Asctec Firefly and using the parameters listed in Table 6.1.

The MP-RRT$^{\#}$ is executed considering a maximum cruise velocity of 2.5 m/s and the reference trajectory is computed with Dubins curves with a curvature radius of 2 m. The admissible control input is defined through the

following constraints:

$$-0.436 \text{ rad} \leq {}^{\mathbb{W}}\phi_d \leq 0.436 \text{ rad} \tag{6.2}$$

$$-0.436 \text{ rad} \leq {}^{\mathbb{W}}\theta_d \leq 0.436 \text{ rad} \tag{6.3}$$

$$-4.80 \text{ N} \leq T \leq 10.19 \text{ N} \tag{6.4}$$

The MPC is manually tuned by setting matrices Q_{x} and $R_{\blacksquare}$ through trial-and-error to attain a satisfactory behavior in tracking the reference trajectory. Hence, Q_{x} and $R_{\blacksquare}$ are defined as follows:

$$Q_{\mathrm{x}} = \begin{bmatrix} 40 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 40 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 60 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 20 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 20 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 25 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \tag{6.5}$$

$$R_{\blacksquare} = \begin{bmatrix} 0.3 & 0 & 0 \\ 0 & 0.3 & 0 \\ 0 & 0 & 0.0025 \end{bmatrix}. \tag{6.6}$$

Moreover, Q_{final} is computed by iteratively solving the Algebraic Riccati Equation [154].

Figure 5.2 illustrates an example of reference trajectory computed with Dubins curves and connecting two vertices. The trajectory is followed by the MPC that reaches the target vertex computing the roll and pitch control commands plotted in Figure 6.1.

6.2.2 Simulation results

The proposed MP-RRT$^{\#}$ algorithm is tested in different scenarios to evaluate its behavior in computing UAS trajectories.

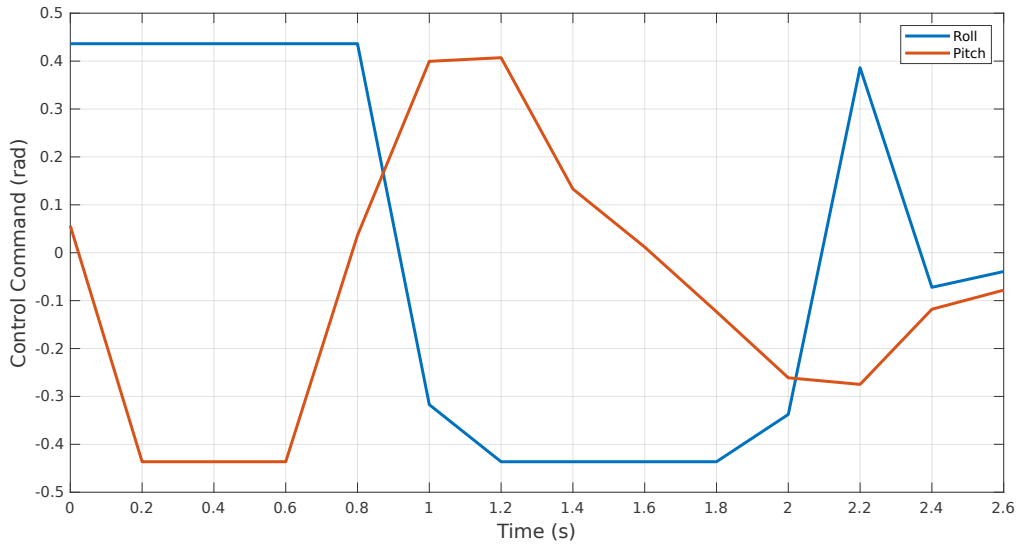


Fig. 6.1 The roll and pitch control inputs computed by MPC to follow the trajectory of Figure 5.2.

Figure 6.2 shows the evolution of the graph during the exploration of the reference space (i.e., the map). Specifically, in Figure 6.2(a), the algorithm computes a graph with 10 vertices finding an initial solution that is far from the optimal one. In Figure 6.2(b), the graph consists of 20 vertices, improving the solution path. On the contrary, the solution is not improved in Figure 6.2(c), with a graph with 60 vertices. Finally, in Figure 6.2(d), a better solution is found with a graph with 100 vertices.
The previously described test highlights the ability of the proposed algorithm to explore the map and to compute a feasible trajectory for the UAS. The quality of the computed trajectory increases with the number of vertices in the graph, converging toward the optimal solution. In order to demonstrate the above mentioned pattern, we performed 50 tests using the same scenario of Figure 6.2. The average cost of the resulting solution path against the number of iterations of the MP-RRT$^{\#}$ algorithm is shown in Figure 6.3.

Considering the same scenario of Figure 6.2, we evaluate the ability of the MPC in tracking the reference trajectory defined using Dubins curves. Table 6.2 reports the average trajectory tracking error in 20 tests. The average tracking error is the average Euclidean distance between the setpoints of the reference trajectory defined using Dubins curves and their corresponding states in the actual state trajectory computed using the MPC and satisfying

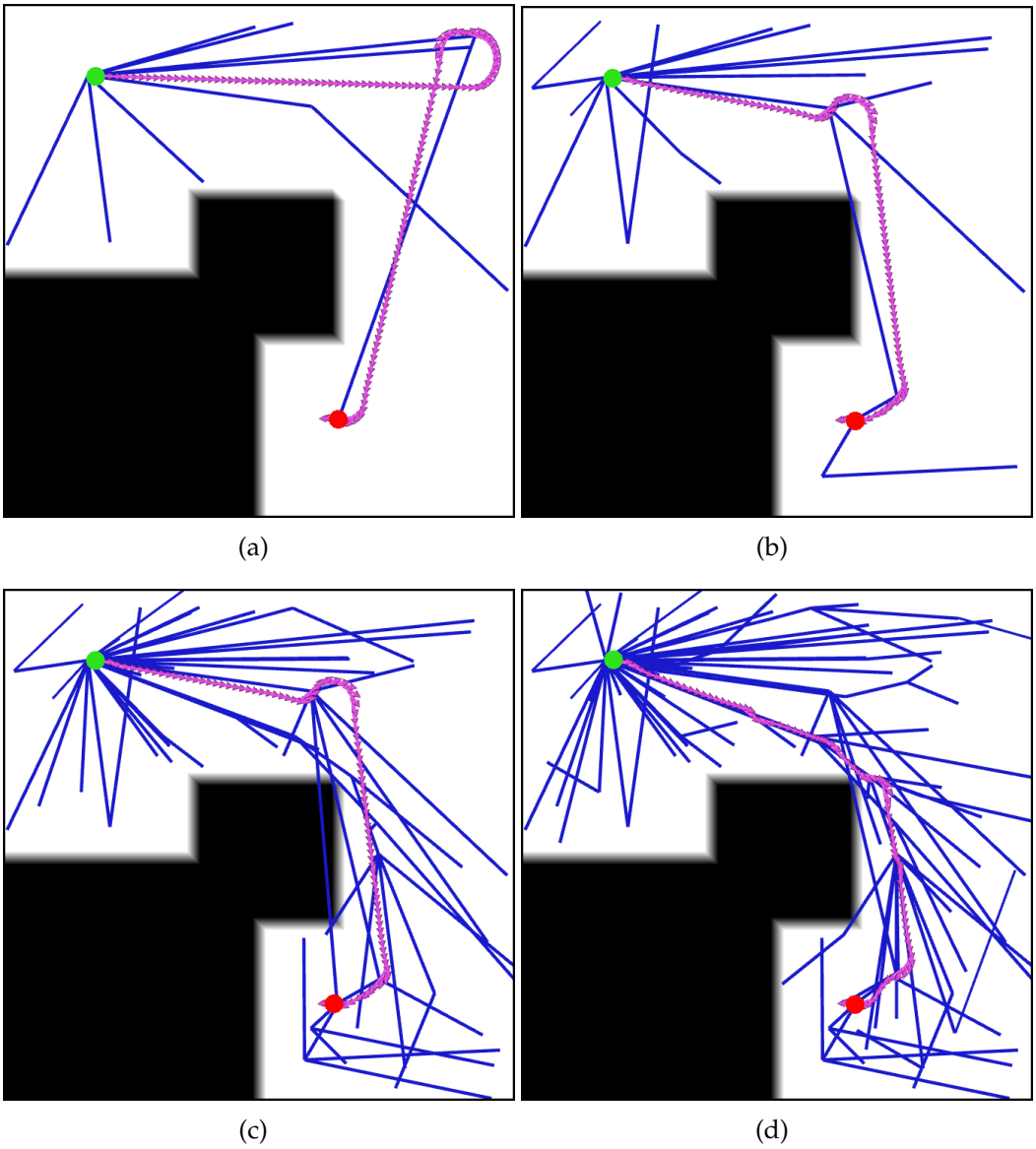


Fig. 6.2 The construction of the exploration tree using the MP-RRT$^{\#}$ algorithm. The start and target positions are in green and in red, respectively. The graph $\mathcal{G}^{\mathcal{Y}}$ in the reference space is colored in blue, while the computed path obtained from the graph $\mathcal{G}^{\mathcal{X}}$ in the state space is colored in magenta. In (a), the graph consists of 10 vertices rooted from the start pose finding an initial solution in the map with a cost (i.e. the path length) of 66.44 m. In (b), the graph with 20 vertices, in which the solution is improved with a cost of 45.09 m. In (c), the graph consists of 60 vertices, but the solution is not improved. In (d), the graph has 100 vertices obtaining a solution with cost 38.70 m.

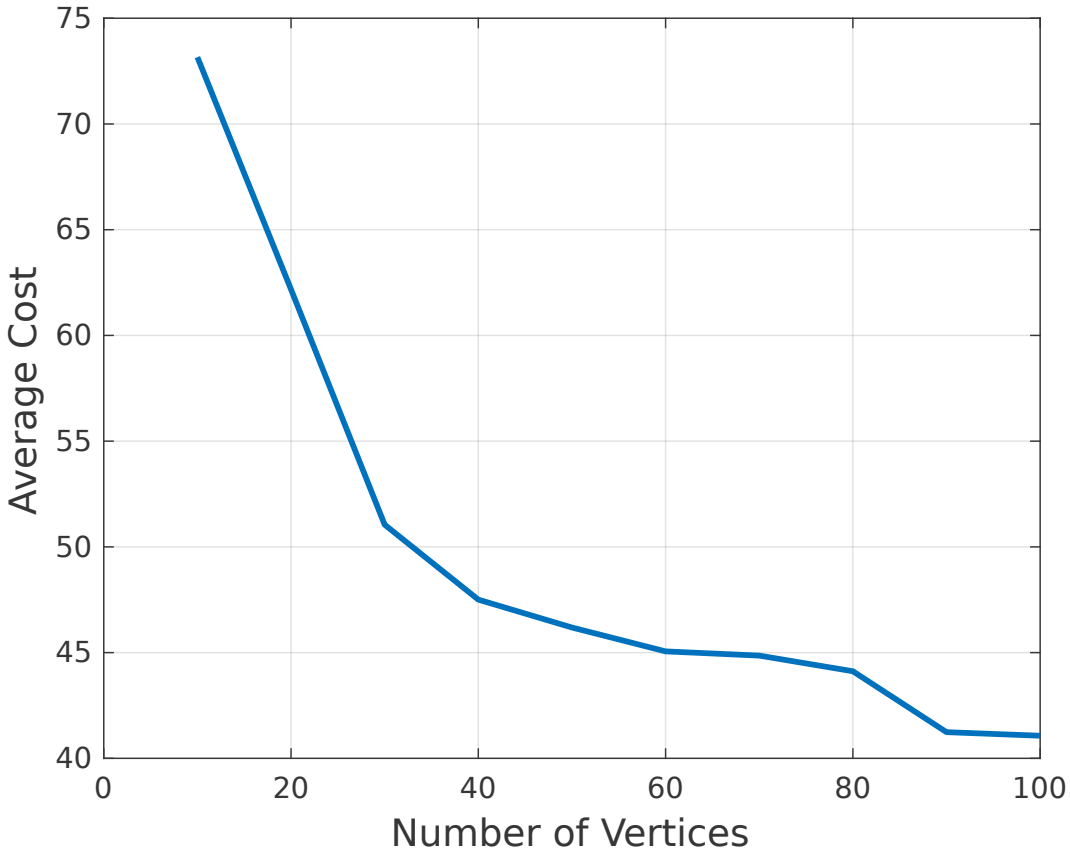


Fig. 6.3 The average cost of the solution path against the number of vertices in the MP-RRT$^{\#}$ algorithm. The average cost is computed running the algorithm 50 times in the same scenario of Figure 6.2.

dynamic constraints.
The average tracking error along the whole path was found to be reasonably small, being always smaller than 0.05 m along trajectories with a length ranging between 43 and 51 m. Figure 6.4 illustrates the average tracking error for each of the 20 tests.

Trajectory	Length [m]	Vertices	Avg Tracking Error [m]
1	47.178677	17	**0.04951**
2	43.488594	17	**0.04924**
3	49.000358	16	**0.04934**
4	48.921484	16	**0.04940**
5	48.720384	18	**0.04930**
6	47.986777	19	**0.05003**
7	46.894054	16	**0.04931**
8	44.604293	15	**0.04949**
9	51.149531	16	**0.04875**
10	45.98843	15	**0.04938**
11	48.523593	18	**0.04869**
12	47.661833	15	**0.05003**
13	43.958282	16	**0.04825**
14	43.825664	14	**0.04939**
15	49.610042	16	**0.04799**
16	48.843731	15	**0.04838**
17	49.636594	17	**0.04898**
18	45.822011	15	**0.04950**
19	45.370298	16	**0.04885**
20	45.527057	16	**0.05007**

Table 6.2 Trajectory tracking performance indices collected over 20 trajectories.

Other tests in more complex maps are shown in Figures 6.5 and 6.6. In particular, Figure 6.5 shows an interesting scenario, in which Figures 6.5(a) and 6.5(b) present the target in similar positions but with opposite directions. As a consequence, the algorithm computes different solutions in order to reach the target with the desired flight direction.

Similarly, in Figure 6.7(a), the MP-RRT$^{\#}$ algorithm explores a map with a graph of 100 vertices computing a solution. The trajectory computed in Fig-

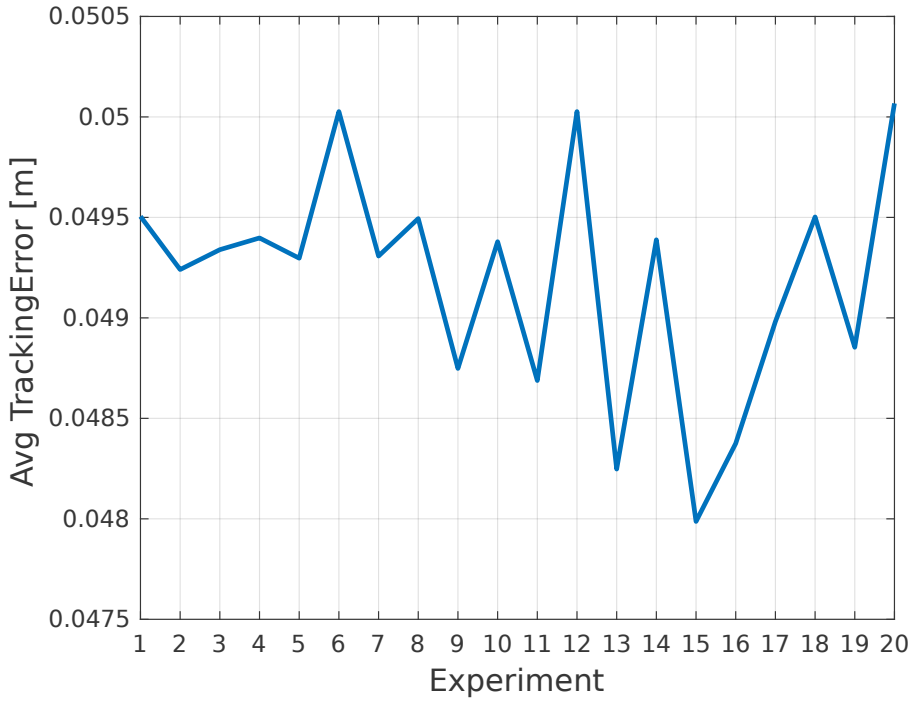
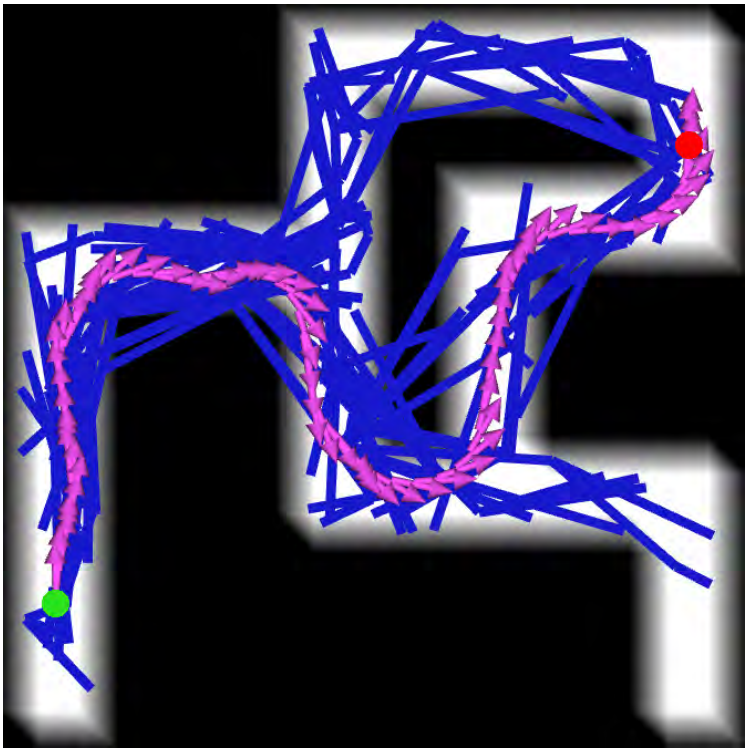


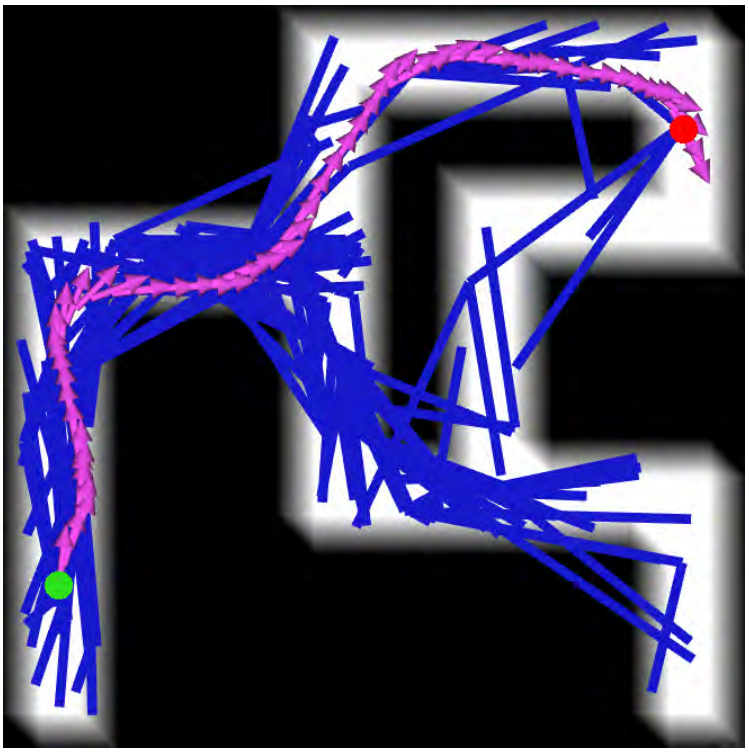
Fig. 6.4 The average tracking error for 20 trajectories running the same scenario of Figure 6.2.

ure 6.7(a) is also executed in a realistic simulation performed using Gazebo and SITL frameworks.
Gazebo is an open-source multi-robot simulator fully compatible with ROS [161] able to simulate robots, sensors, and rigid body dynamics.
SITL (Software In The Loop) [162] is a software to execute an autopilot on a computer, without using a specific and dedicated hardware. In this work, the simulation uses the PX4 autopilot [163], an open-source flight control software for drones and other autonomous vehicles.

In particular, the state trajectory computed with MP-RRT$^{\#}$ is uploaded on the PX4 autopilot and, then, executed as shown in Figure 6.7(b). Although the environment of Figure 6.7(b) does not correspond to the map of Figure 6.7(a), the executed trajectory in Figure 6.7(b) is the same generated in Figure 6.7(b).



(a)



(b)

Fig. 6.5 Trajectories computed with the MP-RRT$^{\#}$ by constructing a graph with 400 vertices. The start and target positions are in green and in red, respectively.

In blue, the graph $\mathcal{G}^{\mathcal{Y}}$ in the reference space, while in magenta, the computed path obtained from the graph $\mathcal{G}^{\mathcal{X}}$ in the state space. In (a) and (b) the target is in a similar position, but with opposite orientation. As a consequence, the solution is completely different, yielding different paths.

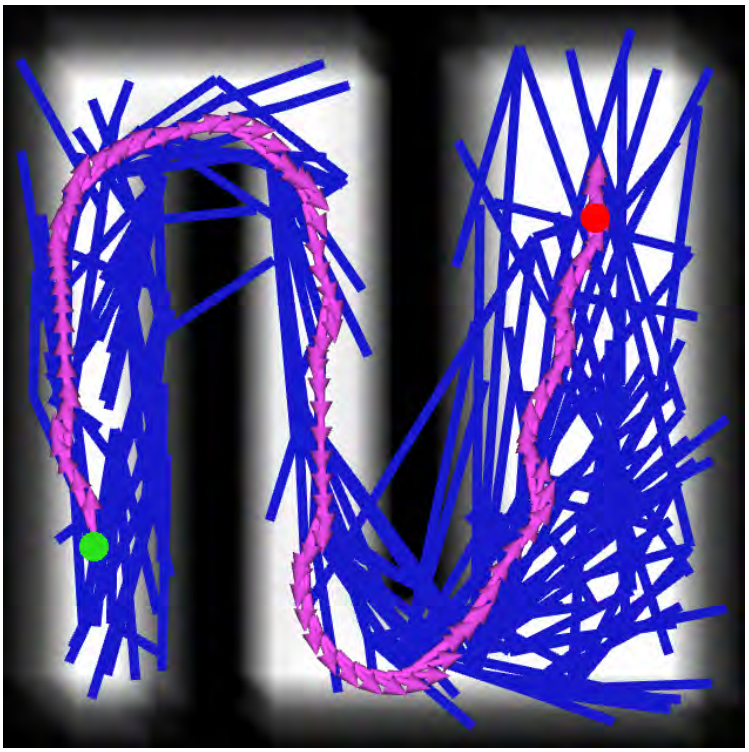
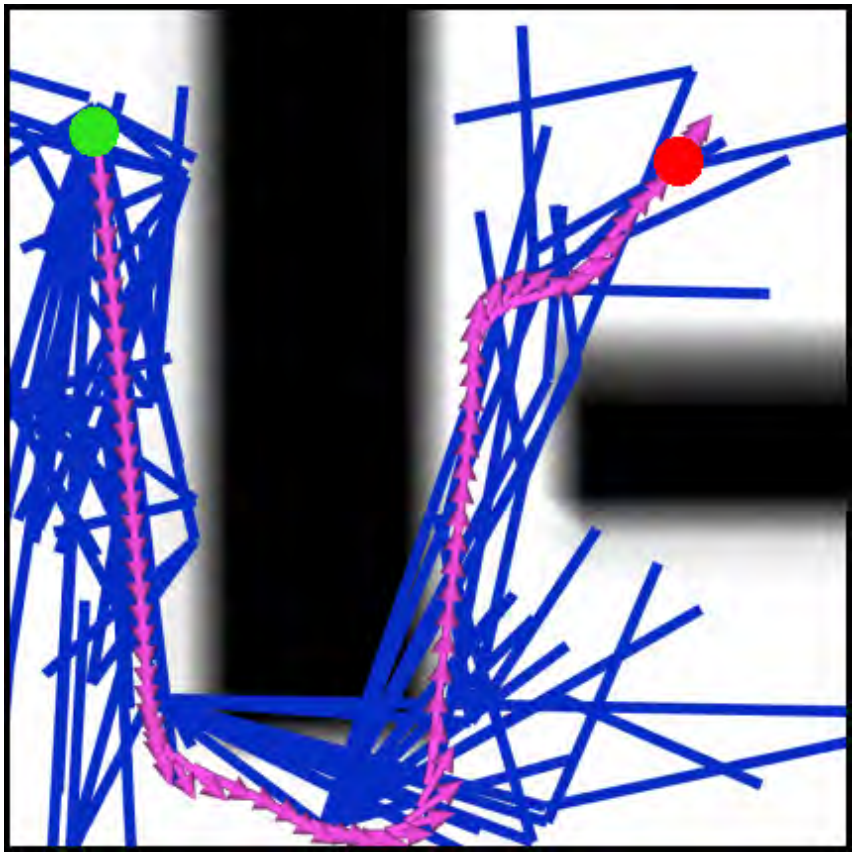
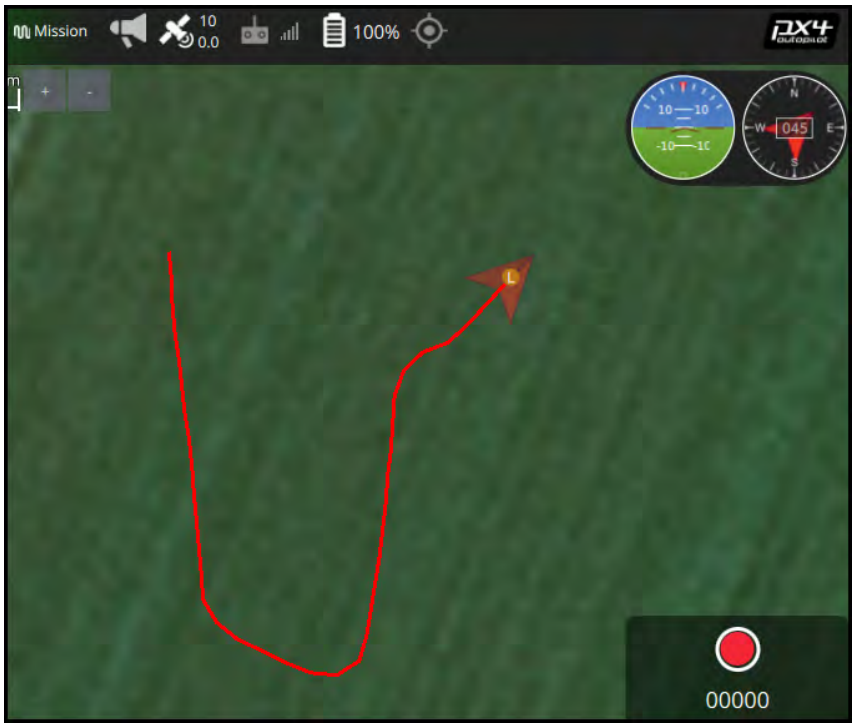


Fig. 6.6 Example of trajectory computed with the MP-RRT$^{\#}$ by constructing a graph with 400 vertices. The start and target positions are in green and in red, respectively. In blue the graph $\mathcal{G}^{\mathcal{Y}}$ in the reference space, while in magenta the computed path obtained from the graph $\mathcal{G}^{\mathcal{X}}$ in the state space.



(a)



(b)

Fig. 6.7 In (a), the trajectory computed with the MP-RRT# algorithm by constructing a graph of 100 vertices. In (b) the computed trajectory is executed by the PX4 autopilot in a simulation.

Chapter 7

Discussion and Conclusion

7.1 Conclusions

For the purposes of autonomous navigation, this study focuses on *Perception* and *Motion Planning* for Unmanned Aircraft System (UAS).

7.1.1 Perception

In perception, we have investigated the many design characteristics that influence the quality of the 3D environment model generated by a vision system, using the richness and accuracy of the 3D environment model as quality indicators. If one tweaks one of these parameters (aperture, shutter speed, or ISO), it will have a direct impact on the quantity of detected light intensity at each given pixel under identical lighting circumstances. The aperture has an influence on the Depth of Field (DOF) and the exposure, but has no effect on the Field of View (FOV), the Aperture Field of View (AFOV), or the effective focal length. While the shutter speed has an impact on the exposure period and the capacity to record rapidly moving objects, it also has an impact on the speed of the Unmanned Aerial Vehicle. When changing the aperture for purposes of depth of field (DOF) or the shutter speed for purposes of flight speed and moving objects in the scene, the ISO can be used as a compensation parameter by altering the sensitivity of the sen-

sor cells in order to alter the contrast for a given shutter and aperture settings.

The configuration recommendations are summarized below:

- Focus: should be set for the distance between the UAS and the highest point in the terrain.
- Aperture: should be set to have enough depth of field, equal to or greater than the distance between the highest point in the terrain and its base (ground). The best corresponding f-stop value can be extracted from the MTF curves of the lens or by trial before the actual flight.
- ISO: should be set to automatic allowing the software can adjust for the lighting conditions by altering the pixels' sensitivity.

The various design parameters, including the camera design and configuration parameters in addition to the flight plan parameters (speed and altitude), should be designed as per the following dependency graph:

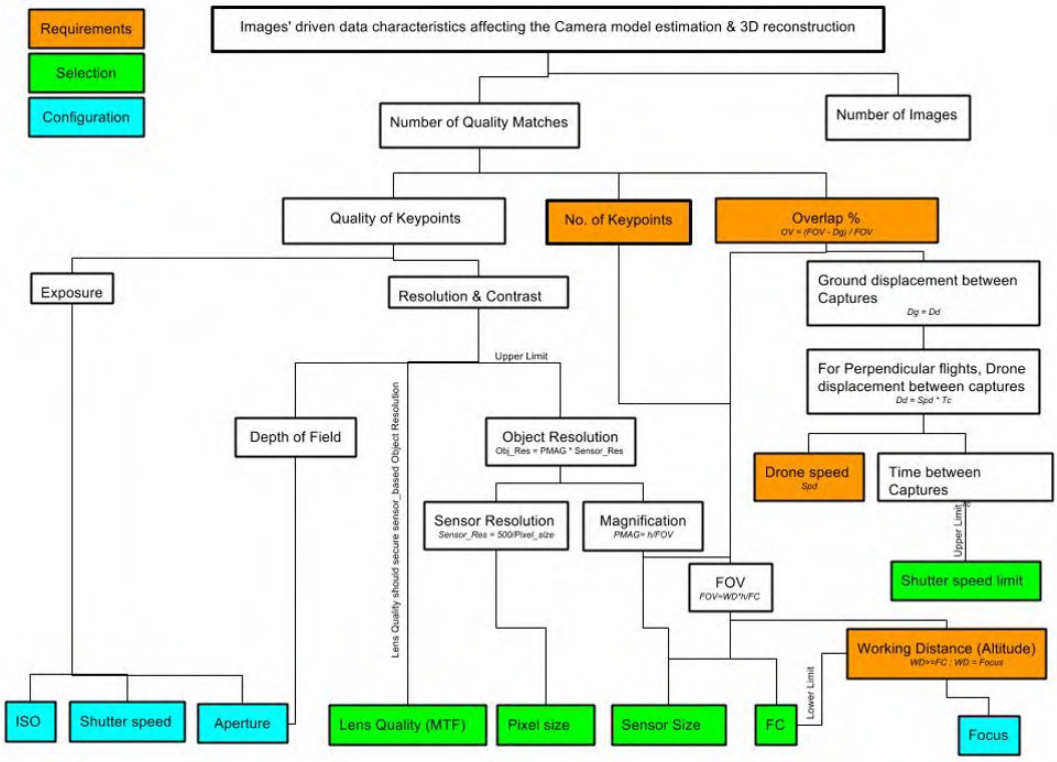


Fig. 7.1 Vision System Design parameters inter-dependencies and the way they affect the final 3D reconstruction through affecting the number of quality matched features

A further investigation was conducted into the usage of two GPS data sets as initials for the camera location (extrinsic parameters) in order to assess their contribution to the total 3D quality indicators that had previously been defined. The comparison of the resulting 3D environment models revealed that the usage of extra data sets would have a limited influence when they are utilized with claimed uncertainty limits that are larger than the actual uncertainty bounds. On the contrary, utilizing them with lower uncertainty limits results in worse camera calibration and, thus, poorer reconstruction outcomes.

This emphasizes the need of determining the real uncertainty limits of any extra data sets, which are often not the same as the nominal ones but are rather significantly affected by system parameters such as camera capture/GPS synchronization. Considering the design trade-offs between

different camera characteristics and the flight plan under certain criteria such as the final model resolution, the UAS speed, and the needed images' overlap, we identified the inter-dependencies and relationships between these parameters and their relationship to the number of quality matches.

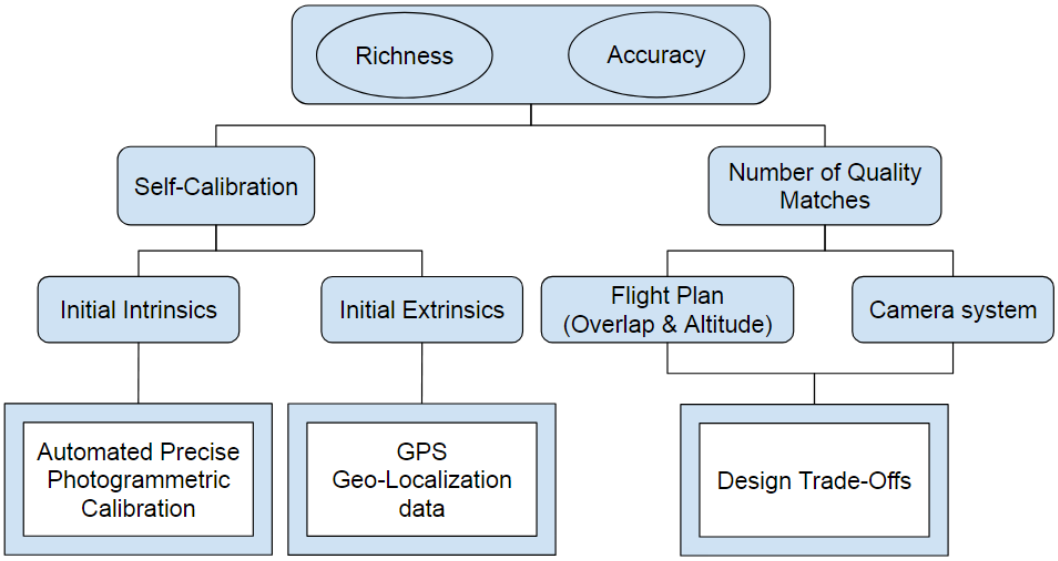


Fig. 7.2 Topics investigated and their place in the SfM framework

In addition, in part two, we have developed an industrial solution for a real-world manufacturing line, in which a challenging object detection and localization task is implemented and evaluated. With the help of an FPGA programmable industrial camera, the developed solution enabled the detection and localization of the target pieces, and then the control of the pieces feeder and conveyor shaker. The vision system was also integrated with a FANUC industrial robotic arm, which was used for component picking and assembly tasks. In a manufacturing facility, the solution has been successfully implemented, and the production cycle time has been significantly reduced. The time between image capture and the declaration of the found parts is less than 200ms, and the overall solution meets the reliability, robustness, and processing time requirements of the application.

7.1.2 Motion Planning

In the area of motion planning, we have developed a novel kinodynamic sampling-based motion planning algorithm known as MP-RRT$^{\#}$, which builds on the existing RRT$^{\#}$ by augmenting it with a Model Predictive Control method used to compute the optimal trajectory for Unmanned Aerial Vehicles. The use of the MPC ensures the feasibility and applicability of the resulting trajectory, as both the obstacles constraints (which restrict the feasible states to the free space) and the vehicle constraints (which limit the control input constraints) are taken into consideration by the MPC during the design process. Furthermore, the usage of a primitive curve such as the Dubins curve as a reference for the MPC to follow results in a smoother trajectory as a consequence of the MPC following a basic curve.

The proposed algorithm build two graphs are concurrently: $\mathcal{G}^{\mathcal{Y}}$ and $\mathcal{G}^{\mathcal{X}}$. First, the graph $\mathcal{G}^{\mathcal{Y}}$ explores the reference space of the UAS. Then, the MPC strategy is used to iteratively evaluate the feasibility of each newly added vertex and to compute the cost of its corresponding edge constructing a graph $\mathcal{G}^{\mathcal{X}}$ of feasible trajectories in the state space. The resulting trajectory computed by the proposed MP-RRT$^{\#}$ algorithm is a near-optimal trajectory that respects both the kinematic and dynamic constraints of the UAS.

The proposed MP-RRT$^{\#}$ algorithm differs from other kinodynamic RRT-based algorithms in sampling the input reference of the closed loop system instead of directly sampling the control input. This gives rise to considerable advantages, especially when dealing with vehicles with complex dynamics where the reference space dimension is considerably smaller than the control space and state space of the vehicle. The simulation results obtained from the implementation of the proposed MP-RRT$^{\#}$ algorithm demonstrate good trajectory quality even for complex maps. Moreover, the computed trajectory is executable by a UAS equipped with a professional autopilot.

Although the proposed algorithm is tested in a simplified scenario, i.e., in a two-dimensional space using a linearized model of the UAS, the proposed

MP-RRT$^{\#}$ algorithm can be extended to more complex scenarios by increasing the complexity of the algorithm. Moreover, although the work presented here focuses on UAS, the proposed motion planning strategy has a general validity, and can be easily adapted to other kinds of robots, such as ground robots, autonomous cars and underwater vehicles.

References

[1] Robert Bogue. Robots poised to revolutionise agriculture. *Industrial Robot: An International Journal*, 2016.

[2] Suraj Ashwath Rajiv and Ahmad Anwar Zainuddin. Review of new trends and challenges of android-based home security robot. *Malaysian Journal of Science and Advanced Technology*, pages 103–108, 2021.

[3] R Wolny. Robots in technological process of painting. *DAAAM International Scientific Book,(Austria)*, pages 195–204, 2011.

[4] Torbjørn S. Dahl and Maged N. Kamel Boulos. Robots in health and social care: A complementary technology to home care and telehealthcare? *Robotics*, 3(1):1–21, 2014.

[5] Azad Shademan, Ryan S Decker, Justin D Opfermann, Simon Leonard, Axel Krieger, and Peter CW Kim. Supervised autonomous robotic soft tissue surgery. *Science translational medicine*, 8(337):337ra64–337ra64, 2016.

[6] International Organization for Standardization. Technical Committee Automation systems, integration. Subcommittee Robots, and robotic devices. *ISO 8373: Robots and Robotic Devices - Vocabulary*. ISO, 2012.

[7] Francisco Rubio, Francisco Valero, and Carlos Llopis-Albert. A review of mobile robots: Concepts, methods, theoretical framework, and applications. *International Journal of Advanced Robotic Systems*, 16(2):1729881419839596, 2019.

[8] Nicoletta Bloise, Stefano Primatesta, Roberto Antonini, Gian Piero Fici, Marco Gaspardone, Giorgio Guglieri, and Alessandro Rizzo. A survey of unmanned aircraft system technologies to enable safe operations in urban areas. In *2019 International Conference on Unmanned Aircraft Systems (ICUAS)*, pages 433–442. IEEE, 2019.

[9] Chun Fui Liew, Danielle DeLatte, Naoya Takeishi, and Takehisa Yairi. Recent developments in aerial robotics: A survey and prototypes overview, 2017.

[10] Niko Sünderhauf, Oliver Brock, Walter Scheirer, Raia Hadsell, Dieter Fox, Jürgen Leitner, Ben Upcroft, Pieter Abbeel, Wolfram Burgard, Michael Milford, et al. The limits and potentials of deep learning for robotics. *The International Journal of Robotics Research*, 37(4-5):405–420, 2018.

[11] Hans Moravec and Alberto Elfes. High resolution maps from wide angle sonar. In *Proceedings. 1985 IEEE international conference on robotics and automation*, volume 2, pages 116–121. IEEE, 1985.

[12] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, and Andrew W. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. *2011 10th IEEE International Symposium on Mixed and Augmented Reality*, pages 127–136, 2011.

[13] D Blake Barber, Stephen R Griffiths, Timothy W McLain, and Randal W Beard. Autonomous landing of miniature aerial vehicles. *Journal of Aerospace Computing, Information, and Communication*, 4(5):770–784, 2007.

[14] Jean-Christophe Zufferey, Antoine Beyeler, and Dario Floreano. Near-obstacle flight with small uavs. Technical report, Springer Verlag, 2008.

[15] Noah Snavely, Steven M Seitz, and Richard Szeliski. Modeling the world from internet photo collections. *International journal of computer vision*, 80(2):189–210, 2008.

[16] Yasutaka Furukawa and Jean Ponce. Accurate, dense, and robust multiview stereopsis. *IEEE transactions on pattern analysis and machine intelligence*, 32(8):1362–1376, 2009.

[17] Jean-Claude Latombe. *Robot motion planning*, volume 124. Springer Science & Business Media, 2012.

[18] Pinghai Gao, Daibing Zhang, Qiang Fang, and Shaogang Jin. Obstacle avoidance for micro quadrotor based on optical flow. In *2017 29th Chinese Control And Decision Conference (CCDC)*, pages 4033–4037. IEEE, 2017.

[19] Kimberly McGuire, Guido De Croon, Christophe De Wagter, Karl Tuyls, and Hilbert Kappen. Efficient optical flow and stereo vision for velocity estimation and obstacle avoidance on an autonomous pocket drone. *IEEE Robotics and Automation Letters*, 2(2):1070–1076, 2017.

[20] Pengyue Li, Xiangyang Hao, Junqiang Wang, Youyi Gu, and Gaojie Wang. Uav obstacle detection algorithm based on improved orb sparse optical flow. In *2019 IEEE 4th Advanced Information Technology, Electronic*

and Automation Control Conference (IAEAC), volume 1, pages 562–569. IEEE, 2019.

[21] Jie Ji, Amir Khajepour, Wael William Melek, and Yanjun Huang. Path planning and tracking for vehicle collision avoidance based on model predictive control with multiconstraints. *IEEE Transactions on Vehicular Technology*, 66(2):952–964, 2016.

[22] Yuxiao Chen, Huei Peng, and Jessy W Grizzle. Fast trajectory planning and robust trajectory tracking for pedestrian avoidance. *Ieee Access*, 5:9304–9317, 2017.

[23] Penghong Lin, Songlin Chen, and Chang Liu. Model predictive control-based trajectory planning for quadrotors with state and input constraints. In *2016 16th International Conference on Control, Automation and Systems (ICCAS)*, pages 1618–1623. IEEE, 2016.

[24] D. Kuan, J. Zamiska, and R. Brooks. Natural decomposition of free space for path planning. In *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, volume 2, pages 168–173, 1985.

[25] Rodney A. Brooks and Tomás Lozano-Pérez. A subdivision algorithm in configuration space for findpath with rotation. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15(2):224–233, 1985.

[26] Gene Eu Jan, Chi-Chia Sun, Wei Chun Tsai, and Ting-Hsiang Lin. An $O(n \log n)$ shortest path algorithm based on delaunay triangulation. *IEEE/ASME Transactions on Mechatronics*, 19(2):660–666, 2014.

[27] P.C. Chen and Y.K. Hwang. Sandros: a dynamic graph search algorithm for motion planning. *IEEE Transactions on Robotics and Automation*, 14(3):390–403, 1998.

[28] Maxim Likhachev, Dave Ferguson, Geoff Gordon, Anthony Stentz, and Sebastian Thrun. Anytime search in dynamic graphs. *Artificial Intelligence*, 172(14):1613–1643, 2008.

[29] Mohamed Elbanhawi and Milan Simic. Sampling-based robot motion planning: A review. *IEEE Access*, 2:56–77, 2014.

[30] Jian-ying Zhang, Zhi-ping Zhao, and Dun Liu. A path planning method for mobile robot based on artificial potential field. *Journal of Harbin Institute of Technology*, 38(8):1306–1309, 2006.

[31] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Proceedings. 1991 IEEE International Conference on Robotics and Automation*, pages 1398–1404 vol.2, 1991.

[32] Bruce Donald, Patrick Xavier, John Canny, and John Reif. Kinodynamic motion planning. *Journal of the ACM (JACM)*, 40(5):1048–1066, 1993.

[33] Steven M LaValle. *Planning algorithms*. Cambridge university press, 2006.

[34] Steven M LaValle and James J Kuffner Jr. Randomized kinodynamic planning. *The international journal of robotics research*, 20(5):378–400, 2001.

[35] Emilio Frazzoli, Munther A Dahleh, and Eric Feron. Real-time motion planning for agile autonomous vehicles. *Journal of guidance, control, and dynamics*, 25(1):116–129, 2002.

[36] Thomas M Howard and Alonzo Kelly. Optimal rough terrain trajectory generation for wheeled mobile robots. *The International Journal of Robotics Research*, 26(2):141–166, 2007.

[37] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The international journal of robotics research*, 30(7):846–894, 2011.

[38] Kourosh Naderi, Joose Rajamäki, and Perttu Hämäläinen. Rt-rrt* a real-time path planning algorithm based on rrt. In *Proceedings of the 8th ACM SIGGRAPH Conference on Motion in Games*, pages 113–118, 2015.

[39] Sertac Karaman, Matthew R Walter, Alejandro Perez, Emilio Frazzoli, and Seth Teller. Anytime motion planning using the rrt. In *2011 IEEE International Conference on Robotics and Automation*, pages 1478–1483. IEEE, 2011.

[40] Michal Čáp, Peter Novák, Jiří Vokřínek, and Michal Pěchouček. Multi-agent rrt*: Sampling-based cooperative pathfinding. *arXiv preprint arXiv:1302.2828*, 2013.

[41] Iram Noreen, Amna Khan, Zulfiqar Habib, et al. Optimal path planning using rrt* based approaches: a survey and future directions. *Int. J. Adv. Comput. Sci. Appl*, 7(11):97–107, 2016.

[42] Oktay Arslan and Panagiotis Tsiotras. Use of relaxation methods in sampling-based algorithms for optimal motion planning. In *2013 IEEE International Conference on Robotics and Automation*, pages 2421–2428. IEEE, 2013.

[43] Yoshiaki Kuwata, Justin Teo, Gaston Fiore, Sertac Karaman, Emilio Frazzoli, and Jonathan P How. Real-time motion planning with applications to autonomous urban driving. *IEEE Transactions on control systems technology*, 17(5):1105–1118, 2009.

[44] Hsuan Chang and Tsai-Yen Li. Assembly maintainability study with motion planning. volume 1, pages 1012–1019, 1995. cited By 96.

[45] Michael Girard and A. A. Maciejewski. Computational modeling for the computer animation of legged figures. *SIGGRAPH Comput. Graph.*, 19(3):263–270, July 1985.

[46] Robert D. Howe and Yoky Matsuoka. Robotics for surgery. *Annual Review of Biomedical Engineering*, 1(1):211–240, 1999. PMID: 11701488.

[47] J.-C. Latombe. Motion planning: A journey of robots, molecules, digital actors, and other artifacts. *International Journal of Robotics Research*, 18(11):1119–1128, 1999. cited By 185.

[48] Herath MPC Jayaweera and Samer Hanoun. Uav path planning for reconnaissance and look-ahead coverage support for mobile ground vehicles. *Sensors*, 21(13):4595, 2021.

[49] Martin Johannes Schuster. Collaborative localization and mapping for autonomous planetary exploration : Distributed stereo vision-based 6d slam in gnss-denied environments, 2019.

[50] Purushothaman Raja and Sivagurunathan Pugazhenthi. Optimal path planning of mobile robots a review. *International journal of physical sciences*, 7(9):1314–1320, 2012.

[51] Anantha Sai Hari Haran V Injarapu and Suresh Kumar Gawre. A survey of autonomous mobile robot path planning approaches. In *2017 International conference on recent innovations in signal processing and embedded systems (RISE)*, pages 624–628. IEEE, 2017.

[52] Priyadarshi Bhattacharya and Marina Gavrilova. Roadmap-based path planning - using the voronoi diagram for a clearance-based shortest path. *Robotics Automation Magazine, IEEE*, 15:58 – 66, 07 2008.

[53] Thomas M. Howard, Colin J. Green, and Alonzo Kelly. *State Space Sampling of Feasible Motions for High Performance Mobile Robot Navigation in Highly Constrained Environments*, pages 585–593. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[54] Mihail Pivtoraiko, Ross A. Knepper, and Alonzo Kelly. Differentially constrained mobile robot motion planning in state lattices. *Journal of Field Robotics*, 26(3):308–333, 2009.

[55] Mihail Pivtoraiko and Alonzo Kelly. Kinodynamic motion planning with state lattice motion primitives. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2172–2179, 2011.

[56] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98, 1986.

[57] Howie M Choset, Kevin M Lynch, Seth Hutchinson, George Kantor, Wolfram Burgard, Lydia Kavraki, Sebastian Thrun, and Ronald C Arkin. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005.

[58] W. Ji, F. Cheng, D. Zhao, Y. Tao, S. Ding, and J. Lü. Obstacle avoidance method of apple harvesting robot manipulator. *Nongye Jixie Xuebao/-Transactions of the Chinese Society for Agricultural Machinery*, 44:253–259, 11 2013.

[59] J. Barraquand, B. Langlois, and J.-C. Latombe. Numerical potential field techniques for robot path planning. *IEEE Transactions on Systems, Man, and Cybernetics*, 22(2):224–241, 1992.

[60] Pei-Yan Zhang, Tian-Sheng Lü, and Li-Bo Song. Soccer robot path planning based on the artificial potential field approach with simulated annealing. *Robotica*, 22:563–566, 09 2004.

[61] J.-Y Zhang, Z.-P Zhao, and D. Liu. Path planning method for mobile robot based on artificial potential field. *Harbin Gongye Daxue Xuebao/Journal of Harbin Institute of Technology*, 38:1306–1309, 08 2006.

[62] Qing Li, Lijun Wang, Bo Chen, Zhou Zhou, and Yixin Yin. An improved artificial potential field method with parameters optimization based on genetic algorithm. *Journal of University of Science and Technology Beijing (Chinese Edition)*, 2, 02 2012.

[63] L. Tun. Optimized path planning of mobile robot based on artificial potential field. 2007.

[64] Min Cheol Lee and Min Gyu Park. Artificial potential field based path planning for mobile robots using a virtual obstacle concept. In *Proceedings 2003 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM 2003)*, volume 2, pages 735–740 vol.2, 2003.

[65] Min Gyu Park and Min Cheol Lee. Real-time path planning in unknown environment and a virtual hill concept to escape local minima. In *30th Annual Conference of IEEE Industrial Electronics Society, 2004. IECON 2004*, volume 3, pages 2223–2228 Vol. 3, 2004.

[66] Jinseok Lee, Yunyoung Nam, Sangjin Hong, and Weduke Cho. New potential functions with random force algorithms using potential field method. *Journal of Intelligent and Robotic Systems - JIRS*, 66, 05 2012.

[67] Tao Zhang, Yi Zhu, and Jingyan Song. Real-time motion planning for mobile robots by means of artificial potential field method in unknown environment. *Industrial Robot: An International Journal*, 37(4):384–400, Jan 2010.

[68] Anugrah K. Pamosoaji and Keum-Shik Hong. A path-planning algorithm using vector potential functions in triangular regions. *IEEE Transactions on Systems, Man, and Cybernetics: Systems*, 43(4):832–842, 2013.

[69] Ya-Chun Chang and Yoshio Yamamoto. Path planning of wheeled mobile robot with simultaneous free space locating capability. *Intelligent Service Robotics*, 2:9–22, 01 2009.

[70] Md Rahman and Md Azad. To escape local minimum problem for multi-agent path planning using improved artificial potential field-based regression search method. pages 371–376, 12 2017.

[71] Qinzhao Wang, Jinyong Cheng, and Xiaolong Li. Path planning of robot based on improved artificial potentional field method. In *Proceedings of the 2017 International Conference on Artificial Intelligence, Automation and Control Technologies*, AIACT '17, New York, NY, USA, 2017. Association for Computing Machinery.

[72] Wang Siming, Zhao Tiantian, and Li Weijie. Mobile robot path planning based on improved artificial potential field method. In *2018 IEEE International Conference of Intelligent Robotic and Control Engineering (IRCE)*, pages 29–33. IEEE, 2018.

[73] Giannis P Roussos, Dimos V Dimarogonas, and Kostas J Kyriakopoulos. 3d navigation and collision avoidance for a non-holonomic vehicle. In *2008 American Control Conference*, pages 3512–3517. IEEE, 2008.

[74] Anantha Sai Hari Haran V Injarapu and Suresh Kumar Gawre. A survey of autonomous mobile robot path planning approaches. In *2017 International Conference on Recent Innovations in Signal processing and Embedded Systems (RISE)*, pages 624–628, 2017.

[75] David Ferguson, Maxim Likhachev, and Anthony (Tony) Stentz. A guide to heuristic-based path planning, June 2005.

[76] František Duchoň, Andrej Babinec, Martin Kajan, Peter Beňo, Martin Florek, Tomáš Fico, and Ladislav Jurišica. Path planning with modified a star algorithm for a mobile robot. *Procedia Engineering*, 96, 12 2014.

[77] E. Frazzoli, M.A. Dahleh, and E. Feron. Real-time motion planning for agile autonomous vehicles. In *Proceedings of the 2001 American Control Conference. (Cat. No.01CH37148)*, volume 1, pages 43–49 vol.1, 2001.

[78] Ismail Altaharwa, Alaa Sheta, and Mohammed Alweshah. A mobile robot path planning using genetic algorithm in static environment. *Journal of Computer Science*, 4, 01 2008.

[79] Yang Wang, David Mulvaney, and Ian Sillitoe. Genetic-based mobile robot path planning using vertex heuristics. In *2006 IEEE Conference on Cybernetics and Intelligent Systems*, pages 1–6, 2006.

[80] Chaymaa Lamini, Said Benhlima, and Ali Elbekri. Genetic algorithm based approach for autonomous mobile robot path planning. *Procedia Computer Science*, 127:180–189, 2018.

[81] Amin Zargar Nasrollahy and Hamid Haj Seyyed Javadi. Using particle swarm optimization for robot path planning in dynamic environments with moving obstacles and target. In *2009 Third UKSim European Symposium on Computer Modeling and Simulation*, pages 60–65. IEEE, 2009.

[82] Dun-wei Gong, Jianhua Zhang, and Yong Zhang. Multi-objective particle swarm optimization for robot path planning in environment with danger sources. *J. Comput.*, 6(8):1554–1561, 2011.

[83] Zhang Qiaorong and Gu Guochang. Path planning based on improved binary particle swarm optimization algorithm. In *2008 IEEE Conference on Robotics, Automation and Mechatronics*, pages 462–466. IEEE, 2008.

[84] Yong Zhang, Dun-wei Gong, Xiao-yan Sun, and Yi-nan Guo. A pso-based multi-objective multi-label feature selection method in classification. *Scientific reports*, 7(1):1–12, 2017.

[85] Hao Mei, Yantao Tian, and Linan Zu. A hybrid ant colony optimization algorithm for path planning of robot in dynamic environment. *Int J Inform Technol*, 12, 01 2006.

[86] Joon-Woo Lee, Jeong-Jung Kim, Byoung-Suk Choi, and Ju-Jang Lee. Improved ant colony optimization algorithm by potential field concept for optimal path planning. In *Humanoids 2008 - 8th IEEE-RAS International Conference on Humanoid Robots*, pages 662–667, 2008.

[87] John H Reif. Complexity of the mover's problem and generalizations. In *20th Annual Symposium on Foundations of Computer Science (sfcs 1979)*, pages 421–427. IEEE Computer Society, 1979.

[88] Jacob T Schwartz and Micha Sharir. On the "piano movers" problem. ii. general techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4(3):298–351, 1983.

[89] George E Collins. Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In *Automata theory and formal languages*, pages 134–183. Springer, 1975.

[90] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[91] Saugata Basu, Richard Pollack, and Marie-Françoise Roy. *Algorithms in Real Algebraic Geometry (Algorithms and Computation in Mathematics)*. Springer-Verlag, Berlin, Heidelberg, 2006.

[92] John Canny. *The complexity of robot motion planning*. MIT press, 1988.

[93] John Canny. Constructing roadmaps of semi-algebraic sets i: Completeness. *Artificial Intelligence*, 37(1-3):203–222, 1988.

[94] John Canny. Computing roadmaps of general semi-algebraic sets. *The Computer Journal*, 36(5):504–514, 1993.

[95] John F Canny and Ming C Lin. An opportunistic global path planner. *Algorithmica*, 10(2):102–120, 1993.

[96] John Canny. Some algebraic and geometric computations in pspace. In *Proceedings of the twentieth annual ACM symposium on Theory of computing*, pages 460–467, 1988.

[97] Dan Halperin and Micha Sharir. A near-quadratic algorithm for planning the motion of a polygon in a polygonal environment. *Discrete & Computational Geometry*, 16(2):121–134, 1996.

[98] Jacob T Schwartz and Micha Sharir. On the piano movers' problem: V. the case of a rod moving in three-dimensional space amidst polyhedral obstacles. *Communications on Pure and Applied Mathematics*, 37(6):815–848, 1984.

[99] Rodney A Brooks and Tomas Lozano-Perez. A subdivision algorithm in configuration space for findpath with rotation. *IEEE Transactions on Systems, Man, and Cybernetics*, (2):224–233, 1985.

[100] Rodney A Brooks. Solving the find-path problem by good representation of free space. *IEEE Transactions on Systems, Man, and Cybernetics*, (2):190–197, 1983.

[101] Bruce R Donald. A search algorithm for motion planning with six degrees of freedom. *Artificial Intelligence*, 31(3):295–353, 1987.

[102] Tomas Lozano-Perez. A simple motion-planning algorithm for general robot manipulators. *IEEE Journal on Robotics and Automation*, 3(3):224–238, 1987.

[103] K.K. Gupta and Zhenping Guo. Motion planning for many degrees of freedom: sequential search with backtracking. *IEEE Transactions on Robotics and Automation*, 11(6):897–906, 1995.

[104] Jérôme Barraquand and Jean-Claude Latombe. Robot motion planning: A distributed representation approach. *The International Journal of Robotics Research*, 10(6):628–649, 1991.

[105] Planning in a continuous space with forbidden regions: The ariadne"s clew algorithm. *Algorithmic Foundations of Robotics*, 1995.

[106] Pang C Chen and Yong K Hwang. Sandros: A motion planner with performance proportional to task difficulty. 9 1991.

[107] B. Glavina. Solving findpath by combination of goal-directed and randomized search. In *Proceedings., IEEE International Conference on Robotics and Automation*, pages 1718–1723 vol.3, 1990.

[108] L.E. Kavraki. Part orientation with programmable vector fields: two stable equilibria for most parts. In *Proceedings of International Conference on Robotics and Automation*, volume 3, pages 2446–2451 vol.3, 1997.

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

[110] K. Kondo. Motion planning with six degrees of freedom by multistrategic bidirectional heuristic free-space enumeration. *IEEE Transactions on Robotics and Automation*, 7(3):267–277, 1991.

[111] Steven M. Lavalle. Rapidly-exploring random trees: A new tool for path planning. Technical report, 1998.

[112] Sertac Karaman and Emilio Frazzoli. Incremental sampling-based algorithms for optimal motion planning. *Robotics Science and Systems VI*, 104(2), 2010.

[113] Lydia E Kavraki, Petr Svestka, J-C Latombe, and Mark H Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE transactions on Robotics and Automation*, 12(4):566–580, 1996.

[114] Lydia E. Kavraki. *Random Networks in Configuration Space for Fast Path Planning*. PhD thesis, Stanford, CA, USA, 1995. UMI Order No. GAX95-16854.

[115] Mark H Overmars and Petr Svestka. *A probablisitic learning approach to motion planning*, volume 1994. Unknown Publisher, 1994.

[116] Lydia E. Kavraki, Jean-Claude Latombe, Rajeev Motwani, and Prabhakar Raghavan. Randomized query processing in robot path planning. *Journal of Computer and System Sciences*, 57(1):50–60, 1998.

[117] J.J. Kuffner and S.M. LaValle. 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)*, volume 2, pages 995–1001 vol.2, 2000.

[118] S.M. LaValle and J.J. Kuffner. Randomized kinodynamic planning. In *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, volume 1, pages 473–479 vol.1, 1999.

[119] Steven M LaValle and JJ Kuffner. Rapidly-exploring random trees: Progress and prospects: Steven m. lavalle, iowa state university, a james j. kuffner, jr., university of tokyo, tokyo, japan. In *Algorithmic and Computational Robotics*, pages 303–307. AK Peters/CRC Press, 2001.

[120] Dave Ferguson and Anthony Stentz. Anytime rrts. pages 5369 – 5375, 11 2006.

[121] Florent Lamiraux, Etienne Ferré, and Erwan Vallée. Kinodynamic motion planning: Connecting exploration trees using trajectory optimization methods. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA'04. 2004*, volume 4, pages 3987–3992. IEEE, 2004.

[122] Chris Urmson and Reid Simmons. Approaches for heuristically biasing rrt growth. In *Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453)*, volume 2, pages 1178–1183. IEEE, 2003.

[123] Léonard Jaillet, Judy Hoffman, Jur van den Berg, Pieter Abbeel, Josep M. Porta, and Ken Goldberg. Eg-rrt: Environment-guided random trees for kinodynamic motion planning with uncertainty and obstacles. In *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2646–2652, 2011.

[124] M. Kalisiak and M. Panne. Rrt-blossom: Rrt with a local flood-fill behavior. pages 1237 – 1242, 06 2006.

[125] R. Kindel, D. Hsu, J.-C. Latombe, and S. Rock. Kinodynamic motion planning amidst moving obstacles. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 1, pages 537–543 vol.1, 2000.

[126] F. Lamiraux and J.P. Laumond. On the expected complexity of random path planning. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 4, pages 3014–3019 vol.4, 1996.

[127] David Hsu, Robert Kindel, Jean-Claude Latombe, and Stephen Rock. Randomized kinodynamic motion planning with moving obstacles. *The International Journal of Robotics Research*, 21(3):233–255, 2002.

[128] L.E. Kavraki, M.N. Kolountzakis, and J.-C. Latombe. Analysis of probabilistic roadmaps for path planning. In *Proceedings of IEEE International Conference on Robotics and Automation*, volume 4, pages 3020–3025 vol.4, 1996.

[129] L.E. Kavraki, M.N. Kolountzakis, and J.-C. Latombe. Analysis of probabilistic roadmaps for path planning. *IEEE Transactions on Robotics and Automation*, 14(1):166–171, 1998.

[130] Lydia E. Kavraki, Jean-Claude Latombe, Rajeev Motwani, and Prabhakar Raghavan. Randomized query processing in robot path planning. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing*, STOC '95, page 353–362, New York, NY, USA, 1995. Association for Computing Machinery.

[131] A.M. Ladd and L.E. Kavraki. Measure theoretic analysis of probabilistic path planning. *IEEE Transactions on Robotics and Automation*, 20(2):229–242, 2004.

[132] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975.

[133] Sunil Arya, David M Mount, Nathan S Netanyahu, Ruth Silverman, and Angela Y Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM (JACM)*, 45(6):891–923, 1998.

[134] Cong Fu and Deng Cai. Efanna : An extremely fast approximate nearest neighbor search algorithm based on knn graph, 2016.

[135] Thomas A Witten Jr and Leonard M Sander. Diffusion-limited aggregation, a kinetic critical phenomenon. *Physical review letters*, 47(19):1400, 1981.

[136] Sertac Karaman and Emilio Frazzoli. Optimal kinodynamic motion planning using incremental sampling-based methods. In *49th IEEE conference on decision and control (CDC)*, pages 7681–7687. IEEE, 2010.

[137] Sertac Karaman and Emilio Frazzoli. Sampling-based motion planning with deterministic μ-calculus specifications. In *Proceedings of the 48h IEEE Conference on Decision and Control (CDC) held jointly with 2009 28th Chinese Control Conference*, pages 2222–2229. IEEE, 2009.

[138] Oktay Arslan and Panagiotis Tsiotras. The role of vertex consistency in sampling-based algorithms for optimal motion planning. 04 2012.

[139] Martin Swaczyna. Several examples of nonholonomic mechanical systems. *Communications in Mathematics*, 19(1):27–56, 2011.

[140] Sertac Karaman and Emilio Frazzoli. Sampling-based optimal motion planning for non-holonomic dynamical systems. In *2013 IEEE International Conference on Robotics and Automation*, pages 5041–5047. IEEE, 2013.

[141] A. Sanchez, J. Arenas, and R. Zapata. Nonholonomic motion planning for car-like robots. 2002.

[142] Petr Švestka and Markus Hendrik Overmars. Probabilistic path planning. In *Robot motion planning and control*, pages 255–304. Springer, 1998.

[143] Boris Lau, Christoph Sprunk, and Wolfram Burgard. Kinodynamic motion planning for mobile robots using splines. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2427–2433. IEEE, 2009.

[144] Ahmad A Masoud. Kinodynamic motion planning. *IEEE Robotics & Automation Magazine*, 17(1):85–99, 2010.

[145] Linjun Li, Yinglong Miao, Ahmed H Qureshi, and Michael C Yip. Mpc-mpnet: Model-predictive motion planning networks for fast, near-optimal planning under kinodynamic constraints. *arXiv preprint arXiv:2101.06798*, 2021.

[146] Hamidreza Chitsaz and Steven M LaValle. Time-optimal paths for a dubins airplane. In *2007 46th IEEE conference on decision and control*, pages 2379–2384. IEEE, 2007.

[147] Lester E Dubins. On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents. *American Journal of mathematics*, 79(3):497–516, 1957.

[148] M. A. Boon, A. P. Drijfhout, and S. Tesfamichael. COMPARISON OF A FIXED-WING AND MULTI-ROTOR UAV FOR ENVIRONMENTAL MAPPING APPLICATIONS: A CASE STUDY. *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, XLII-2/W6:47–54, August 2017.

[149] Jinho Kim, S. Andrew Gadsden, and Stephen A. Wilkerson. A comprehensive survey of control strategies for autonomous quadrotors. *Canadian Journal of Electrical and Computer Engineering*, 43(1):3–16, 2020.

[150] Mina Kamel, Thomas Stastny, Kostas Alexis, and Roland Siegwart. Model predictive control for trajectory tracking of unmanned aerial vehicles using robot operating system. In *Robot operating system (ROS)*, pages 3–39. Springer, 2017.

[151] Peter E Hart, Nils J Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[152] Alejandro Perez, Robert Platt, George Konidaris, Leslie Kaelbling, and Tomas Lozano-Perez. Lqr-rrt*: Optimal sampling-based motion planning with automatically derived extension heuristics. In *2012 IEEE International Conference on Robotics and Automation*, pages 2537–2542. IEEE, 2012.

[153] Eduardo F Camacho and Carlos Bordons Alba. *Model predictive control*. Springer science & business media, 2013.

[154] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive control for linear and hybrid systems*. Cambridge University Press, 2017.

[155] Timothy McLain, Randall W Beard, and Mark Owen. Implementing dubins airplane paths on fixed-wing uavs. 2014.

[156] Karl D Hansen and Anders la Cour-Harbo. Waypoint planning with dubins curves using genetic algorithms. In *2016 European Control Conference (ECC)*, pages 2240–2246. IEEE, 2016.

[157] Jacob Mattingley and Stephen Boyd. Cvxgen: A code generator for embedded convex optimization. *Optimization and Engineering*, 13(1):1–27, 2012.

[158] Michael Grant and Stephen Boyd. Cvx: Matlab software for disciplined convex programming, version 2.1, 2014.

[159] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5, 2009.

[160] Ioan A. Şucan, Mark Moll, and Lydia E. Kavraki. The Open Motion Planning Library. *IEEE Robotics & Automation Magazine*, 19(4):72–82, December 2012. http://ompl.kavrakilab.org.

[161] Nathan P Koenig and Andrew Howard. Design and use paradigms for gazebo, an open-source multi-robot simulator. In *IROS*, volume 4, pages 2149–2154. Citeseer, 2004.

[162] SITL contributors. SITL guide. http://ardupilot.org/dev/docs/sitl-simulator-software-in-the-loop.html, 2020.

[163] Lorenz Meier, Dominik Honegger, and Marc Pollefeys. Px4: A node-based multithreaded open source robotics framework for deeply embedded platforms. In *2015 IEEE international conference on robotics and automation (ICRA)*, pages 6235–6240. IEEE, 2015.